5-1-2018

A Topological Approach to Workspace and Motion Planning for a Cable-controlled Robot in Cluttered Environments

Xiaolong Wang
Lehigh University, wangxiaolong0830@hotmail.com

A Topological Approach to Workspace and Motion Planning for a Cable-controlled Robot in Cluttered Environments

by

Xiaolong Wang

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Mechanical Engineering

Lehigh University

May 2018

Certification of Approval

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

_____

Date

_____

Prof. Subhrajit Bhattacharya

Thesis Advisor

_____

Prof. Gary Harlow

Chair of Department of

Mechanical Engineering and Mechanics

Acknowledgements

This thesis represents a joint work with my thesis advisor, Dr. Subhrajit Bhattacharya, Department of Mechanical Engineering and Mechanics, Lehigh University. The supervision and critical review of this study by Dr. Bhattacharya are sincerely appreciated and gratefully acknowledged. Appreciation is also extended to my cohorts, L. Zhang and S. Wang, for their help set up Linux programming environment.

Thanks are due to my father, Fuying Wang, for his valuable advice on my topology learning, and to my mother Xiaorong Ma and my sister Xiao Wang for their long support and encouragement.

Special thanks are due to my fiancee, Ying Wang, for her understanding and company all the time.

This thesis is generated by LaTeX editor/compiler WinEdt with MiKTeX. All the illustrations in this thesis are drawn by using Adobe Illustrator or processed by using Adobe Photoshop; the figures of test results are generated by C++ programs with OpenCV in Linux (Ubuntu).

Contents

List of Figures

Abstract

There is a rising demand for multiple-cable controlled robots in stadiums or warehouses due to its low cost, longer operation time, and higher safety standards. In a cluttered environment the cables can wrap around obstacles. Careful choice needs to be made for the initial cable configurations to ensure that the workspace of the robot is optimized. The presence of cables makes it imperative to consider the homotopy classes of the cables both in the design and motion planning problems. In this thesis we study the problem of workspace planning for multiple-cable controlled robots in an environment with polygonal obstacles. This goal of this thesis is to establish a relationship between the workspace's boundary and cable configurations of such robots, and solve related optimization and motion planning problems. We first analyze the conditions under which a configuration of a cable-controlled robot can be considered valid, then discuss the relationship between cable configuration, the robot's workspace and its motion state, and finally use graph search based motion planning in h-augmented graph to perform workspace optimization and to compute optimal paths for the robot. We demonstrated corresponding algorithms in simulations.*

*A partial of this thesis has been published in the same title as this thesis in the IEEE Robotics and Automation Letters (RA-L, Volume: 3, Issue: 3, Page: 2600-2607, July 2018).

Chapter 1

Introduction

1.1 Literature Review

Object manipulation is an important problem in robotics. Certainly conventional approaches to manipulation using robot arms with grippers have received considerable attention and are well understood [13]. In contrast, we are interested in the use of cable-controlled robots to contact and manipulate objects. Despite the advances in mobile and aerial robotics, there are various applications in which cable-controlled robots are better suited. Cable-controlled robots have recently attracted interest for large workspace manipulation tasks. The robotic system is controlled by varying-length cables, which are anchored to fixed *control points* and driven by effectors (motors), provides more agility (quickly move in large workspaces), greater reliability (less prone to environmental noise such as wind gusts [30] since the robot is tethered), more payload capability [21], has less onboard power consumption (since the actuation is done by the external cables [22]), does not rely on onboard sensors for localization and control (thus works in GPS-denied and featureless environments), and can be made in large-scale [1].

Robots attached to passive cables for locomotion, power supply and communication have been extensively used for many real-world applications [23, 24, 31] including industrial robotics (dock loading, construction, warehouse management), entertainment and security. Additionally, they have even demonstrated critical capabilities for monitoring terrestrial [18] and aquatic [16, 28, 29] environments. For such robots, the main challenge is to avoid entanglement of the cable with obstacles and to ensure that the tether does

not violate any geometric constraints [17, 19, 27, 32]. The use of cables to manipulate objects in an environment has also been studied extensively [6, 12, 14].

Active control of robots using cables, on the other hand, has gained relatively less attention in robotics literature. Some actuation systems focusing on agility [34] and accuracy [9] have been studied and others focused on controlling the robots' pose (position and orientation) and wrench (force and torque) [10, 11]. The typical controllers for such robots are designed for obstacle-free environments where the inverse-kinematics problem can be solved in a closed form [25, 33]. However, since in some circumstances the existence of obstacles could be inevitable, the problem of negotiating obstacles for a cable-controlled robot and demands prompt solution in both initial cable configuration and control point location. Because of the properties of cable that it can only pull but not push and that it cannot penetrate obstacles, this problem requires significant additional consideration. With the recent advent of topological path planning techniques [3, 6, 19], it has become possible to compute optimal solutions to motion planning problems for systems involving flexible cables by reasoning about topological classes (homotopy and homology classes) of paths and cables in a cluttered configuration space. This thesis uses these recent developments in the field of topological path planning to design algorithms for cable-controlled robots in environments with polygonal obstacles.

1.2 Problem Statement and Motivation

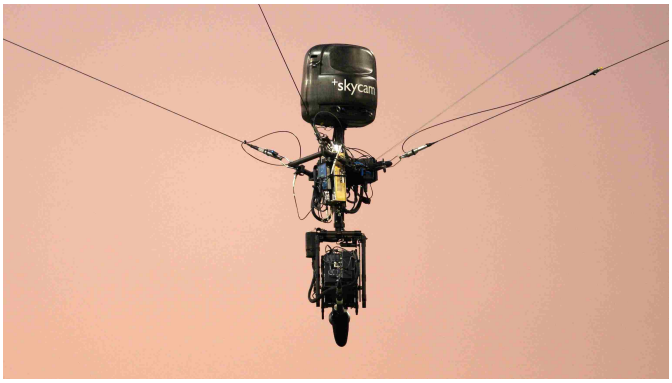
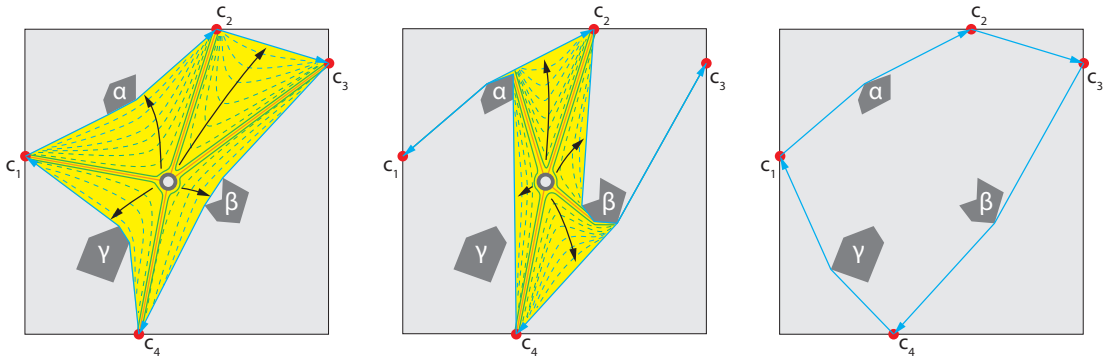


Figure 1.1: A wire camera (Skycam, source: Wikimedia Commons)

We consider a planar environment cluttered by polygonal obstacles. This is a model for robots that can be used to transport goods in a warehouse or to move overhead



(a) Original cables (orange solid) can be seen as a whole (green solid) which can deforming continuously into closed boundary (blue solid).

(b) Another initial configuration of cables. The cables can deform continuously into a valid workspace's boundary as well.

(c) An arbitrary combination of shortest paths between neighboring control points do not necessarily enclose a valid workspace.

Figure 1.2: Same environment, different workspaces enclosed by shortest paths in some homotopy classes.

cameras in a stadium (Figure 1.1)—attached and controlled by cables that are driven by motors at the boundaries of the environment (roof or walls). The obstacles which cables cannot penetrate would inevitably make some of the regions inaccessible to the robot. The initial cable configuration of the system influences the shape and size of the workspace (see the difference between Figure 1.2a and Figure 1.2b). It is thus important to choose the best cable configurations of optimizing workspace's area and ensuring that the robot is able to reach the desired locations. We need a method to search for a boundary of robot's workspace corresponding to its initial cable configuration. A related application is that of sea farming, where a net needs to be anchored at certain points (the control points) on its boundary (the workspace's boundary), ensuring that the net does not get tangled with obstacles, such as boats or buoys, while maximizing the area covered by the net (which is used for farming of marine species).

1.3 Outline of the Thesis

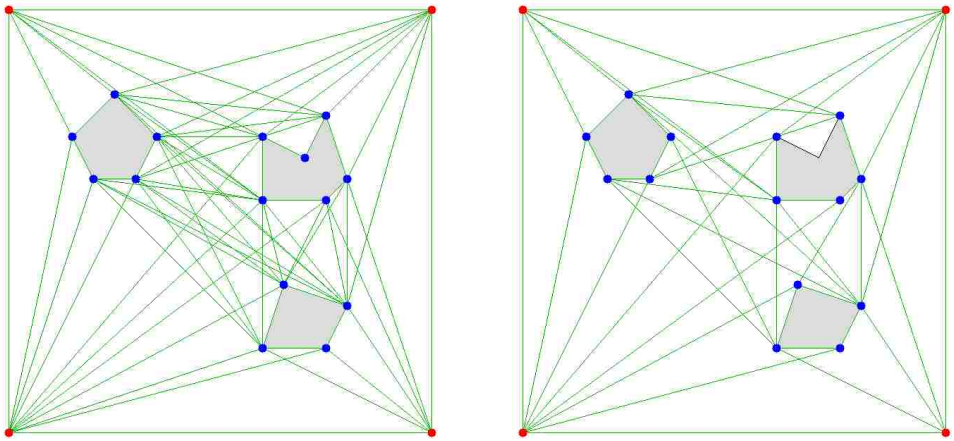
In this thesis, we start by presenting some of the preliminary backgrounds including visibility graph, homotopy/homology class and h-augmented graph. Following that we analyze the properties of the workspace of a multiple-cable controlled robot and its boundary, and propose an algorithm for computing the boundary for which workspace's

area is optimized or certain specific points fall within the workspace. Finally, we describe the algorithms for robot motion planning and cable velocity control and apply these to several example applications. This thesis comes with a supplementary video which contains simulations of valid workspace searching, point-to-point path planning with cable controlling within the boundary, and path planning for multiple-task accomplishment. Besides the University's official website for thesis/dissertation publication, it can also be accessed here: https://youtu.be/4UWtTi-lkus.

Chapter 2

Preliminaries

2.1 Construction of Visibility Graph



(a) Complete (b) Simplified

Figure 2.1: A visibility graph and its simplified version (green line segments are edges, blue/red dots are vertices)

Consider a rectangular planar environment with polygonal obstacles. We establish a visibility graph of the environment, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with vertices, $\mathbf{v} \in \mathcal{V}$, consisting of the vertices at the corners of the polygonal obstacles, the control points (points on the environment boundary at which the cables are attached to the robot), robot's current location (in context of workspace optimization) or the start and the end vertices of a trajectory (in context of path planning), and the edges, $\{\mathbf{v} \to \mathbf{v}'\} \in \mathcal{E}$, consisting of line segments that connect vertices with direct line of sight. For every pair of vertices we

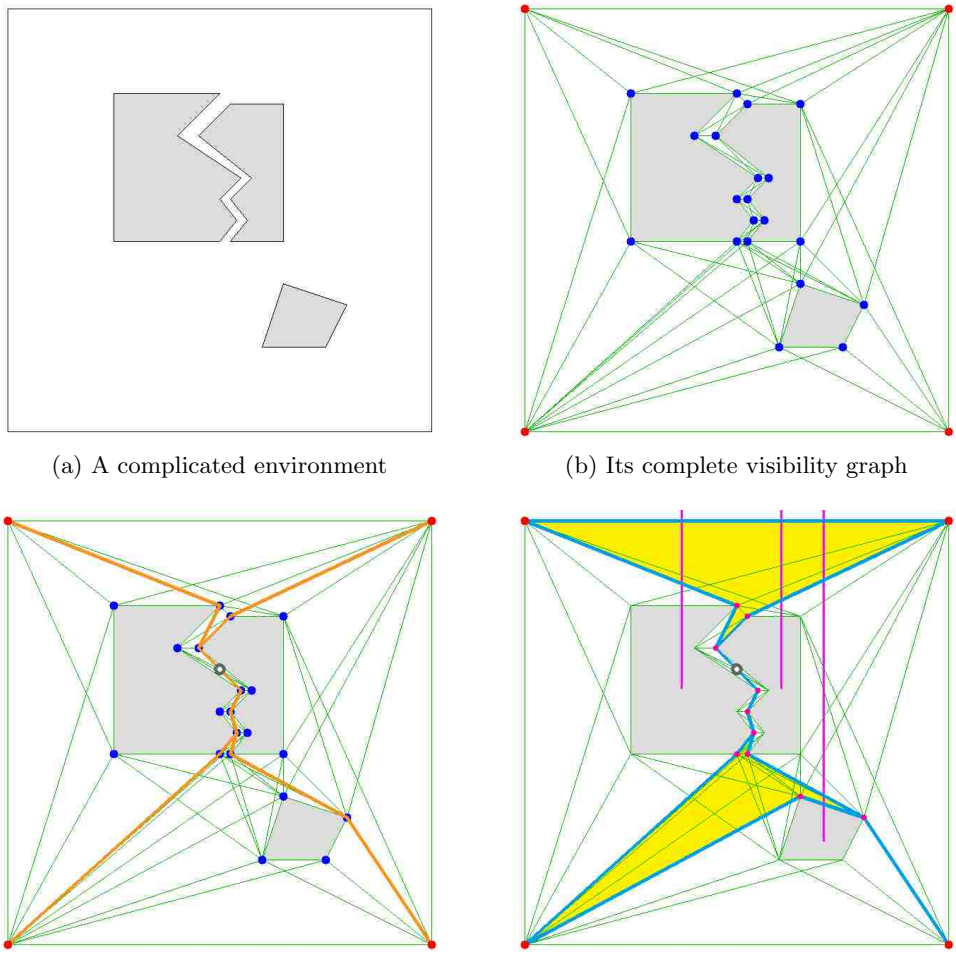
construct an edge if it neither penetrates any obstacles nor goes beyond the boundary of the environment 2.1a. This method is easy and quite straightforward, but some of the edges would be redundant. They would complicate the graph hence increase the running time of algorithms. Therefore, in our tests, we use simplified visibility graphs to decrease running time. That is, for edges, we only keep the common tangent lines of the obstacles and the tangent line from control points (as well as robot's location/start/end point) to the obstacles. When dealing with the concave polygon, we select shortcuts as edges between corners, omitting the vertices at cavity 2.1b. In the case of a concave environment, we represent it as a convex environment with a set of convex obstacles subtracted from it. We construct the tangents of the constructive obstacles, then delete the vertices and edges outside the boundary of the original concave environment from the visibility graph.

