Graph of Convex Sets for Trajectory Optimization on Spot

Johannes Ihle
Dept. Engineering Cybernetics
Norwegian University of Science and Technology
Trondheim, Norway
johannes.s.ihle@gmail.com

Aileen Liao
Dept. Mechanical Engineering
Massachusetts Institute of Technology
Cambridge, MA
ai0liao@mit.edu

Lukas Molnar
Dept. Mechanical Engineering
ETH Zurich
Zurich, Switzerland
lmolnar@ethz.ch

***Abstract*—Trajectory optimization becomes complex when there are obstacles in the environment. The Graph of Convex Sets (GCS) approach for mixed-integer convex optimization has been used for solving trajectory planning problems in drones and manipulator arms. We used GCS to plan footstep trajectories for Spot. The footstep trajectories were translated to target joint positions with inverse kinematics and whole body control. Lastly, we used PD control to command Spot. Our results showed that GCS was able to solve for footstep trajectory solutions but limitations with our current whole body control reduce the set of feasible footstep trajectories that Spot can achieve.**

I. Introduction

Trajectory optimization is a popular approach for motion planning in high-dimensional spaces under dynamic constraints. However, the optimization problem becomes much more complex when obstacles are present in the environment, due to the non-convexity they introduce. We would like to apply the Graph of Convex Sets (GCS) algorithm to the Boston Dynamics Spot® robot in order to plan footstep trajectories that avoid obstacles. Additionally, we aim to integrate whole body control for Spot, which tracks the desired footstep trajectories.

II. Prior Works

The GCS algorithm was first introduced by Marcucci et al [8]. The basic concept of GCS is to find a minimum-cost path from a source vertex to a target vertex through a graph of convex sets. Each vertex v consists of a convex set $\mathcal{X}_v$ and a point x_v within it. The cost of each edge $e := (u, v)$ between vertices u and v is given by the edge-length function $l_c(x_u, x_v)$, which is a convex and non-negative function. The problem formulation is given in Equation 1, where p is a path going from the source to the target vertex, $\mathcal{P}$ is the family of all such paths, and $\mathcal{E}_p$ is all the edges in p [7]. $\mathcal{X}_e$ represents convex constraints on the edges that make up the path p. Marcucci et al. proposed a mixed-integer convex formulation for solving motion planning problems in the presence of obstacles [7]. They applied their method to several robotic platforms such as drones and manipulator arms.

$$\begin{aligned} \text{minimize} \quad & \sum_{e:=(u,v)\in\mathcal{E}_p} l_e(x_u, x_v) \\ \text{subject to} \quad & p \in \mathcal{P}, \\ & x_v \in \mathcal{X}_v, \quad \forall v \in \mathcal{P}, \\ & (x_u, x_v) \in \mathcal{X}_e, \quad \forall e := (u, v) \in \mathcal{E}_p. \end{aligned} \tag{1}$$

III. Problem Formulation

A. Footstep Planning

The standard obstacle avoidance problem in trajectory optimization becomes complex due to the non-convexity of the obstacle-free space. GCS tackles this challenge by partitioning the obstacle-free space into convex sets, through which it plans trajectories. Our original idea was to divide the obstacle-free space in the environment into convex sets using IRIS [1] [4]. However, for simplicity we chose to define the convex sets ourselves as stepping stones, which Spot had to traverse.

We created random 2D terrains of stepping stones, such as the example environment setting depicted in Figure 1. The goal for Spot was to navigate from the blue box to the green box by traversing the convex stones. For Spot to achieve this, it needed to plan a sequence of step configurations for all 4 legs that it is then able to follow using a whole-body controller. Since each step configuration is an 8 dimensional vector (2D position of each foot) the computational complexity grew extremely fast. We tackled this challenge by instead planning a footstep sequence for the front 2 legs, and planned for the rear 2 legs to step into the same footstep positions as the front legs. We used GCS to plan a sequence of the now 4 dimensional footstep configurations, which is described in detail in Section IV-A.

B. Whole Body Control

Once we created the desired footstep configurations that traversed the stepping stones, we needed to control Spot to follow these footsteps. The existing framework we were provided with only contained a PD controller for Spot's joint

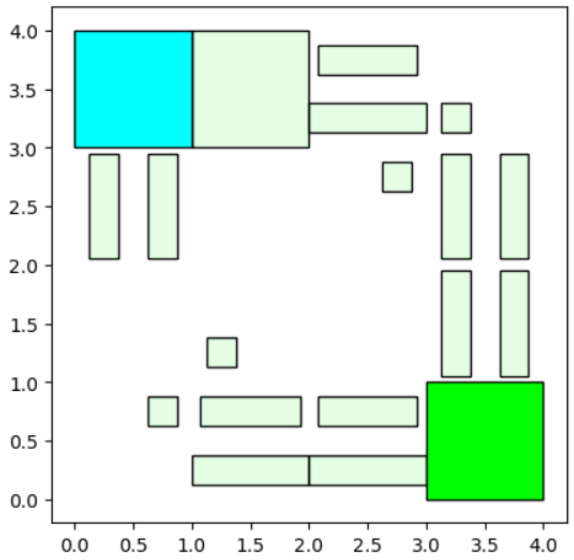


Fig. 1: Example of a terrain GCS can plan footsteps through

actuators. This meant that we needed to calculate desired joint trajectories for the whole body which can be passed to the low level PD controller. We did this by firstly calculating the desired joint positions at each footstep configuration using inverse kinematics. Then we solved a whole body optimization problem in order for Spot to plan joint trajectories between these configurations. For simplicity we only moved one leg at a time and solved a new optimization problem for each step individually. The inverse kinematic calculation is described in Section IV-B and the whole body controller in Section IV-C.

