Timed Fast Exact Euclidean Distance (tFEED) Maps

Theo E. Schouten[a], Harco Kuppens[a] and Egon L. van den Broek[b]

[a]Nijmegen Institute for Computing and Information Science,
Radboud University Nijmegen,
P.O. Box 9010, 6500 GL Nijmegen, The Netherlands
{T.Schouten,H.Kuppens}@cs.ru.nl
http://www.cs.ru.nl/~{ths,harcok}/
[b]Nijmegen Institute for Cognition and Information,
Radboud University Nijmegen,
P.O. Box 9104, 6500 HE Nijmegen, The Netherlands
e.vandenbroek@nici.ru.nl
http://eidetic.ai.ru.nl/egon/

ABSTRACT

In image and video analysis, distance maps are frequently used. They provide the (Euclidean) distance (ED) of background pixels to the nearest object pixel. In a naive implementation, each object pixel feeds its (exact) ED to each background pixel; then the minimum of these values denotes the ED to the closest object. Recently, the Fast Exact Euclidean Distance (FEED) transformation was launched, which was up to 2x faster than the fastest algorithms available. In this paper, first additional improvements to the original FEED algorithm are discussed. Next, a timed version of FEED (tFEED) is presented, which generates distance maps for video sequences by merging partial maps. For each object in a video, a partial map can be calculated for different frames, where the partial map for fixed objects is only calculated once. In a newly developed, dynamic test-environment for robot navigation purposes, tFEED proved to be up to 7x faster than using FEED on each frame separately. It is up to 4x faster than the fastest ED algorithm available for video sequences and even 40% faster than generating city-block or chamfer distance maps for frames. Hence, tFEED is the first real time algorithm for generating exact ED maps of video sequences.

Keywords: Exact Euclidean distance, FEED, tFEED, distance maps/transforms, video processing, robot navigation

1. INTRODUCTION

In the areas of computer vision, image, and video processing, it is usually required to extract information about the shape and the position of the foreground pixels relative to each other. Subsequently, many techniques evolved to accomplish this task; one such technique is the distance transform (DT). The DT converts a binary image to another image, such that each pixel has a value that represents the distance to its nearest object pixel. The new image is called the distance map of the old image.

Rosenfeld and Pfaltz[1, 2] introduced the first movements that could be utilized for the generation of distance maps for digital images: the city-block and chessboard distance (d_4 and d_8). The city-block distance allows measuring only in horizontal and vertical directions, while the chessboard distance takes diagonal directions also in consideration. So, the d_4 or d_8 distance of two points is the number of steps required to reach either point from the other, where only city-block or chessboard movements can be used, respectively. To obtain a better approximation for the Euclidean distance (ED), Rosenfeld and Pfaltz recommended the alternate use of the city-block and chessboard steps, which defines the distance d_{oct}. Geometrically, the corresponding "disks" are diamonds for the distance d_4, squares for d_8, and octagons for d_{oct}. Hence, d_{oct} provided the best approximation of the ED out of these three distances. However, there are many different DTs available using different distance metrics.

Send correspondence to Theo E. Schouten, E-mail: T.Schouten@cs.ru.nl

In principle, one wants to determine the exact ED. Then, a ED map can be determined for all object points (i, j) in a (binary) image. Such a map is determined by a ED transform (EDT), which can be computed as:

$$d_{ij} = min\{((i-x)^2 + (j-y)^2)^{1/2}\},\ i, j \in \mathbb{N}, \tag{1}$$

where x and y are object pixels.

The EDT is a basic operation in computer vision, pattern recognition, and robotics. For instance, if the object pixels represent obstacles, then d_{ij} tells us how far the point (i, j) is from these obstacles. This information is useful when one tries to move a robot in the free space and to keep it away from the obstacles. However, finding the DT with respect to the Euclidean metric is rather time consuming. In order to tackle the computational burden of EDT, two strategies have been adopted: (i) parallel implementations and (ii) approximation of exact EDs.

Ten years after Rosenfeld and Pfaltz[1] introduced their DTs, Borgefors[3] extended them to chamfer DTs, where during the scans different weights are given to neighboring pixels to produce better approximations of the ED. In the 90s, several other (parallel) implementations of EDTs have been proposed[4–7]. However, even among the parallel implementations, mostly the EDs were approximated.

