DiMOpt

Salvado, João; Mansouri, Masoumeh; Pecora, Federico

DOI:
10.1109/IROS47612.2022.9981345

DiMOpt: a Distributed Multi-robot Trajectory Optimization Algorithm

João Salvado, Masoumeh Mansouri, Federico Pecora

***Abstract*— This paper deals with *Multi-robot Trajectory Planning*, that is, the problem of computing trajectories for multiple robots navigating in a shared space while minimizing for control energy. Approaches based on trajectory optimization can solve this problem optimally. However, such methods are hampered by complex robot dynamics and collision constraints that couple robot's decision variables. We propose a distributed multi-robot optimization algorithm (DiMOpt) that addresses these issues by exploiting (1) consensus optimization strategies to tackle coupling collision constraints, and (2) a single-robot sequential convex programming method for efficiently handling non-convexities introduced by dynamics. We compare DiMOpt with a baseline centralized multi-robot sequential convex programming algorithm (SCP). We empirically demonstrate that DiMOpt scales well for large fleets of robots while computing solutions faster and with lower costs. Finally, DiMOpt is an iterative algorithm that finds feasible trajectories before converging to a locally optimal solution, and results suggest the quality of such fast initial solutions is comparable to a converged solution computed via SCP.**

I. Introduction

The Multi-Robot Trajectory Planning (MRTP) problem has been receiving increasing attention in recent years [1]. Some existing methods decompose this problem into known and well-studied problems such as motion planning, coordination, and control, which are then solved in a *loosely-coupled* manner at the cost of optimality [2]. Others focus on computing paths on graphs instead of kinodynamically feasible trajectories, thus making strong assumptions on robot dynamics and/or requiring costly re-engineering of the environment, processes, or robots [3].

This paper considers the motion planning and coordination problems jointly while exploring trajectory optimization methods. These methods offer a powerful tool for modeling the MRTP problem with optimality guarantees [4]. Nevertheless, they are hindered by non-convexities introduced by two factors: complex robot kinodynamics, and a combinatorial explosion of collision constraints due to the coupling between robot decision variables. These factors lead to poor scaling and hinder the practical use of trajectory optimization methods outside of real applications.

There are, however, known methods for dealing with these two factors. Sequential convex programming (SCP) deals with non-convex functions [5], and distributed optimization methods have been proposed for constraints that couple decision variables in optimization [6]. In this paper, we propose to combine these two lines of research to realize an efficient MRTP solver. Specifically, we propose DiMOpt (Distributed Multi-robot Trajectory Optimization), which combines SCP for individual robots with a distributed optimization scheme. More specifically, DiMOpt will leverage processor parallelization on a single multi-core computer and we, therefore, do not consider network communication delays, although there is no technical assumption forbidding its implementation on a distributed multi-computer setting.

DiMOpt is evaluated in comparison to a standard centralized multi-robot sequential convex programming method with non-convex dynamics (SCP) [7]. This standard method is often utilized as a baseline, e.g. [8], since it finds efficiently locally optimal solutions. For comparison fairness, we also include recent developments in SCPs present in [9], as described in Section IV-A. Our results indicate DiMOpt scales exponentially better with the number of robots than the baseline, while also finding lower-cost solutions. Also, the results show that DiMOpt can find an initial feasible solution fast with solution quality comparable to the baseline M-SCP method. Finally, we also investigate how other factors, including path length, affect the performance of DiMOpt.

II. Related Work

Previously developed methods for solving the MRTP problem exploit efficient off-the-shelf motion/trajectory planning solvers to pre-compute individual robot trajectories, while collisions are avoided via scheduling [10], [2], or priority planning [11], [12]. These approaches typically scale well, but underestimate the interplay between robots, leading to sub-optimal solutions. Others exploit efficient Integer Linear Programming (ILP) and Max Flow solvers [13] [14] and are suitable for solving the related Multi-Agent Path Finding (MAPF) problem [15]. Although they scale well with the number of robots [3], [16], these approaches often make unrealistic assumptions about robot geometries and dynamics, as a result of focusing on computing paths on graphs instead of dynamically-feasible trajectories.

Conversely, trajectory optimization methods address the problem of finding a set of controls that minimize a cost functional while adhering to a set of constraints [17]. Such approaches make fewer assumptions about the robots (e.g., adopting a realistic dynamic model) and provide optimality guarantees, although at the cost of poorer scaling. This paper proposes a trajectory optimization method. We build upon existing efficient convex optimization solvers exploited by sequential convex programming methods (SCP) [18] to handle the non-convex parts of the problem. We handle scaling in the number of robots via a distributed consensus optimization method [19].

João Salvado and Federico Pecora are with the AASS Research Centre, Örebro University, <name>.<surname>@oru.se

Masoumeh Manosuri is with the School of Computer Science at the University of Birmingham, m.mansouri@bham.ac.uk

SCP methods have been applied in the context of multiple robots via convexification of collision constraints with linear dynamic models [7]. The popular TrajOpt ROS library for single robots applies a l_1-penalty SCP algorithm [20]. A generalization of the SCP method for optimal control problems is provided in [21], where the authors introduce an additional step of projection to the feasible set. Moreover, it is possible [22] to decouple the problem via artificially considering configurations of other robots as obstacles in case a collision is detected, thus decomposing the problem while losing optimality.

Distributed optimization traces back to primal/dual decomposition methods that rely on the existence of sub-gradients of the non-convex functions of interest. Furthermore, these sub-gradients are fast to compute, which makes distributed optimization applicable, and this has been leveraged in the multi-robot context [23], [24], although not in the context of trajectory optimization in [24], and with few robots (three) and alternative consensus variables in [23]. More recently, the Alternating Direction Method of Multipliers (ADMM) brought additional robustness compared to previous decomposition methods that rely on a dual ascend step with a step size that is difficult to estimate. That is, ADMM in augmented Lagrangian form [25] relies on a factor that is both represented in the cost function and the dual step. ADMM has superior convergence to that of decomposition methods [6]. In the context of multi-robot trajectory optimization, this has been attempted in the case of two robots [26], although they require solving a QP problem on the consensus step and do not leverage SCP to handle non-convex dynamics. Also an evaluation of scalability, as well as the applicability of ADMM to larger multi-robot systems, is missing. Derivatives of the ADMM algorithm were applied to consensus optimization [27] problems where multiple agents converge (i.e., reach a consensus) on quantities while having local information and communicating with each other. However, these quantities are not related to robot trajectories.

