GPU Acceleration of Joint Multi-Agent Trajectory Optimization

Dipanwita Guhathakurta, Fatemeh Rastgar, M Aditya Sharma, Madhava Krishna and Arun Kumar Singh

***Abstract*— Joint multi-agent trajectory optimization is conventionally considered intractable due to the exponential scaling of the number of collision avoidance constraints and linear increase in the number of variables by increasing the number of agents. On the other hand, the joint formulation allows access to more feasible space leading to better coordination maneuvers. In this paper, we try to improve the scalability of joint multi-agent trajectory optimization. Our core idea involves breaking the joint problem into several decoupled smaller quadratic programming (QP) problems and parallelizing them over GPUs. We compare the performance of our optimizer with the state of the arts in terms of trajectory quality and computation time and show substantial improvement in both metrics.**

I. Introduction

Multi-agent trajectory optimization algorithms have attracted a lot of attention in research communities due to their wide range of applications in self-driving cars [1], search and rescue [2], and exploration of large environments.

In multi-agent trajectory optimization, a team of agents aims to find trajectories between their initial and final locations in a shared environment while satisfying conditions such as avoiding collisions with obstacles and other moving agents in their environment [3]. There are two core challenges in this regard: firstly, how to present a suitable formulation of collision avoidance constraints, which are known to be non-convex [4], and secondly, how to handle the cooperation between agents in collision avoidance constraints as the number of robots increases [5], [6]. While several approaches have been proposed to deal with collision avoidance constraints and pair-wise cooperation among agents, they can be broadly classified into two groups: centralized and distributed.

A centralized optimization is an approach that computes the trajectory of all robots together. Centralized optimization can be categorized into sequential methods [7], [6] and joint optimization methods [8], which compute trajectories of agents sequentially and simultaneously, respectively. Centralized methods, particularly joint optimization approaches, provide a rigorous treatment of the collision avoidance constraints, but they often necessitate solving a large optimization problem and typically run offline. In addition, since their computation time increases exponentially with the number of agents, they become intractable for systems with a large number of agents.

On the other hand, distributed methods such as distributed model predictive control (DMPC) obtain the trajectory of each robot decoupled from the others [9]. Distributed approaches are scalable for a large number of agents, and they can run in real-time. But they may have a lower success rate of collision avoidance, especially in obstacle-rich environments, as the computed trajectories are only based on the prediction of the other agents, not the actual trajectories followed by the agents.

Our main motivation in this paper is to improve the computational tractability of the joint variant of the multi-agent trajectory optimization problem in the way that our optimizer can find trajectories for tens of agents in a small fraction of a second ($\approx 0.15s$ for 32 agents). Since this improvement is several orders of magnitudes faster than the existing methods [10], [11], we can ensure that our algorithm is applicable for not only computing global trajectories offline but also online re-planning.

Our main idea is to break the joint multi-agent trajectory optimization at each iteration into several smaller, distributed sub-problems and solve them efficiently in parallel. The author in [11] utilized a similar method and parallelized the optimization problem over separate CPU threads. However, due to the limited number of CPU cores and thread synchronization issues, this parallelization is not scalable with the number of agents [12]. Thus, in this paper, we aim to parallelize trajectory optimization over GPUs. This, in turn, requires bringing some key changes in the algebraic structure of the problem and forms the main contribution of our work as summarized below.

1) We break the joint multi-agent trajectory optimization into several smaller distributed decoupled problems.
2) We reduce the decoupled sub-problems in the form of special Quadratic Programming (QP) problems. Interestingly, all the QPs associated with the decoupled sub-problems have the same matrices, and only the vector part of the QP changes across the problem instances. We show how the solution process of such special QPs can be easily parallelized over GPUs.
3) We compare the performance of our optimizers with the state of the art [7], [8] in terms of computation time and trajectory qualities, including smoothness cost and arc length. For the first comparison, we show our optimizer outperforms [7] in terms of trajectory quality and computation time. For example, in the presence of 16 agents, our computation time is at least 76% lower than [7], and this gap even increases in more crowded environments. Also, our optimizer has better performance than [8] in all the mentioned performance metrics.

The remainder of this paper is organized as follows. Some preliminaries, background, and general problem formulation are provided in Section II. The reformulated distributed

optimization problem and its analysis are proposed in Section III. Benchmarks and statistical results are shown in Section IV. Conclusion and future works are presented in Section V.

II. Background and Preliminaries

This section defines symbols and notations used all over the paper before representing the general problem formulation. Then we review the existing literature about the current problem.

A. Symbols and Notations

In this paper, the lower normal letters show the scalars. The lower and upper bold cases represent vectors and matrices, respectively. The left superscript k and the right superscripts T stand for iteration index and the vector or matrix transpose. The time dependency of the variable is shown by t. The subscripts i and j denote the agent index. n_p and n_r show the number of planning steps and the number of robots. The rest of the symbols will be defined in the first place of use.

