Convex Segmentation and Mixed-Integer Footstep Planning for a Walking Robot

by

Robin L. H. Deits

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2014

Signature of Author .....
Department of Electrical Engineering and Computer Science
July 29, 2014

Certified by .....
Russ Tedrake
Professor of Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Chair, Department Committee on Graduate Theses

Convex Segmentation and Mixed-Integer Footstep Planning for a Walking Robot

by

Robin L. H. Deits

Submitted to the Department of Electrical Engineering and Computer Science
on July 29, 2014, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This work presents a novel formulation of the footstep planning problem as a mixed-integer convex optimization. The footstep planning problem involves choosing a set of footstep locations which a walking robot can follow to safely reach a goal through an environment with obstacles. Rather than attempting to avoid the obstacles, which would require non-convex constraints, we use integer variables to assign each footstep to a convex region of obstacle-free terrain, while simultaneously optimizing its pose within that safe region. Since existing methods for generating convex obstacle-free regions were ill-suited to this task, we also present IRIS (Iterative Regional Inflation by Semidefinite programming), a new method to generate such regions through a series of convex optimizations. Combining IRIS with the mixed-integer optimization gives a complete footstep planning architecture, which can produce complex footstep plans on heightmap data constructed from onboard sensors. We demonstrate the footstep planner in simulated environments and with real data sensed by the Atlas humanoid, and we discuss future applications to running robots, aerial vehicles, and robots with more than two legs.

Thesis Supervisor: Russ Tedrake
Title: Professor of Computer Science

Contents

Acknowledgments

None of this work would have been possible without the guidance and support of my advisor, Russ, who pushed me in all the right directions at all the right times, and whose guidance and encouragement have been unwavering over the last two years. Nor would it have been accomplished without a lifetime of love and support (and proofreading) from my parents, Tom and Connie, without whom I would have been completely lost. I am immensely grateful to my girlfriend Michele, who has been a constant source of encouragement, ideas, and love, no matter how hard it has been to drag me out of lab.

I would also like to thank the Fannie and John Hertz Foundation, who have done far more to help my education than simply funding my research. The Foundation has consistently pushed me to think bigger and put me in touch with the people who have showed me how to do just that. The MIT Energy Initiative has also been a valuable source of support for me over the last two years.

Finally, I cannot overstate my gratitude to all of the members of the Robot Locomotion Group and the MIT DARPA Robotics Challenge team. I have spent the last two years surrounded by an incredible group of brilliant people, and I am inspired by their example and uplifted by their encouragement and conversation.

Preface

This work is organized in four chapters. Chapter 1 discusses the motivation and background of the footstep planning problem and introduces a new approach based on IRIS, a novel method of convex segmentation. Chapter 2 describes IRIS in detail, and Chapter 3 discusses the footstep planner which combines the results of IRIS with a mixed-integer convex optimization. Finally, Chapter 4 discusses future work using the techniques developed here.

Chapters 2 and 3 were originally written as independent papers and can generally be read separately. Chapter 2 was published as

> Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Workshop on the Algorithmic Foundations of Robotics*, Istanbul, Turkey, 2014.

Chapter 3 is currently under review.

Chapter 1

Introduction

1.1 Motivation

In order for robots to leave the laboratory, they must be able to do more than just roll across smooth floors. Legged robots can have a tremendous advantage in environments which are too steep, cluttered, or rough for wheels to traverse, but planning and controlling the movement of a walking robot introduces a wide variety of new challenges, as a walking robot is both an underactuated device and a hybrid dynamical system. A walking robot is underactuated because we cannot directly actuate its entire state–that is, rather than having a controllable joint for every degree of freedom (as an industrial robot arm might), a legged robot can only actuate its own joint positions and must use its dynamics and its contact with the world in order to move through space. Such a walking robot is also a hybrid system because the continuous dynamics with which it behaves undergo discrete changes when the robot makes or breaks contact with the environment.

The task of causing a robot to move through its environment is sufficiently complicated that we will choose to separate it into two pieces: (1) planning, in which we attempt to define a trajectory of some of the state variables from the initial state to the goal and (2) control, in which we find and execute the complete trajectory of states and inputs to drive the robot to the goal. In the case of legged locomotion, it is common to further divide the planning problem into two subparts, which I will label as the *footstep planning* and *walking planning* problems, defined as follows:

Definition (The Footstep Planning Problem)**.** *Given initial state x_0 and target set of states X_f find an ordered sequence of footstep positions $s_1, s_2, \ldots, s_n$ such that following the footstep sequence will safely bring the robot from x_0 to some $x_f \in X_f$.*

Definition (The Walking Planning Problem)**.** *Given x_0, X_f, and $s_1, s_2, \ldots, s_n$, find a dynamically feasible and controllable state trajectory through the footstep sequence from x_0 to $x_f \in X_f$.*

Our approach differs from this description somewhat, in that we add to the footstep planning problem the task of finding regions of safe terrain and assigning footstep to those regions, but

the distinction between planning target footstep locations and planning the full trajectory of the robot's state remains.

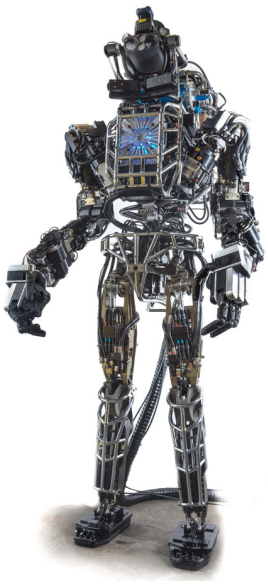
It is not at all obvious that this division of the problem is likely to result in acceptable locomotion plans, but it has proven to be effective in practice, as seen in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. The primary issue with this division is defining and evaluating *safety* as mentioned in Def. 1.1. We can construct some necessary conditions for the safety of a set of footsteps: (1) adjacent foot placements must be kinematically reachable without self-collision and (2) foot placements must avoid unsafe terrain, e.g. terrain which is too rough, too slippery, too weak, too steep, etc. These conditions on their own are certainly not sufficient to ensure that there is a dynamically feasible trajectory through the given footsteps, or that such a trajectory can be executed by our controller. Existing approaches have typically focused on making conservative footstep plans with relatively short strides, which are likely to admit feasible dynamic trajectories. Even neglecting dynamics, it has still proven to be difficult to design a footstep planner which can produce walking motions which avoid obstacles and can handle complex terrain environments. We are motivated to develop such a planner.

An ideal footstep planner would produce kinematically reachable and dynamically feasible footsteps across any terrain, while avoiding obstacles and ensuring a globally optimal solution with regards to a reasonable cost function like energy consumed or time spent walking. While we have not come quite so far yet, this thesis presents a new class of footstep planner that handles rough terrain walking through an optimization which can be solved to its global minimum. This new footstep planner also creates new possibilities for the inclusion of the dynamics of walking in the planning process, and may open up entirely new categories of terrain to biped or multi-legged robots.

1.1.1 Target Application

To develop and test different approaches to footstep planning in the real world, we use the Atlas robot, built by Boston Dynamics and shown in Fig. 1-1. Atlas was developed to participate in the DARPA Robotics Challenge (DRC), a series of competitions designed to

support and inspire the development of robotic hardware and software for disaster response. Its humanoid form results in part from a need to operate in environments designed for humans, as driving cars, climbing ladders, and passing through doorways are all necessary parts of the DRC competition.



(a) The Atlas robot from Boston Dynamics [11].

(b) The rough terrain course used during the DRC Trials held in December 2013.

Figure 1-1: The Atlas robot from Boston Dynamics, along with one of its environments from the DARPA Robotics Challenge (DRC) Trials in 2013. The DRC is intended to promote the development of robots that can respond to disasters in human environments and is inspired by the need to prevent and recover from disasters like the nuclear accident at Fukushima, Japan.

1.2 Existing Approaches to Footstep Planning

The treatment of footstep planning as a separate subproblem is common in the literature. I will discuss a number of planning architectures that have been developed over the past decade, all of which fall into the broad categories defined here:

1. Fixed footstep action sets, in which a small number of relative foot displacements (the

offset from one foot position to the next) are enumerated offline and then used to build a tree of possible footstep plans: Sect. 1.2.1, 1.2.2.

2. Hybrid continuous-discrete motion planners which use a continuous path-planning algorithm to inform the discrete search for step positions: Sect. 1.2.3.

3. Fixed contact locations, which search for transitional robot configurations between predetermined contact states: Sect. 1.2.4.

4. Optimization-based approaches, which perform a continuous optimization over the footstep positions given constraints related to the reachability or dynamics of the robot: Sect. 1.2.5.

1.2.1 Fixed Action Sets

Kuffner et al. describe a very straightforward discrete search approach to footstep planning, which is capable of finding paths through flat terrain with scattered obstacles [1, 3]. Their approach involves enumerating a fixed set of relative footstep displacements (i.e. positions of the left foot relative to the prior position of the right foot and vice versa), then constructing a tree in which each branch is a choice among those possible displacements. Nodes in the tree in which the foot would be placed in contact with an obstacle in the environment are pruned, and an A^* search is used to traverse the tree and find a sequence of foot positions from the start to the goal.

This approach involves a number of restrictive assumptions, which are addressed by the new mixed-integer convex optimization described in Chapter 3:

1. The robot is restricted to a very small set of relative foot positions, rather than performing some continuous search over those positions.

2. The ground is assumed to be flat except where obstacles are present, and the obstacles are never allowed to be used as stepping surfaces.

3. The obstacles are assumed to only restrict the position of the foot itself, rather than, for example, the upper body of the robot. In reality, a wide-shouldered humanoid like Atlas can easily place its foot next to a tall obstacle like a table or wall, only to have its upper body collide with that obstacle when its weight shifts during stepping.

4. The obstacles are all assumed to be low enough for the robot to step over them. That is, the swing trajectory of the foot is not considered in the footstep placement.

However, despite these restrictions, the fact that the fixed action sets allow the complex problem of footstep planning to be neatly formulated as a single discrete search have led to successful applications of this approach by a variety of researchers. Such fixed action sets are seen in the work of Lorch [2], Michel [5], Baudouin [8], and Huang [10].

1.2.2 Adaptive Action Sets

Chestnutt's work in 2003 extends the approach proposed by Kuffner by removing the restriction to perfectly flat terrain and instead including a measure of the roughness and steepness of the terrain in the cost function used during search [4]. Additionally, his 2007 paper further extends the method by allowing some variation in the initial action set (the set of precomputed relative foot positions) [6]. If one of the fixed actions chosen in the search process would put the foot into collision with an obstacle, a local search is performed around that foot position to attempt to find a nearby safe step location. If such a location is found, it can be mapped back into a modified action (a slightly different relative step position), and the search can be continued. This gives the search algorithm more flexibility when dealing with terrain obstacles without increasing the complexity when no obstacles are present. Chestnutt also relaxes the assumption that the ground be flat, but still does not consider the effect of the terrain on any part of the robot other than the foot itself.

Michel, on the other hand, uses the same fixed action set approach as Kuffner, but adds online replanning [5]. After executing each footstep, the planner computes a new plan to reach the goal, which allows some robustness if the robot fails to perfectly achieve a planned step location.

1.2.3 Hybrid Continuous/Discrete

A radically different approach to the footstep planning problem can be found in recent work by Perrin et. al [9]. Their approach involves first searching for a continuous plan which is, in their terminology, *weakly collision-free*. This path describes the motion of a region representing the reachable set of the right and left feet through space, with the condition that at every point in time it must contain a foot location which is not in collision with any obstacle. They can then convert this path into a sequence of discrete footsteps. Additionally, Perrin considers collisions of the upper body with obstacles by defining a hybrid bounding box consisting of a continuous volume swept by the upper body of the robot and a series of separate volumes corresponding to the footstep locations. This enables the planner to differentiate between obstacles which can be stepped over and those which must be avoided by choosing a different route.

Baudouin, Perrin, et al. apply a method which is similar to the hybrid bounding box and to Kuffner and Chestnutt's fixed action sets [8]. Offline, they compute a set of 200 possible relative footstep positions (which attempt to cover the kinematically reachable space of the robot's legs), and for each one they generate a swept volume approximation for the entire body of the robot. Baudouin's experimental results show that an A^* search performs poorly for such a large set of possible actions, so a Rapidly-exploring Random Tree (RRT) method is used to search over the choice of footstep actions in order to generate a path to a given goal around a set of 3D obstacles. Their approach is efficient enough to be used online, replanning the robot's path every few steps, but it still requires that the environment be a flat plane, with the only obstacles consisting of known 3D shapes tracked by motion capture cameras. The authors also note that the RRT frequently produces sub-optimal plans which require many more steps to reach the goal than are truly necessary.

Shkolnik et al. also use a modified RRT to search for step actions, but expands the approach to dynamic motions of LittleDog, a quadrupedal robot capable of traversing rough terrain [12]. Shkolnik defines motion primitives describing half of a dynamic bounding action, then performs a sample-based random planning approach to find contact locations which are

compatible with this action and which bring the robot closer to the goal. The use of a dynamic motion primitive enables them to generate plans in the low-dimensional space of step locations while still respecting the full dynamics of the robot.

1.2.4 Fixed Contact Positions

Rather than treating the footstep planning problem as one of placing flat feet on the terrain, there have also been attempts to plan a contact sequence in a more general way. Hauser and Bretl demonstrated a motion planner for walking and climbing which searches over a graph of *stances*, where a *stance* describes the set of contacts between points on the robot and points on the terrain [13]. Stances are adjacent in the graph if they differ by only one contact and have some configuration in common which satisfies the requirements of reachability and static equilibrium. Thus, if the only allowable contact points are on the robot's feet, then choosing a sequence of stances corresponds to choosing foot placements in the world. Additionally, if contact points on the robot's hands or body are allowed, then this method can encapsulate statically stable walking, crawling, and climbing.

The free-climbing problem appears to be quite amenable to the discrete contact region assumption. If a wall which might be climbed is composed of fixed *holds*, and the limbs of the robot may only make contact with the wall at those holds, then a discrete search over contact configurations and the motions between them becomes more reasonable. Bretl uses such an environment to present a multi-step planning algorithm capable of bringing a 4-limbed robot up a free climbing wall [14].

Unfortunately, the complexity of this problem increases as more possible contact location are allowed. When dealing with a large environment with a continuum of possible contact positions, one must choose between a conservative coarse discretization into a small number of contact positions or a fine discretization into many positions at the cost of increased computational complexity. This problem is by no means insurmountable: Neuhaus et al. demonstrate a walking planning system for the LittleDog quadruped over rough terrain using an efficient search over a large set of possible contact positions [7]. They overcome

the computational complexity of the search among contact positions by first finding an approximate trajectory for the center of the robot's body which reaches the goal position, then performing a depth-first search over sequences of footstep contacts, guided by heuristics which try to maintain that nominal body trajectory and some nominal stride lengths. The depth-first search does not guarantee optimality, but the correct heuristics can enable the system to find a good plan in a matter of seconds.

