Computing Large Convex Regions of Obstacle-Free Space through Semidefinite Programming

Robin Deits and Russ Tedrake

MIT Computer Science and Artificial Intelligence Laboratory, Cambridge, MA
{rdeits,russt}@csail.mit.edu

Abstract. This paper presents IRIS (Iterative Regional Inflation by Semidefinite programming), a new method for quickly computing large polytopic and ellipsoidal regions of obstacle-free space through a series of convex optimizations. These regions can be used, for example, to efficiently optimize an objective over collision-free positions in space for a robot manipulator. The algorithm alternates between two convex optimizations: (1) a quadratic program that generates a set of hyperplanes to separate a convex region of space from the set of obstacles and (2) a semidefinite program that finds a maximum-volume ellipsoid inside the polytope intersection of the obstacle-free half-spaces defined by those hyperplanes. Both the hyperplanes and the ellipsoid are refined over several iterations to monotonically increase the volume of the inscribed ellipsoid, resulting in a large polytope and ellipsoid of obstacle-free space. Practical applications of the algorithm are presented in 2D and 3D, and extensions to N-dimensional configuration spaces are discussed. Experiments demonstrate that the algorithm has a computation time which is linear in the number of obstacles, and our MATLAB [18] implementation converges in seconds for environments with millions of obstacles.

1 Introduction

This work was originally motivated by the problem of planning footsteps for a bipedal robot on rough terrain. We consider areas where the robot cannot safely step as obstacles, and we plan whole-body walking motions of the robot by optimizing over the space of safe foot positions. Planning around obstacles generally introduces non-convex constraints, which typically can only be solved with weak or probabilistic notions of optimality and completeness. In practice, we want a real-time footstep planner that we can trust to find a locally-good path if it exists.

One approach to combat the non-convexity of the constraints is to divide the obstacle-free region of space into a minimal discrete set of (possibly overlapping) convex regions, but this subdivision is nontrivial. For this work, we assume a configuration space consisting of a bounded region in $\mathbb{R}^n$ which contains polyhedral obstacles. When $n = 2$, we can think of the free space as a polygon with polygonal holes. Even for this simple case, the problem of partitioning the free

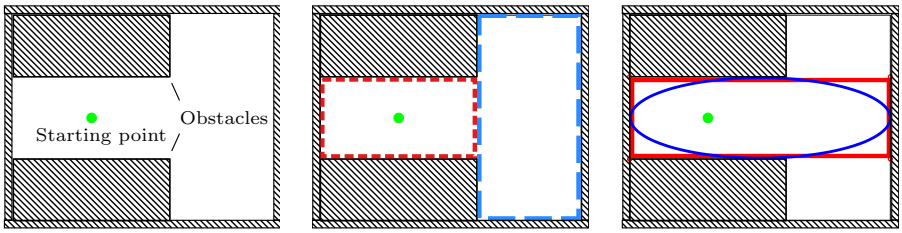

Fig. 1. A simple 2D environment with two rectangular obstacles and a point of interest (left). The minimal non-overlapping convex decomposition of the obstacle-free space produces two polygonal regions (center), while our algorithm produces a larger convex region about the point of interest and an inscribed ellipsoidal region (right).

space into a minimum number of convex parts is NP-hard [13]. Additionally, searching for the minimum number of convex regions may not be the correct problem to solve; we may be willing to give up having a complete cover of the space in order to reduce the number of convex pieces.

In our bipedal robot application, we expect that a human operator or a higher-level planning algorithm can provide helpful guidance about the general area into which the robot should step. If, for example, the operator were to select one or more seed points in space, indicating possible areas into which the robot could step, we would like to find large, convex, obstacle-free regions near those selected points in space so that we can perform an efficient convex optimization of the precise step locations.

A concrete example may be helpful here. Figure 1(a) shows a simple rectangular region with two rectangular obstacles. The obstacle-free region can be minimally decomposed into two non-overlapping convex regions, as shown in 1(b). However, running our algorithm once using the green point as a seed results in a single larger region around the point of interest while maintaining convexity, as shown in 1(c). Additional runs of the algorithm, seeded from the remaining obstacle-free space, could fill the remaining space if desired. Figure 2 shows the same approach applied to real terrain map data captured from an Atlas humanoid robot, using the software developed by Team MIT for the DARPA Robotics Challenge [5].

Our approach, as described in Sect. 3, begins with an initial guess, defined as a point in $\mathbb{R}^n$. We construct an initial ellipsoid, consisting of a unit ball centered on the selected point. We then iterate through the obstacles, for each obstacle generating a hyperplane which is tangent to the obstacle and separates it from the ellipsoid. These hyperplanes then define a set of linear constraints, whose intersection is a polytope. We can then find a maximal ellipsoid in that polytope, then use this ellipsoid to define a new set of separating hyperplanes, and thus a new polytope. We choose our method of generating the separating hyperplanes so that the ellipsoid volume will never decrease between iterations. We can repeat this procedure until the ellipsoid's rate of growth falls below some threshold, at

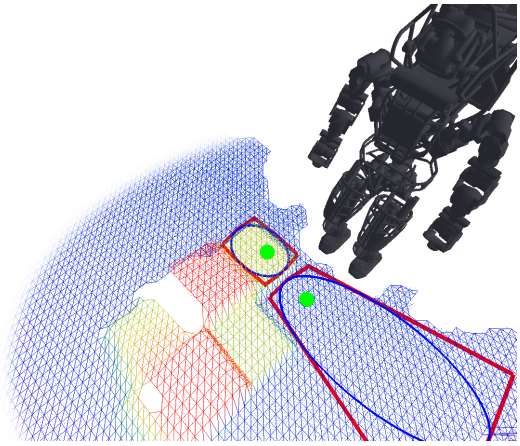

Fig. 2. A visualization of an Atlas humanoid standing in front of a set of tilted steps, as seen in the DARPA Robotics Challenge 2014 trials [5], with two convex regions of safe terrain displayed (blue ellipses and red polytopes). The green circles indicate two points chosen by a human operator for possible locations of the next footstep. To compute the safe regions, we construct a grid of height values from LIDAR scans, check the steepness of the terrain at every point on the grid, and convert any cells with steepness above a threshold into obstacles. We then run the IRIS algorithm with these obstacles starting from the user-selected points.

which point we return the polytope and inscribed ellipsoid. Examples of this procedure in 2D and 3D can be seen in Figs. 3 and 4, respectively.

