A Novel Trajectory Optimization for Affine Systems: Beyond Convex-Concave Procedure

Fatemeh Rastgar, Arun Kumar Singh, Houman Masnavi, Karl Kruusamae, Alvo Aabloo

Abstract—Trajectory optimization problems under affine motion model and convex cost function are often solved through the convex-concave procedure (CCP), wherein the non-convex collision avoidance constraints are replaced with its affine approximation. Although mathematically rigorous, CCP has some critical limitations. First, it requires a collision-free initial guess of solution trajectory which is difficult to obtain, especially in dynamic environments. Second, at each iteration, CCP involves solving a convex constrained optimization problem which becomes prohibitive for real-time computation even with a moderate number of obstacles, if long planning horizons are used.

In this paper, we propose a novel trajectory optimizer which like CCP involves solving convex optimization problems but can work with an arbitrary initial guess. Moreover, the proposed optimizer can be computationally up to a few orders of magnitude faster than CCP while achieving similar or better optimal cost. The reduced computation time, in turn, stems from some interesting mathematical structures in the optimizer which allows for distributed computation and obtaining solutions in symbolic form. We validate our claims on difficult benchmarks consisting of static and dynamic obstacles.

I. INTRODUCTION

In this paper, we consider trajectory optimization problems where the underlying motion model is affine and the cost functions are convex quadratic. These seemingly simple set-up has been extensively used for motion planning of robots like quadrotors [1], [2], [3]. Our long term goal is to develop an optimizer that can run efficiently on embedded hardware with limited computation power. To this end, we intend to develop algorithms where computations can be parallelized, and the use of extensive numerical computations like matrix factorization/inverse is minimized.

The main complexity of the trajectory optimization considered in this paper stems from the collision avoidance constraints that are often modeled as quadratic inequalities. For example, if the robot is modeled as a disk and the obstacles as axis-aligned ellipses, then the collision avoidance constraint can be represented in the following manner.

\[ f_c(x_t, y_t) = -\left(\frac{(x_t - i_{tx})^2}{l_{tx}^2} - \frac{(y_t - i_{ty})^2}{l_{ty}^2}\right) + 1 \leq 0, \forall t, \tag{1} \]

where, \( x_t, y_t \) are the position of the robot and \( i_{tx}, i_{ty} \) are the center of the \( i^{th} \) ellipse at time \( t \). The constants \( l_{tx} \) and \( l_{ty} \) are respectively the length of the major and minor axis of the ellipse inflated by the radius of the robot. The inequality (1) is non-convex (with respect to \( x_t, y_t \)) and thus it is challenging to incorporate it within a trajectory optimization algorithm. Nonetheless, techniques like gradient descent [4] can be used to obtain a locally optimal solution.

An alternate more efficient approach would be to exploit the structure in (1). In particular, note that the inequality (1) is purely concave and thus it can be globally upper bounded by its affine approximation. As a result, instead of using (1) to avoid collisions, we can use its affine approximation. This simple reformulation forms the basis of CCP [5] and has been extensively used in many recent works on motion planning [1], [2], [3], [6], [7] although, sometimes under the name of sequential convex programming. However, in this paper, we use the term CCP as it is more specific to linearizing concave collision avoidance constraints under affine motion model. 1 The advantage of CCP is that it does not require any line search (or learning rate in gradient descent) or trust-region constraints which limits the progress of the optimizer at each iteration in standard non-linear optimization techniques [5].

Although mathematically rigorous, CCP has some critical limitations which impedes its scalability in cluttered and dynamic environments. Firstly, in its original form, CCP requires a collision-free guess of the initial trajectory, which is difficult to obtain, especially in dynamic environments. Some recent works [1], [5] have proposed successful workarounds. For example, [5] proposes the use of slack variables to handle infeasible initial guess. In [1], authors propose a two-step method where they first compute a semi-definite relaxation of (1) to compute a feasible initial guess and then apply CCP. All these techniques substantially increase the computational cost of CCP based motion planning. For example, the approach of [5] introduces one slack variable for each collision avoidance constraints, thus increasing the dimensionality of the optimization problem.