1.2.5 Continuous Optimizations

Herdt et al. move even farther from the treatment of footstep planning as an independent subproblem. Instead, their Model Predictive Control approach uses a quadratic program (QP) to simultaneously solve for the control input and the target footstep locations in order to ensure that the center of pressure (the midpoint of the ground reaction forces) remains within the polygon of contact points of the feet [15]. Herdt uses a set of linear constraints to limit the relative displacement between footsteps, ensuring that, for example, the robot does not try to take a stride which is longer than the reach of its legs and that the right foot does not ever step on or collide with the left. To ensure that all of the constraints in the problem are linear, the orientations of the footsteps must be fixed beforehand. This approach also does not handle obstacle avoidance, which cannot be captured by their linear constraints. However, the pure QP formulation that Herdt produces has the advantage that it can be solved very quickly (on the order of one millisecond or less for a plan of two footsteps), and the success of this algorithm has inspired our own mixed-integer quadratic formulation.

1.2.6 Analysis of the Existing Footstep Planning Approaches

I choose to focus on footstep planning for a relatively fast, powerful biped robot like the Boston Dynamics Atlas humanoid, used for a variety of tasks as part of the DARPA Robotics Challenge, and I will tailor the discussion of these existing approaches to that system.

The fixed action set methods have shown significant experimental success in cluttered environments and are able to produce complex plans around or over obstacles on the ground.

On the other hand, they generally neglect the dynamic constraints introduced by faster walking motions and are restricted to robots which move quasi-statically. The approaches presented above all assume that the set of reachable steps is fixed, regardless of the state of the robot. We have relied on this assumption in the past with success, since it is generally easy for Atlas to change direction from step to step while walking slowly, but as a robot moves faster we expect the reachable set to change significantly. It is extremely difficult for a human to transition from walking quickly or running forward to walking backward in the course of a single step, and the same will likely hold true for Atlas. Additionally, the fixed action sets suffer from the tradeoffs related to the size of the set: a larger action set allows more possible foot placements and thus more optimal plans, but is more difficult or even intractable to search over.

Methods based on fixed contact points are attractive when the environment drastically restricts the number of possible contact locations, such as when climbing up a rock face with a limited number of holds [14]. Extending these methods to more open environments like the DRC terrain course would require a way to make the search across a multitude of contact locations tractable. Neuhaus finds such a tractable search, but at the expense of optimality of the solution [7]. If possible, we would like to maintain a continuous optimization to find the exact footstep locations, rather than enumerating a list of possible poses.

The hybrid bounding box approach used by Perrin is attractive in a course like the rough terrain of Fig. 1-1, where avoiding obstacles and collisions between the body and the terrain is critically important. The weakly collision-free paths introduced in the same work also provide an excellent simplification of the path-planning problem for walking robots. However, robots which walk quickly, run, or climb up and down will have additional constraints on their allowable motions, and these constraints are not yet handled by Perrin's planner.

Finally, the convex optimization proposed by Herdt produces dynamically feasible walking motions on flat ground using an extremely efficient quadratic program. Herdt chooses to ignore footstep rotation and obstacle avoidance to obtain this implementation; we choose instead to introduce integer constraints, sacrificing some of the speed with which we can

solve the problem in order to incorporate those additional terms.

1.3 A New Approach: Convex Decomposition

We can choose to represent a terrain as being divided into free space, where the robot can safely step, and obstacles. Obstacles generally result in the introduction of non-convex constraints to a given planning problem, since the space outside a closed, bounded obstacle is a non-convex set. We cannot generally find strong guarantees of complete and/or global optimal solutions to non-convex problems. I propose that an effective reformulation of the problem is to instead represent the terrain as a set of convex safe regions. The problem of planning is thus transformed from a continuous non-convex optimization into a discrete choice among those regions and a corresponding convex optimization. This separation meshes well with the footstep planning problem, in which we must decide the general location of the footsteps (e.g. which stair on a staircase, which rung on a ladder, which cinder block in Fig. 1-1, etc.) and also the precise location of each step to ensure the plan can be executed by the robot. The resulting mixed-integer convex optimization can be solved to its global optimum efficiently.

In the context of the DARPA Robotics Challenge program, the most compelling advantage of representing a terrain as a set of safe regions is that it should allow for very effective input from our expert operator. The rough terrain faced during the DRC Trials consisted of a simulated rubble field, with many small regions of safe footholds separated by cracks and gaps, as shown in Fig. 1-1. My experience as an operator during the Trials was divided between the high-level strategy of which footholds to choose, which felt like an effective use of my knowledge of the task and goals, and the low-level tedium of carefully and precisely choosing individual foot positions, which was not. If we can transform the available footholds into a few (convex) regions, then the strategic planning maps directly to the choice of which convex region to place the foot into, and the dynamic walking planning software which our lab is developing can take over the precise position of the feet. Such a convex decomposition is shown in Fig. 1-2, in which LIDAR data collected from a practice walking course was

used to generate two convex regions of safe terrain.

Even without operator input, the separation of the non-convex planning problem into a discrete choice among convex regions allows us to use mature tools to autonomously choose footstep placements. Software packages such as Gurobi [16] enable efficient solving of mixed-integer quadratic programs, which involve a combination of continuous optimization with convex constraints and discrete optimization of some integer-valued variables. This maps nicely to the footstep planning problem as formulated in Chapter 3.

1.3.1 Non-Convexity of Obstacle Avoidance

The primary difficulty that we are attempting to overcome is the fact that avoiding obstacle regions typically introduces non-convex constraints into our optimization. This is due to the fact that the area outside a closed, bounded obstacle is a non-convex set, which we can easily show:

> Take an obstacle $\mathcal{O}$ which is bounded by Euclidean distance d, and choose an arbitrary unit vector $\hat{n}$. Choose a point $P \in \mathcal{O}$ and a line through P in the direction specified by $\hat{n}$. Boundedness of $\mathcal{O}$ implies that points $Q = P + (d+\epsilon)\hat{n}$ and $R = P - (d+\epsilon)\hat{n}$ must not be in $\mathcal{O}$. But P is a convex combination of Q and R, so the set of points outside of $\mathcal{O}$ is non-convex.

If we instead assign each footstep to a convex region of obstacle-free space, then we can optimize the precise positions of the feet as a convex optimization. Of course, we must also choose into which region each footstep must be assigned, which introduces integer variables into our optimization. The formulation that results, which is discussed in Chapter 3, is a binary integer program. Of course, this is by no means a free lunch: even binary integer programs with only linear constraints (a subset of the problem class that we use in Chapter 3) are known to be NP-complete [17]. However, by constructing a problem of this type, we can benefit from decades of research and algorithmic development by using cutting-edge mixed-integer programming tools which can readily find globally optimal solutions to many problems.

1.3.2 Generating Convex Regions

If we are to successfully reformulate the footstep planning problem as an assignment of steps to convex regions, then we must find some way to break down an arbitrary environment into such regions. The combinatorial problem of assigning footsteps scales badly with the number of safe regions: if we have r regions and n steps, then we have r^n possible assignments of steps to safe regions. We would like to find as minimal a set of convex safe regions as possible while still covering all of the possible areas to step. Unfortunately, even in relatively simple environments consisting of a polygonal environment with polygonal obstacles, finding the minimum number of (possibly overlapping) convex regions to cover the entire environment is itself an NP-hard problem [18]. For complex environments, then, we must sacrifice some degree of optimality to make the problem tractable.

Approximately Minimal Decompositions

Many existing approaches to the convex segmentation problem choose to sacrifice the requirement that the decomposition of the environment into convex pieces be minimal; that is, they decompose the entire environment into relatively few pieces but produce more pieces than an optimal (NP-hard) decomposition. Feng and Pavlidis described such a decomposition algorithm in 1975 and successfully used it to segment images of chromosomes and Chinese characters into convex pieces for pattern recognition purposes, but they provide no guarantee of optimality [19].

Keil describes an efficient algorithm for solving a restricted form of the convex decomposition problem, in which the vertices of the convex pieces are restricted to lie only at the vertices of the original polygon [20]. This problem can be solved to optimality in a running time of $O(N^2 m \log m)$, where m is the number of vertices and N is the number of *notches*, that is, the number of vertices with reflex interior angles. Unfortunately, as with the similar approach developed by Chazelle [21], this algorithm does not apply in the case of a polygon with holes, so we cannot use it to segment the free space outside the obstacles.

Eidenbenz and Widmayer describe a polynomial-time algorithm which finds an approxi-

mately minimal convex cover for a polygon with holes. Their method involves performing a complete triangulation of the polygon, computing the intersection of all lines in the triangulation, and then restricting the vertices of the convex pieces to only lie on those intersection points. Their algorithm is polynomial, but it still does not scale well: their proof demonstrates a running time of $O(m^{29} \log m)$, where m is the total number of vertices. In the DRC, our complex terrain frequently results in thousands of small obstacles with many vertices each, so this scaling is not encouraging for our use case.

Approximately Convex Decompositions

On the other hand, a number of efficient algorithms exist for computing a decomposition of a polygon or 3D mesh into a set of *approximately* convex pieces. Since a complete decomposition of an environment into convex pieces may result in far too many segments, these approaches allow each piece to have some small degree of concavity. What it means for a shape to be approximately convex is not generally agreed upon, but metrics based on the largest distance from any vertex of a piece to the boundary of its convex hull have been used with success for 2D polygons [22] and 3D meshes [23].

Lien accomplishes an approximate convex decomposition in 2D by recursively identifying notches and splitting the pieces at those points until each piece is sufficiently convex [22]. Mamou extends the approximate convex segmentation to 3D meshes [23]. Rather than finding notches and splitting the shape, Mamou converts a mesh into its dual graph, which maps faces in the mesh to nodes in the graph such that faces which are adjacent in the mesh produce connected nodes in the graph. He then performs a heuristically-informed search of this graph to find large connected components corresponding to pieces of the mesh which are nearly convex.

These approximately convex decompositions can result in dramatically fewer regions being produced [24], but they are not well suited to our task. Our goal is to produce convex descriptions of safe regions to make optimization of footstep placement more tractable, so we must find a convex representation of each region. If we instead receive an *approximately*

convex region, we could take the convex hull of that set, but then our "safe" set might contain part of some obstacle, which negates the entire purpose of finding safe regions. On the other hand, we could attempt to split the approximately convex result into entirely convex pieces, but then we are back to the original convex segmentation problem with its computational difficulty and large number of resulting convex pieces.

1.3.3 A New Approach: IRIS

The IRIS algorithm, which we have recently developed, chooses to sacrifice the requirement that the convex pieces cover the entire space. Instead, it finds one or more large convex safe regions near some area of interest, drastically reducing the number of safe regions presented to the footstep planer and making the search over the r^n possible assignments much more tractable. Two convex safe regions produced by IRIS are shown in Fig. 1-2. IRIS is based on a set of alternating convex optimizations and is designed to perform well even in environments consisting of very many obstacles. The algorithm itself is introduced and discussed in detail in Chapter 2.

1.3.4 Related Applications of Mixed-Integer Optimization

The idea of handling non-convex obstacle avoidance problems through some form of discrete search is not new, but I believe that it is a space with room for new exploration. Richards and Bellingham (in Jonathan How's research group at MIT) study the problem of planning trajectories for multiple aerial vehicles which must reach mission waypoints while avoiding convex polygonal obstacles [26]. They handle the problem of avoiding those obstacles by defining a large set of binary integer variables. For every discrete position in the planned trajectory, they assign a binary variable to every half-space constraint for every obstacle in the environment. These binary variables are linearly constrained to take a value of 1 if the associated trajectory point falls in the half-space corresponding to the associated obstacle's interior. As a result, they can add linear constraints on these binary variables to ensure that every trajectory point lies on the exterior side of at least one half-space

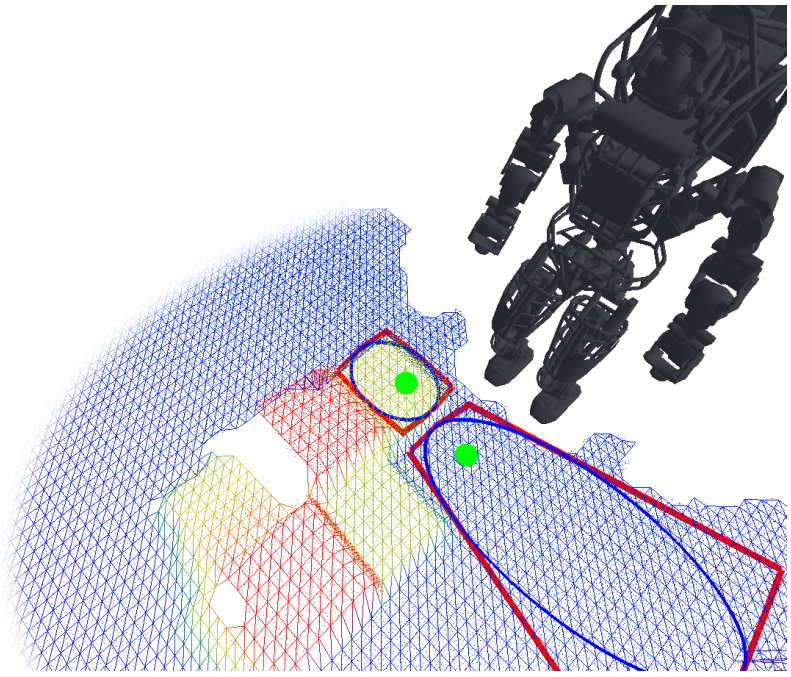


Figure 1-2: A visualization of an Atlas humanoid standing in front of a set of tilted steps, as seen in the DARPA Robotics Challenge 2014 trials [25], with two convex regions of safe terrain displayed (blue ellipses and red polytopes). The green circles indicate two points chosen by a human operator 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.

constraint per obstacle, and thus lies outside the obstacle. This formulation allows complex trajectory planning around obstacles to be defined as a Mixed-Integer Linear Program, for which efficient solvers such as Gurobi [16] can find a globally optimal solution. Unfortunately, the blowup in the number of binary variables with the number of obstacles, obstacle faces, and trajectory points rendered the problem computationally difficult for all but very small numbers of obstacles, and the Richards et al. moved to an approximate formulation for larger problems.

The footstep planning problem, combined with the convex segmentation problem, should be much more amenable to a technique like this for several reasons. For t trajectory points among m obstacles with k faces each, Richards et al. require tmk binary variables with

$\left(2^{mk}\right)^t$ possible assignments to search. By contrast, by pre-computing r convex safe regions and assigning each of the n footsteps to a region, we alter the search space to just r^n integer variable assignments. Whether this is an improvement over Richards depends of course on the relationship between the relative scales of the implementations. For the same number of footsteps and trajectory points ($n = t$), our discrete search should have lower complexity whenever $r < 2^{mk}$. Also, the continuous nature of vehicle planning means that large numbers of trajectory points t may be necessary to ensure that continuous safe trajectories are found, while the discrete footstep planning problem does not require this, so n need not exceed the actual desired number of footsteps (typically 2-20 in the DARPA Robotics Challenge Trials). Additionally, preliminary experiments with real terrain observed by the Atlas robot indicate that common tasks such as moving around a table or climbing over a set of steps can be accomplished with relatively small numbers of convex safe regions (typically less than 10), so our r can be kept relatively small. Optimizing over the assignment of steps to convex regions is actually quite tractable and typical problems can currently be solved in a few seconds to minutes, depending on the number of footsteps.