Although the simplified graph has some advantages for most of time, when meeting some complicated non-convex obstacle configuration, it would be very hard to find common tangent lines. In cases as shown in Figure 2.2, it is preferable to use the complete graph. We carefully consider the graph for searching algorithms based on the configuration of obstacles and that of the environment.

2.2 Brief Introduction of Homotopy and Homology Classes

Definition 1 *(Homotopy classes [3, 15]) Two trajectories τ_1, τ_2 connecting the same start and end points, $\mathbf{v}_s$, $\mathbf{v}_g \in X$ respectively, are homotopic or belong to the same homotopy class iff one can be continuously deformed into the other without intersecting any obstacle.*

Formally, in a topological space X, if $\tau_1 : [0,1] \to X$, $\tau_2 : [0,1] \to X$ represent two trajectories such that $\tau_1(0) = \tau_2(0) = \mathbf{v}_s$ and $\tau_1(1) = \tau_2(1) = \mathbf{v}_g$, τ_1 and τ_2 are homotopic iff there exists a continuous map $\eta : [0,1] \times [0,1] \to X$ such that $\eta(\alpha, 0) = \tau_1(\alpha)$, $\eta(\alpha, 1) = \tau_2(\alpha)$ $\forall \alpha \in [0,1]$, and $\eta(0, \beta) = \mathbf{v}_s$, $\eta(1, \beta) = \mathbf{v}_g$ $\forall \beta \in [0,1]$. Alternatively, in the notation of Hatcher [2], τ_1 and τ_2 are homotopic iff the closed curve $\tau_1 \sqcup \tau_2$ belongs to the trivial class of the first homotopy group (or fundamental group) of X, denoted by $\pi_1(X)$. That is, $\tau_1 \sqcup \tau_2 = \mathbf{0} \in \pi_1(X)$.



(a) A complicated environment

(b) Its complete visibility graph

(c) One of possible cable configurations

(d) The corresponding workspace

Figure 2.2: A complicated environment that is not suited for simplified visibility graph

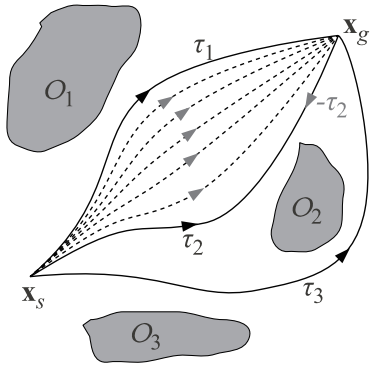
A set of all homotopically equivalent trajectories constitute a homotopy class. We denote the homotopy class of a path τ as $[\tau]$.

Definition 2 *(Homology classes [3]) Two trajectories τ_1 and τ_2 connecting the same start and end points, $\mathbf{v}_s$ and $\mathbf{v}_g$ respectively, are homologous or belong to the same homology class iff τ_1 together with τ_2 (the later with opposite orientation) forms the complete boundary of a 2-dimensional manifold not containing/intersecting any of the obstacles.*

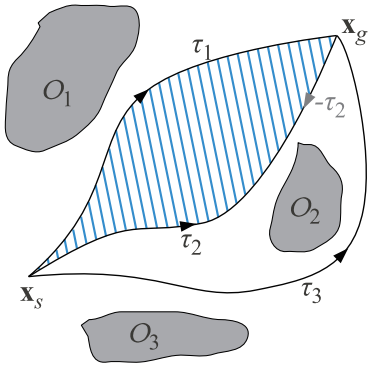
Formally, in the notation of Hatcher [2], τ_1 and τ_2 are homologous iff $\tau_1 \sqcup -\tau_2$ belongs to the trivial class of the first homology group of X, denoted by $H_1(X)$. That is, $[\tau_1 \sqcup -\tau_2] = \mathbf{0} \in H_1(X)$.

A set of all homologously equivalent trajectories constitute a homology class.

Examples are shown in Figure 2.3 accordingly.



(a) τ_1 is homotopic to τ_2 since there is a continuous sequence of trajectories representing deformation of one into the other. τ_3 belongs to a different homotopy class since it cannot be continuously deformed into any of the other two.



(b) τ_1 is homologous to τ_2 since there exists an area A (shaded region) such that $\tau_1 \sqcup -\tau_2$ is the boundary of A. τ_3 belongs to a different homology class since such an area does not exist between τ_3 and any of the other two trajectories.

Figure 2.3: Illustration of homotopy and homology equivalences. In this example τ_1 and τ_2 are both homotopic and homologous

2.3 Fundamental Group

At first, introduce the definition of group in mathematics. A group is a set, G, together with an operation $\bullet$ (called the group operator of G) that combines any two elements a and b to form another element, denoted $a \bullet b$ or ab. To qualify as a group, the set and operation, $(G, \bullet)$, must satisfy four requirements known as the group axioms [26]:

1. *Closure:* $\forall a, b \in G$, the result of the operation, $a \bullet b$, is also in G.

2. *Associativity:* $\forall a, b, c \in G$, $(a \bullet b) \bullet c = a \bullet (b \bullet c)$.

3. *Identity element:* There exists an element $e \in G$ such that, for every element $a \in G$, the equation $e \bullet a = a \bullet e = a$ holds. Such an element is unique, and thus one speaks of the identity element.

4. *Inverse element:* For each $a \in G$, there exists an element $b \in G$, commonly denoted a^{-1} (or $-a$, if the operation is denoted "+"), such that $a \bullet b = b \bullet a = e$, where e is the identity element.

The result of an operation may depend on the order of the operands. In other words, the result of combining element a with element b need not yield the same result as combining

element b with element a; the equation $a \bullet b = b \bullet a$ may not always be true. Groups for which the commutativity equation $a \bullet b = b \bullet a$ always holds are called *abelian* groups.

The *fundamental group* or the first homotopy group of a topological space, X, denoted as $\pi_1(X)$, is the set of all homotopy classes of oriented closed loops based at certain points(trajectories with $\mathbf{v}_s = \mathbf{v}_g = \mathbf{v}_0$) in X with a group structure imposed on the set as follows:

1. The identity element is the class of loops that can be contracted to the point $\mathbf{v}_0$ (null homotopic);

2. The inverse of a homotopy class, $[\tau]$, is the homotopy class of loops constituting of the same loops as in $[\tau]$, but with reversed orientation, and is denoted as $[-\tau]$ or $[\tau]^{-1}$;

3. The group operation of two classes $[\tau_1]$ and $[\tau_2]$ is the class of loops that are obtained by concatenating a trajectory in $[\tau_1]$ with a trajectory in $[\tau_2]$ (the class of loop $\tau(t) = \begin{cases} \tau_1(2t), & t \leq t \leq \frac{1}{2} \\ \tau_2(2t-1), & \frac{1}{2} \leq t \leq 1 \end{cases}$). The fundamental group is in general a non-abelian group.

Fundamental groups of space $X = \mathbb{R}^2 - \mathcal{O}$ and its subspaces X_0, X_1, X_2 are shown in Figure 2.4, where the subspace $X_0 = X - \bigcup_{i=1}^{2} r_i$ and $X_i = X - \bigcup_{i=1, i \neq j}^{2} r_j, i \neq 0$.

2.4 Free Group and Free Product of Groups

A *free group* over a set of letters/symbols, is the group whose elements consists of all expressions (words) constructed out of the letters in the set and their formal *inverses*, with identity element being the *empty word*, and the group operation being word concatenation (with any letter juxtaposed with its inverse reducing to the identity) [26]. The fundamental group is a free group.

Given two groups, G and H, the *free product* of the groups is the group of words that can be constructed with all the elements of the groups as the letters. It is thus written as $G * H = \{g_1 h_1 g_2 h_2 \ldots | g_i \in G, h_i \in H\}$.

For example, it can be shown using the generalized van-Kampen's theorem [5] that the fundamental group $\pi_1(X)$ is a free product of fundamental groups of subspaces, that

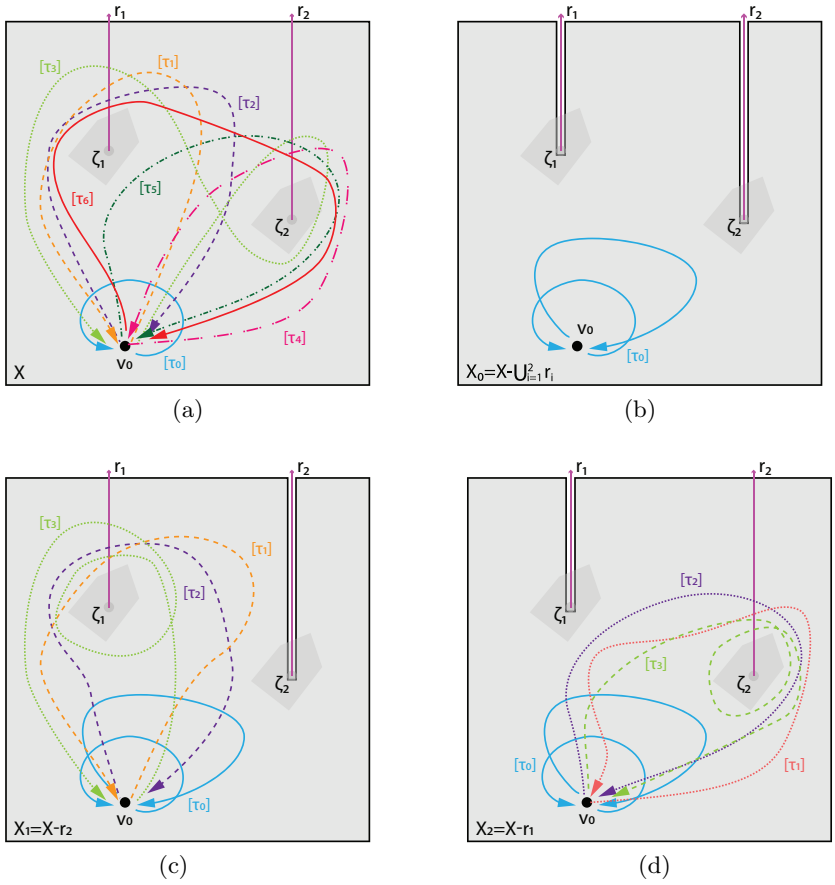


Figure 2.4: Fundamental group of space X with two representative points in it and that of its subspaces X_0, X_1, X_2 (partial homotopy classes shown)

is, $\pi_1(X) \simeq \pi_1(X_0) * \pi_1(X_1) * \pi_1(X_2) * \cdots * \pi_1(X_n) \simeq *_{i=1}^{n} \mathbb{Z}$, a free product of n copies of $\mathbb{Z}$. Thus the fundamental group satisfies the group axiom of *Closure* as well. An element of free product is shown in Figure 2.5.

On the other hand, the first homology group of X, from where the homology signatures generate, denoted by $H_1(X)$, is a cartesian product of first homology groups of subspaces, i.e. $H_1(X) \simeq H_1(X_0) \times H_1(X_1) \times H_1(X_2) \times \cdots \times H_1(X_n) \simeq \times_{i=1}^{n} \mathbb{Z}$, a cartesian product of n copies of $\mathbb{Z}$.

2.5 h-signature and H-signature

Assuming that all obstacles and ends of trajectories are fixed, h-signature and H-signature (homotopy and homology invariants) are respectively the presentations of homotopy and homology classes of trajectories —they are two functions that map from

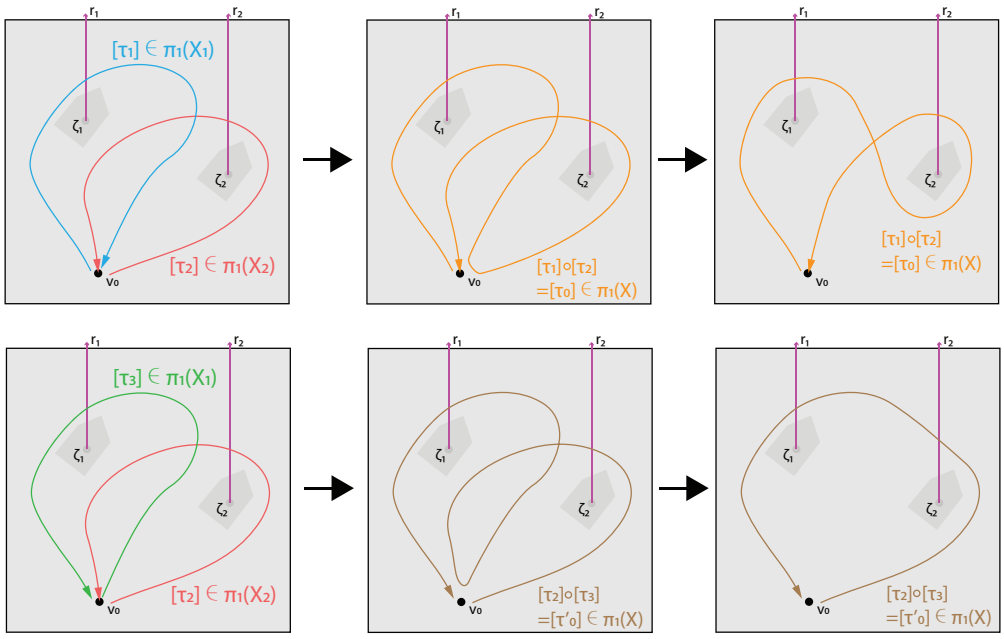


Figure 2.5: Homotopy class $[\tau_0]$ is an element of free group $\pi_1(X)$ which is also a free product of $\pi_1(X_0)$, $\pi_1(X_1)$ and $\pi_1(X_2)$ shown in Figure 2.4, that is, $[\tau_0] \in \pi_1(X) = \pi_1(X_0) * \pi_1(X_1) * \pi_1(X_2)$. $[\tau_0]$ is a concatenation of two homotopy classes $[\tau_0] = [\tau_1] \circ [\tau_2]$, where $[\tau_1] \in \pi_1(X_1)$ and $[\tau_2] \in \pi_1(X_2)$. So is $[\tau_0'] \in \pi_1(X)$, a concatenation of $[\tau_2] \in \pi_1(X_2)$ and $[\tau_3] \in \pi_1(X_1)$

homotopy and homology classes to "word" and to "vector". Two trajectories connecting the same start and end points have the same h- (or H-) signatures iff they are in the same homotopy (or homology) class [8].