IV. Methods

Our approach shown in Figure 2 used GCS to compute Cartesian footstep positions from a random input environment. IK then converted the Cartesian footstep positions into joint positions analytically. We used the whole body controller to solve a quadratic program that finds feasible joint positions based on the ideal ones from IK. The new joint positions were given to the PD controller that tried to command Spot.

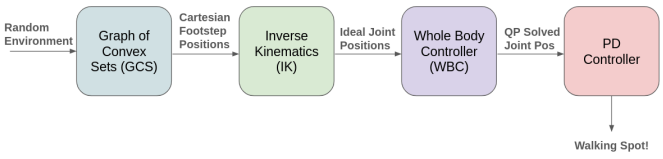


Fig. 2: Flow Diagram

A. GCS for footstep planning

For our problem, the convex sets are 4D concatenations of two 2D foot positions (left and right foot), where each foot is constrained to lie within a convex stepping stone such as the ones making up the terrain in Figure 1. How a pair-of-feet-position's vertex constraints can be constructed is seen in Figure 2. The constraints were constructed from the linear inequalities constraining individual feet to individual stones. For our quadruped footstep planning, a single pair of feet was chosen rather than the more natural quartet because this results in significantly fewer vertices and edges. Additionally, a quadruped gait can easily be extracted from pairs of foot positions by having hind-feet step into previous fore-feet positions. For our edge-length function, we used a constant $l_c(x_u, x_v) = 1$ to minimize the amount of steps irrespective of their lengths.

$$\mathbf{x}_v = \begin{bmatrix} \vec{x}_{left} \\ \vec{x}_{right} \end{bmatrix} = \begin{bmatrix} x_{left} \\ y_{left} \\ x_{right} \\ y_{right} \end{bmatrix} \tag{2}$$

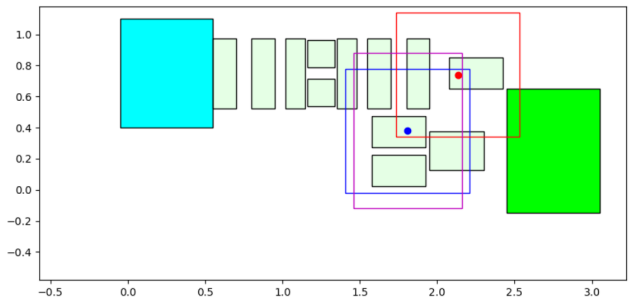
$$A_{\text{stone}} \vec{x}_{left/right} \leq b_{stone} \ , \ \ A_v \mathbf{x}_v \leq b_v$$

$$A_v = \begin{bmatrix} A_{stone1} & 0 \\ 0 & A_{stone2} \end{bmatrix} \ , \ \ b_v = \begin{bmatrix} b_{stone1} \\ b_{stone2} \end{bmatrix}$$

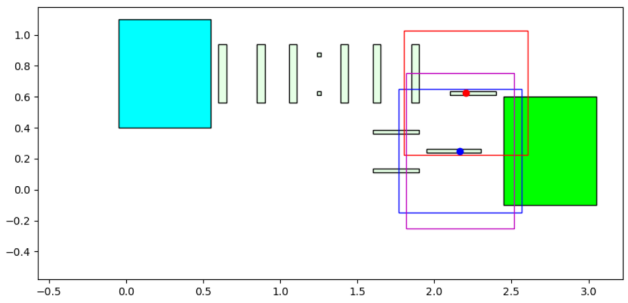
Beyond the vertex constraints, we also enforced constraints on the edges between vertices, ie. transitions between foot positions. These are intended to ensure the physical feasibility of each step. Our three main constraints are 1) our stance foot constraint, ie. only one foot moves at a time, 2) our step_span constraint, and 3) our sprawl_span constraint. The latter two constraints are shown in Figure 3a. If you know a-priori that your robot will be moving roughly in a specific direction, you can also constrain left and right feet to remain to the left/right of each other by a given margin, δ_{lr}, thus making your footstep trajectory more physically feasible. How the δ_{lr} constraint is enforced depends on direction; for trajectories going from the left to the right, the correct constraint is given by 3. A more complete overview of edge constraints is given by 4, where $swing_{x/y}$ and $stance_{x/y}$ refer to moving foot and stance foot indices. Notice how the top equation of 4 is a generalization of 3. The sprawl is sorted into x and y, which is primarily applicable in the above-mentioned case where you know a-priori the direction your robot will be moving. In our case it allowed us to give our quadruped a larger sprawl_span in its shoulder direction than along its body length. In 3 and 4, all constraints were formulated as simple rectangular constraints, with the sides of the rectangles aligning with the unit-vectors of the x-y coordinate-system. In general, more complicated polygonal constraints can also be encoded as linear constraints, though we have chosen not to do so.

$$\mathbf{x}_v[left\_foot\_y\_idx] \geq \mathbf{x}_v[right\_foot\_y\_idx] + \delta_{\text{lr}} \tag{3}$$

.