The binary variables introduced by Richards have also been used to handle homotopy class constraints for continuous trajectories [27]. Two trajectories are *homotopic* if one can be smoothly deformed into the other without intersecting the obstacle set, and a *homotopy class* is a set of trajectories which are mutually homotopic. Homotopy classes can capture broad descriptions of a trajectory, distinguishing e.g. between trajectories which go to the right of an obstacle from those which go to the left. Restrictions to a particular homotopy class can be handled by integer constraints in a manner which is very similar to the more general obstacle avoidance problem. Kim and Sreenath use the technique introduced by Richards of extending the faces of every polygonal obstacle across the entire environment [27]. The intersections of all of these lines form convex cells in the obstacle-free space (in fact, this technique is precisely a form of non-minimal convex decomposition, as discussed earlier). Kim assigns to each of these cells a unique letter, which allows trajectories which pass through many cells to be identified with a corresponding sequence of letters or *word*. Given

a desired homotopy class, Kim constructs all possible words corresponding to trajectories in that homotopy class. For each word in the class, Kim can run a Quadratic Program to find a smooth trajectory which is optimal according to some chosen cost function. Although Kim solves the problem iteratively, rather than directly as an MIQP, the use of discrete variables to capture regions in the obstacle-free space combined with continuous optimization maps well onto my proposed approach of assigning footsteps to discrete safe regions so that a convex optimization can be performed.

1.4 Putting it All Together: The Contributions of this Thesis

The remainder of this work is divided into three chapters. Chapter 2 introduces IRIS, our new method for generating a small number of large convex regions of safe terrain. Chapter 3 develops a mixed-integer convex program which can assign footsteps to those safe regions and optimize their poses in space. Finally, Chapter 4 discusses our future plans for footstep planning, focusing on running, multi-legged walking, and dynamic walking planning.

The footstep planner presented in Chapter 3 handles constraints on obstacle avoidance and kinematic reachability, but it does not consider the constraints involved in producing a dynamically feasible motion. The footstep plans it produces may, for example, require significant accelerations of the robot's center of mass which would cause the robot's feet to slip or its actuators to saturate. However, the particular formulation presented in this thesis should be amenable to the inclusion of dynamic constraints in the future. Herdt. et al have layed the groundwork for this [15]. They present a quadratic program to generate dynamically feasible motions of the center-of-mass in order to keep the center of pressure inside the support polygon of the feet, a necessary condition for dynamic feasibility. Their formulation does not include obstacle avoidance or footstep rotation, but both could be added with integer variables in much the same way that the mixed-integer planner does. This is discussed further in Sect. 4.1.

Chapter 2

Finding Convex Regions with IRIS

This chapter was originally published as "Computing Large Convex Regions of Obstacle-Free Space through Semidefinite Programming" by Robin Deits and Russ Tedrake, and appears in the Proceedings of the 2014 Workshop on the Algorithmic Foundations of Robotics [28].

Abstract

This chapter presents IRIS (Iterative Regional Inflation by Semi-definite 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 [29] implementation converges in seconds for environments with millions of obstacles.

2.1 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 typ-

ically 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 space into a minimum number of convex parts is NP-hard [30]. 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 2-1(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 2-1(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 2-1(c). Additional runs of the algorithm, seeded from the remaining obstacle-free space, could fill the remaining space if desired. Figure 2-2 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 [25].

Our approach, as described in Sect. 2.3, begins with an initial guess, defined as a point

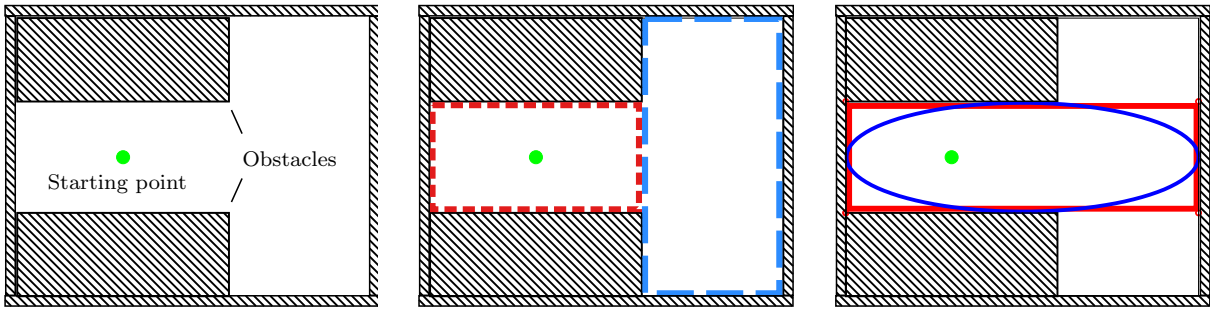


Figure 2-1: A simple 2D environment with two rectangular obstacles and a point of interest (left). The minimal non-overlapping convex decomposition of the obstacle-free space produces two polygonal regions (center), while our algorithm produces a larger convex region about the point of interest and an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

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 which point we return the polytope and inscribed ellipsoid. Examples of this procedure in 2D and 3D can be seen in Figs. 2-3 and 2-4, 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 [22, 23], 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. 2-5, 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

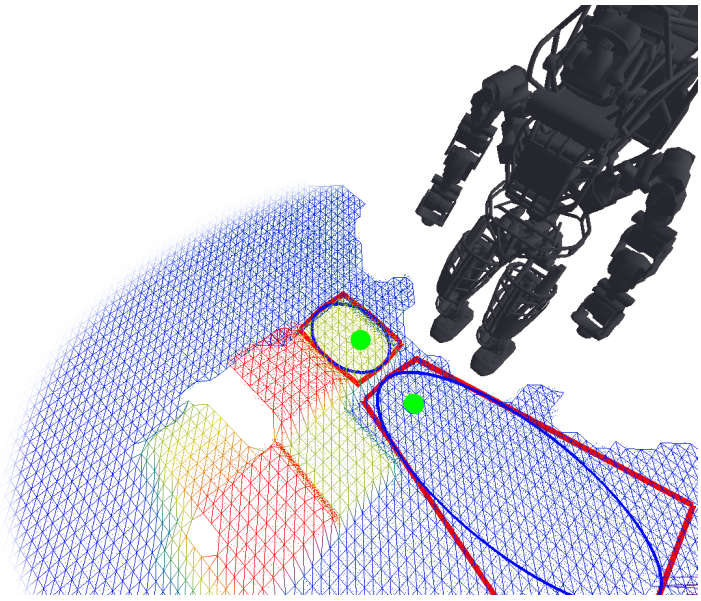


Figure 2-2: A visualization of an Atlas humanoid standing in front of a set of tilted steps, as seen in the DARPA Robotics Challenge 2014 trials [25], with two convex regions of safe terrain displayed (blue ellipses and red polytopes). The green circles indicate two points chosen by a human operator 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.

early should the region found ever cease to include the start point.

In the remainder of this chapter, 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. 2-5.

2.2 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 allowed some small degree of concavity [22]. 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 [23]. Liu's approach [31], 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 [18]. 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 [19]. 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 [32]. Finally, Sarmiento produces convex polytopic regions in N dimensions by sampling points in free space and checking visibility from a set of "guard" positions [33]. 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 [34] 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 [35]. Luchnikov extends this notion of searching for (non-convex) voids over the Vornoi network to non-spherical objects [36].

2.3 Technical Approach

2.3.1 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 [37] 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:

$$\begin{aligned} \underset{A,b,C,d}{\text{maximize}} \quad & \log\det C \\ \text{subject to} \quad & a_j^\top v_k \geq b_j \quad \text{for all points } v_k \in \mathcal{V}_j, \text{ for } j = 1,\ldots,N \\ & \sup_{\|\tilde{x}\|\leq 1} a_i^\top (C\tilde{x} + d) \leq b_i \quad \forall i = [1,\ldots,N] \end{aligned} \tag{2.1}$$

where a_j are the rows of A, b_j are the elements of b, $\mathcal{V}_j$ is the set of points in the convex obstacle $\mathcal{V}_j$, and N is the number of obstacles. The constraint that $a_j^\top v_k \geq b_j$ for all points $v_k \in \mathcal{V}_j$ forces all of the points in obstacle $\mathcal{V}_j$ 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. 2.3.4

$C_0 \leftarrow \epsilon I_{n\times n}$
$d_0 \leftarrow q_0$
$i \leftarrow 0$
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)

2.3.2 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 $C_0 \leftarrow \epsilon I_{n\times n}$.

2.3.3 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 $\mathcal{E}(C,d) = \{C\tilde{x} + d \mid \|\tilde{x}\| \leq 1\}$, we define a uniform expansion of $\mathcal{E}$ as

$$\mathcal{E}_\alpha \equiv \{C\tilde{x} + d \mid \|\tilde{x}\| \leq \alpha\} \quad \text{for some } \alpha \geq 1 \tag{2.2}$$

To find the closest point on an obstacle $\wr_j$ to the ellipsoid, we can search over values of α

$$\begin{aligned} \alpha^* = \arg\min_{\alpha} \quad & \alpha \\ \text{subject to} \quad & \mathcal{E}_\alpha \cap \wr_j \neq \varnothing \end{aligned} \tag{2.3}$$

We label the point of intersection between $\mathcal{E}_{\alpha^*}$ and $\wr_j$ 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 $\wr_j$, and, since $\mathcal{E} \subseteq \mathcal{E}_\alpha$for $\alpha \geq 1$, it also separates $\mathcal{E}$ from $\wr_j$. We choose the sign of a_j and b_j such that $a_j^\top x \geq b_j$ for every $x \in \wr_j$.

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 (2.3), 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 $\wr_j$. Our ellipsoid is defined as an image of the unit ball in $\mathbb{R}^n$: $\mathcal{E} = \{C\tilde{x} + d \mid \|\tilde{x}\| \leq 1\}$, so we construct the inverse of this image map:

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

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:

$$\begin{aligned} \underset{\tilde{x}\in\mathbb{R}^n, w\in\mathbb{R}^m}{\arg\min} \quad & \|\tilde{x}\|^2 \\ \text{subject to} \quad & \begin{bmatrix} \tilde{v}_{j,1} & \tilde{v}_{j,2} & \ldots & \tilde{v}_{j,m} \end{bmatrix} w = \tilde{x} \\ & \sum_{i=1}^{m} w_i = 1 \\ & w_i \geq 0 \end{aligned} \tag{2.4}$$

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 [38]. In our implementation, we achieved the best

performance by solving the original quadratic program in (2.4) using a task-specific solver generated by the CVXGEN tools [39]. 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 [40] and Gurobi [16], 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

$$\mathcal{E} = \left\{ x \mid (x-d)^\top C^{-1} C^{-\top} (x-d) \leq 1 \right\} \tag{2.5}$$

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

$$\begin{aligned} a_j &= \nabla_x \left[(x-d)^\top C^{-1} C^{-\top} (x-d) \right] \big|_{x^*} \\ &= 2C^{-1} C^{-\top} (x^* - d). \end{aligned} \tag{2.6}$$

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

$$b_j = a_j^\top x^*. \tag{2.7}$$

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. 2-3, 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. 2.3.3

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

2.3.4 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:

$$\mathcal{E} = \{C\tilde{x} + d \mid \|\tilde{x}\| \leq 1\} \tag{2.8}$$

with the volume of the ellipsoid proportional to the determinant of C [37]. 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

$$\begin{aligned} \underset{C,d}{\text{maximize}} \quad & \log\det C \\ \text{subject to} \quad & \sup_{\|\tilde{x}\|\leq 1}(a_i^\top C\tilde{x}) + a_i^\top d \leq b_i \quad \forall i = [1,\ldots,N] \\ & C \succeq 0 \end{aligned} \tag{2.9}$$

as stated by Boyd [37], 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:

$$\begin{aligned} \underset{C,d}{\text{maximize}} \quad & \log\det C \\ \text{subject to} \quad & \left\|a_i^\top C\right\| + a_i^\top \leq b_i \forall i = [1,\ldots,N] \\ & C \succeq 0 \end{aligned} \tag{2.10}$$

which is a convex optimization [37]. 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 [41]. Ben-Tal and Nemirovski, meanwhile, present a method for computing the ellipsoid through a semidefinite and conic quadratically constrained optimization [42], and we use this approach, as implemented by Mosek [43], in our code. We have also successfully used CVX, a tool for specifying and solving convex problems [44], to solve

(2.10), 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.

2.3.5 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.

2.4 Results

We implemented the proposed algorithm in MATLAB [29], using CVXGEN [39] to solve each least-distance QP and Mosek [43] 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. 2-3. 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. 2-4, 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 2-6 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.

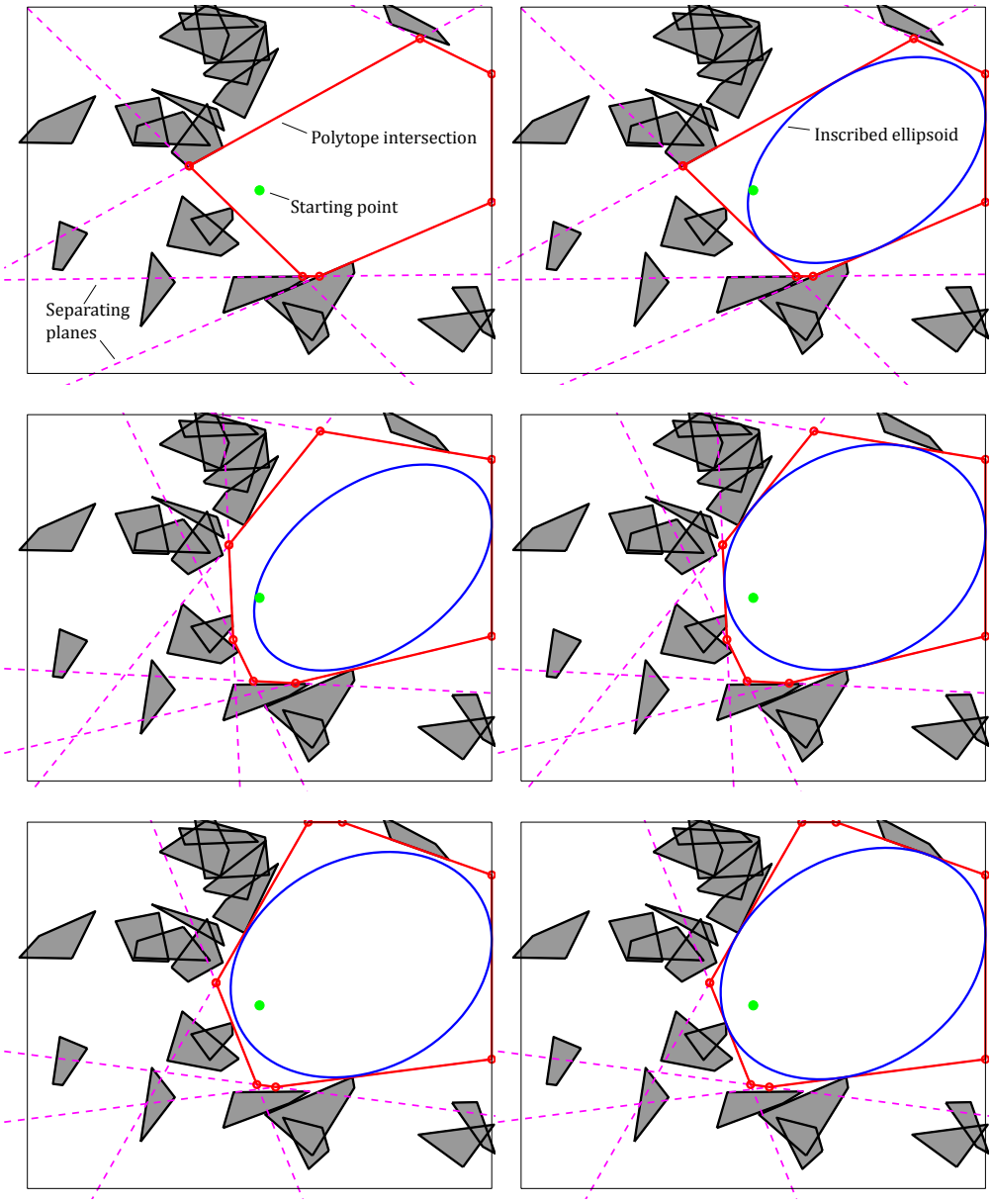


Figure 2-3: A demonstration of the IRIS algorithm in a planar environment consisting of 20 uniformly randomly placed convex obstacles and a square boundary. Each row above shows one complete iteration of the algorithm: on the left, the hyperplanes are generated, and their polytope intersection is computed. On the right, the ellipse is inflated inside the polytope. After three iterations, the ellipse has ceased to grow, and the algorithm has converged.