In 1998 Shih and Liu[8] presented their method to obtain EDTs. They started with four scans on the image. Next, a look-up table method was used to correct the wrong pixels. For a large majority of cases, they were able to determine exact EDTs. Three years later, Costa[9] presented such a method by using the concept of exact dilations. Again, after a period of three years, Shih and Wu[10] introduced their two scan method, with which they claimed to be able to obtain true exact EDTs. In the same year, Schouten and Van den Broek[11] presented their Fast Exact Euclidean Distance (FEED) transformation. With FEED they introduced an algorithm, which obtained an exact EDT in a computational cheap way.

In the next Section (Section 2), we briefly discuss the principle of FEED and the original methods used to obtain a fast execution time. This is followed by descriptions of improvements, increasing the speed by about 25%, and by visualization methods which were used during the developements described in this paper and which further can be used for tuning of parameters to optimise the speed for new image sets. In Section 3, the improved version of FEED is compared with four other DT methods. Next, in Section 4, we discuss applications for EDTs, in particular robot navigation, and introduce a newly developed, dynamic test-environment for robot navigation purposes. It is explained how exact EDTs are generated of video sequences. Next, timed FEED (tFEED) is introduced in Section 5, which uses FEED to generate exact ED maps separately for fixed and moving objects and then combines them. In order to evaluate tFEED for video sequencing, the algorithms used to evaluate FEED (Section 3) were adapted for video sequencing purposes. These adaptations are discussed followed by the comparison of tFEED with the four adapted DT methods. We end this paper with a discussion 6 of the work presented as well as proposals for future research.

2. FAST EXACT EUCLIDEAN DISTANCE (FEED) TRANSFORMATION

The FEED algorithm[11] calculates the EDT starting directly from the definition (see Equation 1), or rather its inverse: each object pixel *feeds* its ED to all pixels in the image, which in turn calculate the minimum of all received EDs. The naive algorithm then becomes:

$$\begin{array}{ll} (1) & initialize\ D(p)\ =\ if\ (p\ \in\ O)\ then\ 0,\ else\ \infty \\ (2) & foreach\ q \in O \\ (3) & \quad foreach\ p \\ (4) & \quad\quad update:\ D(p)\ =\ min\{D(p)\ ,\ ED(q,p)\} \end{array} \tag{2}$$

Compared to the originally presented algorithm, in algorithm 2 a small adaption is made. The third line of the original algorithm was $foreach\ p\ \notin\ O$. However, the restriction $\notin\ O$ is not needed, removing it has no effect on the functionality of the algorithm since its initialization is done in the first line; it even increased the speed of the naive algorithm.

On the one hand, the disadvantage of this algorithm (still) is its computational expensiveness; on the other hand, it can be easily proven to be correct using classical methods. Therefore, we executed the naive algorithm on a large set of test images, which provided us with a reference set of distance maps. Subsequently, this reference set was used for testing the functionality of the following speedup methods applied to it.

