Effective Footstep Planning for Humanoids Using Homotopy-Class Guidance

Vinitha Ranganeni, Oren Salzman, Maxim Likhachev
The Robotics Institute, Carnegie Mellon University
{vrangane, osalzman}@andrew.cmu.edu, maxim@cs.cmu.edu

Abstract

Planning the motion for humanoid robots is a computationally-complex task due to the high dimensionality of the system. Thus, a common approach is to first plan in the low-dimensional space induced by the robot's feet—a task referred to as *footstep planning*. This low-dimensional plan is then used to guide the full motion of the robot. One approach that has proven successful in footstep planning is using search-based planners such as A* and its many variants. To do so, these search-based planners have to be endowed with effective heuristics to efficiently guide them through the search space. However, designing effective heuristics is a time-consuming task that requires the user to have good domain knowledge. Thus, our goal is to be able to effectively plan the footstep motions taken by a humanoid robot while obviating the burden on the user to carefully design local-minima free heuristics. To this end, we propose to use user-defined homotopy classes in the workspace that are intuitive to define. These homotopy classes are used to automatically generate heuristic functions that efficiently guide the footstep planner. We compare our approach for footstep planning with a standard approach that uses a heuristic common to footstep planning. In simple scenarios, the performance of both algorithms is comparable. However, in more complex scenarios our approach allows for a speedup in planning of several orders of magnitude when compared to the standard approach.

1 Introduction

Humanoid robots have been shown as an effective platform for performing a multitude of tasks in human-structured environments (Kajita et al. 2014). However, planning the motion of humanoid robots is a computationally-complex task due to the high dimensionality of the system. Thus, a common approach to efficiently compute paths in this high-dimensional space is to guide the search using footstep motions which induce a lower dimensional search space (Gochev et al. 2011a; Garimort, Hornung, and Bennewitz 2011). This lower dimensional search space is a six-dimensional configuration space of a humanoid robot's feet consisting of x, y positions and orientation for each foot.

One approach that has been successful in footstep planning is using search-based planners such as A* (Chestnutt et al. 2005; 2007) and its anytime variants (Garimort, Hornung, and Bennewitz 2011; Hornung et al. 2012). To effectively plan footstep motions, these planners require heuristics to guide the search. Effective heuristic functions should avoid regions where the search ceases to progress towards the goal or when this progress is extremely slow (a region often referred to as a "local minima" or a "depression region" (Ishida 1992)).

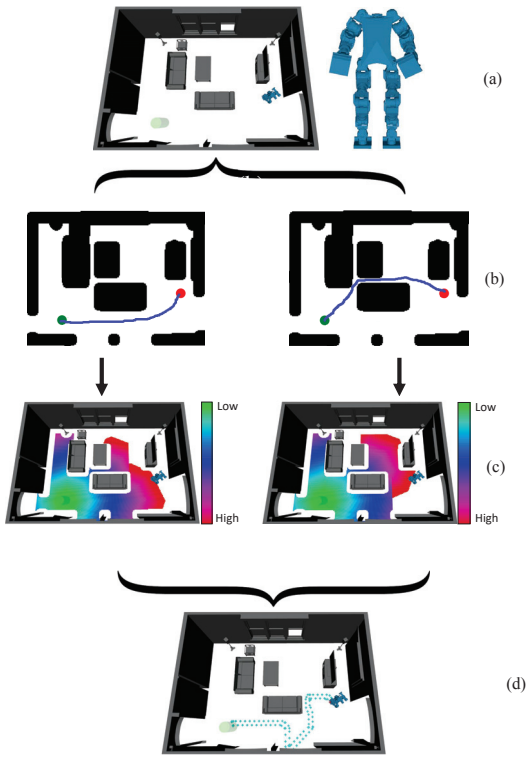


Figure 1: (a) A humanoid robot has to navigate to a goal region denoted by the green cylinder. (b) A user provides two reference paths in a 2D projection of the workspace. (c) A color map of the heuristic values for each homotopy-based heuristic. A single heuristic is constructed for each homotopy class of the reference paths. (d) The footstep planner uses both heuristics to quickly find a path to the goal region.

Consider, for example, Fig. 1.a. Planning a path that passes between the couch and the table is either infeasible or may require expanding a massive number of states in order to precisely capture the sequence of configurations in which the robot does not collide with obstacles. The aforementioned challenges make constructing effective heuristics, that can intelligently reason about areas of the environment to avoid, a time consuming and tedious task that often requires strong domain knowledge.

In this work we effectively plan footstep motions for a humanoid robot while eliminating the need to manually design heuristics. Our key insight is that providing sketches of desirable trajectories, that can be represented as homotopy classes, can be used to automatically generate heuristic functions. Ideally, these homotopy classes would not be user-defined, but also automatically generated. However, this is out of the scope of our work. Instead, we present a method for generating heuristic functions given homotopy classes. We simultaneously use multiple such heuristics to alleviate the requirement of capturing the different complexities induced by an environment in one single heuristic while maintaining guarantees on completeness.

Specifically, for each user-defined homotopy class in the workspace, we generate a heuristic function by running a search from the goal to the start configuration while restricting the search to expand only vertices within the specified homotopy-classes. We call this algorithm Homotopy-Based Shortest Path, or HBSP and detail it in Sec. 4. Additionally, we assume the existence of a simple-to-define heuristic which is admissible and consistent[1].

These heuristics are then used in Multi-Heuristic A* (MHA*) (Aine et al. 2016) which is a recently-proposed method that attempts to leverage information from multiple heuristics. Roughly speaking, MHA* simultaneously runs multiple A*-like searches, one for each heuristic, and combines their different guiding powers in different stages of the search. MHA* is detailed in Sec. 3 and the use of MHA* together with HBSP is detailed in Sec. 4.

While our approach requires computing multiple heuristics before the planner can be executed, this can be done efficiently and thus takes a small fraction of the planning time. Moreover, the extra computation invested in computing these heuristics allows to efficiently guide the footstep planner. In some queries (Sec. 5), we present a speedup of several orders of magnitude when compared to standard approaches.