(a) Physical Terrain



(b) Conservative Terrain

Fig. 3: A visualization of how left foot (red) and right foot (blue) are constrained by step_spans (red and blue) and sprawl_span (purple). Since in reality, the edge of a stepping stone is not safe terrain for a legged robot, a reduced conservative terrain (b) is typically provided to the algorithm rather than the real physical terrain (a)

$$
\begin{aligned}
\mathbf{x}_v[left_{y/x}] &\geq \mathbf{x}_v[right_{y/x}] + \delta_{\text{lr}} \\
(\mathbf{x}_u - \mathbf{x}_v)[swing_x] &\leq \text{step} \\
(\mathbf{x}_u - \mathbf{x}_v)[swing_x] &\geq -\text{step} \\
(\mathbf{x}_u - \mathbf{x}_v)[swing_y] &\leq \text{step} \\
(\mathbf{x}_u - \mathbf{x}_v)[swing_y] &\geq -\text{step} \\
\mathbf{x}_u[stance_x] &= \mathbf{x}_v[stance_x] \\
\mathbf{x}_u[stance_y] &= \mathbf{x}_v[stance_y] \\
\mathbf{x}_u[stance_x] - \mathbf{x}_v[swing_x] &\leq \text{sprawl}_x \\
\mathbf{x}_u[stance_x] - \mathbf{x}_v[swing_x] &\geq -\text{sprawl}_x \\
\mathbf{x}_u[stance_y] - \mathbf{x}_v[swing_y] &\leq \text{sprawl}_y \\
\mathbf{x}_u[stance_y] - \mathbf{x}_v[swing_y] &\geq -\text{sprawl}_y
\end{aligned}
\tag{4}
$$

Because the point $\mathbf{x_v}$ of a vertex v is a point, and has no size, our solution trajectories can in general contain points lying on the very edge of the convex sets. Since such footstep positions are physically infeasible in practice (Spot's legs can slip off), we typically provided our algorithm with a slimmed down version of the terrain, so that the provided trajectory was guaranteed to consist of points well away from the edge of the stepping stones. As an example, for the footsteps illustrated in Figure 3a, the terrain provided to our algorithm was in fact the one shown in Figure 3b. Usage of such more narrow-stoned terrains also affected solve times, as will be discussed in section V.

To minimize the computational complexity of our algorithm, we built our graph of convex sets to be as small as possible. The method used for this is illustrated in algorithm 1. The reason for this exploratory search from source, rather than iterating over all stone pairs and checking feasibility, is that we did not want to include remote islands of stepping stones that would never connect to the source vertex.

As discussed in detail in [8], the shortest path problem through a graph of convex sets is in general NP-Hard. The secret to the efficient implementation in [7] was that they first solved the convex relaxation of the problem, and they then recovered an approximate solution using a cheap randomized rounding technique. For a full description of the convex relaxation and it's implications, the reader is again advised to consult [7], but a brief explanation will be provided here. If the shortest path problem is envisioned as a maximum flow problem, then the exact problem can be described as follows: Any edge in the graph can have a flow of either 1 or 0, and

Algorithm 1 Algorithm for Processing Stones and Edges

```
K ← list of stones
E ← empty list of edges
V ← empty list of vertices
V.append(source vertex)
V_todo ← empty list of vertices
V_todo.append(source vertex)
while ¬V_todo.empty() do
    u ← V_todo.pop()
    a, b ← u.stones()
    for stone ∈ K do
        for each leg do
            v ← replace non-stance leg's stone with new
            if new stone is same as before then
                continue
            end if
            e ← (u, v)
            if ¬edge e satisfies constraints then
                continue
            end if
            if v ∉ V then
                V_todo.append(v)
            end if
            if e ∈ E then
                continue
            else
            end if
        end for
    end for
end while
```

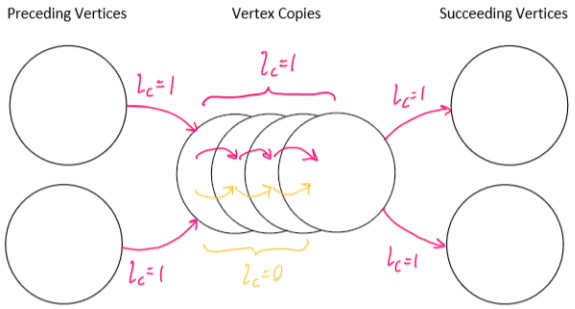


Fig. 4: Visualization of how vertex copies can be efficiently connected. Red arrows are normal edges with normal step length constraints and normal costs, ie. correspond to physical steps along a given stepping-stone pair. Yellow edges have zero cost, and are constrained to a step-length of zero, ie. no movement. The point of the zero-cost edges is to avoid needing edges between every vertex copy and every succeeding vertex. It's important to note that each circle is an abstract representation of a pair-of-feet vertex, ie. individual circles are not equivalent to individual stepping stones.

the only edges with a flow of 1 are the ones connecting the vertices in the path p (see 1), where each vertex has exactly one incoming and one outgoing edge of non-zero flow [7]. In comparison, the convex relaxation consists of relaxing this binary constraint $flow \in \{0, 1\}$ to be convex, $0 \leq flow \leq 1$. This relaxation is not always tight; however, as described in [8], adversarial examples include the symmetric case.