The IRIS algorithm presented here assumes that the obstacles themselves are convex, which is an important limitation. However, existing algorithms for approximate or exact convex decomposition can be easily used to segment the obstacles into convex pieces before running our algorithm [12,17], and the favorable performance of our algorithm for large numbers of obstacles means that the decomposition of the obstacles need not be minimal. It is also important to note that the algorithm as written here does not guarantee that the initial point in space provided by the user will be contained in the final ellipsoid or polytope. In the experiments presented in Fig. 5, the point was contained in the final hull 95% of the time. If this condition is required by the application, then the algorithm can be terminated early should the region found ever cease to include the start point.

In the remainder of this paper, we discuss the precise formulation of the algorithm and its relationship to existing approaches. We demonstrate the algorithm in 2D and 3D cases and discuss its application in N-dimensional configuration spaces. Finally, we show that the algorithm is practical for extremely cluttered environments, demonstrating that we can compute a convex region in an environment containing one million obstacles in just a few seconds, as shown in Fig. 5.

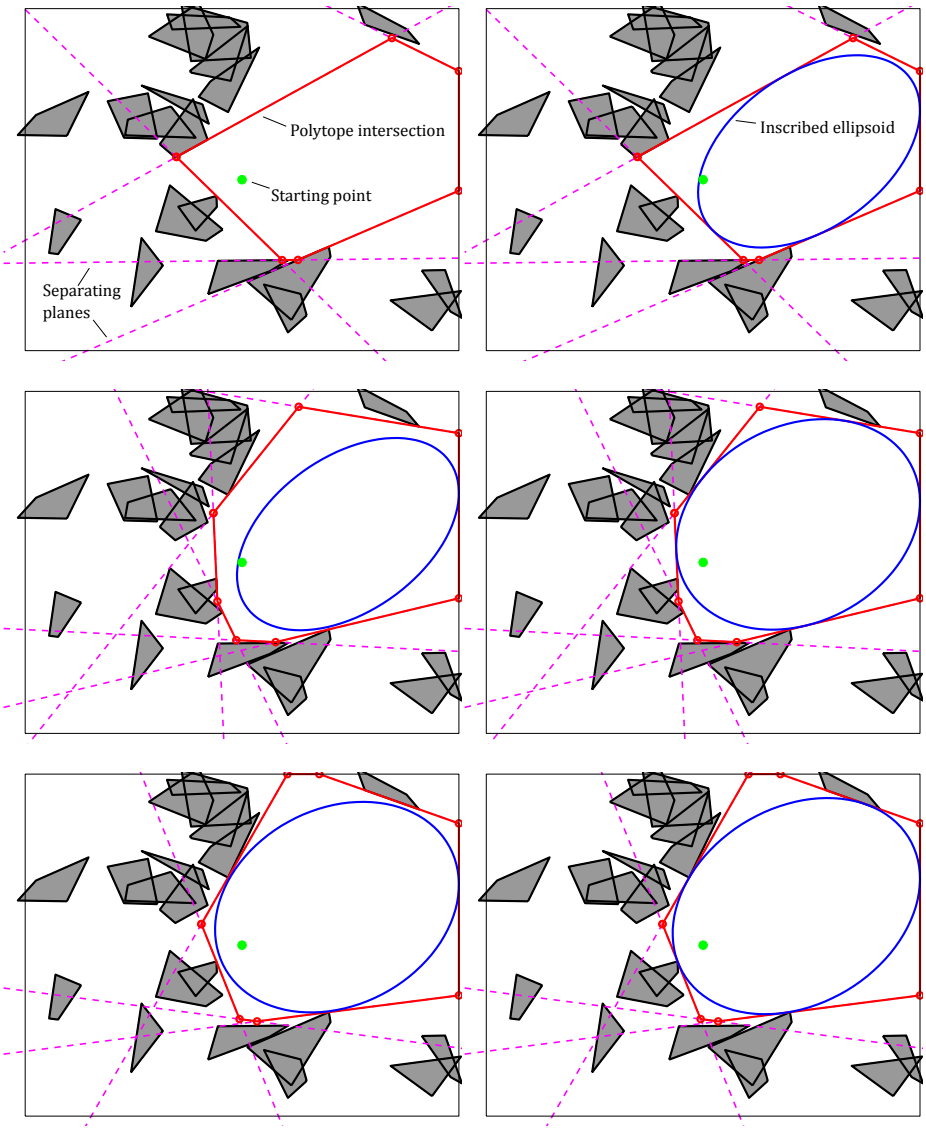

Fig. 3. A demonstration of the IRIS algorithm in a planar environment consisting of 20 uniformly randomly placed convex obstacles and a square boundary. Each row above shows one complete iteration of the algorithm: on the left, the hyperplanes are generated, and their polytope intersection is computed. On the right, the ellipse is inflated inside the polytope. After three iterations, the ellipse has ceased to grow, and the algorithm has converged.

