Motion Planning with Directional Constraints

by

Matin Rohani Larijani

A Thesis submitted to
the Faculty of Graduate Studies and Postdoctoral Affairs
in partial fulfilment of
the requirements for the degree of
Master of Science
in
Computer Science

School of Computer Science
Carleton University
Ottawa, Ontario, Canada
May 2014

Abstract

In this thesis motion planing with directional constraint in a polygon P possibly with holes is studied. We have studied this problem from two different viewpoints. We proposed an algorithm which directly computes a path that adheres to the directional constraints. We also provided an algorithm which approximates any piecewise linear path in the interior of P with a piecewise linear path that adheres to the directional constraints.

Contents

Abstract **ii**

Table of Contents **iii**

1 Introduction **1**

1.1 Motivation and Background 1

1.2 Problem statement 2

1.3 Thesis organization 3

2 Finding G_{Dir}-admissible paths in two dimensions **4**

2.1 Finding a G_{Dir}-admissible path in the plane 6

2.2 Finding a G_{Dir}-admissible path in a triangle 10

2.3 Finding a G_{Dir}-admissible path in a polygon 17

3 Approximating a non-admissible path by a G_{Dir}-admissible path **20**

3.1 Tubular neighborhoods and sleeves 21

3.2 Approximating a line segment if G_{Dir} is the complete graph 27

3.3 Approximating a line segment if G_{Dir} is strongly connected 36

3.4 Approximating a path with a G_{Dir}-admissible path 45

4 Future Works **52**

4.1 Approximation in higher dimensions 52

4.2 Direct calculation of G_{Dir}-admissible path and minimum turn 55

List of References **57**

Chapter 1

Introduction

1.1 Motivation and Background

Path planning algorithms have been the subject of intense investigation in past the fifty years. These efforts have been motivated mainly by robotics but this type of problems arise in many different disciplines and contexts from decision theory and artificial intelligence to control theory and robotics. The general theme of this category of problems is finding a path of consecutive states between an initial state and a goal state through a state space which has a notion of connectivity either defined by a graph or a topology. The path usually is required to satisfy certain constraints and optimality conditions.

Based on the type of the state space these problems can be categorized into discrete, continuous and hybrid group. In the case of discrete problems, the state space is a discrete space and connectivity and constraints are usually defined using a transition graph; this type of problems usually emerge in the context of decision theory. In the continuous case the state space is modeled as a submanifold of $\mathbb{R}^n$ and constraints defined by a set of polynomials. The free space, the set of states which satisfy the constraints, forms a semi-algebraic subset of $\mathbb{R}^n$. There are various types of constraints which can be applied to the path itself, for example the constraints imposed by the

kinematics of the system. This type of constraints often expressed using differential forms that the path should satisfy. The hybrid problems like the one arise in self configuring robots and planning with temporal goals use amalgamation of the two other methods.

Solutions to continuous motion planning problems, which is the subject of the work of this thesis, also can be categorized in two groups, sampling based methods and combinatorial methods. Sampling based methods such as PRM [1], RRT [2], PDST [3] and EST [4] have been widely used in modern robotic systems to solve motion planning problems with a variety of kinematics and differential constraints. This type of solutions are probabilistic in nature and are not guaranteed to provide a solution or even terminate. Combinatorial solutions are mostly based on building a one-dimensional road map using a finite decomposition of the state space. Various solutions to Piano Mover problems are an example of this type of solutions [5], [6]. Combinatorial approach to the solution of motion planning problems with kinematic and differential constraints is not studied as extensively as the sampling based methods. To name a few we can mention [7] in which the decidability of the motion planning problem with differential constraints is studied, [8] in which the complexity of approximating a non-constrained path with a constrained path is studied or [9], [10], [11], [12], [13], and [14] in which path planing with curvature constraint constraints is studied.

1.2 Problem statement

In this thesis we will study the path planning problem with directional constraint. We study the problem of path planning in the presence of differential constraint in very basic setting in which only a finite set of directions are allowed and a transition graph G_{Dir} defines how the directions are switched from one to another. Let $P \in \mathbb{R}^2$ be a polyhedral environment , $V = \{v_1, \cdots, v_n\}$ a discrete set of directions, and G_{Dir}

a transition graph which has V as its vertices and defines how directions can change, we study the problem of finding paths between any two point in P which adhere to the directional constraints imposed by the transition graph G_{Dir}. These constraints dictate that in each leg of the path we can only move in a direction from the set of allowed directions V and if we change our direction from v_1 to v_2 in V then there should be an edge from v_1 to v_2 in G_{Dir}. We call such a path a G_{Dir}-admissible path.

1.3 Thesis organization

In Chapter 2 we define some properties for G_{Dir} and establish a relationship between those properties and the problem of finding an admissible path. In the first section of this chapter we will study the problem of finding a G_{Dir}-admissible path between two points in the absence of any obstacle and show that if G_{Dir} is symmetric and spanning then we can find a G_{Dir}-admissible path with only one change of direction. We also study the problem of finding a G_{Dir}-admissible path in a triangle when the allowed directions are aligned with the x and y axes and G_{Dir} is a complete graph. Using the result from this section we show that the performance of any algorithm that solves the G_{Dir}-admissible path planning problem is sensitive to the start and end points.

In Chapter 3, we study the complexity and performance of the approximation of a piecewise linear path by a G_{Dir}-admissible path. We show that if G_{Dir} is symmetric, spanning and strongly connected then any piecewise linear collision free path can be approximated by a G_{Dir}-admissible path. We also show that the performance of the approximation is related to how dense the set of allowed direction is, i.e., the maximum angular difference between two consecutive direction in V.

Chapter 4 is dedicated to future works and different generalization of materials discussed in the previous Chapters.

Chapter 2

Finding G_{Dir}-admissible paths in two dimensions

In this chapter we first formally define the motion planning problem with directional constraints. We define some properties for G_{Dir} and show that if G_{Dir} have those properties we can find a G_{Dir}-admissible path when the environment is the whole $\mathbb{R}^2$. To solve our problem for polygonal environment we focus our attention to a triangle and the simplest spanning and symmetric direction transition graph G_{Dir}, i.e. complete graph over north, south, east and west directions, and show that motion planning with directional constraints can be solved in this setting. Next we generalize this results to any polygonal environment P using triangulation. If v_1 and v_2 are two arbitrary linearly independent directions in $\mathbb{R}^2$ then one can always find a linear isomorphism T which maps the ordered base (v_1, v_2) to $((1,0),(0,1))$. Since any triangulation of P is defines a natural triangulation for $T(P)$, we can generalize this results to any complete G_{Dir}.

Let P be a polygonal environment, i.e., a simple polygon, possibly with holes, and let n be the number of vertices of P. Let $V = \{v_1, \ldots, v_m\}$ be the set of admissible directions, and let G_{Dir} be the direction transition digraph of V. This graph has vertex set V and (v_i, v_j) is an edge if we can change direction to v_j if v_i is the current

direction.

We define the following properties of V and $G_{Dir}(V)$.

Definition 1. We call V symmetric if for each $v \in V$, $-v$ also belongs to V.

Definition 2. For each $v \in V$ we define $\delta^+(v) = \{v' : (v, v') \in E_{G_{Dir}}\}$.

Definition 3. G_{Dir} is spanning if for each vertex v_i, and for each $v_j \in \delta^+(v_i)$, $\{v_i, v_j\}$ is spanning $\mathbb{R}^2$.

Definition 4. If V is symmetric then we define G_{Dir} to be symmetric if for each $v \in V$, $v' \in \delta^+(v)$ if and only if $-v' \in \delta^+(v)$.

Definition 5. Given a pair (s, e) of points inside P, a piecewise linear path $R = (x_1 = s, x_2, \cdots, x_k = e)$ inside P is called G_{Dir}-admissible, if for each $1 \leq i < k$, there exists a real number $c_i > 0$ and an element v_i in V such that $x_{i+1} - x_i = c_i v_i$ and $v_{i+1} \in \delta^+(v_i)$. R is called strongly G_{Dir}-admissible if it is G_{Dir}-admissible and we have $x_2 - x_1 = c_2 v_s$ and $x_k - x_{k-1} = c_k v_e$ for given start and end directions (v_s, v_e) in V.

It can be easily observed that if G_{Dir} is symmetric then in Definition 5 we can assume $x_{i+1} - x_i = c_i v_i$ for an arbitrary real c_i.

Lemma 6. *Given a pair (s, e) of points inside P, a G_{Dir}-admissible path for a symmetric G_{Dir} is a piecewise linear path $R = (x_1 = s, x_2, \cdots, x_k = e)$ inside P such that for each $1 \leq i < k$, there exists a real number c_i and an element v_i in V such that $x_{i+1} - x_i = c_i v_i$ and $v_{i+1} \in \delta^+(v_i)$.*

The problem considered in this work can be expressed as:

Problem 7. *Given $V = \{v_1, \ldots, v_m\}$, a spanning direction transition graph G_{Dir} and a polygonal environment P. For any two query points (s, e), report a G_{Dir}-admissible path inside P if such a path exists, or report that such a path does not exist.*

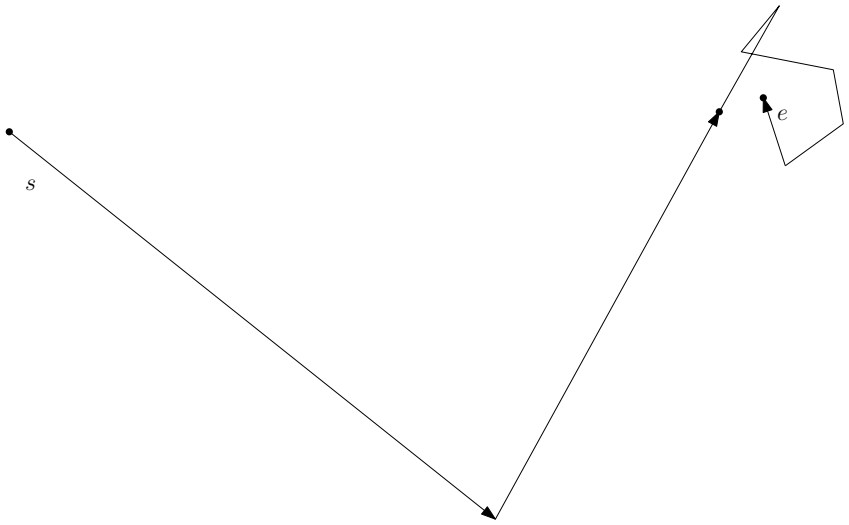


Figure 2.0.1: Sample run of algorithm 1 on (s, e), (v_s, v_e).

The strong version of Problem 7 becomes:

Problem 8. *Given $V = \{v_1, \ldots, v_m\}$, a spanning direction transition graph G_{Dir} and a polygonal environment P. For any two query points (s, e) and start and end directions (v_s, v_e) in V, report a strongly G_{Dir}-admissible path inside P if such a path exists, or report such a path does not exist.*

2.1 Finding a G_{Dir}-admissible path in the plane

If $P = \mathbb{R}^2$ (i.e., there are no constraints on the environment) and G_{Dir} is both symmetric and spanning and if each edge in G_{Dir} has unit weight then both the strong and weak versions of the problem can be solved in time that is within a constant factor of finding shortest path in G_{Dir}.

For simplicity we will assume that G_{Dir} is strongly connected. This assumption is not critical because if there is no path between v_s and v_e then there won't be any G_{Dir} - admissible path between s and e either. We also assume that $v_s \neq v_e$. This is to avoid finding a cycle that starts and ends at v_s. The same technique can be used

to solve the problem when $v_s = v_e$ by concatenating solutions for input data (s, c) and (v_s, v_c), and (c, e) and (v_c, v_s), for which $v_s \neq v_c$.

Algorithm 1: G_{Dir}-admissible path for $P = \mathbb{R}^2$

Data: (s, e), (v_s, v_e) and $v_s \neq v_e$, G_{Dir}
Result: $StronglyG_{Dir}$-admissible path R between (s, e)
1 $SP \leftarrow GetShortestPath(G_{Dir}, v_s, v_e)$;
$p \leftarrow e$;
$R \leftarrow$ path consisting of p;
$l :=$ number of vertices of SP ;
2 **for** $i \leftarrow l$ **to** 2 **do**
  $v \leftarrow SP[i]$;
  $p \leftarrow p - v$;
3 Add p at the beginning of R;
end
$\binom{\alpha}{\beta} \leftarrow (v_s\ v)^{-1}(p - s)$;
Add $s + \alpha \cdot v_s + \beta \cdot v$ at the beginning of R;
Add $s + \alpha \cdot v_s$ at the beginning of R;
Add s at the beginning of R;

Algorithm 1 provides a solution to Problem 8 with the minimum number of direction changes. Given start and end points (s, e) and distinct start and end directions (v_s, v_e), it performs the following steps:

1. Calculate the path $\pi(v_s, v_e)$ between v_s and v_e in G_{Dir} having the minimum number of edges.

2. Starting from e and from the end v_e of $\pi(v_s, v_e)$. it traverses $\pi(v_s, v_e)$ backwards and for each vertex v_i, it moves a distance of v_i in the opposite direction of v_i.

3. In the last step when it is on the first edge of $\pi(v_s, v_e)$, i.e., (v_s, v_1), it calculates the distance we should take in direction v_1 and then v_s in order to reach s. This can also be achieved by finding the intersection of lines $l(p, v_1)$ and $l(s, v_s)$ which pass through p and s in direction v_1 and v_s respectively.

Theorem 9. *If $P = \mathbb{R}^2$ (i.e. there is no constraint on the environment) and G_{Dir} is both symmetric and spanning then Algorithm 1 solves both the strong and weak versions of the 2D problem within a constant factor of the running time of* GetShortestPath. *Here,* GetShortestPath *returns a shortest path between v_s and v_e in G_{Dir} according to a constant uniform edge weight (e.g., 1 for all edges). The path generated by Algorithm 1 has the minimum number of links.*

Proof: We will show that Algorithm 1 solves the strong case. The weak case can be converted to the strong case by adding the vector v_s in V with least angular difference with $\bar{se}$ (the line segment between s and e) and v_e as a vector in $\delta^+(v_s)$ with least angular difference with v_s.

Let $SP = (sp_1, \cdots, sp_l)$ in line 1 be the shortest path between $v_s(= sp_1)$ and $v_e(= sp_l)$ with length l. The for loop starting on line 2 iterates through the last $l-1$ nodes of SP and builds a primary path R between p and e. It is immediately observed that R is G_{Dir}-admissible by definition.

At line 3 it is guaranteed that $v \in \delta^+(v_s)$ and R is a G_{Dir}-admissible path between p and e. Since (v_s, v) is spanning we have $p - s = \alpha v_s + \beta v$ for some real numbers α and β, so $(s, s + \alpha v_s, s + \alpha v_s + \beta v)$ is a G_{Dir}-admissible path between s and p. Let $(v_s\ v)$ be a 2 by 2 matrix with column vectors v_s and v. Then $(v_s\ v)\binom{1}{0} = v_s$ and $(v_s\ v)\binom{0}{1} = v$. Assuming $T = (v_s\ v)^{-1}$ we have

$$T(p - s) = \alpha T(v_s) + \beta T(v) = \alpha \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

So α and β can be calculated by matrix formula in the Algorithm 1.