The second limitation of CCP is that it requires solving a constrained optimization problem at each iteration. Even for moderate number of obstacles (≈ 10) but long planning horizon (≈ 1000 steps), the number of inequalities in this constrained optimization can become large and prohibit real time implementation of CCP.

Contributions: In this paper, we make the case of modeling collision avoidance through the following equations (see Section III-A for the motivation and intuition behind it):

1The term sequential convex programming is very general and is often used in works like [8] where the motion model is non-linear and thus, the collision avoidance constraints would not have a purely concave form.
Equation (2) is a parametric representation of (1) and it is straightforward to extend it to 3D obstacle avoidance as well. Our main contribution in this paper is to propose a convex approximation of a trajectory optimization that has (2) as the collision avoidance constraints. At the core of our formulation is a simple observation that (2) is bi-convex in \( i d_t \cos i \alpha_t \) and \( i d_t \sin i \alpha_t \). We exploit this structure through techniques like Alternating Direction Method of Multipliers (ADMM) [9] and by building on our prior work [10, 11, 12]. In addition to the ability to work with arbitrary initial guess, the proposed optimizer provides the following advantages over CCP:

- The per iteration computation complexity of the proposed optimizer is significantly less than CCP. Furthermore, it increases linearly with the number of collision avoidance constraints, in contrast to a sharp non-linear increase in CCP (see Fig.5). As a result, the proposed optimizer can be up to several orders of magnitude faster than a CCP based approach depending on the number of obstacles and length of the planning horizon.
- The reduced computation time of the proposed optimizer stems from its interesting mathematical structure. For example, some parts of it involve solving unconstrained quadratic programming (QP) or equivalently linear equations in parallel. The remaining parts of our optimizer have symbolic solutions. That is, the solutions are available as formulae that can be evaluated without requiring any computation of matrix products, factorization or inverse.

We show that the proposed optimizer can achieve near real time performance on several difficult benchmarks with static and dynamic obstacles with around 10^4 collision avoidance constraints. We also compare it with CCP with achieved optimal cost and computation time as our metric.

II. PRELIMINARIES AND REVIEW OF CCP

A. Symbols and Notations

We will use lower case normal font letters to represent scalars while bold font variants would represent vectors. We will represent matrices through bold font upper case letters. The subscript \( t \) represents the time stamp of the variables and the superscript \( T \) would represent the transpose of vectors or matrices. The right superscript \( k \) and the left superscript \( i \) would respectively represent the iteration and obstacle index.

B. CCP

Given a solution trajectory \((x^k_t, y^k_t)\) at iteration \( k \), CCP involves solving the following optimization problem at iteration \( k + 1 \) [5].

\[
(x^{k+1}_t, y^{k+1}_t) = \underset{x_t, y_t, u_t}{\arg \min} f_x + f_y + \sum_i w_i u_i. \tag{3a}
\]

\[
\frac{\partial f_x}{\partial x_t} (x^k_t - x^k_t) + \frac{\partial f_y}{\partial y_t} (y^k_t - y^k_t) + f_i(x^k_t, y^k_t) - u_t \leq 0. \tag{3b}
\]

\[
u_t \geq 0. \tag{3c}
\]

The cost functions \( f_x, f_y \) are assumed to be convex quadratic function of \( x_t, y_t \) and their derivatives. This particular form can model objectives like trajectory smoothness, tracking, goal reaching, etc. and can be decoupled for motions along the \( x \) and \( y \) directions. The inequality (3b) is the affine approximation of (1) around \((x^k_t, y^k_t)\) with \( \frac{\partial f_x}{\partial x_t}, \frac{\partial f_y}{\partial y_t} \) representing the gradients. A non-negative slack variable \( u_t \) is added to each collision avoidance constraints. The key idea is to gradually drive the slack variables to zero by increasing the penalty \( w \) at each iteration.

One simple way of solving (3a)-(3c) is to parameterize \( x_t, y_t \) and their derivatives through polynomial (e.g splines) and thereby convert the optimization into a QP over the polynomial coefficients. Since one needs to solve (3a)-(3c) several times to reach a locally optimal solution, the main computational bottleneck depends on how many affine constraints (3b) are included in the optimization. This in turn depends on the number of obstacles and the number of planning steps (or length of the planning horizon). Authors in [5] propose a simple heuristic for reducing the computation time of CCP which involves accommodating only those inequalities in (3b) which are close to the feasible boundary (near zero). Our experiments show that this heuristic is very sensitive to problem parameters such as number of planning steps, choice of cost functions etc. Furthermore, incorporating only a subset of the inequalities at each iteration may affect convergence and sometimes can even lead to an increase in computation time.