Other articles overlap several of the previously mentioned fields or introduce new concepts. One of such works exploits efficient MAPF solvers to generate multi-robot paths utilized in the computation of traversable safe regions for each robot [28]. This effectively decomposes the problem into subsets of robots that do not interact but sacrifice optimality. Similarly, the work [29] explores a MAPF solver to compute a discrete schedule prior to trajectory optimization and therefore ignores robot's dynamics on the planned schedule. Moreover, distributed optimization techniques have been proposed [30] to compute multi-robot paths, although these ignore robot dynamics, e.g. discrete linear system, in the computation of trajectories and have no optimality guarantees. Finally, buffered voronoi cells (BVP) [31] have been used for collision avoidance achieving real-time performance with no optimality guarantees.

III. Problem Statement

In the following, matrices are defined in uppercase bold letters (e.g., $\boldsymbol{M}$), vectors in lowercase bold letters (e.g., $\boldsymbol{v}$) and constants in non-bold uppercase (e.g., C). Moreover, $\boldsymbol{v}[i]$ is the i-th element of vector $\boldsymbol{v}$. Throughout the paper, we will refer to the following sets and symbols: robots $r \in \{1, \dots, R\}$; polygons $p \in \{1, \dots, P\}$; goals $g \in \{1, \dots, G\}$; discrete time $k \in \{1, \dots, N\}$; continuous time $t \in [0, 1]$. Note that continuous time t is normalized as $t = \frac{k}{N}$.

Variables, functions, and constraints pertaining to a single-robot are extended to the multi-robot case by dropping the subscript r; similarly, multi-robot variables are restricted to the single-robot case by introducing the subscript r. Additionally, we introduce an upper-script tilde "$\sim$" on functions that are first-order Taylor's approximated. For instance, if f is a function describing the multi-robot dynamic system then $\tilde{f}_r$ is the approximated dynamics of robot r.

A. Preliminary Definitions

In the interest of defining the *Multi-robot Trajectory Planning* problem, we formulate components of this problem individually. These are the objective function and constraints modeling robot dynamics and collisions between robots and obstacles. For clarity and without loss of generality, we define the problem using the constraints assumed in the experimental evaluation. Note that our method is not specific to a particular dynamic model.

1) Dynamic Model: Each robot is considered to be circle-shaped for robot-to-robot collision modeling purposes and follows a differential drive model defined as,

$$\dot{x}_r = v_r \cos\theta_r, \quad \dot{y}_r = v_r \sin\theta_r, \quad \dot{\theta}_r = \frac{1}{2\,L_r}\omega_r, \qquad (1)$$

where $2\,L_r$ is the distance between right and left wheel (so L_r is the robot's circle radius). The state space pose is $\boldsymbol{x}_r = [x_r\, y_r\, \theta_r] \in SE(2)$ and the control space is $\boldsymbol{u}_r = [v_r\, \omega_r] \in \mathbb{R}^2$, where v_r stands for speed and w_r for angular velocity. Car-like models, such as this one, impose zero side-slip via non-holonomic Pfaffian constraints [32] which lead to non-convex equality constraints of the general form $\dot{\boldsymbol{x}}_r = f_r(\boldsymbol{x}_r, \boldsymbol{u}_r)$. We will refer to the xy-Cartesian position of robot r as $\boldsymbol{q}_r = [x_r\, y_r]$.

2) Objective Function: We optimize for the control energy of the multi-robot fleet using the quadratic function

$$f_0(\boldsymbol{u}) = \sum_r \boldsymbol{u}_r \boldsymbol{W} \boldsymbol{u}_r^T \,:\, \boldsymbol{W} = \begin{bmatrix} w_v & 0 \\ 0 & w_\omega \end{bmatrix}, \qquad (2)$$

where $W \succ 0$ is diagonal positive-definite with associated penalty weights for speed w_v and angular velocity w_ω.

3) Transcription Method: Our solution transforms the continuous time dynamics into their discrete time equivalent. To this end, we deploy a direct multiple shooting transcription method [33], [34], i.e., we go from $\boldsymbol{x}(t)$ to $\boldsymbol{x}[k]$ for $k = \{1, \dots, N\}$. More specifically, we start by approximating the dynamic model equations for each k-knot via the first-order Taylor's expansion,

$$\widetilde{\boldsymbol{x}}_r[k] = f_r(\boldsymbol{x}_r^0[k], \boldsymbol{u}_r^0[k]) + \nabla f_r\left(\boldsymbol{x}_r^0[k], \boldsymbol{u}_r^0[k]\right) \begin{bmatrix} \boldsymbol{x}_r[k] - \boldsymbol{x}_r^0[k] \\ \boldsymbol{u}_r[k] - \boldsymbol{u}_r^0[k] \end{bmatrix}, \qquad (3)$$

where $\boldsymbol{x}_r^0$ and $\boldsymbol{u}_r^0$ are the state and control of robot r's reference trajectory. We integrate this approximation for each discrete time interval and robot, resulting in the set of convex linear equalities

$$\tilde{\boldsymbol{f}}_r[k] = \boldsymbol{x}_r[k] - \text{RK4}\left(\tilde{\boldsymbol{x}}_r[k], \boldsymbol{x}_r[k-1], \boldsymbol{u}_r[k-1]\right) \quad (4)$$

$$\tilde{\mathcal{F}} = \{\tilde{\boldsymbol{f}}_r[k] = 0 \mid r \in \{1, \dots, R\}\, k \in \{1, \dots, N\}\}, \quad (5)$$

where RK4 is a 4^{th}-order Runge-Kutta's integration method.

4) Obstacle Free Space: We approximate the free space in the environment with a set of convex polygons $\{\mathcal{P}_1, \dots, \mathcal{P}_P\}$. A polygon is defined as an intersection of multiple half-spaces. Let $\tau_{\mathcal{P}} : \{1, \dots, R\} \mapsto \{1, \dots, P\}$ be a mapping between each robot and a polygon. We assume this assignment to be given since it could be generated as in [28], [35] via the creation of safe corridors. Furthermore, having L_r as the radius of the circle enclosing the geometry of robot r, we ensure that it remains in the obstacle-free region by imposing the following set of convex linear inequalities

$$\mathcal{H} = \{A_p q_r[k] - b_p \mathbb{1}^T + L_r[||a_1||_2 \dots ||a_{H_p}||_2]^T \leq 0 \mid \forall_r p = \tau_{\mathcal{P}}[r]\}, \quad (6)$$

where $A_p = [a_1 \dots a_{H_p}]^T \in \mathbb{R}^{\mathrm{H}_p \times 2}$ and $b_p \in \mathbb{R}^{1 \times \mathrm{H}_p}$, and $\mathbb{1}$ is an identity row vector of appropriate dimensions, and H_p is the number of half-spaces of polygon $\mathcal{P}_p$.