It is clear that the running time of the Algorithm is within a constant factor of the running time of algorithm GetShortestPath. It is also clear that each vertex in SP represent a link in R and as a result the number of vertices of SP is equal to the number of links in R. This shows that R is a G_{Dir}-admissible path with the minimum

number of links.

■

Both the result and complexity of the shortest path function is dependent on the distance function defined on G_{Dir}. If unit weight is considered, shortest paths can be computed in $O(E_{G_{Dir}} + V_{G_{Dir}})$ using breadth-first search. If different weighting is used Dijkstra's algorithm with Fibonacci heap can be used to achieve $O(E_{G_{Dir}} + V_{G_{Dir}} \log(V_{dir}))$. Various weights can be used on G_{Dir}. For our next result we will define the weight of an edge $(v_i, v_j) \in E_{G_{Dir}}$ to be $d(0, v_i)$ where d is any metric on $\mathbb{R}^2$.

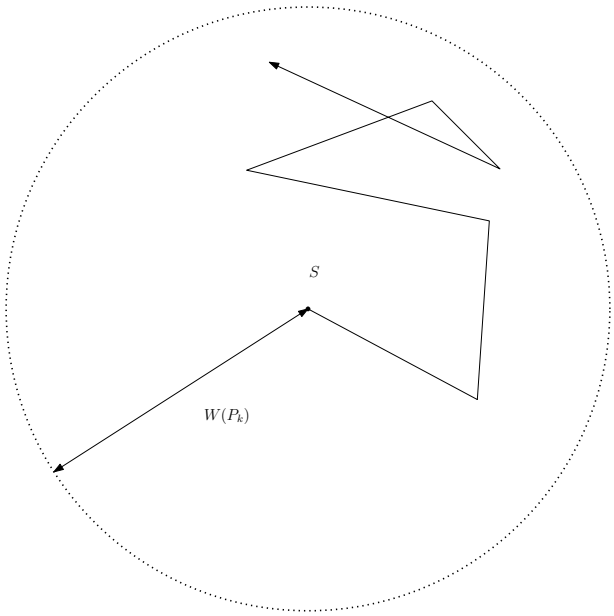


Figure 2.1.1:

Lemma 10. *Let d be a given metric on $\mathbb{R}^2$ and assume the weight of an edge $(v_i, v_j) \in E_{G_{Dir}}$ is $W_{((v_i,v_j))} = d(0, v_j)$. Then for any path $P = (v, v_1, \cdots, v')$ between v and v' in G_{Dir} and any given starting point $s \in \mathbb{R}^n$, the piecewise linear path $R = (r_0, \cdots, r_l)$, in which $r_0 = s$ and $r_i = r_{i-1} + v_i$, is contained in the close ball of radius $W(P) = \Sigma_{(v_i,v_j)\in P} d(0, v_i)$ centered at s.*

Proof: This statement is obvious if $l = 1$. Let us assume that this statement is correct for all paths up to length k. We will show that it is correct for $k+1$.

If the length of P is $k+1$ then for P_k, the first k vertices of P, the above statement is correct so we can assume that R_k, the fist k pieces of the associated piecewise linear path R, is entirely in a ball of radius $W(P_k)$ as depicted in figure 2.1.1. Now if $x \in \overline{r_k r_{k+1}}$ then $d(s,x) \leq d(s,r_k) + d(r_k,x)$ but from the induction hypothesis $d(s,r_k) \leq W(P_k))$ and we also have $d(r_k,x) \leq d(r_k,r_{k+1})$. So $d(s,x) \leq W(P)$ hence $x \in \overline{B_d(s,W(P_k))}$. ■

To avoid complications involved in computing the Euclidean norm due to calculating the square root of sums, the maximum (uniform, supremum) metric d_{Max} can be used where $d_{Max}(x,y) = max_i \mid x_i - y_i \mid$ for x_i, the i-th coordinate of x.

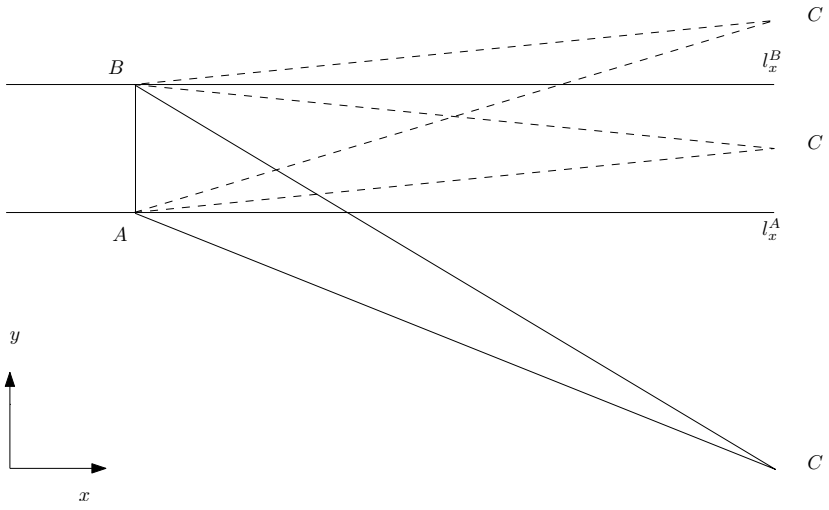


Figure 2.1.2: Position of triangle ABC in relation to l_x^A and l_x^B

2.2 Finding a G_{Dir}-admissible path in a triangle

Now we study the problem of finding a G_{Dir} admissible path in a triangle ABC. If v_1 and v_2 are two arbitrary linearly independent directions in $\mathbb{R}^2$ then one can always find a linear isomorphism T which maps the ordered base (v_1, v_2) to $((1,0),(0,1))$.

Since the image of ABC under T is also a triangle, for simplicity we assume that we are only allowed to move in 4 directions and $G_{dir} = K_4$, the complete digraph of size 4, and we are only moving in perpendicular directions. Let these 4 directions be denoted by $x = (1, 0), -x, y = (0, 1)$ and $-y$ to identify them with the normal $\bar{x}$ and $\bar{y}$ axis of the plane. We also assume that the edge AB of the triangle ABC is parallel to y (or $-y$). We also denote the x and y components of a point $p \in \mathbb{R}^2$ by x_p and y_p. A line passing through a point p in the direction of a vector $v \in \mathbb{R}^2$ is denoted by l_v^p. Consider two lines l_x^A and l_x^B parallel to direction x and passing through A and B respectively. Two different situations can happen depending on the placement of point C.

1. C is between l_x^A and l_x^B.

2. C is outside l_x^A and l_x^B.

If C is between l_x^A and l_x^B then any pair (s, e') of points inside ABC can be connected to each other with a G_{Dir}-admissible path of length 2. This is illustrated in Figure 2.2.1 for the case when s is to the left of e.

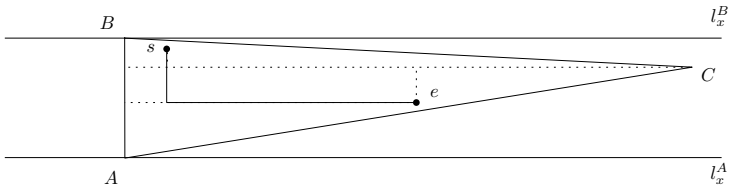


Figure 2.2.1: Finding an admissible path when C is between l_x^A and l_x^B

Lemma 11. *If ABC is a triangle such that the edge AB is parallel to y and vertex C is between l_x^A and l_x^B, then for any given points (s, e) there exist a G_{Dir}-admissible*

path of length 2 connecting s to e. Here, G_{Dir} is the complete graph of size 4 over $\{x, -x, y, -y\}$.

Now let us assume that C is outside of l_x^A and l_x^B. Without loss of generality we can assume that C resides below the line l_x^A as shown in Figure 2.2.2. Let $p_{-1} = B$ and $p_0 = A$ and for $k \geq 1$, $p_{2k-1} = l_x^{p_{2k-2}} \cap BC$ and $p_{2k} = l_y^{p_{2k-1}} \cap AC$. This generates an (infinite) triangulation of ABC with triangles $t_i = p_{i-1}p_ip_{i+1}$.

We have $\lambda = \|AB\| = |y_A - y_B|$ and also $\tan(\beta) = \|OA\|/\|OC\| = |A_x - C_x|/|y_A - y_C|$ and $\cot(\alpha) = \|OC\|/\|OB\| = |x_A - x_C|/|y_A - y_C|$. Using the similarity of triangles t_{2k} with BOC and t_{2k+1} with AOC one can calculate the $\bar{x}$ and $\bar{y}$ coordinates of each p_i.

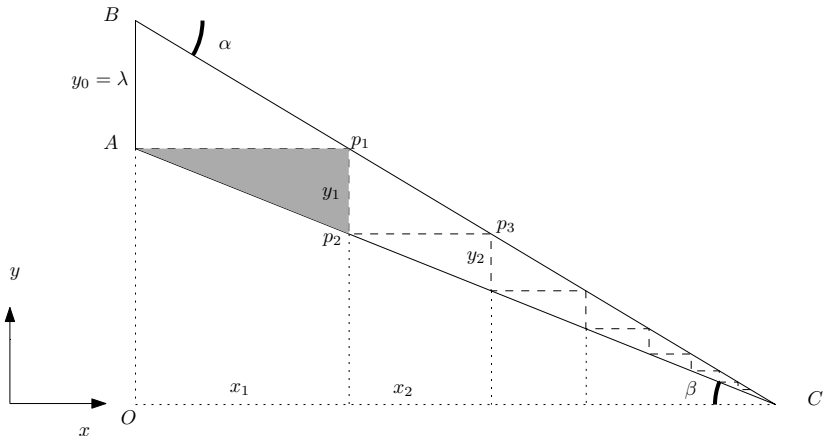


Figure 2.2.2: Partitioning a triangle

We have:

$$\begin{aligned}
|y_{p_{-1}} - y_{p_0}| &= \lambda \\
|x_{p_0} - x_{p_1}| &= \lambda \cot(\alpha) \\
|y_{p_1} - y_{p_2}| &= \lambda \cot(\alpha) \tan(\beta) \\
|y_{p_{2k-1}} - y_{p_{2k}}| &= \lambda (\cot(\alpha) \tan(\beta))^k \\
|x_{p_{2k}} - x_{p_{2k+1}}| &= \lambda (\cot(\alpha) \tan(\beta))^k \cot(\alpha)
\end{aligned}$$

Considering the above formulation the $\bar{y}$ coordinate of each point p_i can be calculated as a geometric series. We have $y_{p_{2k}} = y_B - \lambda \Sigma_{k=0}^{k} (\cot(\alpha) \tan(\beta))^k$ or

$$y_{p_{2k}} = y_B - \lambda \frac{1 - (\cot(\alpha) \tan(\beta))^{k+1}}{1 - \cot(\alpha) \tan(\beta)}$$

For any t_{2k} and $k \geq 0$, it can be observed that $p_{2k-1}p_{2k}$ is parallel to $\bar{y}$ and p_{2k+1} is between $l_x^{p_{2k-1}}$ and $l_x^{p_{2k}}$. For t_{2k+1} and $k \geq 0$, $p_{2k+1}p_{2k+2}$ is parallel to y and p_{2k} is between $l_x^{p_{2k+2}}$ and $l_x^{p_{2k+1}}$. This means that, for any $i \geq 0$, if (s, e) are points inside t_i then they can be connected to each other based on Lemma 11.

Now let us assume that (s, e) are given inside ABC the following algorithm calculates an admissible path between s and e.

Algorithm 2.

Input: *triangle ABC and start and end point (s, e)*

Output: *G_{Dir}- admissible path between s and e.*

1. *Select closest of s and e to the edge AB and denote it by s_0.*

2. *Shoot a ray $R(s_0, x)$ from s_0 parallel to x and away from AB until it hits BC at s_1. If l_y^e, the line passing through e and parallel to y, intersects with $R(s_0, x)$ inside ABC then report the admissible path and stop otherwise continue to the*

next step.

3. *Shoot a ray $R(s_1, y)$ from s_1 vertically down until it hits AC at s_2. If l_x^e, the line passing through e and parallel to x, intersects with $R(s_1, y)$ inside ABC then report the admissible path and stop otherwise set s_0 to s_2 and start from step 2.*

Figure 2.2.3 shows a run of the above algorithm.

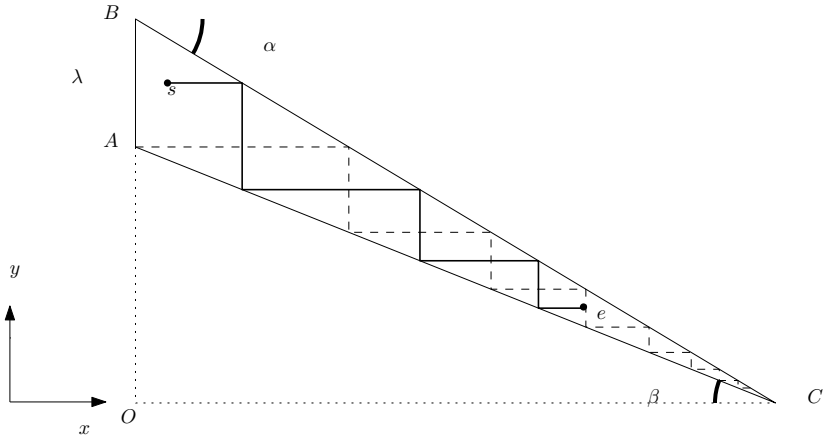


Figure 2.2.3: Finding a G_{Dir}-admissible path in a triangle

Let integers $l \leq m$ be selected such that $s \in t_l$ and $e \in t_m$. It is easy to show that algorithm 2 generates a G_{Dir}-admissible path. It is also easy to observe that if the path generated by algorithm 2 enters t_m it finishes. It can also be observed that any G_{Dir} - admissible path between (s, e) intersects all triangles t_q where $l \leq q \leq m$. This shows that algorithm 2 finishes in finite time. As a matter of fact for any point p in ABC we can find the index i_p of the triangle t_{i_p} to which p belongs. To do that first we find the index k for which we have $y_{p_{2k}} \geq y_p \geq y_{p_{2(k+1)}}$ or

$$y_B - \lambda \frac{1 - (\cot(\alpha)\tan(\beta))^{k+1}}{1 - \cot(\alpha)\tan(\beta)} \geq y_p \geq y_B - \lambda \frac{1 - (\cot(\alpha)\tan(\beta))^{k+2}}{1 - \cot(\alpha)\tan(\beta)}$$

Setting $\cot(\alpha)\tan(\beta) = OA/OB = y_A/y_B$ we have

$$(\frac{OA}{OB})^{k+1} \leq \ 1 - \frac{y_B - y_p}{y_B} \leq \frac{OA}{OB})^{k+2}$$

Since log is increasing we have $k = \lfloor \log_{OA/OB}(1 - \frac{y_B - y_p}{y_B}) \rfloor - 1$. Thus, p belongs to t_{2k} or t_{2k+1} (i.e. $2k \leq i_p \leq 2k+1$). By examining the value of x_p against $x_{p_{2k}}$ and $x_{p_{2k+1}}$ the index i_p can be exactly determined.

Based on the above discussion we require

$$|m - l| = |\lfloor \log_{OA/OB}(1 - \frac{y_B - y_s}{y_B}) \rfloor - \lfloor \log_{OA/OB}(1 - \frac{y_B - y_e}{y_B}) \rfloor| \pm 2$$

steps to finish algorithm 2. Figure 2.2.3 shows a run of algorithm 2.