C. Bi-Convexity, Multi-Convexity and Alternating Minimization

In this subsection, we discuss some key concepts which forms the foundation of the proposed optimizer.

Definition 1. A function \( g(\xi_1, \xi_2) \) is bi-convex (bi-affine) if for a given \( \xi_1 \), \( g \) is convex (affine) in \( \xi_2 \) and vice-versa.

Definition 2. A function \( g(\xi_1, \xi_2, \xi_3) \) is multi-convex (multi-affine) if fixing two of the variables makes it convex (affine) with respect to the remaining variable. For example, fixing \((\xi_1, \xi_2)\) makes \( g \) convex (affine) in \( \xi_3 \).

Alternating Minimization: Alternating (or Gauss Seidel) minimization is an efficient technique for optimizing over bi-convex or multi-convex functions. For example, the iteration for the bi-convex function has the following form [13]

\[
k+1\xi_1 = \arg \min_{\xi_1} g(\xi_1, k+1\xi_2), k+1\xi_2 = \arg \min_{\xi_2} g(k+1\xi_1, \xi_2). \tag{4}
\]

Each of the minimization in (4) is a convex optimization problem.

D. Bounds as Quadratic Penalties

In this subsection, we present a concept from [14] to show how affine constraints can be reformulated as quadratic penalties. To this end, consider the following optimization problem
\min f(\xi), \quad A\xi \leq b \tag{5}

Optimization (5) can be reformulated in the following manner using the concept of ADMM.

\min f(\xi) + \frac{\rho}{2} \|A\xi - b - u - \frac{\lambda}{\rho}\|^2. \tag{6}

Optimization (6) is solved for given values of \(u, \lambda, \rho\) which are in turn updated based on the solution (see [14]).

\section{Main Results}

In this section, we present our main theoretical result, a trajectory optimizer that surpasses CCP in terms of computational efficiency. We begin by pointing out some assumptions used by our formulation.

\textbf{Assumptions}

- Obstacles are modeled as axis aligned ellipses (in 2D) or ellipsoids (in 3D). Extension to rotated ellipses/ellipsoids is straightforward and is briefly discussed in Section V.

- We consider affine systems which are also differentially flat and as a result control inputs can be obtained from trajectory derivatives. This is true for a wide class of robots including quadrotors and wheeled mobile robots [1]. This assumption allows us to not explicitly include affine constraints mapping state to control inputs in our optimizer. However, extension of our formulation to a more general affine system is straightforward and does not affect its computational structure in any way.

\subsection{Motivating the Proposed Collision Avoidance Constraints}

Fig. 1 shows the intuition behind the collision avoidance constraints (2). Consider a point \((x_t, y_t)\) on the trajectory which is inside the obstacle. Collision can be avoided by pushing this point away from the center \((c_t, s_t)\) along a direction \(i\alpha_t\). Our hypothesis is that an appropriate \(i\alpha_t\) is one which aids in the reduction of the cost \(f_x, f_y\). Our idea is loosely related to CHOMP motion planner [4], wherein the trajectory points inside the obstacle are pushed away in a direction orthogonal to the motion direction. This in turn aids in maintaining trajectory smoothness. In contrast, our idea is more general in the sense that the cost functions \(f_x, f_y\) can be arbitrary convex quadratic, modeling not only smoothness but also trajectory tracking, goal-reaching, etc.

