Homotopy-Aware RRT* : Toward Human-Robot Topological Path-Planning

Daqing Yi*, Michael A. Goodrich† and Kevin D. Seppi‡
Computer Science Department, Brigham Young University, Provo, UT, USA
Email: *daqing.yi@byu.edu, †mike@cs.byu.edu, ‡kseppi@byu.edu

***Abstract*—An important problem in human-robot interaction is for a human to be able to tell the robot go to a particular location with instructions on how to get there or what to avoid on the way. This paper provides a solution to problems where the human wants the robot not only to optimize some objective but also to honor "soft" or "hard" topological constraints, i.e. "go quickly from A to B while avoiding C". The paper presents the HARRT* (homotopy-aware RRT*) algorithm, which is a computationally scalable algorithm that a robot can use to plan optimal paths subject to the information provided by the human. The paper provides a theoretic justification for the key property of the algorithm, proposes a heuristic for RRT*, and uses a set of simulation case studies of the resulting algorithm to make a case for why these properties are compatible with the requirements of human-robot interactive path-planning.**

I. Introduction

In search and rescue, police, and military applications of human-robot teaming, a human may want to tell a robot to go to a particular location while giving information that the robot should use to decide what path to take. This is a form of interaction that requires a robot to plan a path that honors the human's intent. In this paper, we focus on two useful elements of intent: the shape and quality of the planned path. Although algorithms exist to implement path properties like continuity and smoothness, there appears to be no algorithm that is computationally tractable while being powerful enough to honor both shape and quality aspects of a path. Such an algorithm would support human instructions like "go quickly around building A and then between the two trees while avoiding region C." In this paper, we present an algorithm that optimizes path quality while allowing a human to specify regions to avoid, preferences for directions of traveling around obstacles, via-point/waypoint constraints, and reference path constraints [1].

The contribution of this paper (a) is a computationally efficient algorithm for detecting when two paths are homotopic that (b) can be used as a heuristic for an RRT* planner to restrict search to a given homotopy class, where (c) the planning is done by the robot and (d) the optimization criteria and shape constraint is specified by the human. The specific contribution to human-robot interaction is that these properties, supported by simulation results, create a set of path affordances that allow a human to specify a wide range of shape constraints and objective preferences that the robot honors in path-planning.

The key to this algorithm is the topological concept of *homotopy*, which is a mathematical formalism of the inherent similarity or dissimilarity of two paths. Given two paths σ_1 and σ_2 with the same endpoints, if one can be continuously deformed into the other without encroaching any obstacle and without moving the endpoints then they are said to be *homotopic* [2], [3]. We write this as $\sigma_1 \simeq \sigma_2$. In a slight abuse of notation, we say that two sets of paths are homotopic $\Gamma_1 \simeq \Gamma_2$ if all paths in the sets are homotopic.

We restrict attention to paths that start at a given initial position x_{init} and end at a given terminal position x_{goal}, and group the set of all such possible paths into classes according to their shape properties. Formally, the set of paths that are homotopic to each other form a *homotopy class*, and the set of homotopy classes partition the set of all possible paths between any two positions x_{init} and x_{goal}. In an environment containing obstacles, we argue that this homotopy partition, provides a mapping between a human-based or colloquial use of the term "shape" and the corresponding topological notion.

Theoretically, there exist infinite homotopy classes, because an obstacle can be encircled an arbitrary number of times. With some loss of generality, we impose the following:

Restriction 1. *We consider only "simple paths", that is, paths that do not form complete loops around obstacles.*

We can now define the problem the algorithm must solve.

Definition 1. ***Homotopy-Based Optimal Path-Plan-ning*** *Let $X \subset \mathbb{R}^d$ denote a bounded connected open set, $X_{obs} \subset X$ an obstacle space, $X_{free} = X \setminus X_{obs}$ the obstacle-free space, x_{init} an initial state , and x_{goal} a goal state. Define a* path *in X as a continuous curve parameterized by s as $\sigma(s) : [0,1] \to X$. Denote the monotonic increasing cost of the path as* $\textsc{Cost}(\sigma)$. *Let $\boldsymbol{H}(x_{init}, x_{goal})$ denote the set of homotopy classes defined by $x_{init} \in X_{free}$ and $x_{goal} \in X_{free}$, $H = h_1, \cdots, h_N \subseteq \boldsymbol{H}$ a particular subset of homotopy classes, and $h(\sigma)$ the homotopy class of σ. The goal is to find paths $\sigma^*_{h_i} \in \Sigma^*, h_i \in H$ such that (a) $\forall s \in [0,1], \sigma^*(s)_{h_i} \in X_{free}$; (b) $\sigma^*_{h_i}(0) = x_{init}$ and $\sigma^*_{h_i}(1) = x_{goal}$; and (c) $\forall h_i \in H, \sigma^*_{h_i} = \arg\min_{\sigma \in X_{free} \wedge h(\sigma) = h_i} \textsc{Cost}(\sigma)$.*

Definition 1 says that, given a particular set of homotopy classes and a cost function, the planner should find paths that minimize the cost in the given homotopy classes.

II. Related Work

From the human side of the HRI problem, many researchers have noted that humans often represent the world using topo-

978-1-4673-8370-7/16/$31.00 © 2016 IEEE

logical rather than metric-based mental models [4]. Various methods have been used to create topological representations that can be used by path-planners for robots [5], [6], [7], [8]. Because these planners represent the relationships between landmarks as a graph and then plan paths using a graph-search algorithm, they satisfy strict topological constraints but do not minimize $\textsc{Cost}(\sigma)$.

