Fast Second-order Cone Programming for Safe Mission Planning

Kai Zhong[1], Prateek Jain[2], Ashish Kapoor[2]

Abstract— This paper considers the problem of safe mission planning of dynamic systems operating under uncertain environments. Much of the prior work on achieving robust and safe control requires solving second-order cone programs (SOCP). Unfortunately, existing general purpose SOCP methods are often infeasible for real-time robotic tasks due to high memory and computational requirements imposed by existing general optimization methods. The key contribution of this paper is a fast and memory-efficient algorithm for SOCP that would enable robust and safe mission planning on-board robots in real-time. Our algorithm does not have any *external* dependency, can efficiently utilize warm start provided in safe planning settings, and in fact leads to significant speed up over standard optimization packages (like SDPT3) for even standard SOCP problems. For example, for a standard quadrotor problem, our method leads to speedup of $1000\times$ over SDPT3 without any deterioration in the solution quality.

Our method is based on two insights: a) SOCPs can be interpreted as optimizing a function over a polytope with infinite sides, b) a linear function can be efficiently optimized over this polytope. We combine the above observations with a novel utilization of Wolfe's algorithm [1] to obtain an efficient optimization method that can be easily implemented on small embedded devices. In addition to the above mentioned algorithm, we also design a two-level sensing method based on Gaussian Process for complex obstacles with non-linear boundaries such as a cylinder.

I. INTRODUCTION

Safe control of dynamics system is critical for robotics and cyber-physical systems. The uncertainty arising in real situations, due to disturbance, sensor noise and modeling errors, makes it a challenging problem. A popular approach is to model the uncertainty using probabilistic approaches. Most popular probabilistic approaches for achieving safe and robust control include controller synthesis via *chance constraints*, [2], [3], [4], [5], [6]. Chance constraints are useful in handling the uncertainty by requiring that the probability of failure of any state violation is always below a prescribed value. Recently, [7], [8] introduced a Probabilistic Signal Temporal Logic (PrSTL) framework which also models uncertainty in a probabilistic manner with focus on uncertainty in model. This framework invariably leads to second-order cone constraints for modeling the probabilistic safety invariants. Such constraints together with an appropriate cost function form a second-order cone programming (SOCP) which can be solved using general purpose optimization

packages, such as GUROBI [9] and SDPT3 [10] that mostly use interior point methods.

However, these off-the-shelf methods tend to place high demand on computation and memory resources and pose significant challenges in implementing them on embedded chip-sets that reside on robots, quadrotors etc, especially because at least one SOCP problem needs to be solved at *each time step*. For example, consider quadrotors or small mobile robots operating in an uncertain world, which are often constrained both in their ability to carry payload and available power. Under such constraints implementing the above methods for achieving safe and robust control is a non-trivial task.

In this paper, we focus on designing fast and memory-efficient optimization routines that enable fast, safe and robust mission planning for robots operating in uncertain environments. In particular, we show how to efficiently solve SOCPs based on two key observations: a) SOCP's dual is a significantly simpler problem and can be written as minimizing an objective function constrained to a "simple" polytope albeit with infinite sides, b) linear optimization over the polytope is efficient. We use the above two observations along with a classic algorithm named Wolfe's algorithm.

Wolfe's algorithm was first proposed by Wolfe [1] in 1976 and there are many appealing properties of the Wolfe's algorithm which makes it particularly suitable for robotic tasks. First, it can be viewed as a variant of the Frank-Wolfe Algorithm [11], [12], which is particularly suitable for polytope constrained problems due to the fast linear minimization step. Such polytope constraints arise naturally in the setting of chance constraints as well as PrSTL. Moreover, Wolfe's algorithm is guaranteed to converge to the optimum linearly for strongly convex or μ_g-strongly convex objectives [11]. Our experiments show that Wolfe's algorithm is much faster than traditional methods and requires only a little memory. To the best of our knowledge, this is the first time that Wolfe's algorithm is applied to solving SOCPs.

As a case study, we focus on the path planing problem with unknown obstacles for Micro Aerial Vehicles (MAVs, quadrotors) which have several applications ranging from package delivery to monitoring farms etc. These missions require that the robots operate in a partially observed environment and achieve the goals while avoiding obstacles, such as trees, buildings, hills and other aerial vehicles. Several previous works [13], [14], [15], [3], [16] on planning obstacle-free trajectories have assumed the environments are known or can be instantaneous detected accurately. However, in real situations, the obstacles are typically unknown a prior

[1]Kai Zhong (zhongkai@ices.utexas.edu) is with University of Texas at Austin. This work was done while interning at Microsoft Research (MSR).

[2]Prateek Jain (prajain@microsoft.com), Ashish Kapoor (akapoor@microsoft.com) are with MSR.

and the real time detection is non-trivial. Also there have been many approaches for obstacle avoidance in partially observed environments[17], [18], [19], [20], [16], [21].

Our work is especially inspired by recent research in safe controller synthesis using Probabilistic Signal Temporal Logic (PrSTL) [7]. The PrSTL framework was designed for safe controller synthesis in a hybrid dynamic system, where the safety invariants are defined via a distribution of logical expressions that operate on real-valued, dense-time signals. These signals could be functions of the robot state, environment and other safety parameters. The safe controller synthesis then is reduced to constraint optimization problems [8], a sequence of SOCP constraints generated from the PrSTL specifications. We seek to solve such sequences of optimization problems in an efficient manner, thereby enabling implementation of such strategies on real-time systems.

We demonstrate our fast optimization routine on the problem of a real-time trajectory planning under uncertainty. Inline with the PrSTL framework, the two key components of such a system is a sensing module that makes inferences about the environment, and a procedure that uses our fast algorithm to determine safe control inputs given the inferences and the safety invariants. The experiments show that the proposed method is much more faster than traditional methods, which include projected gradient descent and SDPT3. We simulate the quadrotor flight with different types of obstacles and show that our method can efficiently find near-optimal trajectories while avoiding the obstacles.

Paper Organization. In Sec. II, we first describe and define a general SOCP problem in the context of safe mission planning under uncertainty. Following that in Sec.III, we show how to solve the dual form of this problem via Wolfe's algorithm. In Sec. IV, we demonstrate our methods to the optimal and safe path planning for the quadrotors in uncertain environments. In Sec. V, we present empirical comparison of the proposed method with existing approaches and highlight the advantages.

Notations. Plain small letters denote scalars. Bold small letters denote vectors. Capital letters denote matrices. We use I_d to denote a $d \times d$ identity matrix. We use some notations in Matlab, such as $[\boldsymbol{a}; \boldsymbol{b}]$ for the vertical conjunction of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$, $\boldsymbol{a}_{i:j}$ for the entries among i-th entry and the j-th entry of $\boldsymbol{a}$, and $A_{i:j,:}$ for the rows of A from i-th row to j-th row. $[L]$ denotes $1, \cdots, L$ and $[L] - 1$ denotes $0, \cdots, L-1$.

