An Approximate Algorithm for Solving Shortest Path Problems for Mobile Robots or Driver Assistance

Fajie Li
College of Computer Science and Technology
Huaqiao University
Quanzhou, Fujian, China
Email: li.fajie@yahoo.com

Reinhard Klette and Sandino Morales
The *.enpeda..* Project
Tamaki campus
The University of Auckland, New Zealand
Email: {r.klette@auckland,pmor085@aucklanduni}.ac.nz

***Abstract*—Finding a shortest path between two given locations is of importance for mobile robots, but also (e.g.) for identifying unique paths in a given surrounding region Π when (e.g.) evaluating vision software in test vehicles, or for calculating the free-space boundary in vision-based driver assistance. We assume that Π is given as a triangulated surface which is not necessary simply connected.**

Based on a known k-shortest paths algorithm and a decomposition of the surrounding region Π, this article presents an approximate algorithm for computing a general Euclidean shortest path (ESP) between two points p and q on Π, with time complexity

$$\kappa(\varepsilon) \cdot \mathcal{O}(k \cdot |V(\Pi)|)$$

and additional preprocessing in time

$$\mathcal{O}(k \cdot |V(\Pi)| \cdot \log |V(\Pi)|)$$

Our algorithm is suitable for approximately solving the 2D ESP problem, the 2.5 ESP problem (i.e., the surface ESP problem, as occurring, for example, in the free-space border application), and even the 3D ESP problem which is thought to be difficult even in the most basic case if all the obstacles are just convex, or if Π is just simply connected.

I. Introduction

Shortest-path calculations have a well-known importance for mobile robots, where shortest paths need to be calculated in 3D space. Currently, problems of shortest (or optimum) path calculations also occur in areas of vision-based driver assistance [8], [19], [20]. For example, a road surface is approximated by a 2D manifold (see Figure 1), and the boundary of the free space needs to be calculated as an optimum path in a space of labeled 3D positions. A non-planar road surface model defines an example for the 2.5 ESP problem as discussed in this paper.

Figure 2 illustrates the mapping of a road view (one of two stereo images) into a birds-eye image; see [2] for this mapping. The ground manifold is approximated by a (linear, cubic B-spline, and so forth) function in y which identifies the "zero-height" for each image row y, and thus also a disparity d_y identifying this zero-height in column y. Every detected disparity greater than $d_y + \delta_y$ defines an *obstacle*. The free-space boundary b is an optimized path, defined for all columns

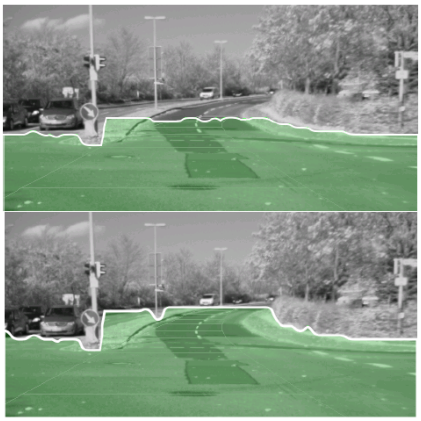

Fig. 1. Estimated free space assuming either a planar road (top), or a B-spline modeled curved road surface. Courtesy of A. Wedel, see [20].

Fig. 2. Top: road view. Bottom: birds-eye view.

x with $y_x = b(x)$, by minimizing the energy

$$E(b) = \sum_{x=0}^{x=x_{max}} D_x(y_x) + V_x(y_{x-1}, y_x, y_{x+1})$$

Start and end point $(0, y_0)$ and $(x_{max}, y_{x_{max}})$ are given from the birds-eye image. For each column x, the *data cost* is defined by $D_x(y) = y$, and the *continuity cost* by $V_x(y_1, y_2, y_3)$.