5) Collisions: Robot-to-robot collisions are modeled via coupling constraints, where the separation between robots is enforced with the following l_2-norm

$$\boldsymbol{c}_{ij}[k] = ||\boldsymbol{q}_i[k] - \boldsymbol{q}_j[k]||_2^2 - (L_i + L_j)^2. \quad (7)$$

Similarly to the approximation of the dynamics in equation (3) we linearize the collision constraint of equation (7) with a first-order Taylor's approximation around a reference trajectory, which we will refer to as $\tilde{\boldsymbol{c}}_{ij}[k]$. Thus, the set of linear inequality constraints that excludes robot-to-robot collisions is defined by the set of constraints

$$\tilde{\mathcal{C}} = \{\tilde{\boldsymbol{c}}_{ij}[k] \geq 0 \mid i \neq j \in \{1, \dots, R\}, k \in \{1, \dots, N\}\}. \quad (8)$$

6) Trust-Region: The approximations of collision and dynamic model constraints are reliable in a trust-region around a reference trajectory $(\boldsymbol{x}^0, \boldsymbol{u}^0)$. Thus we impose that our decision variables have to be inside a trust-region radius τ defined as follows

$$\mathcal{T} = \left\{ (\boldsymbol{x}, \boldsymbol{u}) \mid \left|\left| \begin{matrix} \boldsymbol{x}^T - \boldsymbol{x}^{0^T} \\ \boldsymbol{u}^T - \boldsymbol{u}^{0^T} \end{matrix} \right|\right|_\infty \leq \tau \right\}. \quad (9)$$

A bigger τ value entails the approximation correctly models the problem around the reference trajectory while optimizing at a faster convergence rate, with the *caveat* that a better approximation is more computationally demanding (e.g., requiring a convex second-order Taylor's expansion or particle fitting method).

B. Optimization Problem

We define our problem of interest as follows:

Problem 1 (Multi-Robot Trajectory Planning):

$$\begin{aligned} \underset{\boldsymbol{x}, \boldsymbol{u}}{\text{minimize}} \;\; & f_0(\boldsymbol{u}) \\ \text{subject to} \;\; & \mathcal{F}, \mathcal{H}, \mathcal{C} \\ & \boldsymbol{x}[0] = \boldsymbol{x}_{\text{start}},\, \boldsymbol{x}[N] = \boldsymbol{x}_{\text{goal}} \\ & \underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}} \end{aligned}$$

with upper ($\overline{\boldsymbol{u}}$) and lower ($\underline{\boldsymbol{u}}$) bounds on the input controls. Problem 1 has realistic complex dynamics ($\mathcal{F}$) and coupling non-convex collision constraints ($\mathcal{C}$). In the next section, we exploit the approximations previously defined as a tool to solve this problem efficiently.

IV. Approach

Problem 1 is hard to solve due to the non-convexities introduced by complex robot dynamics and due to the collision constraints that couple decision variables of different robots. Non-convexity can be handled by a process of successive convexification of non-convex optimal problems, a locally optimal incomplete approach that is known as Sequential Convex Programming (SCP) [5]. The problem of coupling collision constraints is addressed by exploiting distributed consensus optimization algorithms [19]. In Section IV-A, we will describe the baseline multi-robot SCP [7], implemented using an l_1-penalty method [20] with recent developments [9] to solve Problem 1. Moreover, a single-robot SCP will be exploited in Section IV-B, where a distributed consensus algorithm efficiently parallelizes the problem in terms of robots while ensuring solution convergence.

A. Multi-robot Sequential Convex Programming (SCP)

We define an l_1-penalty function approximation as

$$\tilde{\phi}(\boldsymbol{x}, \boldsymbol{u}) = f_0(\boldsymbol{u}) + \sum_r \lambda_{f_r} \sum_k |\tilde{\boldsymbol{f}}_r[k]| + \lambda_c \sum_k \sum_{i \neq j} |\tilde{\boldsymbol{c}}_{ij}[k]|^+ \quad (10)$$

where we penalize dynamic infeasibility and robot collisions by manipulating λ_{f_r} and λ_c, respectively. Note that $|c(\cdot)|^+ = \max(c(\cdot), 0)$. In other words, we move dynamic and collision equations from the set of constraints to the cost function, thus allowing and penalizing initial infeasibility.

Next, we define an approximation of Problem 1 around a reference trajectory. This approximated MRTP problem (Problem 2) is a combination of the previously defined l_1-penalty cost function (10) with the approximated constraints (i.e., dynamic and collision constraints $\tilde{\mathcal{F}}$ and $\tilde{\mathcal{C}}$, respectively), whereas the non-approximated constraints remain unaltered (e.g., obstacle free space constraints $\mathcal{H}$).

Problem 2 (Approximate Multi-Robot Trajectory Planning):

$$\underset{\boldsymbol{x},\boldsymbol{u}}{\text{minimize}} \;\; \tilde{\phi}(\boldsymbol{x},\boldsymbol{u})$$

$$\text{subject to} \;\; (\boldsymbol{x},\boldsymbol{u}) \in \mathcal{T} \tag{11}$$

$$\boldsymbol{x}[0] = \boldsymbol{x}_{\text{start}},\; \boldsymbol{x}[N] = \boldsymbol{x}_{\text{goal}} \tag{12}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u} \leq \overline{\boldsymbol{u}} \tag{13}$$

$$\mathcal{H} \tag{14}$$

In summary, the decision variables $(\boldsymbol{x},\boldsymbol{u})$ of Problem 2 must lie in the trust-region (11) and robots must navigate in the free space (14) while starting and finishing in a predefined state (12), with bounds on controls (13).

To solve the overall non-convex Problem 1 we can now exploit sequentially computed solutions of the convex Problem 2 as described in the SCP Algorithm 1. Note that although starting in infeasibility due to convexification, this method will stop in a feasible solution once it converged [5]. The SCP algorithm is composed of three main loops, where the outer loop [line 1] ensures there is no constraint violation, while the inner loops ensure convergence of the approximated problem solution trajectories [line 3] as well as the trust-region [line 5].

We explain the details of Algorithm 1 starting from the inner loops. The trust-region loop [line 5] is responsible for expanding or shrinking this region depending on how accurate the approximated $\tilde{\phi}$ is versus the actual reduction in cost ϕ (n.b. costs are scalars), as measured by the fraction $\tilde{\delta}/\delta$. In the approximation loop [line 3], we approximate/-convexify both dynamic and collision functions over the current trajectory [line 4], using the first-order Taylor's expansion described in equation (3), and we test if either the solution trajectory or the overall cost has converged [line 24]. Moreover, in the outer violation loop [line 1] we check for constraint violations and increase by a factor of k the violation penalties λ if necessary. Finally, Problem 2 [line 6] is solved using an interior-point method implemented by an off-the-shelf solver.