1.1 Motivating Example

Consider Fig. 1.a where a humanoid robot has to navigate to the goal region denoted by the green cylinder. Footstep planning for the humanoid plans in a 6D space defined by the position and orientation of each foot. We calculate a simple heuristic by running a Dijkstra search backwards from the goal vertex q_{goal} to every vertex in the 2D workspace. When executing the footstep planner with only this backward 2D Dijkstra heuristic $\mathcal{H}_{\text{Dijk}}$, the search is guided through the narrow passage between the couch and table, as it is the shortest path to the goal region. However, it is not feasible for the robot to pass through this region. After spending a significant amount of time expanding near the narrow passage, the heuristic eventually guides the search around the obstacles.

HBSP, on the other hand, takes guidance from the user to determine which homotopy classes the heuristic functions should guide the search through. In our example, it was unclear to the user whether the robot could pass through the narrow passage between the couch and table. Thus, the user provided two reference paths: (i) around the obstacles and (ii) through the narrow passage (Fig. 1.b). While the path the footstep planner produced (Fig. 1.d), using both $\mathcal{H}_{\text{Dijk}}$ and the homotopy-based heuristics, is not the shortest path, there was approximately a **447 times** speedup in the planning time. It is also important to note that the performance of the planner is not hindered by poor quality heuristics. While both $\mathcal{H}_{\text{Dijk}}$ and the second homotopy-based heuristic sought to guide the search through the narrow passage, one informative heuristic quickly guided the search around the obstacles.

2 Related Work

In this section we describe related work on using search-based planning algorithms for footstep planning. In Sec. 2.1 we describe commonly-used search-based planning algorithms in the context of footstep planning. We also describe previous work on dynamically generating heuristics. In Sec. 2.2 we briefly mention how homotopy classes have been used in the broader context of motion planning.

2.1 Footstep Planning Using Search-Based Planning

Several approaches for footstep planning have been proposed over the last few years. Chestnutt *et al.* were the first to propose using A* to plan around and over planar obstacles (Chestnutt et al. 2005; 2007). However, the heuristic function used was not consistent, and thus could suffer from long planning time.

Indeed, as mentioned in Sec. 1 it can be extremely difficult to hand craft heuristics that can both efficiently guide the footstep planner and maintain guarantees on the quality of the solution obtained. Thus, one approach to speed up footstep planning without manually designing informative heuristics was to use anytime variants of A* with simple-to-define heuristics. This was done using D* lite (Garimort, Hornung, and Bennewitz 2011) or using ARA* and R* (Hornung et al. 2012)

These anytime algorithms sacrifice optimality for speed. An alternative approach to obtain efficient planners would be using informative heuristics. One approach to obtain such informative heuristics is applying different learning methods (Virseda, Borrajo, and Alcázar 2013; Arfaee, Zilles, and Holte 2011; Thayer, Dionne, and Ruml 2011; Bhardwaj, Choudhury, and Scherer 2017). While highly effective, this

[1] A heuristic function is said to be admissible if it never overestimates the cost of reaching the goal. A heuristic function is said to be consistent if its estimate is always less than or equal to the estimated distance from any neighboring vertex to the goal, plus the step cost of reaching that neighbor.

approach requires a large amount of training data and a good feature set for training.

2.2 Motion Planning Using Homotopy Classes

Homotopy classes have been frequently used to model the motion of a robot tethered to a fixed base point (Bhattacharya, Likhachev, and Kumar 2012; Grigoriev and Slissenko 1998; Salzman and Halperin 2015; Bhattacharya et al. 2015). The presence of obstacles introduces geometric and topological constrains for these robots. The constraints can create scenarios where the goal can only be reached if the cable configuration lies within a specific homotopy class, thereby making homotopy-based motion planning incredibly useful.

Homotopy classes have also been used in the context of human-robot interaction where a human wishes to restrict a robot's motion to specific homotopy classes (Yi, Goodrich, and Seppi 2016). For general approaches to explore and compute shortest paths in different homotopy classes, see (Bhattacharya 2010; Kim et al. 2013; Bhattacharya, Likhachev, and Kumar 2012).

3 Algorithmic Background

In this section we describe the algorithmic background necessary to understand our approach. In Sec. 3.1 we formally define the notion of homotopy classes and how to efficiently identify if two curves are in the same homotopy class—a procedure that will be used in our homotopy-based shortest-path algorithm (HBSP). In Sec. 3.2 we provide background on MHA* which we will use to compute footstep plans by incorporating the heuristics computed by HBSP.

3.1 Homotopy classes of curves

Informally, two continuous functions are called *homotopic* if one can be "continuously deformed" into the other (See Fig. 2a). In general, uniquely identifying the homotopy class of a curve is non-trivial; however, if both curves are embedded in the plane, a straightforward characteristic exists to identifying and computing the homotopy class of a curve (Armstrong 2013).

Specifically, let $\mathcal{W}_2 \subset \mathbb{R}^2$ be a subset of the plane (in our work this will be a two-dimensional projection of the three-dimensional workspace where the robot moves) and let $\mathcal{O} = \{\mathcal{O}_1, ..., \mathcal{O}_m\}$ be a set of obstacles (in our work, these will be projections of the three-dimensional workspace obstacles).

In order to identify if two curves $\gamma_1, \gamma_2 \in \mathcal{W}_2 \setminus \mathcal{O}$ that share the same endpoints are homotopic we use the notion of *h-signature* (see (Grigoriev and Slissenko 1998; Bhattacharya, Likhachev, and Kumar 2012; Salzman and Halperin 2015)). The h-signature uniquely identifies the homotopy class of a curve. That is, γ_1 and γ_2 have identical reduced words if and only if they are homotopic.

In order to define the h-signature, we choose a point $p_k \in \mathcal{O}_k$ in each obstacle such that no two points share the same x-coordinate. We then extend a vertical ray or "beam" b_k towards $y = +\infty$ from p_k. Finally, we associate a letter t_k with beam b_k (See Fig. 2b).