B. Problem Formulation

In this note, we address the trajectory optimization problem for a group of holonomic agents that aims to go from an initial state $t = t_0$ to a final state $t = t_f$ while optimizing the squared norm of acceleration along x, y and z axis at each time instant and avoiding collisions. By considering the agents' model as spheroids with dimensions (a, a, b), the problem can be formulated as follows:

$$\min_{x_i(t),y_i(t),z_i(t)} \sum_{t,i} \Big(\ddot{x}_i^2(t) + \ddot{y}_i^2(t) + \ddot{z}_i^2(t) \Big), \quad \text{(1a)}$$

$$\big[x_i(t),\dot{x}_i(t),\ddot{x}_i(t),y_i(t),\dot{y}_i(t),\ddot{y}_i(t),z_i(t),\dot{z}_i(t),\ddot{z}_i(t)\big]_{t=\overline{t_0}} = \mathbf{b}_{o,i}, \quad \text{(1b)}$$

$$\big[x_i(t),\dot{x}_i(t),\ddot{x}_i(t),y_i(t),\dot{y}_i(t),\ddot{y}_i(t),z_i(t),\dot{z}_i(t),\ddot{z}_i(t)\big]_{t=\overline{t_f}} = \mathbf{b}_{f,i}, \quad \text{(1c)}$$

$$-\frac{(x_i(t)-x_j(t))^2}{a^2} - \frac{(y_i(t)-y_j(t))^2}{a^2} - \frac{(z_i(t)-z_j(t))^2}{b^2} + 1 \leq 0,$$
$$\forall t, \{i, j \in \{1, 2, ..., n_r\}, j \neq i\}, \quad \text{(1d)}$$

where, $(x_i(t), y_i(t), z_i(t))$ represents the position of the i^{th} agent at timestamp t. The cost function (1a) minimizes the acceleration along the x, y, z axis at each time instant for all the agents. Initial and final boundary conditions, applied on positions, velocities, and accelerations, are shown in (1b) and (1c). The collision avoidance constraints among agents can be formulated as inequality constraints (1d).

Solving (1a)-(1d) is complicated as the inequality constraints (1d) are non-convex. This complexity increases with adding the number of agents to the environment as a result of exponential increases in the number of inequality constraints. For each agent, the trajectory along each motion axis can be parameterized with n_v decision variables at each time instant. Now, by having n_p planning horizon steps and n_r agents, the number of decision variables in cost function and collision avoidance constraints are $n_v * n_r$ and $\binom{n_r}{2} * n_p$. Different literature [13], [7], [6] have addressed the obstacle avoidance (1d) in multi-agent trajectory optimization. Thus, in the next subsection, we classify different existing methods and propose our methodology.

C. Related Works

A simple approach to multi-agent trajectory optimization problem (1a)-(1d) is to solve one large optimization problem in which trajectories of all agents are computed simultaneously [10], [14]. Augugliaro et al. in [10] reformulates the trajectory optimization problem (1a)-(1d) as a sequence of quadratic programs (QPs) by using first-order Taylor expansion and linearizing the non-convex collision avoidance constraints. Mellinger et al. in [14] rephrased (1a)-(1d) as mixed-integer quadratic program (MIQP) problem by applying integer constraints for obstacle avoidance. The computation times of both [10], [14] scale poorly with an increase in the number of agents. In our prior work [8], we substantially improved the scalability of the joint multi-agent trajectory optimization. In particular, we reformulated the collision avoidance constraints in polar form and applied the Alternating Direction Method of Multipliers (ADMM) to the problem. In addition, we showed that our optimizer is two orders of magnitude faster than [10].

Sequential optimizers solve the multi-agent trajectory optimization problem (1a)-(1d) by computing trajectories for only one agent at a time [7]. For each planning step, each agent considers other agents as dynamic obstacles and finds its own collision-free trajectory. The dynamic obstacles are, in fact, other agents whose trajectories have been computed in the previous planning cycles. Since decision variables belonging to only one agent are presented in each planning cycle, the number of decision variables does not increase with the number of agents. In sequential planning, the agent that is considered latter on in the cycle has access to lesser feasible space than agents that came before it. Thus, sequential planning often run into infeasibility issues.

To improve the scalability issue, [11], [15] introduce distributed optimizers that decouple the problem into several smaller sub-problems. Distributed methods discard the coupling among pair-wise collision avoidance constraints and reduce the optimization problem to n_r decoupled problems. One way to get this decoupling is as follows: let each agent predict other agents' trajectories in the current iteration and use this prediction for solving optimization problem in subsequent iterations. If we denote the predicted trajectories by $(\overline{x}_j(t), \overline{y}_j(t), \overline{z}_j(t))$, then the collision avoidance constraints can be rewritten as:

$$-\frac{(x_i(t)-\overline{x}_j(t))^2}{a^2} - \frac{(y_i(t)-\overline{y}_j(t))^2}{a^2} - \frac{(z_i(t)-\overline{z}_j(t))^2}{b^2} + 1 \leq 0 \quad (2)$$

Fig.1 shows how predictions of agent's trajectories lead to decoupled sub-problems. Various distributed multi-agent optimizers utilize different prediction approaches [9], [11]. In this paper, similar to [11], we use precomputed trajectories from the previous iterations in our collision avoidance constraints.