Lemma 12. *If ABC is a triangle such that the edge AB of the triangle ABC is parallel to y and vertex C is outside l_x^A and l_x^B, parallel lines passing through A and B and parallel to x, then for any given points (s, e) there exist a G_{Dir} - admissible path with*

$$|\lfloor \log_{OA/OB}(1 - \frac{y_B - y_s}{y_B}) \rfloor - \lfloor \log_{OA/OB}(1 - \frac{y_B - y_e}{y_B}) \rfloor| \pm 2$$

links connecting s to e. Here, G_{Dir} is complete graph of size 4 over $\{x, -x, y, -y\}$.

Now let ABC be an arbitrary triangle in $\mathbb{R}^2$. We can partition ABC into two triangles each sharing a common edge parallel to y. This can be done by sorting the points based on their $\bar{x}$ coordinates and picking the middle point. Assuming C has such a property then the line l_x^C meets the line segment AB at C'. ACC' and BCC' partition ABC into two triangles that meet the requirements of Lemmas 11 and 12 since CC' is parallel to y. As a result algorithm 2 can be extended to general triangles as follows.

Algorithm 3.

Input: *triangle ABC and start and end point (s, e)*

output: *G_{Dir}- admissible path between s and e.*

1. *Order the vertices of ABC based on their x coordinates, select the middle vertex (assume C is the middle one) and shoot a line from C which meet AB at C'. Keep the resulting triangles ACC' and BCC'*

2. *Find T_1 the triangle that contains s, find T_2 the triangle which contains e.*

3. *If $T_1 = T_2$ then run algorithm 2 for T_1 and (s, e).*

4. *If $T_1 \neq T_2$ then from $\{C, C'\}$ select the one which has the closest y coordinate to e (assume C' does). Run algorithm 2 for T_1 and (s, C'), run algorithm 2 for T_2 and (C', e) and concatenate the two resulting paths.*

Figure 2.2.4 illustrates how algorithm 3 is applied for a case where $T_1 \neq T_2$. The above discussion can be formulated in the following lemma.

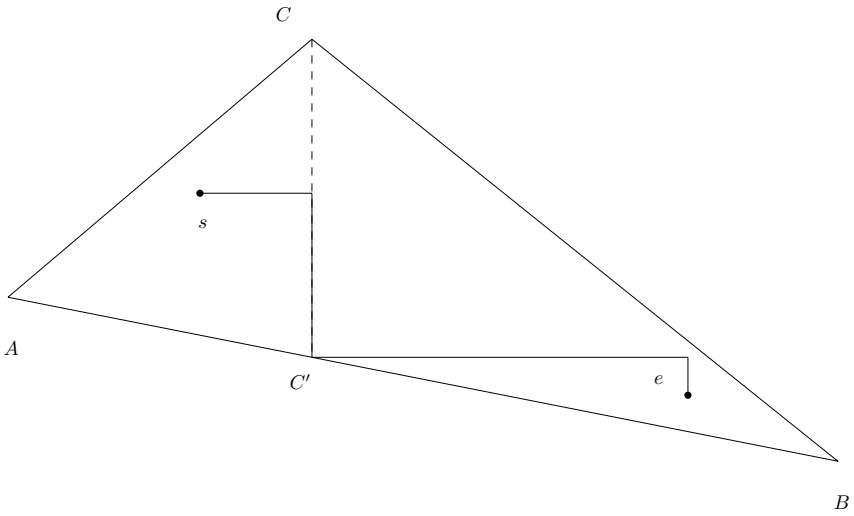


Figure 2.2.4: A line parallel to y splits triangle to two sections each in position required for Lemma 11 and 12

Lemma 13. *If ABC is a triangle in $\mathbb{R}^2$ such that $x_A \leq x_C \leq x_B$ and G_{Dir} is complete graph of size 4 over $\{x, -x, y, -y\}$, then for any given points (s, e), algorithm 3 calculates a G_{Dir} - admissible path of length l connecting s to e, where*

$$l \leq 2(|\lfloor \log_{y_C/y_{C'}}(1 - \frac{y_C - y_s}{y_C})\rfloor - \lfloor \log_{y_C/y_{C'}}(1 - \frac{y_C - y_e}{y_C})\rfloor| \pm 2).$$

2.3 Finding a G_{Dir}-admissible path in a polygon

Let P be a polygonal environment then P can be triangulated in $O(n\log(n))$ time with $O(n)$ storage (see [15] chapter 3 for details). Let $\{T_1, \cdots, T_m\}$ be such a triangulation for P. We define the weak dual graph $\mathbb{G}_P$ for the triangulation of P in which the vertex set is $V_{\mathbb{G}_P} = \{T_1, \cdots, T_m\}$ and $(T_i, T_j) \in E_{\mathbb{G}_P}$ is an edge iff T_i and T_j share an edge. It can easily be observed that $\mathbb{G}_P$ is a planar graph. As a result the size of $V_{\mathbb{G}_P}$ and $E_{\mathbb{G}_P}$ are in order of n, the size of P (since $m = O(n)$). It can easily be observed that each edge $e \in E_{\mathbb{G}_P}$ uniquely associated to an edge of the triangulation. For each edge $e \in E_{\mathbb{G}_P}$ we define m_e to be the midpoint of the triangle edge that e represents.

Based on triangulation $\{T_1, \cdots, T_m\}$ the following algorithm will find a G_{Dir} admissible path between any two points (s, e).

Algorithm 4.

Input: *Polygonal environment P, its triangulation $\{T_1, \cdots, T_m\}$ and start and end points (s, e)*

Output: *G_{Dir}- admissible path between s and e.*

1. *Find T_s and T_e, the triangles which contain s and e respectively.*
2. *If $T_s = T_e$ then run algorithm 3 for T_s and (s, e).*
3. *If $T_s \neq T_e$ then find a path Π between T_i and T_j in $\mathbb{G}_P$. Assuming $|\Pi| = l$ we denote each vertex in this path by $\Pi(i)$ where $\Pi(0) = T_s$ and $\Pi(l) = T_e$. If such a path does not exist then report that there is no G_{Dir}- admissible path between s and e.*

4. *Run algorithm 3 for triangle* $\Pi(0)$ *and* $(s, m_{(\Pi(0),\Pi(1))})$ *and concatenate the result to* R.

5. *For each* $\Pi(i)$ *while* $\Pi(i) \neq T_e$ *run algorithm 3 for triangle* $\Pi(i)$ *and* $(R_{End}, m_{(\Pi(i),\Pi(i+1))})$ *and concatenate the resulting path to* R. *(*R_{End} *is the last point in* R*).*

6. *If* $\Pi(i) = T_e$ *run algorithm 3 for triangle* $\Pi(l)$ *and* (R_{End}, e) *and concatenate the resulting path to* R *and report* R.

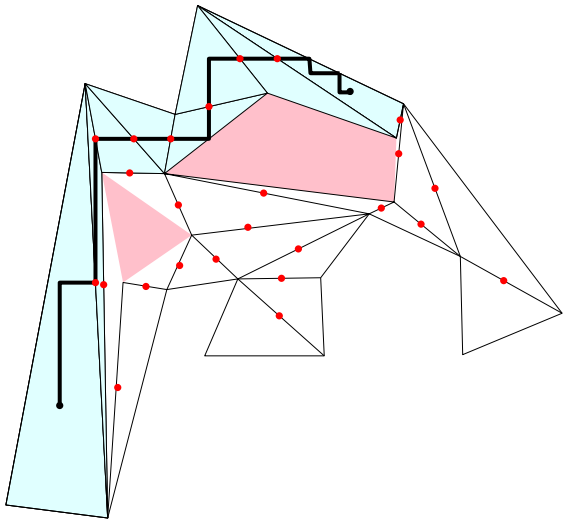


Figure 2.3.1: A sample run of Algorithm 4. The light blue section represents Π

Based on Lemma 13 it is clear that algorithm 4 finishes. It is also obvious that algorithm 4 generates a G_{Dir}- admissible path. If there exists a path from T_s to T_e then Algorithm 4 will generate a G_{Dir}- admissible path. Now if there exists a G_{Dir}-admissible path between s and e then this path corresponds to a walk W on $\mathbb{G}_P$. This walk W can be reduced to a path by removing all its cycles. Let Π_W be such a path for W then algorithm 4 generates a G_{Dir}-admissible path between s and e.

This shows that Algorithm 4 generates a G_{Dir}- admissible path between any given start and end points (s, e) if and only if such a path exists. So we have

Theorem 14. *If P is a polygonal environment in $\mathbb{R}^2$ and G_{Dir} is a complete graph of size 4 over $\{x, -x, y, -y\}$, then for any given points (e, e), Algorithm 4 calculates a G_{Dir} - admissible path connecting s to e.*

Chapter 3

Approximating a non-admissible path by a G_{Dir}-admissible path

The problem of finding a piecewise linear path between two points in a polygonal environment has been extensively studied in two and three dimensions. It has been shown in [16] and [15] that this problem can be preprocessed in polynomial time and path queries can be answered by searching for the shortest path in the preprocessed structure. In this chapter we will assume that we are given a polygonal environment P in $\mathbb{R}^2$, a direction transition graph $G_{Dir}(V, E)$, a pair of points (s, e) and an obstacle avoiding path $R = (r_0 = s, \cdots, r_l = e)$ in the interior of P. We will consider the problem of approximating R by a G_{Dir}-admissible path between s and e.

We first consider the case in which G_{Dir} is a complete graph and show that R can be approximated by a G_{Dir}-admissible path in this setting. Then we relax this restriction and study the problem for any strongly connected G_{Dir} and we conclude that Problem 8 can be solved in this configuration.

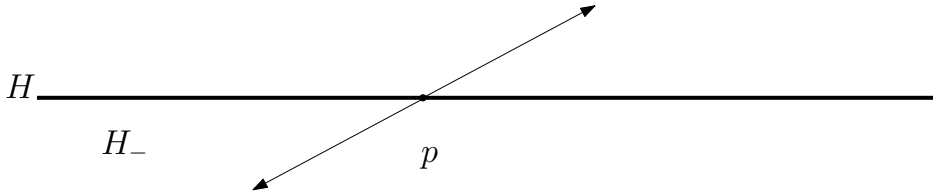


Figure 3.1.1: If G_{Dir} is spanning then there is at least one $v \in V_{G_{Dir}}$ such that $v \notin H$. For such a v either v or $-v$ is in H_-

3.1 Tubular neighborhoods and sleeves

Lemma 15. *If G_{Dir} is symmetric and spanning then for a given half plane $H = \{(x,y) : ax + by + c \leq 0\}$ and any point p on the boundary of H there exist a vertex v in $V_{G_{Dir}}$ such that $p + v = (x_p + x_v, y_p + y_v)$ lies in the open half plane H, $H_- = \{(x,y) : ax + by + c < 0\}$. i.e $a(x_p + x_v) + b(y_p + y_v) + c < 0$*

Proof: Since G_{Dir} is spanning there exists a vector $v \in V_{G_{Dir}}$ such that $p + v$ is not on the boundary of H. As illustrated in Figure 3.1.1, If $p + v$ is outside H (i.e. $a(x_v) + b(y_v) > 0$) then for $-v$ we have $a(x_{-v}) + b(y_{-v}) < 0$. This proves that either v of $-v$ has the desired property. ■

Definition 16. Let a and b be points in P such that the line segment $\overline{ab}$ is in P. A tubular neighborhood for a point $z \in \bar{ab}$ along $\bar{ab}$ in P is defined as,

$$Box(\bar{ab}, \lambda, z) = \{x + \beta\vec{c};\ x \in \bar{ab}, \|z - x\| < \lambda, -\lambda < \beta < \lambda\}$$

where $\vec{c}$ is the normal unit-vector of $\overline{ab}$. We denote its closure by

$$\overline{Box(\overline{ab}, \lambda, z)} = \{x + a\vec{c};\ x \in \bar{ab}, \|z - x\| \leq \lambda, -\lambda \leq a \leq \lambda\}.$$

We similarly define a tubular neighborhood along $\overline{ab}$ as

$$Box(\bar{ab}, \lambda) = \{x + a\vec{c};\ x \in \bar{ab}, -\lambda < a < \lambda\}.$$

Figure 3.1.2 depicts the tubular neighborhood of a point along a line and a line segment.

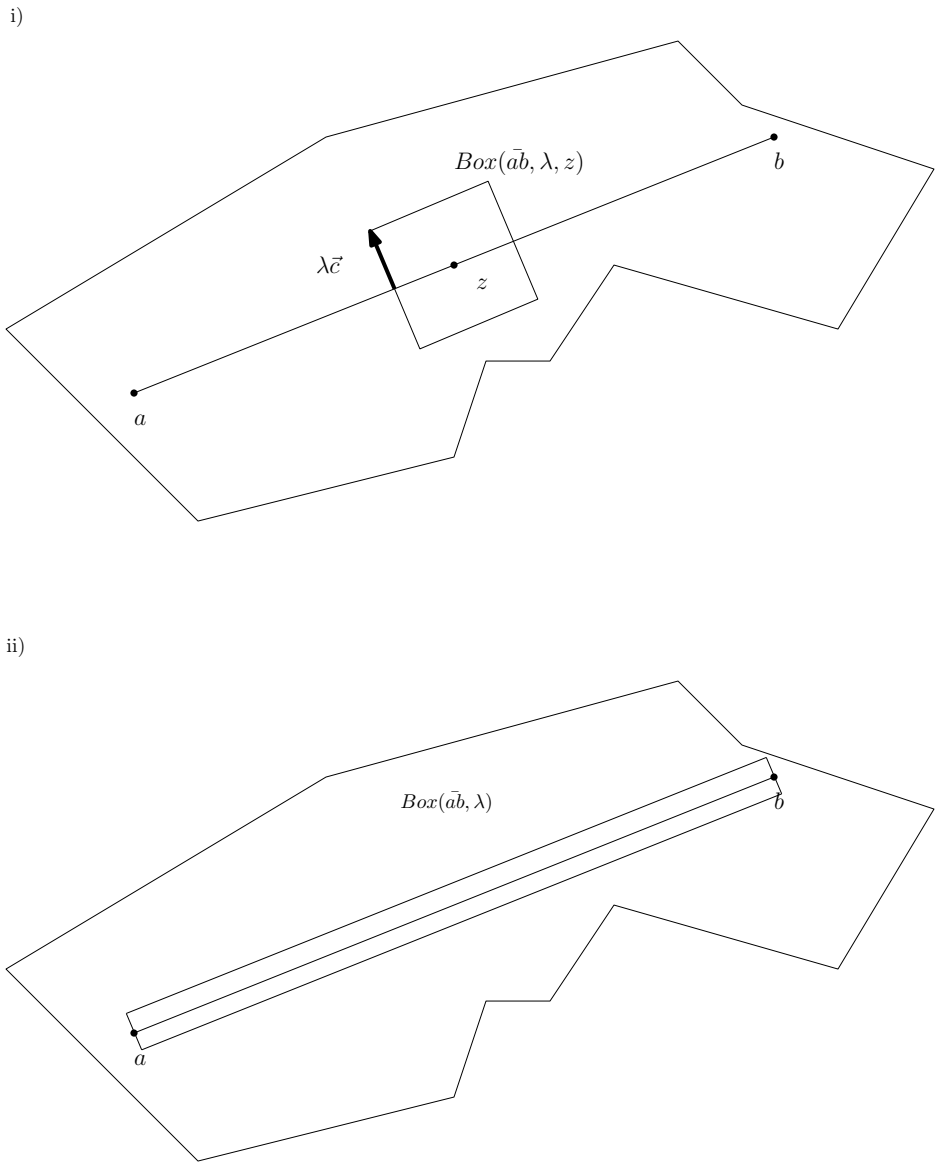


Figure 3.1.2: i) Tubular neighborhood of z along $\bar{ab}$ ii) tubular neighborhood of $\bar{ab}$

Let $R = (r_0 = s, \cdots, r_l = e)$ be a piecewise linear path in P which connect s and e. For each i, let λ_i be the minimum orthogonal distance of $\overline{r_i r_{i+1}}$ from the boundary ∂P of P, i.e,

$$\lambda_i = \min\{D_c(x) : x \in \overline{r_i r_{i+1}}\}$$

where

$$D_c(x) = \min\{\alpha : x + \alpha c \in \partial P\},$$

and c is a unit orthogonal vector to $\overline{r_i r_{i+1}}$. Compactness of $\overline{r_i r_{i+1}}$ guarantees that such a minimum exists. Any tubular neighborhood $Box(\overline{r_i r_{i+1}}, \lambda_i/2)$ of $\overline{r_i r_{i+1}}$ resides entirely in the interior of P.