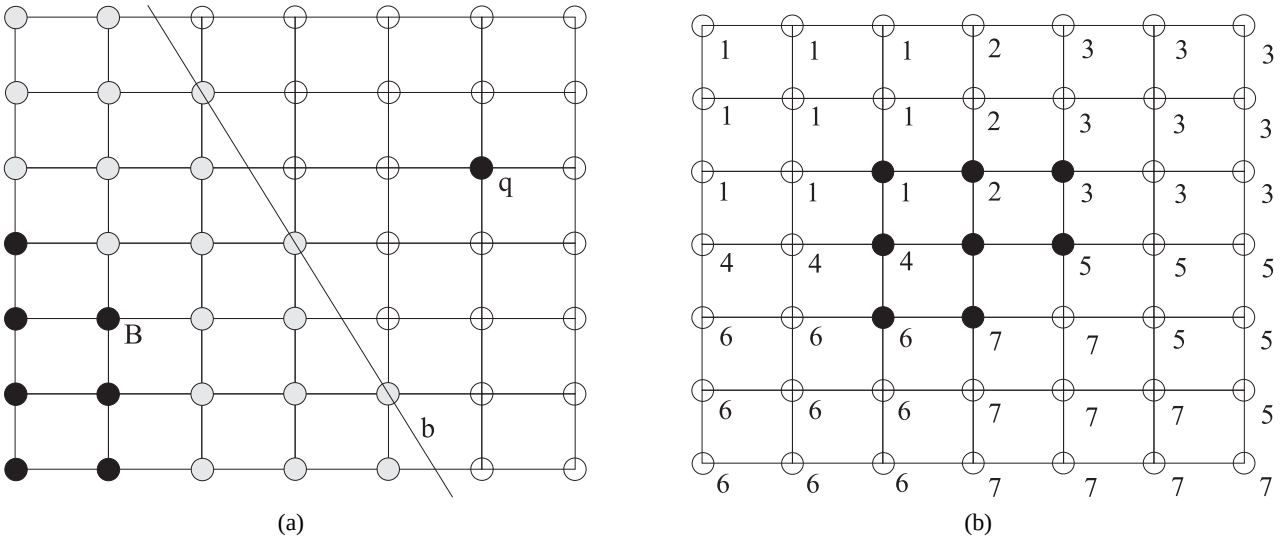

Figure 1. Principle of limiting the number of background pixels to update. (a) Only pixels on and to the left of the bisection line b between a border pixel B and an object pixel q have to be updated. (b) An example showing that each background pixel has to be updated only once. Each background pixel is labeled by the border pixel which updates it.

In line (2) only the "border" pixels B of O have to be considered because the minimal ED from any background pixel to the set O, is the distance from that background pixel to a border pixel B of O. A border pixel B is defined here as an object pixel with at least one of its four 4-connected pixels in the background.

The ED in line (4) can be retrieved from a pre-computed matrix M with a size equal to the image size:

$$ED((x_q, y_q), (x_p, y_p)) = M(|x_q - x_p|, |y_q - y_p|)$$

Due to the definition of $D(p)$, the matrix M can be filled with any non-decreasing function f of ED:

$$f(D(p)) = min(f(D(p)), f(ED(q, p))).$$

For instance, the square of ED allowing the use of an integer matrix M in the calculation to make it faster. Alternately, one can truncate the ED to integer values in M when it is stored in such format in the final $D(p)$ to be used for further image processing. Again this makes the method faster and avoids a conversion in the further processing chain.

Moreover, the number of pixels that have to be considered in line (3) can be limited to only those that have an equal or smaller distance to the current B than to any object pixel q (see Figure 2a). By taking all bisection lines into account, it can be assured that each background pixel is updated only once (see Figure 2b). For that, background pixels on a bisection line are only updated when B is on a chosen side of the line.

However, searching for and bookkeeping of bisection lines takes time and that time should be smaller than the time gained by updating less pixels, otherwise no speedup would be obtained but the algorithm would become slower. Bisection lines closer to the origin B have a larger effect than lines further away. Since in general a number of object points from the same object are close to B, they are located first by scanning a small area around B.

The search for object pixels q further away is done along a set of lines under certain angles m starting from the current border pixel (B). Not all the pixels on a line are checked for being an object pixel, but a certain stepping can be used depending on the expected size of the objects in the image. A q found then defines a bisection line, which can be written in the form $m_a y + m_b x = m_c q_x$, with m_a, m_b and m_c being integers that depend only on m. Further searching along the line can then be stopped.

To keep track of the update area, the maximum x and y values of each quadrant around the current B are updated (see Figure 2a). Only pixels inside the rectangles defined by the maximum values need to be updated, but not all of them as bisection lines might define cuts in the rectangles. A new bisection line b_{new} in a quadrant might update these maximum