Homotopy-based path-planning should enable a combination of constraints and continuous objectives, but determining the homotopic equivalence of two paths is usually computationally expensive or not general. For example, the Voronoi diagram is used to identify a path from any homotopy class in [9], but this algorithm has limitations when there are certain kinds of complex obstacles in the world. A so-called funnel algorithm in the universal covering space yields an improvement [10], but complexity does not scale well when obstacles are not smooth and convex.

Other algorithms for finding homotopic equivalence include (a) using semi-algebraic cuts to convert a candidate path into a "word" [11] and (b) converting a plane into a complex plane and then finding invariant properties of the paths in this plane [2]. Unfortunately, their performance depends on how the map is discretized; computation cost expands greatly if the obstacles are reasonably approximated by a high resolution discretization.

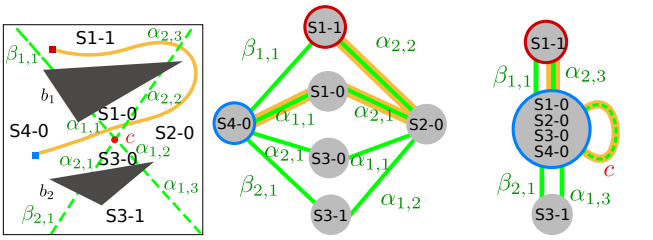
Homotopies have been used in sampling-based algorithms. In a probabilistic road map structure, the paths can be categorized into homotopy classes using a method called homotopic redundancy [12]. Another approach is to divide the space using a set of reference frames crossing each obstacle [13]. This method is particularly relevant because, as we shall show, how a path crosses the reference frames can be represented as a canonical sequence and comparing the sequences allows homotopic equivalence to be determined. We extend the ideas in these algorithms to enable optimal path-planning within a more complete set of homotopically equivalent paths. To accomplish this, we use a variation of the bidirectional RRT* algorithm [14], which is more efficient than the original [15].

III. Homotopic String Classes

This section shows how to create a string-based representation of any simple path, and then use the easy-to-compute strings to efficiently identify homotopic equivalence.

A. Generating String Representations

Strings are generated using an improved method of Jenkins' approach [16] to detecting homotopic equivalence of two paths by separating a map into disjoint subregions [17]. A reference frame segment, which Jenkins called a *reference frame*, is a line segment constructed from a center point and a point in a obstacle, extended to the map boundaries. The collection of reference frames created from a set of obstacles partition the map into disjoint subregions. Figure 1a shows an example of a map with two obstacles and two reference frames (blue and green dashed lines) – one for each obstacle. Strings will be constructed based on the simple idea that if two paths cross the same sequences of reference frames, then they belong to the same homotopy class.



(a) Ref. frames (b) State trans. (ST) (c) Homotopic ST

Fig. 1: Map with obstacles.

In Algorithm 1, the reference frames $\boldsymbol{R}$ are created from a set of points that are generated as follows: In a map with a set of obstacle regions $\boldsymbol{B}$, an obstacle point b_k is randomly sampled from each obstacle region $B_k \in \boldsymbol{B}$. A center point c is then randomly sampled in the non-obstacle region $\boldsymbol{X}_{free} = \boldsymbol{X} \setminus \boldsymbol{B}$ subject to the constraint that it is not in any line that connects two different b_k. Connecting each b_k with c creates a radial structure of reference frames that partition the map.

Algorithm 1 $\textsc{InitRefFrames}$ $(\boldsymbol{X}_{free}, \boldsymbol{B})$

$\boldsymbol{R} = \emptyset, \boldsymbol{b} = \emptyset$
for each $B_k \in \boldsymbol{B}$ **do**
  $\boldsymbol{b} \leftarrow \boldsymbol{b} \cup b_k$ randomly sampled from B_k
$c \leftarrow$ Randomly sampled from $\boldsymbol{X}_{free}$
while $\exists b_k, b_{k'}, c \in \textsc{Line}(\ b_k, b_{k'}\ )$ **do**
  $c \leftarrow$ Randomly sampled from $\boldsymbol{X}_{free}$
for each $b_k \in \boldsymbol{b}$ **do**
  $l_k \leftarrow \textsc{Line}(\ b_k, c\ )$
  $\{l_{k_m}\} \leftarrow \textsc{Intersect}(l_k, \boldsymbol{B}, c)$
  $\boldsymbol{R} \leftarrow \boldsymbol{R} \cup \{l_{k_m}\}$
return $\boldsymbol{R}$

The method $\textsc{Line}(p_1, p_2)$ returns the line defined by p_1 and p_2, and the method $\textsc{Intersect}(r, \boldsymbol{B}, c)$ returns all segments of line r that don't intersect with an obstacle in $\boldsymbol{B}$ or the center point c.

If we assign an ID character to each reference frame, then how a path sequentially crosses the reference frames can be converted into a string of ID characters. For example, in Figure 1a, the path that starts in subregion S_{4-0} and ends in subregion S_{1-1} sequentially visits the reference frames $\alpha_{1,1}, \alpha_{2,2}, \alpha_{2,3}$. Concatenating these characters yields the path string $\alpha_{1,1}\alpha_{2,2}\alpha_{2,3}$. A deterministic finite automata (DFA) formalizes this process; see Figure 1b.

Definition 2. *Let $\boldsymbol{M} = (\boldsymbol{S}, \boldsymbol{R}, \delta, S_0, S_T)$ be a DFA that represents the string generation process from a path, where $\boldsymbol{S}$ is a set of subregions, $\boldsymbol{R}$ is a set of reference frames, $S_0 \in \boldsymbol{S}$ is the start subregion, $S_T \in \boldsymbol{S}$ is the end subregion, and $\delta : \boldsymbol{S} \times \boldsymbol{R} \to \boldsymbol{S}$ is the transition function that defines how one subregion transitions to another subregion by crossing one reference frame in $\boldsymbol{R}$. A string v is created as follows:*

- *v is initialized as an empty string ε.*

- *The path starts at $x_{init} \in S_0$ and ends at $x_{goal} \in S_T$.*
- *When there is a transition across a reference frame $r \in \boldsymbol{R}$, $v \leftarrow vr$.*

Thus, v is the string generated by a path through the map using $\boldsymbol{M}$. A *string block* Γ_v is the set of all paths that generate string v. We now develop conditions under which a set of string blocks partition the set of all simple paths into a set of disjoint homotopy classes. We present these as a series of properties, lemmas, and theorems.