Now, given γ, let $b_{k_1}, \ldots, b_{k_m}$ be the sequence of m beams crossed when tracing γ from start to end. The *signature* of γ, denoted by $s(\gamma)$, is a sequence of m letters. If γ is intersected by the beam b_k, by crossing it from left to right (right to left), then the i'th letter is t_k ($\bar{t_k}$, respectively). The reduced word, denoted by $r(s(\gamma))$, is constructed by eliminating a pair of consecutive letters in the form of $t_k\bar{t_k}$ or $\bar{t_k}t_k$. The reduced word $r(s(\gamma))$ is a *homotopy invariant* for curves with fixed endpoints. It will be denoted as $h(\gamma) = r(s(\gamma))$ and called the *h-signature* of γ.

As we will see, our search algorithm HBSP will incrementally construct paths. After they are fully constructed, they will be in the same homotopy class as a given reference path. Thus, it will be useful to understand how the h-signature of a curve γ, which is a concatenation of two curves γ_1, γ_2 can be easily constructed. This reduced signature of $\gamma = \gamma_1 \cdot \gamma_2$ is simply the reduced signature of the concatenation of two curves' signatures $h(\gamma) = r(s(\gamma_1) \cdot s(\gamma_2))$.

3.2 Multi-Heuristic A* (MHA*)

The performance of heuristic search-based planners, such as A*, depends heavily on the quality of the heuristic function. For many domains, it is difficult to produce a single heuristic function that captures all the complexities of the environment. Furthermore, it is difficult to produce an admissible heuristic which is a necessary condition for providing guarantees on solution quality and completeness.

One approach to cope with these challenges is by using *multiple* heuristic functions. MHA* (Aine et al. 2014; 2016) is one such approach that takes in a single admissible heuristic called the *anchor* heuristic, as well as multiple (possibly) inadmissible heuristics. It then simultaneously runs multiple A*-like searches, one associated with each heuristic, which allows to automatically combine the guiding powers of the different heuristics in different stages of the search.

Aine *et al.* (Aine et al. 2014; 2016) describe two variants of MHA*, Independent Multi-Heuristic A* (IMHA*) and Shared Multi-Heuristic A* (SMHA*). Both of these variants gurantee completeness and provide bounds on suboptimality. In IMHA* each individual search runs independently of the other searches while in SMHA*, the best path to reach each state in the search space is shared among all searches. This allows each individual search to benefit from progress made by other searches. This also allows SMHA* to combine partial paths found by different searches which, in many cases, makes SMHA* more powerful than IMHA*. Therefore in this work we will use SMHA*. For brevity we will refer to SMHA* as MHA*.

4 Homotopy-Based Footstep Planning

Our footstep planner is comprised of HBSP, which generates the heuristic functions from a set of user-defined homotopy classes, and MHA*, which simultaneously uses these heuristics to find a feasible path. We begin by defining a taxonomy of the different spaces we consider (Sec. 4.1). We then detail our footstep planner (Sec. 4.2) and HBSP (Sec. 4.3).

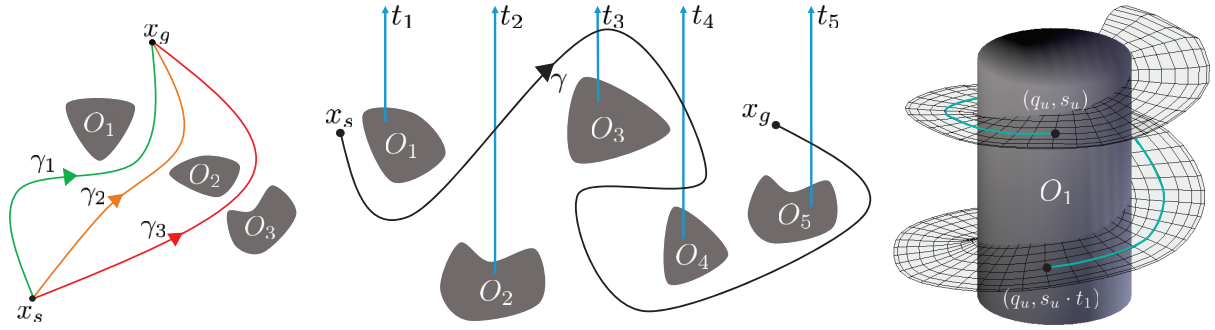


Figure 2: (a) γ_1 and γ_2 are in the same homotopy class, however, γ_3 is in a different homotopy class because of obstacle O_2. (b) The signature for this curve is $t_2 t_3 t_4 \bar{t}_4 \bar{t}_5$. The homotopy invariant of *h-signature* of curve γ is $t_2 t_3 \bar{t}_5$. (c) $\mathcal{G}^h_{\mathcal{W}_2}$ has vertices (q_u, s_u) and $(q_u, s_u \cdot t_1)$ that have the same configuration q_u but different signatures s_u and $s_u \cdot t_1$.

4.1 Taxonomy of Search Spaces

Search spaces Let $\mathcal{W}_3 \subset \mathbb{R}^3$ be the three-dimensional workspace in which the robot operates and $\mathcal{W}_2 \subset \mathbb{R}^2$ be its two-dimensional projection. Let $\mathcal{X}_{\text{robot}}$ and $\mathcal{X}_{\text{footstep}}$ be the configuration spaces of the humanoid robot[2] and the robot's footsteps, respectively. Specifically, $\mathcal{X}_{\text{robot}}$ is high-dimensional (over several dozens of dimensions), while $\mathcal{X}_{\text{footstep}}$ is a (relatively) low-dimensional space. In this work $\mathcal{X}_{\text{footstep}}$ is the six-dimensional space $SE(2) \times SE(2)$ denoting the position and orientation of each of the robot's feet. Let $\mathcal{M} : \mathcal{X}_{\text{footstep}} \to \mathcal{W}_2$ be a mapping projecting footstep configurations to the workspace's projection. In this work we use the mapping $\mathcal{M}(x_1, y_1, \theta_1, x_2, y_2, \theta_2) = ((x_1 + x_2)/2, (y_1 + y_2)/2)$. Finally, we assume that each space $\mathcal{X}$ induces a graph $\mathcal{G}_{\mathcal{X}} = (\mathcal{V}_{\mathcal{X}}, \mathcal{E}_{\mathcal{X}})$ embedded in $\mathcal{X}$. For example, the vertices can be defined by overlaying the space $\mathcal{X}$ with a grid and edges connecting every two nearby vertices.