This will define a collection of tubular neighborhoods for line segments of R,

$$\{\overline{Box(\overline{r_i r_{i+1}}, \lambda_i/2)};\ 0 \leq i < l\}$$

which are completely in P. We call this collection a sleeve for R. The following theorem can be concluded from the above observation.

Theorem 17. *For any piecewise linear path R inside the interior of a polygon P, there exist a sleeve for R such that its closure is completely in the interior of P.*

Next we will show that for a given polygonal environment P (possibly with some holes), a pair of points (s, e) and a piecewise linear path R between s and e, Algorithm 5 computes a sleeve for R which consists of tubular neighborhoods of the line segments of the path.

The basic idea is to find the orthogonal distance between each line segment to the boundary of the polygonal environment P. Basically Algorithm 5 loops through the line segments $\overline{r_i r_{i+1}}$ of piecewise linear path R. For each line segment:

1. It finds an orthonormal base (c_1, c_2) at r_i. c_1 is in the direction of $\overline{r_i r_{i+1}}$ and c_2 is orthogonal to c_1.

Algorithm 5: Tubular Neighborhood 1

Data: (s, e), a polygon $P \subset \mathbb{R}^2$, A piecewise linear path $R = (r_0 = s, \cdots, r_l = e)$ between s and e

Result: Array Λ of (λ_i) that defines a tubular neighborhood around the line segments of R with closure inside P

$\Lambda := Array(l)$;

for $0 \leq i < l$ **do**

`/* calculating an orthogonal frame along` $r_i\bar{r_{i+1}}$ `*/`

$c'_i \leftarrow (r_{i+1} - r_i / \|r_{i+1} - r_i\|)$;

$c_i \leftarrow GetNormalTo(c'_i)$;

`/* finding a rotation that maps` c'_i `to` $(1,0)$ `(i.e.,` $r_i\bar{r_{i+1}}$ `to x-axis) and` c_i `to` $(0,1)$ `*/`

$T \leftarrow \begin{bmatrix} c'_i & c_i \end{bmatrix}^{-1}$;

$start \leftarrow$ the one with smaller x in $\{T(r_i), T(r_{i+1})\}$;

$end \leftarrow$ the one with bigger x in $\{T(r_i), T(r_{i+1})\}$;

$y \leftarrow start.y$;

$min \leftarrow \infty$;

`/* find the closest points to the segment */`

1 **for** *each edge* $e = (p, p') \in P$ **do**

`/* Apply the rotation to` p `and` p' `*/`

$p_T \leftarrow ApplyTransform(T, p)$;

$p'_T \leftarrow ApplyTransform(T, p)$;

`/* if` $[p_T.x, p'_T.x] \subset [start.x, end.x]$ `*/`

if $start.x \leq p_T.x \leq end.x$ *and* $start.x \leq p'_T.x \leq end.x$ **then**

if $min \geq \mid p_T.y - y \mid$ **then** $min \leftarrow \mid p_T.y - y \mid$;

if $min \geq \mid p'_T.y - y \mid$ **then** $min \leftarrow \mid p'_T.y - y \mid$;

end

if $[p_T.x, p'_T.x] \not\subset [start.x, end.x]$ *and* $[p_T.x, p'_T.x] \cap [start.x, end.x] \neq \emptyset$ **then**

if $p.x < start.x$ **then**

$y_{start}(e) \leftarrow e(start.x)$;

if $min \geq y_{start}$ **then** $min \leftarrow y_{start}$;

end

if $p'.x > end.x$ **then**

$y_{end}(e) \leftarrow e(end.x)$;

if $min \geq y_{end}$ **then** $min \leftarrow y_{end}$;

end

end

end

$\Lambda(i) \leftarrow min/2$;

end

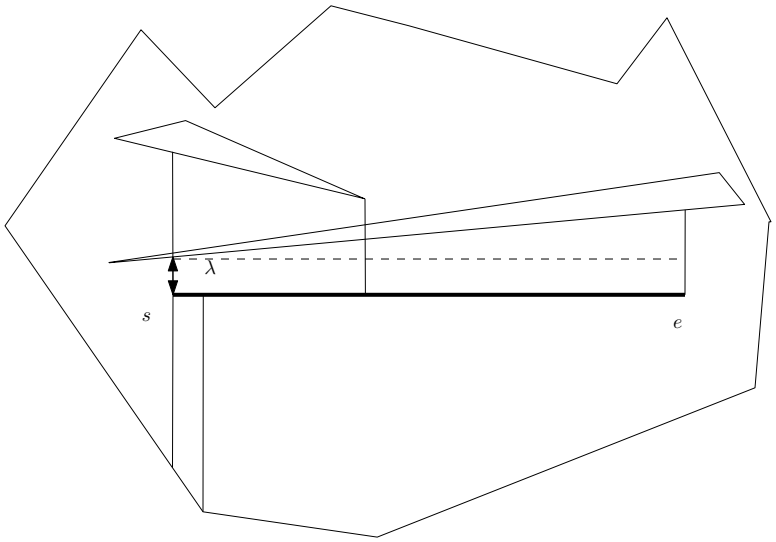


Figure 3.1.3: Sample run of Algorithm 5 for the path $R = (s, e)$.

2. The whole environment is rotated and translated in a such way that (c_1, c_2) maps to $((1,0),(0,1))$. As a result r_i is mapped to $(0,0)$ and r_{i+1} to $(0, \alpha_{i+1})$. (Figure 3.1.3)

3. For each edge $\overline{pp'} \in P_T$ we consider the projection of $\overline{pp'}$ on the x-axis, $\pi_x(\overline{pp'}) = [p_x, p'_x]$. Let $I = [p_x, p'_x] \cap [0, \alpha_{i+1}]$. We calculated the height of the end points of I when $\overline{pp'}$ is considered as a linear function over I and keep track of minimum of such heights. Since I is convex and $\overline{pp'} \in P_T$ considered as a line over I is both convex and concave the height of the end points have the extreme values. So the minimum of these extreme values defines the orthogonal distance of $\overline{r_i r_{i+1}}$ with P.

4. We report half of the minimum calculated in the above step as Λ_i.

Theorem 18. *Algorithm 5 computes a sleeve for R in $O(ln)$ time.*

Proof: Algorithm 5 is iterating over the line segments of R and for each line segment $\overline{r_i r_{i+1}}$ it finds a $\Lambda(i)$ such that the tubular neighborhood $\overline{Box(\overline{r_i, r_{i+1}}, \lambda)}$ resides in P. To do so for each line segment $\overline{r_i r_{i+1}}$, the algorithm finds the biggest λ_i such that

$Box(\overline{r_i, r_{i+1}}, \lambda_i)$ is in P. λ_i is equal to

$$\min\{Dist(x, x + \alpha\vec{c});\ x + \alpha\vec{c} \in boundry(P) \& x \in \overline{r_i r_{i+1}}\},$$

where $\vec{c}$ is a unit vector perpendicular to $\overline{r_i r_{i+1}}$. It can be observed that such a minimum only happens on the extreme points of $\overline{r_i r_{i+1}}$ or points of P.

As the first step to facilitate the calculation, we rotate P in such a way that $\overline{r_i, r_{i+1}}$ becomes parallel to the x-axis. The rotation needs to be applied to the n vertices of P, which takes $O(n)$ time. On the next step we go through all edges e of P and categorize them by how their projection on the x-axis intersects the projection of $\overline{r_i r_{i+1}}$ on the x-axis. This can be done just by examining the x-coordinate of each vertex of e against the x-coordinate of transformed r_i and r_{i+1}. Since there are n edges, this operation takes $O(n)$ time. If the projection of e on the x-axis is included in the projection of $\overline{r_i r_{i+1}}$ then the minimum will happen on the extreme points of e hence its end vertices. If the projection of e on the x-axis has a non-empty intersection with the projection of $\overline{r_i r_{i+1}}$ then the minimum can happen at the end points of $\overline{r_i r_{i+1}}$ or the end points of e. To examine the intersection of the projection of e we examine the x-coordinates of the rotated edge against the end points of $\overline{r_i r_{i+1}}$ after the rotation is applied. The categorization of the edges can be done in constant time.

Based on the discussion above the whole process will take $O(ln)$ to complete.

■

3.2 Approximating a line segment if G_{Dir} is the complete graph

In this section we study the problem of approximating a piecewise linear path $R = (r_0 = s, \cdots, r_l = e)$ in polygon P, by a G_{Dir}-admissible path when G_{Dir} is the

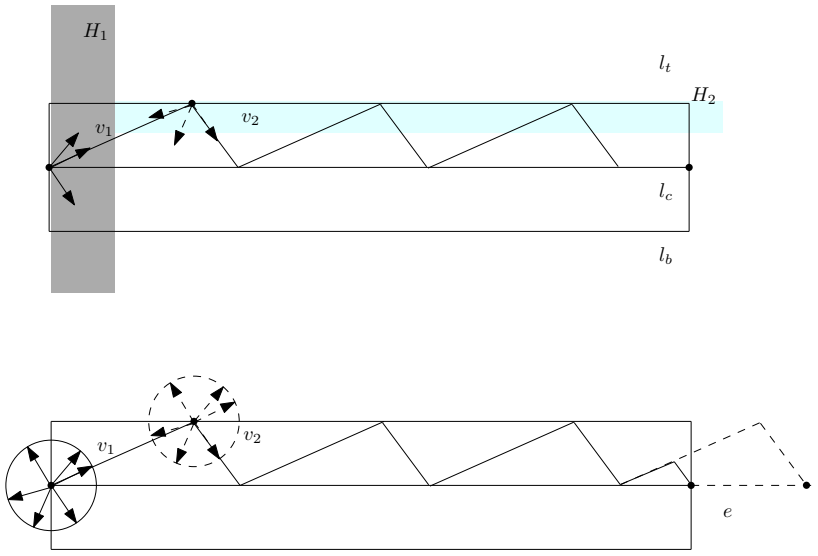


Figure 3.2.1: Sample run of Algorithm ZigZag

complete graph. We calculate a tubular neighborhood $Box(\overline{r_i, r_{i+1}}, \lambda_i)$ for each $\overline{r_i r_{i+1}}$ and approximate $\overline{r_i r_{i+1}}$ in $Box(\overline{r_i, r_{i+1}}, \lambda_i)$ by a G_{Dir}-admissible path, as illustrated in Figure 3.2.1.

Now let us assume that G_{Dir} is a complete graph and is spanning and symmetric. Assume the members of the set V of admissible directions are sorted by their slopes. We will denote the set of all rays out of a point x in directions in V, by V_x. Given a line segment $\overline{se}$ and a tubular neighborhood $Box(\overline{se}, \lambda)$, let l_t and l_b be the top and bottom lines supporting $Box(\overline{se}, \lambda)$ and l_C the centerline that contains $\overline{se}$.

Definition 19. Let v_1 and v_2 be two vectors in $\mathbb{R}^2$. We say that $v_1 \leq_\alpha v_2$ if and only if either v_2 is to the left of O and v_1 is to the right of O, or v_1 and v_2 are both on the same side of O and v_2 is to the left of v_1. Here, O is an arbitrary reference vector.

It can easily be observed that $v_1 \leq_\alpha v_2$ if and only if v_2 has a greater signed angle with O_v than v_1.

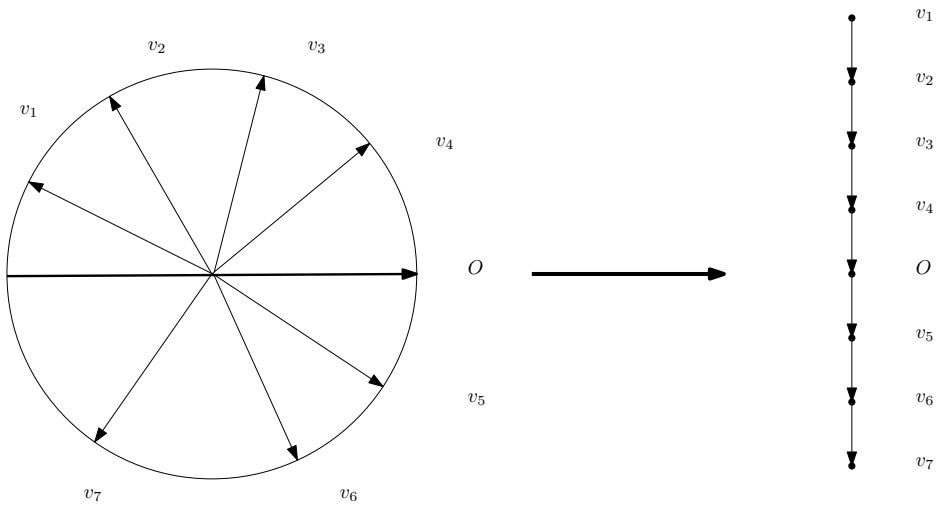


Figure 3.2.2: i) Illustration of or angular ordering imposed by left turn test

Let us assume that the direction of $\vec{se}$ is not in G_{Dir} and consider the half space $H^+ = \{x : c \cdot (s - x) \geq 0)$, where c is a normal vector of $\vec{se}$. This half space contains $\bar{se}$ and its boundary is perpendicular to $\bar{se}$. By Lemma 15, $V_s \cap H^+$ is not empty and has cardinality at least two, where V_s is the translation of vectors in V to point s. Let v_1 be the vector in V with minimum angular difference with $\bar{se}$. We assume that v_1 is to the left of $\vec{se} \subset l_c$, so any ray starting from l_c meets l_t. The result below is dually repeatable for the case that v_1 is to the right of $\vec{se}$.

As shown in Figure 3.2.1, we move from s along v_1 until we reach l_t at x_1. Let $L^+_{s\vec{x}_1}$ be the half plane on the right of $s\vec{x}_1$ which contains $\bar{se}$. $V_{x_1} \cap L^+_{s\vec{x}_1}$ has at least one element by Lemma 15. Let L^+_t be the half plane to the right of l_t which contains $Box(\bar{se}, \lambda)$. We show that $V_{x_1} \cap L^+_{s\vec{x}_1} \cap L^+_t$ also has at least one element. This easily follows from the fact that $L^+_t \cap L^+_{s\vec{x}_1}$ cannot contain any ray from V_{x_1} because v_1 has minimum angular difference with l_c and hence l_t.