Recall that we are restricting attention to simple paths, and let Γ denote the set of all simple paths. We begin with properties of paths and the strings that they generate through $\boldsymbol{M}$. Property 1 states that there are a finite number of unique strings generated by $\boldsymbol{M}$ that induce a partition over the Γ.

Property 1. *$\Gamma = \bigcup_{i=1}^{m} \Gamma_{v_i}$ and $v_i \neq v_j \Rightarrow \Gamma_{v_i} \cap \Gamma_{v_j} = \emptyset$.*

The second property is that $\simeq$ is an equivalence relation.

Property 2. *$\simeq$ in $\Gamma_{v_i} \simeq \Gamma_{v_j}$ is an equivalence relation.*

Property 3 states that two paths are homotopic when they belong to the same string block Γ_v. In other words, if σ_i and σ_j generate the same string v, σ_i and σ_j are homotopic.

Property 3. *$\forall \sigma_i, \sigma_j \in \Gamma_v, \sigma_i \simeq \sigma_j$.*

Property 4 states that if two touching paths σ_i and σ_j are concatenated together to form $\sigma_i \circ \sigma_j$ then $\boldsymbol{M}$ generates a string $v_i v_j$ that is the concatenation of the strings generated by the two individual paths v_i and v_j.

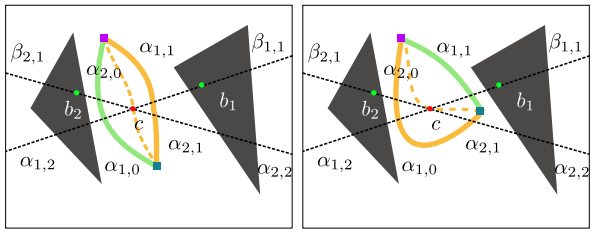
Property 4. *If $\sigma_i \in \Gamma_{v_i}, \sigma_j \in \Gamma_{v_j}$ and $\sigma_i(1) = \sigma_j(0)$ then $\sigma_i \circ \sigma_j \in \Gamma_{v_i v_j}$.*

To determine the homotopic equivalence of two paths that belong to different string blocks, we remove an ambiguity from $\boldsymbol{M}$. Observe that the reference frames form a radial structure emanating from the center point c. We denote the set of all reference frames segments between the center point c and an obstacle boundary by $\boldsymbol{R}_c$ and the set of subregions that connect with the center point c by $\boldsymbol{S}_c$.

Property 5. *A path segment that sequentially crosses several reference frames in $\boldsymbol{R}_c$ between two different subregions in $\boldsymbol{S}_c$ is homotopic to a path segment that crosses only the center point c.*

For example, in Figure 2a, two paths with different strings $\alpha_{2,0}\alpha_{1,0}$ and $\alpha_{1,1}\alpha_{2,1}$ indicate two paths in the same homotopy class. By Property 5, we have $\Gamma_{\alpha_{2,0}\alpha_{1,0}} \simeq \Gamma_c \simeq \Gamma_{\alpha_{1,1}\alpha_{2,1}}$. Figure 2b illustrates a second example.

An important consequence of Property 5 is that paths that only go through regions $\boldsymbol{S}_c$ are homotopic to a simple path segment that starts and ends at the same position within $\boldsymbol{S}_c$. Furthermore, all of these paths are homotopic to a path that generates the empty string. This means that we can merge all the subregions in $\boldsymbol{S}_c$ into a new subregion $\widehat{S_c}$. We can now create a new DFA, illustrated in Figure 1c, that removes the ambiguity associated with the center region.



(a) Example A (b) Example B

Fig. 2: Equivalence in Homotopy.

Definition 3. *Let $\boldsymbol{M}^h = (\boldsymbol{S}^h, \boldsymbol{R}^h, \delta^h, S_0^h, S_T^h)$ be a* homotopic DFA *that represents the string generation process from a path, where $\boldsymbol{S}^h = (\boldsymbol{S} \setminus \boldsymbol{S}_c) \cup \{\widehat{S_c}\}$ is a set of subregions, $\boldsymbol{R}^h = \boldsymbol{R} \setminus \boldsymbol{R}_c$ is a set of reference frames, $S_0^h \in \boldsymbol{S}^h$ is the start subregion, $S_T^h \in \boldsymbol{S}^h$ is the end subregion, and $\delta^h : \boldsymbol{S}^h \times \boldsymbol{R}^h \rightarrow \boldsymbol{S}^h$ is the transition function that defines how one subregion transitions to another subregion along the path by reference frames in $\boldsymbol{R}^h$. Strings are generated as follows:*

- *v is initialized as an empty string ε.*
- *The path starts at $x_{init} \in S_0^h$ and ends at $x_{goal} \in S_T^h$.*
- *When there is a transition across a reference frame $r \in \boldsymbol{R}^h$, $v \leftarrow vr$.*
- *When there is transition $\boldsymbol{R}_c$, $v \leftarrow v\varepsilon = v$.*

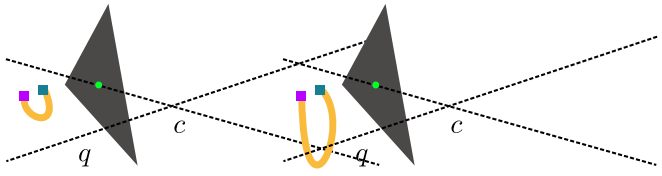
Now that we have removed this ambiguity, we observe an important relationship between a simple path and the string that $\boldsymbol{M}^h$ generates from this path. Let $v = \boldsymbol{M}^h(\sigma)$ denote the string generated from a path σ.