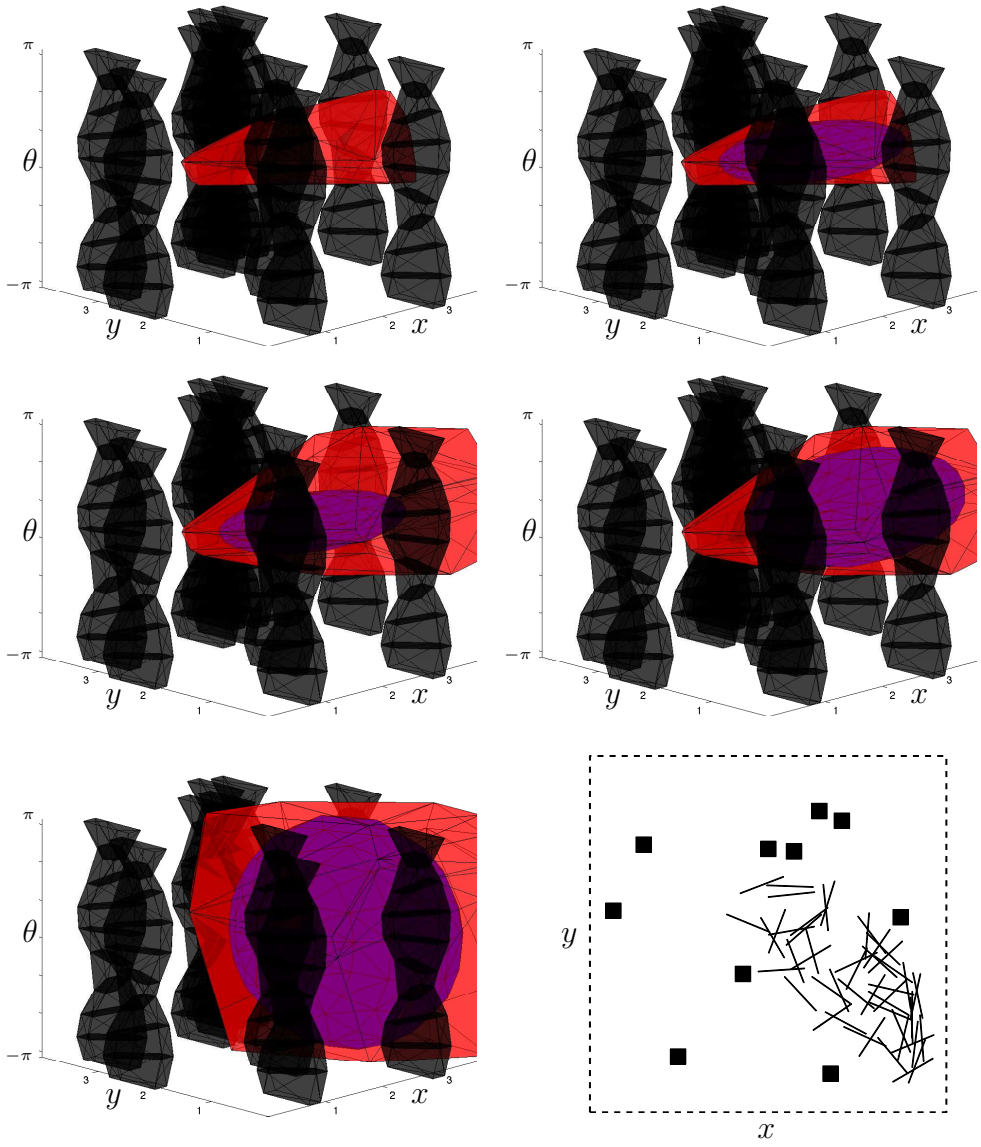


Figure 2-4: An example of generating a large convex region in configuration space. A 2D environment containing 10 square obstacles was generated, and the configuration space obstacles for a rod-shaped robot in that environment were built by dividing the orientations of the robot into 10 bins and constructing a convex body for each range of orientations [45]. 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.

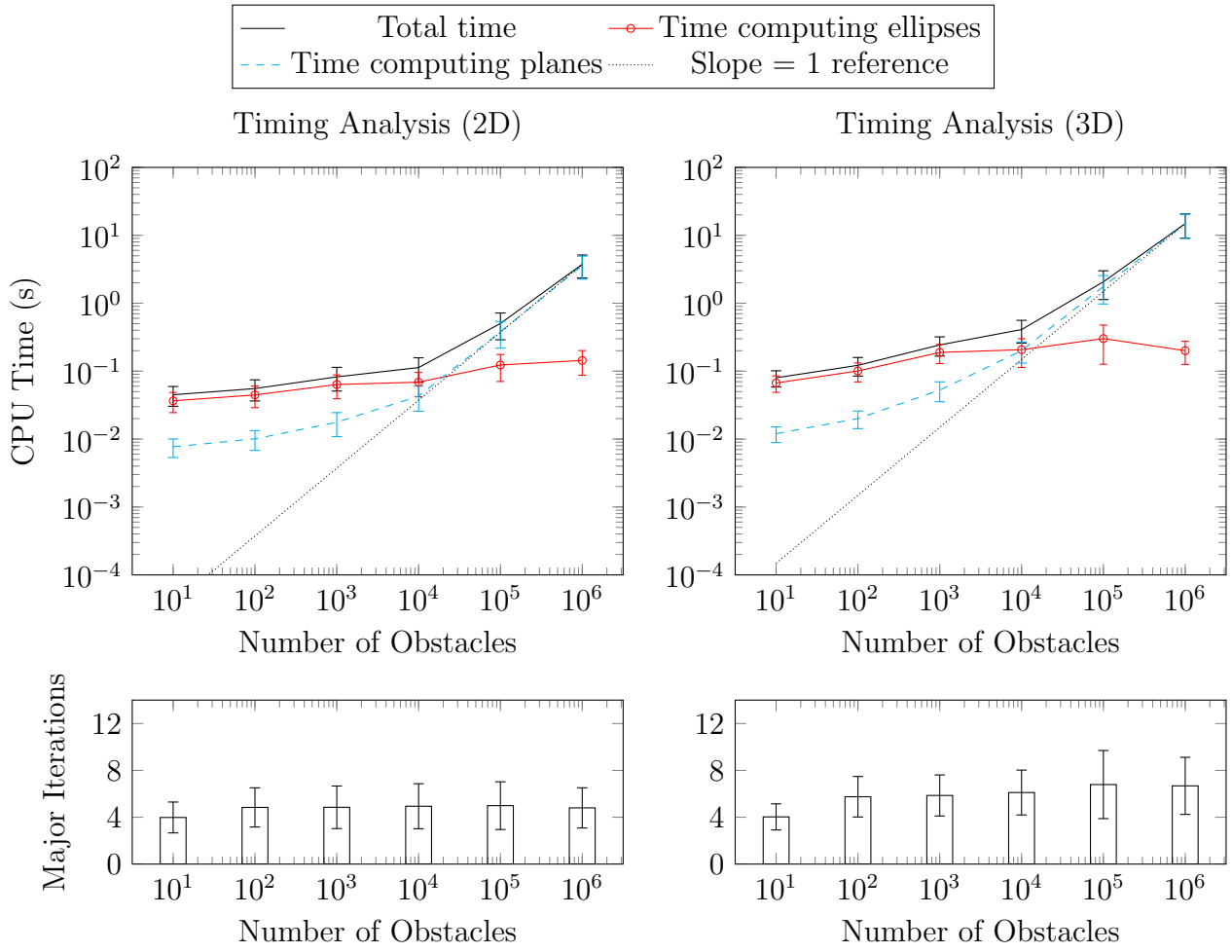


Figure 2-5: Timing results of 1200 runs of the IRIS algorithm implemented in MATLAB on an Intel i7 processor at 2.5 GHz with 8 Gb of RAM. In each of the 2D and 3D cases, we generated 100 environments at 6 logarithmically spaced numbers of obstacles between 10^1 and 10^6. Obstacles were uniformly randomly 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.

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

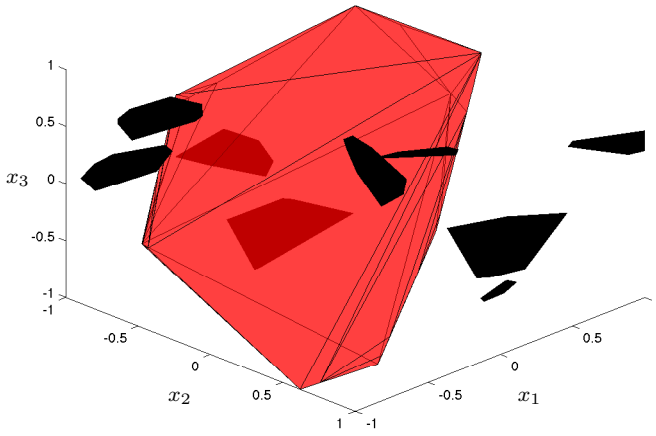


Figure 2-6: An example of the output of the algorithm in 4-dimensional space. We generated 4-dimensional obstacles consisting of uniformly random points centered on uniformly randomly chosen locations in $[-1, 1]^4$. The figure shows the 3-dimensional intersection with the $x_4 = 0$ plane of the obstacles and the polytope produced by the IRIS algorithm.

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. 2.3.3. 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. 2-5.

2.5 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.

In the next chapter, we demonstrate the application of IRIS to the footstep planning problem for the Atlas biped. We allow the user to select a point in space on a terrain map, compute an obstacle free region, and find a set of footstep positions which optimize 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.

2.6 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 chapter, as well as animations of IRIS running in 2D, 3D, and 4D.

Chapter 3

Planning Footsteps with Mixed-Integer Convex Optimization

This chapter has been submitted as a separate publication and is currently under review.

Abstract

We present a new method for planning footstep placements for a robot walking on uneven terrain with obstacles, using a mixed-integer quadratically-constrained quadratic program (MIQCQP). Our approach is unique in that it handles obstacle avoidance, kinematic reachability, and rotation of footstep placements, which typically have required non-convex constraints, in a single mixed-integer optimization that can be efficiently solved to its global optimum. Reachability is enforced through a convex inner approximation of the reachable space for the robot's feet. Rotation of the footsteps is handled by a piecewise linear approximation of sine and cosine, designed to ensure that the approximation never overestimates the robot's reachability. Obstacle avoidance is ensured by decomposing the environment into convex regions of obstacle-free configuration space and assigning each footstep to one such safe region. We demonstrate this technique in simple 2D and 3D environments and with real environments sensed by a humanoid robot. We also discuss computational performance of the algorithm, which is currently capable of planning short sequences of a few steps in under one second or longer sequences of 10-30 footsteps in tens of seconds to minutes on common laptop computer hardware. Our implementation is available within the Drake MATLAB toolbox [46].

3.1 Introduction

The purpose of a footstep planner is to find a list of footstep locations that a walking robot can follow safely to reach some goal. Footstep planning is a significant simplification of motion planning through contact, one in which the whole-body kinematics and dynamics are typically coarsely approximated or ignored in order to produce a tractable problem. The challenge of footstep planning thus consists of finding a path through a constrained environment to a goal while respecting constraints on the locations of and displacements

between footsteps. An example of such a plan is shown in Fig. 3-1.

Broadly speaking, there exist two families of approaches to footstep planning: discrete searches and continuous optimizations. The discrete search approaches have typically involved a precomputed action set: either represented as a set of possible displacements from one footstep to the next or a set of possible footholds in the environment. Chaining actions together forms a tree of possible footstep plans, which can be explored using existing discrete search methods like A^* and RRT. Action set approaches using pre-computed step displacements have been used by Michel [5], Baudouin [8], Chestnutt [4, 6], and Kuffner [1, 3]. Similarly, Shkolnik et al. used a precomputed set of dynamic motions, rather than foot displacements, and an RRT search to find motions for a quadruped [12]. The fixed foothold sets have been used by Bretl for climbing robots [14] and by Neuhaus for the LittleDog quadruped [7]. These approaches can easily handle obstacle avoidance by pruning the tree of actions when a particular action would put a foot in collision with an obstacle [1, 3, 8], including obstacle avoidance in the cost function evaluated at each leaf of the tree [4, 5], or adapting the set of actions when a collision is detected [6]. However, they have also tended to suffer from the tradeoff between a small action set, which reduces the branching factor of the search tree, and a large action set, which covers a larger set of the true space of foot displacements but is much harder to search [8]. In addition, applying A^* or other informed search methods to our problem is complicated by the difficulty in defining a good heuristic for partial footstep plans: we cannot generally know how many additional footsteps a partial plan will need in order to reach the goal without actually searching for those steps [4].