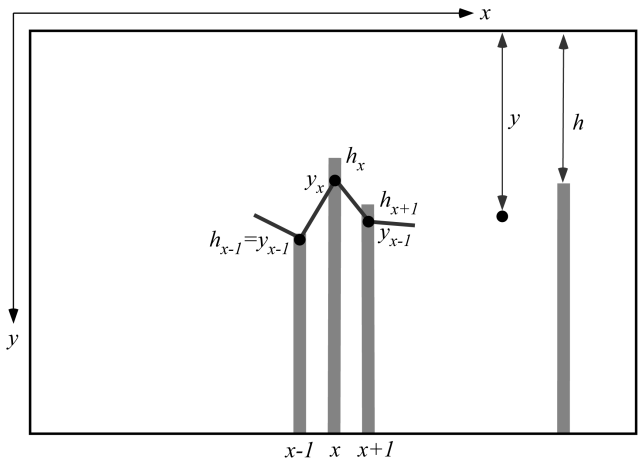

Fig. 3. Parameters in birds-eye image.

For each column x, let h_x be the minimum y-coordinate such that there is no obstacle between (x, y_{max}) and (x, h_x). Boundary value y_x is constraint to be in $[0, h_x]$. See Figure 3.

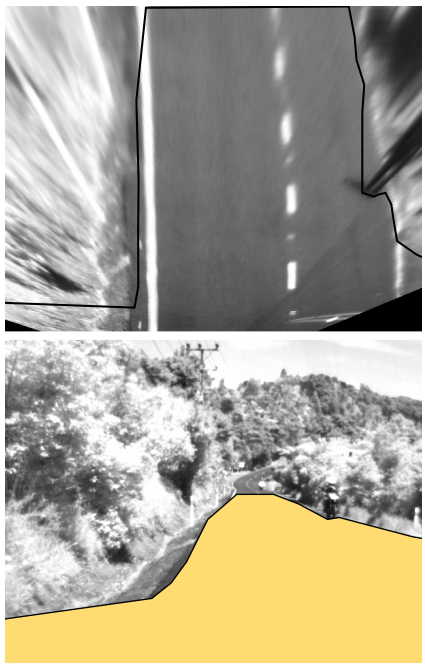

Fig. 4. Top: free-space in birds-eye image. Bottom: free-space in road view.

For example, by using a linear continuity term

$$V_x(y_1, y_2, y_3) = |y_1 - y_2| + |y_2 - y_3|$$

the continuity cost is basically (The difference in x-coordinates is equals to 1 and constant.) the L_1-length, and the intervals $[0, h_x]$ identify the step sets (using the data term as constraint). Thus, energy optimization may be estimated by length minimization.

Figure 4 illustrates a solution - first in the birds-eye image, and than back-projected into the road view.

The paper is structured as follows: Section II introduces into the current situation in Euclidean shortest path algorithms (time complexity, for 2D, 2.5D, or 3D cases). Section III provides necessary definitions and notation. Section IV presents our algorithm and explanation of its correctness. Section V illustrates the algorithm by a small example. Section VI analyses the time complexity of the algorithm. Section VII concludes the paper.

II. ESP Problems

Let Π be a polygon in 2D space (plane), and assume two points $p, q \in \Pi$, with $p \neq q$. The task to find a shortest path ρ between p and q, such that all the vertices of ρ are inside of Π, is an instance of an *Euclidean shortest path (ESP)* problem; it is called the *2D ESP problem*.

The 2D ESP problem has some generalizations such as the 2.5D ESP problem where Π is the surface of a polytope, or the 3D ESP problem where Π is the closure of the interior of a connected polyhedron (which is not necessarily simple). Obviously, ESP computation has an application in robot route planning, but also in more specific applications as the briefly mentioned calculation of free space in driver assistance.

Based on applying a linear (time) triangulation algorithm for a simple polygon [4], there exists a linear algorithm (see [17]) for solving the 2D ESP problem if Π is a simple polygon. Also based upon triangulation, [11] describes a more straightforward algorithm (a version of a so-called *rubberband algorithm*) which has a $\kappa(\varepsilon) \cdot \mathcal{O}(n)$ time for the same 2D ESP problem, where

$$\kappa(\varepsilon) = (L_0 - L_1)/\varepsilon$$

n is the number of vertices of Π, ε is the accuracy (say, the numerical accuracy of the used computational environment, something like $\varepsilon = 10^{-10}$), L_0 is the length of the initial path and L_1 is the true (i.e., optimum) path length.