Property 6. *A duplicate ID character in a string $\boldsymbol{M}^h(\sigma)$ indicates that σ has visited a subregion at least twice.*

We call strings that don't have duplicate ID characters *non-repeating strings* v^*. Non-repeating strings can only be generated by simple paths that never leave a subregion by crossing a reference frame and then returning by recrossing that same reference frame. This implies the next property.

Property 7. *In every simple homotopy class there exists a path σ such that $\boldsymbol{M}^h(\sigma)$ has no duplicate characters.*

Consider a string constructed in the following manner: begin with the empty string ε and recursively insert a palindromic substring ww^R, where the R operator reverses the characters in the string, into any position of a string. We denote a string made up of recursively embedded palindromic substrings an *REP string*. Note that ε and strings of the form ww^R are REP strings.



(a) Before deformation (b) After deformation

Fig. 3: Path deformation.

We now present the culminating lemma of this subsection.

Lemma 1. *If σ is a simple path segment that begins and ends in the same subregion and encloses no obstacle, then $\boldsymbol{M}^h(\sigma)$ is a REP string.*

Proof. The proof is by induction on the number of subregions visited by a path.

- **Base case:** If a simple path segment σ never leaves a subregion then $\boldsymbol{M}^h(\sigma) = \varepsilon$.
- **Induction step:** Assume a simple path segment σ that begins and ends in the same subregion, and $\boldsymbol{M}^h(\sigma)$ is a REP string. Deform the path segment σ into a different simple segment σ' by crossing only one more reference frame with ID q, as illustrated in Figure 3. $\boldsymbol{M}^h(\sigma')$ is $\boldsymbol{M}^h(\sigma)$ embedded with qq, where $qq = qq^R$ is a palindromic substring. Thus, $\boldsymbol{M}^h(\sigma')$ is also a REP string.
- **Conclusion:** Any simple path segment σ that begins and ends in the same subregion can be obtained by recursively applying deformation to a simple path that never leaves a subregion in the inductive step.

□

A useful consequence of this Lemma is the following.

Corollary 1. *All simple paths that begin and end in the same subregion and enclose no obstacle are homotopic to each other and to a path σ such that $\boldsymbol{M}^h(\sigma) = \varepsilon$.*

B. Identifying the Equivalence

Having characterized several relationships between a path σ and its corresponding string $\boldsymbol{M}^h(\sigma)$, we now want to identify string properties that tell us when two paths are homotopic. Although Property 3 tells us that two paths are homotopic when they are in the same string block, we need more. Specifically, we also need to know when two paths from different string blocks are homotopic.

Write the set of all the simple paths as the union of several homotopy classes $\Gamma = \Gamma_{h_1} \cup \Gamma_{h_2} \cdots \cup \Gamma_{h_g}$, in which Γ_{h_i} is the set of all the paths in homotopy class h_i. By Property 1 and Property 3, we know that each homotopy class is a union of several string blocks, that is $\Gamma_{h_i} = \bigcup \Gamma_{v_i^j}$. Let V_i denote the set of strings v_i^j associated with the homotopy class h_i, and define $\Gamma_{V_i} = \bigcup \Gamma_{v_i^j}$.

Property 8 tells us that given a homotopy class h_i and its corresponding V_i, if $\boldsymbol{M}^h(\sigma) \in V_i$, then $\sigma \in \Gamma_{h_i}$, and vice versa.

Property 8. $\forall \Gamma_{h_i}, \exists V_i = \cup v_i^j, \Gamma_{h_i} = \Gamma_{V_i} = \cup \Gamma_{v_i^j}$.

This induces a hierarchy of path partitions and their associated string patterns, as illustrated in Figure 4.

This hierarchy tells us that we can identify the homotopy class of a path, $\sigma \in \Gamma_{h_i}$, by finding $\boldsymbol{M}^h(\sigma) \in V_i$.

We now show that we can use a non-repeating string v^* to determine whether $\boldsymbol{M}^h(\sigma_1)$ and $\boldsymbol{M}^h(\sigma_2)$ belong to the same

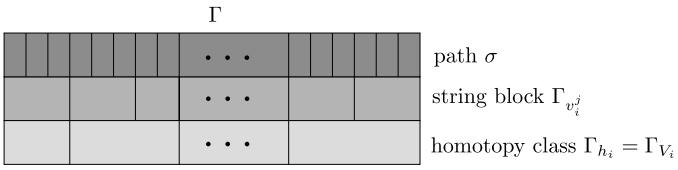


Fig. 4: A hierarchy of path partitions.

V_i. We begin with Lemma 2, which shows that every path is homotopic to a path that generates a non-repeating string.

Lemma 2. $\forall v, \exists v^*, \Gamma_v \simeq \Gamma_{v^*}$

Proof. Every path in a homotopy class is homotopic to the shortest path in that same homotopy class. By Property 7, the shortest path generates a non-repeating string v^*, which means $\Gamma_v \simeq \Gamma_{v^*}$. □

The next lemma states that any two different non-repeating string blocks are not homotopic.