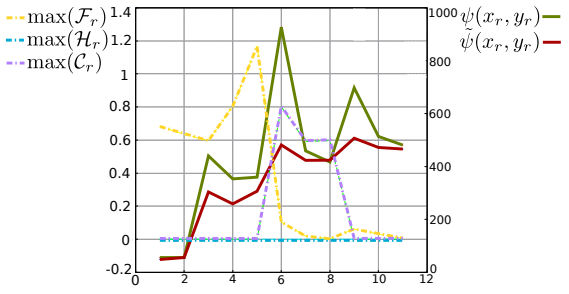

Fig. 1: Approximated and actual penalty costs (red and green curves with right y-axis) and maximum violation of dynamic, free space, and collision constraints (yellow, blue, and purple curves, respectively, with left y-axis) over each iteration (x-axis) of Algorithm 1. This represents the evolution of these quantities for a problem with 5 robots.

In conclusion, we apply a standard l_1-penalty SCP method to the MRTP problem at hand. For additional details on the properties of the SCP method and their use in single-robot

Algorithm 1: SCP

Parameters:
$\boldsymbol{\tau^- \leq 1, \tau^+ \geq 1}$: expand and shrink factors
$\boldsymbol{\gamma_0 \approx 0, \gamma_1 \leq 1}$: convergence parameters
$\boldsymbol{x^{(0)}, u^{(0)}}$: initial guess trajectory
$\boldsymbol{k}$: violation penalty factor
$\boldsymbol{c_{tol} \approx 10^{-3}, f_{tol} = \Delta_{tol} \approx 10^{-2}}$: tolerances
$\boldsymbol{i \leftarrow 0}$: iteration number
Output: $\boldsymbol{x^{(i)}, u^{(i)}}$: multi-robot trajectory

1 **while** *(Constraint Violations)* **do**
2 $\tau \leftarrow \tau_0$ /* reset trust-region */
3 **while** *(¬Approximation Converged)* **do**
4 $\tilde{f}, \tilde{c} \leftarrow$ convexity$(f, c, \boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)})$
5 **while** *(¬Trust-Region Converged)* **do**
6 $\boldsymbol{x}^*, \boldsymbol{u}^* \leftarrow$ solve *Problem 2*
  /* Approx. and Actual Reduction */
7 $\tilde{\delta} = \phi(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}) - \tilde{\phi}(\boldsymbol{x}^*, \boldsymbol{u}^*)$
8 $\delta = \phi(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}) - \phi(\boldsymbol{x}^*, \boldsymbol{u}^*)$
9 **if** $\delta/\tilde{\delta} \geq \gamma_0$ **then** /* accept solution */
10 $(\boldsymbol{x}^{(i+1)}, \boldsymbol{u}^{(i+1)}) \leftarrow (\boldsymbol{x}^*, \boldsymbol{u}^*)$
11 $i \leftarrow i + 1$
12 **if** $\delta/\tilde{\delta} \geq \gamma_1$ **then**
13 $\tau \leftarrow \tau^+ \tau$ ▷**go to** 4 /* expand $\mathcal{T}$ */
14 **else**
15 $\tau \leftarrow \tau^- \tau$ /* shrink $\mathcal{T}$ */
16 **if** $\delta/\tilde{\delta} < 0$ **then** /* line-search */
17 $(\boldsymbol{x}^{(i+1)}, \boldsymbol{u}^{(i+1)}) \leftarrow$recover$(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}, \boldsymbol{x}^*, \boldsymbol{u}^*)$
18 $i \leftarrow i + 1$
19 ▷ **go to** 25
20 **if** $\tau < \tau_{min}$ **then** ▷**go to** 4 /* $\mathcal{T}$ small */
21 /* cost change */
22 $\Delta f = |f_0(\boldsymbol{u}^{(i+1)}) - f_0(\boldsymbol{u}^{(i)})|$
  /* solution change */
23 $\Delta = \left\| \begin{matrix} \boldsymbol{u}^{(i+1)} - \boldsymbol{u}^{(i)} \\ \boldsymbol{x}^{(i+1)} - \boldsymbol{x}^{(i)} \end{matrix} \right\|_\infty$
24 **if** $(\Delta f \leq f_{tol} \cup \Delta \leq \Delta_{tol})$ **then** ▷**go to** 25
25 **for** $r = \{1, \ldots, R\}$ **do** /* dynamics violation */
26 **if** $\max(\mathcal{F}_r) \leq c_{tol}$ **then** $\lambda_{f_r} \leftarrow k\, \lambda_{f_r}$
27 **if** $\max(\mathcal{C}) \leq c_{tol}$ **then** $\lambda_c \leftarrow k\, \lambda_c$ /* colliding */

systems, or, more generally, in optimal control problems, please consult [20], [9]. Given that our approach deals with a multi-robot problem, there are some differences worth mentioning. Firstly, the violation penalty λ_{f_r} increases per individual robot once dynamic constraints are violated. Secondly, we tailor the exiting criterion of each loop to the problem at hand. Finally, we introduce a backtracking line-search mechanism that recovers a trajectory if the predicted cost diverges (shown in Algorithm 2).

Note that the penalty function with no approximation $\phi(\boldsymbol{x},\boldsymbol{u})$ is similarly defined as $\tilde{\phi}(\boldsymbol{x},\boldsymbol{u})$ but with the actual dynamics (f_r) and collision constraints (c_{ij}).