The 2.5D ESP problem is more difficult to solve than the 2D problem. So far, the best known result for the surface ESP problem is due to [7]; this paper improved in 1999 the time complexity to $\mathcal{O}(n\ log^2 n)$, assuming that there are $\mathcal{O}(n)$ vertices and edges on Π. [10] apples a version of a rubberband algorithm to solve the 2.5D ESP problem in time $\kappa_1(\varepsilon) \cdot \kappa_2(\varepsilon) \cdot \mathcal{O}(n^2)$, basically using a simpler approach compared to [7].

For calculating an ESP on the surface of a convex polytope (in $\mathbb{R}^3$), [17] states on page 667 the following *open problem*:

> Can one compute shortest paths on the surface of a convex polytope in $\mathbb{R}^3$ in subquadratic time? In $\mathcal{O}(n\ log\ n)$?

The 3D ESP problem is thought to be "very difficult". In 1985, [18] described an algorithm for the general 3D ESP problem in time

$$\mathcal{O}(n^4(L + \log(n/\varepsilon))^2/\varepsilon^2)$$

In 1987, [5] gave an algorithm for computing an $(1 + \varepsilon)$-shortest path between p and q which has a time complexity of

$$\mathcal{O}(n^2\lambda(n)\log(n/\varepsilon)/\varepsilon^4 + n^2 \ \log \ nr \ \log(n \ \log r))$$

where r is the ratio of the Euclidean distance $d_e(p, q)$ to the length of the longest edge of any given obstacle, and

$$\lambda(n) = \alpha(n)^{\mathcal{O}(\alpha(n)^{\mathcal{O}(1)})}$$

where $\alpha(n) = A^{-1}(n, n)$ is an inverse Ackermann function [14], which grows very slowly (because A grows very rapidly).

Let there be a finite set of polyhedral obstacles in $\mathbb{R}^3$. Let p, q be two points outside of the union of all obstacles. Assume that $0 < \varepsilon < 1$; [6] gives an $\mathcal{O}(log(n/\varepsilon))$ algorithm to compute an $(1 + \varepsilon)$-shortest path from p to q such that it avoids the interior of any obstacle. The algorithm is based on a subdivision of $\mathbb{R}^3$ which is computed in $\mathcal{O}(n^4/\varepsilon^6)$.

For some special cases of 3D ESP problems, [17] states on page 666 the following:

> The problem is difficult even in the most basic Euclidean shortest-path problem (ESP) in a three-dimensional polyhedral domain P, and even if the obstacles are convex, or the domain P is simply connected.

In this paper, based on an k-shortest paths algorithm ([15]) and the decomposition (see the Definition in Section III) of Π, we apply a version of a rubberband algorithm to present a

$$\kappa(\varepsilon) \cdot \mathcal{O}(kn)$$

approximate algorithm (described in Section IV) for 3D ESP calculation if P is a connected polyhedron (which is not necessarily simple), with preprocessing time complexity $\mathcal{O}(kn \ log \ n)$. The algorithm has the same time complexity for the general 2.5D surface ESP problem or the 2D ESP problem (where Π is not necessarily a simple polygon).

III. Definitions and Notation

A (surrounding) region Π of a Euclidean shortest path (ESP) problem may be a connected closed set in 2D space (plane), in 2.5D space (surface), or in 3D space such that Π is a union of triangles which all have the same normal (in 2D space), a union of triangles which may have different normals (in 2.5D space), or a union of tetrahedra (in 3D space).