II. PROBLEM STATEMENT

We start with describing the primal and the dual problem that arise in SOCP. Recall that the safe mission planning under uncertainty can be formulated as minimizing a quadratic cost function over future controls constrained in second-order cones [7], [8]. Hence, we focus on the SOCP problem of the following form:

$$\min_{\hat{\boldsymbol{u}}} \|\hat{\boldsymbol{u}} + \hat{\boldsymbol{p}}/2\|^2$$
$$\text{s.t. } \|B_i\hat{\boldsymbol{u}} + \boldsymbol{b}_i\| \le \boldsymbol{c}_i^T\hat{\boldsymbol{u}} + d_i, \forall i = 1, 2, \cdots, L, \quad (1)$$

where $\hat{\boldsymbol{u}} \in \mathbb{R}^n$ can be viewed as some linear transformation of the controls. $\hat{\boldsymbol{p}} \in \mathbb{R}^n, B_i \in \mathbb{R}^{m \times n}, \boldsymbol{b}_i \in \mathbb{R}^m, \boldsymbol{c} \in \mathbb{R}^n, d_i \in$

$\mathbb{R}$ are constants that are given by the control equation as well as the uncertainty distribution. Sec. IV presents a mapping of the safe path planning problem for quadrotors to Problem (1).

Now, note that:

$$\max_{\|\boldsymbol{v}\| \le \lambda} \boldsymbol{v}^T \boldsymbol{s} - \lambda t = \max_{\lambda \ge 0} \lambda(\|\boldsymbol{s}\| - t) = \begin{cases} 0, & \|\boldsymbol{s}\| \le t, \\ \infty, & o.w., \end{cases}$$

Using the above fact we can rewrite Lagrangian of (1) as:

$$\min_{\hat{\boldsymbol{u}}} \max_{\|\boldsymbol{v}_i\| \le \lambda_i} \|\hat{\boldsymbol{u}} + \hat{\boldsymbol{p}}/2\|^2 + \sum_{i=1}^{L}[\boldsymbol{v}_i^T(B_i\hat{\boldsymbol{u}} + \boldsymbol{b}_i) - \lambda_i(\boldsymbol{c}_i^T\hat{\boldsymbol{u}} + d_i)]$$

By exchanging $\min$ and $\max$, we obtain the following dual problem to (1):

$$\min_{\boldsymbol{z}} \|U\boldsymbol{z}\|^2 + \boldsymbol{p}^T\boldsymbol{z},$$
$$\text{s.t.} \|\boldsymbol{v}_i\| \le \lambda_i, \ \forall i = 1, 2, \cdots, L, \quad (2)$$

where $\boldsymbol{z} = [\boldsymbol{v}_1; \lambda_1; \boldsymbol{v}_2; \lambda_2; \cdots; \boldsymbol{v}_L; \lambda_L] \in \mathbb{R}^{(m+1)L}$ is the vector of dual variables. Also,

$$U = \frac{1}{2}[B_1^T, -\boldsymbol{c}_1, B_2^T, -\boldsymbol{c}_2, \cdots, B_L^T, -\boldsymbol{c}_L],$$
$$\boldsymbol{p} = U^T\hat{\boldsymbol{p}} - [\boldsymbol{b}_1; -d_1; \boldsymbol{b}_2; -d_2; \cdots; \boldsymbol{b}_L; -d_L].$$

Given dual optimal $\boldsymbol{z}$, the primal optimal to (1) can be obtained using:

$$\hat{\boldsymbol{u}} = -(\hat{\boldsymbol{p}}/2 + U\boldsymbol{z}). \quad (3)$$

III. WOLFE'S ALGORITHM

In this section, we present a method to solve (2) and equivalently (1). Our method is based on Wolfe's algorithm which is a popular algorithm for submodular optimization [12]. Here, we exploit the two key properties of Wolfe's algorithm: a) it can be used to optimize a convex smooth function over a prototype, b) it requires optimizing a linear function over the polytope.

Note that the constraint set in (2) is a polytope albeit with infinite many sides. Moreover, linear function optimization over the polytope can be performed efficiently (see Section III-B).

Let the objective function in (2) be denoted by: $g(\boldsymbol{z}) = \|U\boldsymbol{z}\|^2 + \boldsymbol{p}^T\boldsymbol{z}$. Wolfe's algorithm proceeds with an outer iteration and an inner iteration. In the outer iteration, the main task is to optimize a linear function over the constraint polytope given in (2). In the inner iteration, we need to solve an affine minimization problem which can be solved efficiently using standard techniques. Algorithm 1 provides a detailed pseudo-code of the Wolfe's algorithm adapted for our problem and the functions in the algorithm are explained in the following subsections. Note that the Wolfe's algorithm requires the constraint set is closed. Therefore, we add one more constraint, which is $\lambda_i \le \lambda_{max}$ for all i. It turns out that a large λ_{max} doesn't affect the speed of the algorithm in practice.

Algorithm 1 [Fail,z^*]=DualSOCP($U, p, L, \lambda_{max}, T, \delta_0, z_0$): Wolfe's algorithm for SOCP.

Input: Objective function $g(z) = \|Uz\|^2 + p^T z$. Constraint set, $\mathcal{P} = \cup_i \{\|v_i\| \leq \lambda_i \leq \lambda_{max}\}$. Maximum number of iterations T. The stopping precision δ_0. A warm start z_0.

Output: Fail, z^*

1: Initialize the active set $\mathcal{A} = \{z_0\}$. $\alpha_{z_0} = 1$. Fail=true
2: **for** $t = 0, 1, 2, \cdots, T$ **do**
3:   $s = \text{LMO}_{\mathcal{P}}(\nabla g(z^{(t)}))$
4:   $\mathcal{A} = \mathcal{A} \cup s$, $\alpha_s = 0$
5:   **while** True **do**
6:     $\beta = \text{AffineMinimize}(U, p, \mathcal{A})$.
7:     **if** $\beta \succeq 0$ **then**
8:       $\alpha \leftarrow \beta$; break
9:     **else**
10:       $\gamma^* = \text{argmin}_{\gamma \geq 0, \gamma\beta + (1-\gamma)\alpha \succeq 0} \; \gamma$
11:       $\alpha = \gamma^* \beta + (1 - \gamma^*)\alpha$
12:       $\mathcal{A} \leftarrow \{a \in \mathcal{A} | \alpha_a > 0\}$, $\alpha \leftarrow \{\alpha_a | a \in \mathcal{A}\}$
13:     **end if**
14:   **end while**
15:   **if** the sub-optimality $\delta < \delta_0$ **then**
16:     Fail=false, $z^* = \sum_{a \in \mathcal{A}} \alpha_a a$, return
17:   **end if**
18: **end for**

A. Sub-optimality

Given a dual variable z, the primal objective can be computed as $f_p = \|Uz\|_2^2$ according to Eq. (3). The duality gap is $d_{gap} = 2\|Uz\|_2^2 + p^T z$. The primal infeasibility of each constraint can be calculated as $I_p(i) = \|B_i \hat{u} + b_i\| - (c_i^T \hat{u} + d_i)$. The dual infeasibility of each constraint is $I_d(i) = \|v_i\| - \lambda_i$. Now we use the following sub-optimality,

$$\delta = \max\left\{\frac{d_{gap}}{|f_p|+1}, \frac{\max\{I_p\}}{\max\{|c_i^T \hat{u} + d_i|+1\}}, \frac{\max\{I_d\}}{\max\{|\lambda_i|+1\}}\right\}$$

as the stopping criterion, which is also called precision in the remaining of this paper.