1) General Notes: Figure 1illustrates the standard behavior of the SCP method described in Algorithm 1 per iteration until a solution is found. For instance, the actual (ϕ) and model $(\tilde{\phi})$ costs converge over iterations 3–5, whereas they diverge at iteration 6. This means that iterations 3–5 occurred inside the while loop at line 3, and once the costs converged (iteration 5), the while loop is exited. Next, the violation of the dynamic constraints (yellow curve iteration 5) is detected at line 25 and penalized, thus resulting in a reduction of the dynamic violation and an initial divergence in the costs at

Algorithm 2: recover$(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}, \boldsymbol{x}^*, \boldsymbol{u}^*)$

Parameters: $\boldsymbol{\alpha} < 0.5, \boldsymbol{\beta} > 1, \boldsymbol{t} = 1$

$\Delta_x = \boldsymbol{x}^* - \boldsymbol{x}^i$, $\Delta_u = \boldsymbol{u}^* - \boldsymbol{u}^i$

while $\phi\left(\boldsymbol{x}^i - t\,\Delta_x, \boldsymbol{u}^i - t\,\Delta_u\right) \leq \phi(\boldsymbol{x}^i, \boldsymbol{u}^i) - \alpha\, t\left(\phi(\boldsymbol{x}^i, \boldsymbol{u}^i) - \tilde{\phi}(\boldsymbol{x}^*, \boldsymbol{u}^*)\right)$ **do**

  $t = \frac{t}{\beta}$

  $\boldsymbol{x}^{(i)} \leftarrow \boldsymbol{x}^{(i)} + t\,\Delta_x$

  $\boldsymbol{u}^{(i)} \leftarrow \boldsymbol{u}^{(i)} + t\,\Delta_u$

return $\left(\boldsymbol{x}^{(i)}, \boldsymbol{u}^{(i)}\right)$

iteration 6. The same behavior occurs at iteration 8 but now with a violation of the collision constraints (purple curve). Finally, as expected, the free space constraints (blue curve) are always ensured and SCP finds a solution at iteration 11 with converged costs and no constraint violations.

B. Distributed Multi-robot Optimization Method (DiMOpt)

The previously described method is able to handle non-convexity efficiently. Nevertheless, the number of constraints increases combinatorially with the number of robots, resulting in poor problem scaling. Therefore, we have developed a Distributed Multi-robot Optimization method (DiMOpt) to overcome this issue, while leveraging a single-robot SCP to efficiently handle robot's non-convex dynamics.

A key idea of this distributed method is to redefine collision constraints such that Problem 2 can be decoupled in terms of robots. The method then iteratively solves for each robot, in parallel, while keeping the trajectories of other robots fixed. As a result, the collision constraint becomes

$$\boldsymbol{c}_{i\hat{j}}[k] = ||\boldsymbol{q}_i[k] - \hat{\boldsymbol{q}}_j[k]||_2^2 - (L_i + L_j)^2, \tag{15}$$

where $\boldsymbol{q}_i[k]$ is a decision variable representing i-th robot's xy-coordinates, while $\hat{\boldsymbol{q}}_j[k]$ is a constant representing j-th robot's xy-coordinates. As a consequence, it is now possible to decompose the l_1-penalty cost function shown in equation (10) per robot in the following manner,

$$\tilde{\phi}_r(\boldsymbol{x_r}, \boldsymbol{u_r}) = f_{0_r}(\boldsymbol{u_r}) + \lambda_{f\,r} \sum_k |\tilde{\boldsymbol{f}}_r[k]| + \lambda_{c_r} \sum_k \sum_{r \neq j} |\tilde{\boldsymbol{c}}_{r\hat{j}}[k]|^+ \tag{16}$$

where $\tilde{\boldsymbol{c}}_{r\hat{j}}[k]$ is the first-order Taylor's approximation of $\boldsymbol{c}_{r\hat{j}}[k]$. Notice that the pair of collision constraints $\boldsymbol{c}_{i\hat{j}}[k]$ and $\boldsymbol{c}_{\hat{i}j}[k]$ refer to the same distance between robot i and j, wherein $\boldsymbol{c}_{i\hat{j}}[k]$ robot j is fixed and vice-versa. In other words, each robot has a local representation of where the others are. Therefore, the missing piece of the puzzle is that these quantities need to converge, which is the same as stating that in the end, each robot will know the trajectories of the other robots. For that, we propose a consensus optimization algorithm, described in Algorithm 3, where we introduce a consensus variable $\boldsymbol{z}_r[k] \in \mathbb{R}^2$ for each robot position $\boldsymbol{q}_r$. We penalize for deviations of the consensus variable in augmented Lagrangian form with scaled dual variables as

$$\tilde{\psi}_r(\boldsymbol{x_r}, \boldsymbol{u_r}) = \tilde{\phi}_r(\boldsymbol{x_r}, \boldsymbol{u_r}) + \frac{1}{2}\rho_{\text{al}} \sum_k \left(||\boldsymbol{q}_r[k] - \boldsymbol{z}_r[k] + \boldsymbol{\lambda}_{z_r}[k]||_2^2\right), \tag{17}$$

Algorithm 3: DiMOpt

Parameters:

$\hat{\boldsymbol{x}}, \hat{\boldsymbol{u}}$: initial guess trajectory

$\boldsymbol{\rho}_{\text{al}} \leftarrow \mathbf{0.1}$: augmented Lagrangian constant

$\boldsymbol{z_r} \leftarrow \hat{\boldsymbol{q}}_{\boldsymbol{r}}$: consensus variables

$\boldsymbol{\lambda_r} \leftarrow \mathbf{0}$: consensus multipliers

Output: $\boldsymbol{x^*}, \boldsymbol{u^*}$: multi-robot trajectory

1 **while** (¬ *Cost Converged* ∪ *Colliding*) **do**

  /* 1) Solve - In parallel */

2 **In Parallel**

3 **for** $r = \{1, \ldots, R\}$ **do** /* robot SCP */

4 update_*Problem_3* $(\hat{\boldsymbol{x}}_r, \boldsymbol{z}_r, \boldsymbol{\lambda}_r)$

5 $\boldsymbol{x}_r^*, \boldsymbol{u}_r^* \leftarrow$ Algorithm 1 [solve *Problem 3*]

  /* 2) Update / Share Knowledge */

6 **for** $r = \{1, \ldots, R\}$ **do** /* trajectories */

7 $\hat{\boldsymbol{x}}_r \leftarrow \frac{1}{2}\left(\boldsymbol{x}_r^* + \hat{\boldsymbol{x}}_r\right)$

8 **for** $r = \{1, \ldots, R\} \mid i \neq j$ **do** /* consensus */

9 **for** $k = \{1, \ldots, N\}$ **do**

10 $\boldsymbol{z}_r[k] \leftarrow \frac{1}{2}\left(\boldsymbol{q}_r^*[k] + \boldsymbol{z}_r[k]\right) + b\left(\boldsymbol{q}_r^*[k] - \boldsymbol{z}_r[k]\right)$

11 $\boldsymbol{\lambda}_r[k] \leftarrow \boldsymbol{\lambda}_r[k] + \left(\boldsymbol{q}_r^*[k] - \boldsymbol{z}_r[k]\right)$