A common technique to speed up multi-agent trajectory optimization is to leverage parallel architectures such as CPUs and GPUs. Due to the ability to process arbitrary numerical computations, CPUs can solve a batch of optimization problems without changing the underlying numerical

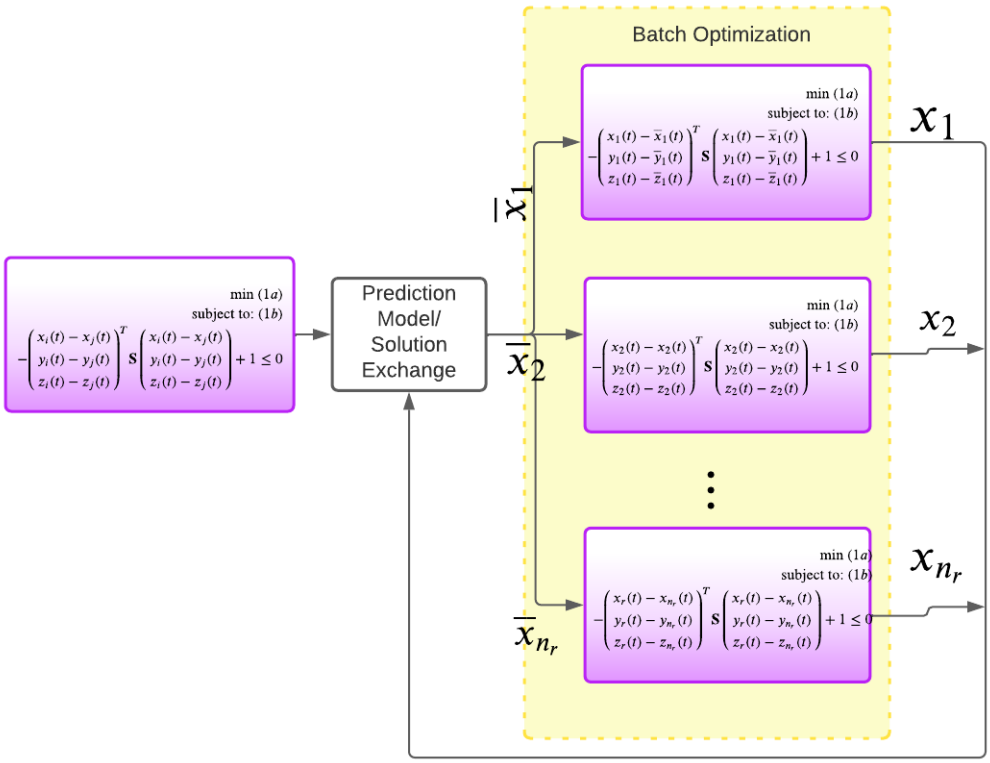

Fig. 1. All the agents communicate their current trajectories. In the next iteration, each agent this prior communicated trajectories to form the collision avoidance constraints at the next planning cycle. This in turn allows each agent to act independently. In other words, the communication strategy takes a joint trajectory optimization problem (first block on the left) and converts it into n_r decoupled problems. Our approach is GPU accelerated parallelized solution of the decoupled sub-problems

algebra of the optimizer [11], [12]. However, as the number of cores is limited in CPUs, they are not suitable for handling a large batch size [16]. On the contrary, although GPUs are only efficient for parallelizing primitive operations such as summation and multiplication, they have many cores and can be used to massively parallelize computations [16]. Thus, to leverage the true potential of GPUs, we need to rewrite the underlying numerical algebra of the optimizers in a suitable form. Gradient Descent (GD) is one of the most well-known methods for accelerating over GPUs since it boils down to just matrix-vector multiplication [17]. However, GD typically works poorly on constrained sub-problems and is sensitive to hyper-parameters like weights, learning rate, etc [17].

III. Method

In this section, we present our main algorithmic result, an efficient joint optimizer for multi-agent trajectory optimization. We begin by stating a simple remark that will be useful throughout the paper.

Remark 1: The time dependent variable $x_i(t)$ and its derivatives can be parameterized using way-point parameterization as follows:

$$\begin{bmatrix} x_i(t_1) \\ x_i(t_2) \\ \dots \\ x_i(t_n) \end{bmatrix} = \mathbf{P}\mathbf{c}_{x_i}, \begin{bmatrix} \dot{x}_i(t_1) \\ \dot{x}_i(t_2) \\ \dots \\ \dot{x}_i(t_n) \end{bmatrix} = \dot{\mathbf{P}}\mathbf{c}_{x_i}, \begin{bmatrix} \ddot{x}_i(t_1) \\ \ddot{x}_i(t_2) \\ \dots \\ \ddot{x}_i(t_n) \end{bmatrix} = \ddot{\mathbf{P}}\mathbf{c}_{x_i}, \quad (3)$$

where $\mathbf{P}$ is generated using time-dependent polynomial basis functions and $\mathbf{c}_{x_i}$ are coefficients associated with the basis functions. Similar parametric representations can be applied for $y_i(t)$ and $z_i(t)$ and their derivatives (see [18] for more detail).

A. Overview

This section introduces an overview of our main idea. In particular, we present a special class of QPs and how their solution can be accelerated over GPUs. To this end, consider (4)

$$\min_{\boldsymbol{\xi}_i} \left(\frac{1}{2} \boldsymbol{\xi}_i^T \overline{\mathbf{Q}} \boldsymbol{\xi}_i + \overline{\mathbf{q}}_i^T \boldsymbol{\xi}_i \right), \quad \text{st: } \overline{\mathbf{A}} \boldsymbol{\xi}_i = \overline{\mathbf{b}}_i, \ i \in \{1, 2, ..., n_r\} \quad (4)$$

where $\boldsymbol{\xi}_i$ is the optimization variable required to be solved for n_r different QP problems. The QPs have a special structure consisting of the Hessian matrix, $\overline{\mathbf{Q}}$, and the equality constrained matrix, $\overline{\mathbf{A}}$, are constant across the batches. This feature helps us to reformulate the optimization problem as a set of linear equations as (5)

$$\begin{bmatrix} \overline{\mathbf{Q}} & \overline{\mathbf{A}}^T \\ \overline{\mathbf{A}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_i \\ \boldsymbol{\mu}_i \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{q}}_i \\ \overline{\mathbf{b}}_i \end{bmatrix}, \qquad \forall i \in \{1, 2, ..., n_r\} \quad (5)$$

where $\boldsymbol{\mu}_i$ are the dual optimization variables. Since the matrix in (5) does not depend on the batch index i, the

optimization variables for all batches can be computed in parallel through (6).