The set of such triangles or tetrahedra is called the *decomposition* of Π, denoted by Π_t. Each triangle or tetrahedron is called an *element* of Π_t, denoted by t. Let $Dim(t)$ be the dimension of t. Let w be the centroid of t; t is also called the *corresponding* element with respect to w, denoted by $t(w)$.

A rubberband algorithm proceeds by identifying vertices of a path in subsequent "steps". A *step region* is a closed segment if Π is a 2D or 2.5D region, or it is a closure of the interior of a triangle if Π is a 3D region. A *step set* is a finite sequence of disjoint step regions.

Let $V(\Pi)$ be the set of vertices of Π. Let $V(G)$ be the set of vertices of a graph G.

IV. Three Algorithms

[3] proposed a rubberband algorithm for calculating shortest path in a 3D world subdivided into cubes of uniform size; this algorithm was extensively studied in [9].

This section starts describing a "general" version of a rubberband algorithm and a k-shortest paths algorithm [15]; then it presents the main algorithm based on these two procedures.

Algorithm 1: A "General" Rubberband Algorithm

Input: A step set $\{S_1, S_2, \ldots, S_k\}$, where i = 1, 2, ..., k; two points p, $q \notin S_i$.

Output: An approximate (Euclidean) shortest path which starts at p, then visits S_i in order, and finally ends at q.

1: Let $\varepsilon = 10^{-10}$ (i.e., this is an example of a chosen accuracy).
2: **for** each $i \in \{1, 2, \ldots, k\}$ **do**
3: Let p_i be a point in S_i.
4: **end for**
5: Compute the length L_0 of the path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
6: Let $q_1 = p$ and i = 1.
7: **while** $i < k - 1$ **do**
8: Let $q_3 = p_{i+1}$.
9: Compute a point $q_2 \in S_i$ such that $d_e(q_1, q_2)+d_e(q_3, q_2) = \min\{d_e(q_1, q')+d_e(q_3, q') : q' \in S_i\}$.
10: Update ρ by replacing p_i by q_2.
11: Let $q_1 = p_i$ and $i = i$ + 1.
12: **end while**
13: Let $q_3 = q$.
14: Compute $q_2 \in S_k$ such that $d_e(q_1, q_2)+d_e(q_3, q_2) = \min\{d_e(q_1, q)+d_e(q_3, q) : q \in S_k\}$.
15: Update ρ by replacing p_k by q_2.
16: Compute the length L_1 of the updated path $\rho = \langle p, p_1, p_2, \ldots, p_k, q\rangle$.
17: Let $\delta = L_0 - L_1$.
18: **if** $\delta > \varepsilon$ **then**
19: Let $L_0 = L_1$ and go to Step 6.
20: **else**
21: Output $\{p, p_1, p_2, \ldots, p_k, q\}$ and Stop.
22: **end if**

In Algorithm 1, we let the step region S_i be a closed segment for the 2D ESP problem [11], and a closure of the

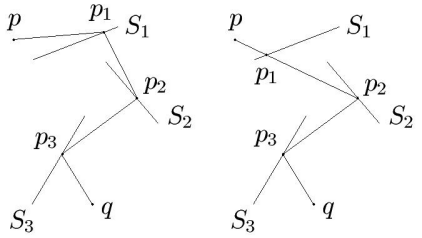

Fig. 5. Illustration of Algorithm 1: Step 3 and optimal p_1.

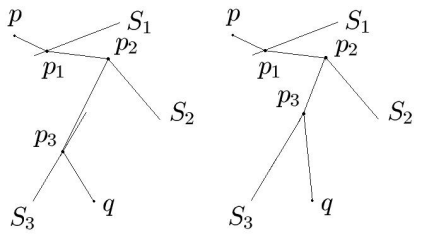

Fig. 6. Illustration of Algorithm 1: Optimal p_2 and p_3.

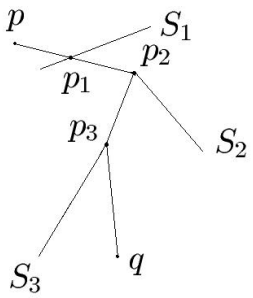

Fig. 7. Illustration of Algorithm 1: Final optimal path.

interior of a triangle for the 2.5D or 3D ESP problem (see also Algorithm 1 in [12] and Algorithm 4 in [13]).