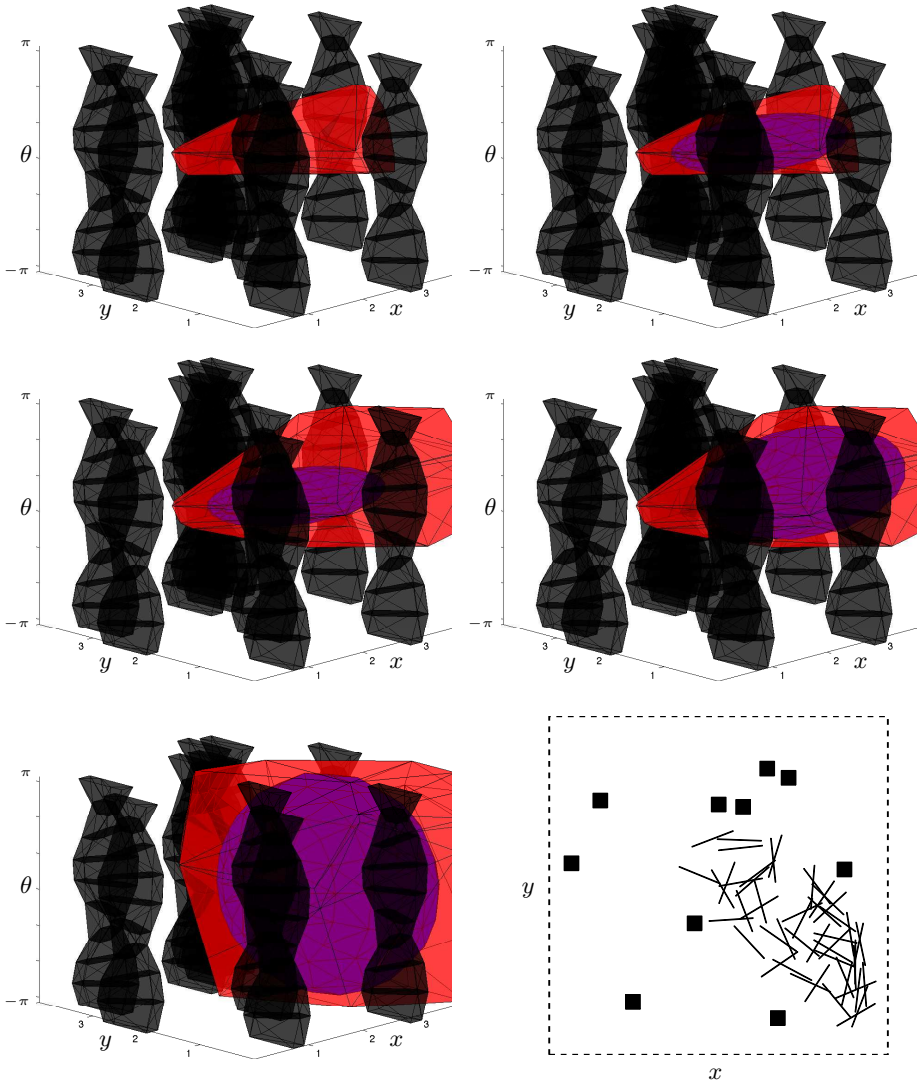

Fig. 4. An example of generating a large convex region in configuration space. A 2D environment containing 10 square obstacles was generated, and the configuration space obstacles for a rod-shaped robot in that environment were built by dividing the orientations of the robot into 10 bins and constructing a convex body for each range of orientations [15]. The top two rows show the first two iterations of the algorithm, generating the separating planes on the left and generating the ellipsoid on the right. The obstacles are shown in black, the polyhedral intersection of the hyperplanes in red, and the ellipsoid in purple. At the bottom left are the final ellipsoid and polytope after convergence, and at the bottom right is the original 2D environment with 50 configurations of the robot uniformly sampled from the obstacle-free polytope.