The role of \(i\alpha_t\) in (2) is rather simple as it just dictates how much the \((x_t, y_t)\) needs to be pushed from the obstacle center. We present the following analytical choice for it. For a given \(i\alpha_t\), (7) essentially pushes the point inside the obstacle to the boundary and leaves any point outside the obstacle unperturbed.

\[\left(\alpha_t \right)_i = \max \left(1, \sqrt{\frac{(x_t - x_i)^2}{l_x^2} + \frac{(y_t - y_i)^2}{l_y^2}}\right), \forall i, t \tag{7}\]

In (10), \(n, m\) represent the number of obstacles and planning steps respectively. As evident, the equality constraints in (8b) has been relaxed using a combination of quadratic penalties and linear terms multiplied with Lagrange multipliers \(\lambda_{c_t}^i, \lambda_{s_t}^i\). Along similar lines, we have also relaxed the equality constraints in (8c) as quadratic penalties. We call it the consensus penalty because it tries to bring an agreement between \(i\alpha_t\) and \(\lambda_{c_t}^i, \lambda_{s_t}^i\). The terms involving Lagrange multipliers \(\lambda_{c_t}^i, \lambda_{s_t}^i\) in the consensus penalty helps to drive its residual to zero.
On the first look, (9) looks like a generic non-linear programming problem but rather it has a nice mathematical structure. Specifically, for a given $i \alpha_k$, $\mathcal{L}$ is bi-convex in the space of $(x_t, y_t, d_t)$ and $(c_t, s_t)$. To highlight how this structure is useful, we next apply Alternating Minimization (AM) concept to solve (9). Starting with a guess for $i d^0_k, i c^0_k$ at $k = 0$, the AM proceeds through the iterates (14a)-(14e) presented in Algorithm 1. As before, the superscript $k$ is used to denote the value of the respective variable at iteration $k$. Optimization (14a)-(14e) are all convex problems and we present a deeper insight into these optimizations next.

**Steps (14a)-(14b):** For a given $i d^k, i c^k, i s^k$, these two optimizations are de-coupled from each other and thus can be solved in parallel. Further, let us assume that $x_t$ and its derivatives are parameterized in the following form

$$
\begin{bmatrix}
x_t^1 \\
x_t^2 \\
\vdots \\
x_t^n
\end{bmatrix} = P_{c_t},
\begin{bmatrix}
x_t^1 \\
x_t^2 \\
\vdots \\
x_t^n
\end{bmatrix} = \tilde{P}_{c_t},
$$

where, $P$ is a matrix formed with time-dependent basis functions (e.g. polynomials) and $c_t$ are the coefficients of the polynomials. Using (11), $\mathcal{L}_c$ and the cost function $f_c$ which models trajectory smoothness, goal-reaching, trajectory tracking etc. can be put in the following matrix form.

$$f_c(x_t, \dot{x}_t, \ddot{x}_t) + \mathcal{L}_c(\dot{x}_t, \ddot{x}_t) = \frac{1}{2} x^T Q_c x_t + q^T_c x_t$$

(12)

With the help of (11) and (12), the unconstrained QP (14a) can be reduced to solving the following linear equations.

$$(Q_x + \rho_{xy} P_y^T P_x) c_x = -(q_x + \sum_{i=1}^{i=n} P^{T} i \lambda^x \cdot \rho_{xy} P_y^T (x + i d^i d^i c)),$$

(13)

where $i \lambda^x$ is formed by stacking $i \lambda^x_k$ at different time instants. Similar derivation follows for $x, d, c$ as well. We can also reduce the optimization (14b) in the same form as (13) by choosing a similar parametrization as (11) for $y_t$ and its derivatives.

**Steps (14c)-(14d):** These two optimizations are again de-coupled from each other and are thus amenable for parallel computation. Furthermore, we can derive their solutions in symbolic form. To see how, note that for a given obstacle index $i$, $i c_t, i s_t$ at different time instants are decoupled from each other. Similarly, if we form $i c, i s$ by stacking the variables at different time instants, then they will be decoupled across obstacle index $i$. Consequently, optimization (14c)-(14d) can be split into $n \times m$ parallel optimizations, each involving the minimization of a single variable quadratic function, the solution of which can be obtained in a symbolic form.

**Steps (14e):** Optimization (14e) is based on our prior work [10], [11], [12]. The solution of this optimization reduces to just setting $i \alpha_k = \arctan(2 \rho \lambda_k / r_k^2)$. The geometrical intuition of this step is the following: the pair of feasible $i c_t, i s_t$ constitute a circle $(c^2 + s^2) = 1$. Thus, in this step, we take the $i c_t, i s_t$ obtained through steps (14c)-(14d) and project them radially on the boundary of the circle.