Figure 5 (left) shows an initial path in Algorithm 1 when k = 3 and each step region S_i is a closed segment in a plane. In the first iteration, we update p_1 (Figure 5, right), then p_2 (Figure 6, left) and finally p_3 (Figure 6, right) in this order. The final optimal path is shown in Figure 7.

We recall from [15] the following (with referring to the source for details):

Algorithm 2: k-Shortest Paths Algorithm

Input: A weighted directed graph $G = [V, E]$, two vertices u, $v \in V$, and an integer $k > 0$.

Output: The first k shortest paths between vertices u and v.

Now we are ready to formulate the main algorithm of this paper.

Algorithm 3: Main Algorithm

Input: Two points p and q in a closed connected region S; the decomposition of Π, denoted by Π_t, and an integer k.

Output: An approximate Euclidean shortest path ρ between p and q inside of Π.

1: Let $\Pi_t = \{t_p, t_q, t_1, t_2, \ldots, t_m\}$.
2: **for** each $i \in \{1, 2, \ldots, m\}$ **do**
3: Compute the centroid of t_i, denoted by u_i.
4: **end for**
5: Construct a weighted directed graph $G = [V, E]$ as follows: Let $V = \{p, q, u_1, u_2, \ldots, u_m\}$. E is defined such that, for any w_i, $w_j \in V$ and $w_i \neq w_j$, there exist two weighted directed arcs, denoted by (w_i, w_j) and (w_j, w_i), if and only if [1] $Dim(t(w_i) \cap t(w_j)) = Dim(t(w_i)) - 1$. The weight of (w_i, w_j) [or of (w_j, w_i)] is the length of the shortest path that starts at u_i [or u_j], then visits $t(w_i) \cap t(w_j)$, and finally ends at u_j [or at u_i].
6: Apply Algorithm 2 to find the first k shortest paths between p and q in G, denoted by ρ_1, ρ_2, ..., ρ_k.
7: **for** each $i \in \{1, 2, \ldots, k\}$ **do**
8: Let w_{i_1}, w_{i_2}, ..., $w_{i_{n_i}}$ be the vertices of ρ_i (excluding the starting vertex p and ending vertex q).
9: Let $L = \infty$ (i.e., a sufficiently large number).
10: **for** each $j \in \{1, 2, \ldots, n_i\}$ **do**
11: Let $S_{i_0} = t_p \cap t(w_{i_1})$, $S_{i_j} = t(w_{i_j}) \cap t(w_{i_{j+1}})$, where j = 1, 2, ..., n_i - 1, and $S_{i_{n_i}} = t(w_{i_{n_i}}) \cap t_q$.
12: Let $\{S_{i_0}, S_{i_1}, S_{i_2}, \ldots, S_{i_{n_i}}\}$ (as a step set) and p, q as input, apply Algorithm 1 to compute an approximate ESP, denoted by ρ'_i.
13: Let L_i be the length of ρ'_i.
14: **if** $L_i < L$ **then**
15: Let $\rho = \rho'_i$ and $L = L_i$.
16: **end if**
17: **end for**
18: **end for**
19: Output ρ.

In Step 12, elements in step sets could be removed until remaining step sets are all pairwise disjoint. For example, just simply remove a sufficiently small segment from both ends of a line segment in case of a 2D ESP problem ([11]).

In Step 5, G is called the *corresponding* graph with respect to the decomposition Π_t.

The correctness of Algorithm 3 follows by Theorem 2 of [12] (or Theorem 2 in [13]).

V. An Example

This section illustrates some steps of Algorithm 3 under the assumption that Π is the simple 2.5D surface as shown in Figure 8. (A sampled cubic B-spline manifold, approximating the road surface [20], is a real-world example for a 2.5D case.)