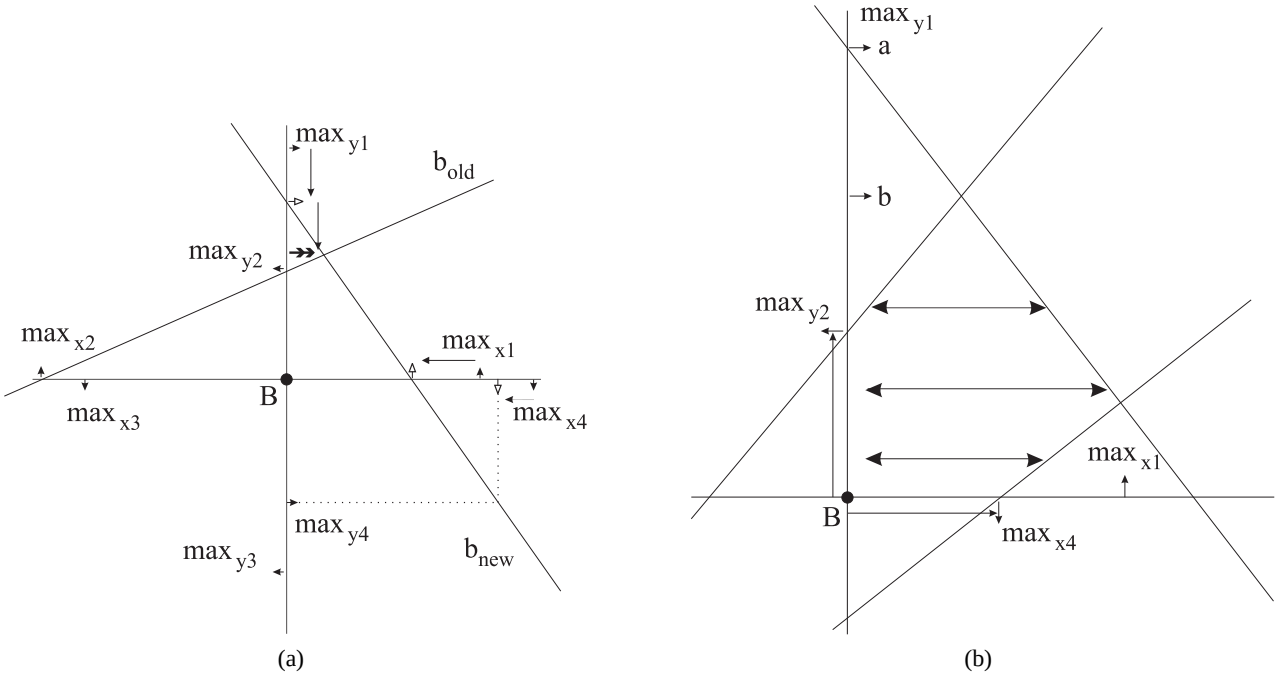

Figure 2. Principle of the bookkeeping and updating. (a) Keeping track of the maximum x and y values per quadrant around a border pixel. (b) Updating along scan lines: bisection lines determine start and end points on them.

values in that and the two neighboring quadrants. In Figure 2b this is indicated with the arrows along the axis, the closed arrows perpendicular to the axis give the old position of the maximum values while the open arrows indicate the new positions. Note that in this case there is no update on max_{y2}.

The intersection point of two bisection lines in different quadrants might also give a new maximum value. This is shown in the figure where the intersection point of b_{new} and b_{old} decreases max_{y1} to the position indicated by the double arrow. Again, this takes time and doing it for all possible intersections might reduce the speed.

The maximum values also determine the maximum distance to search along a new search line, because a found new bisection line should cut into the rectangular area's to have an effect. Further if the area that they define is small enough, searching is stopped because no further time gain can be expected.

The final selection of pixels to update is made in the update process. For each scan line in a quadrant, the maximum x and y values of the quadrant and the found bisection lines in that and neighboring quadrants, determine start and end points, as is indicated with the arrowed lines in Figure 2b.

In addition, some further speedups are implemented, using information saved from a border point for next border points. By searching for border pixels along horizontal scan lines, the search for object pixels along the $m = 0$ line can be combined with it. For searching in the vertical direction, a binning of the image in the vertical direction is used, which is done combined with the initialization of $D(p)$. Pixels exactly on a bisection line, have only to be updated once. This is done in quadrants 1 and 2, by simply decreasing $m_c q_x$ by 1 for quadrants 3 and 4.

2.1. Speedup of FEED

We will now discuss a set of changes and additions that have been applied to FEED. The three most important changes are:

1. The decision to stop searching because the remaining area is small enough, is now based on an estimation of that area, which also takes bisection lines under 45° into account.