**Steps (14g)-(14h):** These steps update the Lagrange multipliers based on the residuals. The update rule for $i \lambda^x_i, i \lambda^y_i$ follows from [9] and that for $i \lambda^c_i, i \lambda^s_i$ follows from our prior work [10]. The weights of the quadratic penalties $\rho_{xy}, \rho$ are also increased in each iteration if the residuals are not below the specified threshold.

### C. 3D Trajectory Optimization

We now extend our results to the 3D case. The collision avoidance constraints in this context takes the following form.

$$x_t = x_t + l_x d_t \sin \alpha_t \cos \beta_t,$$

$$y_t = y_t + l_y d_t \sin \alpha_t \sin \beta_t,$$

$$z_t = z_t + l_z d_t \cos \alpha_t, \alpha_t \in [0, \pi], \beta_t \in [-\pi, \pi], d_t \geq 1,$$

(16)

where, we have assumed that the obstacles are modeled as axis-aligned ellipsoids. The trajectory optimization can be formulated in the following manner.

$$\min_{x_t, y_t, z_t, d_t, i \alpha_t, i \beta_t, i \tau_t} f_x + f_y + f_z + i \lambda^x + i \lambda^y + i \lambda^z$$

(17a)

$$x_t = x_t + l_x d_t \sin \alpha_t \cos \beta_t,$$

$$y_t = y_t + l_y d_t \sin \alpha_t \sin \beta_t,$$

$$z_t = z_t + l_z d_t \cos \alpha_t, \alpha_t \in [0, \pi], \beta_t \in [-\pi, \pi]$$

(17b)

$$\min_{x_t, y_t, z_t, d_t, i \alpha_t, i \beta_t, i \tau_t} f_x + f_y + f_z + i \lambda^x + i \lambda^y + i \lambda^z$$

(17c)

$$\alpha_t \in [0, \pi], \beta_t \in [-\pi, \pi]$$

(17d)

$$d_t \geq 1, \alpha_t \in [0, \pi], \beta_t \in [-\pi, \pi]$$

(17e)

The above formulation is similar to the 2D case and thus we can construct the augmented Lagrangian following the same approach.

$$\mathcal{L} = f_x + f_y + f_z + \mathcal{L}_x + \mathcal{L}_y + \mathcal{L}_z$$

(18)

$$\sum_{i, t} i \lambda^x_i(x_t - x_t - l_x d_t s_t \tau_t) + \frac{\rho_{xy}}{2}(x_t - x_t - l_x d_t s_t \tau_t)^2$$

$$\sum_{i, t} i \lambda^y_i(y_t - y_t - l_y d_t s_t \tau_t) + \frac{\rho_{xy}}{2}(y_t - y_t - l_y d_t s_t \tau_t)^2$$

$$\sum_{i, t} i \lambda^z_i(z_t - z_t - l_z d_t c_t) + \frac{\rho_{xy}}{2}(z_t - z_t - l_z d_t c_t)^2$$