B. Linear Minimization Oracle (LMO)

The LMO step in Algorithm 1 minimizes the linear approximation of objective function subject to the constraints. It can be formulated as follows: given a gradient g,

$$\min_z \; z^T g$$
$$\text{s.t.} \|v_i\| \leq \lambda_i \leq \lambda_{max}, \; \forall i = 1, 2, \cdots, L \quad (4)$$

where $z = [v_1; \lambda_1; v_2; \lambda_2; \cdots; v_L; \lambda_L]$. Denote $g =: [w_1; \gamma_1; w_2; \gamma_2; \cdots; w_L; \gamma_L]$. Then this problem can be decomposed into L small independent problems,

$$\min_{v_i, \lambda_i} \; v_i^T w_i + \lambda_i \gamma_i$$
$$\text{s.t.} \|v_i\| \leq \lambda_i \leq \lambda_{max}. \quad (5)$$

Given λ_i, to achieve the minimal objective, the optimal $v_i = -\lambda_i w_i / \|w_i\|$. So,

$$\begin{cases} v_i = 0, \lambda_i = 0, & \text{if } \|w_i\| - \gamma_i \leq 0 \\ v_i = -\lambda_{max} w_i / \|w_i\|, \; \lambda_i = \lambda_{max}, & \text{o.w.} \end{cases}$$

C. Affine Minimization

The affine minimization function AffineMinimize() minimizes the objective subject to the affine space spanned by $\mathcal{A}$, which can be formulated as the following problem,

$$\min_{s, \beta} \; \|Us\|^2 + s^T p$$
$$\text{s.t. } s = \sum_{a \in \mathcal{A}} \beta_a a, \sum_{a \in \mathcal{A}} \beta_a = 1. \quad (6)$$

To solve Eq. (6), we consider two cases: if there exist an all-zero atom in $\mathcal{A}$ or not. Let $A \in \mathbb{R}^{(m+1)L \times |\mathcal{A}|}$ be the matrix stacked by the atoms in $\mathcal{A}$.

Case 1: all atoms in $\mathcal{A}$ are non-zero vectors and we assume $A^T U^T U A$ is non-singular. This formulation is equivalent to:

$$\min_\beta \; \|UA\beta\|^2 + \beta^T A^T p, \quad \text{s.t. } \beta^T \mathbf{1} = 1. \quad (7)$$

A closed-form solution exists for this problem:

$$\nu^* = -\frac{\mathbf{1}^T Q^{-1} A^T p + 2}{\mathbf{1}^T Q^{-1} \mathbf{1}}, \quad \beta^* = -\frac{1}{2}(Q^{-1} A^T p + \nu^* Q^{-1} \mathbf{1})$$

where $Q = A^T U^T U A$.

Case 2: one atom in $\mathcal{A}$ is a zero vector. Let $\hat{\mathcal{A}} = \mathcal{A} \backslash \mathbf{0}$ and $\hat{A} \in \mathbb{R}^{(m+1)L \times (|\mathcal{A}|-1)}$ be the matrix stacked by the atoms of $\hat{\mathcal{A}}$. We also assume $\hat{A}^T U^T U \hat{A}$ is non-singular. In this case, we solve the following problem,

$$\min_{\hat{\beta}} \; \|U\hat{A}\hat{\beta}\|^2 + \hat{\beta}^T \hat{A}^T p \quad (8)$$

where $\hat{\beta}$ are the coefficients corresponding to the non-zero atoms. So $\hat{\beta}^* = -\frac{1}{2}(\hat{A}^T U^T U \hat{A})^{-1} \hat{A}^T p$. And the coefficient of the zero atom $\beta_0^* = 1 - \mathbf{1}^T \hat{\beta}^*$. Hence, $\beta^* = \hat{\beta}^* \cup \beta_0^*$.

D. Speeding up the algorithm by utilizing the structure

In this section, we try to exploit the structure of the Wolfe's algorithm and reduce its computational complexity. Calculation of the gradient $\nabla g(z) = 2U^T U z + p$ and computing the AffineMinimize function are the two most computationally intensive steps in Alg. 1. While the dimension of the dual problem $(m+1)L$ is very large for large L or m, typically the number of atoms in the algorithm is small as the algorithm actively removes redundant atoms. So we leverage this structure by maintaining vectors $\{Ua | a \in \mathcal{A}\}$, the values $\{p^T a | a \in \mathcal{A}\}$ and the coefficients of the atoms α instead of maintaining the dual variable $z = \sum_{a \in \mathcal{A}} \alpha_a a$. Note that the outer loop adds at most one more atom to the set $\mathcal{A}$, so maintaining UA takes at most $(m+1)L$ flops (floating point operations) and we can even reduce this complexity by utilizing the sparsity of the new atom.

Now using the maintained items, the calculation of the gradient, $\nabla g(z) = 2U^T (\sum_{a \in \mathcal{A}} \alpha_a (Ua)) + p$, costs $O(n(m+1)L + n|\mathcal{A}|)$ flops.

Similarly, we can compute the AffineMinimize function in $O(|\mathcal{A}|^3 + n|\mathcal{A}|^2)$ which in practice tends to be very efficient as the number of non-zero atoms ($|\mathcal{A}|$) is small. Moreover, the number of inner iterations also tend to be small. Hence, the computational and memory requirement of the approach is dominated by $O(n(m+1)L)$ term.

IV. CASE STUDY: PATH PLANNING FOR QUADROTORS

We consider a real-time safe trajectory planning for quadrotors with unknown obstacles. Lets assume that $x \in \mathbb{R}^{12}$ is a 12 dimensional representation of the state of the quadrotor. Here the first three entries $x_{1:3}$ are the position, $x_{4:6}$ are the translational velocities, $x_{7:9}$ are the Euler angles, and $x_{10:12}$ are the angular velocities. Let $u \in \mathbb{R}^4$ be the control vector consisting of roll, pitch, yaw and vertical thrust input respectively. We use discretized time representations with time interval dt. If x^i, x^{i+1} are the states of i-th and $i + 1$-th time steps respectively, then given the control input u^{i+1} for the transition from x^i to x^{i+1}, we have the following dynamics equation:

$$x^{i+1} = x^i + f(x^i, u^{i+1})dt, \qquad (9)$$

The function f takes the same form as described in [7], [22]. At each discrete time step, the system first senses the nearby obstacles, forms a belief about the feasible region, then finds an optimal safe trajectory. The key challenge here is that the sensing and trajectory planning steps are conducted on-board at each time step, so they need to be fast enough to avoid too much latency. The two components of such a system are: a) a probabilistic framework for modeling obstacles, and b) a procedure to determine the safe trajectory by solving a SOCP (1).

A. Probabilistic Obstacle Detection

We follow [7] to form the probabilistic safety constraints. In the case of collision avoidance, PrSTL results in chance constrains that impose the probability of a collision occurring to be lower than some threshold (i.e. $\mathbb{P}[\text{Collision}] < \epsilon$). This prescribed threshold ϵ is a parameter determined by our risk appetite.