Function $h(\cdot)$ is for denoting the h-signature of a trajectory. We use representative points (inside the obstacles), ζ_i, and the non-intersecting rays r_i emanating from the representative points for constructing h-signatures. We form a word by tracing τ, and consecutively placing the letters of the rays that it crosses, with a superscript of '+1' if the crossing is from right to left, and '-1' if the crossing is from left to right. The word thus formed is written as $h(\tau)$, and $h(\tau) = h([\tau]), [\tau] \in \pi_1(X)$. For example, in Figure 2.6a, if τ first crosses r_b right to left then r_a from right to left as well, the h-signature is 'ba'(short for '$b^{+1}a^{+1}$'); if τ first crosses r_a left to right then r_b from left to right too, the h-signature is '$a^{-1}b^{-1}$'. If in an h-signature 'a^{-1}' appears next to 'a', indicating that the trajectory crosses r_a followed by crossing back, these two letters can cancel each other, like the trajectory never crosses r_a. We use simplified h-signatures. For instance, in Figure 2.6a the empty h-signature of the upper curve was simplified

from the initial '$baa^{-1}b^{-1}$' to 'bb^{-1}', then from 'bb^{-1}' to ' '(empty). The h-signature is internally non-commutative. The direction of a curve is important, for the same path in reverse direction will result in an inverse h-signature where both the order of letters and superscripts of letters are opposite [6]. If a curve is a closed loop and it encloses no representative points (obstacles), it is null homotopic with an empty h-signature. More examples are shown in Figure 2.6a.

Likewise, $H(\cdot)$ is a function for denoting the H-signature of a trajectory. H-signature is a vector, the i^{th} element of which has the simple interpretation of counting the number of times the curve, τ, intersects the ray emanating from ζ_i (see Figure 2.6b). In particular, define $\#_i\tau$ = (Number of times τ crosses the ray r_i emanating from ζ_i from left to right) − (Number of times τ crosses the ray r_i emanating from ζ_i from right to left). Then, $H(\tau) = [\#_1\tau, \#_2\tau, \ldots, \#_n\tau]^\intercal$ [6].

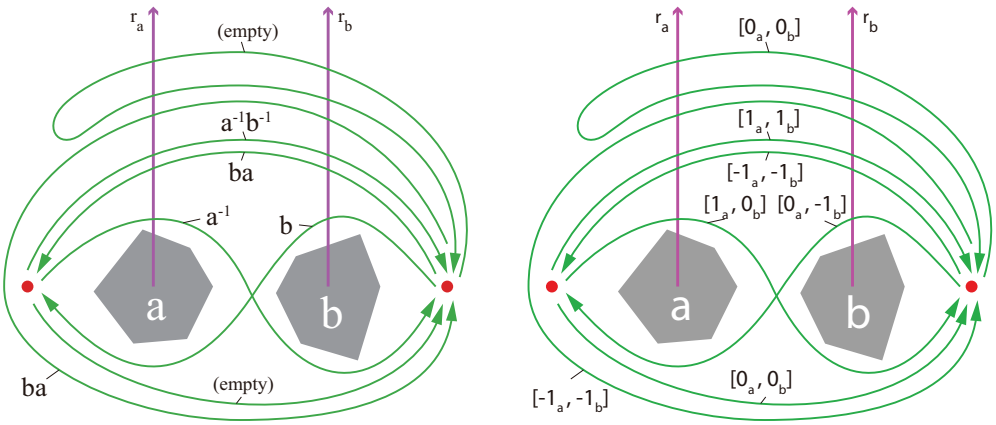
For instance, 3 obstacles in the environment, if the H-signature of a trajectory τ is $H(\tau) = [1, 0, 1]^\intercal$, it shows that after all τ cross ray r_1 once and r_3 once from left to right, not crossing r_2. If τ is a closed loop, $H(\tau) = [1, 0, 1]^\intercal$ shows that ζ_1 and ζ_3 are inside the loop and ζ_2 is outside.

There is a conversion from h-signature to H-signature. It interchanges the letters in h-signature, putting the same letters together and, if there are more than one of a certain letter, merge them by summing up the superscripts, then take the opposite value. E.g. 2 obstacles in the environment, $h(\tau)$ ="$b^{-1}abab$" could be firstly converted into "$aab^{-1}bb$", then into $[(1+1)_a, (-1+1+1)_b]$, finally into $H(\tau) = [-2_a, -1_b]$.

2.6 The h-signature Augmented Graph

In order to keep track of the homotopy invariants, we define an h-augmented graph [3], $\mathcal{G}_h = (\mathcal{V}_h, \mathcal{E}_h)$, based on a visibility graph, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, such that a vertex in $\mathcal{G}_h$ contains the additional information of the h-signature of the trajectory leading from a start vertex $\mathbf{v}_s$ up to this vertex besides its coordinate. A transition from vertex $(\mathbf{v}, \mathfrak{h})$ to vertex $(\mathbf{v}', \mathfrak{h}')$ means that the h-signature, $\mathfrak{h}'$, is a concatenation of $\mathfrak{h}$ and the h-signature of trajectory from $\mathbf{v}$ to $\mathbf{v}'$.

1.



(a) h-signature (b) H-signature

Figure 2.6: h- and H-signatures of different trajectories connecting two same points in a space with two obstacles

$$\mathcal{V}_h = \left\{ (\mathbf{v}, \mathfrak{h}) \;\middle|\; \begin{array}{l} \mathbf{v} \in \mathcal{V}, \text{ and,} \\ \mathfrak{h} = h(\widetilde{\mathbf{v}_s \mathbf{v}}) \text{ for some trajectories from} \\ \quad \text{the start vertex } \mathbf{v}_s \text{ to this vertex } \mathbf{v} \end{array} \right\}$$

2. An edge $\{(\mathbf{v}, \mathfrak{h}) \to (\mathbf{v}', \mathfrak{h}')\}$ is in $\mathcal{E}_h$ for $(\mathbf{v}, \mathfrak{h}) \in \mathcal{V}_h$ and $(\mathbf{v}', \mathfrak{h}') \in \mathcal{V}_h$, iff $(\mathbf{v} \to \mathbf{v}') \in \mathcal{E}$, and, $\mathfrak{h}' = \mathfrak{h} \circ h(\mathbf{v} \to \mathbf{v}')$, where, "$\circ$" is a concatenate operator.

3. The cost associated with an edge $\{(\mathbf{v}, \mathfrak{h}) \to (\mathbf{v}', \mathfrak{h}')\} \in \mathcal{E}_h$ is the same as that associated with edge $\{\mathbf{v} \to \mathbf{v}'\} \in \mathcal{E}$.

The h-augmented graph is unbounded. Its vertices are generated on-the-fly and as required during the execution of *Dijkstra*'s/$A*$ search on the graph, which we describe later.

2.7 *Dijkstra*'s and *A** Search Algorithm

In 1959, Edsger W. Dijkstra proposed an algorithm, which is called *Dijkstra*'s algorithm, for finding the shortest paths between a start vertex $\mathbf{s}$ and a goal vertex $\mathbf{g}$ in a graph G. The algorithm contains a set E of "explored" vertices $\mathbf{u}$ for which we have determined a shortest-path distance $d(\mathbf{u})$ from the start vertex $\mathbf{s}$; this is the "explored" part of graph [20]. The algorithm explores one vertex in the graph at each step (an example shown in Figure 2.7).

1. Initially $E = \emptyset$ (Figure 2.7a).

2. Set all vertices $\mathbf{v} \in G - E$ with $d(\mathbf{v}) = +\infty$, and $d(\mathbf{s}) = 0$. Select the start vertex $\mathbf{s}$ as current vertex $\mathbf{u}$ (Figure 2.7b).

3. For the current vertex $\mathbf{u}$, consider all of its neighboring vertices $\mathbf{v} \in G - E$ connected to $\mathbf{u}$ with single edge $\mathbf{e} = \{\mathbf{u} \to \mathbf{v}\}$ and update all their tentative distances $d(\mathbf{u}) + l_{\mathbf{e}}$ through vertex u, where $l_{\mathbf{e}}$ is the length of the edge $\mathbf{e}$. Compare the tentative distances with the current distance and assign the smaller one. For example, in Figure 2.7d if the vertex $\mathbf{a}$ is marked with a distance $d(\mathbf{a}) = 2$, and the edge connecting it with a neighbor $\mathbf{c}$ has length $l_{\mathbf{e}=\{\mathbf{a}\to\mathbf{c}\}} = 1$, then the distance from $\mathbf{s}$ to $\mathbf{c}$ via $\mathbf{a}$ will be $d(\mathbf{c}) = 2 + 1 = 3$. If $\mathbf{c}$ was previously marked with a distance (in Figure 2.7c it is 5) which is greater than 3, then change it to 3. If the previous value is less than or equal to the newly calculated one, keep the previous one (vertex $\mathbf{b}$ in Figure 2.7d).

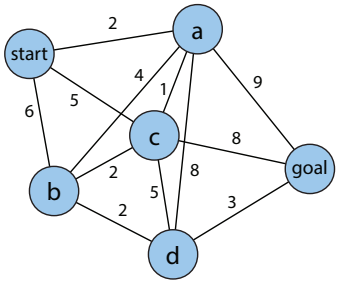
4. When we are done calculating tentative distances of all of the neighbors of the current vertex $\mathbf{u}$, add $\mathbf{u}$ to E. Vertex in set E will never be checked again.

5. If the goal vertex $\mathbf{g}$ has been added to E (when planning a complete traversal) or if the smallest tentative distance among the vertices in $G - E$ is infinity (occurs when there is no connection between the start vertex and remaining unexplored vertices), then stop. The algorithm has finished (Figure 2.7h).

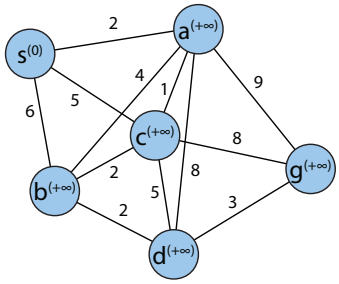
6. Otherwise, select the unexplored vertex $\mathbf{v} \in G - E$ that has the smallest $d(\mathbf{v})$, set it as the new "current vertex", and go back to step 3.

When planning a route, it is actually not necessary to wait until all the neighbors of goal vertex are "explored" as above: the algorithm can stop once the goal vertex has the smallest tentative distance in $G - E$ (and thus could be selected as the next "current").

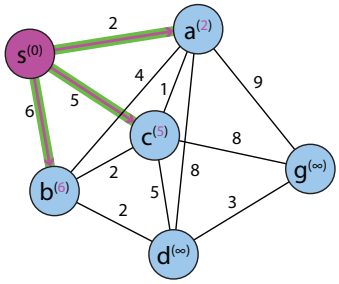
It is simple to produce the $\mathbf{s} - \mathbf{g}$ path corresponding to the distances found by Dijkstra's algorithm. As each node $\mathbf{v}$ is added to the set E, we simply record the edge $\{\mathbf{u} \to \mathbf{v}\}$ on which it *last* updates the value $d(\mathbf{v})$. The path $P_{\mathbf{g}}$ is implicitly represented by these edges: if $\{\mathbf{v} \to \mathbf{g}\}$ is the edge we have stored for $\mathbf{g}$, then $P_{\mathbf{g}}$ is just (recursively) the path $P_{\mathbf{v}}$ followed by the single edge $\{\mathbf{v} \to \mathbf{g}\}$. In other words, to construct $P_{\mathbf{g}}$, we



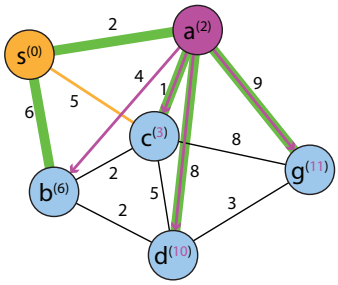
(a) Initial graph G, no vertex has been checked, $E = \emptyset$.



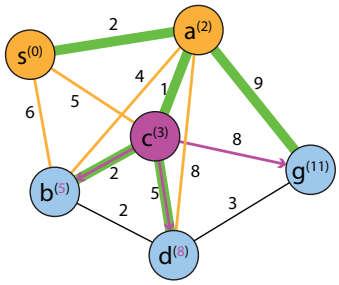
(b) Mark the start vertex **s**'s tentative distance as 0, others' initials as $+\infty$.



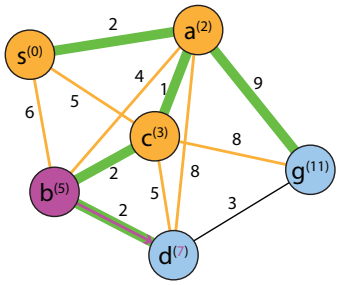
(c) Check **s**'s all neighbors, vertex **a**, **b** and **c**, update their tentative distances and then put **s** into E.