$$\begin{bmatrix} \boldsymbol{\xi}_1 & | & \cdots & | & \boldsymbol{\xi}_{n_r} \\ \boldsymbol{\mu}_1 & | & \cdots & | & \boldsymbol{\mu}_{n_r} \end{bmatrix} = \overbrace{\left(\begin{bmatrix} \overline{\mathbf{Q}} & \overline{\mathbf{A}}^T \\ \overline{\mathbf{A}} & \mathbf{0} \end{bmatrix}^{-1}\right)}^{matrix} \overbrace{\begin{bmatrix} \overline{\mathbf{q}}_1 & | & \overline{\mathbf{q}}_2 & | & \cdots & \overline{\mathbf{q}}_{n_r} \\ \overline{\mathbf{b}}_1 & | & \overline{\mathbf{b}}_2 & | & \cdots & \overline{\mathbf{b}}_{n_r} \end{bmatrix}}^{\text{stacked vectors}}, \tag{6}$$

where $|$ represents that the columns are horizontally stacked. The solution (6) can be solved by multiplying a constant matrix into a batch of vectors in parallel using off-the-shelf libraries like JAX.

In the next section, we decribe how each decoupled sub-problems shown in Fig.1 can be reformulated to have the same structure as QP 4. As a result, all the sub-problems can be solved in parallel and further the computations can be reduced to GPU sutiable form presented in (6).

B. Distributed Optimization Problem Reformulation

By rewriting the collision avoidance constraints (1d) in the polar form (see [8]-[18]), the i^{th} decoupled sub-problem shown in Fig. (1) can be formulated in the following manner.

$$\min_{\substack{x_i(t),\, y_i(t),\, z_i(t),\\ d_{ij}(t),\, \alpha_{ij}(t),\, \beta_{ij}(t)}} \sum_t \Big(\ddot{x}_i^2(t) + \ddot{y}_i^2(t) + \ddot{z}_i^2(t)\Big), \tag{7a}$$

$$\big[x_i(t),\dot{x}_i(t),\ddot{x}_i(t),y_i(t),\dot{y}_i(t),\ddot{y}_i(t),z_i(t),\dot{z}_i(t),\ddot{z}_i(t)\big]_{t=\overline{t_0}} = \mathbf{b}_{o,i}, \tag{7b}$$

$$\big[x_i(t),\dot{x}_i(t),\ddot{x}_i(t),y_i(t),\dot{y}_i(t),\ddot{y}_i(t),z_i(t),\dot{z}_i(t),\ddot{z}_i(t)\big]_{t=\overline{t_f}} = \mathbf{b}_{f,i}, \tag{7c}$$

$$\mathbf{f}_c\!:\!\left\{\begin{array}{l} x_i(t) - \overline{x}_j(t) - a d_{ij}(t)\sin\beta_{ij}(t)\cos\alpha_{ij}(t) = 0 \\ y_i(t) - \overline{y}_j(t) - a d_{ij}(t)\sin\beta_{ij}(t)\sin\alpha_{ij}(t) = 0 \\ z_i(t) - \overline{z}_j(t) - b d_{ij}(t)\cos\beta_{ij}(t) = 0 \end{array}\right\}, \tag{7d}$$

$$d_{ij}(t) \geq 1, \qquad \forall t, j, \{j | j \in \{1,2,...,n_r\}, j \neq i\} \tag{7e}$$

where unknown variables $\alpha_{ij}(t), \beta_{ij}(t)$ and $d_{ij}(t)$ need to be computed. Intuitively, $\alpha_{ij}(t), \beta_{ij}(t)$ and $d_{ij}(t)$ show the ratio of the length of the line-of-sight vector and 3D solid angles of the line of sight vector between agents i and j (see [8]). Also, $(\overline{x}_j(t), \overline{y}_j(t), \overline{z}_j(t))$ are the j^{th} agent's predicted position at time t.

Using Remark (1) and simplifying (7a)-(7e) by defining $\boldsymbol{\xi}_{1,i} = (\mathbf{c}_{x,i}, \mathbf{c}_{y,i}, \mathbf{c}_{z,i})$, $\boldsymbol{\xi}_{2,i} = \boldsymbol{\alpha}_{ij}$, $\boldsymbol{\xi}_{3,i} = \boldsymbol{\beta}_{ij}$ and $\boldsymbol{\xi}_{4,i} = \mathbf{d}_{ij}$, the optimization problem can be written as:

$$\min_{\boldsymbol{\xi}_{1,i}, \boldsymbol{\xi}_{2,i}, \boldsymbol{\xi}_{3,i}, \boldsymbol{\xi}_{4,i}} \Big(\frac{1}{2}\boldsymbol{\xi}_{1,i}^T \mathbf{Q} \boldsymbol{\xi}_{1,i}\Big), \tag{8a}$$