We assume the on-board sensors generate a point cloud, i.e. a mesh grid of points, around the quadrotor, where each point has two states, in-obstacle or out-of-obstacle. Such a point cloud can be realized by LIDAR and vision sensors as discussed in [17]. Similar to Sadigh and Kapoor [7], we employ a linear Gaussian process (GP) to provide the safety constraints. In particular, each point in the point cloud can be labeled either as -1 (out-of-obstacle) or 1 (in-obstacle). We denote the position-label pairs as $\{\xi_j, y_j\}_{j=1,2,\cdots,N}$ for N sensing points. Applying a linear GP on these labeled samples results in a posterior distribution of a linear predictor $n \sim \mathcal{N}(\mu, \Sigma)$ where:

$$\Sigma = (\frac{1}{\lambda}Z^T Z + S_4)^{-1}, \; \mu = \Sigma Z^T y. \qquad (10)$$

Here $Z = [\xi_1^T, 1; \xi_2^T, 1; \cdots; \xi_N^T, 1]$, $y = [y_1; y_2; \cdots; y_N]$ and $S_4 = \text{diag}([1; 1; 1; 0])$. The additional 1 following ξ_j in Z is for the bias. Then the probability of a given new data point ξ being clear of the obstacle is $\mathbb{P}[n^T \bar{\xi} < 0]$, where $\bar{\xi} = [\xi; 1]$. Now the safety constraint via PrSTL on the probability of collision can be formulated as:

$$\mathbb{P}[n^T \bar{\xi} < 0] \geq 1 - \epsilon \Leftrightarrow \mu^T \bar{\xi} - \Phi^{-1}(\epsilon)\|\Sigma^{1/2}\bar{\xi}\| \leq 0, \quad (11)$$
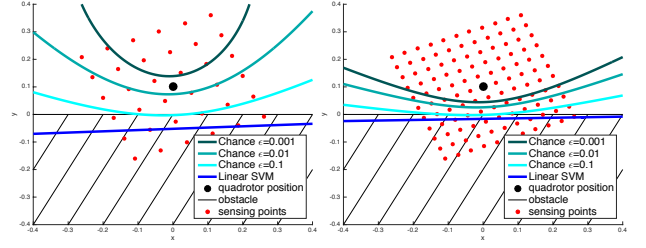

Fig. 1: Chance constraints with different failure probabilities ϵ using GP and a linear inequality constraint using SVM. The feasible regions for the four constraints are the areas above the corresponding lines drawn in the figure. The left figure has 5×5 grid sensing points while the right has 11×11 points.

where Φ^{-1} is the inverse of cumulative distribution function of Gaussian distribution. Note that the above equation is a second-order cone constraint arising due to the safety constraint. These constraints are non-linear in nature and both more reliable and less conservative than a linear inequality constraint. Fig. 1 highlights a simple example comparison between a second-order cone generated from linear GP and a linear boundary estimation from linear SVM. As shown in the figure, linear SVM can't generate reliable constraints, i.e. those constraints have intersection with the obstacles, especially when there are only a few number of sensing points. In contrast, our chance constraints are more reliable and close to the boundary when ϵ is not too small.

While the above sensing scheme (Fig. 1) is based on the assumption that the obstacle is linear, it can be useful even in the case of non-linear obstacles. Note that we reason about obstacles at every time step, consequently we can exploit the local linearity of the object close to the quadrotor. In particular, we propose an efficient two-level sensing method. In the first level, we obtain the nearest point closest to the quadrotor from a sparse but large sensing grid. For computational efficiency in finding the nearest point, the first level can be further decomposed to multiple sub-levels. In the second level, we use a dense but small sensing grid around the nearest point to estimate the boundary. We show the two-level strategy in Fig. 2. The estimated boundary is intended to avoid the obstacles around the nearest point. If the obstacle is convex, then the estimated boundary will also avoid any other parts of the obstacle. If the obstacle is non-convex, there will exist some areas in the obstacles that have intersections with the constrained set. However, since, in the next time step, we will re-estimate the boundary using the new nearest point to the quadrotor, the quadrotor will not easily hit the obstacles.

B. Transformation to the Standard SOCP Form

Assume that the current state of the system is x^0 and the initial control is u^0. Then we are interested in reasoning out about future L states $x^1, x^2, \cdots, x^L$ and the respective controls, $u^1, u^2, \cdots, u^L$ that are safe and will take the robot towards the goal x^*. Formally, the cost function can be
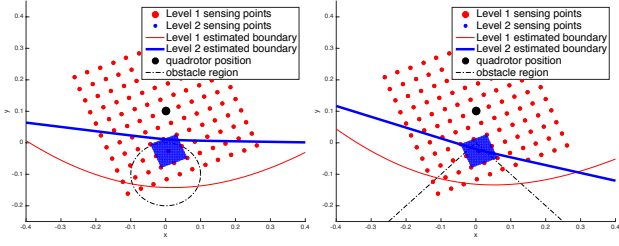
Fig. 2: Two-level sensing method for estimating the nonlinear boundary. To facilitate the illustration, the experiments here consider 2D case. The left and right figures are for a circle obstacle and a triangle obstacle respectively. As shown in the figure, Level 2 sensing points provide a better estimation of the boundary (the failure probability is set as 0.01) than that of Level 1.

written as:

$$F(\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^L) :=$$

$$\sum_{i=1}^{L} \|\boldsymbol{x}_{1:3}^i - \boldsymbol{x}_{1:3}^*\|^2 + \mu_0\|\boldsymbol{x}_{7:9}^i\|^2 + \lambda_0\|\boldsymbol{x}_{4:6}^i\|^2 + \lambda_0\|\boldsymbol{x}_{10:12}^i\|^2 \tag{12}$$

where λ_0, μ_0 are two hyper-parameters, and in our implementation set as $\lambda_0 = 0.2$ and $\mu_0 = 2$. The first term in Eq. (12) provides a potential to approach the final destination, and the following three terms are to regularize the velocities and angles of the quadrotor such that it is stable. Besides the safety cone constraints Eq. (11), the dynamic equations Eq. (9) should be satisfied. However, Eq. (9) is nonlinear, which is difficult to solve, so we use a linear approximation to substitute this constraint. In particular, for $\forall i = 0, 1, 2, \cdots, L-1$

$$\boldsymbol{x}^{i+1} = \boldsymbol{x}^i + A\boldsymbol{x}^i + B\boldsymbol{u}^{i+1} + \boldsymbol{c}, \tag{13}$$

where,

$$A = \left.\frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}^0, \boldsymbol{u}=\boldsymbol{u}^0}, \quad B = \left.\frac{\partial f(\boldsymbol{x}, \boldsymbol{u})}{\partial \boldsymbol{u}}\right|_{\boldsymbol{x}=\boldsymbol{x}^0, \boldsymbol{u}=\boldsymbol{u}^0},$$
$$\boldsymbol{c} = f(\boldsymbol{x}^0, \boldsymbol{u}^0) - A\boldsymbol{x}^0 - B\boldsymbol{u}^0 \tag{14}$$

The optimization problem now can be written as:

$$\min_{\boldsymbol{x}^i, \boldsymbol{u}^i, i \in [L]} F(\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^L)$$
$$\text{s.t.} \quad \boldsymbol{x}^{i+1} = \boldsymbol{x}^i + A\boldsymbol{x}^i + B\boldsymbol{u}^{i+1} + \boldsymbol{c}, \forall i \in [L] - 1$$
$$\boldsymbol{\mu}^T[\boldsymbol{x}_{1:3}^i; 1] - \Phi^{-1}(\epsilon)\|\Sigma^{1/2}[\boldsymbol{x}_{1:3}^i; 1]\| \le 0, \forall i \in [L] \tag{15}$$