Augmented graphs In this work, we use homotopy classes to guide our footstep planner. Thus, we will use the notion of *augmented graphs* which, for a given graph $\mathcal{G}$ and a goal vertex u_{goal}, capture the different homotopy classes to reach every vertex in $\mathcal{G}$ from u_{goal}. To define the augmented graphs, we first need to define the *signature set* $S(O)$ of a set of obstacles O. Let $B(O)$ be the vertical beams associated with the obstacles in O. The signature set is defined as all the different h-signatures that can be constructed using $B(O)$. Note that $S(O)$ is a countably infinite set.

The first graph we will augment is $\mathcal{G}_{\mathcal{W}_2}$. Let $\mathcal{G}^h_{\mathcal{W}_2} = \left(\mathcal{V}^h_{\mathcal{W}_2}, \mathcal{E}^h_{\mathcal{W}_2}\right)$ denote this augmented graph induced by the projected workspace $\mathcal{W}_2$. The set of vertices is defined as $\mathcal{V}^h_{\mathcal{W}_2} = \mathcal{V}_{\mathcal{W}_2} \times S(O)$. Namely, it consists of all pairs (q, s) where q is a vertex in $\mathcal{V}_{\mathcal{W}_2}$ and $s \in S(O)$ is a signature. The set of edges is defined as

$$\mathcal{E}^h_{\mathcal{W}_2} = \{((q_u, s_u), (q_v, s_v)) \,|\, (q_u, q_v) \in \mathcal{E}_{\mathcal{W}_2} \text{ and } h(s_u \cdot s^{\mathcal{W}_2}_{u,v}) = h(s_v)\}.$$

Here $s^{\mathcal{W}_2}_{u,v}$ is the signature of the trajectory in $\mathcal{W}_2$ associated with the edge (q_u, q_v). Namely, $\mathcal{E}^h_{\mathcal{W}_2}$ consists of all edges (u, v) connecting vertices such that (i) there is an edge in $\mathcal{E}_{\mathcal{W}_2}$ between the q_u and q_v and (ii) the reduced signature obtained by concatenating s_u with the signature of the trajectory associated with the edge (q_u, q_v) yields s_v. It is important to note that $\mathcal{G}^h_{\mathcal{W}_2}$ can have vertices that have the same configuration $q \in \mathcal{V}_{\mathcal{W}_2}$ but with different signatures (see Fig. 2c).

Similar to $\mathcal{G}^h_{\mathcal{W}_2}$, the augmented graph induced by $\mathcal{X}_{\text{footstep}}$ is denoted by $\mathcal{G}^h_{\mathcal{X}_{\text{footstep}}} = \left(\mathcal{V}^h_{\mathcal{X}_{\text{footstep}}}, \mathcal{E}^h_{\mathcal{X}_{\text{footstep}}}\right)$. Here, the set of vertices is defined as $\mathcal{V}^h_{\mathcal{X}_{\text{footstep}}} = \mathcal{V}_{\mathcal{X}_{\text{footstep}}} \times S(O)$ which is analogous to the definition of $\mathcal{V}^h_{\mathcal{W}_2}$. The set of edges is slightly more complex because $\mathcal{G}^h_{\mathcal{X}_{\text{footstep}}}$ is not embedded in the plane. Specifically, $\mathcal{E}^h_{\mathcal{X}_{\text{footstep}}}$ is defined as

$$\mathcal{E}^h_{\mathcal{X}_{\text{footstep}}} = \{((q_u, s_u), (q_v, s_v)) \,|\, (q_u, q_v) \in \mathcal{E}_{\mathcal{X}_{\text{footstep}}} \text{ and } h(s_u \cdot s^{\mathcal{X}_{\text{footstep}}}_{u,v}) = h(s_v)\}.$$

Here $s^{\mathcal{X}_{\text{footstep}}}_{u,v}$ is the signature of the trajectory in $\mathcal{W}_2$ associated with the projection of the edge (q_u, q_v). Namely, let $\tilde{q}_u = \mathcal{M}(q_u)$ and $\tilde{q}_v = \mathcal{M}(q_v)$ be the projections of the two footstep vertices q_u and q_v, respectively and similarly let $(\tilde{q}_u, \tilde{q}_v)$ denote the projection of the trajectory associated with edge (q_u, q_v). Signature $s^{\mathcal{X}_{\text{footstep}}}_{u,v}$ is simply the signature of $(\tilde{q}_u, \tilde{q}_v)$.

Heuristic functions A heuristic function for the footstep planner is a mapping $\mathcal{H} : \mathcal{V}^h_{\mathcal{X}_{\text{footstep}}} \to \mathbb{R}_{\geq 0}$. In this work, we use the projected workspace $\mathcal{W}_2$ to guide the footstep planner. Specifically, we use $\mathcal{W}_2$ with no homotopy-based information to obtain a simple heuristic $\mathcal{H}_{\text{Dijk}}$ as well as a set of heuristics defined using a set of homotopy classes.

[2]Planning in $\mathcal{X}_{\text{robot}}$ is out of the scope of this paper. We mention this space to provide the reader with a complete picture of all the search spaces relevant to planning the motion of a humanoid robot.