The Lagrangian defined in (18) has a multi-convex structure in contrast to the bi-convex structure obtained for the 2D case in (10). Specifically, for a given $i \alpha_t, i \beta_t$, $\mathcal{L}$ is multi-convex (recall Defn.2) in $(i c_t, i s_t)$, $(x_t, y_t, d_t)$, and $(i \tau_t, i \alpha_t)$. As before, we apply alternating minimization to solve to minimize $\mathcal{L}$ subject to (17g). The solution iterates are presented in Algorithm 2, wherein, the optimizations (15a)-(15i) are convex having exact same mathematical structure as the corresponding optimization in Algorithm 1. Specifically, (15a)-(15c) are decoupled and steps (15d)-(15i) have a symbolic
Algorithm 1 Alternating Minimization for solving (9)
1: Initialize $i\alpha_k, k \leq \text{maxiter}$ or till norm of the residuals are below some threshold do
2: Compute $i\alpha_k^k = \cos i \alpha_k^k, i_z^k = \sin i \alpha_k^k$
   \[ x_t^{k+1} = \arg \min_{x_t} f_x(x_t, \dot{x}_t, \ddot{x}_t) + I_x(x_t, \ddot{x}_t) + \sum_{i=1}^{m} \sum_{t=0}^{m} \alpha_i^k (x_t - \dot{x}_t - \dot{t}_z^k i_z^k i_t^k)^2 + \frac{\rho_{xy}^2}{2} (x_l - \dot{x}_l - \dot{t}_z^k i_z^k i_t^k)^2 \] (14a)
   \[ y_t^{k+1} = \arg \min_{y_t} f_y(y_t, \dot{y}_t, \ddot{y}_t) + I_y(y_t, \ddot{y}_t) + \sum_{i=1}^{m} \sum_{t=0}^{m} \alpha_i^k (y_t - \dot{y}_t - \dot{t}_z^k i_z^k i_t^k)^2 + \frac{\rho_{xy}^2}{2} (y_l - \dot{y}_l - \dot{t}_z^k i_z^k i_t^k)^2 \] (14b)
   \[ i_{\alpha}^{k+1} = \arg \min_{i_{\alpha}} \sum_{i=1}^{m} \sum_{t=0}^{m} \lambda_i (y_t^{k+1} - y_t - \dot{t}_z^k i_z^k i_t^k)^2 + \frac{\rho_{xy}^2}{2} (y_l - \dot{y}_l - \dot{t}_z^k i_z^k i_t^k)^2 \] (14c)
   \[ i_{\alpha}^{k+1} = \arg \min_{i_{\alpha}} \sum_{i=1}^{m} \sum_{t=0}^{m} \lambda_i (y_t^{k+1} - y_t - \dot{t}_z^k i_z^k i_t^k)^2 + \frac{\rho_{xy}^2}{2} (y_l - \dot{y}_l - \dot{t}_z^k i_z^k i_t^k)^2 \] (14d)
3: end while

solution. The Lagrangians can be updated following the same approach as in Algorithm 1. The constraints on $i\alpha_i$ can be ensured by simply clipping the solution at each iteration to $[0, \pi]$.

IV. VALIDATION AND BENCHMARKING

The objective of this section is two-fold: to empirically validate the convergence of the proposed optimizer and qualitatively and quantitatively compare its performance with the CCP.

Setup: We prototyped both the proposed optimizer and CCP in Python using NumPy libraries. Additionally, we used CVXOPT [15] to solve the QPs in each iteration of the CCP. Both the approaches were run on a 2.60GHz laptop with 32 GB RAM, i7-9750H CPU on a single core set up. To minimize the computation time for the CCP, we employed the heuristic proposed in [5] and considered only 6 – 8% of the total number of collision avoidance constraints at any given iteration. However, we reiterate that this heuristic is highly sensitive to problem parameters and it took us several trial and error to arrive at this specific number. We employed the following cost function in our analysis.

\[ f_x = \sum_t \omega_1 x_t^2 + \omega_2 (x_t - x_t^d)^2, \] (19)

where, the first term is the smoothness cost while the second term pertains to tracking of the desired trajectory $x_t^d$. The weights $\omega_1, \omega_2$ trades off the effect of each term. Similar cost functions can be written for the $y$ and $z$ component of the motion as well.

Benchmarks and Typical Paths: We generated three benchmarks with $n = 10$ static or dynamic obstacles. Fig.2(a)-2(b) show our 2D static obstacle benchmark, wherein we considered a narrow corridor like scene and an environment with randomly placed obstacles. We fitted ellipses to the corridor boundaries to formulate the collision avoidance constraints. Fig.3(a)-3(b) show the benchmark with dynamic obstacles. Fig.4(a) show the 3D collision avoidance benchmark. For each benchmark, we generated 15 different problem instances by varying the initial position and velocity for a given final state. The time interval for planning ranged from 15s – 60s depending on the distance between the start and goal position. It was further always discretized into $m = 1000$ steps. Thus, the total number of collision avoidance constraints in all the benchmarks were 10000.

Fig.2(a)-2(b), 3(a)-3(b), and 4(a) also show typical paths obtained with the proposed optimizer (blue) and the CCP (cyan). The desired trajectory to be tracked is shown in magenta. Except for the corridor scene, in general both approaches led to paths in very different homotopies. For the benchmark with dynamic obstacles, we show the distance between the robot and each obstacle over time (Fig.3(c)) to validate collision avoidance.