The continuous optimization approaches, on the other hand, operate directly on the poses of the footsteps as continuous decision variables. This avoids the restriction to a small set of fixed actions and thus allows more possible footstep plans to be explored, but correctly handling rotation and obstacle avoidance turns out to be difficult in a continuous optimization. Both footstep rotation and obstacle avoidance generally require non-convex constraints to enforce them, since the set of rotation matrices and the set of points outside a closed obstacle

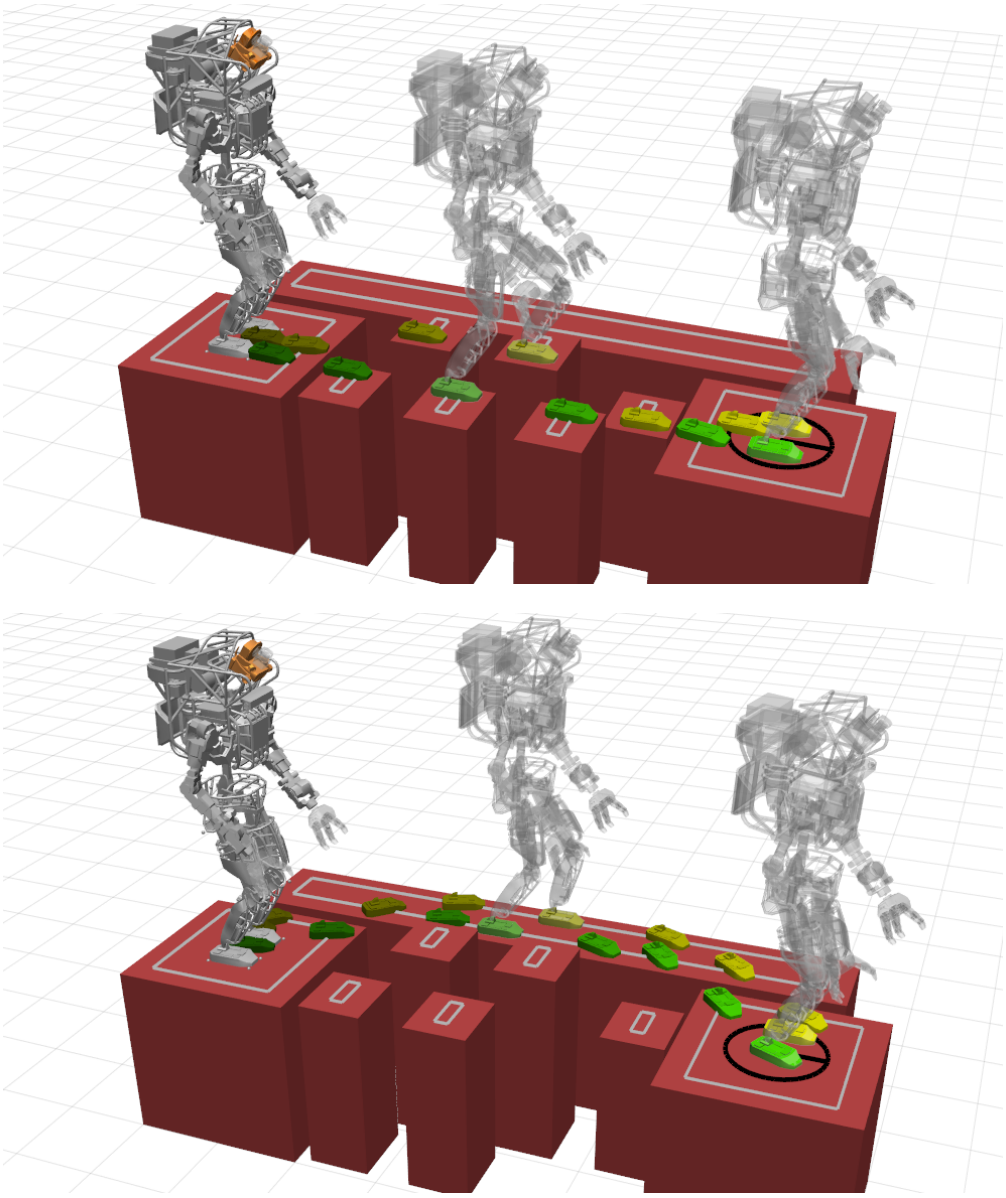


Figure 3-1: Two examples of the output of our MIQCQP footstep planner. Above: An Atlas biped planning footsteps across a set of stepping stones. Below: With one stepping stone removed, Atlas must take the longer detour around the stepping stones. The gray rectangles are the boundaries of convex regions of obstacle-free configuration space generated by IRIS [28] projected into the xy plane.

are non-convex. We typically cannot find guarantees of completeness or global optimality for such non-convex problems [37]. We have presented a non-convex continuous optimization for footstep planning, used by Team MIT during the DARPA Robotics Challenge 2013 Trials [25], but this optimization could not guarantee optimality of its solutions or find paths around obstacles. Alternatively, footstep rotation and obstacle avoidance can simply be ignored: Herdt et al. fix the footstep orientations and do not consider obstacle avoidance. This allows them to form a single quadratic program (QP) which can choose optimal footstep placements and control actions for a walking robot model [15].

We choose to use a mixed-integer convex program (specifically, a mixed-integer quadratically constrained quadratic program) to provide a more capable continuous footstep planner. Such a program allows us to perform a continuous optimization of the footstep placements, while using integer variables to absorb any non-convex constraints. We handle orientation of the footstep placements by approximating the trigonometric sin and cos functions with piecewise linear functions, using a set of integer variables to choose the appropriate approximation. We also avoid the non-convex constraints inherent in avoiding obstacles by instead enumerating a set of convex obstacle-free configuration space regions and using additional integer variables to assign footsteps to those regions. The presence of integer constraints significantly complicates our formulation, but a wide variety of commercial and free tools for mixed-integer convex programming exist, all of which can provide globally optimal solutions or proofs of infeasibility, as appropriate [16, 40, 47, 48]. Thus, we can solve an entire footstep planning problem to optimality while ensuring obstacle avoidance, a task that, to our knowledge, has not been accomplished before.

This is not unlike the work of Richards et al., who constructed a mixed-integer linear program to plan UAV trajectories while avoiding obstacles [26]. They represented each (convex, 2D) obstacle as a set of linear constraints, each of which generates a pair of half-spaces, one containing the obstacle and one not. They assigned a binary integer variable to every pair of half-spaces and required that, for every obstacle, the vehicle's location must be in at least one of the non-obstacle half-spaces. Instead of adding binary variables for

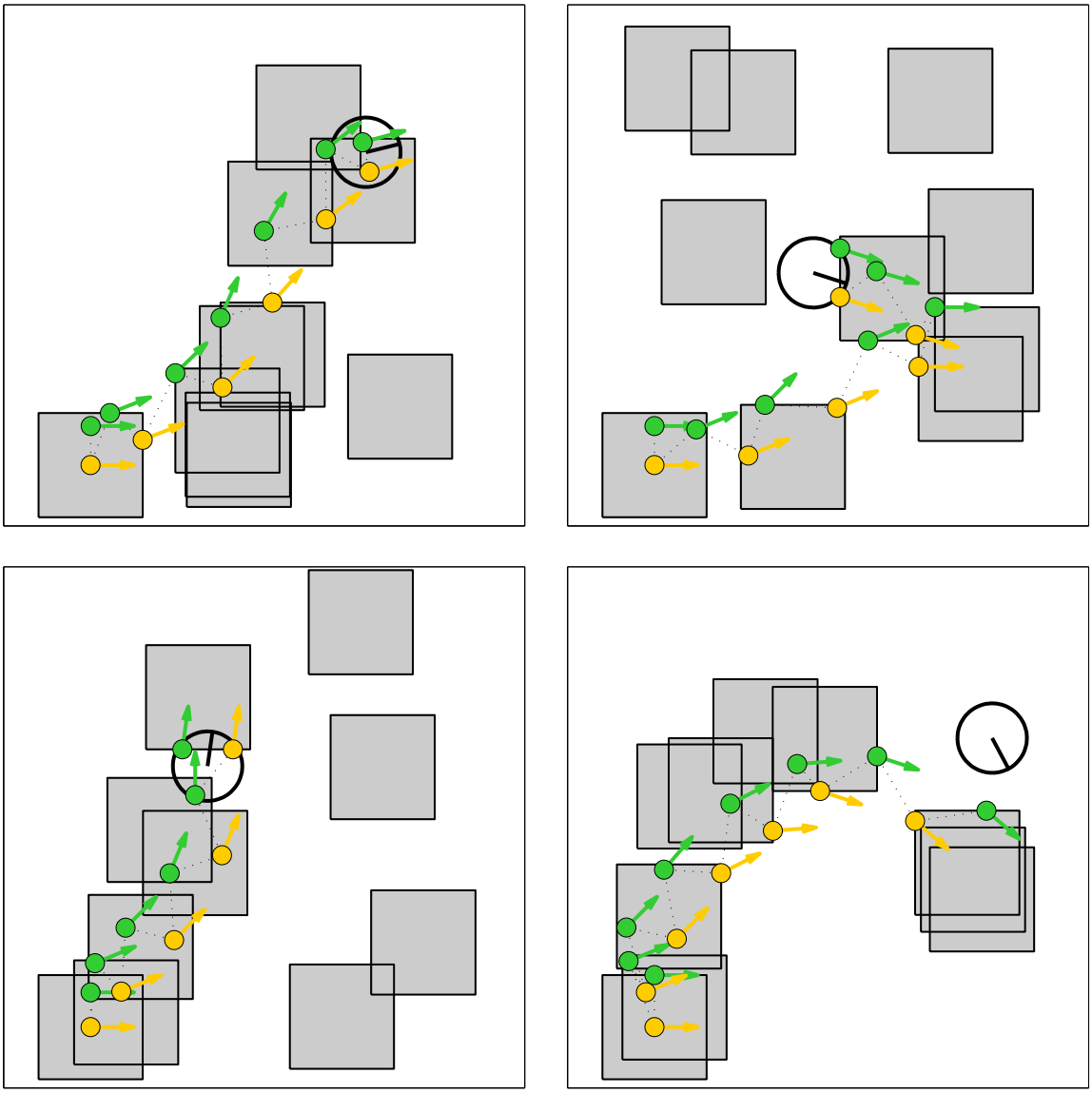


Figure 3-2: Four randomly generated 2D environments demonstrating the MIQCQP footstep planner. The gray squares are the randomly placed regions of safe terrain; the black circle is the goal location, with a line indicating the desired orientation of the robot; and the green and yellow markers are the locations of the right and left footsteps, respectively, with arrows showing the orientation of each step. The foot is assumed to be a point for these examples.

every single face of every obstacle, we precompute several convex obstacle-free regions, then assign a single binary variable to each region and require that every footstep is assigned to one such region, which dramatically reduces the number of binary variables required. Our ability to generate these convex obstacle-free regions relies on our recent development of IRIS, an algorithm for computing large convex regions of obstacle-free space [28].

3.2 Technical Approach

Our task is to determine the precise x, y, z and θ (yaw) positions of N footsteps, subject to the constraints that

1. Each step does not intersect any obstacle
2. Each step is within some convex reachable region relative to the position of the prior step

To accomplish this, we invert the non-convex problem of *avoiding* every obstacle and instead reformulate the problem as one of assigning each footstep to some pre-computed convex region of obstacle-free terrain. We then add quadratic constraints to ensure that each footstep is reachable from the prior step, using a piecewise linear approximation of sin and cos to handle step rotation.

3.2.1 Assigning Steps to Obstacle-Free Regions

Our first task is to decompose the 3D environment into a set of convex regions in which the foot can safely be placed. The IRIS algorithm, first presented in [28] and designed for this particular task, quickly generates obstacle-free convex regions in the x, y, and θ (yaw) configuration space of the robot's feet. Each of these obstacle free terrain regions is represented as a set of linear constraints defining a polytope in x, y, θ. Additionally, for each region we fit a plane in x, y, and z to the local terrain and add additional linear constraints to force footsteps in that region to lie on the plane. We label the total number of convex

regions as R and for each region r we identify the associated linear constraints with the matrix A_r and vector b_r such that if footstep f_j is assigned to region r, then

$$A_r f_j \leq b_r$$

where

$$f_j \equiv \begin{bmatrix} x_j \\ y_j \\ z_j \\ \theta_j \end{bmatrix} \text{ and } r \in \{1, \ldots, R\}$$

To describe the assignment of footsteps to safe regions, we construct a matrix of binary variables $H \in \{0,1\}^{R \times N}$, such that if $H_{r,j} = 1$ then footstep j is assigned to region r:

$$H_{r,j} \implies A_r f_j \leq b_r \tag{3.1}$$

$$\sum_{r=1}^{R} H_{r,j} = 1 \quad \forall j = 1, \ldots, N \tag{3.2}$$

The *implies* operator in (3.1) can be converted to a linear constraint using a standard big-M formulation [49] or handled directly by mixed-integer programming solvers such as IBM ILOG CPLEX [47]. The constraint (3.2) requires that every footstep be assigned to exactly one safe terrain region.

3.2.2 Ensuring Reachability

We choose to approximate the reachable set of footstep positions as an intersection of circular regions in the xy plane, with additional linear constraints on footstep displacements in yaw and z. The reachable set defined by the intersection of two circular regions is shown in Fig. 3-4. Other approaches have typically used polytope representations of the reachable set [25, 15], but such an approach results in a non-convex constraint when rotation is allowed,

even under our piecewise linear approximation of sin and cos.

Setting aside the question of footstep rotation for the moment, we can describe our reachable region with a set of convex quadratic constraints. For each footstep j, we require that

$$\left\| \begin{bmatrix} x_j \\ y_j \end{bmatrix} - \left(\begin{bmatrix} x_{j-1} \\ y_{j-1} \end{bmatrix} + \begin{bmatrix} \cos\theta_{j-1} & -\sin\theta_{j-1} \\ \sin\theta_{j-1} & \cos\theta_{j-1} \end{bmatrix} p_1 \right) \right\| <= d_1 \tag{3.3}$$

$$\left\| \begin{bmatrix} x_j \\ y_j \end{bmatrix} - \left(\begin{bmatrix} x_{j-1} \\ y_{j-1} \end{bmatrix} + \begin{bmatrix} \cos\theta_{j-1} & -\sin\theta_{j-1} \\ \sin\theta_{j-1} & \cos\theta_{j-1} \end{bmatrix} p_2 \right) \right\| <= d_2 \tag{3.4}$$

where p_1, p_2 are the centers of the circles, expressed in the frame of footstep $j-1$ and d_1, d_2 are their radii. When θ_{j-1} is fixed, constraints (3.3) and (3.4) are convex quadratic constraints, but including θ as a decision variable makes the constraint non-convex. We will handle this problem by introducing two new variables for every footstep: s_j and c_j, which will approximate $\sin\theta_j$ and $\cos\theta_j$, respectively. Constraints (3.3, 3.4) thus become:

$$\left\| \begin{bmatrix} x_j \\ y_j \end{bmatrix} - \left(\begin{bmatrix} x_{j-1} \\ y_{j-1} \end{bmatrix} + \begin{bmatrix} c_j & -s_j \\ s_j & c_j \end{bmatrix} p_1 \right) \right\| <= d_1 \tag{3.5}$$

$$\left\| \begin{bmatrix} x_j \\ y_j \end{bmatrix} - \left(\begin{bmatrix} x_{j-1} \\ y_{j-1} \end{bmatrix} + \begin{bmatrix} c_j & -s_j \\ s_j & c_j \end{bmatrix} p_2 \right) \right\| <= d_2. \tag{3.6}$$

Since p_1, p_2, d_1, d_2 are fixed, this is still a convex quadratic constraint.

Our work is not yet complete, however, since we now must enforce that s_j and c_j approximate sin and cos without introducing non-convex trigonometric constraints. We choose instead to create a simple piecewise linear approximation of sin and cos and a set of binary variables to determine which piece of the approximation to use. We construct binary matrix $S \in \{0,1\}^{L \times N}$, where L is the number of piecewise linear segments and add constraints of

the form

$$S_{\ell,j} \implies \begin{cases} \phi_\ell \leq \theta_j \leq \phi_{\ell+1} \\ s_j = g_\ell \theta_j + h_\ell \end{cases} \tag{3.7}$$

$$\sum_{\ell=1}^{L} S_{\ell,j} = 1 \quad \forall j = 1, \ldots, N \tag{3.8}$$

where g_ℓ and h_ℓ are the slope and intercept of the linear approximation of $\sin\theta$ between ϕ_ℓ and $\phi_{\ell+1}$. We likewise add piecewise linear constraints of the same form for c_j.