Lemma 3. *If $v_i^* \neq v_j^*$ then $\Gamma_{v_i^*} \not\simeq \Gamma_{v_j^*}$.*

Proof. Assume that $\exists v_i^* \neq v_j^*$ such that $\Gamma_{v_i^*} \simeq \Gamma_{v_j^*}$. Let $\sigma_i \in \Gamma_{v_i^*}$ and $\sigma_j \in \Gamma_{v_j^*}$. Because σ_i and σ_j are homotopic to each other, we can assume without loss of generality that σ_j and σ_i end at the same point, $\sigma_i(1) = \sigma_j(1)$. Again because $\sigma_i \simeq \sigma_j$, the continuous path formed when we connect σ_i to $\sigma_j^R(s) = \sigma_j(1-s)$, a reversed version of σ_j, encloses no obstacle. By Lemma 1, $\boldsymbol{M}^h(\sigma_i \circ \sigma_j^R)$ is a REP string. Observe that $\boldsymbol{M}^h(\sigma_i \circ \sigma_j^R) = v_i^* v_j^{*R}$ by construction. The only way to cut a REP string v into two non-repeating strings is that the number of any character in the REP string v is two and the REP string v is palindromic. Cutting the palindromic string v in the middle gets v_1 and v_2, in which $v_1 = v_2^R$. But since $v_i^* \neq v_j^*$, then $v_i^* v_j^{*R}$ cannot be a REP string. □

This implies that each V_i associated with each homotopy class Γ_{h_i} contains only one non-repeating string.

Theorem 1 characterizes the relationship between V_i and its one and only non-repeating string v_i^*.

Theorem 1. *For all V_i there exists a unique non-repeating string $v_i^* \in V_i$, such that $\forall v_i^j \in V_i, \Gamma_{v_i^j} \simeq \Gamma_{v_i^*}$.*

Proof. Lemma 2 states that such a string much exist and Lemma 3 states that this string is unique. □

The next theorem gives us a construction by which we can identify the non-repeating string representative from for each V_i.

Theorem 2. *Removing all the REP substrings of $\boldsymbol{M}^h(\sigma)$ yields the v_i^* for which $\Gamma_{\boldsymbol{M}^h(\sigma)} \simeq \Gamma_{v_i^*}$.*

Proof. The differences between $\boldsymbol{M}^h(\sigma)$ and v_i^* are the REP substrings $\boldsymbol{M}^h(\sigma)$. The REP substrings are generated by adding a path segment that leaves a subregion by crossing one or more reference frames and then returning across the same reference frames in reverse order. By Lemma 1, this

path segment is homotopic to a path $\sigma \in \Gamma_\varepsilon$ (empty string). By Property 4, we have $\Gamma_{\boldsymbol{M}^h(\sigma)} \simeq \Gamma_{v_i^*}$. □

This gives us a powerful tool for finding when two strings represent homotopic or non-homotopic paths. Consider Algorithm 2, which removes REP substrings using the simple principle that if a character is on the top of the stack when you encounter the next one, you've found a REP substring and should eliminate it from the string.

Algorithm 2 REPTrim(v)

```
stack T = ∅
for char ∈ v do
    if TOP(T) == char then
        POP(T)
    else
        T ← char
return T
```

When combined with Theorem 2, this has the marvelous effect of finding a non-repeating string called REPTrim(v) such that $\Gamma_v \simeq \Gamma_{\text{REPTrim}(v)}$. This yields the following corollary.

Corollary 2. $\text{REPTrim}(\boldsymbol{M}^h(\sigma_i)) = \text{REPTrim}(\boldsymbol{M}^h(\sigma_j))$ *iff* $\sigma_i \simeq \sigma_j$.

IV. HOMOTOPY-AWARE RRT*

Corollary 2 gives us a useful way to determine whether two paths belong to the same homotopy class, but it doesn't tell us anything about how to construct the path that minimizes the cost objective within that homotopy class. This section uses a heuristic based on the REP trim algorithm to restrict paths generated by RRT* to a desired homotopy class. Future work will explore improved heuristics and better variations of RRT* using the explicit REP Trim algorithm.

RRT* explores the map to generate an optimal tree structure based on the cost distribution on the map. While the tree structure explores the planning space, the DFA $\boldsymbol{M}^h$ can be used to generate the strings of the branches. The string of each branch indicates the homotopic information of the corresponding subpath. The resulting algorithm, Algorithm 3, is called Homotopy-aware RRT* (HARRT*).

Algorithm 3 HARRT* (x_{init}, x_{goal})

```
i ← 0
N_s ← {x_init}; E_s ← ∅; T_s ← (N_s, E_s)
N_g ← {x_goal}; E_g ← ∅; T_g ← (N_g, E_g)
while i < N do
    T_s, x_s^new ← EXPLORE(T_s, i)
    T_g, x_g^new ← EXPLORE(T_g, i)
    p_s ← CONNECT(x_s^new, T_g)
    p_g ← CONNECT(x_g^new, T_s)
    P ← UPDATEBESTPATHBYCLASS(p_s, P)
    P ← UPDATEBESTPATHBYCLASS(p_g, P)
    i ← i + 1
P ← MERGEPATHS(P) return P
```

The algorithm uses a bi-directional structure. There is a *start tree* $T_s = (N_s, E_s)$, which is an RRT* structure from the start position for the optimal cost-to-arrive. N_s is the set of vertices in T_s, and E_s is the set of edges in T_s. Similarly, there is a *goal tree* $T_g = (N_g, E_g)$, which is an RRT* structure from the goal position for the optimal cost-to-go.

In each iteration, a new vertex is created and added to each tree using EXPLORE(). CONNECT() is then called to create a path with a vertex in the other tree. In order to guarantee optimality, a set of near vertices in T_g is provided to find the best vertex to be connected with the new vertex x_s^{new} in T_s, and vice versa. The created path will be compared with the current best path that belongs to the same string block. If it is a better one, the best path in this string block will be updated, which is implemented in UPDATEBESTPATHBYCLASS().