Our GCS formulation allows for more complicated constraints and objectives than our constant edge cost and the constraints described in equation 4. For example, in the newer PyDrake implementations of GCS, it is possible to add non-convex constraints to the optimization problem during the relaxation part of the problem. In our case, we attempted on separate occasions to add the non-linear convex cost shown in equation 5 and the nonlinear constraint shown in equation 6. For some small problems, we also added the capability to step on large stepping stones more than once by creating copies of vertices. We added selective edges between the copied vertices as well as between copy vertices and succeeding normal vertices, like shown in Figure 4.

$$l_c(x_u, x_v) = 1 + (x_u - x_v)^2 \tag{5}$$

$$|x_u - x_v|_2 \geq balance\_req \tag{6}$$

B. Inverse Kinematics (IK)

Given the desired footstep configurations from GCS we used inverse kinematics to determine the desired joint positions, which form the start and end configurations to our whole body optimization. Figure 5 depicts the leg model which contains three actuated joints, with the commanded joint angles being: $\theta_1, \theta_2, \theta_3$. The inverse kinematics module takes the desired body position, as well as the desired footstep positions and calculates these angles for each leg individually.

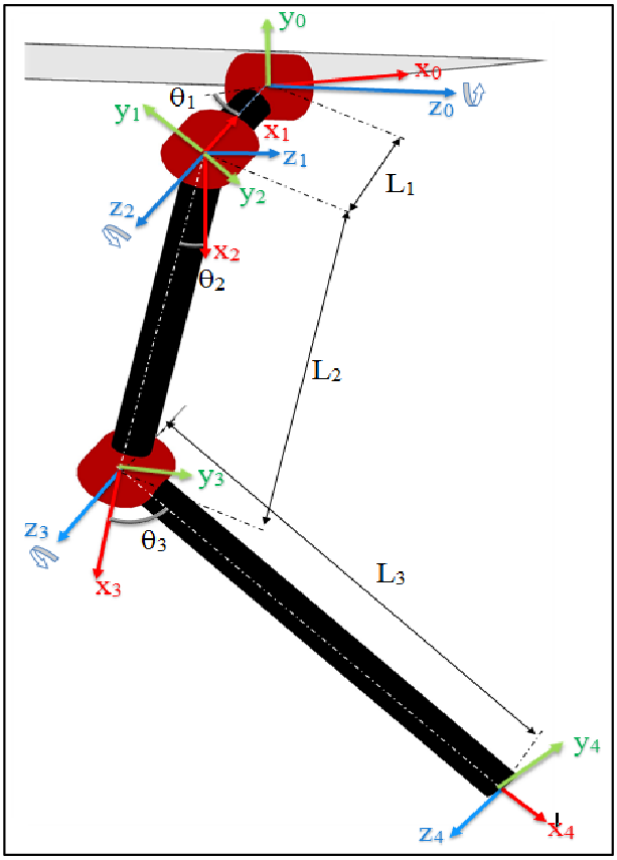


Fig. 5: Spot leg model for inverse kinematics [9]

For the desired body position we chose a fixed height of 45cm above the ground and the body center to be the mean of the 4 footstep locations. Given this, we firstly computed the desired leg endpoint location in respect to the shoulder frame, which is the start of the kinematic chain of the leg. These relative coordinates are given by x_4, y_4, z_4, as can bee seen in Figure 5. Then the inverse kinematic solution is found using analytical methods as described by Sen et al. in [9] for a general quadruped. We used a closed form solution which is based off the general description by Sen at al. [9] and is available publicly at: https://github.com/mike4192/spot_micro_kinematics_python. The resulting expressions for the desired joint angles are expressed in Equation 7. It is important to note that for every mathematical expression computed, there may not be a physical solution.

$$\theta_1 = atan2(y_4, x_4) + atan2(\sqrt{x_4^2 + y_4^2 - L_1^2}, -L_1)$$

$$\theta_2 = atan2(z_4, \sqrt{x_4^2 + y_4^2 - L_1^2}) - atan2(L_3 \sin(\theta_3), L_2 + L_3 \cos(\theta_3)) \tag{7}$$

$$\theta_3 = \begin{cases} atan2(\sqrt{1 - D^2}, D) & \text{(for right legs)} \\ atan2(-\sqrt{1 - D^2}, D) & \text{(for left legs)} \end{cases}$$

where:

$$D = \frac{x_4^2 + y_4^2 + z_4^2 - L_1^2 - L_2^2 - L_3^2}{2L_2L_3}$$

C. Whole Body Control (WBC)

Whole body control was used to compute a trajectory of feasible joint positions from Spot's current initial position $\mathbf{q}_0$ to its desired final position $\mathbf{q}_{\text{end}}$ (calculated through IK). We based our algorithm off the open-source implementation of gait optimization for LittleDog in Drake [10], which follows the whole-body control description proposed by Dai et al. in [3]. Our resulting formulation solves the quadratic problem described below, using SNOPT [5] as the solver. We chose a time horizon of $N = 10$ steps and $T = 1.5$ seconds for each footstep trajectory.