We choose $v_2 \in V_{x_1} \cap L^+_{s\vec{x}_1} \cap L^+_t$ to have minimum angular difference with l_t. Now we continue from x_1 along with v_2 until we reach l_c again at x_2. The minimality in the selection of v_1 and v_2 guarantees that $x_2 - s = \beta(e - s)$ for some $\beta > 0$.

This process can be continued further by looking at the half space $L^-_{x_1\vec{x}_2}$ to the

left of $x_1\vec{x}_2$ and L_c^-, the half space to the left of $\vec{se}$ which contains v_1, and selecting the direction v_3 with minimal angular difference with l_c. The next lemma shows that $v_3 = v_1$.

Lemma 20. *Assuming v_1, v_2 and v_3 have been selected as above. Then $v_1 = v_3$.*

Proof: v_3 belongs to $L^-_{x_1\vec{x}_2} \cap L_c^- \cap V_{x_2}$ and is the right most member of it. Based on the assumption above about v_1, we have $v_1 \in H^+ \cap L_c^- \cap V_s \subset L^-_{x_1\vec{x}_2} \cap L_c^- \cap V_{x_2}$ but since every element of $L^-_{x_1\vec{x}_2} \cap L_c^- \cap V_{x_2}$ is to the left or equal to an element of $H^+ \cap L_c^- \cap V_s$ (i.e., greater or equal in term of angle)the minimum element will be equal, hence $v_1 = v_3$. ■

The following lemma will also help to simplify the process of calculating v_1 and v_2.

Lemma 21. *For a given pair of start and end points (s, e) if $\vec{se}$ does not match the direction of any of the vectors in $V_{G_{Dir}}$, v_1 and v_2 can be calculated by finding the placement of $\vec{se}$ in the ordered list $(V_{G_{Dir}}, \leq_\alpha)$. The placement of $\vec{se}$ in the list gives us two elements $v_{max} = \max\{v \in V_{dir};\ v \leq_\alpha \vec{se}\}$ and $v_{min} = \min\{v \in V_{G_{Dir}};\ \vec{se} \leq_\alpha v\}$. If the absolute value of the signed angle between v_{max} and $\vec{se}$ is greater than the angle between v_{min} and $\vec{se}$, then $v_1 = v_{min}$ and $v_2 = v_{max}$, and vise versa otherwise.*

Proof: The proof of this lemma is rather obvious based on how v_1 and v_2 are selected in the above process. Assume the absolute value of the angle between v_{max} and $\vec{se}$ is greater than angle between v_{min} and $\vec{se}$. Let v_1 be the closest element in $V_{G_{Dir}} \cap H^+ \cap L_c^+$ to $e - s$ angularly. v_{min} is in L_c^+ by definition, so $v_{min} \leq_\alpha v_1$. We also have $v_{min} \in H^+$ since otherwise $v_1 \leq_\alpha c \leq_\alpha v_{min}$ which contradicts the above result so $v_{min} = v_1$. Similarly it can be concluded that $v_2 = v_{max}$. For the case that the absolute value of the angle between v_{max} and $\vec{se}$ is less than the angle between v_{min} and $\vec{se}$, all the results will be dual of the above, i.e., $v_{max} = v_1$ and $v_{min} = v_2$. ■

Algorithm Zigzag is approximating a line segment with an G_{Dir}-admissible path. We can list the steps of it as follows:

1. Set $H := \{c \mid c.(e - s) \geq 0\}$.

2. Define v_1 to be the closest vector in $V_s \cap H$ to the direction of $\bar{se}$ according to the ordering $\leq_\alpha$.

3. Based on whether $v_1 \leq_\alpha \bar{se}$ or $\bar{se} \leq_\alpha v_1$, continue from s in the direction of v_1 until we meet l_b or l_t at $\vec{zig}$.

4. Calculate an inward normal vector n, for example for l_c, and set $H = \{c \mid c \cdot (n) \geq 0\}$. n is calculated in such a way that H always contains l_c.

5. Define v_2 to be the closest vector in $V_s \cap H$ to the direction of $\bar{se}$ according to the ordering $\leq_\alpha$.

6. Continue from zig in the direction of v_2 until we meet l_c in zag.

7. Continue alternating in the directions of v_1 and v_2 and zigzag between l_t (or l_b depending on v_1) and l_c until zag meets l_c past e.

8. Scale and adjust the last zig zag so it meets l_c at e.

Function Zigzag

Data: (s, e), λ, $V_{G_{Dir}}$ considered as a complete graph and sorted by slope
Result: A G_{Dir}-admissible path $R = (r_0 = s, \cdots, r_l = e)$ between s and e such that $R \in Box(\overline{s,e}, \lambda)$

$lastPoint \leftarrow s$;
$Zigzag \leftarrow (lastPoint)$;
$H \leftarrow \{x : (e-s)(x-s) \geq 0\}$;
$v_1 \leftarrow GetClosestDirection(V, \overline{es}, H)$;
if v_1 *to the left of* $(e-s)$ **then**
  $\vec{zig} \leftarrow GetIntersection(l_t, (v_1, s))$;
  $n_1 \leftarrow XYCoordinate((0,0,1) \times (v_1, 0))$;
  $H \leftarrow \{x : x - s.n_1 \geq 0\}$;
else
  $\vec{zig} \leftarrow GetIntersection(l_b, (v_1, s))$;
  $n_1 \leftarrow XYCoordinate((v_1, 0) \times (0,0,1))$;
  $H \leftarrow \{x : x - s.n_1 \geq 0\}$;
end
$v_2 \leftarrow GetClosestDirection(V, \overline{es}, H)$;
$\vec{zag} \leftarrow GetIntersection(l_c, (v_2, s))$;
`/* zigzagging along` $\overline{se}$ `*/`
while $lastPoint.x < vecse.x$ **do**
  $Zigzag.Append(lastPoint + \vec{zig})$;
  $Zigzag.Append(lastPoint + \vec{zig} + \vec{zag})$;
  $lastPoint \leftarrow lastPoint + \vec{zig} + \vec{zag}$;
end
if $lastPoint.x > vecse.x$ **then**
  $Zigzag \leftarrow AdjustPath(Zigzag, s, e)$;
end

The following lemma proves the correctness of Algorithm Zigzag.

Lemma 22. *If G_{Dir} is complete, spanning, and symmetric ,(s, e) is a pair of points and $Box(\bar{se}, \lambda)$ is a tubular neighborhood of $\overline{se}$ in $\mathbb{R}^2$ then there exists a G_{Dir}- Admissible path R between s and e which is completely inside $Box(\bar{se}, \lambda)$. Algorithm Zigzag computes such a path in $O(\log(m) + \|s-e\|/\lambda \cot(\angle v_{i+1} - v_i))$ time. The length of R satisfies $\|R\| \leq \|s-e\|(1 + 1/\cot(\angle v_{i+1} - v_i))$. Here, v_i and v_{i+1} are two consecutive vectors (with respect to $\leq_\alpha$) with maximum angular difference.*

Function GetClosestDirection

Data: $V_{G_{Dir}}$ considered as a complete graph and sorted by slope, a vector $\vec{d}$ representing the direction, a half space H

Result: A member of $V_{G_{Dir}}$ which is in H and has minimal angular difference with $\vec{d}$

$(v, v') \leftarrow$ binary searching V for placement of $\vec{d}$ in V;

`/* This can be achieved by doing a left turn test between` $\vec{d}$ `and the visiting node on the binary search tree */`

$\tilde{v} \leftarrow$ closest of (v, v') to $\vec{d}$;

if $\tilde{v}$ *in* H **then**

  return $\tilde{v}$

else

  return $-\tilde{v}$

end

Function GetIntersection

Data: $(\vec{d_1}, \vec{b_1})$ representing the first line, $(\vec{d_2}, \vec{b_2})$ representing the second line

Result: the intersection point of line 1 and line 2

$x_{\vec{d_1}} \leftarrow$ x-coordinate of $\vec{d_1}$;

$y_{\vec{d_1}} \leftarrow$ y-coordinate of $\vec{d_1}$;

$x_{\vec{d_2}} \leftarrow$ x-coordinate of $\vec{d_2}$;

$y_{\vec{d_2}} \leftarrow$ y-coordinate of $\vec{d_2}$;

$x_{\vec{b_1}} \leftarrow$ x-coordinate of $\vec{b_1}$;

$y_{\vec{b_1}} \leftarrow$ y-coordinate of $\vec{b_1}$;

$x_{\vec{b_2}} \leftarrow$ x-coordinate of $\vec{b_2}$;

$y_{\vec{b_2}} \leftarrow$ y-coordinate of $\vec{b_2}$;

$\alpha \leftarrow \frac{(y_{b_2} - y_{b_1})x_{d_1} - (x_{b_2} - x_{b_1})y_{d_1}}{x_{d_2}y_{d_1} - x_{d_1}y_{d_2}}$;

Return $\vec{b_2} + \alpha\vec{d_2}$;

Function AdjustPath

Data: R list of the points of the approximation, e, s

Result: Adjusted path R' which start from s and ends in e

`/* If the last leg of` R `passes` e `on` l_c `this function adjusts last leg to connect to` e `*/`

$l \leftarrow$ length of R ;

$R[l-1] \leftarrow GetIntersection((R[l-1] - R[l-2], R[l-2]), (R[l] - R[l-1], e))$;

$R[l] \leftarrow e$;

Return R;

Proof: It is obvious that the path generated by algorithm Zigzag is G_{Dir}-admissible. To start we show that the algorithm finishes. The only iteration in algorithm Zigzag happens in its sole while loop in which two points zig and zag are added to the approximating path $Zigzag$ in each iteration. Since zag is always on l_c by definition, in order to check if we have reached the end point e we only need to compare the x-coordinate of $lastPoint$ against the x-coordinate of e. This is because for each value of zag there is a constant α such that $zag = s + \alpha(e - s)$. The value of $lastPoint$ increases in each iteration to $lastPoint + zig + zag = lastPoint + \beta(e - s)/\|e - s\|$ for some $\beta > 0$. We have

$$\beta = \lambda\left(\frac{\cos(\angle v_1 - \angle \vec{se})}{\sin(\angle v_1 - \angle \vec{se})} - \frac{\cos(\angle v_2 - \angle \vec{se})}{\sin(\angle v_2 - \angle \vec{se})}\right) \tag{3.2.1}$$

i.e.,

$$\beta = \lambda\left(\frac{\sin((\angle v_1 - \angle \vec{se}) - (\angle v_2 - \angle \vec{se}))}{\sin(\angle v_1 - \angle \vec{se})\sin(\angle v_2 - \angle \vec{se})}\right).$$

As a result we have:

$$\beta = \lambda\left(\frac{\sin(\angle v_1 - \angle v_2)}{\sin(\angle v_1 - \angle \vec{se})\sin(\angle v_2 - \angle \vec{se})}\right).$$

Now we find out for a given v_1 and v_2, which angle of $\vec{se}$ minimize the value of β.To do so we examine the zeros of the derivative of β in terms of the angle of $\vec{se}$, $d\beta/d\angle \vec{se}$ is

$$-\frac{\sin(\angle v_1 - \angle v_2)(\sin(\angle v_1 - \angle \vec{se})\cos(\angle v_2 - \angle \vec{se}) + \sin(\angle v_1 - \angle \vec{se})\cos(\angle v_2 - \angle \vec{se}))}{(\sin(\angle v_1 - \angle \vec{se})\sin(\angle v_2 - \angle \vec{se}))^2},$$

which can be simplified to

$$-\frac{\sin(\angle v_1 - \angle v_2)\sin(\angle v_1 + \angle v_1 - 2\angle\vec{se})}{(\sin(\angle v_1 - \angle\vec{se})\sin(\angle v_2 - \angle\vec{se}))^2}.$$

Based on the above identity $\angle\vec{se} = (\angle v_1 + \angle v_1)/2$ is a zero of $d\beta/d\angle\vec{se}$. Since the denominator of $d\beta/d\angle\vec{se}$ is always positive and based on the sign of nominator, it can be concluded that $(\angle v_1 + \angle v_1)/2$ is a local minima.

By putting back the minimum value into (3.2.1) we have

$$\beta = 2\lambda(\frac{\cos(\angle v_1 - \angle v_2)/2)}{\sin((\angle v_1 - \angle v_2)/2)}) = 2\lambda\cot((\angle v_2 - \angle v_1)/2).$$

Since cot is decreasing, β acquires its minimum value when the difference between v_1 and v_2 is maximum.

Based on Lemma 21, v_1 and v_2 are always two consecutive vectors in $V_{G_{Dir}}$ ordered by $\leq_\alpha$. Let i be an index for which the value $v_{i+1} - v_i$ is maximum. Then we have $\beta \geq 2\lambda\cot(\angle v_{i+1} - v_i/2)$. So the while loop will add at least $2\lambda\cot(\angle v_{i+1} - v_i)$ to $lastPoint - s$.Therefore the while loop finishes at most after $\|s-e\|/2\lambda\cot(\angle v_{i+1} - v_i)$ iterations. The number of edges in *Zigzag* is at least twice the number of iterations of the while loop, i.e., $\|s - e\|/\lambda cot(\angle v_{i+1} - v_i)$.

It takes $O(\log m)$ to find v_1 and v_2 in the sorted list of directions $V_{G_{Dir}}$. Plus the time spent in the while loop this proves our claim about the complexity of algorithm Zigzag.

It also can be observed that for each edge e_i of *Zigzag* we have $\|e_i\| \leq \|f_i\| + \lambda$ where f_i is the orthogonal projection of e_i on $\overline{se}$. Therefore $\sum\|e_i\| \leq \sum\|f_i\| + l.\lambda$ where l is the number of edges in *Zigzag*. By substituting the value of l from the above we have $\sum\|e_i\| \leq \|s - e\|(1 + \tan(\angle v_{i+1} - v_i))$

■

3.3 Approximating a line segment if G_{Dir} is strongly connected

Now we draw our attention to the more general case where G_{Dir} is only strongly connected. If G_{Dir} is strongly connected then for any two directions v_i and v_j in $V_{G_{Dir}}$ there exists a path from v_i to v_j and one from v_j to v_i.

We define an ε-padding of $Box(\bar{se}, \lambda)$ to be the box around $Box(\bar{se}, \lambda)$ with edges at distance ε of the edges of $Box(\bar{se}, \lambda)$, as illustrated in Figure 3.3.1. More precisely,

$$Box_\varepsilon(\bar{se}, \lambda) = Box(\bar{s_\varepsilon e_\varepsilon}, \lambda + \varepsilon),$$

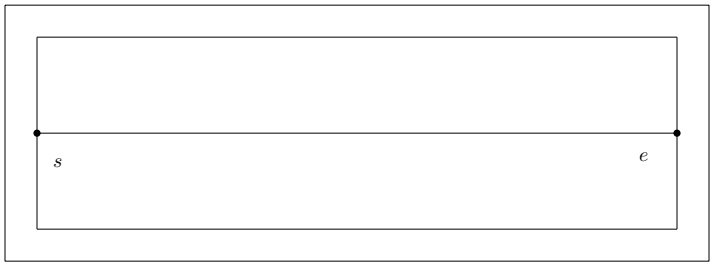
where $s_\varepsilon = s - \varepsilon(e - s/\|e - s\|)$ and $e_\varepsilon = e + \varepsilon(e - s/\|e - s\|)$. We won't need to calculate s_ε or e_ε in our algorithm picking a small enough ε will suffice.