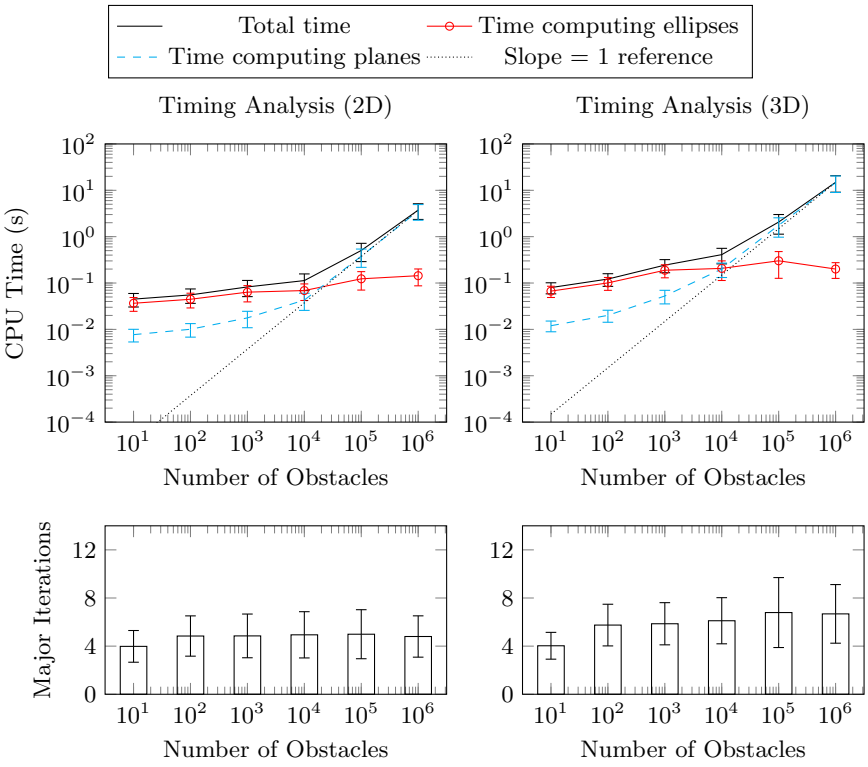

Fig. 5. Timing results of 1200 runs of the IRIS algorithm implemented in MATLAB on an Intel i7 processor at 2.5 GHz with 8 Gb of RAM. In each of the 2D and 3D cases, we generated 100 environments at 6 logarithmically spaced numbers of obstacles between 10^1 and 10^6. Obstacles were uniformly randomly placed in each environment. Total time required to converge to a single convex region is shown above, along with the breakdown of time spent computing the separating hyperplanes and time spent finding the maximal ellipsoid. These plots demonstrate the empirically linear scaling of computation time with number of obstacles: time spent computing planes increases linearly with obstacle count, approaching a slope of 1 on this log-log plot, while time spent finding the ellipsoid is nearly constant. Below, we show the number of iterations of the algorithm (each iteration consists of finding the entire set of hyperplanes and the maximal ellipsoid) before convergence to a relative change in ellipsoid volume of less than 2%. Error bars are all one standard deviation.

2 Related Work

There are a variety of algorithms for approximate or approximately minimal convex decompositions, most of which focus on creating a convex or nearly convex cover of some space. Lien proposes an algorithm for segmenting non-convex polygons containing polygonal holes into a small number of pieces, each of which is al-

lowed some small degree of concavity [12]. Similarly, Mamou's approach converts a triangulated 3D mesh into a set of approximately convex pieces by iteratively clustering faces of the mesh together according to heuristics based on convexity and aspect ratio [17]. Liu's approach [14], on the other hand, is applicable in spaces of arbitrary dimension and relies on an integer linear programming formulation to compute a set of cuts which divide the obstacle into approximately convex pieces. These approaches are not well suited to convex optimization over obstacle-free space: we require convex regions, and taking the convex hull of the approximately convex pieces may result in regions which intersect the obstacle set.

There also exist polynomial-time approximation algorithms for approximately minimal convex covers. Eidenbenz describes an algorithm which computes a nearly-minimal set of overlapping convex pieces for a polygon with holes [4]. Their method achieves a number of pieces within an error bound which is logarithmic in the number of vertices, but it requires running time of $O(n^{29} \log n)$, where n is the number of vertices in the polygon. Feng also describes an approach that divides an input polygon with holes into pieces, which can be convex if desired, and generates a tree structure of adjacent pieces [6]. This is a promising approach, but their algorithm as presented is not applicable beyond the 2D case.

Convex decompositions which do not attempt to find the minimum number of segments have also been used: Demyen's approach involves triangulating the entire free space by connecting all mutually visible vertices on the obstacles, then performing path search among the triangulated regions [3]. Finally, Sarmiento produces convex polytopic regions in N dimensions by sampling points in free space and checking visibility from a set of "guard" positions [22]. This work produces results which appear to be the most similar to ours, but requires as input a set of samples which cover the workspace. Instead, we focus on creating a single, large, convex region in some local area, allowing later optimizations to be run inside this region without further consideration given to the positions of obstacles.

Fischer solves a similar problem of finding a single maximal convex polygon in a discrete environment [7] in polynomial time. His problem formulation consists of a set of points which are labeled as positive or negative, with the goal being to find a convex polygon of maximal area which has vertices only on positive points and which contains no negative points on its boundary or interior. This is a restricted form of our task, but it is one which can be solved to optimality with effort which is polynomial in the number of points in the set.

The problem of finding obstacle-free regions is also relevant in structural biology, in which a user might wish to find the void volumes enclosed by a molecular structure represented as a collection of solid spheres. For example, Sastry performs a search over the vertices of the Voronoi cells containing the spherical molecules to find the connected cavities, but these cavities are not necessarily convex [23]. Luchnikov extends this notion of searching for (non-convex) voids over the Vornoi network to non-spherical objects [16].

3 Technical Approach

3.1 Proposed Algorithm

Our algorithm searches for both an ellipsoid and a set of hyperplanes which separate it from the obstacles. We choose to represent the ellipsoid as an image of the unit ball: $\mathcal{E}(C,d) = \{x = C\tilde{x} + d \mid \|\tilde{x}\| \leq 1\}$ and we represent the set of hyperplanes as linear constraints: $P = \{x \mid Ax \leq b\}$. We have chosen this definition of the ellipsoid because it makes maximization of the ellipsoid volume straightforward: volume of the ellipsoid is proportional to the log of the determinant of C, which is a concave function of C [2] and can therefore be efficiently maximized. In searching both for the ellipsoidal region and the hyperplanes which separate it from the obstacles, we are attempting to solve the following nonconvex optimization problem:

$$\begin{aligned} \underset{A,b,C,d}{\text{maximize}} \quad & \log\det C \\ \text{subject to} \quad & a_j^\top v_k \geq b_j \quad \text{for all points } v_k \in \wr_j, \text{ for } j = 1,\ldots,N \\ & \sup_{\|\tilde{x}\|\leq 1} a_i^\top (C\tilde{x} + d) \leq b_i \quad \forall i = [1,\ldots,N] \end{aligned} \tag{1}$$

where a_j are the rows of A, b_j are the elements of b, $\wr_j$ is the set of points in the convex obstacle j, and N is the number of obstacles. The constraint that $a_j^\top v_k \geq b_j$ for all points $v_k \in \wr_j$ forces all of the points in obstacle $\wr_j$ to lie on one side of the plane defined by $a_j^\top x = b_j$. The second constraint ensures that all $x = C\tilde{x} + d$ where $\|\tilde{x}\| \leq 1$ fall on the other side of that plane. Satisfying these constraints for every obstacle j ensures that the ellipsoid is completely separated from the obstacles. Rather than solving this directly, we will alternate between searching for the planes defining the linear constraints a_j and b_j and searching for the maximal ellipsoid which satisfies those constraints. The general outline of the IRIS procedure is given in Algorithm 1.