The particular choice of g_ℓ and h_ℓ turns out to be quite important: we must ensure that our approximation never overestimates the reachable space of foot placements. We can verify this empirically for the approximation shown in Fig. 3-3 by checking that the intersection of constraints (3.5, 3.6) is contained within the intersection of constraints (3.3, 3.4) for all values of θ. The reachable sets for footsteps at a variety of orientations are shown in Fig. 3-5.

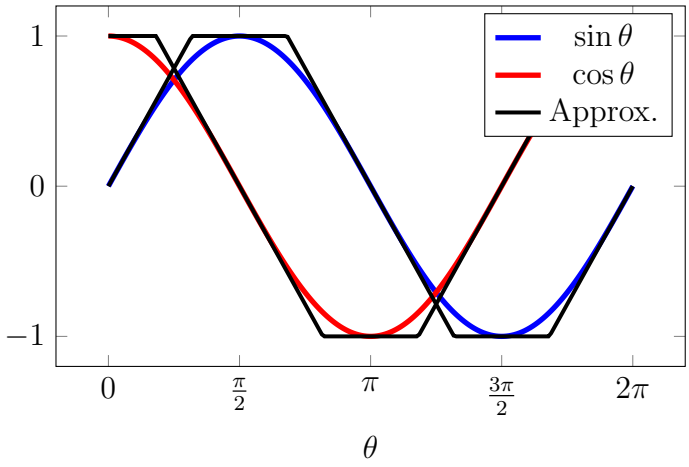


Figure 3-3: Piecewise linear approximation of sine and cosine

3.2.3 Determining the Total Number of Footsteps

In general we cannot expect to know *a priori* the total number of footsteps which must be taken to bring the robot to a goal pose, so we need some method for determining N efficiently.

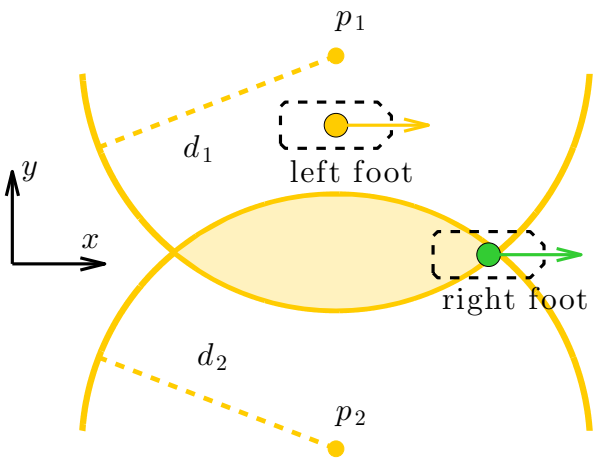


Figure 3-4: Approximation of the reachable set of locations for the right foot, given the position of the left foot. The gold arrow shows the position and orientation of the left foot, viewed from above. The shaded region shows the set of reachable poses for the right foot in the xy plane, defined as the intersection of constraints (3.5, 3.6) at orientation of $\theta_j = 0$, for which our approximation of sin and cos is exact. One possible future pose of the right foot is shown for reference.

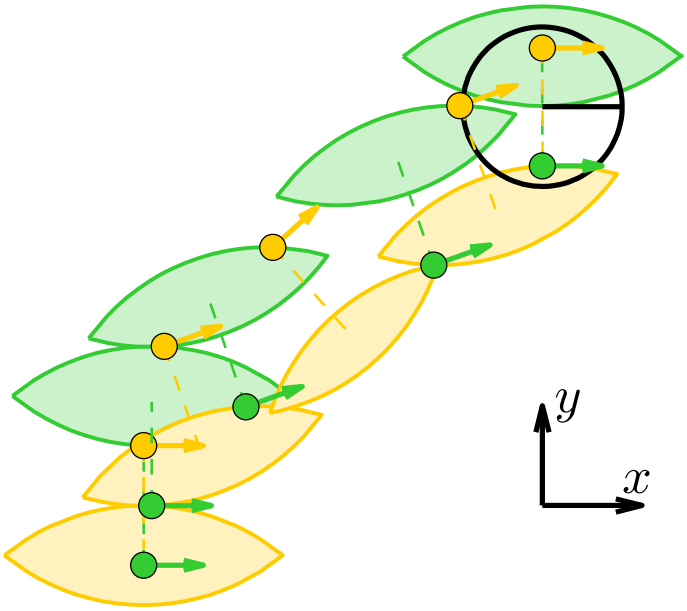


Figure 3-5: A simple footstep plan with 2D reachable regions shown. The goal is the black circle in the top right, and each arrow shows the position and orientation of one footstep. For each step, we draw the shaded region defined by constraints (3.5, 3.6) into which the next step must be placed. Note that the feasible region shrinks when the step orientation is not a multiple of $\pi/2$, as our approximation of sin and cos becomes inexact.

We could certainly just repeat the optimization for different values of N, performing a binary search to find the minimum acceptable number of steps, but for efficiency's sake we would prefer to avoid the many runs of the optimizer that this would require.

We might attempt to determine the number of required steps by setting N sufficiently large and simply adding a cost on the squared distance from each footstep to the goal pose, which will stretch our footstep plan towards the goal. If the footsteps reach the goal before N steps have been taken, then we can trim off any additional steps at the end of the plan. However, this approach allows the footstep planner to produce strides of the maximum allowable length for every footstep, even on obstacle-free flat terrain. During experiments leading up to the DARPA Robotics Challenge trials, we determined that a forward stride of 40 cm was achievable on the Atlas biped, but that a nominal stride of approximately 20 cm was safer and more stable. We would thus like to express in our optimization a preference for a particular nominal stride length while still allowing occasional longer strides needed to

cross gaps or clear obstacles. We can try to create this result by adding additional quadratic costs on the relative displacement between footsteps, but this requires very careful tuning of the weights of the distance-to-goal cost and the relative step cost for each individual step in order to ensure that the costs balance precisely at the nominal step length for each step.

Instead, we choose a much simpler cost function, with a quadratic cost on the distance from the last footstep to the goal and identical cost weights on the displacement from each footstep to the next. To control the number of footsteps used in the plan, and thus the length of each stride, we add a single binary variable to each footstep, which we will label as t_j (for 'trim'). If t_j is true, then we require that step j be fixed to the initial position of that same foot:

$$t_j \implies f_j = \begin{cases} f_1 & \text{if } j \text{ is odd} \\ f_2 & \text{if } j \text{ is even.} \end{cases} \tag{3.9}$$

Note that f_1 and f_2 are the *fixed* current positions of the robot's two feet.

Since each footstep for which $t_j = 1$ is fixed to the current position of the robot's feet, the number of footsteps which are actually used to move the robot to the goal is $N - \sum_{j=1}^{N} t_j$. By assigning a negative cost value to each binary variable t_j, we can create an incentive to reduce the number of footsteps used in the plan. To tune the nominal stride length, we can simply adjust the cost assigned to the t_j. Increasing the magnitude of this cost will lengthen the nominal stride uniformly, and decreasing the magnitude will shorten the stride. Thus, we have a single value to tune in order to set the desired stride length, while still allowing strides which exceed this length. After the optimization is complete, we can remove any footsteps at the beginning of the plan for which t_j is true.

3.2.4 Complete Formulation

Putting all of the pieces together gives us the entire footstep planning problem:

$$\underset{f_1,\ldots,f_j,S,C,H,t_1,\ldots,t_j}{\text{minimize}} (f_N - g)^\top Q_g (f_N - g) + \sum_{j=1}^{N} q_t t_j + \sum_{j=1}^{N-1} (f_{j+1} - f_j)^\top Q_r (f_{j+1} - f_j))$$

subject to, for $j = 1, \ldots, N$

$$H_{r,j} \implies A_r f_j \leq b_r \qquad r = 1, \ldots, R \quad \text{(safe terrain regions)}$$

$$S_{\ell,j} \implies \begin{cases} \phi_\ell \leq \theta_j \leq \phi_{\ell+1} \\ s_j = g_\ell \theta_j + h_\ell \end{cases} \qquad \ell = 1, \ldots, L \quad \text{(piecewise linear } \sin\theta\text{)}$$

$$C_{\ell,j} \implies \begin{cases} \phi_\ell \leq \theta_j \leq \phi_{\ell+1} \\ c_j = g_\ell \theta_j + h_\ell \end{cases} \qquad \ell = 1, \ldots, L \quad \text{(piecewise linear } \cos\theta\text{)}$$

$$\left\| \begin{bmatrix} x_j \\ y_j \end{bmatrix} - \left(\begin{bmatrix} x_{j-1} \\ y_{j-1} \end{bmatrix} + \begin{bmatrix} c_j & -s_j \\ s_j & c_j \end{bmatrix} p_i \right) \right\| <= d_i \quad i = 1, 2 \qquad \text{(approximate reachability)}$$

$$t_j \implies f_j = \begin{cases} f_1 & \text{if } j \text{ is odd} \\ f_2 & \text{if } j \text{ is even.} \end{cases} \qquad \text{(fix unused steps to initial pose)}$$

$$\sum_{r=1}^{R} H_{r,j} = \sum_{\ell=1}^{L} S_{\ell,j} = \sum_{\ell=1}^{L} C_{\ell,j} = 1$$

$$H_{r,j}, S_{\ell,j}, C_{\ell,j}, t_j \in \{0, 1\}$$

$$f_{\min} \leq f_j \leq f_{\max}$$

$$\Delta f_{\min} \leq (f_j - f_{j-1}) \leq \Delta f_{\max}$$

where $g \in \mathbb{R}^4$ is the x, y, z, θ goal pose, $Q_g \in \mathbb{S}^4_+$ and $Q_r \in \mathbb{S}^4_+$ are objective weights on the distance to the goal and between steps, $q_t \in \mathbb{R}$ is an objective weight on trimming unused steps, and $f_{\min}, f_{\max}, \Delta f_{\min}, \Delta f_{\max} \in \mathbb{R}^4$ are bounds on the absolute footstep positions and their differences, respectively. We also fix f_1 and f_2 to the initial poses of the robot's feet.

3.2.5 Solving the Problem

We have implemented this approach in MATLAB [29], using the commercial solver Gurobi [16] to solve the MIQCQP itself. Typical problems involving 10 to 20 footsteps and 10 convex safe terrain regions can be solved in a few seconds to one minute on a Lenovo laptop with an Intel i7 clocked at 2.9 GHz. Smaller footstep plans involving just a few steps can be solved in well under one second on the same hardware, so this method is capable of providing short-horizon footstep plans at realtime rates while walking or longer footstep plans involving complex path planning while stationary. The Mosek and CPLEX optimizers [40, 47] were also capable of solving the problem, but we generally found that Gurobi found optimal solutions or proofs of infeasibility more quickly in our experiments.

3.3 Results

To demonstrate the MIQCQP footstep planning algorithm, we first generated a collection of random 2D environments. Each environment consisted of 10 square regions of safe terrain, 9 of which were uniformly randomly placed within the bounds of the environment, and 1 of which was placed directly under the starting location of the robot to represent the robot's currently occupied terrain. A goal pose was uniformly randomly placed within the xy bounds of the environment with a desired orientation between $\pm\frac{\pi}{2}$ relative to the robot's starting orientation. Several such environments and the resulting footstep plans are shown in Fig. 3-2. All the footstep plans shown in this chapter are the result of convergence to within 0.1% or less of the globally optimal cost value, as reported by Gurobi.

Next, we manually generated several 3D example environments using Drake, a software toolbox for planning and control [46]. These environments and the resulting footstep plans are shown in Figs. 3-1, 3-6, 3-8. The IRIS algorithm [28] was used to generate convex regions in the configuration space of a very simple box model of the entire robot, shown in Fig. 3-7, in order to avoid collisions between the upper body and the environment. This approach is sufficient to generate rich behaviors such as turning sideways to move through a narrow gap,

as in Fig. 3-8. Currently, we only ensure that each footstep admits a collision-free posture of the robot, but we do not account for collisions during the transitions between those postures; we will discuss possible ways to address this in Sect. 3.4.

We have also demonstrated the MIQCQP planner on real terrain, using sensor data collected by Atlas. To generate this plan, we captured LIDAR scans of a stack of cinderblocks like those encountered in the DRC Trials and constructed a heightmap of the scene using the perception tools developed by Team MIT for the DRC [25]. A Sobel filter was used to classify areas of the terrain which were steeper than a predefined threshold [50], and these steep areas were represented as obstacles for the footstep planner. We used the IRIS algorithm [28] to generate seven convex safe terrain regions which covered the terrain of interest in front of the robot. Several footstep plans to different goal poses using these regions are shown in Fig. 3-9.

3.4 Conclusion and Future Work

We have demonstrated a novel footstep planning approach, which replaces nonlinear, non-convex constraints with mixed-integer convex constraints. This allows us to solve footstep planning problems to their global optimum, within our linear approximation of rotation. We have inverted the problem of avoiding obstacles into a problem of assigning steps to known convex safe regions, which we can construct efficiently. The primary advantage of our MIQCQP footstep planner is its ability to generate rich footstep sequences in difficult terrain with guarantees of completeness and global optimality. We do not rely on sampling or fixed action sets, which may miss small regions of safe terrain entirely. Our approach also handles terrain of varying height gracefully, as the same object can be treated as both an obstacle and a walking surface when appropriate, as in Fig. 3-6.