In Step 1, $\Pi_t = \{t_p, t_q, t_1, t_2, \ldots, t_{11}\}$, where $t_p = \triangle v_1 v_2 v_5$, $t_q = \triangle v_4 v_7 v_9$, $t_1 = \triangle v_1 v_5 v_{10}$, $t_2 = \triangle v_2 v_8 v_5$, $t_3 = \triangle v_1 v_{10} v_4$, $t_4 = \triangle v_5 v_7 v_{10}$, $t_5 = \triangle v_5 v_8 v_6$, $t_6 = \triangle v_2 v_3 v_8$, $t_7 = \triangle v_4 v_{10} v_7$,

[1] For example, $t_i \in S_t$ is the triangle corresponding to $u_i \in V$, where i = 1, 2, ..., m.

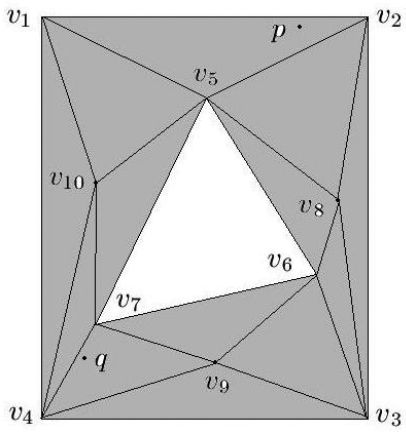

Fig. 8. Illustration of a 2.5D surface Π of Algorithm 3.

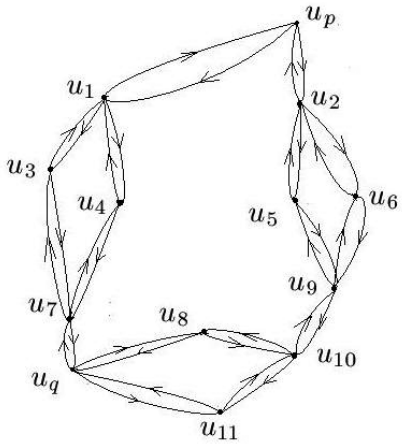

Fig. 9. Illustration for the corresponding graph G in Step 5 of Algorithm 3.

$t_8 = \triangle v_6 v_9 v_7$, $t_9 = \triangle v_3 v_6 v_8$, $t_{10} = \triangle v_3 v_9 v_6$, and $t_{11} = \triangle v_3 v_4 v_8$.

Figure 9 shows the corresponding graph G with respect to decomposition Π_t.

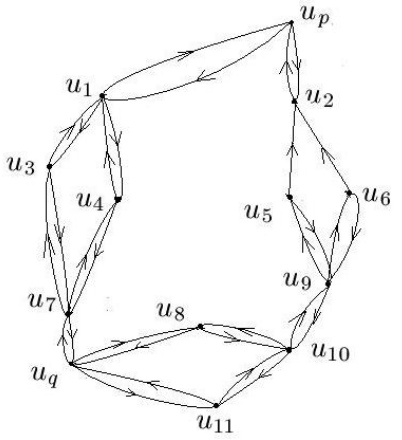

Fig. 10. Illustration for another possible corresponding graph G in Step 5 of Algorithm 3.

Figure 10 shows a possible corresponding graph G with respect to decomposition Π_t. Figure 10 is obtained by removing two arcs (u_2, u_5) and (u_2, u_6) in Figure 9 under the assumption that a move is not possible from u_2 to u_5 or from u_2 to u_6.

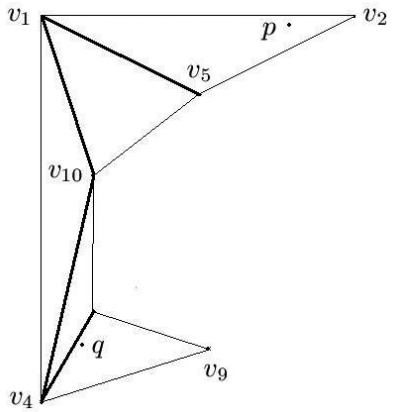

Fig. 11. Illustration for a step set (bold edges) obtained in Step 11 of Algorithm 3.