Note that $A, B, \boldsymbol{c}, \boldsymbol{\mu}, \Sigma$ only depend on $\boldsymbol{x}^0, \boldsymbol{u}^0$, so they are constant for the problem Eq. (15). Thus, we can simplify Eq. (15). Let $\tilde{\boldsymbol{x}} = [\boldsymbol{x}^1; \boldsymbol{x}^2; \cdots; \boldsymbol{x}^L]$ and $\tilde{\boldsymbol{u}} = [\boldsymbol{u}^1; \boldsymbol{u}^2; \cdots; \boldsymbol{u}^L]$. The first constraint in Eq. (15) can be formulated as follows,

$$A_1\tilde{\boldsymbol{x}} + A_2\tilde{\boldsymbol{u}} + \boldsymbol{b}_1 = 0$$

where $A_2 = I_L \otimes B$ ($\otimes$ is the Kronecker product), $\boldsymbol{b}_1 = [\boldsymbol{x}^0 + f(\boldsymbol{x}^0, \boldsymbol{u}^0) - B\boldsymbol{u}^0; \boldsymbol{c}; \boldsymbol{c}; \cdots; \boldsymbol{c}] \in \mathbb{R}^{12L}$, and

$$A_1 = \begin{bmatrix} -I_{12} & \boldsymbol{0} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ I_{12}+A & -I_{12} & \boldsymbol{0} & \cdots & \boldsymbol{0} \\ \boldsymbol{0} & I_{12}+A & -I_{12} & \cdots & \boldsymbol{0} \\ & \ddots & \ddots & \ddots & \\ \boldsymbol{0} & \cdots & \boldsymbol{0} & I_{12}+A & -I_{12} \end{bmatrix}$$

Therefore, $\tilde{\boldsymbol{x}} = -A_1^{-1}(A_2\tilde{\boldsymbol{u}} + \boldsymbol{b}_1)$. Let $\bar{A} := -A_1^{-1}A_2$ and $\bar{b} := -A_1^{-1}\boldsymbol{b}_1$, then

$$\tilde{\boldsymbol{x}} = \bar{A}\tilde{\boldsymbol{u}} + \bar{b} \tag{16}$$

The objective now can be rewritten as $F(\tilde{\boldsymbol{x}}) = \tilde{\boldsymbol{x}}^T D\tilde{\boldsymbol{x}} + \boldsymbol{q}^T\tilde{\boldsymbol{x}} + \bar{c}$, where $\bar{c}$ is a constant, $D = I_L \otimes \text{diag}([1, 1, 1, \lambda_0, \lambda_0, \lambda_0, \mu_0, \mu_0, \mu_0, \lambda_0, \lambda_0, \lambda_0])$ and $\boldsymbol{q}_{(j-1)*12+1:(j-1)*12+3} = -2\boldsymbol{x}_{1:3}^*$ for $j = 1, 2, \cdots, L$ and 0 for all the other entries. We replace $\tilde{\boldsymbol{x}}$ by $\tilde{\boldsymbol{u}}$ using Eq. (16). Thus, the objective function w.r.t. $\tilde{\boldsymbol{u}}$ is $\tilde{F}(\tilde{\boldsymbol{u}}) = \tilde{\boldsymbol{u}}^T\bar{A}^T D\bar{A}\tilde{\boldsymbol{u}} + (2\bar{A}^T D\bar{b} + \bar{A}^T\boldsymbol{q})^T\tilde{\boldsymbol{u}}$. Let $V = (\bar{A}^T D\bar{A})^{1/2}$ and assume it is non-singular. Set $\hat{\boldsymbol{u}} = V\tilde{\boldsymbol{u}}$ and $\hat{\boldsymbol{p}} = V^{-1}(2\bar{A}^T D\bar{b} + \bar{A}^T\boldsymbol{q})$. Then the objective w.r.t. $\hat{\boldsymbol{u}}$ is

$$\hat{F}(\hat{\boldsymbol{u}}) = \|\hat{\boldsymbol{u}}\|^2 + \hat{\boldsymbol{p}}^T\hat{\boldsymbol{u}}$$

Now we consider the second constraint in Eq. (15).

Let $\Sigma^{1/2} = [H, \boldsymbol{h}]$, where $H \in \mathbb{R}^{4\times3}$, $\boldsymbol{h} \in \mathbb{R}^4$. Let $\bar{\boldsymbol{\mu}} = \boldsymbol{\mu}_{1:3}/\Phi^{-1}(\epsilon)$ and $\bar{\mu}_4 = \boldsymbol{\mu}_4/\Phi^{-1}(\epsilon)$. The second constraint can be written as

$$\|H\boldsymbol{x}_{1:3}^i + \boldsymbol{h}\| \le \bar{\boldsymbol{\mu}}^T\boldsymbol{x}_{1:3}^i + \mu_4$$

Replacing $\boldsymbol{x}_{1:3}^i$ by $\hat{\boldsymbol{u}}$, we have

$$\|B_i\hat{\boldsymbol{u}} + \boldsymbol{b}_i\| \le \boldsymbol{c}_i^T\hat{\boldsymbol{u}} + d_i$$

where

$$B_i = H\bar{A}_{12(i-1)+1:12(i-1)+3,:}V^{-1}$$
$$\boldsymbol{b}_i = H\bar{b}_{12(i-1)+1:12(i-1)+3} + \boldsymbol{h}$$
$$\boldsymbol{c}_i = V^{-T}\bar{A}_{12(i-1)+1:12(i-1)+3,:}^T\bar{\boldsymbol{\mu}}$$
$$d_i = \bar{\boldsymbol{\mu}}^T\bar{b}_{12(i-1)+1:12(i-1)+3} + \bar{\mu}_4$$

Now we have eliminated $\boldsymbol{x}^i$, and formed the new problem w.r.t. $\hat{\boldsymbol{u}} \in \mathbb{R}^{4L}$, which is exactly Eq. (1). Once we obtain the solution $\boldsymbol{z}^*$ of the dual problem Eq. (2), we can obtain the controls by $\tilde{\boldsymbol{u}}^* = -V^{-1}(\hat{\boldsymbol{p}}/2 + U\boldsymbol{z}^*)$ and the states in the horizon by Eq.(16). However, we don't use the predicted next position $\boldsymbol{x}_{1:3}^{*1}$ in the horizon, instead we calculate the next position $\boldsymbol{x}_{1:3}^1$ using the original dynamic system along with $\boldsymbol{u}^{*1}$ (see (9)). We present the complete procedure in Alg. 2. The transform2Dual function transforms the problem (15) to the dual form (2) (see Sec. IV-B and Sec. II for more details).

V. EXPERIMENTS

In this section, we present experimental results to demonstrate significant speedup of the proposed control algorithm over existing off-the-shelf methods while still successfully avoiding unknown obstacles. First we show that our proposed algorithm is much faster than other existing methods for