12 **if** $\max(\mathcal{C}) \leq 0 \cup \Delta_f \leq f_{tol}$ **then break**

where $\boldsymbol{\lambda}_{z_r}[k] \in \mathbb{R}^2$ are multipliers that penalize for non-consensus, and ρ_{al} is the augmented Lagrangian constant. The consensus variable $\boldsymbol{z}_r$ can be understood as a local copy of robot r's trajectory that all the other robots have, and therefore we penalize robot r deviating from this trajectory. Finally, the full optimization problem each robot will iteratively solve is

Problem 3 (Single-Robot Consensus Optimization):

$$\begin{aligned} \underset{\boldsymbol{x}_r, \boldsymbol{u}_r}{\text{minimize}} \quad & \tilde{\psi}_r(\boldsymbol{x}_r, \boldsymbol{u}_r) \\ \text{subject to} \quad & (\boldsymbol{x}_r, \boldsymbol{u}_r) \in \mathcal{T}_r \\ & \boldsymbol{x}_r[0] = \boldsymbol{x}_{\text{start}_r},\ \boldsymbol{x}_r[N] = \boldsymbol{x}_{\text{goal}_r} \\ & \underline{\boldsymbol{u}}_r \leq \boldsymbol{u}_r \leq \overline{\boldsymbol{u}}_r \\ & \mathcal{H}_r \end{aligned} \tag{18}$$

In contrast with Problem 2, here we have single robot decision variables and the cost function is a combination of exact penalty and consensus terms.

The distributed optimization described in Algorithm 3 starts by solving single robot SCPs in parallel, followed by a consensus optimization step where trajectories and penalties are shared between robots. This is an iterative process that ends when the cost has converged and there are no collisions.

The single robot's SCP is similar to the multi-robot case described in Algorithm 1. However, instead of solving for multi-robot optimization Problem 2 we now solve for single-robot consensus optimization Problem 3 [Algorimth 1 line 6] and the exact penalty function $\phi(\cdot)$ is substituted with $\psi(\cdot)$ [Algorimth 1 lines 7- 8]. Moreover, the single-robot consensus problem 3 is updated after the consensus optimization step with the last trajectory of the other robots $\hat{x}_r$, consensus variables z_r and penalty multipliers λ_r [line 4]. Effectively the newly updated problem penalizes deviation from the consensus trajectory z_r with penalty λ_r while fixing other robots trajectories x_r. This combined with ADMM procedure to compute z_r and λ_r leads to solution convergence [6].

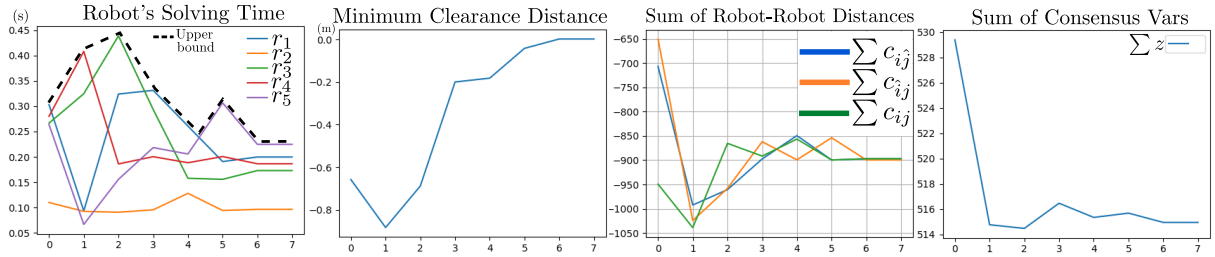

Fig. 2: Evolution of several quantities for each iteration until DiMOpt (Algorithm 3) finds an optimal solution.

Finally, the single robot SCP exit criterion for non-collision [line 27] is alternatively the maximum violation of the set of approximated collision constraints $\tilde{\mathcal{C}}$ instead of the set $\mathcal{C}$.

Once robot SCPs are solved, the trajectories are updated as well as the consensus multipliers and variables according to what is described in lines 10–11. That is, the new trajectory will be an average between the last trajectory and the newly computed one [line 7], while the consensus variable and multipliers are computed according to consensus optimization via the ADMM algorithm with scaled dual variable described in [6] such that $\boldsymbol{z}_r$ converges to $\boldsymbol{q}_r$. Notice that we introduce a momentum term $b\,(\boldsymbol{q}_r^*[k] - \boldsymbol{z}_r[k])$ [line 10] characteristic of heavy ball methods, aimed at accelerating the convergence rate [36], where we decide $b = \frac{R-1}{R}$.

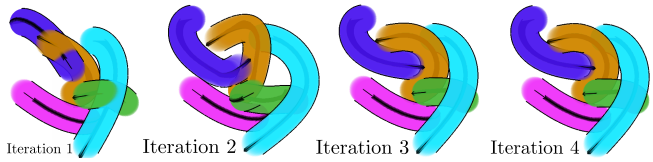

Fig. 3: Trajectories are computed at each iteration of DiMOpt. It. 1: robots compute trajectories without considering collisions. It. 2: orange and purple robots avoid each other, while the blue robot moves outwards; at this iteration, robots are no longer colliding. It. 3: the cost of the trajectories is reduced, correcting a previous overshooting due to over-penalizing of collisions. It. 4: a locally optimal solution is found. See also the video in Section V-C.

1) General Notes: The standard behavior of Algorithm 3 is illustrated in Figures 2 and 3. This represents a particularly difficult problem with 5 robots since in almost all iterations the robots are colliding (minimum clearance distance is negative). The graph entitled Sum of Robot-Robot Distances shows three curves: the real sum of distances between robots c_{ij}, the sum $c_{i\hat{j}}$ as seen from robot i, and the other sum $c_{\hat{i},j}$ as seen from robot j (i.e., collision constraints are pairwise). We can see these converging as the consensus variables z converge. Notice that the solving time of the algorithm iteration is equal to the most computationally demanding robot, i.e., the dotted upper bound in Figure 2. Note also that initial iterations have often a lower solving time. This happens because we reuse the previously computed solution as an initial guess between iterations. Moreover, this downward effect would be more accentuated when the minimum clearance distance is not negative, since intuitively the trajectories would not change as much. Note that if the original Problem 1 were convex, both SCP and DiMOpt methods would converge to the same optimal solution, whereas since the problem is non-convex they are both incomplete and locally optimal. This means the two algorithms can find different locally optimal solutions [5], [6].

V. Experiments and Results

A. Setup

The presented method was tested on a PC running Ubuntu 20.04.3 LTS equipped with a AMD® Ryzen 9 5900x 12-core processor and 24 threads. Our software is available as open source[1], and uses the *CasADi* library [37] for non-linear optimization and optimal control, and the *Open MPI* [38] message passing interface for parallel computing. Each robot is associated with one MPI process with its own memory, thus the need for message passing. When there are more MPI processes/robots than processors ($R > 12$), the performance of OpenMPI and consequently DiMOpt degrades.