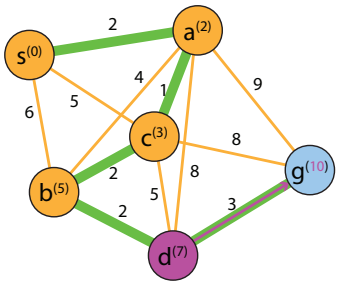
(d) Vertex **a** has the smallest value in $G - E$. Hence check **a**'s neighbors, **b**, **c**, **d** and **g**, update **c**, **d** and **g**'s tentative distances and then put **a** into E.



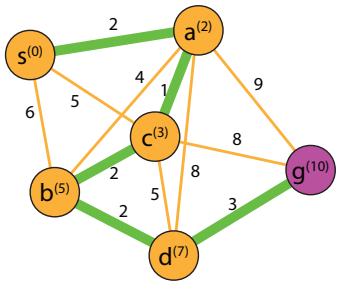
(e) Vertex **c** has the smallest value in $G-E$. Hence check **c**'s neighbors, **b**, **d** and **g**, update **b** and **d**'s tentative distances and then put **c** into E.



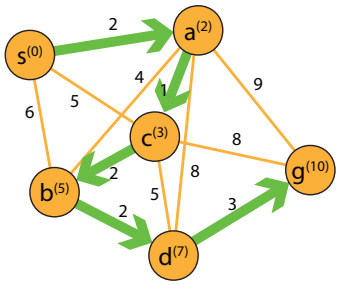
(f) Vertex **b** has the smallest value in $G-E$. Hence check **b**'s neighbors, **d**, update **d**'s tentative distance and then put **b** into E.



(g) Vertex **d** has the smallest value in $G-E$. Hence check **d**'s neighbors, **g**, update **g**'s tentative distance and then put **d** into E.



(h) The goal vertex **g** is the only vertex in $G - E$, having no unexplored neighbor. Put **g** into E. Algorithm stops.



(i) Reconstruct the shortest path in green from **s** via **a**, **c**, **b**, **d** to **g**.

Figure 2.7: An example of implementation of Dijkstra's algorithm, given lengths of all edges. Orange vertices are in the explored set E. The current vertex and edges connecting it to its to-be-explored neighbors in each picture are colored in purple. Edges that last update tentative distances are colored in green.

simply start at **g**; follow the edge we have stored for **g** in the reverse direction to **v**; then follow the edge we have stored for **v** in the reverse direction to its predecessor; and so on until we reach **s**. Note that **s** must be reached, since our backward walk from **v**

visits nodes that were added to E earlier and earlier (green path in Figure 2.7i).

*A** algorithm is an extension of *Dijkstra*'s algorithm. *A** achieves better performance by using heuristics to guide its search. It is an informed search algorithm, or a best-first search, meaning that it solves problems by searching among all possible paths to the goal vertex **g** for the one that incurs the smallest cost, and among these paths it first considers the ones that appear to lead most quickly to the goal. It is formulated in terms of weighted graphs: starting from the start vertex of a graph, it constructs a tree of paths starting from that vertex, expanding paths one step at a time, until one of its paths ends at the predetermined goal vertex.

At each iteration of its main loop, *A** needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal vertex. Specifically, *A** selects the path that minimizes

$$f(\mathbf{v}) = d(\mathbf{v}) + h(\mathbf{v})$$

where $\mathbf{v}$ is the last vertex on the path, $d(\mathbf{v})$ is the cost of the path from the start vertex $\mathbf{s}$ to $\mathbf{v}$, and $h(\mathbf{v})$ is a heuristic that estimates the cost of the cheapest path from $\mathbf{v}$ to $\mathbf{g}$. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal vertex. Different from *Dijkstra*'s algorithm's step 6 above, in *A** it should be:

"6. Otherwise, select the unexplored vertex $\mathbf{v} \in G - E$ that has the smallest $f(\mathbf{v}) = d(\mathbf{v}) + h(\mathbf{v})$, set it as the new "current vertex", and go back to step 3."

Typical implementations of *A** use a priority queue to perform the repeated selection of minimum (estimated) cost vertices to expand. This priority queue is known as the open set or fringe. At each step of the algorithm, the vertex $\mathbf{v}$ with the lowest $f(\mathbf{v})$ value is removed from the queue, the f and d values of its neighbors are updated accordingly, and these neighbors are added to the queue. The algorithm continues until a goal vertex has a lower f value than any vertex in the queue (or until the queue is empty). The f value of the goal is then the length of the shortest path, since h at the goal is zero in an

admissible heuristic.

The algorithm described so far gives us only the length of the shortest path. To find the actual sequence of steps, the algorithm can be easily revised so that each vertex on the path keeps track of its predecessor. After this algorithm is run, the ending node will point to its predecessor, and so on, until some node's predecessor is the start vertex.

As an example, when searching for the shortest route on a map, h might represent the straight-line distance to the goal, since that is physically the smallest possible distance between any two vertices.

Chapter 3

Algorithm Design for Workspace Computation

In this section, we describe some specific features, properties and algorithms related to the cable configurations and the workspace of the cable-driven robot. These include the topological properties of workspace's boundary and algorithms for obtaining all valid workspaces from given obstacle configurations.

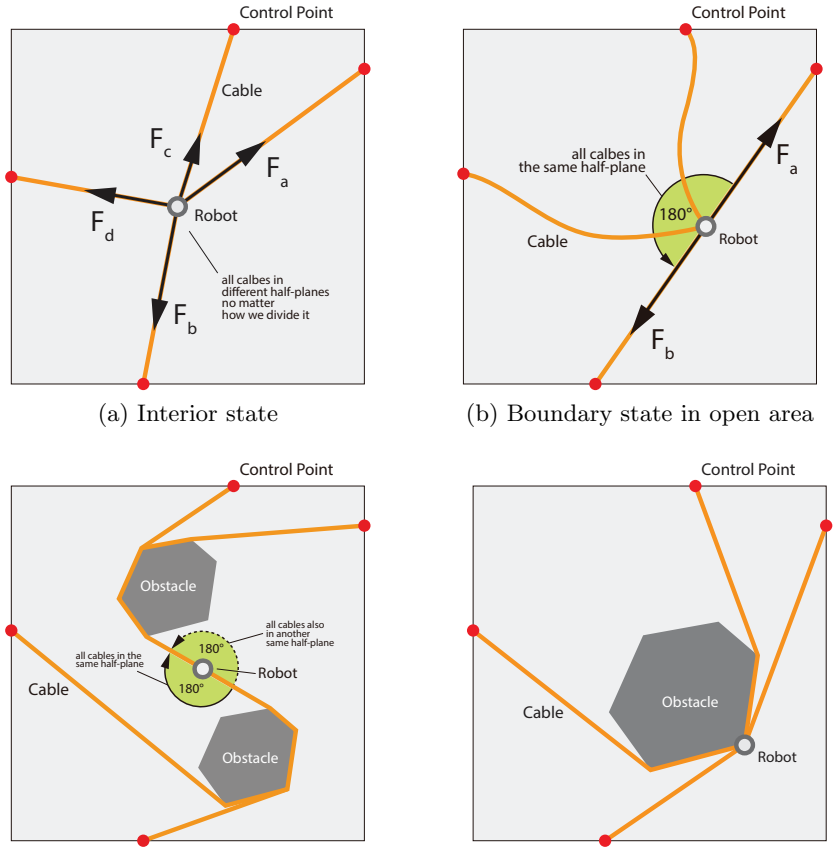
3.1 Definition of Workspace's Boundary

The configurations of a cable-robot system in which the robot is capable of moving in any direction are called an *interior point* (Figure 3.1a). On the contrary, a *boundary point* is a configuration where the robot can move only in some specific directions. These directions can constitute a half plane (Figure 3.1b) or more generally union of cones (Figure 3.1d). All the boundary points constitute a *boundary* of the workspace.

3.2 Force Analysis

The physical constraints of the cable is that there can only be tension but no pressure acting on each cable and that net force on the robot must be zero to make it stays at a certain position (see Figure 3.1a), denoted as $\mathbf{0} = \sum_i F_i \hat{\mathbf{e}}_i, i = 1, 2, \ldots, n$, where $F_i \geq 0$ is a non-negative scalar of tension, $\hat{\mathbf{e}}_i$ is a unit vector of the direction in which the i^{th}

cable points at the location of the robot and n is the number of cables. When the robot is maneuverable, not all of the cables can be slack, that is, $\sum_i F_i \neq 0$. To make the net force zero with some taut cables, there must exist at least two of them and they must lie in different half-planes to counteract each other. If all cables are pointing in the same half-plane, the robot cannot go any further towards the other half-plane (Figure 3.1b and 3.1c). We call this case the boundary state in the open area.



(a) Interior state

(b) Boundary state in open area

(c) A special boundary state in open area

(d) Boundary state touching an obstacle

Figure 3.1: Robot states and cable configuration

3.3 Boundary State

There are two kinds of boundary states.

3.3.1 In the Open Area

If $\hat{\mathbf{e}}_i$ of all taut cables span a one-dimensional line, it is a boundary state and the robot is located at the boundary of the workspace (see Figure 3.1b and 3.1c).

3.3.2 Touching an Obstacle

When the robot driven by cables touches an obstacle, the robot is in touching-obstacle boundary state and the obstacle's edge or corner that is touched is a part of boundary of the workspace, where the net force of cables may not be zero, shown in Figure 3.1d.

3.4 Shortest Paths and Boundary

Lemma: The workspace's boundary curve connecting a pair of neighboring control points is the shortest path in the same homotopy class connecting that pair (by "neighboring" we refer to *adjacent* control points encountered as we trace the boundary of the environment).

Sketch of Proof: When in boundary state, if we remove all slack cables, the robot's position and taut cables will keep stable. We can regard the pair of taut cables as a whole which is also taut, going from one control point to a neighboring one though the robot which can be regarded as a mass point on that whole cable. This whole taut cable is the shortest path in current homotopy class connecting those two control points, shown as blue curves in Figure 1.2a and 1.2b.

Hence, when the robot touches any obstacle and the net force of cables may not be zero (especially at the corner of obstacle), the robot is still on the boundary of its workspace because it is still on the shortest path between control points.

3.5 Boundary's Features

When the robot is at an interior point, we can move the robot along arbitrary trajectories inside the boundary. As the robot is moving toward a part of boundary, a pair of neighboring cables can deform continuously into that part of boundary, without

interfering with any obstacle, shown in Figure 1.2a and 1.2b. This deformation from the initial configuration into a part of boundary holds for any pair, which indicates that all cables pairs can deform into a complete and closed-loop boundary. Since no obstacles are crossed during this deformation, there must be no obstacles inside the boundary. Just for comparison, Figure 1.2c shows an invalid boundary, even if it is made of shortest paths.

Proposition 1 *The closed-loop formed by the boundary of a workspace is null homotopic (i.e. its h-signature is 'empty' word), stated in Section 2.5.*

3.6 Shortest Paths Searching

After constructing the visibility graph, we use *Dijkstra*'s search in the h-augmented graph to get the shortest paths with various h-signatures. We construct n threads for multi-threading search, $\mathbf{T}_i$, where $i = 1, 2, \ldots, n$ and n is the number of control points. Each thread contains *Dijkstra*'s search returning paths connecting a pair of neighboring control points which are indexed along clockwise or anticlockwise direction. For example, $\mathbf{T}_1$ is for paths connecting control point $\mathbf{c}_1$ and $\mathbf{c}_2$, namely $\mathbf{T}_2$ for $\mathbf{c}_2$ and $\mathbf{c}_3$, $\ldots$, $\mathbf{T}_n$ for $\mathbf{c}_n$ and $\mathbf{c}_1$, as shown in Figure 1.2a. We insert the output τ into n corresponding sets of shortest paths, $\mathcal{P}_i = \{\tau_{i1}, \tau_{i2}, \ldots\}$. These n threads keep searching till the length of boundary (consist of n paths, one from each set) must exceed a limit $\mathcal{L}$ we properly preset.

Since there are only n control points, the indices of control points in Algorithm 1 can run from 1 to n, thus in Line 3, if $i = n$, then by $i + 1^{\text{th}}$ we refer to 1^{st} control point (i.e.: we take the modulo with respect to n with shift of 1). Hereafter whenever we refer to $i + 1$ for a control point index, we assume this convention. The paths returned from the *Dijkstra*'s search are in an order from least cost to higher cost. Thus the 1^{st} path, τ_{i1}, in every thread's outcome is of the least cost. In Line 4-7, where the function $C(\cdot)$ is the cost of some trajectory, the length l is the sum of the cost of the latest path in the current thread and those of the 1^{st} paths from other threads, a possible least boundary cost for this latest outcome. If l_i is greater than $\mathcal{L}$, indicating all subsequent outcomes of the current thread must form a boundary that has a length greater than $\mathcal{L}$, thus we stop

Algorithm 1 Shortest path searching in the i^{th} thread $\mathbf{T}_i$

Input: The h-augmented graph, $\mathcal{G}_h = \{\mathcal{V}_h, \mathcal{E}_h\}$; the set of control points for start vertices and goal vertices, $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \dots, \mathbf{c}_n\}$; a limit of boundary cost $\mathcal{L}$.

Output: Set of shortest paths in different homotopy classes, $\mathcal{P}_i = \{\tau_{i1}, \tau_{i2}, \dots\}$; set of h-signatures of paths, $\mathcal{H}_i = \{\mathfrak{h}_{i1}, \mathfrak{h}_{i2}, \dots\}$.