Lemma 23. *Let $x \in B(s, \varepsilon)$, be a point inside the ball with center s and radius ε, v a vector, and l a line not parallel to v. Let l_v and l'_v be two lines tangent to $B(s, \varepsilon)$ and parallel to v. Then the line l_x passing through x parallel to v, meets l between l_v and l'_v.*

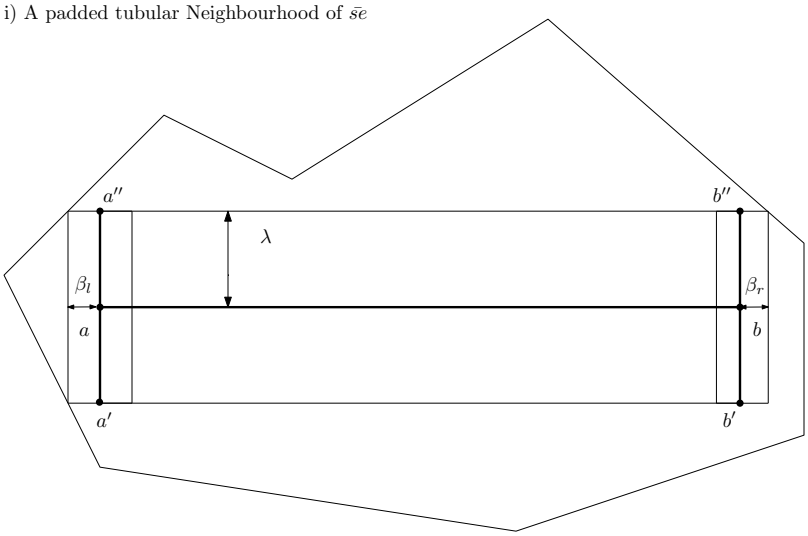
Proof: This results from the fact that l_v and l'_v and l_x are all parallel to each other. ■

First we show that if G_{Dir} is strongly connected, spanning and symmetric, then the weak version of admissible path planning is possible.

The new algorithm ZigzagEx has a template very similar to algorithm Zigzag. First we find v_1 and v_2 as in algorithm Zigzag. The only difference is that here we cannot switch from v_1 to v_2 and vise versa, instead we should go through the paths between v_1 and v_2. Based on Theorem 10, we can control how far away we wander from the intersection of each leg of zigzags and l_t (or l_b) or l_c. Given a path $(v_1, \cdots, v_n) \in G_{Dir}$ and a point c and a number β, function BuildPath is building a



i) A padded tubular Neighbourhood of $\bar{se}$



ii) Findling a tubularr neighbourhood

Figure 3.3.1: ε padding of $Box(\bar{se}, \lambda)$

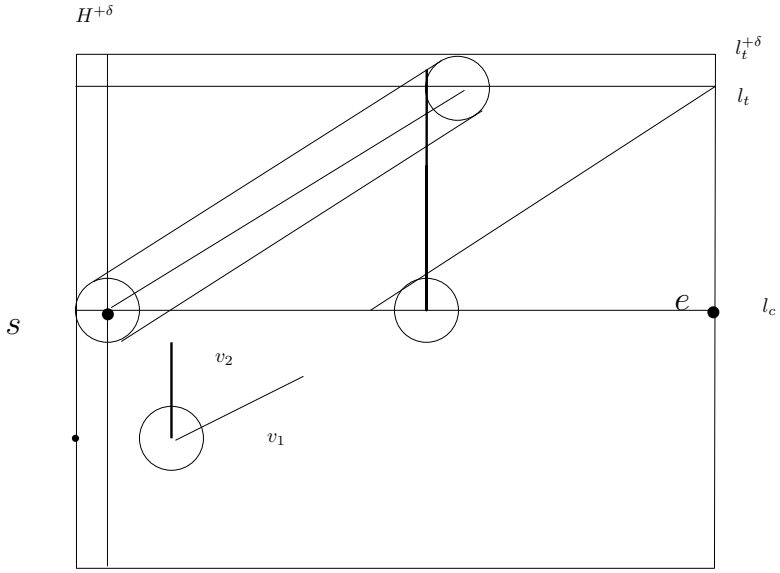


Figure 3.3.2: Approximating a line segment when G_{Dir} is strongly connected.

piecewise linear path $(c, c+\beta v_1 = c_1, \cdots, c_n+\beta v_n)$ which by Lemma 10 ends in a ball of radius βW around c, where W is the length of $(v_1, \cdots, v_n)$ with Euclidean metric. If $(v_1, \cdots, v_n)$ is the shortest path between v_1 and v_n then the path generated by BuildPath is in a ball with radius $\beta Diam(G_{Dir})$, where $Diam(G_{Dir})$ is the diameter of Graph G_{Dir}, i.e., the maximum length of all shortest paths in G_{Dir}. For a given ε, if $\beta = \varepsilon/Diam(G_{Dir})$, the path generated by BuildPath will always be in a ball of radius ε around c.

To prove that the method of algorithm Zigzag still works, we show that if ε is selected small enough, the section of the path generated by application of BuildPath does not wander off the padded tubular neighborhood and also Algorithm ZigzagEx still moves forward by a constant distance on l_c in each iteration. The first is achieved easily by selecting $\varepsilon < \delta$.

Figures 3.3.2 and 3.3.3 depict an iteration of Algorithm ZigzagEx. In each iteration we start from the last point of *Zigzag*, which we denote by s_i and is initialized by the starting point s. we apply *BuildPath* with $P_{v_2v_1}$ to generate a path $(s_i, \cdots, c_i^{v_1}, c_i^{v_1} + \beta v_1)$ in the ε-neighborhood of the starting point s_i. Then we start from $c_i^{v_1}$ in direction

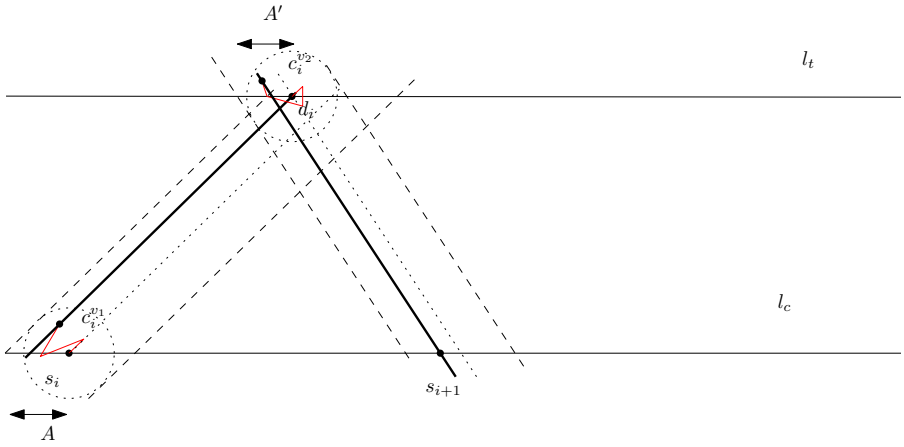


Figure 3.3.3: An iteration of the Crisscross function

of v_1 and continue until it intersect with l_t (or l_b depending on v_1) at d_i. Reapplying BuildPath with $P_{v_2v_1}$, generates a path $(d_i, \cdots, c_i^{v_2}, c_i^{v_2} + \beta v_2)$ in the ε-neighborhood of d_i. Then continue from $c_i^{v_2}$ in the direction of v_2 until it reaches l_c at s_{i+1}.

Now let us compare the above iteration with an iteration of Algorithm Zigzag. We start from s_i. Follow the direction of v_1 until we meet l_t at d_i' and continue from there in the direction v_2 until we intersect l_c in s_{i+1}'. By Lemma 22, $\|s_i - s_{i+1}'\|$ is always greater or equal to $\lambda \cot(Max_{dif}/2)$, where Max_{dif} is the maximum angular difference between two consecutive directions in $V_{G_{Dir}}$ ordered by $\leq_\alpha$. Let z_1 and z_2 be the two lines representing the envelope of the boundary of the ε-disks around $s_i + \beta v_1$ for any β. z_1 and z_2 are tangent to all ε-disks around $s_i + \beta v_1$ and hence parallel to the vector $d_i' - s_i$. Since the vector $d_i - c_i^{v_1}$ is also parallel to $d_i' - s_i$ then by Lemma 23 d_i is located between $m_1 = z_1 \cap l_t$ and $z_2 \cap l_t$. As a result we have $\|d_i - d_i'\| < A_{v_1} = \|m_1 - d_i'\|$ (see Figure 3.3.5). If f_{d_i} is the intersection of $d_i + \beta v_2$ with l_c, parallelity gives us $\|s_{i+1}' - f_{d_i}\| < A_{v_1}$. Based on the same argument we can also show that $\|s_{i+1} - f_{d_i}\| < A_{v_2}$, and again Lemma 23 gives us $\|s_{i+1} - s_{i+}'\| < A_{v_1} + A_{v_2}$. From above we have $\|s_i - s_{i+1}\| > \|s_i - s_{i+1}'\| - A_{v_1} - A_{v_2}$ and hence $\|s_i - s_{i+1}\| > 2\lambda \cot(Max_{dif}/2) - A_{v_1} - A_{v_2}$.

Let us focus on v_1 as it is depicted in figure 3.3.4. Let $A = A_{v_1}$. From the triangular

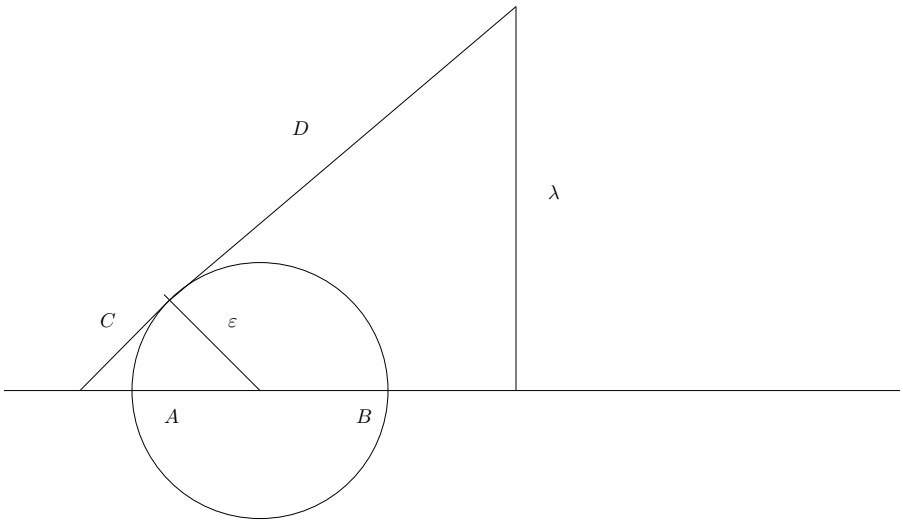


Figure 3.3.4: $\varepsilon/(\lambda) = A/(C+D) = C/(A+B)$

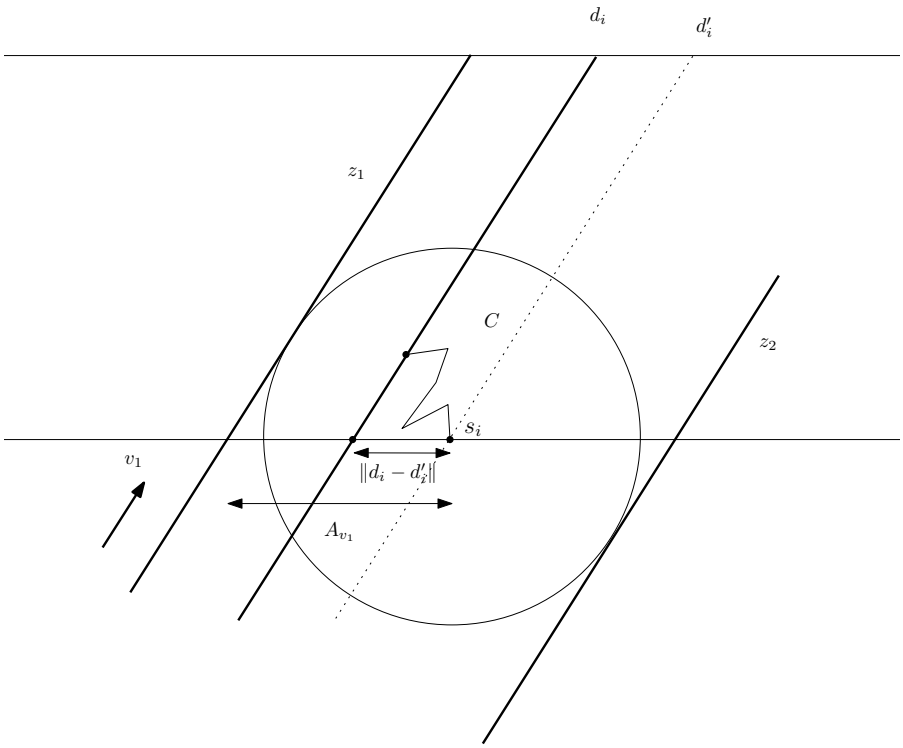


Figure 3.3.5: An ε deviation from from starting point s

similarities in Figure 3.3.4 we have $\varepsilon/\lambda = C/(A+B)$ hence $C = \varepsilon(A+B)/\lambda$. If we calculate A in term of C and ε we have $A^2 = C^2+\varepsilon^2$ so $A^2 = (\varepsilon(A+B)/\lambda)^2+\varepsilon^2$, from which we have $A = \sqrt{(\varepsilon(A+B)/\lambda)^2+\varepsilon^2}$ or $A = \varepsilon/\lambda\sqrt{((A+B))^2+\lambda)^2}$. Since both $(A+B)$ and λ are greater than zero we have $(A+B)^2+\lambda^2 \leq (A+B)^2+\lambda^2+2(A+B)\lambda$ and hence $A \leq \varepsilon/\lambda((A+B)+\lambda)$. Unless the algorithm finishes in one iteration, we can assume that $A+B \leq \|s-e\|$ and hence $A \leq \varepsilon((\|s-e\|/\lambda)+1)$. This inequality is independent of the selection of v_1 and applies to v_2 as well. To avoid the complexity of calculating $\|s-e\|$, $(e-s)\cdot(e-s) = \|s-e\|^2$ can be used as well. Assume that ε is chosen such that

$$K := \lambda \cot((\angle v_1 - \angle v_2)/2) - \varepsilon(\|s-e\|/\lambda + 1) > 0 \tag{3.3.1}$$

Now let us assume algorithm ZigzagEx finishes in more than 1 iterations (otherwise it is a trivial case) and ε is selected small enough such that (3.3.1) holds. Then in each iteration algorithm ZigzagEx moves forward along $\bar{es}$ at least as far as K. This ensures the termination of the algorithm in at most $\|s-e\|/K$ iterations.

The next difference between algorithm Zigzag and ZigzagEx is the path adjustment step. In algorithm Zigzag, the adjustment step is only adjusting the last leg of the approximated path but in this case it is more complex because more than two edges are involved when e is in the ε-neighborhood of $Zigzag.LastPoint$. To overcome that adjustment step, the algorithm scales down the computed path $Zigzag$ by a factor of $k = e.x/Zigzag.LastPoint().x$ as formulated in function AdjustPath.