Algorithm 4 EXPLORE(T, i)

```
x_rand ← SAMPLE(i) ;
x_nearest ← NEAREST(T, x_rand)
x_new ← STEER(x_nearest, x_rand, η)
if OBSTACLEFREE(x_nearest, x_new) then
    s ← STR(x_nearest) ∘ CRF((x_nearest, x_new))
    if STRINGCHECK(s) then
        x_min ← x_nearest
        X_near ← NEAR(T, x_new, |N|)
        for each x_near ∈ X_near do
            if OBSTACLEFREE(x_new, x_near) then
                s ← STR(x_near) ∘ CRF((x_near, x_new))
                if STRINGCHECK(s) then
                    if COST(x_near) +c( LINE(x_near, x_new) ) < COST(x_new) then
                        x_min ← x_near
        E' ← E' ∪ {(x_min, x_new)}
        for each x_near ∈ X_near \ {x_min} do
            if OBSTACLEFREE(x_new, x_near) then
                s ← STR(x_new) ∘ CRF((x_new, x_near))
                if STRINGCHECK(s) then
                    if COST(x_near) > COST(x_new) + c(LINE(x_new, x_near)) then
                        x_parent ← PARENT(x_near)
                        E' ← E' \ {(x_parent, x_near)}
                        E' ← E' ∪ {(x_new, x_near)}
return T, x_new
```

Algorithm 4 gives the exploration process of a tree structure and is similar with that used in RRT* [18]. The first difference is that the string associated with each branch is updated (implementing $\boldsymbol{M}^h$ on that branch) and the second difference is the use of the STRINGCHECK() method (implementing a heuristic version of the REP Trim algorithm) to check whether the string of a branch satisfies the string constraint. The methods in Algorithm 4 are defined as follows:

- CRF(l): Return the ID characters that represent the crossed reference frames of a line segment l if any.
- STR(x): Return the string that represents the crossed reference frames of the subpath from the root to the node

x sequentially. This implements $\boldsymbol{M}^h$.

We assume that a human has specified one or more homotopy classes, and therefore string blocks, as the constraint of the planned paths. STRINGCHECK() compares whether a string of a subpath is a substring of the strings generated by the human. In effect, this eliminates branches that deviate from the non-repeating string representation of the human constraint. It is a heuristic because it does not test whether a branch has a REP substring but rather prevents RRT* from exploring branches that might have such substrings. Notice that the goal tree T_g compares the strings in a reversed order.

Because RRT* maintains a tree structure, each vertex has only one path to arrive from the root. This path, which starts from the root to the vertex, can be converted into a string of ID characters by $\boldsymbol{M}^h$. The STRINGCHECK() guarantees that a new node is added or rewired so that all the branches of the tree structure are in the constraint of strings. For example, suppose we have a string constraint "ab". A branch of the start tree T_s with string "a" satisfies the constraint, because "a" can be extended into "ab" by concatenating a "b". However, a branch beginning with string "b" cannot be extended into "ab", and therefore does not satisfy the string constraint. It is similar for the goal tree T_g but with reversed string order. Note that this is an early check of the homotopy class constraint and may eliminate some paths that would explore areas outside of the current subregion; we will say more about this in the results section.

Algorithm 5 CONNECT(x_{new}, T)

```
p_min = ∅
X_near ← NEAR(T, x_new, |N|)
for each x_near ∈ X_near do
    if OBSTACLEFREE(x_new, x_near) then
        if x_new ∈ T_s then
            p ← CONCATENATE(x_new, x_near)
        else
            p ← CONCATENATE(x_near, x_new)
        if STRINGCHECK(p) and c(p) < c(p_min) then
            p_min = p
return p_min
```

The methods in Algorithm 5 are defined as follows:

- PATH(v, T): Return the path from the root of the tree T to the vertex v.
- CONCATENATE(p_a, p_b): Return a concatenated path of p_a and p_b. If p_a and p_b are from different directions, one of them will be reversed for the concatenation.

When the exploration process is finished, there is a set of the best paths of all the string blocks. By the REP Trim algorithm we can merge the optimal paths in the string blocks that belong to the same homotopy class. The MERGEPATHS() merges the equivalent string blocks into homotopy classes. Thus, the set of paths P will be updated.

V. EXPERIMENTS

Recall the following claim from the introduction: "The contribution of this paper is an algorithm that has guaranteed properties ... [that] create a set of path affordances that allow a human to specify a wide range of hard and soft constraints that the robot is guaranteed to honor when it plans its path." To this point in the paper, all the text has focused on either the theoretic analysis of Palindrome Trim algorithm or the heuristic implementation of this algorithm to help RRT* restrict exploration to a given homotopy class. This section provides evidence that the algorithm can support an important need in HRI.

Consider a path-planning problem where a human supervisor defines the task for a robot. Consider further different ways in which the human can express hard and soft constraints as well as performance objectives. The first two examples of human intent translate into homotopic path constraints over an optimization problem. Note that we use the word "quickly" to represent the optimization criterion; in practice, many possible criteria exist.

1) *Quickly go from point A to point B through a sequence of specific regions.* Topologically, such a path is constrained to one homotopy class, so this homotopy class becomes the constraint of the optimization problem and "quickly" becomes the objective to optimize [10].
2) *Quickly go from point A to point B making sure to visit some regions and avoid other regions.* Topologically, such a path is constrained to be among the set of homotopy classes that include the desired regions and avoid the undesired regions. The corresponding homotopic constraint restricts the optimal path to the set of homotopy classes that satisfy the requirements.

The next example of human intent allow a human to express preference among different path shapes, but also allow the human to trade off between following a desired path shape and optimizing another performance objective.