Algorithm 2 Complete Procedure for Quadrotor Flying

Input: Final destination $\boldsymbol{x}^*$, failure probability of obstacle detection ϵ, stopping distance δ_d, horizon length L, estimation of maximum λ, λ_{max}, maximum SOCP iterations T, the stopping precision of SOCP δ_0.
1: Initialize $\boldsymbol{x}^{(0)} = \boldsymbol{0}$, $\boldsymbol{u}^{(0)} = \boldsymbol{0}$, $t = 0$, $\boldsymbol{z}^t = \boldsymbol{0}$.
2: **while** $\|\boldsymbol{x}_{1:3}^{(t)} - \boldsymbol{x}_{1:3}^*\| \geq \delta_d$ **do**
3:    $[Z, \boldsymbol{y}] = \text{sense}(\boldsymbol{x}^{(t)})$ using two-level mesh grids.
4:    $[\Sigma^{(t)}, \boldsymbol{\mu}^{(t)}] = \text{LinearGP}(Z, \boldsymbol{y}, \epsilon)$ by Eq. (10).
5:    $[U, \boldsymbol{p}, V, \hat{\boldsymbol{p}}] = \text{transform2Dual}(\boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)}, \Sigma^{(t)}, \boldsymbol{\mu}^{(t)})$.
6:    $[\text{Fail}, \boldsymbol{z}^{t+1}] = \text{DualSOCP}(U, \boldsymbol{p}, L, \lambda_{max}, T, \delta_0, \boldsymbol{z}^t)$.
7:    **if** Fail **then**
8:       Print "infeasible or T is too small" and exit.
9:    **end if**
10:    $\tilde{\boldsymbol{u}}^{t+1} = -V^{-1}(\hat{\boldsymbol{p}}/2 + U\boldsymbol{z}^{t+1})$
11:    $\boldsymbol{u}^{(t+1)} = \tilde{\boldsymbol{u}}_{1:4}^{t+1}$
12:    $\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} + f(\boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t+1)})dt$
13:    $t = t + 1$
14: **end while**

general second-order cone problems. Then, we show that when applied to quadrotor problem, our algorithm again provides significant speedup over existing methods. In particular, we compare our Wolfe's algorithm with interior point method (SDPT3), projected gradient descent (PGD), cutting plane methods (CPM) and Fast Alternating Minimization Algorithm (FAMA) [23], [24]. We apply all the algorithms on the dual problem Eq. (2). Therefore, the projection step of PGD has a closed form and we use Amijo rule to do line search with fine-tuned initial step size. CPM is based on the analytic center cutting plane method as described in [25]. More details for CPM can be found in Appendix. All the experiments are implemented in Matlab on a laptop with 2.9 GHz Intel Core i7 and MAC OS. SDPT3 is implemented in C, is in general highly optimized, and is industry standard.

A. General SOCP on synthetic data

In this section, we consider a general SOCP, Eq. (1). We set $B_i \in \mathbb{R}^{n \times n}$ as an element-wise Gaussian matrix, $\boldsymbol{c}_i$ and $\hat{\boldsymbol{p}}$ as element-wise Gaussian vectors, and set $d_i = 10$ and $\boldsymbol{b}_i = \boldsymbol{0}$ for $i \in [L]$. As shown in Fig. 3, our Wolfe's algorithm is much faster than SDPT3 and PGD for different n and L when achieving the same precision as SDPT3.

B. Quadrotor Flying with a Ceiling

We simulate the quadrotor flying from the original point $[0, 0, 0]$ to a destination $[1, 1, 0]$. We consider the similar environment in [7] for quadrotors, i.e., we have a ceiling on the top of the starting point and the ending point. The goal is to achieve the ending point as fast as possible while avoiding hitting the ceiling. Different heights of the ceiling will lead to different trajectories. We observed the following for different heights of the ceiling.

(a) When the ceiling is very high, such as 0.35 as set in [7], the quadrotor actually flies like in the environment without the ceiling, which means the constraints in the
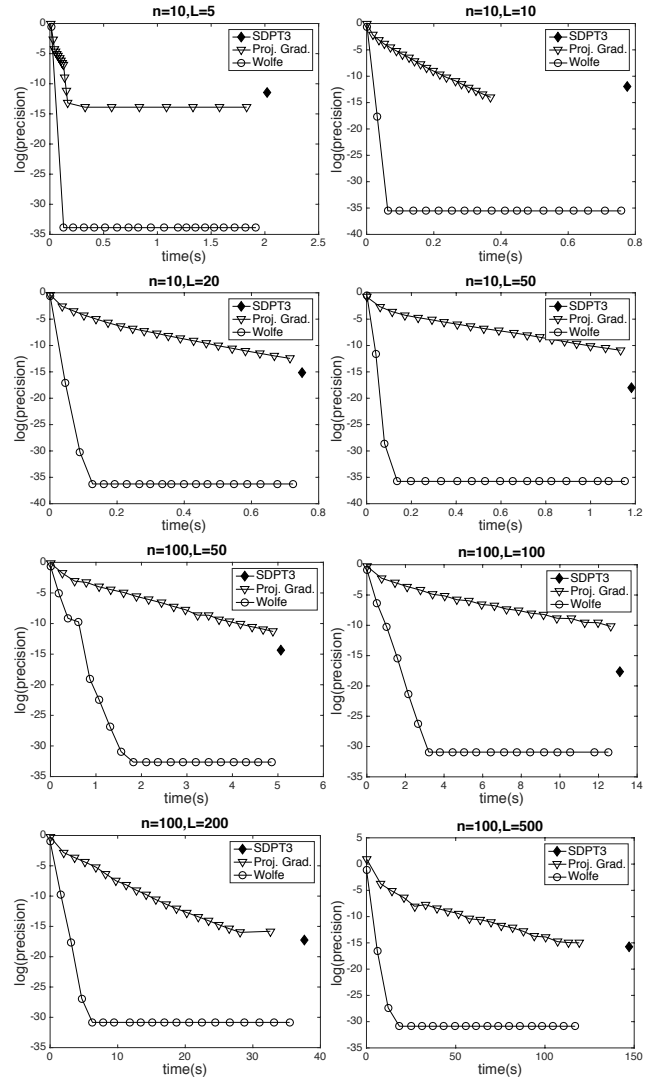

Fig. 3: Comparison among SDPT3, Wolfe's and Projected Gradient Descent on solving a general dual problem Eq. (2) on synthetic random data. n: no. of variables, L: no. of constraints in SOCP.

primal problem are not binding in the optimum. Hence the dual problem has a trivial solution $\boldsymbol{z}^* = \boldsymbol{0}$. As a result, Wolfe's algorithm and PGD only need one iteration if initialized as zero vectors, thus are much faster than other methods.

(b) When the ceiling is very low, such as 0.01, the generated SOCP problem will easily have infeasible constraints.

(c) When the ceiling is not too high or too low, such as 0.08, the quadrotor can finally achieve the destination but the trajectory is different from the case without obstacles.

We illustrate the trajectories when the heights of the ceiling are 0.35 and 0.08 in Fig. 4. As we can see in the figure, the trajectory is affected by a low ceiling of height 0.08 compared to a ceiling with height of 0.35.

Now we compare Wolfe's, SDPT3, PGD, FAMA and CPM applied on specific SOCP problems generated from some
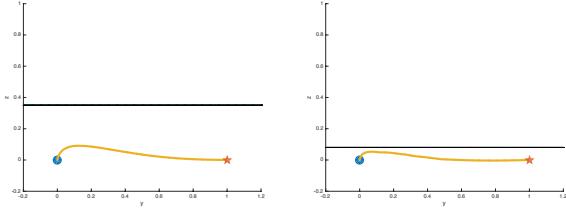
Fig. 4: Trajectories (yellow lines) when the height of the ceiling are 0.35 (left) and 0.08 (right). The experiments are conducted in 3D space, and the viewpoint of the figures is in the y direction. The circle marker is the starting point and the star marker is the ending point. The black lines denote the ceiling.