On the other hand, we are entirely reliant on the existence of sufficiently many convex obstacle-free regions to ensure that a path through the environment can be found. The IRIS algorithm generates these regions efficiently, but currently requires user input to seed the position of each region. This is a mixed blessing: by seeding such a region, the human

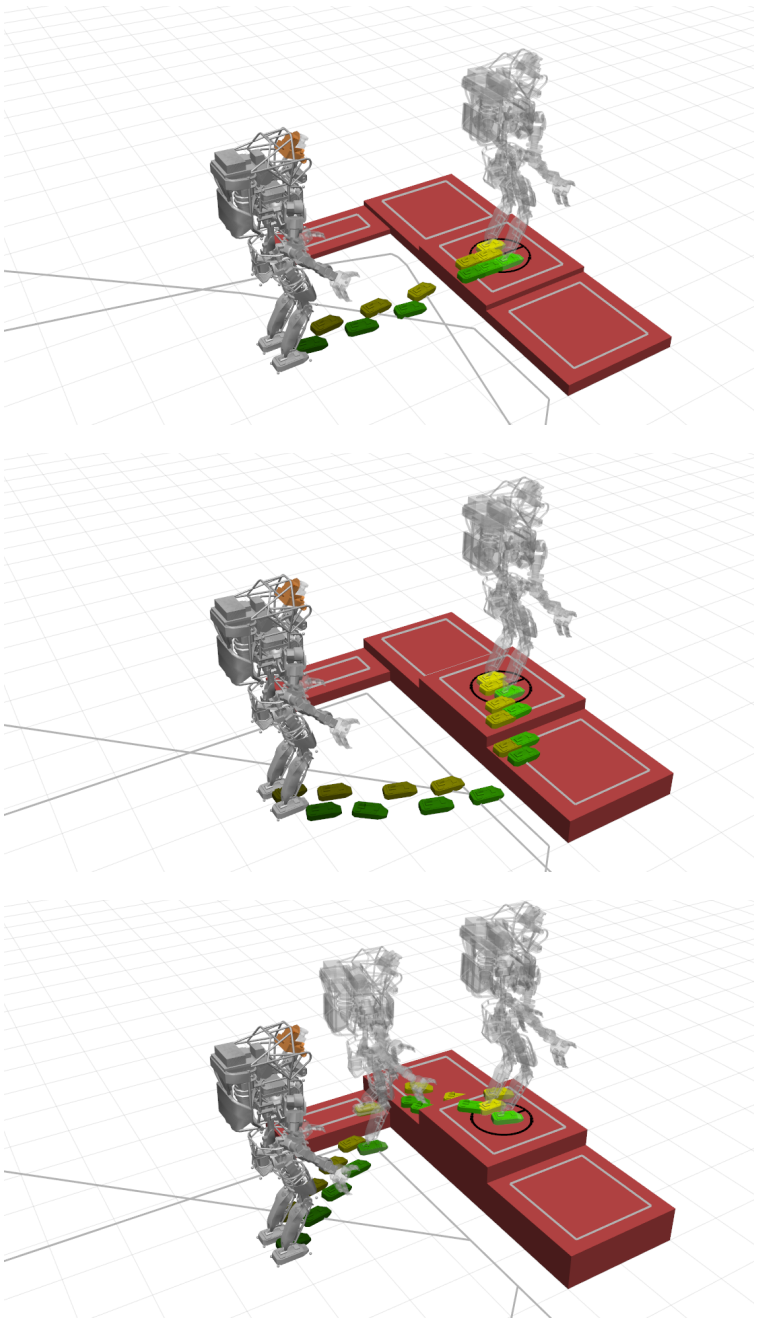


Figure 3-6: Three similar environments, with footstep plans for each. Top: Atlas can mount the center pedestal in one stride. Middle: Raising the center pedestal to twice Atlas' maximum vertical stride forces the robot to detour to the right. Bottom: Raising the pedestals even further requires Atlas to use three platforms to get to the required height. Gray lines are the boundaries of the convex regions of obstacle-free configuration space, generated by IRIS.

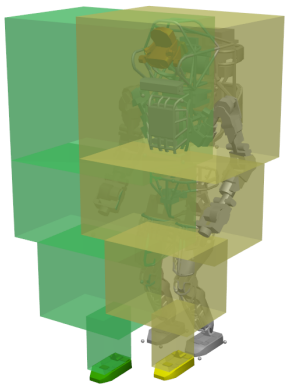


Figure 3-7: Simplified bounding box model of the robot used to plan footstep locations that will be collision-free for the entire robot.

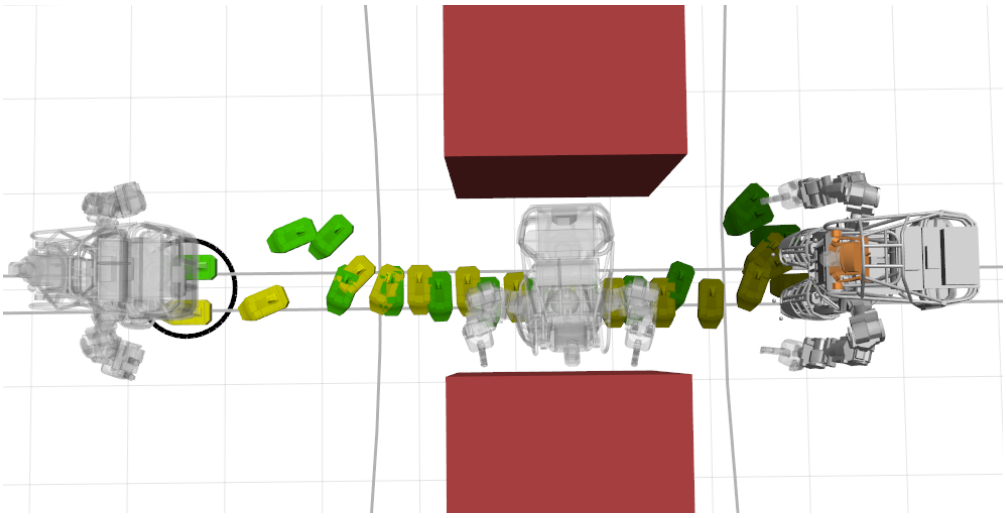


Figure 3-8: A footstep plan through a narrow gap, for which Atlas must turn sideways.

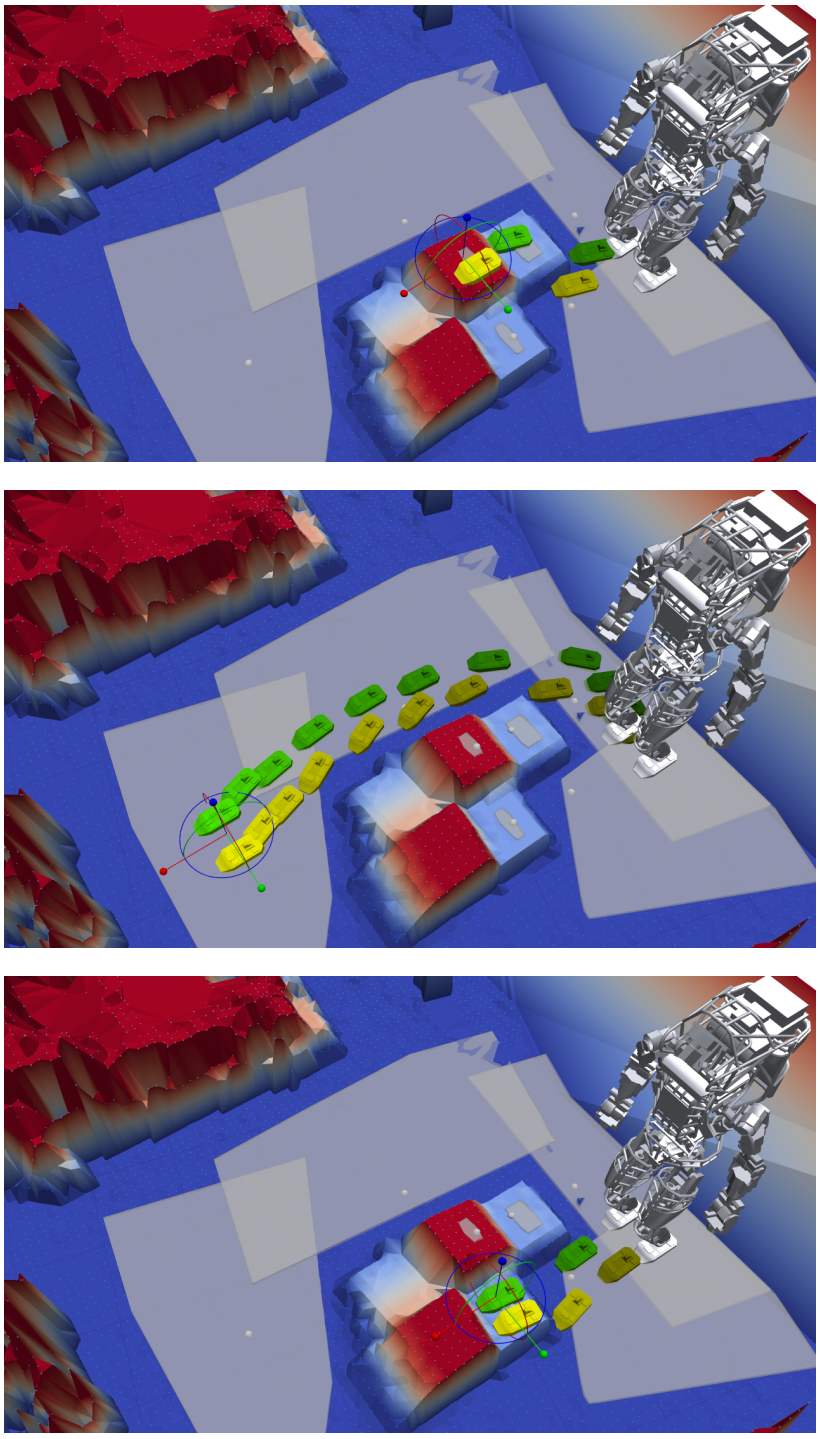


Figure 3-9: Footstep plans for to navigate over and around a set of cinderblocks, using data sensed by the Atlas humanoid. Gray shaded areas are the obstacle-free regions of configuration-space, into which the center of each footstep must be placed.

operator provides valuable input about the possible walking surfaces for the robot, but this requirement clearly sacrifices autonomy of the robot. We are currently investigating ways to fully automate the generation of safe terrain regions. In addition, we require that each safe region represent a planar area of terrain: non-planar terrain regions introduce non-convex constraints in our formulation and cannot be allowed.

In the future, we intend to improve the speed with which we can generate footstep plans, since waiting up to a minute before the robot can start moving may be frustrating in a real-world scenario. We may be able to do this by relaxing the reachability criteria for footsteps which are relatively far in the future. For example, we might consider the reachability and orientation of the first 10 footsteps in a plan, but only ensure that the next 10 or 20 footsteps can be assigned to safe terrain regions without the distance between them becoming too large. This will significantly reduce the complexity of the MIQCQP, at the cost of some likelihood of generating plans through areas that appear feasible under the simplified reachability criteria but through which the robot cannot actually travel.

In addition, we are interested in combining this work with the research on dynamic walking planning currently underway in the Robot Locomotion Group at MIT. Our colleagues intend to use the MIQCQP to produce initial positions and orientations for the footsteps, along with their assignments to safe terrain regions, then run an additional optimization to plan the contact forces on the feet, the center of mass trajectory, and the complete collision-free motions of the robot's limbs. The assignments to convex safe regions produced by the MIQCQP should be a valuable set of convex constraints for this later optimization.

Source Code

Our experimental implementation of the MIQCQP footstep planner in MATLAB is available within Drake, located at `https://github.com/RobotLocomotion/drake`.

Chapter 4

Future Work

4.1 Dynamic Walking Planning

The mixed-integer convex optimization presented here offers significant advantages over existing footstep planners, but it still does not provide any guarantees of dynamic feasibility, even for a simplified model of the robot. We must find the forces between the robot and the environment, the motion of the robot's mass, and perhaps the motion of each individual joint in order to ensure that there is a physically realizable set of actions that will bring the robot to its goal. We can do so either as part of the footstep planning optimization or as a separate optimization; in either case, the mixed-integer footstep planner can be very helpful in finding dynamic walking motions.

4.1.1 Option 1: Including Dynamics in the Footstep Planner

As mentioned previously, Herdt proposes a quadratic program which can choose footstep placements, the center of pressure, and the trajectory of the center of mass of the robot in order to find a walking plan which is feasible with respect to a simplified dynamic model of the robot. Herdt assumes that the position of the center of pressure (CoP) and the third derivative (jerk) of the robot's center of mass (CoM) position are piecewise constant. If angular momentum is neglected, this allows him to represent the discrete dynamics of the CoP and the CoM trajectory as a set of linear constraints. Additional linear constraints on the displacement between the footsteps ensure an approximation of kinematic reachability, provided the orientations of the feet are fixed and chosen beforehand. Finally, he adds a quadratic objective to minimize the jerk of the CoM and encourage the CoP to track a reference trajectory. The result is a single quadratic program which can choose footstep placements while ensuring an approximation of dynamic stability.

This is quite compatible with the footstep planner described in Chapter 3. We can add additional variables to our formulation corresponding to the positions of the CoM and CoP (and their respective derivatives) and add linear constraints to enforce the discretized dynamics. At the cost of increasing the already substantial computation needed to plan the footstep locations, we can receive a plan for the trajectory of the CoM and CoP of the

robot which will be mutually consistent under the discretization of the robot's dynamics. Of course, this does not guarantee that the actual robot can execute the walking plan: joint limits, actuator limits, self-collisions, and angular momentum effects can all result in failure of the walking plan. It is unlikely that we can include all of these constraints in our mixed-integer convex optimization, so a second optimization may be necessary.

4.1.2 Option 2: Nonlinear Optimization for Dynamic Feasibility

Andres Valenzuela and Hongkai Dai in the Robot Locomotion Group at MIT are currently developing a nonlinear optimization for whole-body motion planning. Their approach encompasses decisions about the location and magnitude of the contact forces between the robot and its environment, the kinematics of the entire robot, and the avoidance of collisions with the environment and with the robot itself. While this might ordinarily be an overwhelming problem, an efficient implementation of these constraints and a focus on exploiting the sparsity of the problem has allowed them to plan walking, running, jumping, and climbing motions for Atlas. We plan to combine the mixed-integer footstep planner with this approach in order to create a complete system for motion planning. Our proposed approach involves several steps:

1. Automatically, or with user input, we use IRIS to generate one or more convex regions of safe terrain for the robot's feet

2. The MIQCQP footstep planner chooses the number of footsteps needed to reach the goal, nominal positions of those footsteps, and their assignments to the safe terrain regions.

3. The nonlinear optimization attempts to find a feasible dynamic motion of the robot, using the results of the MIQCQP as its initial guess for the footstep locations. The footsteps are not required to remain in exactly the positions found by the MIQCQP, but the assignment of footsteps to convex safe regions provides bounds on their positions and ensures that the feet do not intersect any region of unsafe terrain.

In this case, the MIQCQP provides both a useful start for the nonlinear program and a set of constraints to ensure that every foot placement remains in safe terrain throughout the second optimization.

4.2 Running

The fundamental difference between walking and running for a biped is the existence of a flight phase, in which there is no contact between the runner and the ground. Adapting the footstep planner to the task of running should be possible, with some modifications. An important assumption inherent in the footstep planner presented here is that the position of the stance foot is the only relevant state which must be tracked when planning footsteps. That is, each footstep is only directly limited by the position of its immediate predecessor, and not by any of the footsteps before that. This means that there is nothing to prevent the robot from taking a long step forward and then backward, which seems to be a reasonable assumption for walking. When running, however, the momentum of the robot must be considered (imagine trying to transition from running forward at full speed to running backward in a single step). In addition, the reachability of the robot's feet must also change as its velocity changes: a higher speed means that more distance is covered during the flight phase, so the distance between subsequent steps must increase. Both of these conditions suggest that basing the reachable set of foot positions on the pose of the prior footstep is unsuitable for running.

Instead, I propose adding additional decision variables corresponding to the expected position and velocity of the center of mass at each footstep, much as Herdt does in [15]. Instead of the prior foot pose, we can base the reachable set on the position of the center of mass at the time of each footstep. This should have the desired effect that as the robot increases its overall velocity, the center of mass moves farther between each footstep, lengthening the maximal stride in the direction of travel. We might choose to replicate Herdt's optimization of the center of mass pose in order to create a dynamically feasible motion, or we might enforce a simpler constraint on the magnitude and direction of any changes in the robot's

CoM velocity as a function of the footstep placement. With only a minor reworking of the footstep planner, we should be able to produce running or walking motions with the same mixed-integer optimization.