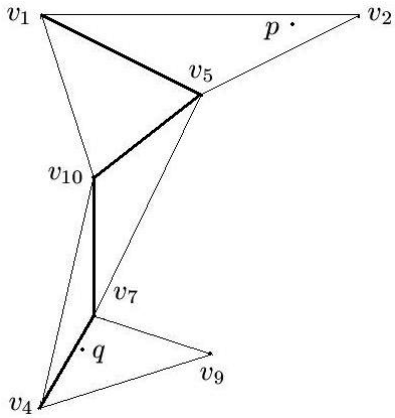

Fig. 12. Illustration for another step set (bold edges) obtained in Step 11 of Algorithm 3.

Figures 11 and 12 show two step sets obtained in Step 11 of Algorithm 3 from $\rho_i = \langle p, u_1, u_3, u_7, q\rangle$ and $\langle p, u_1, u_4, u_7, q\rangle$, respectively, in Step 8 of Algorithm 3.

VI. Time Complexity

In Algorithm 3, the main preprocessing step is Step 6 with costs (in time) from

$$\mathcal{O}(k|E|) \text{ to } \mathcal{O}(k|V|\log|V|)$$

[15]. As each vertex of G can be at most of degree three (i.e., number of incident edges) if Π is in 2D or 2.5D space, and at most of degree four if Π is in 3D space, $|E| \leq 2|V|$. Thus, Step 6 can be computed in $\mathcal{O}(k|V|\ log\ |V|)$. By Lemma 1 of [12], Step 12 can be computed in time $\kappa(\varepsilon) \cdot \mathcal{O}(n_i) \leq \kappa(\varepsilon) \cdot \mathcal{O}(|V|)$. The main computation in Algorithm 3 occurs in Steps 7–18, and those steps require $\kappa(\varepsilon) \cdot \mathcal{O}(k|V|)$ time. Therefore, Algorithm 3 has a time complexity of

$$\kappa(\varepsilon) \cdot \mathcal{O}(kn)$$

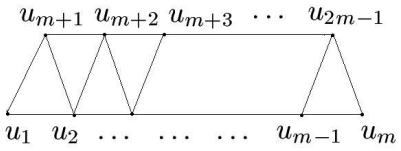

Fig. 13. A simple graph that has more than $2^{|V|-1}$ paths between u_1 and u_m.

with preprocessing in time

$$\mathcal{O}(kn \log n)$$

where n is the number of vertices of Π.

VII. Concluding Remarks

We have presented an approximate algorithm based on a k-shortest paths algorithm and a decomposition of the surrounding region Π for computing a general ESP between two points p and q inside of Π, with time complexity

$$\kappa(\varepsilon) \cdot \mathcal{O}(k|V(\Pi)|)$$

excluding preprocessing in time

$$\mathcal{O}(k|V(\Pi)| \log |V(\Pi)|)$$

Although there exist some algorithms such as in [16] for computing all paths between two vertices in a graph, the time complexity of such an algorithm is exponential due to the existence of graphs G as shown in Figure 13, which has $2^{|V(G)|-1}$ paths between two of its vertices. Thus, in general, the smallest upper bound for parameter k is probably exponential.

It is known that there does not exist (!) any algorithm for finding exact solutions for general 3D ESP problems (see Theorem 9, [1]). Therefore, an approximate algorithm is actually the only option to approach these problems.

Applications of the reported algorithm in driver assistance (e.g., for the mentioned free-space problem) are currently underway. [20] applies dynamic programming for calculating the optimum path b, identifying the boundary of the free-space. A run-time and accuracy comparison with the proposed rubberband algorithm will be a subject of future work.

The authors hope that researchers in robotics may also find it of interest to test the given algorithm for their purposes.