$$\mathbf{A}_{eq}\boldsymbol{\xi}_{1,i} = \mathbf{b}_{eq}, \tag{8b}$$

$$\mathbf{F}\boldsymbol{\xi}_{1,i} = \mathbf{g}_i(\boldsymbol{\xi}_{2,i}, \boldsymbol{\xi}_{3,i}, \boldsymbol{\xi}_{4,i}), \tag{8c}$$

$$\boldsymbol{\xi}_{4,i} \geq \mathbf{1}, \tag{8d}$$

where $\mathbf{Q}$ is a block diagonal matrix with elements $\ddot{\mathbf{P}}^T\ddot{\mathbf{P}}$ in the main diagonal and

$$\mathbf{A}_{eq} = \begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{0} & \mathbf{A} \end{bmatrix}, \mathbf{A} = \big[\mathbf{P}_0 | \dot{\mathbf{P}}_0 | \ddot{\mathbf{P}}_0 | \mathbf{P}_{-1} | \dot{\mathbf{P}}_{-1} | \ddot{\mathbf{P}}_{-1}\big]^T,$$
$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_o & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_o & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{F}_o \end{bmatrix}, \mathbf{g}_i = \begin{bmatrix} \overline{\mathbf{x}}_j + a\mathbf{d}_{ij}\sin\boldsymbol{\beta}_{ij}\cos\boldsymbol{\alpha}_{ij} \\ \overline{\mathbf{y}}_j + a\mathbf{d}_{ij}\sin\boldsymbol{\beta}_{ij}\sin\boldsymbol{\alpha}_{ij} \\ \overline{\mathbf{z}}_j + b\mathbf{d}_{ij}\cos\boldsymbol{\beta}_{ij} \end{bmatrix} \tag{9}$$

where, $\mathbf{P}_1, \dot{\mathbf{P}}_1, \ddot{\mathbf{P}}_1, \mathbf{P}_{-1}, \dot{\mathbf{P}}_{-1}, \ddot{\mathbf{P}}_{-1}$ represent the first and last elements of the corresponding matrices. $\mathbf{F}_o$ is generated by vertically stacking $\mathbf{P}$, $n_r - 1$ times. Also, $\mathbf{b}_{eq}$ is generated by stacking the initial and final boundary values, $\mathbf{b}_{0,i}$ and $\mathbf{b}_{f,i}$, respectively.

Remark 2: All the non-convex constraints (7d) are reformulated as equality constraints (8c).

Now utilizing the Augmented Lagrangian Method, the non-convex equality constraints (7d) are added as penalty terms to our cost function (8a) in the following manner:

$$\min_{\boldsymbol{\xi}_{1,i}, \boldsymbol{\xi}_{2,i}, \boldsymbol{\xi}_{3,i}, \boldsymbol{\xi}_{4,i}} \Big(\frac{1}{2}\boldsymbol{\xi}_{1,i}^T \mathbf{Q} \boldsymbol{\xi}_{1,i} - \langle \boldsymbol{\lambda}_i, \boldsymbol{\xi}_{1,i}\rangle + \frac{\rho}{2}\left\|\mathbf{F}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i(\boldsymbol{\xi}_{2,i}, \boldsymbol{\xi}_{3,i}, \boldsymbol{\xi}_{4,i})\right\|_2^2\Big) \tag{10}$$

The optimization problem (10) can be solved through Algorithm 1 which will be explained in the next subsection.

Algorithm 1 Distributed Batch Optimizer Algorithm for the i^{th} Agent

1: *Initialize* ${}^k\boldsymbol{\xi}_{2,i}$, ${}^k\boldsymbol{\xi}_{3,i}$ *and* ${}^k\boldsymbol{\xi}_{4,i}$ *at iteration* $k = 0$
2: **while** $k \leq$ max iteration or till norm of the residuals are below some threshold **do**
3: *Update* ${}^{k+1}\boldsymbol{\xi}_{1,i}$ *through*

$${}^{k+1}\boldsymbol{\xi}_{1,i} = \min_{\boldsymbol{\xi}_{1,i}} \Big(\frac{1}{2}\boldsymbol{\xi}_{1,i}^T \mathbf{Q} \boldsymbol{\xi}_{1,l} - \langle {}^k\boldsymbol{\lambda}_i, \boldsymbol{\xi}_{1,i}\rangle + \frac{\rho}{2}\Big\|\mathbf{F}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^k\boldsymbol{\xi}_{2,i}, {}^k\boldsymbol{\xi}_{3,i}, {}^k\boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) \; st. \mathbf{A}_{eq}\boldsymbol{\xi}_1 = \mathbf{b}_{eq} \tag{11}$$

4: *Update* ${}^{k+1}\boldsymbol{\xi}_{2,i}$ *through*

$${}^{k+1}\boldsymbol{\xi}_{2,i} = \min_{\boldsymbol{\xi}_{2,i}} \Big(\frac{\rho}{2}\Big\|\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i(\boldsymbol{\xi}_{2,i}, {}^k\boldsymbol{\xi}_{3,i}, {}^k\boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) \tag{12}$$

5: *Update* ${}^{k+1}\boldsymbol{\xi}_{3,i}$ *through*

$${}^{k+1}\boldsymbol{\xi}_{3,i} = \min_{\boldsymbol{\xi}_{3,i}} \Big(\frac{\rho}{2}\Big\|\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^{k+1}\boldsymbol{\xi}_{2,i}, \boldsymbol{\xi}_{3,i}, {}^k\boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) \tag{13}$$