Algorithm 1 Given an initial point q_0 and list of obstacles $\mathcal{O}$, find an obstacle-free polytopic region P defined by $Ax \leq b$ and inscribed ellipsoid $\mathcal{E} = \{C\tilde{x} + d \mid \|\tilde{x}\| \leq 1\}$ such that $\mathcal{E} \subseteq P$ and P intersects $\mathcal{O}$ only on its boundary. Subroutine SeparatingHyperplanes is expanded in Algorithm 2, and subroutine InscribedEllipsoid is described in Sect. 3.4

$C_0 \leftarrow \epsilon I_{n\times n}$
$d_0 \leftarrow q_0$
$i \leftarrow 0$
repeat
  $(A_{i+1}, b_{i+1}) \leftarrow$ SeparatingHyperplanes$(C_i, d_i, \mathcal{O})$
  $(C_{i+1}, d_{i+1}) \leftarrow$ InscribedEllipsoid(A_{i+1}, b_{i+1})
  $i \leftarrow i + 1$
until $(\det C_i - \det C_{i-1}) / \det C_{i-1} < tolerance$
return (A_i, b_i, C_i, d_i)

3.2 Initializing the Algorithm

The IRIS algorithm begins with an initial point in space, which we will label as q_0. The formal algorithm described here requires q_0 to be in the obstacle-free space, but in practice we can sometimes recover from a seed point which is inside an obstacle by reversing the orientation of one or more of the separating hyperplanes. We initialize the algorithm with an arbitrarily small sphere around q_0 by setting $d_0 \leftarrow q_0$ and $C_0 \leftarrow \epsilon I_{n\times n}$.

3.3 Finding Separating Hyperplanes

We attempt to find separating hyperplanes which will allow for further expansion of the ellipsoid while still ensuring that the interior of the ellipsoid never intersects the interior of any obstacle. Conceptually, the procedure for finding the separating hyperplanes involves finding planes that intersect the boundaries of the obstacles and that are tangent to uniform expansions of the ellipsoid. Given an ellipsoid $\mathcal{E}(C,d) = \{C\tilde{x}+d \mid \|\tilde{x}\| \le 1\}$, we define a uniform expansion of $\mathcal{E}$ as

$$\mathcal{E}_\alpha \equiv \{C\tilde{x}+d \mid \|\tilde{x}\| \le \alpha\} \quad \text{for some } \alpha \ge 1 \tag{2}$$

To find the closest point on an obstacle $\mathcal{l}_j$ to the ellipsoid, we can search over values of α

$$\begin{aligned} \alpha^* = \arg\min_{\alpha} \quad & \alpha \\ \text{subject to} \quad & \mathcal{E}_\alpha \cap \mathcal{l}_j \neq \varnothing \end{aligned} \tag{3}$$

We label the point of intersection between $\mathcal{E}_{\alpha^*}$ and $\mathcal{l}_j$ as x^*. We can then compute a hyperplane, $a_j^\top x = b$, with $a_j \in \mathbb{R}^n$ and $b_j \in \mathbb{R}$ which is tangent to $\mathcal{E}_{\alpha^*}$ and which passes through x^*. This hyperplane separates $\mathcal{E}_{\alpha^*}$ and $\mathcal{l}_j$, and, since $\mathcal{E} \subseteq \mathcal{E}_\alpha$for $\alpha \ge 1$, it also separates $\mathcal{E}$ from $\mathcal{l}_j$. We choose the sign of a_j and b_j such that $a_j^\top x \ge b_j$ for every $x \in \mathcal{l}_j$.

Using this procedure, we can find for every obstacle a plane which separates it from the ellipsoid at every iteration. In practice, we perform several optimizations to allow for efficient computation with very large numbers of obstacles, and we are generally able to avoid computing a new plane for every single obstacle.

Finding the Closest Point to the Ellipse. Rather than actually searching over values of α as in (3), we can instead simplify the problem of finding a separating plane to a single least-distance programming problem, which we can solve very efficiently.

Let $\mathcal{E}(C,d)$ be our ellipsoid and let $v_{j,1}, v_{j,2}, \ldots, v_{j,m}$ be the vertices of the convex obstacle $\mathcal{l}_j$. Our ellipsoid is defined as an image of the unit ball in $\mathbb{R}^n$: $\mathcal{E} = \{C\tilde{x}+d \mid \|\tilde{x}\| \le 1\}$, so we construct the inverse of this image map:

Ellipse Space	Ball Space
$\mathcal{E} = \{C\tilde{x}+d \mid \|\tilde{x}\| \le 1\}$	$\tilde{\mathcal{E}} = \{\tilde{x} \in \mathbb{R}^n \mid \|\tilde{x}\| \le 1\}$
$\mathcal{l}_j = \text{ConvexHull}(v_{j,1}, \ldots, v_{j,m})$	$\tilde{\mathcal{l}}_j = \text{ConvexHull}(\tilde{v}_{j,1}, \ldots, \tilde{v}_{j,m})$
$v_{j,k} = C\tilde{v}_{j,k} + d$	$\tilde{v}_{j,k} = C^{-1}(v_{j,k} - d)$