References

[1] C. Bajaj. The algebraic complexity of shortest paths in polyhedral spaces. In Proc. *Allerton Conf. Commun. Control Comput.*, pages 510–517, 1985.

[2] M. Bertozzi, A. Broggi: GOLD:A parallel real-time stereo vision system for generic obstacle and lane detection. *IEEE Trans. Image Processing*, **7**:62–81, 1998.

[3] T. Bülow and R. Klette. Digital curves in 3D space and a linear-time length estimation algorithm. *IEEE Trans. Pattern Analysis Machine Intelligence*, **24**:962–970, 2002.

[4] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete Computational Geometry* **6**:485–524, 1991.

[5] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In Proc. *Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[6] S. Har-Peled. Constructing approximate shortest path maps in three dimensions. In Proc. *Annu. ACM Sympos. Comput. Geom.*, pages 125–130, 1998.

[7] S. Kapoor. Efficient computation of geodesic shortest paths. In Proc. *Annu. ACM Sympos. Theory Comput.*, pages 770–779, 1999.

[8] J. Klappstein, T. Vaudrey, C. Rabe, A. Wedel, and R. Klette. Moving object segmentation using optical flow and depth information. In Proc. *PSIVT* (T. Wada, F. Huang, and S. Lin, editors), LNCS 5414, pages 611–623, 2009.

[9] F. Li and R. Klette. Exact and approximate algorithms for the calculation of shortest paths. Report 2141 on www.ima.umn.edu/preprints/oct2006.

[10] F. Li, R. Klette and X. Fu. Approximate ESPs on Surfaces of Polytopes Using a Rubberband Algorithm. In Proc. *2007 IEEE Pacific-Rim Symposium on Image and Video Technology*, LNCS 4872, pages 236–247, Springer, Berlin, 2007.

[11] F. Li and R. Klette. Euclidean Shortest Paths in Simple Polygons. Indian Statistical Institute Platinum Jubilee Volume on Algorithms, Architecture, and Information Security, World Scientific, Delhi, pages 3–24, 2008.

[12] F. Li and R. Klette. Approximate Shortest Path Calculations in Simple Polyhedra. MI-tech TR 23, The University of Auckland, 2008 (http://www.mi.auckland.ac.nz/tech-reports/MItech-TR-23.pdf).

[13] F. Li and R. Klette. Approximate Algorithms for Touring a Sequence of Polygons. MI-tech TR-24, The University of Auckland, Auckland, 2008 (http://www.mi.auckland.ac.nz/tech-reports/MItech-TR-24.pdf).

[14] Y. A. Liu. and S. D. Stoller. Optimizing Ackermann's function by incrementalization. In Proc. *ACM SIGPLAN Sympos. Partial Evaluation Semantics-Based Program Manipulation*, pages 85–91, 2003.

[15] E. de Q. V. Martins, M. M. B. Pascoal, and J. L. E. dos Santos. A new improvement for a K shortest paths algorithm. *Investigacão Operacional* **21**:47–60, 2001. (http://www.mat.uc.pt/ marta/Publicacoes/ms _improved.ps.gz)

[16] M. Migliore, V. Martorana, and F. Sciortino. An algorithm to find all paths between two nodes in a graph. *Journal of Computational Physics*, **87**:231–236, 1990.

[17] J. S. B. Mitchell. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry* (J.-R. Sack and J. Urrutia, editors), pages 633–701, Elsevier, 2000.

[18] C. H. Papadimitriou. An algorithm for shortest path motion in three dimensions. *Inform. Process. Lett.*, **20**:259–263, 1985.

[19] A. Wedel, U. Franke, H. Badino, and D. Cremers. B-spline modeling of road surfaces for freespace estimation. In Proc. *IEEE Intelligent Vehicles Symposium*, pages 828–833, 2008.

[20] A. Wedel, U. Franke, H. Badino, and D. Cremers. B-spline modeling of road surfaces with an application to free space estimation. *IEEE Trans. ITS*, special issue for IV'08, 2008.