4.3 Quadrupedal Locomotion

The mixed-integer convex footstep planner may also be of use for robots with 4 or more legs. Comparatively little work has been done on detailed footstep planning for non-bipedal robots, but the DARPA LittleDog project prompted the development of several different approaches. Shkolnik uses a set of precomputed dynamic actions as the basis for a discrete search for a bounding motion across rough terrain [12], not unlike the fixed quasi-static action set used by Kuffner [1] and Chestnutt [4]. In a different approach, Neuhaus uses a precomputed set of possible contact positions and chooses a walking motion through only those points on the terrain. Both approaches resulted in successful plans for LittleDog across very difficult terrain, but they both also suffer from the same problems seen in discrete footstep planning approaches for bipeds: as the number of possible actions or contact locations increases, the branching factor of the search blows up. In fact, Neuhaus uses a depth-first search with carefully designed heuristics to get around the large branching factor of the search among contact locations, which sacrifices optimality of the solution. Once again, a well-designed continuous optimization, with integer variables to handle tasks like obstacle avoidance, may be able to outperform the existing approaches and find optimal quadrupedal footstep plans.

The quadruped planning problem introduces new challenges in defining the robot's reachable space, just as the running problem did. The possible location of any limb of a quadruped is restricted by the positions of all the other limbs, some of which may be in contact or out of contact depending on the gait. For example, in a diagonal trot gait, the right front and left rear limbs swing while the left front and right rear are in contact with the ground, so the two swinging legs can be freely moved so long as they respect the limits imposed by the legs which are fixed to the terrain. Such a gait is shown in Fig. 4-1. Encoding the various interrelationships between the positions of all of the legs, which depend on the particular

gait being used, is likely to be prohibitively difficult.

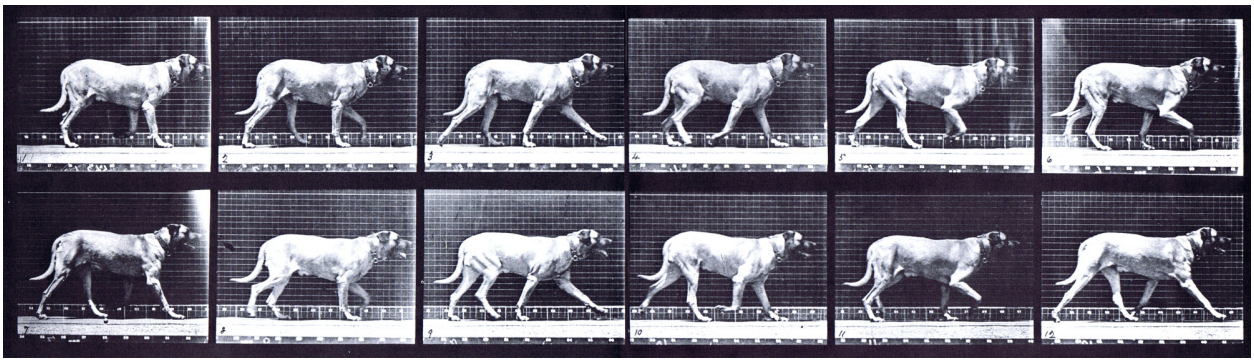


Figure 4-1: A dog demonstrating a trot gait, as photographed by Eadweard Muybridge [51].

Instead, we can try a similar approach to the one proposed for bipedal running. Rather than enforcing constraints between the positions of all of the feet, we can instead explicitly plan for the location of the quadruped's body, subject to the constraints that all of the feet be reachable from that body pose and that the feet in contact with the terrain do not move. This may require fixing the sequence of leg movements beforehand, but it is also possible that we could introduce additional integer variables to allow the optimization to vary the timing of the gait. This could be very helpful because the order with which the feet make and break contact with the terrain dramatically affects the reachable space (for example, a quadruped can lunge with both forelegs farther than it can step with one foreleg while keeping the other on the ground).

4.4 Applications beyond Walking

It may also be possible to apply the techniques developed here to tasks other than footstep planning. As discussed in Sect. 1.3.4, Richards et al. used a mixed-integer linear program to plan a discrete trajectory (represented as a sequence of positions in space) for a group of unmanned aerial vehicles. Their implementation suffered from two scaling issues, both of which may be avoidable in the future. First, by assigning a binary integer variable to every face of every obstacle, their formulation rapidly became more complex as the number

and complexity of the obstacles grew. In the worst case performance, adding a single binary variable doubles the time required to find an optimal solution, so adding many new variables can be quite costly. Second, rather than a continuous trajectory, they planned a sequence of positions separated by no more than some threshold distance, which provides no guarantee that there will be a continuous collision-free path from one position to the next. Imagine, for example, planning two sequential poses on opposite sides of a thin wall.

Both of these problems may be avoidable using the IRIS-based approach discussed here. First, if the user has some prior knowledge about which areas of the environment are likely to be useful for flying and which are not, then it should be possible to generate a relatively small number of convex safe regions with IRIS, resulting in fewer binary variables (one per IRIS region rather than one per obstacle face) and a more tractable integer program. In addition, the IRIS regions provide useful information about the adjacency of obstacle-free areas of configuration space. Before generating a plan, we can precompute an adjacency matrix among those regions, where two regions are adjacent if they share a face or have a non-empty intersection. If two regions of free configuration space are adjacent by this definition, then there must exist a continuous collision-free path between them, as discussed by Kim et al. in their work on planning with homology class constraints [52]. This condition is sufficient regardless of the size of the shared face or overlap because the basis of configuration-space planning is that in an environment of suitably inflated obstacles, the robot can be treated as a single point [45]. To ensure that there will exist a continuous free path for the robot, we can require that if one of the planned poses of the UAV is within a particular IRIS region, then the subsequent pose must be in the same region or an adjacent region. This condition can be easily expressed as a set of linear constraints on the binary variables which assign poses to convex regions.

4.5 Conclusion

The convex segmentation produced by IRIS, combined with the mixed-integer optimization, provides a capable framework for planning sequences of footsteps or poses for a walking,

running, or flying robot. The organization of the problem as an MIQCQP allows us to benefit from tremendous advances in new tools designed to solve convex problems efficiently. This approach gives optimal footstep plans (within our approximations and choice of cost function) for walking, and it shows promise for future work on a variety of robotics applications. We look forward to combining the results discussed here with the new developments in whole-body dynamic motion planning in the Robot Locomotion Group, and we hope that this will be a useful framework for future robot locomotion.

Bibliography

[1] James J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Footstep planning among obstacles for biped robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 500–505, Maui, Hawaii, 2001.

[2] O. Lorch, A. Albert, J. Denk, M. Gerecke, R. Cupec, J.F. Seara, W. Gerth, and G. Schmidt. Experiments in vision-guided biped walking. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2002*, volume 3, pages 2484–2490 vol.3, 2002.

[3] James J. Kuffner, Koichi Nishiwaki, Satoshi Kagami, Masayuki Inaba, and Hirochika Inoue. Online footstep planning for humanoid robots. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 932–937, 2003.

[4] Joel E. Chestnutt, James Kuffner, Koichi Nishiwaki, and Satoshi Kagami. Planning biped navigation strategies in complex environments. In *IEEE-RAS International Conference on Humanoid Robots*, Karlsruhe, Germany, 2003.

[5] Philipp Michel, Joel Chestnutt, James Kuffner, and Takeo Kanade. Vision-guided humanoid footstep planning for dynamic environments. *IEEE-RAS International Conference on Humanoid Robots*, pages 13–18, 2005.

[6] Joel E. Chestnutt, Koichi Nishiwaki, James Kuffner, and Satoshi Kagami. An adaptive action model for legged navigation planning. In *IEEE-RAS International Conference on Humanoid Robots*, page 196202, 2007.

[7] Peter D. Neuhaus, Jerry E. Pratt, and Matthew J. Johnson. Comprehensive summary of the institute for human and machine cognitions experience with LittleDog. *The International Journal of Robotics Research*, 30(2):216–235, February 2011.

[8] Lo Baudouin, Nicolas Perrin, Thomas Moulard, Florent Lamiraux, Olivier Stasse, and Eiichi Yoshida. Real-time replanning using 3d environment for humanoid robot. In *IEEE-RAS International Conference on Humanoid Robots*, pages p.584–589, Bled, Slovnie, 2011.

[9] Nicolas Perrin, Olivier Stasse, Florent Lamiraux, Young J Kim, and Dinesh Manocha. Real-time footstep planning for humanoid robots among 3d obstacles using a hybrid bounding box. *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 977–982, 2012.

[10] Weiwei Huang, Junggon Kim, and C.G. Atkeson. Energy-based optimal step planning for humanoids. In *2013 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3124–3129, Karlsruhe, Germany, May 2013.

[11] DARPA. Atlas front view, 2013.

[12] A Shkolnik, M Levashov, I R Manchester, and R Tedrake. Bounding on rough terrain with the LittleDog robot. *The International Journal of Robotics Research*, 30(2):192–215, 2011.

[13] K. Hauser, T. Bretl, and J.-C. Latombe. Non-gaited humanoid locomotion planning. In *2005 5th IEEE-RAS International Conference on Humanoid Robots*, pages 7–12, December 2005.

[14] Tim Bretl, Sanjay Lall, Jean-Claude Latombe, and Stephen Rock. Multi-step motion planning for free-climbing robots. In Michael Erdmann, Mark Overmars, David Hsu, and Frank van der Stappen, editors, *Algorithmic Foundations of Robotics VI*, number 17 in Springer Tracts in Advanced Robotics, pages 59–74. Springer Berlin Heidelberg, January 2005.

[15] Andrei Herdt, Holger Diedam, Pierre-Brice Wieber, Dimitar Dimitrov, Katja Mombaur, and Moritz Diehl. Online walking motion generation with automatic foot step placement. *Advanced Robotics*, 24(5-6):719–737, 2010.

[16] Gurobi Optimization, Inc. Gurobi optimizer reference manual, 2014.

[17] Richard M. Karp. Reducibility among combinatorial problems. In Michael Jnger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 219–241. Springer Berlin Heidelberg, 2010.

[18] Stephan J. Eidenbenz and Peter Widmayer. An approximation algorithm for minimum convex cover with logarithmic performance guarantee. *SIAM Journal on Computing*, 32(3):654–670, 2003.

[19] Hou-Yuan F. Feng and Theodosios Pavlidis. Decomposition of polygons into simpler components: Feature generation for syntactic pattern recognition. *Computers, IEEE Transactions on*, 100(6):636–650, 1975.

[20] J. Mark Keil. Decomposing a polygon into simpler components. *SIAM Journal on Computing*, 14(4):799–817, November 1985.

[21] Bernard Chazelle and David Dobkin. Decomposing a polygon into its convex parts. *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 38–48, 1979.

[22] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polygons. *Proceedings of the twentieth annual symposium on Computational geometry*, pages 17–26, 2004.

[23] K. Mamou and F. Ghorbel. A simple and efficient approach for 3d mesh approximate convex decomposition. In *2009 16th IEEE International Conference on Image Processing (ICIP)*, pages 3501–3504, 2009.

[24] Zhou Ren. *Improved Approximate Convex Decomposition of Polygons and Its Applications*. PhD thesis, Huazhong University of Science and Technology, 2010.

[25] Maurice Fallon, Scott Kuindersma, Sisir Karumanchi, Matthew Antone, Toby Schneider, Hongkai Dai, Claudia Perez D'Arpino, Robin Deits, Matt DiCicco, Dehann Fourie, Twan Koolen, Pat Marion, Michael Posa, Andres Valenzuela, Kuan-Ting Yu, Julie Shah, Karl Iagnemma, Russ Tedrake, and Seth Teller. An architecture for online affordance-based perception and whole-body planning. *Journal of Field Robotics*, September 2014.

[26] Arthur Richards, John Bellingham, Michael Tillerson, and Jonathan How. Coordination and control of multiple UAVs. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*. American Institute of Aeronautics and Astronautics, Monterey, CA, August 2002.

[27] Soonkyum Kim, Koushil Sreenath, Subhrajit Bhattacharya, and Vijay Kumar. Trajectory planning for systems with homotopy class constraints. In *Latest Advances in Robot Kinematics*, pages 83–90. Springer Netherlands, Dordrecht, 2012.

[28] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Workshop on the Algorithmic Foundations of Robotics*, Istanbul, Turkey, 2014.

[29] MATLAB. *version 8.2.0.701 (R2013b)*. The MathWorks Inc., Natick, MA, 2013.

[30] Andrzej Lingas. The power of non-rectilinear holes. In Mogens Nielsen and Erik Meineche Schmidt, editors, *Automata, Languages and Programming*, number 140 in Lecture Notes in Computer Science, pages 369–383. Springer Berlin Heidelberg, 1982.

[31] Hairong Liu, Wenyu Liu, and L.J. Latecki. Convex shape decomposition. In *2010 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 97–104, 2010.

[32] Douglas Demyen and Michael Buro. Efficient triangulation-based pathfinding. *AAAI*, 6:942–947, 2006.

[33] Alejandro Sarmiento, Rafael Murrieta-Cid, and Seth Hutchinson. 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*, page 34863491. IEEE, 2005.

[34] Paul Fischer. Finding maximum convex polygons. In Zoltn sik, editor, *Fundamentals of Computation Theory*, number 710 in Lecture Notes in Computer Science, pages 234–243. Springer Berlin Heidelberg, 1993.

[35] Srikanth Sastry, David S. Corti, Pablo G. Debenedetti, and Frank H. Stillinger. 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.

[36] V. A. Luchnikov, N. N. Medvedev, L. Oger, and J.-P. Troadec. Voronoi-delaunay analysis of voids in systems of nonspherical particles. *Physical review E*, 59(6):7205, 1999.

[37] Stephen P Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, Cambridge, UK; New York, 2004.

[38] Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. SIAM, 1995.

[39] Jacob Mattingley and Stephen Boyd. CVXGEN: Code generation for convex optimization, 2013.

[40] Mosek ApS. The MOSEK optimization software, 2014.

[41] Leonid G. Khachiyan and Michael J. Todd. On the complexity of approximating the maximal inscribed ellipsoid for a polytope. *Mathematical Programming*, 61(1-3):137–159, 1993.

[42] Aharon Ben-Tal and Arkadi Nemirovski. More examples of CQ-representable functions/sets. In *Lectures on Modern Convex Optimization: Analysis, Algorithms and*

Engineering Applications, MPS-SIAM Series on Optimization, pages 105–110. SIAM, Philadelphia, PA, 2001.

[43] Mosek ApS. Inner and outer lowner-john ellipsoids, 2014.

[44] Michael Grant and Stephen Boyd. CVX: Matlab software for disciplined convex programming, version 2.1, 2014.

[45] T Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, (2):108–120, 1983.

[46] Russ Tedrake. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2014.

[47] IBM Corp. User's manual for CPLEX, 2010.

[48] GNU linear programming kit.

[49] Johan Lofberg. Big-m and convex hulls, 2012.

[50] Per-Erik Danielsson and Olle Seger. Generalized and separable sobel operators. In Herbert Freeman, editor, *Machine vision for three-dimensional scenes*. Academic Press, Inc., Sand Diego, CA, 1990.

[51] Eadweard Muybridge. Dog (mastiff) trotting: an irregular stride, plate 706, 1880.

[52] Koushil Sreenath Subhrajit Bhattacharya Vijay Kumar Soonkyum Kim. Optimal trajectory generation under homology class constraints. In *51st IEEE Conference on Decision and Control*, Maui, HI, 2012.