1: $k \leftarrow 1$
2: **loop**
3: $\quad \tau_{ik} \leftarrow$ a shortest path, in the k^{th} homotopy class connecting $\{\mathbf{c}_i, \text{`\ '}\}$ and $\{\mathbf{c}_{i+1}, \mathfrak{h}\}$ for some $\mathfrak{h}$
4: $\quad l_i \leftarrow C(\tau_{ik}) + \sum_j C(\tau_{j1}), j = 1, 2, \dots, n, j \neq i$
5: $\quad$ **if** $(l_i > \mathcal{L})$ **then**
6: $\quad\quad$ **break loop**
7: $\quad$ **end if**
8: $\quad$ Insert τ_{ik} into $\mathcal{P}_i$
9: $\quad \mathfrak{h}_{ik} \leftarrow h(\tau_{ik})$
10: $\quad$ Insert $\mathfrak{h}_{ik}$ into $\mathcal{H}_i$
11: $\quad k \leftarrow k + 1$
12: **end loop**

this thread. In Line 10, h-signatures of all shortest paths in different homotopy classes are stored in the set $\mathcal{H}_i$ for later boundary validation. Here we introduce a function P for later use to get a part of the boundary such that $P(\mathbf{v}_s, \mathbf{v}_g, \mathfrak{h}_g^i)$ returns the shortest path from $\{\mathbf{v}_s, \text{`\ '}\}$ to $\{\mathbf{v}_g, \mathfrak{h}_g^i\}$.

3.7 Valid Boundary Construction

After we finished searching in all threads, we need to find out all proper combinations that have an empty concatenation of the h-signatures. Algorithm 2 retrieves one path's h-signature at a time from each h-signature set $\mathcal{H}_i$, n h-signatures in total, to check whether their full concatenation is empty in Line 6 and 7. If so, we store the whole boundary, ω, into the set of all boundaries, $\mathcal{W}$, shown in Line 11. Although the complexity of the algorithm rises exponentially with the number of control points or the size of set $\mathcal{H}_i$, we have limited number of control points and the upper bound of the length of cables during the search hence make it computationally feasible.

Algorithm 2 Getting valid closed boundary

Input: Set of shortest paths in different homotopy classes, $\mathcal{P}_i = \{\tau_{i1}, \tau_{i2}, \dots\}$; set of h-signatures, $\mathcal{H}_i = \{\mathfrak{h}_{i1}, \mathfrak{h}_{i2}, \dots\}$, where $\mathfrak{h}_{ij} = h(\tau_{ij})$;
Output: Set of valid closed boundary, $\mathcal{W}$;

$m \leftarrow 1$
for all $\mathfrak{h}_{1i}$ in $\mathcal{H}_1$ **do**
    for all $\mathfrak{h}_{2j}$ in $\mathcal{H}_2$ **do**
        ...
        for all $\mathfrak{h}_{nk}$ in $\mathcal{H}_n$ **do**
            $q \leftarrow \mathfrak{h}_{1i} \circ \mathfrak{h}_{2j} \circ \cdots \circ \mathfrak{h}_{nk}$
            if $q =$ ' ' **then**
                if (any of $\tau_{1i}, \tau_{2j}, \dots, \tau_{nk}$ self-tangles) **then**
                    continue loop
                end if
                Insert $\omega_m = \{\tau_{1i}, \tau_{2j}, \dots, \tau_{nk}\}$ into $\mathcal{W}$
                $m \leftarrow m + 1$
            end if
        end for
        ...
    end for
end for

3.8 Area Computation

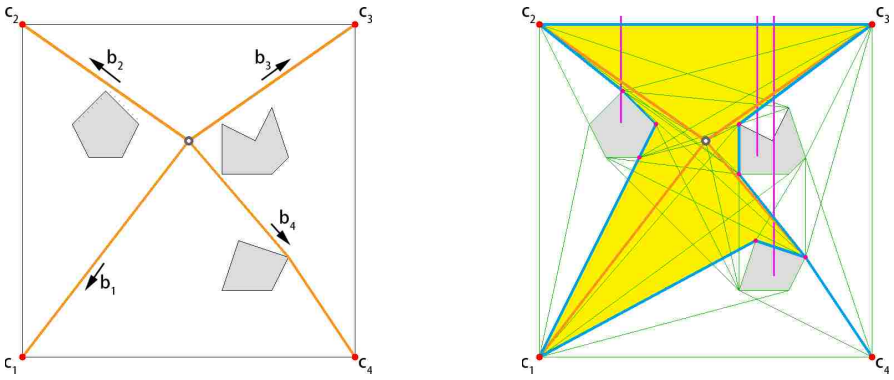
Every boundary is made up of a few vertices, $\omega = \{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n\}$, where vertex, $\mathbf{v}_i = \{x_i, y_i\}$, is either a control point or a vertex of the polygon obstacles. We use the following formula to compute the area of the workspace:

$$A(\omega) = \frac{1}{2} \left| (x_1 y_2 - y_1 x_2) + (x_2 y_3 - y_2 x_3) + \cdots + (x_n y_1 - y_n x_1) \right|.$$

3.9 Computing Workspace's Boundary and Area from Initial Cable Configuration

The methods described hereafter were implemented in C++ and Discrete Optimal Search Library (DOSL) [4]. In the following sections below we mostly use 200 × 200 environment with two convex and one concave polygon as obstacles, and four control points placed at each corner of the environment. Some variant environments were also used.

If we have the initial cable configuration, we are able to compute the corresponding



(a) Input: initial cable configuration and robot, control points and obstacles

(b) Output: closed boundary and the area: 16200

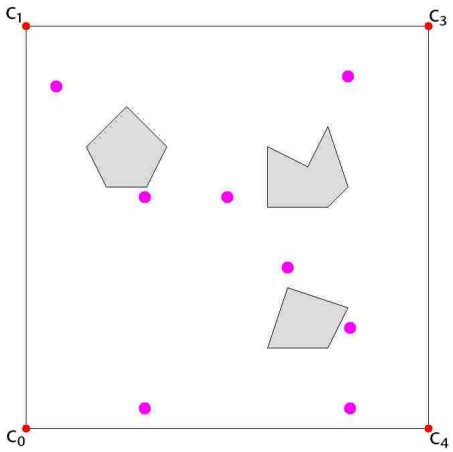
Figure 3.2: The workspace from given cable configuration

workspace. The cable b_i is given in form of vertices in visibility graph, going from the robot to the i^{th} control point. We need goal h-signatures $\mathfrak{h}_g^i$ of pairs of cables for searching for corresponding boundaries. We concatenate the inverse of the i^{th} cable's h-signature with the $i+1^{\text{th}}$ cable's, $\mathfrak{h}_g^i = (h(b_i))^{-1} \circ h(b_{i+1})$, where superscript "$-1$" indicates inverse operation explained in Section 2.5.

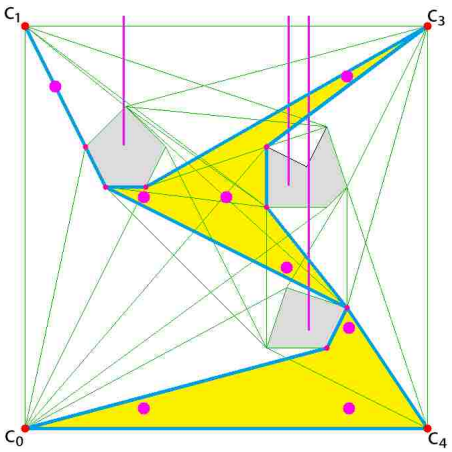
Use function $P(\mathbf{c}_i, \mathbf{c}_{i+1}, \mathfrak{h}_g^i)$ to get all corresponding shortest paths, then concatenate them into a closed boundary ω. For we have shown that cables can deform continuously into a closed boundary, no need to check the concatenation of goal h-signatures. In the end compute the area of this boundary, $A(\omega)$. Examples are shown in Figure 2.2 and 3.2.

3.10 Maximization of Workspace Covering Multiple Task Points

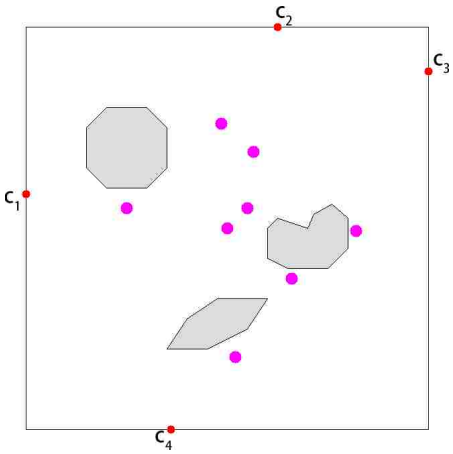
If we expect the robot to perform multiple tasks at static points in the environment, we should choose an appropriate initial cable configuration which can generate a workspace covering all task points. For this kind of problem, we need to use H-signature (homology signature) to check if all the task points are inside the workspace. If the components in the H-signature of boundary corresponding to task points are non-zero, that vertex is inside the boundary. If we get an H-signature that does not have any zero component, all task points are enclosed. Additionally, the task points which are right on



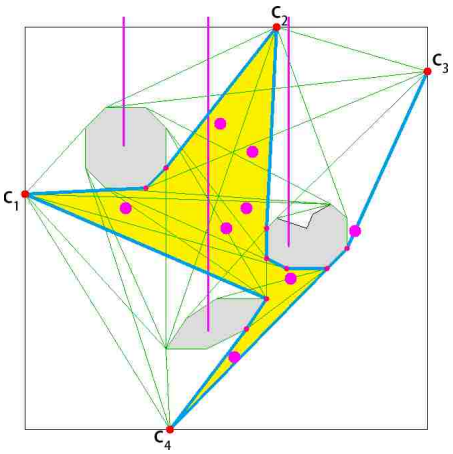
(a) Input: case 1



(b) Output: only one valid boundary when $\mathcal{L} = 1000$



(c) Input: case 2



(d) Output: only one valid boundary when $\mathcal{L} = 900$

Figure 3.3: Planning of workspace that covers multiple task points. Task points in purple, control points in red. In each case there is one task point right on the boundary of the workspace.

the boundary of the workspace are reachable too. They can also be counted as covered by the workspace. After getting all workspaces that satisfy the criteria and the areas of them, the algorithm returns the one that has the largest area. A case is shown in Figure 3.3.

3.11 Maximization of Expected Workspace's Area in an Environment with Moving Obstacle

In real-world scenarios, despite fixed control points, there could be moving obstacles. Although uncertain about the future locations of the obstacles, if we have prior knowledge of probabilities associated with different configurations of obstacles in the environment, we can choose a cable configuration with the max expectation of workspace's area.

3.11.1 Boundaries Change upon Obstacle Reconfiguration

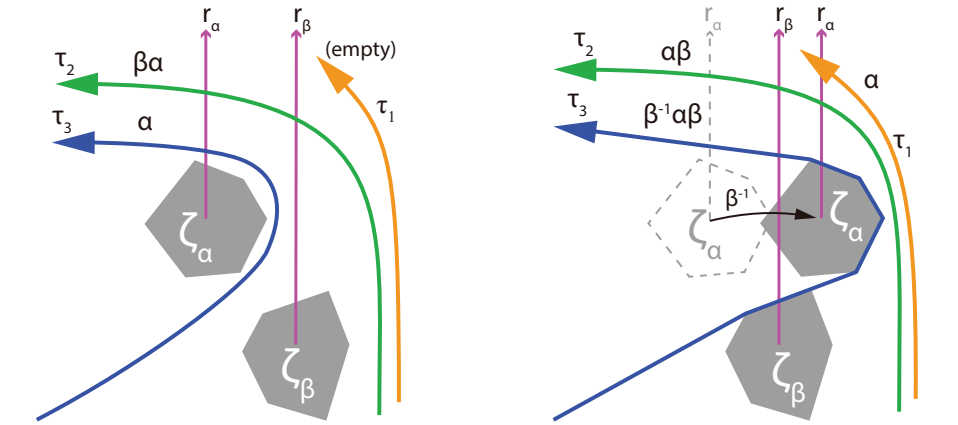
If the current workspace's boundary is ω_m, the m^{th} one in set $\mathcal{W}$ we previously established, when the obstacles move from current configuration to the j^{th} potential configuration, ω_m deforms as well into a new boundary ω_m^j , where $\omega_m = \{\tau_{m1}, \tau_{m2}, \ldots, \tau_{mn}\}$ and $\omega_m^j = \{\tau_{m1}^j, \tau_{m2}^j, \ldots, \tau_{mn}^j\}$ (for example, the initial configuration of Figure 3.5a deforms into Figures 3.5b, 3.5c and 3.5d when the obstacles move). Because control points do not move and cables do not intersect obstacles, τ_{mk} has the same start and end vertices as τ_{mk}^j and is homotopic to τ_{mk}^j, where $\tau_{mk} \in \omega_m$, $\tau_{mk}^j \in \omega_m^j$, and $k = 1, 2, \ldots, n$. Based on the homotopy invariants, we are able to search for τ_{mk}^j,

$$\tau_{mk}^j = P(\mathbf{c}_k, \mathbf{c}_{k+1}, R_j(h(\tau_{mk}))) \tag{3.1}$$

thus the new boundary in the j^{th} obstacle configuration is

$$\omega_m^j = \mathbf{P}_j(\omega_m) = \left\{ \tau \,\middle|\, \begin{array}{l} \tau = P(\mathbf{c}_k, \mathbf{c}_{k+1}, R_j(h(\tau_{mk}))), \\ \text{where } \tau_{mk} \in \omega_m, k = 1, 2, \ldots, n \end{array} \right\} \tag{3.2}$$

where function $R_j(\cdot)$ is for revising the h-signature of the j^{th} potential configuration. We use $R_j(h(\tau_{mk}))$ instead of $h(\tau_{mk})$ because sometimes $h(\tau_{mk}^j) = h(\tau_{mk})$ may not hold (Figure 3.4). The motion of obstacles could be so significant that the rays emanating from obstacles interchange of their coordinates. As a result, although τ_{mk}^j and τ_{mk} belong to the same homotopy class, the h-signatures of them in terms of crossing rays may be different. Hence we need to revise h-signatures.