2. The maximum values in a quadrant can be at several locations, depending on whether crossings of bisection lines were always calculated during the search process. In Figure 2b this is shown for max_{y1}, which might be at locations

a or b. In order to find the crossing of a bisection line in a quadrant with a bisection line in the next quadrant, it is now checked whether the minimum value of a scan line is larger than the maximum value. In that case no further scan lines in that quadrant are considered. This also means that the effect on the execution time of the determination of crossing of bisections directly during the search process is less. Subsequently, this is only done for bisection lines of the same angle m.

3. The update process is speeded up by distinguishing special cases:
   - When max_x is smaller than a certain value, begin and end points are not determined for each scan line but the whole rectangular area is updated.
   - Pixels along the horizontal and vertical axis are updated separately from the inside quadrant area's. A further speedup is achieved here by not taking the EDs from the matrix M but by recalculating them as they are simply equal to the coordinate along the axis.
   - Quadrants for which only pixels along the $45°$ line (and possible a line parallel to it, shifted by 1 pixel) have to be updated, are also handled separately by stepping along the lines.
4. Finally, the binning of the image in the vertical direction is replaced by a combination of updating top values during the scan over the image and by more searching in the forward vertical direction.

The consequences of all changes were carefully considered and thoroughly tested before they were incorporated. Together they resulted in an average speedup of 25%, compared to the first release of FEED, as introduced by Schouten and Van den Broek.[11]

2.2. VISUALIZATION OF PARAMETER EFFECT

To get a grip on the consequences of tuning, we have developed two visualization methods. One shows the changes in the maximum x and y values in the quadrants and the bisection lines during the search process separately for each border pixel. The other shows constantly during the execution of the program how often pixels have been updated. The final result of this is shown in Figure 3 for three settings of the parameters, accompanied by the original input image.

Figure 3 shows the effect of the search effort on the number of updates and on the execution times. The input image (Figure 3a) consists of 76800 pixels of which there are 13942 object (non-white) pixels, with 1725 of them being border pixels. The other images show the number of updates per pixel with black indicating 0 updates and white indicating 4 or more updates. Pixels with 1,2 or 3 updates get an intermediate gray level of respectively 64, 128, and 192. Figure 3b shows too little searching, resulting in the large number of 290771 updates (where 62858 is the minimum number of updates) and subsequently in a large execution time (5.7 ms). In contrast, Figure 3c shows too much searching with its 86487 updates, close to the minimum. The execution time is very large with 8.4 ms. Figure 3d shows the optimal settings of the parameters, which produces 179373 updates and the minimal execution time of only 4.5 ms.

The execution times were determined on a standard PC with an AMD Athlon XP®1666 MHz processor (64kB L1 cache, 256kB L2 cache) and 512 MB memory and using the Microsoft®Visual C++ 6.0 programming environment in the standard release setting. Please note that no effort was spend on reducing the execution time by varying compiler optimization parameters, by using pointers instead of indices or by exploiting the particular characteristics of the machine.

3. COMPARISON

We are interested in obtaining an exact EDT in a limited amount of time, such that it can be used for real time image processing purposes. Therefore, we compared FEED with two fast approximations of the ED and with two state-of-the-art algorithms, which produce (almost) exact EDs. So, in total FEED is evaluated against four other methods for obtaining EDTs, with respect to both accuracy and speed. These four other methods are:

1. The city-block distance (CH1,1), as introduced by Rosenfeld and Pfaltz[1, 2], which gives the crudest but fastest approximation of the ED.
2. The Chamfer 3,4 distance (CH3,4) from Borgefors[3], which gives a more accurate approximation of the ED.