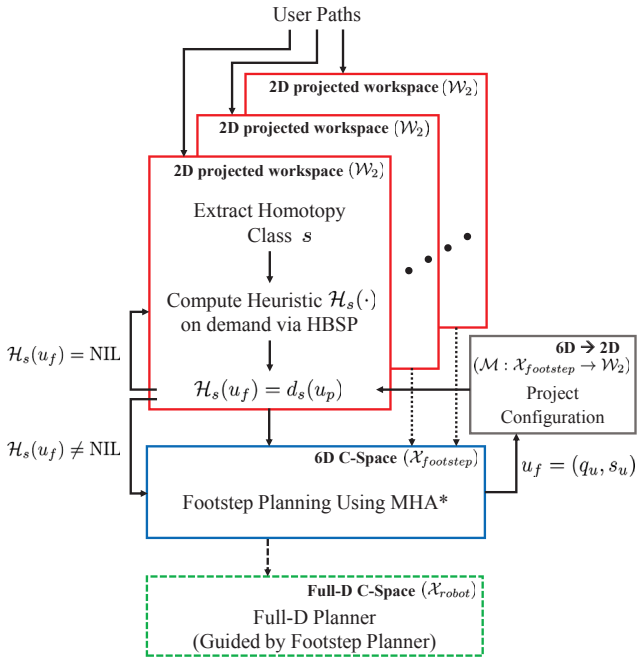


Figure 3: Overview of algorithmic approach for footstep planning.

The heuristic $\mathcal{H}_{\text{Dijk}}$ is obtained by running a Dijkstra search from the projection of the goal configuration to every vertex in $\mathcal{G}_{\mathcal{W}_2}$. This heuristic is often referred to as a "backward 2D Dijkstra heuristic".

Our homotopy-based heuristics are defined using mappings $d_s : \mathcal{V}^h_{\mathcal{W}_2} \to \mathbb{R}_{\geq 0}$ (one for each signature $s \in S$). Each mapping associates distances with augmented vertices of the projected workspace. Here, a heuristic function will be defined as $\mathcal{H}_s(q_u, s_u) = d_s(\mathcal{M}(q_u), s_u)$ and $d_s(u)$ is the shortest path to reach the goal from vertex u by following the homotopy path defined by s.

4.2 Algorithmic approach

In this section we describe our algorithmic approach for footstep planning which is depicted in Fig. 3. The algorithm starts by obtaining a set of user-defined homotopy classes in the projected workspace $\mathcal{W}_2$ using an intuitive graphical user interface (see Fig. 4). Each homotopy class is represented by a signature s. Let S represent the set of all such signatures. Recall that each signature $s \in S$ is used to compute a heuristic $\mathcal{H}_s$ by computing a distance function d_s using our Homotopy-Based Shortest Path (HBSP) algorithm (Fig. 3, red boxes).

Our footstep planner (Fig. 3, blue box) runs a MHA*-search over the augmented graph $\mathcal{G}^h_{\mathcal{X}_{\text{footstep}}}$ guided by the set of heuristics $\{\mathcal{H}_s | s \in S\}$ as well as the anchor heuristic $\mathcal{H}_{\text{Dijk}}$. Since $\{\mathcal{H}_s | s \in S\}$ is constructed based on user-defined homotopy classes, it is possible for the user to provide homotopy classes such that every heuristic biases the search towards regions where there is no feasible path. However, since we are using MHA*, the algorithm is complete and the user cannot prevent a path from being found. Futhermore, since our anchor heuristic is admissible, we maintain guarantees on suboptimality.

It is important to note that the distance functions $\{d_s | s \in S\}$ are *not* pre-computed for every vertex $u \in \mathcal{V}^h_{\mathcal{W}_2}$ by HBSP (this would be infeasible as $\mathcal{V}^h_{\mathcal{W}_2}$ is not a finite set). Instead, they are computed in an on-demand fashion. Specifically, given a vertex $u \in \mathcal{V}^h_{\mathcal{W}_2}$ and a signature s, HBSP checks if $d_s(u)$ has been computed. If it has, the value is returned and if not the algorithm continues to run a search from its previous state until $d_s(u)$ has been computed. This process is depicted in Fig. 3 by the two different edges leaving the red box to its left.

4.3 Homotopy-Based Shortest Path

We now describe our algorithm for computing homotopy-based shortest paths, or HBSP. Given a goal configuration q_{goal} and the graph $\mathcal{G}_{\mathcal{W}_2}$, HBSP incrementally constructs the augmented graph $\mathcal{G}^h_{\mathcal{W}_2}$ by running a variant of Dijkstra's algorithm from the vertex $(q_{\text{goal}}, \wedge)$. Here, $\wedge$ denotes the empty signature. For all vertices in $\mathcal{V}^h_{\mathcal{W}_2}$ that were constructed, the algorithm maintains a map $dist : \mathcal{V}^h_{\mathcal{W}_2} \to \mathcal{R}_{\geq 0}$ which captures the cost of the shortest path to reach vertices in $\mathcal{V}^h_{\mathcal{W}_2}$ from the vertex $(q_{\text{goal}}, \wedge)$. Given a vertex $(q_u, s_u) \in \mathcal{G}^h_{\mathcal{W}_2}$ and some user-defined signature s, this map $dist$ is used to compute the mapping d_s (which, in turn, is required to compute the heuristic function $\mathcal{H}_s$). Specifically,

$$d_s(q_u, s_u) = dist[q_u, h(s \cdot s_u)]. \quad (1)$$

Note that s corresponds to a signature of the path defined from the vertex $(q_{\text{goal}}, \wedge)$ towards the vertex (q_{start}, s), where q_{start} is a projection of the robot's start configuration. However, s_u is computed by the search as it progresses from $(q_{\text{start}}, \wedge)$ to (q_{goal}, s). Therefore $h(s \cdot s_u)$ corresponds to the remaining portion of the homotopy-based path specified by s after we remove its prefix that corresponds to s_u. For example, let $s = \bar{t_3}\bar{t_2}\bar{t_1}$ and $s_u = t_1 t_2$, then $h(s \cdot s_u) = h(\bar{t_3}\bar{t_2}\bar{t_1}t_1t_2) = \bar{t_3}$. Here, $\bar{t_3}$ is the signature of remaining portion of the path to the goal specified by s.