(a) Before significant motion of obstacle

(b) Obstacle ζ_α right crosses ζ_β's ray r_β along trajectory of 'β^{-1}'

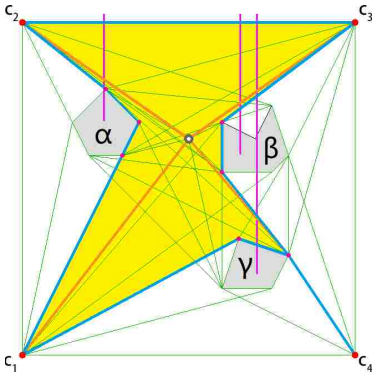
Figure 3.4: h-signature revision

3.11.2 Revision of h-signatures

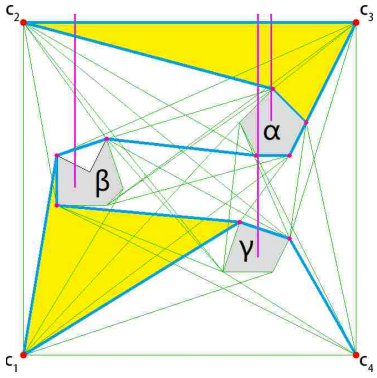
We revise h-signatures depending on how the obstacles move. Assume that when we are calculating the h-signatures, there will not be any overlap or intersecting between the rays of obstacles, and between rays and control points. The revision applies in four steps: add/remove, interchange, insert and simplify. The following is an example of revision triggered by the representative point ζ_α crossing the ray of the representative point ζ_β from left to right, namely, the trajectory along which ζ_α moves has an h-signature of 'β^{-1}', shown in Figure 3.4.

Add/remove When the representative point ζ_α's ray r_α moves from the left of the end vertex of one path to its right, we add a corresponding letter 'α' at the back of the path's h-signature, like τ_1 in Figure 3.4. Likewise, if it is the start vertex that any representative point's ray crosses, add/remove that letter at the front of path's h-signature.

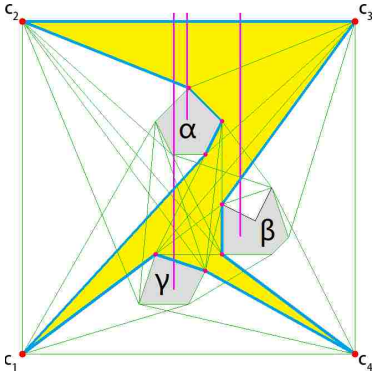
Interchange In the h-signature of the path, when 'α' is next to 'β', we interchange positions of 'α' and 'β', like τ_2 in Figure 3.4. Likewise, if 'α^{-1}' is next to 'β^{-1}', interchange positions of 'α^{-1}' and 'β^{-1}'.



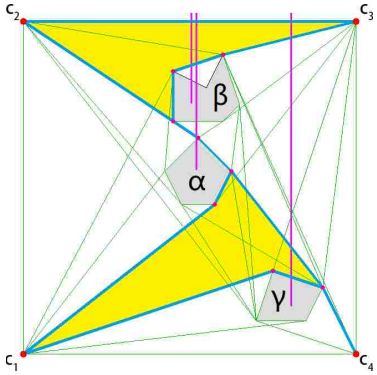
(a) Current configuration, area: 16200, max cable length sum: 1138.39



(b) Potential configuration 1, $\mathfrak{p}_1 = 0.5$, area: 10250, max cable length sum: 2155.56



(c) Potential configuration 2, $\mathfrak{p}_2 = 0.3$, area: 11400, max cable length sum: 1241.1



(d) Potential configuration 3, $\mathfrak{p}_3 = 0.2$, area: 9450, max cable length sum: 1364.59

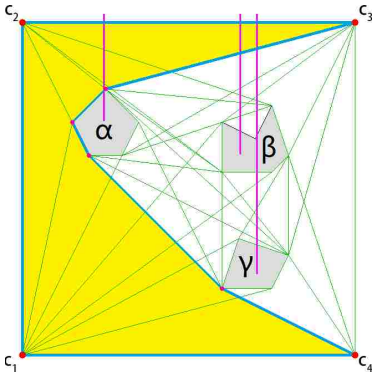
Figure 3.5: Boundary of Figure 3.2b changes in potential configurations, expectation: 10435, max cable length sum: 2155.56

Insert pair If 'α' or 'α^{-1}' appears alone (its previous and next letters are not 'β' or 'β^{-1}'), insert 'β^{-1}' into its left and 'β' into its right, like τ_3 in Figure 3.4.

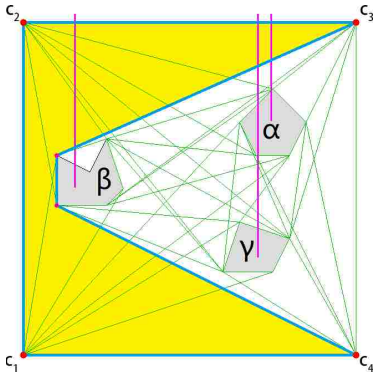
Simplify Check if a letter and its inverse appear side-by-side. If so, cancel both of them. Keep checking until there is no such a case.

On the other way around, when representative point ζ_α crossed r_β's right to left, going along 'β', do "add/remove", "interchange" and "simplify" in the same way described above; but when coming to "insert pair", if there is a lone 'α' or 'α^{-1}', we insert 'β' into its left and 'β^{-1}' into its right.

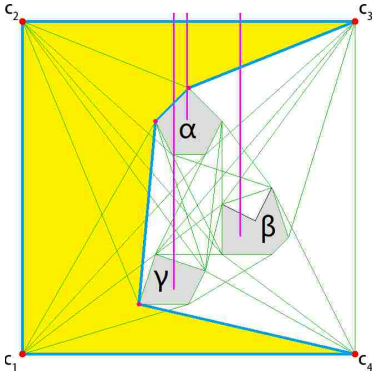
A representative point may cross multiple rays. Thus we have to decompose the crossing into several stages of coordinates interchanges, then revise h-signatures stage



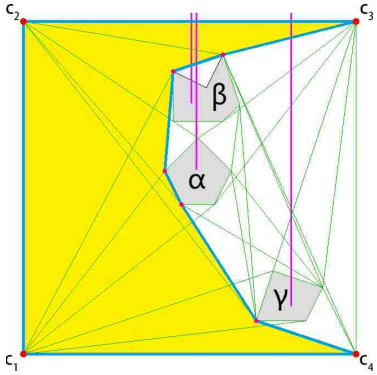
(a) Current configuration, area: 19300, max cable length sum: 1435.66



(b) Potential configuration 1, $\mathfrak{p}_1 = 0.5$, area: 19300, max cable length sum: 1488.13



(c) Potential configuration 2, $\mathfrak{p}_2 = 0.3$, area: 20100, max cable length sum: 1374.21



(d) Potential configuration 3, $\mathfrak{p}_3 = 0.2$, area: 22925, max cable length sum: 1283.95

Figure 3.6: A boundary with max expectation (20265) in an environment with moving obstacles, max cable length sum: 1488.13

by stage till reaching the final configuration. E.g. in Figure 3.6, we can split the transformation from Figure 3.6a to 3.6b into two stages: firstly ζ_α going along a trajectory 'β^{-1}', secondly ζ_α going along 'γ^{-1}'; transformation from Figure 3.6a to 3.6b in two stages: ζ_β going along 'γ^{-1}', then ζ_α going along 'γ^{-1}'; transformation from Figure 3.6a to 3.6d in one stage: ζ_β going along 'α'. Here we use the boundary in Figure 3.2b as a current configuration to illustrate how h-signatures are revised for potential configurations, shown in TABLE 3.1, and how the boundary changes accordingly in Figure 3.5.

3.11.3 Expectation Computation

The probability of the i^{th} potential configuration is denoted as $\mathfrak{p}_i, i = 1, 2, \ldots, \rho$, where ρ is the number of potential configurations and $\sum_{i=1}^{\rho} \mathfrak{p}_i = 1$. The area expectation

Table 3.1: Stage-by-stage h-signature revision for paths in Figure 3.5

Configuration	Stage	$\widetilde{\mathbf{c}_1\mathbf{c}_2}$	$\widetilde{\mathbf{c}_2\mathbf{c}_3}$	$\widetilde{\mathbf{c}_3\mathbf{c}_4}$	$\widetilde{\mathbf{c}_4\mathbf{c}_1}$
Current	N/A	α	$\alpha^{-1}\beta^{-1}\gamma^{-1}$	$\gamma\beta\gamma^{-1}$	γ
Potential 1	1	$\beta^{-1}\alpha\beta$	$\beta^{-1}\alpha^{-1}\gamma^{-1}$	$\gamma\beta\gamma^{-1}$	γ
	2	$\beta^{-1}\gamma^{-1}\alpha\gamma\beta$	$\beta^{-1}\gamma^{-1}\alpha^{-1}$	$\gamma\beta\gamma^{-1}$	γ
Potential 2	1	α	$\alpha^{-1}\gamma^{-1}\beta^{-1}$	β^{*}	γ
	2	$\gamma^{-1}\alpha\gamma$	$\gamma^{-1}\alpha^{-1}\beta^{-1}$	β	γ
Potential 3	1	α	$\beta^{-1}\alpha^{-1}\gamma^{-1}$	$\gamma\alpha\beta\alpha^{-1}\gamma^{-1}$	γ

* Initially it was '$\beta\gamma\gamma^{-1}$' before simplification.

of the m^{th} valid boundary, $\omega_m = \{\tau_{m1}, \tau_{m2}, \ldots, \tau_{mn}\} \in \mathcal{W}$, is

$$E(\omega_m) = \sum_{j=1}^{\rho} A(\mathbf{P}_j(\omega_m))\mathfrak{p}_j,$$

where function $\mathbf{P}_j(\omega_m)$ is defined by Equation (3.2) in Section 3.11.1. Next we can choose a valid boundary with the max expectation from set $\mathcal{W}$.

3.11.4 Maximum Cable Length Computation

In order to ensure that cables are long enough for all potential configurations, we need to know the maximum length of each cable. Since the workspace is a polygon that must have convex vertices at control points, the max length from the i^{th} control point is used when it reaches one of the other control points. In a particular obstacle and workspace configuration, that is

$$\max\{C(\widetilde{\mathbf{c}_i\mathbf{c}_1}), C(\widetilde{\mathbf{c}_i\mathbf{c}_2}), ..., C(\widetilde{\mathbf{c}_i\mathbf{c}_{i-1}}), C(\widetilde{\mathbf{c}_i\mathbf{c}_{i+1}}), ..., C(\widetilde{\mathbf{c}_i\mathbf{c}_{n-1}}), C(\widetilde{\mathbf{c}_i\mathbf{c}_n})\},$$

where $\widetilde{\mathbf{c}_i\mathbf{c}_j}$ is the shortest path between the i^{th} and the j^{th} control points inside the workspace. To get the goal h-signatures for searching, we concatenate the h-signatures tracing the boundary from the i^{th} control point to the j^{th} in clockwise or counter-clockwise direction. For instance, if $j > i$, the shortest path between $\mathbf{c}_i$ and $\mathbf{c}_j$ is $\widetilde{\mathbf{c}_i\mathbf{c}_j} = P(\mathbf{c}_i, \mathbf{c}_j, h(\tau_i) \circ h(\tau_{i+1}) \circ \cdots \circ h(\tau_{j-1}))$. In Figure 3.5 and Figure 3.6, the sum of four maximum cable lengths were calculated for each configuration.

Chapter 4

Algorithm Design for Robot Path Planning within Workspace

We move a robot from one point to another by controlling the motors to change the cables' lengths at each one's desired speed. The cable control algorithm has the following inputs:

(1) h-augmented graph of environment;

(2) coordinate of start and goal vertices, $\mathbf{v}_s$ and $\mathbf{v}_g$;

(3) coordinate of control points (x_c^i, y_c^i);

(4) initial cable configuration b_i;

(5) desired robot speed v_r;

and outputs:

(1) shortest trajectory from the start vertex to the end vertex;

(2) velocities of each cable v_c^i over time;

where $i = 1, 2, \ldots, n$.

4.1 h-signature of Shortest Trajectory within Boundary

A correct h-signature can ensure the path search outcome is within the workspace's boundary. If $\mathbf{v}_s$ and $\mathbf{v}_g$ are both on the boundary, we can directly obtain the h-signature when tracing the boundary from $\mathbf{v}_s$ to $\mathbf{v}_g$, since this part of the boundary can continuously deform into (homotopic to) the desired shortest path. If any of them is inside the boundary, we can use *alternative vertices* —points ($\mathbf{v}'_s$ and $\mathbf{v}'_g$) on the boundary adjacently above $\mathbf{v}_s$ and $\mathbf{v}_g$, shown in Figure 4.1. $\widetilde{\mathbf{v}_s\mathbf{v}_g}$ inside the boundary can continuously deform into $\widetilde{\mathbf{v}_s\mathbf{v}'_s\mathbf{v}'_g\mathbf{v}_g}$ partially on the boundary—they are in the identical homotopy class:

$$h(\widetilde{\mathbf{v}_s\mathbf{v}_g}) = h(\widetilde{\mathbf{v}_s\mathbf{v}'_s\mathbf{v}'_g\mathbf{v}_g}) = h(\widetilde{\mathbf{v}_s\mathbf{v}'_s}) + h(\widetilde{\mathbf{v}'_s\mathbf{v}'_g}) + h(\widetilde{\mathbf{v}'_g\mathbf{v}_g}).$$

Since path $\widetilde{\mathbf{v}_s\mathbf{v}'_s}$ and $\widetilde{\mathbf{v}'_g\mathbf{v}_g}$ are vertical, they do not cross any rays, having empty h-signatures. Hence,

$$h(\widetilde{\mathbf{v}_s\mathbf{v}_g}) = h(\widetilde{\mathbf{v}'_s\mathbf{v}'_g}).$$

The shortest path from $\mathbf{v}_s$ to $\mathbf{v}_g$ is $\widetilde{\mathbf{v}_s\mathbf{v}_g} = P(\mathbf{v}_s, \mathbf{v}_g, h(\widetilde{\mathbf{v}'_s\mathbf{v}'_g}))$. Besides, we can use $\mathbf{v}''_s$ and $\mathbf{v}''_g$ (below $\mathbf{v}_s$ and $\mathbf{v}_g$, respectively) instead. In addition, we can apply this method for obtaining cables' new h-signatures while the robot is moving, by replacing $\mathbf{v}_s$ with $\mathbf{v}_r$ (the robot's location) and $\mathbf{v}_g$ with $\mathbf{c}_i$. The cable trajectory from the robot $\mathbf{v}_r$ to control point $\mathbf{c}_i$ is $\widetilde{\mathbf{v}_r\mathbf{c}_i} = P(\mathbf{v}_r, \mathbf{c}_i, h(\widetilde{\mathbf{v}'_r\mathbf{c}_i}))$.