Lemma 24. *Let G_{Dir} be strongly connected, spanning, and symmetric, (s,e) a pair of points and $Box_\delta(\bar{se},\lambda)$ a δ-padding of the tubular neighborhood of $\bar{se}$ in $\mathbb{R}^2$. If ε is selected small enough then algorithm ZigzagEx computes a G_{Dir}-admissible path between s and e which resides inside $Box_\delta(\bar{se},\lambda)$.*

It can be observed that if ε is selected in such a way that inequality (3.3.1)

Function ZigzagEx

Data: (s, e), λ, G_{Dir} strongly connected, $V_{G_{Dir}}$ sorted by slope, ε, δ
Result: A G_{Dir}-admissible path $R = (r_0 = s, \cdots, r_l = e)$ between s and e such that $R \in Box_{\varepsilon}(\overline{s,e}, \lambda)$

$lastPoint \leftarrow s$;
$Zigzag.Append(lastPoint)$;
$H \leftarrow HSbyNormal(s - e)$;
/* find v_1 and v_2 */
$v_1 \leftarrow GetClosestDirection(V, \overline{es}, H)$;
if v_1 *to the left of* $(e - s)$ **then**
  $n \leftarrow XYCoordinate((0, 0, 1) \times (v_1, 0))$;
  $H \leftarrow HSbyNormal(n)$;
else
  $n \leftarrow XYCoordinate((v_1, 0) \times (0, 0, 1))$;
  $H \leftarrow HSbyNormal(n)$;
end
$v_2 \leftarrow GetClosestDirection(V, \overline{es}, H)$;
/* Calculate shortest path from v_1 to v_2 and v_2 to v_1 */
$P_1 \leftarrow GetShortestPath(G_{Dir}, v_1, v_2)$;
$P_2 \leftarrow GetShortestPath(G_{Dir}, v_2, v_1)$;
$lastPoint \leftarrow ZigZag.lastPoint$;
while $ZigZag.lastPoint.x < \vec{se}.x$ **do**
  $ZigZag \leftarrow CRISSCROSS(ZigZag, l_t, l_c, \varepsilon, P_1, P_2)$;
end
$ZigZag \leftarrow BuildPath(ZigZag, P_e - \{v_2\}, \varepsilon)$;
if $ZigZag.lastPoint.x > \vec{se}.x$ **then**
  $Zigzag \leftarrow AdjustPath(Zigzag, s, e)$;
end

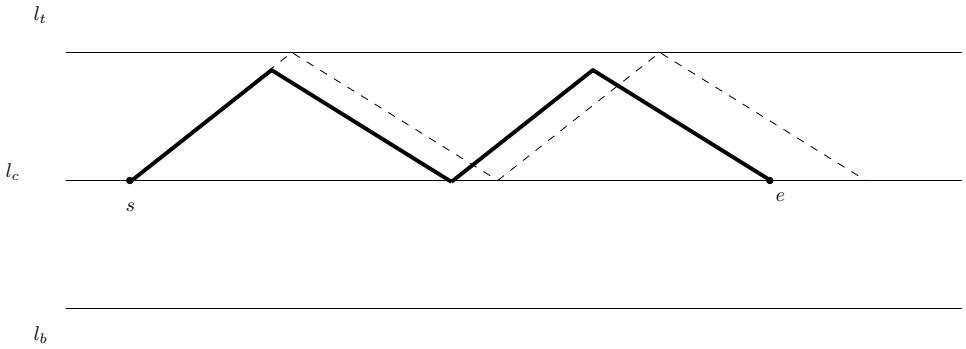


Figure 3.3.6: Dotted path generated by algorithm ZigzagEx before adjustment. Heavier path illustrated the same path after adjustment.

Function CRISSCROSS

Data: s, l_t, l_b, l_c, piecewise linear path $Zigzag$, v_1, v_2, P_1, P_2
Result: A G_{Dir}-admissible path $Zigzag$ that runs between l_t (or l_b) and l_c
if $Zigzag.LastPoint \neq s$ **then**
  `/* Generated path is always in a ball of radius` ε `*/`
  $Zigzag.Append(BuildPath(Zigzag, P_1, \varepsilon).RemoveLastPoint())$;
end
if v_1 *to the left of* $(e - s)$ **then**
  $\vec{zig} \leftarrow GetIntersection(l_t, (v_1, lastPoint))$;
else
  $\vec{zig} \leftarrow GetIntersection(l_b, (v_1, lastPoint)$;
end
$Zigzag.Append(BuildPath(Zigzag, P_2, \varepsilon).RemoveLastPoint())$;
$\vec{zag} \leftarrow GetIntersection(l_c, (v_2, Zigzag.lastPoint))$;
$Zigzag.Append(zag)$;

Function BuildPath

Data: Center c, path $(v_1, \cdots, v_n)$ in G_{dir}, and a scaling factor β.
Result: A piecewise linear G_{Dir}-admissible path Π based on $(v_1, \cdots, v_n)$ and starting from c.
Add c to Π;
$p \leftarrow c$;
for $1 \leq i \leq n$ **do**
  $p \leftarrow p + \beta \cdot v_i$;
  Add p to Π;
end

Function BuildPathRevers

Data: Center c, path $(v_1, \cdots, v_n)$ in G_{dir}, and a scaling factor β.
Result: A piecewise linear G_{Dir}-admissible path Π based on $(v_1, \cdots, v_n)$ and starting from c.
$P \leftarrow$ Revers of $(-v_1, \cdots, -v_n)$;
Call BuildPath with c, P and β;

Function AdjustPath

Data: R list of the points of the approximation, e, s
Result: Adjusted path R' which starts from s and ends in e
`/* If the last leg of` R `passes` e `on` l_c `this function adjusts the path to connect to` e `*/`
$l \leftarrow$ length of R ;
$K \leftarrow e.x/R[l].x$;
$AdjustedPointsAppend(R[1])$;
for $i = 2$ *to* l **do**
  $p \leftarrow R[i] - R[i-1]$;
  $p.x \leftarrow Kp.x$;
  $p.y \leftarrow Kp.y$;
  $AdjustedPoints.Append(AdjustedPoints[i] + p)$;
end
Return $AdjustedPoints$;

holds then the same ε will also work for any (e', s') with $\|e' - s'\| \leq \|e - s\|$. We can use this property to show that the strong form of the problem is also solvable. Let (v_s, v_e) be two given start and end directions and let $P_{v_s v_1}$ and $P_{v_2 v_e}$ be the shortest paths from v_s to v_1 and v_2 to v_e respectively. Starting from e we run function BuildPathRevers to get the piecewise linear path $(c_{v_2}, \cdots, e)$ based on $P_{v_2 v_e}$. This is done by traversing $P_{v_2 v_e}$ in reverse order and building the sequence of points $p[i] = p[i+1] - \beta/Diam(G_{Dir})P_{v_2 v_e}[i+1]$ where $p[l] = e$ and l is the length of $P_{v_2 v_e}$. This will put $p[1] = c_{v_2}$ in the β-neighborhood of e.

Next we use BuildPath to build a path for $P_{v_s v_1}$ starting from s in the β-neighborhood of s. Let q be the second last point of this path which means we can move forward in the direction of v_1 from this point. Starting from q, we will follow v_1 until we reach l_t or l_c. If we reach l_c first we choose the intersection point as the new starting point s', otherwise from the intersection point with l_t, we follow the direction of v_2 back to l_c until we meet l_c at the new starting point s'.

Similarly for $(c_{v_2}, \cdots, e)$, starting from c_{v_2} we follow the direction of $-v_2$ until we reach either l_c or l_t. If we reach l_c first and the intersection point is between s and

e we will choose the new end point e' to be this intersection point. Otherwise from the intersection point with l_t we follow $-v_1$ until we reach l_c and we will choose the intersection as the new end point e'. Now if β is less than ε we can apply algorithm ZigzagEx to find a G_{Dir}-admissible path between s' and e' and hence between s and e, starting in direction v_s and ending in direction v_e.

Lemma 25. *Let G_{Dir} be strongly connected, spanning, and symmetric, (s, e) a pair of points, (v_s, v_e) a pair of start and end directions and $Box_\delta(\bar{se}, \lambda)$ a δ-padding of the tubular neighborhood of $\bar{se}$ in $\mathbb{R}^2$. If ε is selected small enough then Algorithm ZigzagEx can be adjusted to construct a G_{Dir}-admissible path between s and e which is inside $Box_\delta(\bar{se}, \lambda)$.*

3.4 Approximating a path with a G_{Dir}-admissible path

We conclude this chapter by showing that if P is a polygonal environment, G_{Dir} is a strongly connected direction constraint graph and R is a piecewise linear path between start and end points (s, e) then R can be approximated by a G_{Dir}- admissible path. We first need to adjust Algorithm 5 to generate a padded tubular neighborhood. Function PaddedTubularNeighborhood accepts the polynomial environment P, a line segment (r_i, r_{i+1}) and a padding factor $1/2^k$ and calculates a padded tubular neighborhood of (r_i, r_{i+1}) which has a padding less than or equal to the width of the neighborhood scaled by the padding factor.

In this version of the algorithm we will first calculate an orthogonal frame over $r_i r_{i+1}$ by calculating $(c_i, 0) = (r_{i+1} - r_i, 0) \times (0, 0, 1)$, where $(r_{i+1} - r_i, 0)$ is $r_{i+1} - r_i$ considered as a 3-dimensional vector and based on that we calculate T which maps $r_{i+1} - r_i$ to $(1, 0)$ and $c_i = (r_{i+1} - r_i, 0) \times (0, 0, 1)$ to $(1, 0)$, when r_i considered as the

origin. We should note that

$$\|(r_{i+1} - r_i, 0) \times (0, 0, 1)\| = \|r_{i+1} - r_i\| \sin(\pi/2) = \|r_{i+1} - r_i\|.$$

As a result T can be written as $\|r_{i+1} - r_i\|IU$ where U is a unitary rotation and I the identity. Since both U and I are isometric, this means that if $(x, y) = T(p)$ then $x = d(r_i, \Pi_{r_i r_{i+1}}(p))/\|r_{i+1} - r_i\|$ and $y = d(p, r_i r_{i+1})/\|r_{i+1} - r_i\|$ where $\Pi_{r_i r_{i+1}}(p)$ is the orthogonal projection of p on $r_i r_{i+1}$.

After applying the transformation $T = (c_i' \; c_i)^{-1}x - r_i$ to P, which involves translating the origin to r_i and applying linear transformation $(c_i' \; c_i)^{-1}x$, we find the maximum width λ and a box $B_{\lambda'}(r_i r_{i+1})$ containing $r_i r_{i+1}$ exactly in the same manner as we did in Algorithm 5 where $\lambda' = \lambda(1 - 2^{-k})$. To calculate a padding we rotate P by 90 degrees and apply function TubularNeighborhoodForUnit to the upper and lower bounding line segments of $B_{\lambda'}(r_i r_{i+1})$ as illustrated in Figure 3.3.1.

Function PaddedTubularNeighborhood

Data: A polygon $P \in \mathbb{R}^2$, a line segment (r_i, r_{i+1}) inside P, padding factor of the from $1/2^k$

Result: A padded tubular neighborhood of $\bar{01}$ with its closure inside P distinguished by $R = (p_{hPadding}, p_{vPadding}, padding, \lambda)$

`/* calculating an orthogonal frame along` $r_i r_{i+1}$ `*/`

$c'_i \leftarrow (r_{i+1} - r_i)$;

$c_i \leftarrow$ normal vector to c'_i ;

`/* finding a rotation that maps` c'_i `to` $(1,0)$ `(i.e.` $r_i \bar{r_{i+1}}$ `to x-axis) and` c_i `to` $(0,1)$ `*/`

$T \leftarrow (c'_i\ c_i)^{-1}$;

$start \leftarrow (0,0)$;

$end \leftarrow\ T(r_{i+1} - r_i) = (0,1)$;

$P_T \leftarrow$ apply transformation T to P with r_i as origin ;

$\lambda \leftarrow TubularNeighborhoodForUnit(P_T, start, end)$;

$P' \leftarrow$ apply a $90'$ rotation to P_T ;

$l \leftarrow TubularNeighborhoodForUnit(P', (-\lambda(1 - \frac{1}{2^k}), 0), (\lambda(1 - \frac{1}{2^k}), 0))$;

$\hat{P} \leftarrow$ translate P' by $(0, -1)$;

$r \leftarrow TubularNeighborhoodForUnit(\hat{P}, (-\lambda(1 - \frac{1}{2^k}), 0), (\lambda(1 - \frac{1}{2^k}), 0))$;

$sidePadding \leftarrow Min\{l, r\}$;

$padding \leftarrow Min\{sidePadding, \lambda/2^k\}$;

$$paddedLeftTop \leftarrow \begin{bmatrix} c'_i & c_i \end{bmatrix} \begin{bmatrix} 0 \\ \lambda \end{bmatrix};$$

$$leftTopCorner \leftarrow \begin{bmatrix} c'_i & c_i \end{bmatrix} \begin{bmatrix} 0 \\ \lambda(1 - 1/2^k) \end{bmatrix};$$

$$rightpPad \leftarrow \begin{bmatrix} c'_i & c_i \end{bmatrix} \begin{bmatrix} padding \\ 0 \end{bmatrix};$$

Function TubularNeighborhoodForUnit

Data: A polygon $P_T \in \mathbb{R}^2$, a line segment $R = (start, end)$ inside P_T and on x-axis

Result: A padded tubular neighborhood of $(0, 1)$ with its closure inside P_T

$min \leftarrow \infty$;

`/* find the closest points to the segment */`

1 **for** *each edge* $e = (p_T, p'_T) \in P_T$ **do**

  `/* if` $[p_T.x, p'_T.x] \subset [start.x, end.x]$ `*/`

  if $start.x \leq p_T.x \leq end.x$ *and* $start.x \leq p'_T.x \leq end.x$ **then**

    if $min \geq \mid p_T.y \mid$ **then** $min \leftarrow \mid p_T.y - y \mid$;

    if $min \geq \mid p'_T.y \mid$ **then** $min \leftarrow \mid p'_T.y - y \mid$;

  end

  if $[p_T.x, p'_T.x] \not\subset [start.x, end.x]$ *and* $[p_T.x, p'_T.x] \cap [start.x, end.x] \neq \emptyset$ **then**

    if $p.x < start.x$ **then**

      $y_{start}(e) \leftarrow e(start.x)$;

      if $min \geq y_{start}$ **then** $min \leftarrow y_{start}$;

    end

    if $p'.x > end.x$ **then**

      $y_{end}(e) \leftarrow e(end.x)$;

      if $min \geq y_{end}$ **then** $min \leftarrow y_{end}$;

    end

  end

end

$Return(min)$;

Theorem 26. *Let P be a polygonal environment (possibly with holes), G_{Dir} strongly connected, spanning, and symmetric, (s, e) a pair of points in P and $R = (s = r_1, \cdots, r_l = e)$ a collision free piecewise linear path in P between s and e. Then for sufficiently small ε, Algorithm 6 computes a G_{Dir}-admissible path between s and e which is inside P.*

Proof: Let Δ_{Max} be the maximum angular difference between two consecutive direction v_j and v_{j+1} in $V(G_{Dir})$ ordered by $\leq_\alpha$. To calculate a G_{Dir}-admissible approximation of R we will perform the following steps for each line segment $r_i r_{i+1}$ of R.

Algorithm 6.