Minimize the cost function:

$$J = \sum_{n=0}^{N-1} \left(\mathbf{q}_n^T \mathbf{Q} \mathbf{q}_n + \mathbf{v}_n^T \mathbf{V} \mathbf{v}_n\right) + 10\left(\mathbf{q}_0^T \mathbf{Q} \mathbf{q}_0 + \mathbf{q}_{N-1}^T \mathbf{Q} \mathbf{q}_{N-1}\right)$$

over the variables:

$$\mathbf{q}_n, \mathbf{v}_n, h_n, \mathbf{f}^{(i)}_{\text{contact},n}, \mathbf{r}_{\text{com},n}, \dot{\mathbf{r}}_{com,n}, \ddot{\mathbf{r}}_{\text{com},n}, \mathbf{H}_n, \dot{\mathbf{H}}_n$$

subject to following constraints:

1) Time Steps:

$$0.5 \cdot T/N \leq h_n \leq 2 \cdot T/N,$$
$$0.9 \cdot T \leq \sum_{n=0}^{N-1} h_n \leq 1.1 \cdot T$$

2) Joint Limits:

$$\mathbf{q}_{\text{min}} \leq \mathbf{q}_n \leq \mathbf{q}_{\text{max}},$$
$$\mathbf{v}_{\text{min}} \leq \mathbf{v}_n \leq \mathbf{v}_{\text{max}}$$

3) Unit Quaternions (Drake):

$$\text{AddUnitQuaternionConstraintOnPlant}(\mathbf{q}_n)$$

4) Body Orientation (Drake):

$$\text{OrientationConstraint}(\mathbf{q}_n, \theta_{bound} = 0.1)$$

5) Velocity Dynamics:

$$\mathbf{v}_n = \frac{\mathbf{q}_{n+1} - \mathbf{q}_n}{h_n}$$

6) Stance Feet Friction Cones:

$$\mathbf{f}^{(i)}_{\text{contact},n}[0] \leq \mu \cdot \mathbf{f}^{(i)}_{\text{contact},n}[2],$$
$$-\mathbf{f}^{(i)}_{\text{contact},n}[0] \leq \mu \cdot \mathbf{f}^{(i)}_{\text{contact},n}[2],$$
$$\mathbf{f}^{(i)}_{\text{contact},n}[1] \leq \mu \cdot \mathbf{f}^{(i)}_{\text{contact},n}[2],$$
$$-\mathbf{f}^{(i)}_{\text{contact},n}[1] \leq \mu \cdot \mathbf{f}^{(i)}_{\text{contact},n}[2],$$

7) Stance Feet Contact Forces:

$$0 \leq \mathbf{f}^{(i)}_{\text{contact},n}[2] \leq \begin{cases} 4 \cdot m_{tot} \cdot \mathbf{g}, \text{ if foot } i \text{ in stance} \\ 0, \text{ else} \end{cases}$$

8) Center of Mass and Angular Momentum (Drake):

$$\mathbf{r}_{\text{com},n} = \text{CenterOfMass}(\mathbf{q}_n)$$
$$\mathbf{H}_n = \text{CalcSpatialMomentumInWorldAboutPoint}(\mathbf{q}_n, \mathbf{v}_n, \mathbf{r}_{\text{com},n})$$

9) Center of Mass Constraints:

$$\mathbf{r}_{\text{com},0,x} = \mathbf{q}_{0,x},$$
$$\mathbf{r}_{\text{com},0,y} = \mathbf{q}_{0,y},$$
$$\mathbf{r}_{\text{com},n,z} \geq 0.5 \cdot \mathbf{q}_{0,z},$$
$$\mathbf{r}_{\text{com},N-1,x} = \mathbf{q}_{\text{end},x},$$
$$\mathbf{r}_{\text{com},N-1,y} = \mathbf{q}_{\text{end},y},$$
$$\dot{\mathbf{r}}_{\text{com},0,z} = 0$$

10) Center of Mass Dynamics:

$$\mathbf{r}_{\text{com},n+1} = \mathbf{r}_n + h_n \cdot \dot{\mathbf{r}}_{\text{com},n}$$
$$\dot{\mathbf{r}}_{\text{com},n+1} = \dot{\mathbf{r}}_n + h_n \cdot \ddot{\mathbf{r}}_{\text{com},n}$$
$$\ddot{\mathbf{r}}_{\text{com},n} = \frac{\sum_i \mathbf{f}^{(i)}_{\text{contact},n} + m_{\text{tot}} \cdot \mathbf{g}}{m_{\text{tot}}}$$

11) Angular Momentum Constraints:

$$\mathbf{H}_{n+1} = \mathbf{H}_n + h_n \cdot \dot{\mathbf{H}}_n$$
$$\dot{\mathbf{H}}_n = \sum_i \left(\mathbf{r}^{(i)}_{\text{foot,WF},n} - \mathbf{r}_{\text{com},n}\right) \times \mathbf{f}^{(i)}_{\text{contact},n}$$

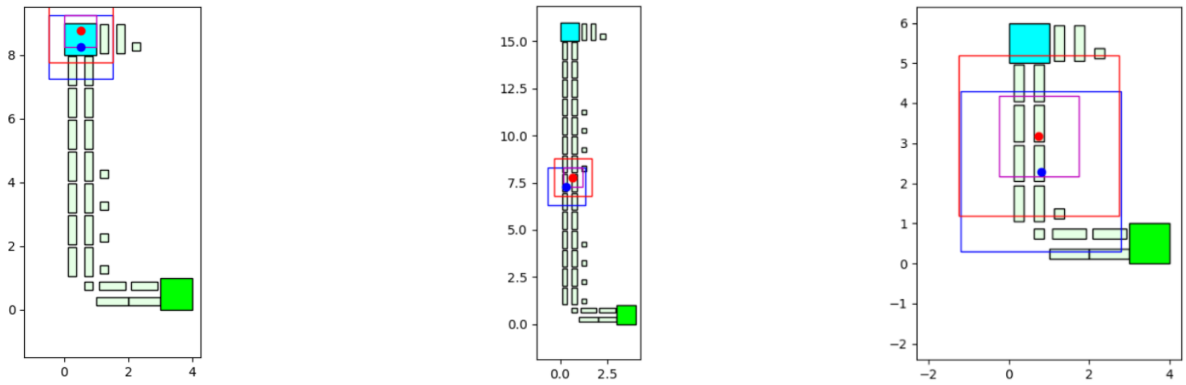
12) Stance Feet Positions (World Frame):

$$\mathbf{r}^{(i)}_{\text{foot,WF},n,z} = 0,$$
$$\mathbf{r}^{(i)}_{\text{foot,WF},n} = \mathbf{r}^{(i)}_{\text{foot,WF},n-1}$$