Results are presented below. We compare SCP and DiMOpt in terms of scalability, and we further analyze the performance of DiMOpt in five challenging scenarios.

B. Comparison between SCP and DiMOpt

We have conducted a set of 200 experiments per fleet sizes ranging from 2 to 18 robots, where robots with diameter 0.1 m navigate in a square room of 25 m^2 (i.e., four free space constraints per robot, as described in equation (6)). Additionally, we choose the discretization N of the transcribed dynamics such that a robot footprint overlaps between consecutive configurations (i.e., computed using maximum velocity 1m/s^2), and we use as an initial guess a straight line trajectory between start and goal configurations. We compare SCP, DiMOpt and DiMOpt 1. DiMOpt 1 is an altered version of DiMOpt that stops once the first feasible solution is found. Both SCP and DiMOpt stops once paths are collision free and cost has converged.

1) Scaling — Number of Robots: Figure 4 depicts the computation time with respect to the number of robots for SCP and DiMOpt, showing an exponential growth for SCP and a linear growth for DiMOpt. It is also shown that the first feasible (but likely suboptimal) solution of DiMOpt 1 can be found in less than 3 s for 18 robots.

[1] https://github.com/joaosalvado/DiMOpt

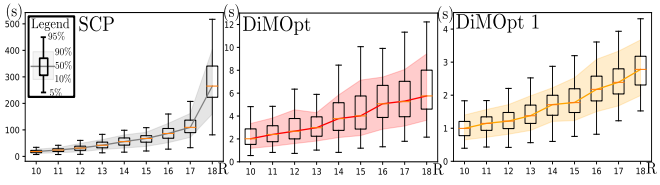

Fig. 4: Computation time (y-axis) of SCP, DiMOpt and DiMOpt 1 vs. number of robots.

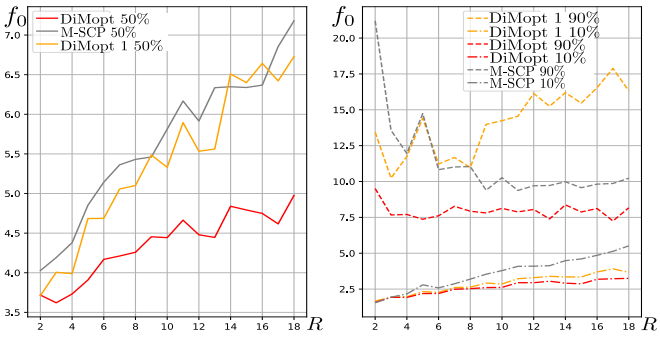

Fig. 5: Value of the cost function for the SCP and DiMOpt methods and for the first solution (DiMOpt 1) for the 10%, 50%, and 90% quantile.

2) Solution Quality: Figure 5 plots the quality of the solutions obtained by SCP, DiMOpt, and DiMOpt 1. DiMOpt yields solutions with lower cost (both on average and for the 95-percentile) than SCP. This is unexpected, since SCP is known to compute solutions with the lowest cost in the literature; however DiMOpt is also locally optimal and therefore the solution of both methods depends on the initial guess. Since the problem is non-convex, DiMOpt will explore the solution space differently, which could explain the result. Interestingly, we can see that the cost of the final solution of SCP and DiMOpt 1 is rather similar on average, however when the number of robots is greater than 8, DiMOpt 1 incurs a higher cost. This can be because robots require more coordination in difficult cases, which makes it more likely that the initial solution obtained by DiMOpt is suboptimal.

C. DiMOpt on selected challenging scenarios

The scenarios illustrated in Figure 6 are particularly difficult to solve using DiMOpt. This is due to high interference between robots, as seen in the multiple crossings between robot trajectories. For a better intuition about the level of interference, please see the video[2] showing the execution of the computed trajectories. Table I shows that the "Circle" scenario is the hardest, followed by the "Take Over", "Cross" and "One Down" scenarios with similar difficulty, and finally the "Square Sided" scenario.

VI. Conclusions and Future Work

We have proposed DiMOpt, an approach for solving the Multi-robot Trajectory Planning problem effectively with a mixture of sequential convex programming and distributed consensus optimization methods. We have evaluated our approach by comparing it with a baseline multi-robot SCP method. We have shown that DiMOpt outperforms the baseline, and that it can quickly find a feasible suboptimal solution before converging to the final locally optimal one.

We have detected that DiMOpt and SCP both fail or require longer computation time when the fleet scenario occupation is high. For instance, when the scenario occupation is 50% the success rate is around 70%. Such highly constrained scenarios are out of the scope of our work and are known to be hard to solve.

DiMOpt assumes that each robot navigates in a polygon free of obstacles, hence, in order to deploy this algorithm in an environment with obstacles, we would require a receding horizon strategy that provides consecutive obstacle-free convex polygons [35]. Also, trajectory duration in DiMOpt is estimated using the distance between start and goal and a constant reference speed. However, the longer the trajectory, the harder it is to estimate goal boundary state constraints associated with trajectory duration. This may limit applicability to online fleet planning, but could potentially be addressed by using a function that models the progress of robots from start to goal state, as done in [39] to measure the progress of a vehicle following a road centerline.

In future work, we intend to integrate DiMOpt with a multi-robot coordination framework suitable for industrial settings [2], where trajectories computed via DiMOpt are used as references, and safety and liveness are ensured via online revision of precedences.

Acknowledgments. This work is supported by Vinnova project AutoHauler and KKS Synergy TeamRob.

[2] https://odysee.com/@joao.salvado:7/dimopt:a

References

[1] J. K. Verma and V. Ranga, "Multi-robot coordination analysis, taxonomy, challenges and future scope," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 1, pp. 1–36, 2021.

[2] A. Mannucci, L. Pallottino, and F. Pecora, "On provably safe and live multi-robot coordination with online goal posting," *IEEE Transactions on Robotics*, pp. 1–19, 2021.

[3] J. Li, A. Tinka, S. Kiesel, J. W. Durham, T. S. Kumar, and S. Koenig, "Lifelong multi-agent path finding in large-scale warehouses." in *AAMAS*, 2020, pp. 1898–1900.

[4] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. Siam, 2010, vol. 19.

[5] J. Duchi, S. Boyd, and J. Mattingley, "Sequential convex programming," in *Lecture Notes EE364b*. Stanford Univ., 2018.