Convergence Validation: A key validation point for an ADMM based optimizer (such as Algorithm 1, 2) is the decrease in different residuals over iteration. Fig.2(c)-2(d), 3(d)-3(e) present this analysis for different benchmarks by showing the general trend in the residuals of the equality constraints (8b) and the consensus penalty in (10). We observed that on an average 100 iterations are enough to obtain a residual in the order of $10^{-3}$. The rate of decrease of residual was slightly faster for the dynamic obstacle benchmark while it was slowest for the difficult corridor scene shown in Fig.2(a). Due to the lack of space, we do not present the convergence plots for the 3D obstacle benchmark.
Algorithm 2 Alternating Minimization for 3D Trajectory Optimization

1: Initialize $i d_k, i a_k, i b_k$
2: while $k \leq \text{maxiter}$ or till norm of the residuals are below some threshold do

$\begin{aligned}
x_t^{k+1} &= \arg\min_{x_t} f_x(x_t, \dot{x}_t, \ddot{x}_t) + \mathcal{L}_x(\dot{x}_t, \ddot{x}_t) + \sum_{i,t} \lambda_t^x (x_t - x_t - t_i d_k^{i,x} + \lambda_i^{x,t} c_t) + \frac{\rho x y z}{2} (x_t - x_t - t_i d_k^{i,x} + \lambda_i^{x,t} c_t)^2 \tag{15a} \\
y_t^{k+1} &= \arg\min_{y_t} f_y(y_t, \dot{y}_t, \ddot{y}_t) + \mathcal{L}_y(\dot{y}_t, \ddot{y}_t) + \sum_{i,t} \lambda_t^y (y_t - y_t - t_i d_k^{i,y} + \lambda_i^{y,t} c_t) + \frac{\rho x y z}{2} (y_t - y_t - t_i d_k^{i,y} + \lambda_i^{y,t} c_t)^2 \tag{15b} \\
z_t^{k+1} &= \arg\min_{z_t} f_z(z_t, \dot{z}_t, \ddot{z}_t) + \mathcal{L}_z(\dot{z}_t, \ddot{z}_t) + \sum_{i,t} \lambda_t^z (z_t - z_t - t_i d_k^{i,z} + \lambda_i^{z,t} c_t) + \frac{\rho x y z}{2} (z_t - z_t - t_i d_k^{i,z} + \lambda_i^{z,t} c_t)^2 \tag{15c} \\
\lambda_i^{x,t} &= \arg\min_{\lambda_i^{x,t}} \sum_{i,t} \lambda_t^x (x_t^{k+1} - x_t - t_i d_k^{i,x} + \lambda_i^{x,t} c_t) + \frac{\rho x y z}{2} (x_t^{k+1} - x_t - t_i d_k^{i,x} + \lambda_i^{x,t} c_t)^2 + \frac{\rho y}{2} (c_t - \cos \alpha_t + \frac{\lambda_i^{x,t}}{\rho})^2 \tag{15d} \\
\lambda_i^{y,t} &= \arg\min_{\lambda_i^{y,t}} \sum_{i,t} \lambda_t^y (y_t^{k+1} - y_t - t_i d_k^{i,y} + \lambda_i^{y,t} c_t) + \frac{\rho x y z}{2} (y_t^{k+1} - y_t - t_i d_k^{i,y} + \lambda_i^{y,t} c_t)^2 + \frac{\rho y}{2} (c_t - \sin \alpha_t + \frac{\lambda_i^{y,t}}{\rho})^2 \tag{15e} \\
\lambda_i^{z,t} &= \arg\min_{\lambda_i^{z,t}} \sum_{i,t} \lambda_t^z (z_t^{k+1} - z_t - t_i d_k^{i,z} + \lambda_i^{z,t} c_t) + \frac{\rho x y z}{2} (z_t^{k+1} - z_t - t_i d_k^{i,z} + \lambda_i^{z,t} c_t)^2 + \frac{\rho y}{2} (c_t - \sin \beta_t + \frac{\lambda_i^{z,t}}{\rho})^2 \tag{15f} \\
\alpha_i^{k+1} &= \arg\min_{\alpha_i^{k+1}} \sum_{i,t} \lambda_i^{x,t} + \frac{\nu c_t}{\rho} (c_t - \cos \alpha_t + \frac{\lambda_i^{x,t}}{\rho})^2 \tag{15g} \\
\beta_i^{k+1} &= \arg\min_{\beta_i^{k+1}} \sum_{i,t} \lambda_i^{z,t} + \frac{\nu c_t}{\rho} (c_t - \sin \beta_t + \frac{\lambda_i^{z,t}}{\rho})^2 \tag{15h} \\
d_t^{k+1} &= \max(1, \sqrt{(x_t^{k+1} - x_t)^2 + (y_t^{k+1} - y_t)^2 + (z_t^{k+1} - z_t)^2}) \tag{15i} \\
\end{aligned}$