TABLE I: Comparing different methods on solving an easy subproblem Eq. (2) in terms of time (sec.) for the quadrator problem.

precision	WOLFE	PGD	FAMA	CPM	SDPT3
1.00E-01	0.0005	0.0201	0.0025	4.8245	0.6006
1.00E-02	0.0007	0.0186	0.0027	9.0347	0.6012
1.00E-03	0.0010	0.0268	0.0027	20.4166	0.5953
1.00E-04	0.0011	0.0203	0.0026	25.0489	0.6430
1.00E-05	0.0018	0.0206	0.0034	45.9270	0.6628
1.00E-06	0.0197	0.0294	0.0145	60.0520	0.6660

intermediate steps during the quadrotor navigation. In the sequence of SOCP problems along the navigation, some SOCPs are easy, while others are difficult. So we consider both cases.

(a) *Easy Case:* We define a problem (2) as an easy case when its solution is simply a zero vector. So Wolfe's algorithm or PGD only need one iteration to terminate if initialized as zero vectors. According to our observation, the SOCP will be an easy case if the quadrotor is far away from obstacles.

(b) *Hard Case:* The hard case can be considered as cases when the solution is non-zero. So typically in hard cases, the Wolfe's algorithm or PGD require many iterations to find a solution. When the ceiling is not too high or not too low, we will find such cases in some intermediate steps when the quadrotor is close to the obstacles.

The comparison results are shown in Table I and II. The easy case and the hard case are represented respectively by the first step and the 8-th step of the flight trajectory under a ceiling of height 0.08. We also checked the optimization time using the code provided by [7]. [7] uses GUROBI to solve SOCP. We instead use SDPT3 (via YALMIP) due to the lack of GUROBI license. The optimization time of [7] turns out to be 3-6 seconds for each SOCP problem. It is slower than our time for SDPT3 as shown in Table I and II, which is probably because we are solving a highly simplified dual problem, while [7] directly optimizes over the original problem Eq. (15).

Next, we answer the question: *How much precision is sufficient for the quadrotor problem, using Wolfe's algorithm?*

TABLE II: Comparing different methods on solving a hard subproblem Eq. (2) in terms of time (sec.) for the quadrator problem.

precision	WOLFE	PGD	FAMA	CPM	SDPT3
1.00E-01	0.0006	0.0032	0.0040	3.4142	0.6388
1.00E-02	0.0022	0.0137	0.0080	7.1268	0.6309
1.00E-03	0.0031	0.0920	0.0151	14.7122	0.6190
1.00E-04	0.0085	0.9445	0.1017	30.0934	0.6552
1.00E-05	0.0095	1.3614	0.1834	60.1621	1.1803
1.00E-06	0.0342	1.9909	0.5534	60.1854	0.7947

TABLE III: Timing and number of steps of the complete procedure from the starting point $[0, 0, 0]$ to a neighborhood of radius 0.01 around the destination $[1, 1, 0]$ when SOCPs are solved to different precisions under different ceiling heights. We also include the sensing time and the optimization time, which are main components of total time.

Ceiling	Prec.	Total(s)	Sense(s)	Opt.(s)	Steps
0.08	1.0E-06	4.53	2.01	2.32	128
	1.0E-04	4.41	2.11	2.08	129
	1.0E-02	3.57	2.03	1.33	128
	1.0E00	3.53	2.10	1.29	128
0.35	1.0E-06	2.85	0.92	1.75	127
	1.0E-04	2.33	0.90	1.25	127
	1.0E-02	2.40	0.92	1.26	127
	1.0E00	2.58	0.99	1.38	127

We show the total computation time for the complete flight when SOCP subproblems are solved to different precisions. We consider the case of ceilings with different heights. We set the number of future steps in receding horizon as $L = 20$, the discrete time $dt = 0.03$, the probability of failure $\epsilon = 0.01$ for the chance constraint, and $\lambda_{max} = 10^4$. For the sensing part, we use two-level sensing. The second level for estimating the boundary has a mesh grid of $-0.04 : 0.01 : 0.04$ in each dimension. The first level used to find the nearest point to the quadrotor is further decomposed into two sub-levels, which have mesh grids $-0.3 : 0.06 : 0.3$ and $-0.06 : 0.02 : 0.06$ in each dimension respectively. Detailed timing results can be found in Table III.

Table III shows that for higher ceilings (0.35), the running time of the algorithm does not change with more precise solutions. This is because when the ceiling is high, the SOCP problems are easy in most steps and hence one iteration is enough to achieve the optimal solution. In contrast, when the ceiling is low (0.08), the total computation time increases as the precision increases. Moreover, the quadrotor achieves the final destination successfully for all the precisions, i.e., we don't need to solve SOCPs in each step up to very high precision. This indicates that one can use fast and cheap first order methods (like our proposed approach) rather than more computationally expensive interior points methods.

C. Quadrotor Flying with a Half Wall, a Hill, or a Cylinder

In this subsection, we illustrate the simulation results for several other types of obstacles, including a half wall, a hill
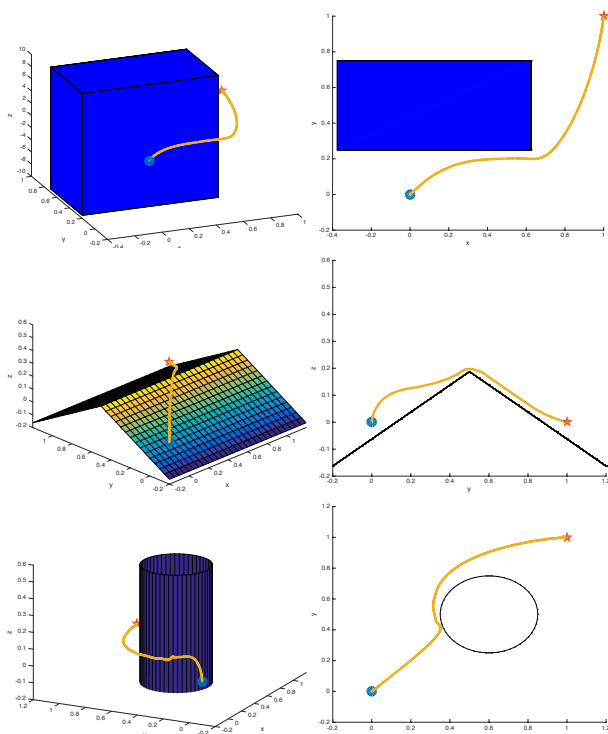
Fig. 5: Trajectories (yellow lines) when avoiding different types of obstacles. The viewpoint of the right picture for the half wall and the cylinder is in the z direction, while the viewpoint of the right picture of the hill is in the x direction. The circle marker is the starting point and the star marker is the ending point.

and a cylinder. The trajectories and the obstacles are shown in Fig. 5. The parameter settings are the same as those for Table III, which is described previously. SOCPs here are all solved to a precision 0.01. All the three cases in Fig. 5 take about 5-6 seconds totally for computation and 150-200 steps for the complete trajectories.

VI. Conclusion

In this paper, we proposed an efficient algorithm for the second-order cone programming that arises in safe controller synthesis for robots that operate in an uncertain environment. Our algorithm enables on-board computation, reduces the latency of planning, saves battery power, and can be easily implemented on embedded chipset on the robot without any external library dependencies. We apply our algorithm to the safe path planning of quadrotors. We also designed a two-level sensing method that efficiently estimates the boundary of the obstacles from which we form the chance constraints for the SOCP. The experiments highlight that the proposed method is much more efficient than the traditional methods and can be used to implement controller on the quadrotor itself rather than an external controller.

References

[1] P. Wolfe, "Finding the nearest point in a polytope," *Mathematical Programming*, vol. 11, no. 1, pp. 128–149, 1976.