[6] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[7] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*. IEEE, 2012, pp. 1917–1922.

[8] J. Tordesillas and J. P. How, "Mader: Trajectory planner in multiagent and dynamic environments," *IEEE Transactions on Robotics*, 2021.

[9] Y. Mao, M. Szmuk, X. Xu, and B. Açikmese, "Successive convexification: A superlinearly convergent algorithm for non-convex optimal control problems," *arXiv preprint arXiv:1804.06539*, 2018.

[10] M. Bennewitz, W. Burgard, and S. Thrun, "Optimizing schedules for prioritized path planning of multi-robot systems," in *Proceedings 2001 ICRA. IEEE Int. Conference on Robotics and Automation (Cat. No. 01CH37164)*, vol. 1. IEEE, 2001, pp. 271–276.

[11] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, no. 1-4, p. 477, 1987.

[12] D. Bareiss and J. van den Berg, "Generalized reciprocal collision avoidance," *Int. J. Rob. Res. (IJRR)*, vol. 34, no. 12, pp. 1501–1514, 2015.

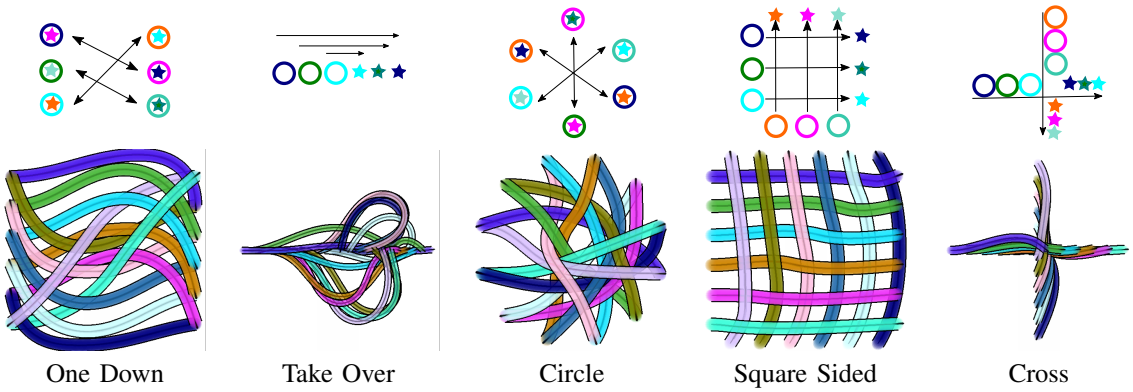

Fig. 6: Trajectories for five symmetrical problems with 12 robots.

	One Down			Take Over			Circle			Square Sided			Cross		
Number of Robots (R)	6	12	20	3	6	12	6	12	20	6	12	20	6	8	12
Iteration of First Solution	7	2	7	4	11	2	24	10	12	2	2	2	2	7	1
Time to compute first solution (s)	1.1	1.4	13	0.6	2.4	1.5	5	6	30	0.2	1	2	0.2	1.4	0.7
Time to compute final solution (s)	1.9	2.8	17	1	3	4.2	6	7	36	0.8	2	6	1	2	3.7

TABLE I: Results for the scenarios depicted in Figure 6 with different numbers of robots.

[13] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.

[14] J. Salvado, M. Mansouri, and F. Pecora, "A network-flow reduction for the multi-robot goal allocation and motion planning problem," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, 2021, pp. 2194–2201.

[15] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[16] K. Solovey and D. Halperin, "k-color multi-robot motion planning," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 82–97, 2014.

[17] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[18] D. P. Bertsekas, "Nonlinear programming," *Journal of the Operational Research Society*, vol. 48, no. 3, pp. 334–334, 1997.

[19] A. Nedić and J. Liu, "Distributed optimization for control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 77–103, 2018.

[20] J. Schulman, J. Ho, A. X. 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.

[21] Y. Mao, M. Szmuk, and B. Açıkmeşe, "Successive convexification of non-convex optimal control problems and its convergence properties," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 3636–3641.

[22] Y. Chen, M. Cutler, and J. P. How, "Decoupled multiagent path planning via incremental sequential convex programming," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 5954–5961.

[23] M. T. Shahab and M. Elshafei, "Distributed optimization of multi-robot motion with time-energy criterion," in *Path Planning for Autonomous Vehicles-Ensuring Reliable Driverless Navigation and Control Maneuver*. IntechOpen, 2019.

[24] A. Falsone, K. Margellos, S. Garatti, and M. Prandini, "Dual decomposition for multi-agent distributed optimization with coupling constraints," *Automatica*, vol. 84, pp. 149–158, 2017.

[25] B. Houska, J. Frasch, and M. Diehl, "An augmented lagrangian based algorithm for distributed nonconvex optimization," *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1101–1127, 2016.

[26] A. Engelmann, Y. Jiang, B. Houska, and T. Faulwasser, "Decomposition of nonconvex optimization via bi-level distributed aladin," *IEEE Transactions on Control of Network Systems*, vol. 7, no. 4, pp. 1848–1858, 2020.

[27] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[28] J. Park and H. J. Kim, "Online trajectory planning for multiple quadrotors in dynamic environments using relative safe flight corridor," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 659–666, 2020.

[29] W. Hönig, J. A. Preiss, T. S. Kumar, G. S. Sukhatme, and N. Ayanian, "Trajectory planning for quadrotor swarms," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 856–869, 2018.

[30] C. E. Luis, M. Vukosavljev, and A. P. Schoellig, "Online trajectory generation with distributed model predictive control for multi-robot motion planning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 604–611, 2020.

[31] B. Şenbaşlar, W. Hönig, and N. Ayanian, "Robust trajectory execution for multi-robot teams using distributed real-time replanning," in *Distributed autonomous robotic systems*. Springer, 2019, pp. 167–181.

[32] R. M. Murray, Z. Li, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[33] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *Journal of guidance, control, and dynamics*, vol. 10, no. 4, pp. 338–342, 1987.

[34] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[35] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Algorithmic Foundations of Robotics XI*. Springer, 2015, pp. 109–124.

[36] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *Ussr computational mathematics and mathematical physics*, vol. 4, no. 5, pp. 1–17, 1964.

[37] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[38] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall, "Open MPI: Goals, concept, and design of a next generation MPI implementation," in *Proceedings, 11th European PVM/MPI Users' Group Meeting*, Budapest, Hungary, September 2004, pp. 97–104.

[39] P. F. Lima, G. C. Pereira, J. Mårtensson, and B. Wahlberg, "Progress maximization model predictive controller," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2018, pp. 1075–1082.