We now need only to find the closest point to the origin on the transformed obstacle $\tilde{\wr_j}$, then apply the $C\tilde{x} + d$ map once more to find the closest point to the ellipse on $\wr_j$. We can construct the problem of finding this point as:

$$\begin{aligned} \underset{\tilde{x}\in\mathbb{R}^n, w\in\mathbb{R}^m}{\arg\min} \quad & \|\tilde{x}\|^2 \\ \text{subject to} \quad & \left[\tilde{v}_{j,1}\ \tilde{v}_{j,2}\ \ldots\ \tilde{v}_{j,m}\right] w = \tilde{x} \\ & \sum_{i=1}^{m} w_i = 1 \\ & w_i \geq 0 \end{aligned} \tag{4}$$

in which we search for the point $\tilde{x}$ which is a convex combination of the $\tilde{v}_{j,k}$ and which is closest to the origin. As written, this is a quadratic program, but it can be transformed into a least-distance programming instance and solved very efficiently as a least-squares problem with nonnegativity constraints [11]. In our implementation, we achieved the best performance by solving the original quadratic program in (4) using a task-specific solver generated by the CVXGEN tools [19]. The CVXGEN solver is able to compute the closest point for a typical obstacle with 8 vertices in 3 dimensions in under 20 µs on an Intel i7. We have also had success with the standard commercial QP solvers Mosek [21] and Gurobi [9], but both required upwards of 1 ms for similar problems.

This optimization yields a point $\tilde{x}^*$. Applying the original map gives $x^* = C\tilde{x}^* + d$, which is the point on obstacle $\wr_j$ closest to the ellipsoid.

Finding the Tangent Plane. The simplest way to find the tangent plane to the ellipsoid is to consider the inverse representation of $\mathcal{E}$ as

$$\mathcal{E} = \left\{x \mid (x-d)^\top C^{-1} C^{-\top} (x-d) \leq 1\right\} \tag{5}$$

We can find a vector normal to the surface of the ellipse by computing the gradient of the ellipsoid's barrier function at x^*:

$$\begin{aligned} a_j &= \nabla_x \left[(x-d)^\top C^{-1} C^{-\top} (x-d)\right]\big|_{x^*} \\ &= 2C^{-1}C^{-\top}(x^* - d). \end{aligned} \tag{6}$$

Once we have a_j, we can trivially find b_j, since the plane passes through x^*:

$$b_j = a_j^\top x^*. \tag{7}$$

Removing Redundant Planes. In an environment with very many obstacles, most of the separating hyperplanes found using the above procedure turn out to be unnecessary for ensuring that the ellipsoid is obstacle-free. This can be seen in Fig. 3, in which at every iteration just 4 or 5 planes are required to completely separate the ellipse from all 20 obstacles. By eliminating redundant planes, we can dramatically improve the efficiency of the ellipsoid maximization step.

For a given obstacle $\wr_j$ we compute a_j and b_j such that $a_j^\top x \geq b_j$ for all $x \in \wr_j$. We can then search through all other obstacles $\wr_k, k \neq j$ and check whether $a_j^\top v \geq b_j$ also holds for every point $v \in \wr_k$. Since the obstacles are required to be polyhedral, we need only to check the inequality at the vertices of each $\wr_k$. If it holds, then obstacle $\wr_k$ is also separated from $\mathcal{E}$ by the hyperplane in question, so we can skip computing a separating hyperplane for obstacle $\wr_k$. To improve this further, we can start with the obstacle containing the closest vertex to the ellipse, since a hyperplane separating that obstacle from the ellipse will likely also separate many more distant obstacles, and then work outward until all obstacles have been separated from $\mathcal{E}$ by some plane. This procedure is detailed in Algorithm 2.

Algorithm 2 Given matrix C and d defining an ellipse $\mathcal{E}$, as in Algorithm 1, and a set of convex obstacles $\mathcal{O}$, find A and b defining a set of hyperplanes which are tangent to the uniform expansion of $\mathcal{E}$ and with $\{x \in \mathbb{R}^n \mid Ax \leq b\} \cap \mathcal{O} = \varnothing$. Subroutines ClosestObstacle, ClosestPointOnObstacle, and TangentPlane are described in Sect. 3.3

function SeparatingHyperplanes(C, d, $\mathcal{O}$)
  $\mathcal{O}_{excluded} \leftarrow \varnothing$
  $\mathcal{O}_{remaining} \leftarrow \mathcal{O}$
  $i \leftarrow 1$
  while $\mathcal{O}_{remaining} \neq \varnothing$ **do**
    $\wr^* \leftarrow$ ClosestObstacle$(C, d, \mathcal{O}_{remaining})$
    $x^* \leftarrow$ ClosestPointOnObstacle$(C, d, \wr^*)$
    $(a_i, b_i) \leftarrow$ TangentPlane(C, d, x^*)
    for all $\wr_i \in \mathcal{O}_{remaining}$ **do**
      if $a_i^\top x_j \geq b_i \quad \forall x_j \in \wr_i$ **then**
        $\mathcal{O}_{remaining} \leftarrow \mathcal{O}_{remaining} \setminus \wr_i$
        $\mathcal{O}_{excluded} \leftarrow \mathcal{O}_{excluded} \cup \wr_i$
      end if
    end for
    $i \leftarrow i + 1$
  end while
  $A \leftarrow \begin{bmatrix} a_1^\top \\ a_2^\top \\ \vdots \end{bmatrix}, b \leftarrow \begin{bmatrix} b_1 \\ b_2 \\ \vdots \end{bmatrix}$
  return (A, b)
end function