4.2 Cable Velocity

The cable velocity is the projection of the robot's velocity in the cable's direction. Denote coordinates of the i^{th} control point and that of the robot as (x_c^i, y_c^i) and (x_r, y_r), and the speed of robot as (v_r^x, v_r^y). The velocity of cable attached to the i^{th} control point is

$$v_c^i = \frac{(x_c^i - x_r)v_r^x + (y_c^i - y_r)v_r^y}{\sqrt{(x_c^i - x_r)^2 + (y_c^i - y_r)^2}}.$$

where v_c^i is a scalar and its positive direction is from the robot to the i^{th} control point, and robot coordinates (x_r, y_r) are integrals of its velocity over time plus start coordinates.

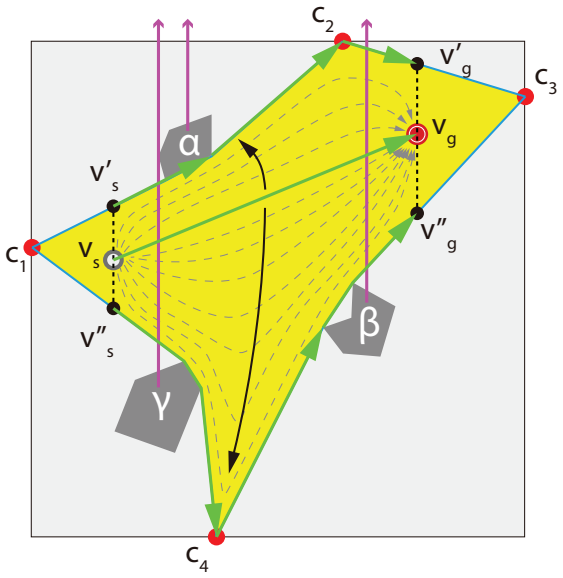
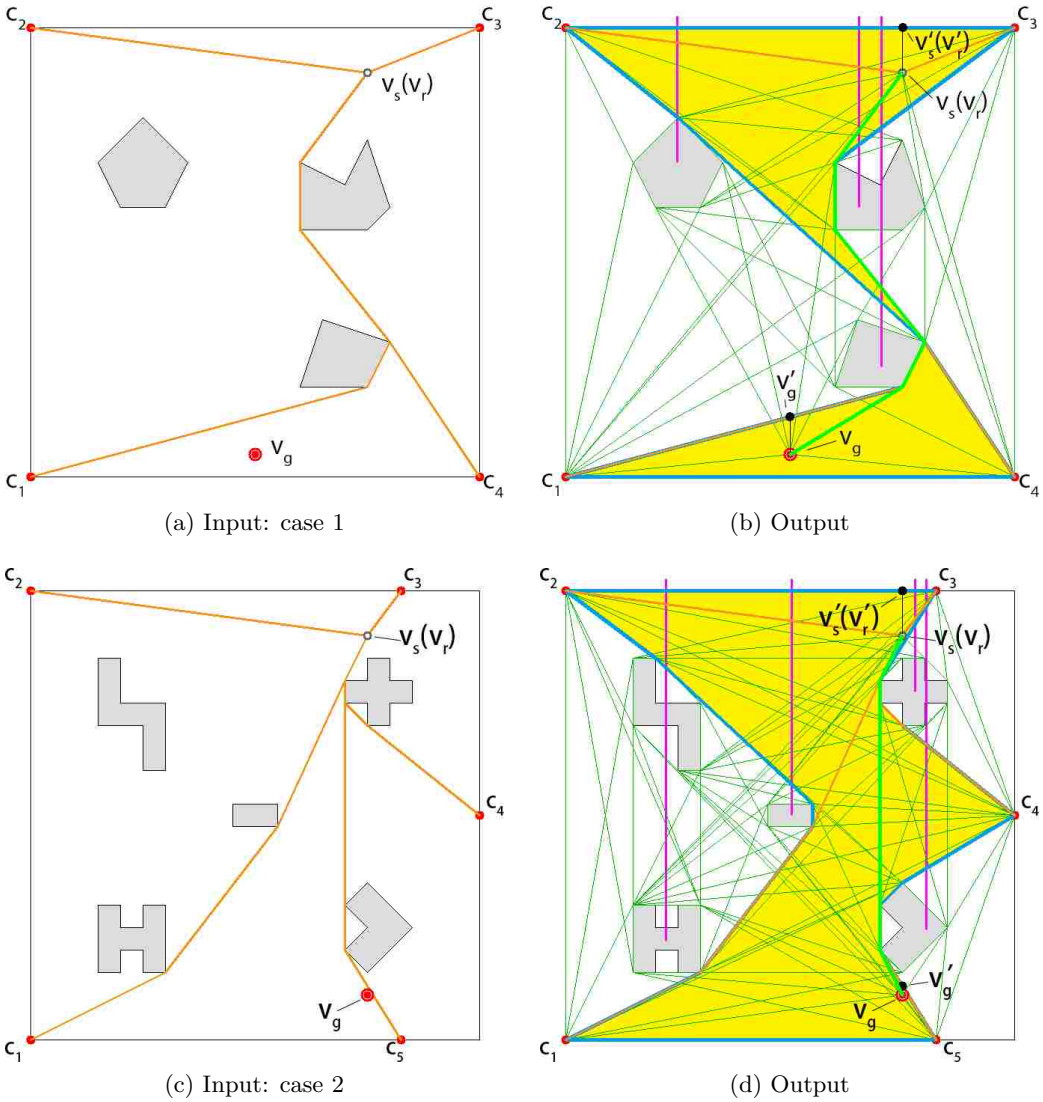


Figure 4.1: Shortest path searching within the workspace by using alternative vertices ($\mathbf{v}'_s$, $\mathbf{v}''_s$, $\mathbf{v}'_g$, $\mathbf{v}''_g$) at the boundary instead of start and goal vertices ($\mathbf{v}_s$, $\mathbf{v}_g$) to get the goal h-signature.

Sometimes the cable may go around an obstacle, which makes it no longer straight. Instead we use a temporary control point which is at the nearest turn of cables to the robot. Thus we search for a new cable configuration based on the robot's position. We compute the h-signature from robot's alternative vertex $\mathbf{v}'_r$ to each control point along the boundary in a same direction. In the test shown in Figure 4.2, we got $h(\widetilde{\mathbf{v}'_r\mathbf{c}_3})$ first, and then got $h(\widetilde{\mathbf{v}'_r\mathbf{c}_4})$, $h(\widetilde{\mathbf{v}'_r\mathbf{c}_1})$ and $h(\widetilde{\mathbf{v}'_r\mathbf{c}_2})$.

4.3 Path Planing for Multi-task Accomplishment

Suppose the cable-controlled robot needs to execute M unordered tasks, each described as static points in the workspace, before it arrives at the goal vertex. We need to solve the *traveling salesman problem*—find the shortest trajectory that goes through these static task points. One way to accomplish this is to construct a task indicator augmented graph and search in it [7]. A task indicator is a string of M binary digits, denoted by β, in which each bit is a flag or indicator of whether the corresponding task has been completed. For example, if there are 4 tasks to be finished, '0101' means that the 1st and the 3rd task is finished while others are not. The robot must set off at the start vertex with $\beta = 0000$ and arrive at goal vertex with $\beta = 1111$. The task indicator



(a) Input: case 1

(b) Output

(c) Input: case 2

(d) Output

Figure 4.2: Two examples of goal-directed path planning within the workspace. Path is colored in green. Alternative points at the boundary in black.

augmented graph $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t)$ is defined as

1. $\mathcal{V}_t = \{(\mathbf{v}, \beta) | \mathbf{v} \in \mathcal{V}, \beta \in \mathcal{Y}\}$

2. An edge $\{(\mathbf{v}, \beta) \rightarrow (\mathbf{v}', \beta')\} = P(\mathbf{v}, \mathbf{v}', h(\widetilde{(\mathbf{v})'(\mathbf{v}')'}))$ is in $\mathcal{E}_t$ for $(\mathbf{v}, \beta) \in \mathcal{V}_t$ and $(\mathbf{v}', \beta') \in \mathcal{V}_t$, $(\mathbf{v})'$ and $(\mathbf{v}')'$ being alternative vertices of $\mathbf{v}$ and $\mathbf{v}'$ respectively, iff one of the followings holds

   (a) The edge $\{(\mathbf{v}, \beta) \rightarrow (\mathbf{v}', \beta')\} \in \mathcal{E}$, and $\mathbf{v}' \notin \{\tau_l |$ the l^{th} bit of β is $0\}$, and $\beta = \beta'$

   (b) The edge $\{(\mathbf{v}, \beta) \rightarrow (\mathbf{v}', \beta')\} \in \mathcal{E}$, and $\mathbf{v}' \in \{\tau_l |$ the l^{th} bit of β is $0\}$, with $\mathbf{v}' = \tau_\lambda$, and $\beta \rightarrow \beta' \in \mathcal{Y}$ such that the λ^{th} bit of β' is 1.

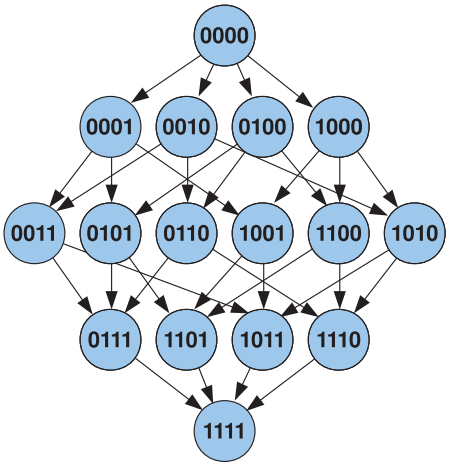
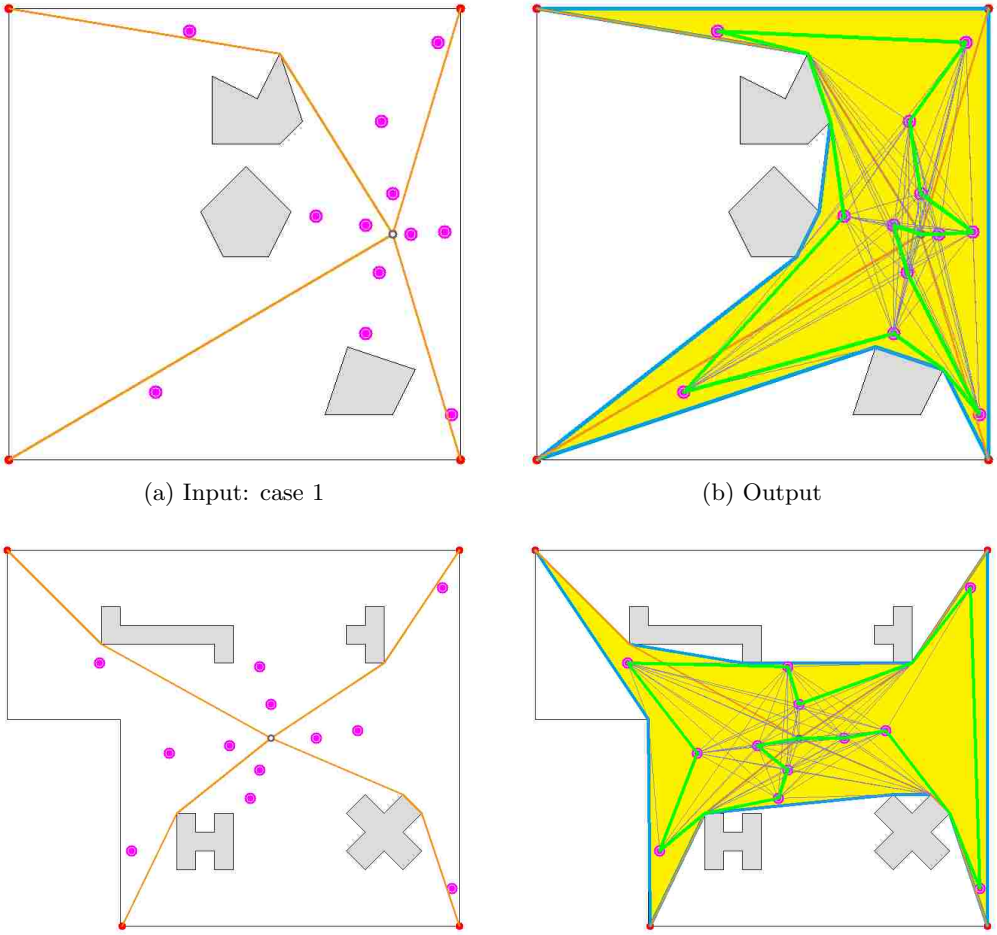


Figure 4.3: The task graph $\mathcal{Y}$ showing the possible transitions of the task indicator, β, for 4 tasks.

3. The cost associated with an edge $\{(\mathbf{v}, \beta) \rightarrow (\mathbf{v}', \beta')\}$ is the same as that associated with edge $\{\mathbf{v} \rightarrow \mathbf{v}'\} \in \mathcal{E}$.

In the example of Figure 4.4 we place 12 task points inside the workspace, the start vertex in the middle overlapping the end vertex. We find the workspace first then build the t-augmented graph to find the shortest path that visits all the task points.



(a) Input: case 1

(b) Output

(c) Input: case 2

(d) Output

Figure 4.4: Using t-augmented graph for multi-task planning within the workspace in a non-convex environment. The robot returns to the start node after finishing all 12 tasks along the green shortest trajectory in each case.