As the graph $\mathcal{G}^h_{\mathcal{W}_2}$ contains an infinite number of vertices, two immediate questions come to mind regarding this Dijkstra's-like search:

Q.1 When should the search be terminated?

Q.2 Should the search attempt to explore all of $\mathcal{G}^h_{\mathcal{W}_2}$?

We address **Q.1** by only executing the search if it is queried for a value of d_s which has not been computed. Thus, the algorithm is also given a vertex u and runs the search only until $d_s(u)$ is computed. This approach turns HBSP to an online algorithm that produces results in a just-in-time fashion. It is important to note that when the search is terminated, its current state (namely its priority queue) is stored. When the algorithm continues its search, it is simply done from the last state encountered before it was previously terminated.

We address **Q.2** when computing the successors of a vertex described in Alg. 2 by restricting the vertices we expand.

Algorithm 1 Homotopy-Based Shortest Path Algorithm

```
1: function HBSP(Q, G_{W_2}, q_g, u, S)                    ▷ u = (q_u, s_u)
2:     if Q = ∅ and dist[(q_g, ∧)] = NIL then
3:         dist[(q_g, ∧)] ← 0
4:         Q.add_with_priority((q_g, ∧), 0)
5:     while Q ≠ ∅ do
6:         v ← Q.extract_min()                              ▷ v = (q_v, s_v)
7:         if v = u then
8:             return (Q, dist[u])
9:         V_succ ← succ(v, S, G_{W_2})
10:        for v' ∈ V_succ do                               ▷ v' = (q'_v, s_v)
11:            alt ← dist[v] + length(v, v')
12:            if dist[v'] = NIL then
13:                dist[v'] ← alt
14:                Q.add_with_priority(v', dist[v'])
15:            else if alt < dist[v'] then
16:                dist[v'] ← alt
17:                Q.decrease_priority(v, dist[v'])
18:    return (Q, ∞)
```

Algorithm 2 HBSP Successor Function

```
1: function succ(u, S, G_{W_2})                            ▷ u = (q_u, s_u)
2:     V_nbr ← neighbors(u)
3:     if S ≠ ∅ then
4:         S ← suffixes(S)
5:         for v ∈ V_nbr do                                 ▷ v = (q_v, s_{u,v})
6:             if valid(v) then
7:                 if h(s_u · s_{u,v}) ∉ S then
8:                     V_nbr.remove(v)
9:                 else
10:                    v = (q_v, s_u · s_{u,v})
11:            else
12:                V_nbr.remove(v)
13:    return V_nbr
```

During the search, when we expand a vertex $u \in \mathcal{V}^h_{\mathcal{W}_2}$ in our Dijkstra-like search, we prune away all its neighbors $v \in \mathcal{V}^h_{\mathcal{W}_2}$ that have invalid signatures $h(s_u \cdot s_{u,v})$ (Alg. 2, lines 7-10). We define a valid signature $h(s_u \cdot s_{u,v})$ as one that is a *suffix* of a signature $s \in S$. Let $\mathbb{S}$ be the collection of all such signatures. That is, any signature $h(s_u \cdot s_{u,v})$, such that s could potentially be reached as the search progresses. More specifically, these suffixes identify the order in which certain beams can be crossed to reach a signature $s \in S$. For example, in Fig. 2b, $S = \{t_2 t_3 \bar{t_5}\}$ and $\mathbb{S}$ of S is $\{t_2 t_3 \bar{t_5}, t_2 t_3, t_2, \wedge\}$. Here, $t_1 \notin \mathbb{S}$ as this beam does not need to be crossed to reach $t_2 t_3 \bar{t_5}$. Additionally, $t_3 t_2 \notin \mathbb{S}$ as the beams need to be crossed in the opposite order to obtain the signature $t_2 t_3 \bar{t_5}$.

The high-level description of our algorithm is captured in Alg. 1. The algorithm is identical to Dijkstra's algorithm[3] except that (i) when the cost $dist[u]$ is returned, the priority queue Q is also returned (Lines 7-8, 18) and (ii) the way the successors of an edge are computed (Line 9). Returning the queue Q allows the algorithm to be called in the future with Q in order to continue the search from the same state.

[3]The only places where HBSP differs from Dijkstra's algorithm are the lines highlighted in blue.

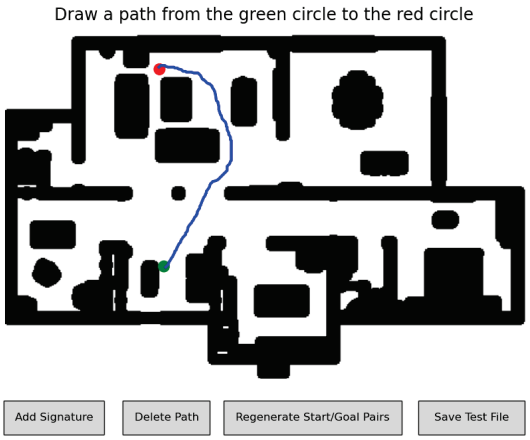


Figure 4: Example of interface to provide reference paths for constructing homotopy-based heuristics.

5 Experiments and Results

To test the capabilities of the footstep planner with various heuristics, we task a humanoid robot to plan footstep motions to a goal region in a house environment. We run our planner on 2 types of queries varying in their degree of complexity and evaluate the performance of our footstep planner by comparing the overall planning times (heuristic computation times and planning times) when using 3 different sets of heuristic functions:

S.1 One backward 2D Dijkstra heuristic $\mathcal{H}_{\text{Dijk}}$.

S.2 One backward 2D Dijkstra heuristic $\mathcal{H}_{\text{Dijk}}$ and one homotopy-based heuristic $\mathcal{H}_s$.

S.3 One backward 2D Dijkstra heuristic $\mathcal{H}_{\text{Dijk}}$ and three homotopy-based heuristics $\{\mathcal{H}_s | s \in \{s_1, s_2, s_3\}\}$.