6: *Update* ${}^{k+1}\boldsymbol{\xi}_{4,i}$ *through*

$${}^{k+1}\boldsymbol{\xi}_{4,i} = \min_{\boldsymbol{\xi}_{4,i}} \Big(\frac{\rho}{2}\Big\|\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^{k+1}\boldsymbol{\xi}_{2,i}, {}^{k+1}\boldsymbol{\xi}_{3,i}, \boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) \tag{14}$$

7: *Update Lagrange multiplier coefficient through*

$${}^{k+1}\boldsymbol{\lambda}_i = {}^k\boldsymbol{\lambda}_i - \rho(\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^{k+1}\boldsymbol{\xi}_{2,i}, {}^{k+1}\boldsymbol{\xi}_{3,i}, {}^{k+1}\boldsymbol{\xi}_{4,i}))\mathbf{F} \tag{15}$$

8: **end while**
9: *Return* ${}^{k+1}\boldsymbol{\xi}_{1,i}, {}^{k+1}\boldsymbol{\xi}_{2,i}, {}^{k+1}\boldsymbol{\xi}_{3,i}, {}^{k+1}\boldsymbol{\xi}_{4,i}$

C. Distributed Batch Optimizer Solution Steps

Computing optimization variable $\boldsymbol{\xi}_{1,i}$: Since the optimization problem (11) has a similar structure to (4) with $\overline{\mathbf{Q}} = \mathbf{Q} + \rho\mathbf{F}^T\mathbf{F}$ and $\overline{\mathbf{q}}_i = -{}^k\boldsymbol{\lambda}_i - (\rho\mathbf{F}^T\mathbf{g}_i({}^k\boldsymbol{\xi}_{2,i}, {}^k\boldsymbol{\xi}_{3,i}, {}^k\boldsymbol{\xi}_{4,i}))^T$, it can be solved for all the agents in parallel and the exact solution is given by (6).

Computing optimization variable $\boldsymbol{\xi}_{2,i}$: The problem for computing optimization variable in (12) can be rewritten as (6) by using (3) and updated optimization variable $\boldsymbol{\xi}_{1,i}$ from the previous step.

$$ {}^{k+1}\boldsymbol{\alpha}_{ij} = \min_{\boldsymbol{\alpha}_{ij}} \frac{\rho}{2} \left\| \begin{matrix} \overbrace{{}^{k+1}\mathbf{x}_i - \overline{\mathbf{x}}_j}^{{}^{k+1}\tilde{\boldsymbol{x}}_i} - a^k \mathbf{d}_{ij} \sin {}^k\boldsymbol{\beta}_{ij} \cos \boldsymbol{\alpha}_{ij} \\ \overbrace{{}^{k+1}\mathbf{y}_i - \overline{\mathbf{y}}_j}^{{}^{k+1}\tilde{\boldsymbol{y}}_i} - a^k \mathbf{d}_{ij} \sin {}^k\boldsymbol{\beta}_{ij} \sin \boldsymbol{\alpha}_{ij} \end{matrix} \right\|_2^2 \tag{16} $$

where $\overline{\mathbf{x}}_j, \overline{\mathbf{y}}_j$ are formed by stacking $\overline{x}_j(t), \overline{y}_j(t)$ at different time instants.

Although (16) is non-convex, the solution can be computed using a geometrical intuition; the equation (16) can be seen as a projection of ${}^{k+1}\tilde{\boldsymbol{x}}_i$ and ${}^{k+1}\tilde{\boldsymbol{y}}_i$ onto an axis-aligned ellipse centered at origin with dimensions ad_{ij} and bd_{ij}. Thus, it can be reformulated as

$$ {}^{k+1}\boldsymbol{\xi}_{2,i} = {}^{k+1}\boldsymbol{\alpha}_{ij} = \arctan 2({}^{k+1}\tilde{\boldsymbol{y}}_i, {}^{k+1}\tilde{\boldsymbol{x}}_i), \tag{17} $$

Computing optimization variable $\boldsymbol{\xi}_{3,i}$: Similar to the previous solution step, we update the $\boldsymbol{\xi}_{3,i}$ in the following manner

$$ \boldsymbol{\xi}_{3,i}{}^{k+1} = {}^{k+1}\boldsymbol{\beta}_{ij} = \arctan 2(\frac{{}^{k+1}\tilde{\boldsymbol{x}}_i}{a \cos {}^{k+1}\boldsymbol{\alpha}_{ij}}, \frac{{}^{k+1}\tilde{\boldsymbol{z}}_i}{b}) \tag{18} $$

Computing optimization variable $\boldsymbol{\xi}_{4,i}$: For given positions for the i^{th} and j^{th} agents, ${}^{k+1}\mathbf{d}_{ij}$ are not only independent from each other, but also independent at different time instances. Thus, (14) splits into $n_p * (n_r - 1)$ decoupled problems which can be solved analytically.