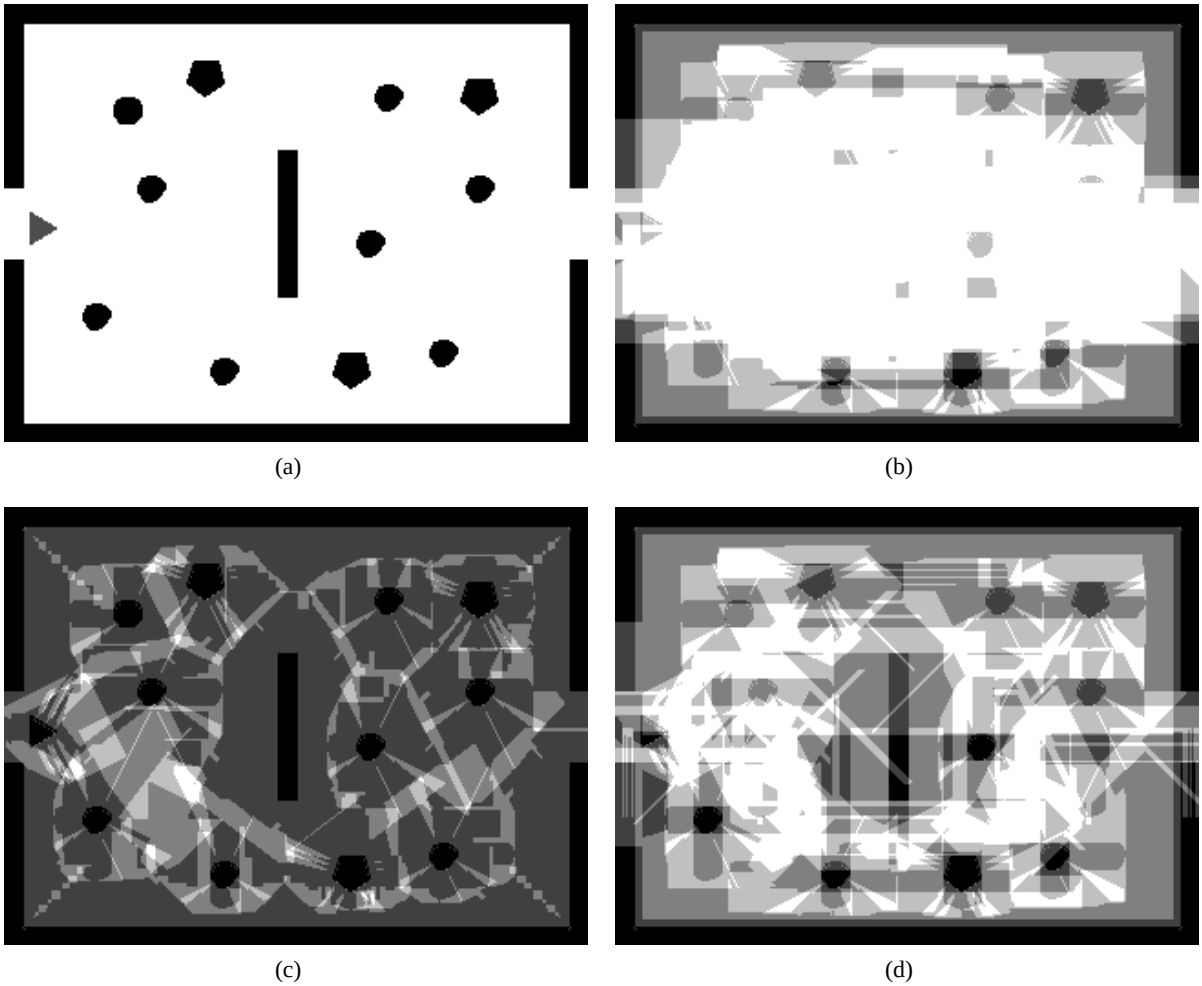

Figure 3. The number of updates used for several settings of the search parameters. Pixels with 0 updates are indicated in black, pixels with 4 or more updates in white. (a) An input image with 76800 pixels, 13942 object pixels and 1725 border pixels. (b) Too little searching resulting in 290771 updates and an execution time of 5.7 ms. (c) Too much searching resulting in 86487 updates and 8.4 ms execution time. (d) Optimal searching resulting in 179373 updates and 4.5 ms execution time.

3. EDT4, the EDT of Shih and Liu[8], which uses 4 scans with a 3x3 neighborhood over the image. The scans alone produce an approximated ED in the sense that for most pixels the obtained distance is correct but sometimes it is a bit to high. This is due to the fact[12] that the tiles of the Voronoi diagram are not always connected sets on a discrete lattice. This is shown in Figure 5, the pixel above the arrow is closer to object 2 than to objects 1 and 3, but all its 8-connected neighbors are closer to objects 1 or 3. Shih and Liu[8] provide a method to detect these situations and to correct the distance. We did not implement this correction as FEED is already faster.

4. EDT2, the EDT from Shih and Wu[10], which uses 2 scans with a 3x3 neighborhood over the image. The authors claim that this produces the exact ED, however this was not reproduced by our implementation of their method.