1. *First we pick a k to be used as a padding factor. We can easily choose $k = 1$ for a padding factor of $1/2$. Larger k also can be selected to enforce some grade of accuracy for ε.*

2. *We find a padded tubular neighborhood by running PaddedTubularNeighborhood on input P, 2^{-k} and $r_i r_{i+1}$, and get $Box_\delta(r_i r_{i+1}, \lambda)$ with padding $\delta \leq 1/2^k \lambda$.*

3. *Choose $\varepsilon < \min\{\delta, \cot(\Delta_{Max}/2)\frac{\lambda^2}{1+\lambda}\|r_i - r_{i+1}\|\}$ and $\beta = \varepsilon / Diam(G_{Dir})$. Since $d_{max}(x, y) \leq \|x - y\|$ for any (x, y) we can use d_{max} here to have computationally feasible option. We will discuss this step in more detail.*

4. *Calculate an approximation of $r_i r_{i+1}$ in $Box_\delta(r_i r_{i+1}, \lambda)$ using ε, β, (r_i, r_{i+1}) as start and end points and $(v_s(i), v_e(i))$ as start and end directions. If we are processing $r_1 = s$ we set $v_s(1) = v_s$ as start direction otherwise $v_s(i) = v_e(i-1)$. If we are processing the last leg of R, $r_{l-1} r_l$, we will set $v_e(l) = v_e$ otherwise $v_e(i) = v_2(i-1)$.*

To be more clear on Step 3 it should be noted that the transformation used by PaddedTubularNeighborhood is transforming the orthogonal frame $\{c' = r_{i+1} - r_i, c\}$ to $(\{(1, 0), (0, 1)\})$. As a result the values acquired by this algorithm for λ and δ are scaled by $\|r_{i+1} - r_i\|$. In other word the distance between r_i and the left top corner of the tubular neighborhood λ' is equal to $\lambda\|r_i - r_{i+1}\|$. Based on Lemma 24, ε should be selected in a way that $\varepsilon < \delta$ and $\lambda' \cot(\Delta_{Max}/2) - \varepsilon(\|r_{i+1} - r_i\|/\lambda' + 1) > 0$. By replacing λ' (the width of the tubular neighborhood) with $\lambda\|r_i - r_{i+1}\|$, the inequality in (3.3.1) converts to $\varepsilon < \cot(\Delta_{Max}/2)\frac{\lambda^2}{1+\lambda}\|r_i - r_{i+1}\|$ and since $d_{max}(x, y) \leq \|x - y\|$ any

$$\varepsilon < \cot(\Delta_{Max}/2)\frac{\lambda^2}{1+\lambda} d_{Max}(r_i, r_{i+1})$$

will also work. We have $\cot(\Delta_{Max}/2) = \frac{1+\cos(\Delta_{Max})}{\sin(\Delta_{Max})}$. Since $v_j.v_{j+1} = \|v_j\|\|v_{j+1}\|\cos(\Delta_{Max})$ and

$$\mathbf{c} = (0,0,1).v_j \times v_{j+1} = \|v_j\|\|v_{j+1}\|\sin(\Delta_{Max}),$$

we have

$$\kappa := \frac{\|v_j\|_{Max}\|v_{j+1}\|_{Max} + v_j.v_{j+1}}{\mathbf{c}} \leq \cot(\Delta_{Max}/2).$$

As a result condition (3.3.1) can be further simplified to

$$\varepsilon < \kappa\frac{\lambda^2}{1+\lambda}d_{Max}(r_i, r_{i+1}).$$

Now we choose a k' such that $\varepsilon = d_{max}(r_{i+1}, r_i)/2^{k'}$ satisfies

$$d_{max}(r_{i+1}, r_i)/2^{k'} < \kappa\frac{\lambda^2}{1+\lambda}d_{Max}(r_i, r_{i+1})$$

or in other words $1/2^{k'} < \kappa\frac{\lambda^2}{1+\lambda}$. The integer k' can be selected so large that the number of iterations in Step 3 stays the same as Algorithm Zigzag.

After selecting a suitable ε we will choose $\beta < \varepsilon/Diameter(G_{Dir})$. The diameter of G_{Dir} can be calculated using any all pairs shortest path algorithm but any upper bound such as the sum of the lengths of all edges in G_{Dir} will also work. The above discussion shows that Step 3 can be done in constant time.

Step 1 is also done in constant time. Step 2 can be done in $O(n)$ for each linear piece of R. Step 4 can be computed in at most $Dimeter(G_{Dir}).\|s-e\|/\lambda cot(\angle v_{i_0+1} - v_{i_0})$ iterations. As a result the whole process can be done in

$$O(l(n + Dimeter(G_{Dir}).\|r_i - r_{i+1}\|/\lambda cot(\angle v_{i_0+1} - v_{i_0})))$$

time.

In this section, we have shown that for any polygonal environment P (possibly with holes) and any symmetric, spanning, and strongly connected direction graph G_{Dir}, any piecewise linear map between two points (s, e) can be converted to a G_{Dir}-admissible path. Since a piecewise linear map between two points in a polygonal environment can efficiently be found using classical methods like the one in [15], we can conclude that Problem 8 can be solved in two dimension.

If a pair of points (s, e) is given in a polygonal environment P, then a piecewise linear map between (s, e) can be found by traversing the graph built by connecting the center of the adjacent cells of the cylindrical algebraic decomposition of P as outlined in [15]. The environment can be preprocessed in $O(n \log(n))$ expected time using the trapezoid method and then for any pair of query points (s, e) a piecewise linear path R can be reported in $O(n)$ time where n is the size of P. Using the method in Theorem 26 this path can be approximated by a G_{Dir}-admissible path.

Chapter 4

Future Works

4.1 Approximation in higher dimensions

We studied the problem of approximating an existing piecewise linear path between two points in a polygonal environment in 2 dimensions but most of the practical control problems involve more than 2 dimensions. This makes any generalization of the materials discussed in this thesis to higher dimensions very important. Unfortunately, calculating an obstacle avoiding piecewise linear path between two points is not very efficient in higher dimensions. This is because a cylindrical algebraic decomposition of a polytope may result in many cells. As shown in [6] and [17] the cylindrical algebraic decomposition of a d dimensional polytope of size n can generate $O(n^{2^d-1})$ cells. This number can be improved to $O(n^{2d-1})$ using techniques in [18].

Before we go further let us exactly define what do we mean by polyhedral environment:

Definition 27. Let $A \subset \mathbb{Z}[x_1, \cdots, x_d]$ and $S = \{s_1, \cdots, s_l\}$ a decomposition of $\mathbb{R}^d$. We say S is A sign invariant (A-invariant for brevity) if for each $f \in A$, the sign of $f(x)$ does not change for $x \in s_i$. S is called algebraic if each cell is a semi-algebraic set

Definition 28. A set $S \subseteq \mathbb{R}^d$ is called semi-algebraic if it satisfies one of the following properties:

1. S can be constructed by finite union , intersection, and completion of sets

$$\{x \in \mathbb{R}^d, F(x) \geq 0\}$$

   where $F(x) \in \mathbb{Z}[x_1, \cdots, x_d]$ the ring of polynomial in d variable with integer coefficients.

2. is a definable set with respect to theory of real closed fields with a quantifier free formula

A quantifier free formula is well formed formula of the theory of real closed fields which does not have any existential or general quantifier and a well formed formula is defined recursively by the following rules:

1. The formulas $a > b$, $a = b$ and $a < b$, for polynomials a and b in $\mathbb{Z}[x_1, \cdots, x_d]$ are well-formed.

2. if $\varphi(x)$ is a well-formed formula with free variable x then so are $(\exists x;\ \varphi(x))$ and $(\forall x;\ \varphi(x))$.

3. if φ and ϕ are well-formed formulas then so are $\varphi \wedge \phi$, $\varphi \vee \phi$ and $\neg\varphi$

A set D in $\mathbb{R}^d$ is definable if there exist a well-formed formula φ such that $D = \{x \in \mathbb{R}^d;\ \varphi(x)\, is\, true\}$ since by the quantifier elimination theorem we can eliminate quantifier for each well formed formula of the theory of real closed fields, the two properties in Definition 28 are equivalent.

Definition 29. A d dimensional polyhedral environment is a semi-algebraic subset of $\mathbb{R}^d$ which is entirely defined by linear formulas (i.e., polynomials of degree 1).

Most of the results discussed in previous chapters are based on G_{Dir} being spanning and symmetric so the first step for generalizing our result to higher dimensions should involve a generalization of these properties to higher dimensions. Symmetricity of G_{Dir} can be naturally generalized to higher dimensions as it does not assume any dependency on the dimension of the space. The generalization of the property of being spanning is not straightforward. Let us assume G_{Dir} is a complete graph. We call $V(G_{Dir}) \subset T_{\vec{0}}(\mathbb{R}^d)$ spanning if for any set of directions $\{v_1 \cdots v_{i-1}\}$ for any $i \leq d$ there exist a direction v_i such that $\{v_1 \cdots v_i\}$ is linearly independent. It is easy to observe that for $d = 2$ this is equivalent to Definition 3. Now for a generic directional constraint graph G_{Dir} we have:

Definition 30. Let G_{Dir} be a direction constraint digraph and $V(G_{Dir}) \subset T_{\vec{0}}(\mathbb{R}^d)$ its direction set. G_{Dir} is called spanning if for any path $(v_1 \cdots v_{i-1})$ of length $i-1$ for any $i \leq d$, there exists a v_i in $\delta^+(v_{i-1})$ such that $\{v_1 \cdots v_i\}$ is linearly independent.

Based on this definition for being spanning, the higher dimensional version of Theorem 26 can be stated as follows:

Theorem 31. *Let $P \subset \mathbb{R}^d$ be a polyhedral environment, G_{Dir} strongly connected, spanning and symmetric, (s, e) a pair of points in P and $R = (s = r_1, \cdots, r_l = e)$ is a collision free piecewise linear path between s and e. Then for sufficiently small ε , there exists a G_{Dir}-admissible path between s and e which is inside P.*

Let se be a line segment in $\mathbb{R}^d$ and $E = \{e_1, \cdots, e_d\}$ be an orthonormal base for $\mathbb{R}^d$ such that $se \cdot e_1 = 0$, i.e., e_1 is in direction of se. We define the tubular neighborhood along se and compatible with E as

$$Box(E, \lambda, se)\{x;\ x = s + \alpha(e - s) + \Sigma_{i=2}^{n} \beta_i e_i, \alpha \leq 1, \beta_i < \lambda\}.$$

Now let P be a polytope in $\mathbb{R}^n$ and se a line segment inside P. Then for each selection

of E, there exists a λ_E such that $Box(E, \lambda, se) \subset P$.

The same techniques used in previous chapters seem to be generalizable to higher dimension, If the above definition for tubular neighborhood is used. The behavior and complexity of this generalization is yet to be studied.

4.2 Direct calculation of G_{Dir}-admissible path and minimum turn

We have mainly focused on approximating an existing piecewise linear path between two points by a G_{Dir}-admissible path. We also provided an algorithm to calculate a G_{Dir}-admissible path directly in a triangulated polygonal environment when G_{Dir} is a complete graph over the set $\{(1,0),(-1,0),(0,1),(0,-1)\}$. The same technique seems to be generalizable to higher dimensions for any polyhedral environment $P \subset \mathbb{R}^d$ and a simplical complex of P, when G_{Dir} is the complete graph over the set of standard orthonormal bases of $\mathbb{R}^d$ plus their negatives. The exact behavior and complexity of this technique in higher dimensions is unknown.

We observed that the length of the G_{Dir}-admissible path generated by Algorithm 3 is related to how close the query points are to the vertices of the triangle. On contrary to classical versions of path planning problems this prevents us from giving a meaningful bound on the length of the G_{Dir}- admissible path generated by Algorithm 4 in terms of the size of the environment. However the length of the G_{Dir}-admissible paths between the midpoint of edges of a triangle T_i in triangulation of P seems to be either one or two in most of the cases. This brings up the question of the expected length of a G_{Dir} admissible path generated by Algorithm 4.

The consequence of relaxing the constraints on G_{Dir} to be only strongly connected, symmetric and spanning on Algorithm 4 is also the subject of future studies and not

well understood.

Another interesting problem in this context is the problem of finding a G_{Dir}-admissible path between (s, e) in such a way that the number of changes in directions are minimum. In this version of the problem we have a polygonal environment P and a direction transition graph G_{Dir} and for any start and end points (s, e) we are interested in a G_{Dir}-admissible path between s and e inside P which connect s to e with minimum number of turns.

List of References

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars. "Probabilistic roadmaps for path planning in high dimensional configuration spaces." *IEEE Transactions on Robotics, 12 566-580* (1996).

[2] S. M. LaValle. "Rapidly-exploring random trees: A new tool for path planning." Technical report, Computer Science Department, Iowa State University (1998).

[3] A. M. Ladd and L. E. Kavraki. "Motion planning in presence of drift, underactuation and discerete system changes." In "Proceedings of Robotics: Science and System," (2005).

[4] D. Hsu, J.-C. Latombe, and R. Motwani. "Path planning in expansive configuration space." In "IEEE international Conference on Robotics and Automation, Volume 3, 2719-2726," (1997).

[5] J. F. Canny. *Complexity of Robot Motion Planning.* MIT Press (1988).

[6] J. T. Schwartz and M. Sharir. "On the piano movers problem ii: Topological properties of real algebraic manifolds." Technical report, Department of computer science Currunt Institute of Mathematical Sciences, New York University (1982).

[7] P. Cheng, G. Pappas, and V. Kumar. "Decidability of motion planning with differential constraints." In "Proceedings of IEEE International Conference on Robotics and Automation," (2007).

[8] F. Jean. "Complexity of nonholonomic motion planning." *International Journal of Control, Volume 74, Issue 8* (2001).

[9] P. K. Agarwal, T. C. Biedl, S. Lazard, S. Robbins, S. Suri, and S. Whitesides. "Curvature-constrained shortest paths in a convex polygon (extended abstract)." In "Symposium on Computational Geometry," pages 392–401 (1998).

[10] J.-D. Boissonnat and S. Lazard. "A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles (extended abstract)." In "Symposium on Computational Geometry," pages 242–251 (1996).

[11] P. K. Agarwal, T. C. Biedl, S. Lazard, S. Robbins, S. Suri, and S. Whitesides. "Curvature-constrained shortest paths in a convex polygon." *SIAM J. Comput.* **31**(6), 1814–1851 (2002).

[12] J.-D. Boissonnat, S. K. Ghosh, T. Kavitha, and S. Lazard. "An algorithm for computing a convex and simple path of bounded curvature in a simple polygon." *Algorithmica* **34**(2), 109–156 (2002).

[13] J.-D. Boissonnat and S. Lazard. "A polynomial-time algorithm for computing shortest paths of bounded curvature amidstmoderate obstacles." *Int. J. Comput. Geometry Appl.* **13**(3), 189–229 (2003).

[14] X. Goaoc, H.-S. Kim, and S. Lazard. "Bounded-curvature shortest paths through a sequence of points using convex optimization." *SIAM J. Comput.* **42**(2), 662–684 (2013).

[15] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer (2008).

[16] S. M. L. Valle. "Planning algorithms." *Unknown Journal* (2006).

[17] D. Arnon, G. E. Collins, and S. McCallum. "Cylindirical algebraic decomposition i: The basic algorithm." Technical report, Purdue University (1982).

[18] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. "A singly exponential stratification scheme for real semi-algebraic varieties and its applications." *Theoretical Computer Science 84* (1991).