3.4 Computing the Inscribed Ellipsoid

The problem of computing an ellipsoid of maximum volume inscribed in a polytope is well studied, and efficient practical algorithms for solving it can be easily

found. We represent the inscribed ellipsoid as an image of the unit ball:

$$\mathcal{E} = \{C\tilde{x} + d \mid \|\tilde{x}\| \leq 1\} \tag{8}$$

with the volume of the ellipsoid proportional to the determinant of C [2]. The problem of finding the maximum volume ellipse contained in the polytope $P = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ can be expressed as

$$\begin{aligned} \underset{C,d}{\text{maximize}} \quad & \log\det C \\ \text{subject to} \quad & \sup_{\|\tilde{x}\|\leq 1}(a_i^\top C\tilde{x}) + a_i^\top d \leq b_i \quad \forall i = [1,\ldots,N] \\ & C \succeq 0 \end{aligned} \tag{9}$$

as stated by Boyd [2], where the a_i and b_i are the rows and elements, respectively, of A and b and $A \in \mathbb{R}^{N\times n}$. The constraints can be rewritten without mention of $\tilde{x}$, yielding:

$$\begin{aligned} \underset{C,d}{\text{maximize}} \quad & \log\det C \\ \text{subject to} \quad & \left\|a_i^\top C\right\| + a_i^\top \leq b_i \forall i = [1,\ldots,N] \\ & C \succeq 0 \end{aligned} \tag{10}$$

which is a convex optimization [2]. Khachiyan and Todd describe an approximation algorithm to solve this problem through a sequence of convex optimizations with linear constraints with a guaranteed convergence to within a given relative error from the maximum possible ellipsoid volume [10]. Ben-Tal and Nemirovski, meanwhile, present a method for computing the ellipsoid through a semidefinite and conic quadratically constrained optimization [1], and we use this approach, as implemented by Mosek [20], in our code. We have also successfully used CVX, a tool for specifying and solving convex problems [8], to solve (10), but we found that the Mosek implementation was at least an order of magnitude faster, primarily due to the overhead of constructing the problem in CVX.

3.5 Convergence

The IRIS algorithm makes no guarantee of finding the largest possible ellipsoid in the environment, but it still provides some assurance of convergence. Since our separating hyperplanes are, by construction, tangent to an expanded ellipsoid $\mathcal{E}_\alpha$ for some $\alpha \geq 1$, the original ellipsoid $\mathcal{E}$ will always be contained in the feasible set of $Ax \leq b$. Additionally, because the ellipsoid maximization SDP is a convex optimization which is solved to its global maximum, it must be true that the volume of the ellipsoid produced no less than the volume of $\mathcal{E}$. If this were not the case, then $\mathcal{E}$ would be a feasible solution with larger volume, which contradicts global optimality of the SDP. As long as the environment is bounded on all sides, there is an upper limit on the volume of the ellipsoid, corresponding to the whole volume of the environment. Since the ellipsoid volume is bounded above and monotonically increasing, it will converge to a final value, although we do not currently make any claims about how many iterations this will require.

4 Results

We implemented the proposed algorithm in MATLAB [18], using CVXGEN [19] to solve each least-distance QP and Mosek [20] to solve each maximal-ellipsoid SDP. Given a list of convex obstacles, a boundary around the environment, and a starting point, the implemented algorithm rapidly finds a large convex region and its inscribed ellipsoid. A simple 2D example of the results can be seen in Fig. 3. The algorithm is also equally applicable in 3D, or in the 3D representation of the configuration space of a 3-degree of freedom robot. Such an application is shown in Fig. 4, in which a convex region of configuration space for a rod-shaped robot in the plane is found and sampled. The algorithm also extends without modification to higher dimensions. Figure 6 shows a 3D slice of the output of the IRIS procedure in 4 dimensions, and the algorithm can also be run in higher-dimensional configuration spaces, assuming that the N-dimensional configuration space obstacles can be generated.

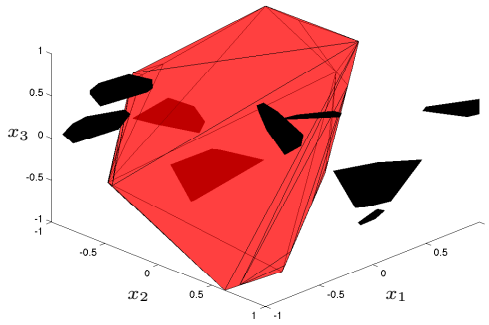

Fig. 6. An example of the output of the algorithm in 4-dimensional space. We generated 4-dimensional obstacles consisting of uniformly random points centered on uniformly randomly chosen locations in $[-1, 1]^4$. The figure shows the 3-dimensional intersection with the $x_4 = 0$ plane of the obstacles and the polytope produced by the IRIS algorithm.

A major advantage of this algorithm is the efficiency with which it can handle extremely cluttered environments. Computing each separating hyperplane requires work which is linear in the number of obstacles, since each obstacle must be checked against the newly found hyperplane to determine if it is also excluded, as in Sect. 3.3. The total number of planes required to exclude all the obstacles, however, turns out to be nearly constant in practice. This means that the entire hyperplane computation step requires nearly linear time in the number of obstacles. Additionally, since each hyperplane found creates one constraint for the ellipsoid maximization step, the constant number of hyperplanes means that the ellipsoid maximization requires approximately constant time as the number of obstacles increases. We demonstrate this by running the algorithm

in 2D and 3D for 10 to 1,000,000 obstacles and displaying the linear increase in computation time in Fig. 5.

5 Conclusion

We have demonstrated a new algorithm for finding large regions of obstacle-free space in a cluttered environment. These regions can be rapidly computed and then used later to aid some future optimization problem, such as the problem of planning robot footstep locations while avoiding obstacles.