As described in the previous section, FEED can produce an exact representation of the ED by using the square of the ED in its matrix M. But it can also use less exact representations, like floating point or truncated integer, in M and its final result. As these representations are often used in the further processing chain, using them in FEED avoids a format conversion, which is costly because of the square root operation. Therefore, for this comparison, two different output formats were chosen: EDs as single precision (32 bit) floating point values and EDs truncated to 32 bit integer values. Moreover, the methods used to compare FEED with, were adapted to obtain these output formats and internal and

intermediate formats were chosen to obtain the fastest execution time while reaching the required accuracy.

Two sets of 10 images each were used for this comparison. One set with a size of 320x240 pixels as shown in Figure 3(a) with the triangular object on the left translated and rotated over the image. The other set had a size of 640x480 pixels also with a triangular object translated and rotated over the image; an example is shown in Figure 4. In these images white denotes a background pixel and non-white denotes an object pixel.

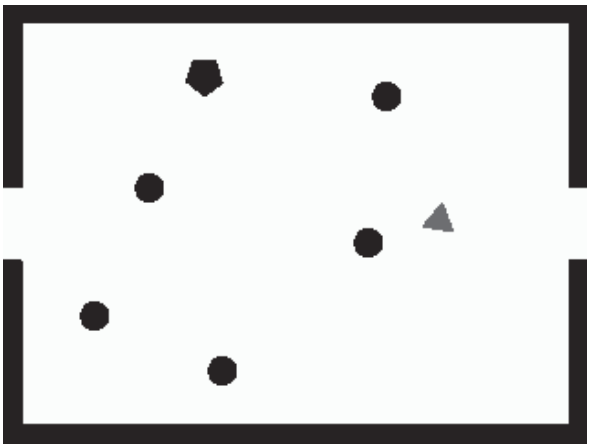

Figure 4. A large input image of size 640x480 with 47811 objects pixels of which 2654 are border pixels.

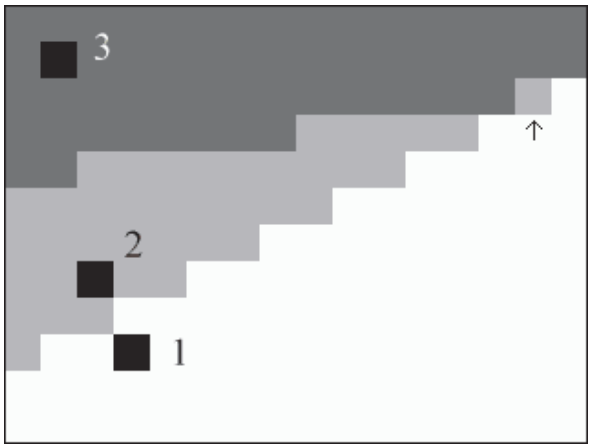

Figure 5. A test image with 3 objects. The set of pixels which are closer to object 2 than to the other objects is a disconnected set. Hence the ED can not exactly be determined using a scan which uses only local information.

format	truncated integer						floating point					
images	320 x 240			640 x 480			320 x 240			640 x 480		
method	time	errors		time	errors		time	errors		time	errors	
	(ms)	max.#	%	(ms)	max.#	%	(ms)	max.#	%	(ms)	max.#	%
FEED	4.5	0	0.0	17.3	0	0.0	5.2	0.0	0.0	18.5	1.0	0.0
EDT2	8.4	2	0.5	36.5	5	3.1	8.9	2.0	1.2	37.3	4.2	5.2
EDT4	14.5	2	0.3	62.1	3	1.3	14.3	1.5	0.6	61.6	2.9	2.6
CH3,4	2.2	3	22.2	11.3	6	36.6	3.4	2.5	41.4	16.3	6.3	44.6
CH1,1	1.3	14	42.0	7.2	45	44.7	2.1	13.9	42.0	10.2	44.8	44.7

Table 1. Timing (in ms) and accuracy results. The maximum error in units of pixels (max.#) and in percentage of pixels (%), which have received a different distance than the ED.