13) Swing Foot Positions (World Frame):

$$\mathbf{r}^{(i)}_{\text{foot,WF},n,z} \begin{cases} = 0, & n \in \{0, N-1\}, \\ \geq 0.1, & n = N/2, \\ \geq 0.02, & \text{else} \end{cases}$$

The cost function includes larger weights of 10 on the initial and final positions $\mathbf{q}_0$ and $\mathbf{q}_{N-1}$. It also includes costs on the interpolated positions and velocities to guide the trajectory. The reason for the choice of high initial and final position weights, is that adding hard constraints on the initial and final positions made the optimization problem infeasible. This is likely due to the large number of other constraints within the problem formulation. For example constraints (12) and (13) state that the z-coordinates of all four feet positions in the world frame need to be zero at the first time step. However, this is likely not feasible to simultaneously fulfill together with the initial joint angle configuration $\mathbf{q}_0$, due to numerical rounding errors or bad PD control tracking of $\mathbf{q}$.



(a) GCS solve time: 11.6 seconds
Number of vertices: 147
Number of edges: 1073

(b) GCS solve time: 23.8 seconds
Number of vertices: 251
Number of edges: 1893

(c) GCS solve time: 83.2 seconds
Number of vertices: 120
Number of edges: 1307

Fig. 6: As can be seen, depending on terrain, the growth in problem size can be much worse when span sizes increase than when terrain increases in size.

D. Low Level PD Control

A PD controller was used to command the desired $\mathbf{q}$ and $\dot{\mathbf{q}}$ outputted from the whole body control. The PD tracks the positions and velocities at 32 Hz. Careful tuning of Kp and Kd gains balanced speed and accuracy of position following. A Kp of 2000 and Kd of 100 was selected ultimately.

V. Results

A. GCS

Our GCS results can naturally be sorted into a few separate cases, which will be described in detail in their corresponding subsections below. Where precise run-times, etc. are mentioned, unless otherwise stated, assume the solver and solver options match the ones provided in table I.

Option	Value
Preprocessing	False
Maximum Rounded Paths	100
Maximum Rounding Trials	100
Convex Relaxation	True
Solver	Clarabel [6]

TABLE I: Solver Configuration

1) Omnidirectional Terrain: For the terrain illustrated in Figure 1 GCS was able to solve the problem in 1.88 seconds. Due to the way our graph is built, which is shown in algorithm 1, the amount of vertices and edges scale with the amount of stones "neighbouring" any other stone. In the worst-case, every stone neighbours every other stone, and we get $O(K^2)$ vertices and $O(K^4)$ edges. Since the amount of "neighbours" scales not only with stones, but also with the sizes of the step and sprawl spans, this motivates the hypothesis that the latter two can potentially affect run-time much more drastically than simply increasing the size of the terrain. As can be seen in Figure 6, this was also the case.

```
Interior-point solution summary
  Problem status  : PRIMAL_AND_DUAL_FEASIBLE
  Solution status : OPTIMAL
  Primal.  obj: 1.8939734357e+01    nrm: 4e+00    Viol.  con: 8e-10    var: 1e-11    acc: 2e-09
  Dual.    obj: 1.8939734294e+01    nrm: 1e+02    Viol.  con: 5e-09    var: 2e-08    acc: 0e+00
```

Fig. 7: When looking at the solver logs of the Mosek Solver, we observe that the primal-dual feasibility gap is almost non-existent.

2) The Trajectory's Cardinal Direction Known A-Priori: For terrains such as the ones shown in Figures 3a and 3b, we were able to enforce the constraint seen in equation 3, giving us more physically reasonable footstep trajectories. In the illustrated example, feet were forced to be more than 0.3 meters apart from each other in the y direction. As mentioned in section IV-A, the footsteps for the terrain in Figure 3a would typically be attained using the slimmer terrain shown in Figure 3b to avoid stepping on the very edges of stepping stones. This choice of terrain, however, also affects solve times. While GCS took 4.5 seconds to solve the graph corresponding to the slimmer terrain, it took 50.1 seconds to solve the graph corresponding to the more wide-stoned terrain.

3) More Complicated Edge Costs: We attempted to implement edge costs such as the one seen in equation 5; however, to our surprise, trivial problems failed to solve. We believe there was a bug in Drake's GCS implementation shown in Figure 7, which displays the logging output of the solver used (Mosek, in this case). The theory behind the significance of the primal and dual objectives will not be explained in this report, but in short, the primal and dual objectives constitute upper and lower bounds for the optimal solution of the problem. For a detailed account of why this is the case, the reader is recommended to consult section 8 in [8].

4) Copies of Vertices for Large Stepping Stones: We also implemented copies of vertices with connections as illustrated in Figure 4. This allowed us to solve trivial problems such as the one shown in Figure 8, which solved in 0.04 seconds.