$$ {}^{k+1}\boldsymbol{\xi}_{4,i} = {}^{k+1}\mathbf{d}_{ij} = $$

$$ \min_{\mathbf{d}_{ij} \geq 1} \frac{\rho}{2} \left\| \begin{matrix} \overbrace{{}^{k+1}\mathbf{x}_i - \mathbf{x}_j}^{{}^{k+1}\tilde{\mathbf{x}}_i} - a\mathbf{d}_{ij} \sin {}^{k+1}\boldsymbol{\beta}_{ij} \cos {}^{k+1}\boldsymbol{\alpha}_{ij} \\ \overbrace{{}^{k+1}\mathbf{y}_i - \mathbf{y}_j}^{{}^{k+1}\tilde{\mathbf{y}}_i} - a\mathbf{d}_{ij} \sin {}^{k+1}\boldsymbol{\beta}_{ij} \sin {}^{k+1}\boldsymbol{\alpha}_{ij} \\ \overbrace{{}^{k+1}\mathbf{z}_i - \mathbf{z}_j}^{{}^{k+1}\tilde{\mathbf{z}}_i} - b\mathbf{d}_{ij} \cos {}^{k+1}\boldsymbol{\beta}_{ij} \end{matrix} \right\|_2^2 \tag{19} $$

Remark 3: Computing optimization variables in (17)-(19) does not require any matrix-matrix products or matrix factorization. It only needs an element-wise operation, which can be done for all the agents in parallel.

IV. Benchmarks

In this section, we validate the performance of our optimizer and compare our results with the state-of-the-art [7] and [8] in terms of computation time and trajectory quality including smoothness and arc-length.

A. Implementation Details:

Simulations for this section are run on a desktop computer with 32 GB RAM and RTX 2080 NVIDIA GPU. Also, to accelerate linear algebraic computations on GPU, we run our optimizers in Python using JAX [19]. Static obstacles are modeled as agents with fixed zero velocities. The code is available at `https://github.com/susiejojo/distributed_GPU_multiagent_trajopt`.

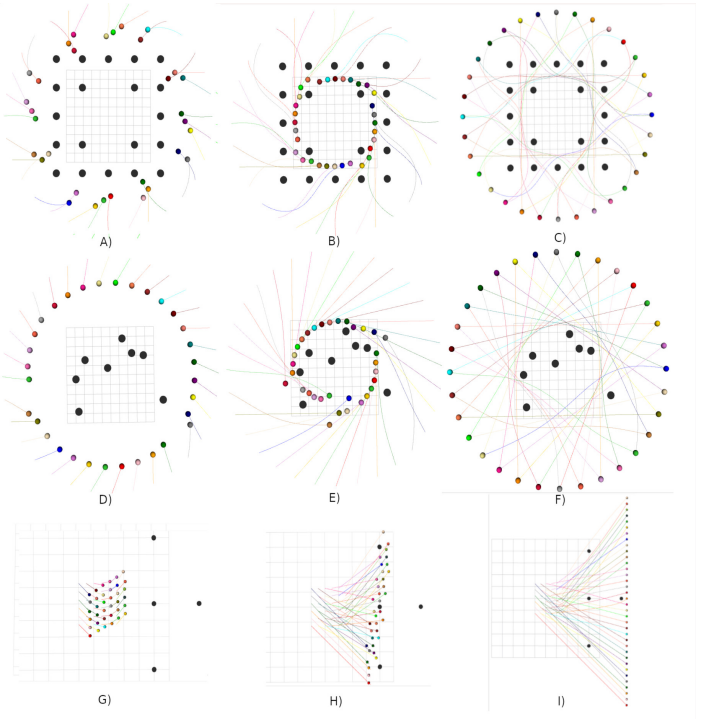

Fig. 2. Trajectory snapshots for (A-C) 32 agents, with radius 0.3m and 20 obstacles of radius 0.4m, (D-F) 32 agents, with radius 0.3m and 8 randomly placed obstacles of radius 0.4m, (G-I) 36 agents with radius 0.1m arranged in a grid configuration are required to move to a line formation. Also, there are 4 static obstacles with radius 0.15m.

B. Benchmarks

We tested our optimizer under the following benchmarks.

- The agents' start and goal positions are sampled along the circumference of a circle.
- The agents are initially located on a grid and are tasked to converge to a line formation.

To show qualitative results, several configurations by changing the number and position of agents and obstacles are created. For example, Fig. (2)(A-I) illustrates three snapshots of results with different agents, obstacles, and various initial and final positions.

We conceptually validate the convergence of our optimizer by plotting the constraints residual over iterations Fig. (3). If these residuals have a decreasing trend over iterations and converge to zero, trajectories are collision-free. Since this trend is satisfied in Fig. (3), the trajectories returned by our optimizer ensure the agents do not collide with each other and obstacles.

Table I compares our optimizer with the state of the art [7] and [8]. The distributed batch optimizer has the best computation time in all the experiments, between 0.15s and 0.20s, and this time does not increase on increasing the number of agents and obstacles. Our optimizer is in the worst case is about two and four times faster than [7] and [8]. By increasing the number of obstacles and agents, the computation time gap between our optimizer and [7] and [8] increases even further. For the case of 32 agents with 20

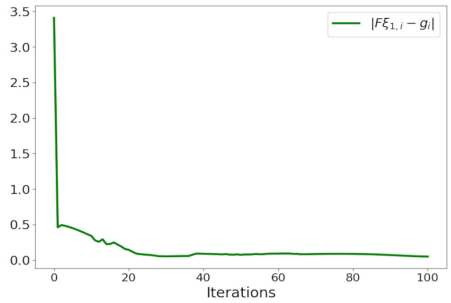

Fig. 3. Validating optimizer convergence empirically. Residuals $\|\mathbf{F}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i\|$ are averaged over all agents and across different benchmarks.

obstacles, [7] and [8] are around 60 and 6 times slower than our optimizer, respectively.