5.1 Test Setup

We run our footstep planner with each set of heuristic functions on 20 *easy* and 20 *complex* queries. Each of the easy queries have randomly generated start and goal configurations. The path between these configurations in the easy queries are not obstructed by narrow passages. The complex queries, on the other hand, have hand-picked start and goal configurations and paths between these configurations are obstructed by at least one narrow passage that the robot may or may not be able to pass through.

To generate our homotopy-based heuristic functions we developed an interface that displays a projection of the inflated obstacles in the environment as well as the start (red circle) and goal configuration (green circle) (Fig. 4). The user can then draw paths from the goal to start configuration. All the signatures for the the reference paths are auto-

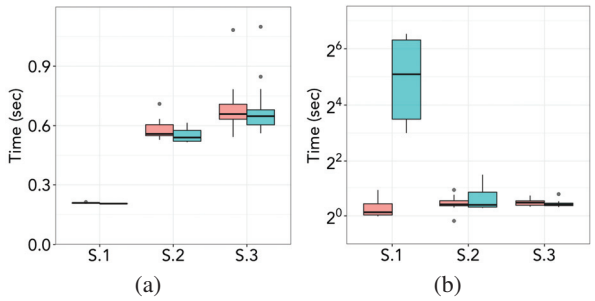


Figure 5: Box plots of heuristic computation time (a) and planning time (b) for the different sets of heuristic functions **S.1**-**S.3** for easy (●) and complex (●) queries. The middle line in the box plot represents the median. We can see (a) that computing heuristics for **S.2** and **S.3** incurs slightly more computation time when compared to **S.1**. However, this only slightly increases the planning time while for complex queries yields a speedup by several orders of magnitude (see also Fig. 6). Notice that y-axis (time) is in logarithmic scale.

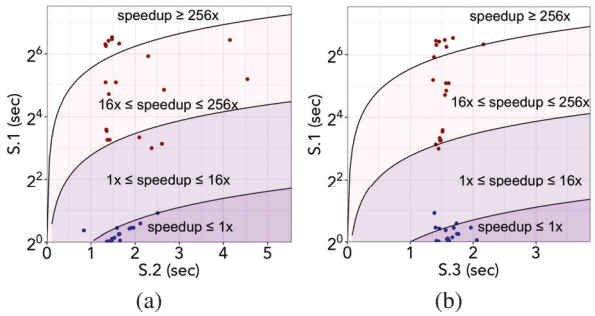


Figure 6: Speedup in total planning time (search time and heuristic computation) between **S.2** and **S.1** (a) and **S.3** and **S.1** (b). The running times for easy (●) queries is comparable among all algorithms (all points are close to the line speedup $\leq 1\times$). However, in complex (●) queries there is a speed up in planning times of **16 to more than 256 times** when using **S.2** and **S.3**. Notice that y-axis is in logarithmic scale.

matically generated and are used to compute the homotopy-based heuristic functions.

5.2 Easy Queries

In the easy queries the overall performance of the footstep planner for all sets of heuristic functions is comparable. The time taken to compute the homotopy-based heuristic functions is slightly longer than that of $\mathcal{H}_{\text{Dijk}}$. Therefore, the overall planning time while using **S.2** or **S.3** is either similar to or slightly less than the planning time when using **S.1** (See Fig. 5).

5.3 Complex Queries

In the complex queries there was a speedup in planning of several orders of magnitude when using **S.2** or **S.3** in comparison to **S.1**. While it took longer to compute **S.2** and **S.3**, these times were effectively negligible when comparing the planning times to when **S.1** was used (See Fig. 5).

Additionally, when using **S.2** there were several scenarios where there was a **16 to 256 times** speedup in planning times and a few scenarios with more than **256 times** speedup in planning times (Fig. 6.a). When using **S.3** there are many scenarios with far greater speedup in planning times in comparison to **S.2** (Fig. 6.b) . Furthermore, many of the heuristic functions in **S.3** were of poor quality, however, with at least one informative heuristic function for each scenario, the footstep planner was able to very quickly guide the search to the goal.

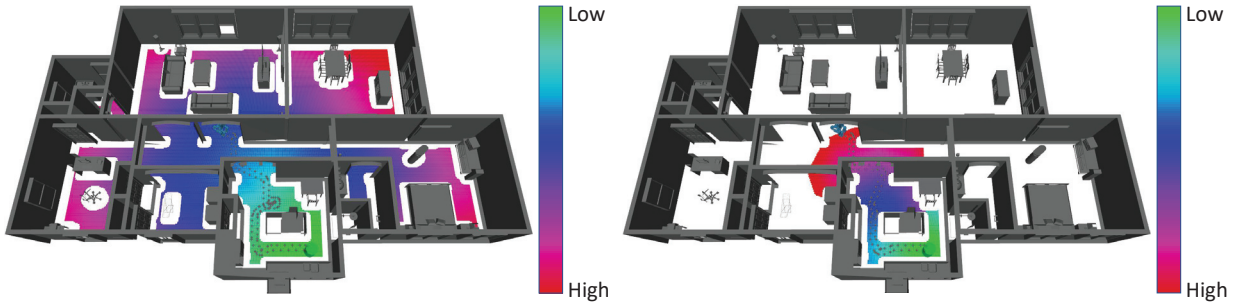
The footstep planner not only takes a very long time to find a path when using **S.1** but also produces very poor quality paths. In these scenarios, **S.1** attempts to guide the search through a narrow passage that the robot cannot pass through and expands near that region for a significant amount of time. Eventually, **S.1** guides the search around the obstacles and forms an unnecessary loop in the path (Fig. 7.a) while **S.2** circumvents the narrow passage and moves towards the goal (Fig. 7.b).

6 Discussion

We presented an approach for automatically generating heuristic functions given user-defined homotopy classes to effectively plan footstep motions for a humanoid robot. We showed that generating informative heuristics can significantly reduce planning times. Additionally, we presented an approach that allows users to easily construct these heuristics.