3) *In quickly going from point A to point B, I prefer some types of paths over others, but I recognize that tradeoffs may be needed.* This indicates that the human has preferences over different homotopy classes, and also acknowledges that certain homotopy classes may not allow an acceptable optimization of another task-based objective. If the preference on the homotopy classes can be modeled using an objective function, then non-dominant solutions over the task-based objective (e.g., "quickly") and homotopic objective (e.g., "north of building A") can be found, and the human can select one of these solutions by balancing tradeoffs.

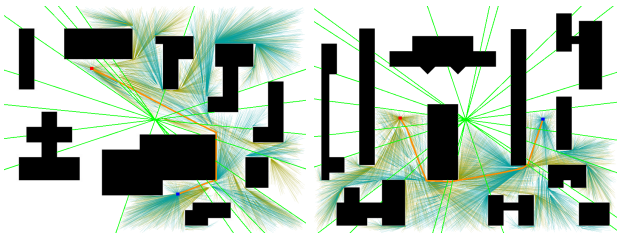
Clearly, these examples do not cover the set of all possible ways a human can express intent, but they do represent an important (and we would argue a natural) subset of ways that a human can express intent.

In the results in this section, we use the Euclidean distance as the objective to minimize because optimality can easily be verified. The objective can be replaced with any other type.

A. Single Homotopy Class

This subsection considers the first way of expressing intent: *Quickly go from point A to point B through a sequence of*

specific regions. In this case, the algorithm simply seeks to find the path that minimizes the Euclidean distance between two points subject to the path belonging to a homotopy class.



(a) Example A (b) Example B

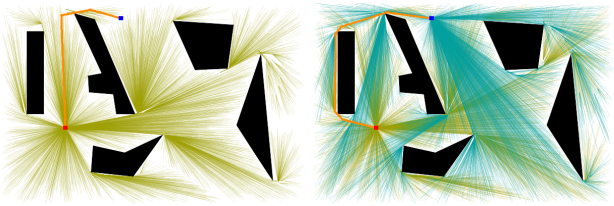
Fig. 5: Optimal paths with hard constraints.

Figure 5 shows results for two different worlds. In Example A, the authors sketched the fuzzy red path, turned that path into a single homotopy constraint, and then used the HARRT* algorithm to find the shortest path that connected the points subject to the constraint. The green radial lines indicate the reference frames, the yellowish lines indicate the branches of the forward tree of the RRT, the turqoise lines indicate the backward tree of the RRT, and the orange line represents the path found by the algorithm. Inspection shows that that the path is indeed the shortest path within the homotopy class. Further inspection shows that the algorithm concentrated its search effort in the homotopy class, abandoning branches of the tree either when costs became high or when the path crossed a reference frame that deviated from the non-repeating string pattern. Example B shows a similar result but with the human input suppressed to help improve clarity.

Figure 5 also illustrates that the current implementation of HARRT* is an approximation. The theoretic results show that we can determine when any two paths are in the same homotopy class by removing REP substrings, but where we check for homotopic consistency in the RRT implementation has a big impact on how big the trees grow in the algorithm. At one extreme, future work should explore whether it is possible (a) to check the homotopy class of the path returned by the RRT-based exploration once a complete path from start to finish has been found and (b) to reject paths that do not satisfy the homotopy constraint. In the current version of HARRT*, the StringCheck() method prevents the extension of the branches into a string that does match the non-repeating string. This is an "early check" on the path that can reject potentially optimal paths within a homotopy class if the optimal path would naturally generate some REP substrings.

As illustrated by the simple examples in this section, this "early check" avoids exploring a lot of the state space, resulting in search efficiency and producing acceptable paths, but it is possible to construct examples where this early check would prevent the discovery of the optimal path. Future work should explore variations of the StringCheck() method that allow a budgeted amount of deviation from a path that generates a non-repeating string. For example, the algorithm could allow a two character palindrome to be part of the string, allowing exploration of paths that leave a subregion to avoid an area of high cost and then return to the subregion once they have circumvented the high cost area. Certainly, this future work would need to explore tradeoffs in the deviation budget, the spacing of sample points that generate the radial structure, and the structure of the cost function.

We conclude this subsection with an example world that exposes a problem that is avoided by the bi-directional approach we have taken but which a single direction RRT encounters.



(a) Single directional tree (b) Bidirectional tree

Fig. 6: Search results from different structures.

Figure 6 gives the results from the single directional and bidirectional tree structures for a problem where multiple homotopy classes are allowed. Each tree structure finds an optimal path in each of several homotopy classes (though the optimal path is shown for only one homotopy class) optimal paths in several homotopy classes but only one of them is illustrated, but the single directional tree structure could not find an optimal path that swings by the left side of the left-most obstacle; see Figure 6a. By contrast, the via-point constraints in the bidirectional tree structure enforce the exploration of all the possible homotopy classes; see Figure 6b.

Although this result is encouraging, it exposes a challenge in combining RRT* with the Palindrome Trim algorithm or a related heuristic. Stated simply, it is possible for RRT* to "rewire" nodes in such a way that homotopy constraints are violated. Using bidirectional search fixed the problem for this world, but it is easy to construct other worlds for which bidirectional search won't work. Future work will address this challenge.

B. Multiple Homotopy Classes

This section considers the second way of expressing intent: *Quickly go from point A to point B making sure to visit some regions and avoid other regions.* In this case, the set of regions to visit and regions to avoid create a set of possible homotopy classes.

This section gives an example of the kinds of possible solutions that can be generated when multiple homotopy classes are explored simultaneously. Figure 7, which will be referenced again in the next section, shows the optimal solutions returned by HARRT* for six different homotopy classes. Observe that each path is optimal for the homotopy class given that the cost function is Euclidean distance. (As an aside, when the cost function is Euclidean distance the heuristic implementation of the REP trim algorithm won't cause problems since the shortest path is also a path that generates a non-repeating string.) For this second type of

human intent expression, the path with the lowest cost would be returned.