Our immediate future plans are to apply this algorithm to footstep planning for a real humanoid robot. We will allow the user to select a point in space on a terrain map, compute an obstacle free region, and find a footstep position which optimizes reachability and stability within that region. We are also interested in exploring other applications of this algorithm to problems beyond footstep planning, in which one or more convex regions are preferable to a large set of non-convex constraints.

6 Source Code and Animations

A development version of the IRIS implementation can be found on GitHub at `https://github.com/rdeits/iris-distro`. It includes all of the algorithms presented in this paper, as well as animations of IRIS running in 2D, 3D, and 4D.

7 Acknowledgements

This work was supported by the Fannie and John Hertz Foundation and by MIT CSAIL. The authors also wish to thank the members of the Robot Locomotion Group at CSAIL for their advice and help.

References

1. Ben-Tal, A., Nemirovski, A.: More examples of CQ-representable functions/sets. In: Lectures on Modern Convex Optimization: Analysis, Algorithms and Engineering Applications, pp. 105–110. MPS-SIAM Series on Optimization, SIAM, Philadelphia, PA (2001)
2. Boyd, S.P., Vandenberghe, L.: Convex optimization. Cambridge University Press, Cambridge, UK; New York (2004)
3. Demyen, D., Buro, M.: Efficient triangulation-based pathfinding. AAAI 6, 942–947 (2006), `http://www.aaai.org/Papers/AAAI/2006/AAAI06-148.pdf`
4. Eidenbenz, S.J., Widmayer, P.: An approximation algorithm for minimum convex cover with logarithmic performance guarantee. SIAM Journal on Computing 32(3), 654–670 (2003), `http://epubs.siam.org/doi/abs/10.1137/S0097539702405139`

5. Fallon, M., Kuindersma, S., Karumanchi, S., Antone, M., Schneider, T., Dai, H., Perez D'Arpino, C., Deits, R., DiCicco, M., Fourie, D., Koolen, T., Marion, P., Posa, M., Valenzuela, A., Yu, K.T., Shah, J., Iagnemma, K., Tedrake, R., Teller, S.: An architecture for online affordance-based perception and whole-body planning. Submitted to: Journal of Field Robotics (2014), http://dspace.mit.edu/handle/1721.1/85690
6. Feng, H.Y.F., Pavlidis, T.: Decomposition of polygons into simpler components: Feature generation for syntactic pattern recognition. Computers, IEEE Transactions on 100(6), 636–650 (1975), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1672869
7. Fischer, P.: Finding maximum convex polygons. In: sik, Z. (ed.) Fundamentals of Computation Theory, pp. 234–243. No. 710 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (1993), http://link.springer.com/chapter/10.1007/3-540-57163-9_19
8. Grant, M., Boyd, S.: CVX: Matlab software for disciplined convex programming, version 2.1 (2014), http://cvxr.com/cvx
9. Gurobi Optimization, Inc.: Gurobi optimizer reference manual (2014), http://www.gurobi.com/
10. Khachiyan, L.G., Todd, M.J.: On the complexity of approximating the maximal inscribed ellipsoid for a polytope. Mathematical Programming 61(1-3), 137–159 (1993), http://link.springer.com/article/10.1007/BF01582144
11. Lawson, C.L., Hanson, R.J.: Solving Least Squares Problems. SIAM (1995)
12. Lien, J.M., Amato, N.M.: Approximate convex decomposition of polygons. Proceedings of the twentieth annual symposium on Computational geometry pp. 17–26 (2004), http://dl.acm.org/citation.cfm?id=997823
13. Lingas, A.: The power of non-rectilinear holes. In: Nielsen, M., Schmidt, E.M. (eds.) Automata, Languages and Programming, pp. 369–383. No. 140 in Lecture Notes in Computer Science, Springer Berlin Heidelberg (1982), http://link.springer.com/chapter/10.1007/BFb0012784
14. Liu, H., Liu, W., Latecki, L.: Convex shape decomposition. In: 2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 97–104 (2010)
15. Lozano-Perez, T.: Spatial planning: A configuration space approach. IEEE Transactions on Computers (2), 108–120 (1983), http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1676196
16. Luchnikov, V.A., Medvedev, N.N., Oger, L., Troadec, J.P.: Voronoi-delaunay analysis of voids in systems of nonspherical particles. Physical review E 59(6), 7205 (1999), http://pre.aps.org/abstract/PRE/v59/i6/p7205_1
17. Mamou, K., Ghorbel, F.: A simple and efficient approach for 3d mesh approximate convex decomposition. In: 2009 16th IEEE International Conference on Image Processing (ICIP). pp. 3501–3504 (2009)
18. MATLAB: version 8.2.0.701 (R2013b). The MathWorks Inc., Natick, MA (2013)
19. Mattingley, J., Boyd, S.: CVXGEN: Code generation for convex optimization (2013), http://cvxgen.com/docs/index.html
20. Mosek ApS: Inner and outer lowner-john ellipsoids (2014), http://docs.mosek.com/7.0/matlabfusion/Inner_and_outer_L_wner-John_Ellipsoids.html
21. Mosek ApS: The MOSEK optimization software (2014), http://www.mosek.com/
22. Sarmiento, A., Murrieta-Cid, R., Hutchinson, S.: A sample-based convex cover for rapidly finding an object in a 3-d environment. In: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on. p. 34863491. IEEE (2005), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1570649

23. Sastry, S., Corti, D.S., Debenedetti, P.G., Stillinger, F.H.: Statistical geometry of particle packings.i.algorithm for exact determination of connectivity, volume, and surface areas of void space in monodisperse and polydisperse sphere packings. Physical Review E 56(5), 5524–5532 (1997), http://link.aps.org/doi/10.1103/PhysRevE.56.5524