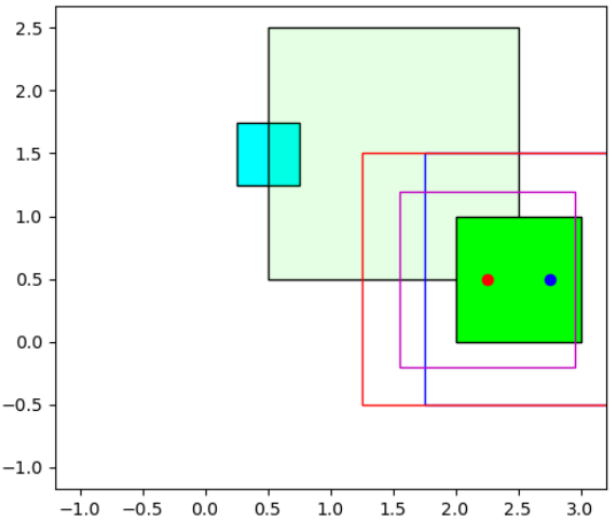


Fig. 8: Using the copy vertices, we planned a footstep trajectory where the same stone had to be stepped on by both feet multiple times.

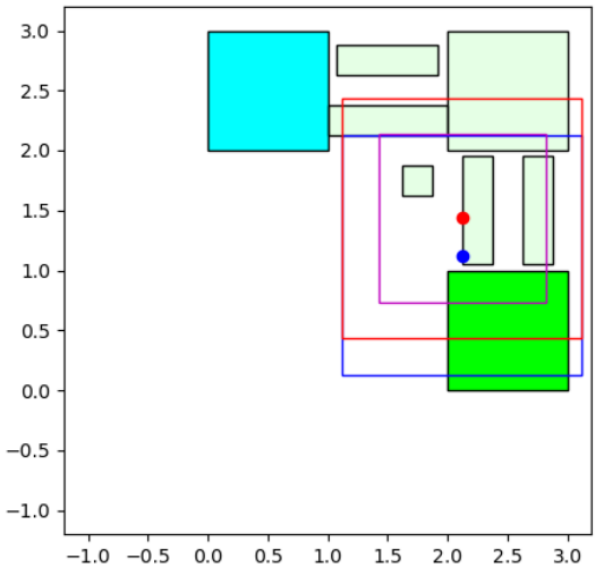
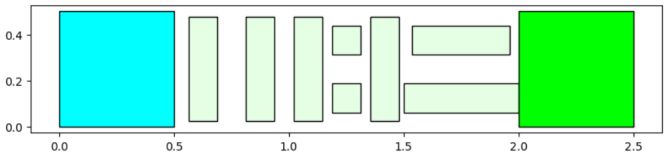


Fig. 9: Using non-convex constraints during the restriction stage of GCS, we were able to force feet further apart than normal.

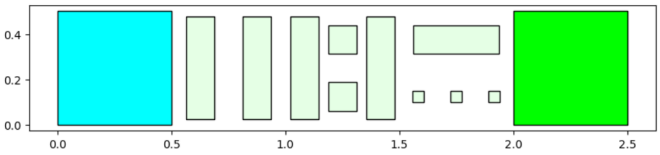
Unfortunately, not a lot of work was done on utilizing the copy capabilities for larger terrains because initial implementations and results seemed unpromising. In some cases, this was due to the symmetry problem, which, though obvious in hindsight, was not apparent when the copy-solution was dropped. It was deemed unnecessary for most terrains of interest. It also seems plausible that the convex relaxation becomes loose due to some other unrealized reason, or that there is a bug/unrealized flaw in the implemented solution.

5) Nonconvex Constraints during GCS Restriction Stage: As discussed in section IV-A, we also attempted to add constraints of the form shown in equation 6 to the restriction stage of the optimization. We were successful in this, as shown in Figure 9, where due to such non-convex constraints, the feet are about 0.01 further apart than they would be without the non-convex constraints. This constitutes a trivial difference, but if balance_req is chosen more aggressively, the optimization fails because these constraints are incompatible with the solution found during the pre-restriction stages. No attempts were made to utilize GCS's non-convex capabilities better, because it was not believed it would be very useful for the quadruped case. Why this is the case will be discussed in more detail in section VI

6) An Adversarial Example: Symmetry: As described in [8], the convex relaxation can become loose in the presence of symmetry. We experienced this being the case for the terrain shown in Figure 10a. When attempting to solve the optimization problem for this graph without a left-right constraint as described in 3, neither Mosek [2] nor Clarabel [6] seemed able to solve this problem; we allowed the optimization to run for 7 minutes, after which we terminated the attempt. While we realize that many problems can require more than 7 minutes, we chose this threshold because similar problems without the same symmetry, such as the one seen in Figure 10b, typically solved in less than a second (0.7 seconds). When further constraints were added, such as the left-right constraint, Clarabel remained unable to solve the problem, while Mosek once more became able to solve the problem in less than a second (0.57 seconds).



(a) Symmetric Terrain



(b) Asymmetric Terrain

Fig. 10: Symmetric (a) and Asymmetric (b) Terrains. When supplied with only the bottom ten constraints in 4, our algorithm is unable to find a footstep plan for the symmetric terrain during the time we allowed it to run (7 minutes). The asymmetric terrain is solved in less than a second under the same conditions.