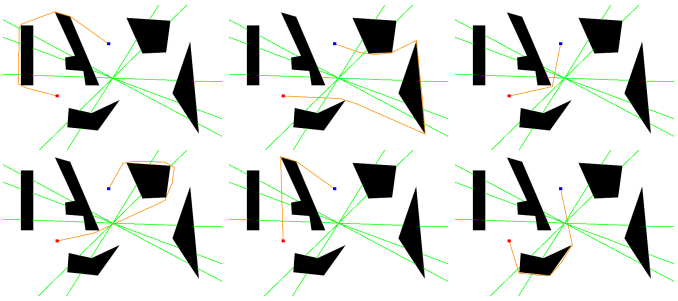


Fig. 7: Optimal paths in six homotopy classes.

C. Soft Constraint and Tradeoffs

This section considers the third way of expressing intent: *In going from point A to point B, I prefer some types of paths over others, but I recognize that tradeoffs may be needed.* This method is actually quite different than the two previous methods because it treats path shape not as a hard constraint but rather as an objective to be optimized. Fortunately, HARRT* can support a human trying to find tradeoffs between preferences among homotopy classes and the costs associated with choosing a path from a specific homotopy class.

The algorithm for finding these tradeoffs would use HARRT* to find the optimal path from the set of all relevant homotopy classes, as was done in the previous subsection. Recall that HARRT* can simultaneously search in different homotopy classes and returns the solutions in one run. The output of the algorithm would then change its what it presents to the human; rather than returning the best path across homotopy classes as in the previous subsection, the algorithm would find the minimum cost path for each homotopy class and then output both the path and the cost of the path for each homotopy class. This generates a tradeoff space that can be evaluated by the human.

For example, Figure 7 shows the optimal paths in six different homotopy classes found by HARRT*. Associated with each class/path pair is the cost of the optimal path. If the cost is displayed for each class/path pair, a human could determine which shape/cost pair provides the best tradeoff.

VI. Conclusion and Future Work

It is possible to create a computationally efficient algorithm that uses strings to determine when two continuous paths are homotopic. This algorithm can be used as a heuristic in a bi-directional RRT* algorithm to prune paths from the search that are not compatible with an intended homotopy class. Furthermore, the algorithm seems compatible with various degrees of approximation, allowing for tradeoffs between computation speed and quality of the path found by the algorithm, but future work must confirm this.

The paper did not explore the usability, workload, or naturalness of the interactions between the human and the robot afforded by the algorithm, but the empirical results suggest that efficient interactions are plausible given the properties of the algorithm. Future work should explore how natural language or a graphical user interface can be combined with the algorithm, and whether resulting interactions are compatible with human workload bounds and situation awareness needs.

Finally, future work should explore how the homotopy-aware RRT* algorithm can be extended to problems with multiple performance objectives, enabling either hard shape constraints or tradeoffs between a set of objectives and the shape of the resulting path.

References

[1] D. Yi, M. Goodrich, and K. Seppi, "Informative path planning with a human path constraint," in *Systems, Man and Cybernetics (SMC), 2014 IEEE International Conference on*, Oct 2014, pp. 1752–1758.

[2] S. Bhattacharya, "Search-based path planning with homotopy class constraints," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[3] A. Hatcher, "Algebraic topology. 2002," *Cambridge UP, Cambridge*, vol. 606, no. 9.

[4] B. Kuipers, "The spatial semantic hierarchy," Tech. Rep. AI99-281, 29, 1999.

[5] M. J. Mataric, "Integration of representation into goal-driven behavior-based robots," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 3, pp. 304–312, 1992.

[6] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998.

[7] J. Fasola and M. J. Mataric, "Modeling dynamic spatial relations with global properties for natural language-based human-robot interaction," in *RO-MAN, 2013 IEEE*. IEEE, 2013, pp. 453–460.

[8] D. C. Shah and M. E. Campbell, "A qualitative path planner for robot navigation using human-provided maps," *The International Journal of Robotics Research*, vol. 32, no. 13, pp. 1517–1535, 2013.

[9] B. Banerjee and B. Chandrasekaran, "A framework of voronoi diagram for planning multiple paths in free space," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 25, no. 4, pp. 457–475, 2013.

[10] J. Hershberger and J. Snoeyink, "Computing minimum length paths of a given homotopy class," *Computational Geometry*, vol. 4, no. 2, pp. 63 – 97, 1994.

[11] D. Grigoriev and A. Slissenko, "Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane," in *Proceedings of the 1998 International Symposium on Symbolic and Algebraic Computation*, ser. ISSAC '98. New York, NY, USA: ACM, 1998, pp. 17–24.

[12] E. Schmitzberger, J. Bouchet, M. Dufaut, D. Wolf, and R. Husson, "Capture of homotopy classes with probabilistic road map," in *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, vol. 3, 2002, pp. 2317–2322 vol.3.

[13] E. Hernandez, M. Carreras, and P. Ridao, "A comparison of homotopic path planning algorithms for robotic applications," *Robotics and Autonomous Systems*, vol. 64, no. 0, pp. 44 – 58, 2015.

[14] J. Starek, E. Schmerling, L. Janson, and M. Pavone, "Bidirectional fast marching trees: An optimal sampling-based algorithm for bidirectional motion planning," in *Workshop on Algorithmic Foundations of Robotics*, 2014.

[15] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2013-021, August 2013.

[16] K. D. Jenkins, "The shortest path problem in the plane with obstacles: A graph modeling approach to producing finite search lists of homotopy classes." Master's thesis, Naval Postgraduate School, Monterey, CA, June 1991.

[17] E. Hernandez, M. Carreras, J. Antich, P. Ridao, and A. Ortiz, "A topologically guided path planner for an AUV using homotopy classes," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2337–2343.

[18] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010.