Chapter 5

Conclusion and Discussions

In this thesis, we have studied problems related to workspace and motion planing for cable-control robot in a cluttered environment, and established the topological relationship between the boundary of robot's workspace and the cable configuration, and introduced some novel and efficient methods for addressing these problems, using the h-signature augmented graph. We have presented some algorithms for workspace planning and trajectory planning based on these methods. Particularly, we have developed several applications in computation of boundary of workspace, maximization of workspace covering multiple given task points, and maximization of expected workspace's area with moving obstacles. We have also introduced some methods for trajectory planning and cable controlling of the robot within the workspace. Additionally, we have demonstrated them through simulations in environments with both static and stochastic dynamic obstacles. The following problems are within the scope of future research.

1. **Given the configuration of obstacles and the boundary of the environment, optimizing control points' locations for maximized area.**

   Since we have been able to optimize the cable configuration with the control points being fixed, the only thing in design problems left is the optimization of control points' locations which is as important as the configuration of cables. A straightforward way is to compute the area while moving the control points from corner to corner on the boundary of the environment, but it could be very expensive if there are multiple control points and a couple of possible cable configurations. We need to find a

more efficient algorithm to solve this problem which may or may not use topological methods.

2. **Extension of this planning problem to a full 3-D workspace and environment with columns or other kinds of obstacles.**

   In practical situations, the robots may need to move in a 3-D environment with more complicated obstacle configurations. We will be starting with definitions of homology and homotopy in 3-D environment, then analyzing forces and looking for any possible properties of workspace's boundary.

3. **Multiple cable-controlled robots planning and collaboration to manipulate objects in a 2-D/3-D workspace.**

   There could be multiple cable-controlled robots collaborating to finish a task. Their workspace may need to overlap for passing an object or manipulating an object together and in that case the cables may tangle together. We will design algorithms to keep each robot maneuverable and manage its workspace, ensuring the accomplishment of the tasks.

4. **Workspace planning of cable robots with gravity or other field forces as one form of locomotion in a 3-D workspace.**

   This can be regarded as a variant of the first problem. When the robot is in a gravitational/magnetic/electric field, we can use that field force as a cable pulling the robot in a desired direction (downward for gravity). The difference is that the field force can act wherever the robot is, which will enlarge the robot's workspace.

5. **Practical problems in applications.**

   In this thesis we have studied the planning problems for a cable-controlled robot mostly from a theoretical standpoint. We have not explicitly considered some of the implementational problems such as control error, environmental noise, cable elasticity and friction. Most of these can be accounted for by carefully designing feedback controllers for cable length control. In future research we will thus take such practical details into consideration and design more elaborate controllers for real-world implementations.

Besides theses applications, we believe that there will be more theoretical and implementational endeavors that can be undertaken as part of future research.

Bibliography

[1] Alireza Alikhani, Saeed Behzadipour, Aria Alasty, and S. Ali Sadough Vanini. Design of a large-scale cable-driven robot with translational motion. *Robotics and Computer-Integrated Manufacturing*, 27(2):357 – 366, 2011. Translational Research – Where Engineering Meets Medicine.

[2] Hatcher Allen. Algebraic topology, 2001.

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

[4] Subhrajit Bhattacharya. Discrete optimal search library (dosl): A template-based c++ library for discrete optimal search, 2017. Available at https://github.com/subh83/DOSL.

[5] Subhrajit Bhattacharya and Robert Ghrist. Path homotopy invariants and their application to optimal trajectory planning. *arXiv preprint arXiv:1710.02871*, 2017.

[6] Subhrajit Bhattacharya, Soonkyum Kim, Hordur Heidarsson, Gaurav S. Sukhatme, and Vijay Kumar. A topological approach to using cables to separate and manipulate sets of objects. *The International Journal of Robotics Research*, 34(6):799–815, 2015.

[7] Subhrajit Bhattacharya, Maxim Likhachev, and Vijay Kumar. Multi-agent path planning with multiple tasks and distance constraints. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 953–959. IEEE, 2010.

[8] Subhrajit Bhattacharya, David Lipsky, Robert Ghrist, and Vijay Kumar. Invariants for homology classes with application to optimal search and planning problem in

robotics. *Annals of Mathematics and Artificial Intelligence*, 67(3):251–281, Mar 2013.

[9] P. H. Borgstrom, B. L. Jordan, B. J. Borgstrom, M. J. Stealey, G. S. Sukhatme, M. A. Batalin, and W. J. Kaiser. Nims-pl: A cable-driven robot with self-calibration capabilities. *IEEE Transactions on Robotics*, 25(5):1005–1015, Oct 2009.

[10] P. Bosscher and I. Ebert-Uphoff. Wrench-based analysis of cable-driven robots. In *Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on*, volume 5, pages 4950–4955 Vol.5, April 2004.

[11] Samuel Bouchard, Clément Gosselin, and Brian Moore. On the ability of a cable-driven robot to generate a prescribed set of wrenches. *Journal of Mechanisms and Robotics*, 2(1):011010, 2010.

[12] Peng Cheng, Jonathan Fink, Vijay Kumar, and Jong-Shi Pang. Cooperative towing with multiple robots. *ASME Journal of Mechanisms and Robotics*, 1(1):011008–011008–8, 2008.

[13] M Dogar, Kaijen Hsiao, Matei Ciocarlie, and Siddhartha Srinivasa. Physics-based grasp planning through clutter. 2012.

[14] Bruce Randall Donald, Larry Gariepy, and Daniela Rus. Distributed manipulation of multiple objects using ropes. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation, ICRA*, pages 450–457, San Francisco, CA, USA, April 24-28 2000.

[15] Robert W Ghrist. *Elementary applied topology*. Createspace, 2014.

[16] Thomas C Harmon, Richard F Ambrose, Robert M Gilbert, Jason C Fisher, Michael Stealey, and William J Kaiser. High-resolution river hydraulic and water quality characterization using rapidly deployable networked infomechanical systems (nims rd). *Environmental Engineering Science*, 24(2):151–159, 2007.

[17] R. H.Teshnizi and D. A. Shell. Planning motions for a planar robot attached to a stiff tether. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2759–2766, May 2016.

[18] Brett L Jordan, Maxim A Batalin, and William J Kaiser. Nims rd: A rapidly deployable cable based robot. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 144–150. IEEE, 2007.

[19] Soonkyum Kim, Subhrajit Bhattacharya, and Vijay Kumar. Path planning for a tethered mobile robot. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, May 31 - June 7 2014.

[20] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[21] Moharam Habibnejad Korayem, H Tourajizadeh, and Mahdi Bamdad. Dynamic load carrying capacity of flexible cable suspended robot: robust feedback linearization control approach. *Journal of Intelligent & Robotic Systems*, 60(3-4):341–363, 2010.

[22] Yongguo Mei, Yung-Hsiang Lu, Y. C. Hu, and C. S. G. Lee. A case study of mobile robot's energy consumption and conservation techniques. In *ICAR '05. Proceedings., 12th International Conference on Advanced Robotics, 2005.*, pages 492–497, July 2005.

[23] Nathan Michael, Shaojie Shen, Kartik Mohta, Yash Mulgaonkar, Vijay Kumar, Keiji Nagatani, Yoshito Okada, Seiga Kiribayashi, Kazuki Otake, Kazuya Yoshida, Kazunori Ohno, Eijiro Takeuchi, and Satoshi Tadokoro. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. *Journal of Field Robotics*, 29(5):832–841, 2012.

[24] K.S. Pratt, R.R. Murphy, J.L. Burke, J. Craighead, C. Griffin, and S. Stover. Use of tethered small unmanned aerial system at berkman plaza ii collapse. In *Safety, Security and Rescue Robotics, 2008. SSRR 2008. IEEE International Workshop on*, pages 134–139, 2008.

[25] Rodney G. Roberts, Todd Graham, and Thomas Lippitt. On the inverse kinematics, statics, and fault tolerance of cable-suspended robots. *Journal of Robotic Systems*, 15(10):581–597, 1998.

[26] William Raymond Scott. *Group theory*. Courier Corporation, 2012.

[27] I. Shnaps and E. Rimon. Online coverage by a tethered autonomous mobile robot in planar unknown environments. *IEEE Transactions on Robotics*, 30(4):966–974, Aug 2014.

[28] Amarjeet Singh, Maxim A Batalin, Victor Chen, Michael Stealey, Brett Jordan, Jason C Fisher, Thomas C Harmon, Mark H Hansen, and William J Kaiser. Autonomous robotic sensing experiments at san joaquin river. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4987–4993. IEEE, 2007.

[29] Amarjeet Singh, Maxim A Batalin, Michael Stealey, Victor Chen, Mark H Hansen, Thomas C Harmon, Gaurav S Sukhatme, and William J Kaiser. Mobile robot sensing for environmental applications. In *Field and service robotics*, pages 125–135. Springer, 2008.

[30] N. Sydney, B. Smyth, and D. A. Paley. Dynamic control of autonomous quadrotor flight in an estimated wind field. In *52nd IEEE Conference on Decision and Control*, pages 3609–3616, Dec 2013.

[31] F. Takemura, M. Enomoto, T. Tanaka, K. Denou, Y. Kobayashi, and S. Tadokoro. Development of the balloon-cable driven robot for information collection from sky and proposal of the search strategy at a major disaster. In *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.*, pages 658–663, July 2005.

[32] R. H. Teshnizi and D. A. Shell. Computing cell-based decompositions dynamically for planning motions of tethered robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6130–6135, May 2014.

[33] Dimitris Theodorakatos, Ethan Stump, and Vijay Kumar. Kinematics and pose estimation for cable actuated parallel manipulators. In *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 8, pages 1053–1062, 2007.

[34] N. Zoso and C. Gosselin. Point-to-point motion planning of a parallel 3-dof underactuated cable-suspended robot. In *2012 IEEE International Conference on Robotics and Automation*, pages 2325–2330, May 2012.

Appendix:

List of Symbols and Notations

(In order of appearance)

$\mathbf{v}$	a vertex in a graph
$\mathbf{v}_s$	the start vertex
$\mathbf{v}_g$	the goal vertex
$\mathcal{G}$	a visibility graph
$\mathcal{V}$	the set of vertices in $\mathcal{G}$
$\mathbf{c}_i$	the i^{th} control point
$\{\mathbf{v} \to \mathbf{v}'\}$	an edge from $\mathbf{v}$ to $\mathbf{v}'$
$\mathcal{E}$	the set of edges in $\mathcal{G}$
X	a topological space
τ	a trajectory (with orientation)
$-\tau$ or τ^{-1}	an inverse of a trajectory (the same curve with reversed orientation)
X	a topological space
$\times$	Cartesian product
$\eta : A \to B$	mapping η from A to B
$\sqcup$	trajectory concatenation
$\pi_1(X)$	the first homotopy group of X
$H_1(X)$	the first homology group of X
$[\tau]$	a homotopy class the trajectory τ is in
$*$	free product

$\mathbb{Z}$	the set of integers
$\simeq$	isomorphism
$h(\tau)$	a function returning the homotopy signature of τ
$H(\tau)$	a function returning the homology signature of τ
ζ_i	the i^{th} representative point
r_i	the i^{th} ray (emanating from ζ_i)
$\mathcal{G}_h$	an homotopy signature augmented graph
$\mathcal{V}_h$	the set of vertices in $\mathcal{G}_h$
$\mathcal{E}_h$	the set of edges in $\mathcal{G}_h$
$\mathfrak{h}$	an h-signature
$\widetilde{\mathbf{vv}'}$	a trajectory from $\mathbf{v}$ to $\mathbf{v}'$ inside the workspace if not specified
$\circ$	h-signature concatenation
n	the number of control points
$\mathbf{T}_i$	the i^{th} searching thread
$\mathcal{P}_i$	the i^{th} set of shortest paths returned from $\mathbf{T}_i$
τ_{ij}	the j^{th} shortest path in $\mathcal{P}_i$
$\mathcal{L}$	a preset limit of perimeter of workspace
$\mathcal{C}$	the set of control points
$\mathcal{H}_i$	the set of h-signatures of paths returned from $\mathbf{T}_i$
$\mathfrak{h}_{ij}$	the j^{th} h-signature in $\mathcal{H}_i$ (corresponding to τ_{ij})
$C(\tau)$	a function returning the length of τ
$P(\mathbf{v}, \mathbf{v}', \mathfrak{h})$	a function returning a shortest path from $\mathbf{v}$ to $\mathbf{v}'$ with an h-signature of $\mathfrak{h}$
ω	a closed loop (boundary) constituted of a set of trajectories
$\mathcal{W}$	a set of boundaries (closed loops)
$A(\omega)$	a function returning the area enclosed by ω
$\mathfrak{h}^{-1}$	an inverse of $\mathfrak{h}$ (letters in reversed order with opposite superscripts)
$R_i(\mathfrak{h})$	a function returning the revised h-signature of $\mathfrak{h}$ for the i^{th} potential obstacle configuration
$\mathbf{P}_i(\omega)$	a function returning a new boundary deformed from ω for the i^{th} potential obstacle configuration

$\mathfrak{p}_i$	the probability of the i^{th} potential obstacle configuration
$\mathcal{Y}$	a task graph
$\mathcal{G}_t$	a task indicator augmented graph
$\mathcal{V}_t$	the set of vertices in $\mathcal{G}_t$
$\mathcal{E}_t$	the set of edges in $\mathcal{G}_t$

Vita

Xiaolong Wang was born in Dazhou, Sichuan, China on August 30th, 1988 to Xiaorong Ma and Fuying Wang. After graduating from Sichuan Hangtian High School in Chengdu, Sichuan, he attended Beihang University (formerly Beijing University of Aeronautics and Astronautics) in Beijing, China. In July of 2011 he received his Bachelor of Engineering in Mechanical Engineering and Automation, graduating with honors. As an undergraduate at Beihang, Xiaolong received two-second prizes of Fengru Cup of Beihang in mechanical design group and science paper group respectively in 2010, and the Scholarship of Science and Technology Competition in the same year. Xiaolong worked for four years as an Engine Process Planning Engineer for BAIC Motor Powertrain in Beijing, China, then he attended Lehigh to pursue a Master of Science in Mechanical Engineering under the supervision of Prof. Subhrajit Bhattacharya. Xiaolong plans to graduate with his Master of Science in May of 2018.