B. Whole Body Control and PD

The whole body controller can successfully plan joint trajectories that transition between footstep configurations. We were able to evaluate this by publishing the trajectories in the Drake visualization without running the physics simulation. What

we did visually notice is that there was slight teleportation between the start and end configurations of each step. This makes sense due to the fact that the initial and end joint positions were not able to be added to the WBC optimization as hard constraints. When passing the planned joint trajectories to the low level PD controller and running the physics simulation in Drake we could tell that the PD controller was not able to track the trajectories perfectly. As a result of this Spot would tend to lose balance at some point later on in the footstep plan and fall over. One reason for this is likely the slight teleportation between start and end configurations. Another cause may be that we were not able to constrain the center of mass to specifically be within the support polygon of the three stance legs. These issues indicate clear potential for improvement regarding the WBC formulation, so that it can be tracked more accurately by the PD controller.

Another significant downside of our WBC implementation is the computational complexity. For the example terrain configuration displayed in Figure 3a our average solve time was 11.1 seconds per footstep with a standard deviation of 2.0 seconds. Our implementation could definitely be improved upon in order to reduce the solve time. In general, the WBC formulation we described in section IV-C seems quite overkill for solving a single footstep trajectory at a time. Other methods exist that could be more suitable to this problem, like using Bezier curves and pure inverse kinematics along the trajectory instead of solving a QP optimization. The benefit of our WBC is that it is a very general formulation for Spot in Drake that can be used for more challenging and dynamic situations. With further improvements to the code, it could also become an efficient control method for testing footstep planning with trajectory optimization on Spot.

C. Full Pipeline

After integrating GCS into the environment and IK-WBC-PD pipeline, Spot was able to traverse some terrains. An example terrain is shown in Figure 11. However, limitations in the current WBC and PD restricted Spot from larger extensions and body angling necessary for large angled steps. Results can be viewed in the accompanying video: https://youtu.be/cigEIIFpSio.

VI. Discussion

While our approach was able to solve feasible footsteps with teleportation, there were some limitations reaching footsteps with PD. Future work can be done in the whole body controller and PD controller to reduce some of these limitations. For example, eliminating body orientation constraints in WBC and finding alternative methods to calculate the ideal location of the center of mass in IK may produce more stable joint configurations for Spot.

As for the GCS formulation, our solution was able to generate footstep plans for a moderate range of terrains in seconds. Further, we were able to enforce some physical feasibility constraints in the form of the left-right constraints (top equation in equation 4). Unfortunately, these constraints

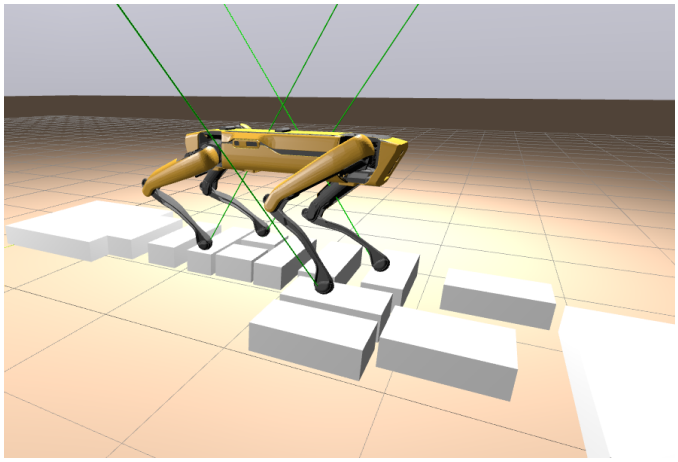


Fig. 11: Spot traversing environment with full pipeline (GCS-IK-WBC-PD)

are dependent on the trajectory being in more or less the same direction for the entirety of the footstep plan. The reason for this is that, with two legs, you cannot know the orientation of the body, and thus not along which direction you should enforce left-right constraints, without knowledge of earlier edges, something GCS does not allow. This seems to be the most significant limitation of our method, and thus future work would involve ways to circumvent this. Possible solutions could involve: 1) finding likely body-directions before the GCS optimization. This could possibly be done by constructing candidate higher level paths in the x-y plane from source to target, and have the body directions of these paths be functions of x and y, and thus accessible to GCS at runtime. Alternatively, 2) finding a cheap way to encode body orientation as part of the convex set, necessarily a cheaper way than adding 2 extra legs.

Acknowledgment

We would like to thank Professor Russ Tedrake, Savva Morozov, and all the Underactuated Robotics staff.

References

[1] A. Amice, H. Dai, P. Werner, A. Zhang, and R. Tedrake. Finding and optimizing certified, collision-free regions in configuration space for robot manipulators, 2022.

[2] M. ApS. Mosek. https://www.mosek.com/.

[3] H. Dai, A. Valenzuela, and R. Tedrake. Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, pages 295–302, 2014.

[4] R. Deits and R. Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. 107:109–124, 04 2015.

[5] P. E. Gill, W. Murray, and M. A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002.

[6] P. Goulart and Y. Chen. Clarabel. https://oxfordcontrol.github.io/ClarabelDocs/stable/#Credits.

[7] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake. Motion planning around obstacles with convex optimization, 2022.

[8] T. Marcucci, J. Umenberger, P. A. Parrilo, and R. Tedrake. Shortest paths in graphs of convex sets. *CoRR*, abs/2101.11565v3, 2021.

[9] M. Sen, V. Bakircioglu, and M. Kalyoncu. Inverse kinematic analysis of a quadruped robot. *International Journal of Scientific Technology Research*, 6:285–289, 01 2017.

[10] R. Tedrake and the Drake Development Team. Drake: A planning, control, and analysis toolbox for nonlinear dynamical systems, 2016.