[2] A. Prékopa, *Stochastic programming*. Springer Science & Business Media, 2013, vol. 324.

[3] L. Blackmore, M. Ono, and B. C. Williams, "Chance-constrained optimal path planning with obstacles," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1080–1094, 2011.

[4] L. Blackmore and M. Ono, *Convex chance constrained predictive control without sampling*. Proceedings of the AIAA Guidance, Navigation and Control Conference, 2009.

[5] G. C. Calafiore and L. El Ghaoui, "On distributionally robust chance-constrained linear programs," *Journal of Optimization Theory and Applications*, vol. 130, no. 1, pp. 1–22, 2006.

[6] D. Lenz, T. Kessler, and A. Knoll, "Stochastic model predictive controller with chance constraints for comfortable and safe driving behavior of autonomous vehicles," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 292–297.

[7] D. Sadigh and A. Kapoor, "Safe control under uncertainty with probabilistic signal temporal logic," 2016.

[8] D. Dey, D. Sadigh, and A. Kapoor, "Fast safe mission plans for autonomous vehicles," 2016.

[9] I. Gurobi Optimization, "Gurobi optimizer reference manual," 2015. [Online]. Available: http://www.gurobi.com

[10] R. Tütüncü, K. Toh, and M. Todd, "Sdpt3a matlab software package for semidefinite-quadratic-linear programming, version 3.0," 2001.

[11] S. Lacoste-Julien and M. Jaggi, "On the global linear convergence of frank-wolfe optimization variants," in *Advances in Neural Information Processing Systems*, 2015, pp. 496–504.

[12] D. Chakrabarty, P. Jain, and P. Kothari, "Provable submodular minimization using wolfe's algorithm," in *NIPS*, 2014, pp. 802–809.

[13] M. Hehn and R. D'Andrea, "Real-time trajectory generation for interception maneuvers with quadrocopters," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4979–4984.

[14] C. Richter, A. Bry, and N. Roy, "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments," in *Robotics Research*. Springer, 2016, pp. 649–666.

[15] J. Bellingham, A. Richards, and J. P. How, "Receding horizon control of autonomous aerial vehicles," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, vol. 5. IEEE, 2002, pp. 3741–3746.

[16] M. Watterson and V. Kumar, "Safe receding horizon control for aggressive mav flight with limited range sensing," in *IROS*. IEEE, 2015, pp. 3235–3240.

[17] S. Liu, M. Watterson, S. Tang, and V. Kumar, "High speed navigation for quadrotors with limited onboard sensing," in *ICRA*. IEEE, 2016, pp. 1484–1491.

[18] P. Florence, J. Carter, and R. Tedrake, "Integrated perception and control at high speed: Evaluating collision avoidance maneuvers without maps."

[19] M. Pivtoraiko, D. Mellinger, and V. Kumar, "Incremental micro-uav motion replanning for exploring unknown environments," in *ICRA*. IEEE, 2013, pp. 2452–2458.

[20] C. Richter, W. Vega-Brown, and N. Roy, "Bayesian learning for safe high-speed navigation in unknown environments." Proceedings of the International Symposium on Robotics Research, 2015.

[21] L. Matthies, R. Brockers, Y. Kuwata, and S. Weiss, "Stereo vision-based obstacle avoidance for micro air vehicles using disparity space," in *ICRA*. IEEE, 2014, pp. 3242–3249.

[22] H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering," in *ICRA*. IEEE, 2009, pp. 3277–3282.

[23] Y. Pu, M. N. Zeilinger, and C. N. Jones, "Fast alternating minimization algorithm for model predictive control," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 11 980–11 986, 2014.

[24] ——, "Complexity certification of the fast alternating minimization algorithm for linear mpc," *IEEE Transactions on Automatic Control*, 2016.

[25] S. Boyd, "Analytic center cutting-plane method." [Online]. Available: http://web.stanford.edu/class/ee364b/lectures/accpm_slides.pdf

APPENDIX

A. Cutting Plane Method

We follow the analytic center cutting plane method in [25] to solve Eq. (2). We also need an upper bound estimation for all λ_i, i.e., $\lambda_i \leq \lambda_{max}$ as the initialization planes. The constraints $\lambda_i \geq 0$ are used as intial planes, so these constraints will be satisfied all through the process. Therefore we relax the constraint $\|\boldsymbol{v}_i\| \leq \lambda_i$ to $\boldsymbol{v}_i^T \boldsymbol{v}_i - \lambda_i^2 \leq 0$. Since $\lambda_i \geq 0$ are always satisfied, $\boldsymbol{v}_i^T \boldsymbol{v}_i - \lambda_i^2 \leq 0$ are convex sets.

Algorithm 3 Cutting Plane Method for SOCP

Input: Objective function $g(\boldsymbol{z}) = \|U\boldsymbol{z}\|^2 + \boldsymbol{p}^T\boldsymbol{z}$. Maximum λ λ_{max}, Maximum number of iterations T. The stopping precision δ_0. A warm start $\boldsymbol{z}_0$.

Output: Fail, $\boldsymbol{z}^*$

1: Initialize with $2nL + 2L$ cutting planes. The first $2L$ cutting planes are $0 \leq \lambda_i \leq \lambda_{max}$, and the re-maining $2nL$ planes are $\|\boldsymbol{v}_i\|_\infty \leq \lambda_{max}$. Use $\mathcal{P} = \{(\boldsymbol{a}_i, b_i)\}_{i=1,2,\cdots,2nL+2L}$ to denote the plane set, i.e., the feasible set formed by the planes is $\boldsymbol{a}_i^T\boldsymbol{z} + b_i \leq 0$ for $i = 1, 2, \cdots, 2nL + 2L$.

2: Initialize $\boldsymbol{z} = \boldsymbol{z}_0$. Fail=true.

3: **for** $t = 1, 2, \cdots, T$ **do**

4:      Calculate $w_i = \boldsymbol{v}_i^T \boldsymbol{v}_i - \lambda_i^2$, for all $i = 1, \cdots, L$

5:      Let $j = \operatorname{argmax}_i w_i$

6:      **if** $w_j > 0$ **then**

7:          $\boldsymbol{a} \leftarrow \boldsymbol{0}$, $\boldsymbol{a}_{(m+1)j-m:(m+1)j-1} = 2\boldsymbol{v}_j, \boldsymbol{a}_{(m+1)j} = -2\lambda_j$.

8:          $b \leftarrow w_j - \boldsymbol{p}^T\boldsymbol{z}$

9:      **else**

10:         $\boldsymbol{a} \leftarrow 2U^T(U\boldsymbol{z}) + \boldsymbol{p}$

11:         $b \leftarrow -\boldsymbol{a}_k^T\boldsymbol{z}$

12:      **end if**

13:      $\mathcal{P} \leftarrow \mathcal{P} \cup (\boldsymbol{a}, b)$

14:      Use infeasible-start Newton method as described in Slide 8 of [25] to find the new query $\boldsymbol{z}$.

15:      **if** No feasible $\boldsymbol{z}$ is found **then**

16:         Fail=true, return.

17:      **end if**

18:      **if** the sub-optimality $\delta < \delta_0$ **then**

19:         Fail=false, $\boldsymbol{z}^* = \boldsymbol{z}$, return.

20:      **end if**

21:      According to the irrelavance as described in Slide 10-11 of [25], drop some planes (keep at most $5(n+1)L$ planes) and update $\mathcal{P}$.

22: **end for**