Our experiments showed that in *easy* queries the performance of the footstep planner, when using the heuristics generated through our approach, is comparable to that of the baseline approach. However, in *complex* queries we showed that when the footstep planner uses the heuristics generated through our approach, it plans several orders of magnitude faster than the baseline approach. Additionally, we showed that providing the footstep planner heuristic functions of poor quality does impede its performance. Furthermore, while this is out of the scope of our paper, our approach produced paths that the full-dimensional planner could use to guide its search (Gochev et al. 2011b). The baseline approach, on the other hand, produced very obscure paths that made unnecessary loops. These paths did not effectively guide the full-dimensional search.

While our results are promising, this work can be more cleverly applied within this domain. We believe there are three promising directions that may improve the quality of our footstep planner. First, we can automatically generate homotopy classes for our heuristic functions to allow for a fully autonomous planner. Second, we can use our approach only when the planner ceases to make progress towards the goal (Islam, Salzman, and Likhachev 2017). For example, in *easy* queries our approach does not improve planning times; it only has an impact on *complex* queries. Therefore we can



(a) (b)

Figure 7: The color map of the heuristic values and the path produced by the footstep planner when using **S.1** (a) and **S.2** (b). Notice (color map values) that the heuristic values for **S.1** drive the search to a narrow passage while the heuristics value for **S.2** circumvent it. Furthermore, notice that since HBSP is called on demand, the heuristic values for **S.2** were only computed for a small portion of the space.

utilize our approach only when the planner gets stuck to minimize planning and heuristic computation times. Finally, our approach is currently not applicable to scenarios where a robot may have to climb stairs or ladders as we can no longer project the workspace. We can extend this approach to make it applicable to a variety of complex environments.

Acknowledgments

This work was supported by National Science Foundation Grants IIS-1659774 and IIS-1409549.

References

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2014. Multi-heuristic A. In *Robotics: Science and Systems*.

Aine, S.; Swaminathan, S.; Narayanan, V.; Hwang, V.; and Likhachev, M. 2016. Multi-heuristic A*. *The International Journal of Robotics Research* 35(1-3):224–243.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence* 175(16-17):2075–2098.

Armstrong, M. A. 2013. *Basic topology*. Springer Science & Business Media.

Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. Learning heuristic search via imitation. *CoRR* abs/1707.03034.

Bhattacharya, S.; Kim, S.; Heidarsson, H. K.; Sukhatme, G. S.; and Kumar, V. 2015. A topological approach to using cables to separate and manipulate sets of objects. *The International Journal of Robotics Research* 34(6):799–815.

Bhattacharya, S.; Likhachev, M.; and Kumar, V. 2012. Topological constraints in search-based robot path planning. *Autonomous Robots* 33(3):273–290.

Bhattacharya, S. 2010. Search-based path planning with homotopy class constraints. In *AAAI Conference on Artificial Intelligence*.

Chestnutt, J. E.; Lau, M.; Cheung, G. K. M.; Kuffner, J.; Hodgins, J. K.; and Kanade, T. 2005. Footstep planning for the honda ASIMO humanoid. In *IEEE International Conference on Robotics and Automation*, 629–634.

Chestnutt, J. E.; Nishiwaki, K.; Kuffner, J.; and Kagami, S. 2007. An adaptive action model for legged navigation planning. In *IEEE-RAS International Conference on Humanoid Robots*, 196–202.

Garimort, J.; Hornung, A.; and Bennewitz, M. 2011. Humanoid navigation with dynamic footstep plans. In *IEEE International Conference on Robotics and Automation*, 3982–3987.

Gochev, K.; Cohen, B. J.; Butzke, J.; Safonova, A.; and Likhachev, M. 2011a. Path planning with adaptive dimensionality. In *SOCS Symposium on Combinatorial Search*.

Gochev, K.; Cohen, B. J.; Butzke, J.; Safonova, A.; and Likhachev, M. 2011b. Path planning with adaptive dimensionality. In *Proceedings of the Fourth Annual Symposium on Combinatorial Search, SOCS 2011, Castell de Cardona, Barcelona, Spain, July 15.16, 2011*.

Grigoriev, D., and Slissenko, A. 1998. Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane. In *International symposium on Symbolic and algebraic computation*, 17–24.

Hornung, A.; Dornbush, A.; Likhachev, M.; and Bennewitz, M. 2012. Anytime search-based footstep planning with suboptimality bounds. In *IEEE-RAS International Conference on Humanoid Robots*, 674–679.

Ishida, T. 1992. Moving target search with intelligence. In *Proceedings of the 10th National Conference on Artificial Intelligence. San Jose, CA, July 12-16, 1992.*, 525–532.

Islam, F.; Salzman, O.; and Likhachev, M. 2017. Online, interactive user guidance for high-dimensional, constrained motion planning. *CoRR*.

Kajita, S.; Hirukawa, H.; Harada, K.; and Yokoi, K. 2014. *Introduction to humanoid robotics*, volume 101. Springer.

Kim, S.; Bhattacharya, S.; Ghrist, R.; and Kumar, V. 2013. Topological exploration of unknown and partially known environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3851–3858.

Salzman, O., and Halperin, D. 2015. Optimal motion planning for a tethered robot: Efficient preprocessing for fast shortest paths queries. In *IEEE International Conference on Robotics and Automation*, 4161–4166.

Thayer, J. T.; Dionne, A. J.; and Ruml, W. 2011. Learning inadmissible heuristics during search. In *International Conference on Automated Planning and Scheduling*.

Virseda, J.; Borrajo, D.; and Alcázar, V. 2013. Learning heuristic functions for cost-based planning. *Planning and Learning* 6.

Yi, D.; Goodrich, M. A.; and Seppi, K. D. 2016. Homotopy-aware rrt*: Toward human-robot topological path-planning. In *ACM/IEEE International Conference on Human-Robot Interaction*, 279–286.