In terms of arc-length, our algorithm and [7] are almost similar and are at least 32% better than [8] for 16 agents. For 32 agents, the distributed batch method provides slightly shorter arc lengths than [8], and around 10% shorter than [7].

Compared with [7], the smoothness cost for the distributed batch method achieves an average reduction of 35.86% and 59.06% for the 16 and 32 agents benchmarks, respectively. Also, the smoothness cost for 16 agents shows at least 20% improvement in comparison with [8]. For the case of 32 agents, our optimizer and [8] almost provide similar results.

TABLE I

COMPARISON OF OUR OPTIMIZER WITH [7] IN TERMS OF COMPUTATION TIME, ARC-LENGTH AND SMOOTHNESS FOR 16 AND 32 AGENTS WITH DIFFERENT NUMBER OF OBSTACLES.

		16 agents					32 agents		
		2 Obs	4 Obs	8 Obs	12 Obs	24 Obs	12 Obs	16 Obs	20 Obs
Comp. time	[8]	0.34	0.37	0.70	0.79	1.49	1.68	1.752	1.80
	ours	**0.15**	**0.16**	**0.16**	**0.17**	**0.17**	**0.19**	**0.20**	**0.20**
	[7]	0.62	0.70	0.68	0.66	0.79	12.50	12.42	11.82
Arc-length	[8]	13.48	14.25	15.53	15.93	24.19	23.85	24.04	24.14
	ours	**9.99**	11.69	11.11	11.19	10.24	**22.59**	**22.03**	**23.15**
	[7]	10.30	**10.67**	**10.74**	**10.60**	15.79	26.21	26.15	26.29
Smoothness	[8]	0.10	0.11	0.14	0.15	0.16	0.15	0.16	**0.17**
	ours	**0.048**	**0.093**	**0.08**	**0.106**	**0.06**	**0.13**	**0.12**	0.21
	[7]	0.11	0.11	0.11	0.11	0.3	0.36	0.36	0.36

V. CONCLUSION AND FUTURE WORKS

By leveraging mathematical reformulations and GPU-based parallelization, our optimizer computes trajectories for tens of agents in cluttered environments within a fraction of a second. In comparison with state-of-the-art baseline approaches, we achieve improvement in terms of not only the computation time, but also trajectory quality. Our approach has the potential to be extended to non-holonomic multi-agent systems such as cars, as well as for interaction-aware trajectory prediction.

REFERENCES

[1] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 1826–1848, 2019.

[2] J. Berger and N. Lo, "An innovative multi-agent search-and-rescue path planning approach," *Computers & Operations Research*, vol. 53, pp. 24–31, 2015.

[3] S. H. Semnani, H. Liu, M. Everett, A. De Ruiter, and J. P. How, "Multi-agent motion planning for dense and dynamic environments via deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3221–3226, 2020.

[4] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2020.

[5] S. Kandhasamy, V. B. Kuppusamy, and S. Krishnan, "Scalable decentralized multi-robot trajectory optimization in continuous-time," *IEEE Access*, vol. 8, pp. 173 308–173 322, 2020.

[6] 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.

[7] J. Park, J. Kim, I. Jang, and H. J. Kim, "Efficient multi-agent trajectory planning with feasibility guarantee using relative bernstein polynomial," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 434–440.

[8] F. Rastgar, H. Masnavi, J. Shrestha, K. Kruusamäe, A. Aabloo, and A. K. Singh, "Gpu accelerated convex approximations for fast multi-agent trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3303–3310, 2021.

[9] 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.

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

[11] J. Bento, N. Derbinsky, J. Alonso-Mora, and J. S. Yedidia, "A message-passing algorithm for multi-agent trajectory planning," *Advances in neural information processing systems*, vol. 26, 2013.

[12] V. K. Adajania, A. Sharma, A. Gupta, H. Masnavi, K. M. Krishna, and A. K. Singh, "Multi-modal model predictive control through batch non-holonomic trajectory optimization: Application to highway driving," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4220–4227, 2022.

[13] J. Li, M. Ran, and L. Xie, "Efficient trajectory planning for multiple non-holonomic mobile robots via prioritized trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 405–412, 2020.

[14] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *2012 IEEE international conference on robotics and automation*. IEEE, 2012, pp. 477–483.

[15] T. Halsted, O. Shorinwa, J. Yu, and M. Schwager, "A survey of distributed optimization methods for multi-robot systems," *arXiv preprint arXiv:2103.12840*, 2021.

[16] K. Barkalov and V. Gergel, "Parallel global optimization on gpu," *Journal of Global Optimization*, vol. 66, no. 1, pp. 3–20, 2016.

[17] M. Hamer, L. Widmer, and R. D'andrea, "Fast generation of collision-free trajectories for robot swarms using gpu acceleration," *IEEE Access*, vol. 7, pp. 6679–6690, 2018.

[18] F. Rastgar, A. K. Singh, H. Masnavi, K. Kruusamae, and A. Aabloo, "A novel trajectory optimization for affine systems: Beyond convex-concave procedure," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 1308–1315.

[19] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, *et al.*, "Jax: composable transformations of python+ numpy programs," *Version 0.2*, vol. 5, pp. 14–24, 2018.