3: end while

Optimal Cost Analysis: Fig.2(e)2-(f), 3(f)-3(g), and 4(b) show the statistics of the optimal cost achieved across different benchmarks. We observe a very mixed trend across different 2D benchmarks. However, if we average out the costs across all of them (Table I), we observe that both the proposed optimizer and CCP achieve very similar smoothness and tracking cost, with CCP being marginally better. For the 3D obstacle benchmark (4(b)), the proposed optimizer produced solutions with substantially lower smoothness cost.

Computation Time Comparison: We now present the first of the two most important result of this paper. Fig.2(g), 3(h), and 4(c) present the statistics of the computation time observed across different benchmarks. As can be seen, for the 2D static obstacle benchmark (Fig.2(g)), the proposed optimizer can achieve an average speed of up to two orders of magnitude with respect to CCP. Moreover, computation times for CCP show a high variance suggesting that the worst case difference in computation time would be even higher. The proposed optimizer achieves similar speed-up in the dynamic obstacle (3(h)) and 3D benchmark (4(c)).

Computation Time Scaling: Fig.5 presents the second most important result of this paper. It shows the the factor by which the computation time for a fixed number of iteration changes with increase in number of collision avoidance constraints by the same factor. The CCP approach shows a sharp increase for a higher number of constraints, which also agrees with a similar observation presented in [3] (see Fig.3 (in [3]))

2Although [3] deals with multi-agent planning, Fig.3 in it is still relevant in our context since it just shows computation time as a function of inequality constraints.
Fig. 2. 2D static obstacle benchmark consisting of a corridor scene (a) and an environment with randomly placed obstacles (b). The paths obtained from the proposed optimizer (blue) and CCP (cyan) are also shown along with the desired trajectory to be tracked (magenta). Fig.(c) and (d) show the general trend of residuals of equality constraints (8b) and the consensus term in (10) with iterations, observed across different problem instances generated for the obstacle configuration (a) and (b). Fig (e)-(f) show the cost statistics seen across problem instances. Fig.(d) present the computation time statistics.

But numerical libraries like Numpy exploit multi-threading for these operations. Secondly, since the rest of our algorithm have symbolic solutions, its evaluation can be easily vectorized to leverage parallel computation. Now, we note that CCP and the proposed optimizer can require different number of iterations to solve a given problem. Nevertheless, this result still highlights an important computational difference between both the approaches. Moreover, as shown earlier, the proposed optimizer does outperform CCP in the benchmarks considered in the paper.
The proposed optimizer substantially outperforms CCP, which is the current state of the art, in terms of computation time while achieving similar or better costs in different benchmark problems. This performance gain in turn was achieved on the back of useful mathematical structures in our formulation which allowed for distributed computation and symbolic solutions. Our formulation modeled obstacles as axis-aligned ellipses/ellipsoids. The extension to rotated variants would have one key difference: the optimization over \( x_t, y_t, z_t \) will be coupled and cannot be solved in parallel. However, they can still be solved one at a time similar to the process shown in Algorithm 1.2.

There are several directions to expand our work. Firstly, we are extending it to a multi-agent setting. Secondly, we intend to introduce a GPU variant of the proposed optimizer to fully exploit the distributed and symbolic structure. More efficient methods can also be developed to optimize over sines and cosine functions in Algorithm 1.2. Finally, we are looking to extend our work to non-linear motion models with applications to manipulation and autonomous driving by building on our prior works [11] and [12].

### References