In Table 1 timing and accuracy results are presented using the hardware and software configuration as described in Section 2.2. Timings are given in ms. The maximum error is provided in units of pixels (max.#). In addition, the percentage of pixels, which have received a different distance than the ED (%) is provided. The values are averages over the images in each image set. For reliable timing results, each method was repeated a number of times for each image, such that the time per image per method was about 4 seconds. The reproducibility of the given time due to other processes running on the computer is better than 0.1 ms.

FEED is about a factor 2 faster than EDT2, which in turn is a factor 1.6 faster than EDT4. For EDT2 and EDT2 a few percent of the pixels have the wrong value after the scans, but this could be corrected by the method described by Shih and Liu[8]. These results confirm the earlier results presented by Schouten and Van den Broek[11]. More than for EDT2 and EDT4, the execution time of FEED depends on the content of the image. For instance, on random dot images there will be a lot of border pixels usually with no adjacent object pixels to provide a large cut on the number op pixels to update. Hence, FEED will be slower than EDT2 under certain filling conditions of the random dots. But we can state that FEED is the fastest exact EDT method for object like images on a sequential machine.

FEED compares favorable with the chamfer methods. It is only a factor 2 to 3 slower than the city-block distance and a factor 1 to 2 than the chamfer 3,4 distance. Regarding space requirements in addition to the output matrix, FEED uses a pre calculated matrix for EDs having the size of the image. EDT2 and EDT4 use during their operation two auxiliary matrices having the size of the image to store pixel coordinates. Hence, FEED uses much less memory capacity than EDT2 and EDT4.

4. ROBOT NAVIGATION

Distance maps, such as generated by FEED, can be applied in a range of settings, either by itself or as an important intermediate or auxiliary method in applications; e.g., robot navigation[13], trajectory planning[14], skeletonization[15], Voronoi tessellations[16], fMRI data analysis[17], neuromorphometry[18], volume rendering, reconstruction of surface normals, and penetration of distances for applications in haptics and physics-based modeling[19], Bouligand-Minkowsky fractal dimension[20], and Watershed algorithms[21].

Where EDTs can be applied in a range of settings, our primary focus lays in a general applicable algorithm, not in a specific application. However, a field of application needed to be chosen to illustrate and test EDTs working in a varying, preferably controlled, environment. As field of application we have chosen robot navigation, since it requires analysis of video sequencing.

Searching for shortest paths on surfaces with stationary obstacles is a classical problem in robot navigation. Solutions to the problems are based on computational geometry methods[22], differential geometry and hybrid techniques[23, 24], as well as graph search based algorithms[25]. In practice, most approaches are based on heuristic approaches and provide 'a' path rather than an optimal one. In contrast, DT can be used in finding the optimal path of a robot in the presence of both stationary and moving obstacles.

In order to experiment with such settings, we developed a virtual, dynamic robot navigation environment. Initially we wanted to make video-streams of a real robot moving in a certain environment. However, to tackle the potential problems with real world video sequencing, we decided to create a binary stream directly by animation using Macromedia®Flash. This provides us with a fully controlled environment and its video sequences. Moreover, using animation has the following advantages:

- By using layers one can separate the background from the moving objects.
- The ease of creating moving objects: drawing a line in a layer. Given a number of frames, Flash will generate the different frames with the object shifted along the path (e.g., see the path drawn in Figure 6). In addition, rotation of the object can be specified.
- Flash uses vector graphics. Therefore, it is relatively easy to change the animations.
- Most animation programs interpolate color values at edges of objects, with the export of bitmap images. This results in a loss of object indexation. In contrast, Flash has the ability to preserve the color map and thus the object indexation, with the export of its animations to a set of gif images.

To put it in a nutshell, the utilization of an animation environment is preferred since it can be used to generate a controlled experimentation environment and Flash, in particular, was chosen for its ease in use and its advantages of a more technical nature. Therefore, Macromedia®Flash was used to rapidly generate binary video streams of moving objects between stationary objects in a controlled manner.

For this paper two sets of generated video sequences are used. One is a sequence of 60 frames of 320x240 pixels, one of them is shown in Figure 3(a). There are 12 fixed objects inside a field whose border had two openings. Note that in the FEED algorithm the border pixels are also object pixels. The moving object is a triangle which rotated such that one of the corners always points to the direction of movement. Of the 76800 pixels there are about (because of the rotation) 13942 object pixels of which 1725 are border pixels. The other sequence consists of 120 frames of 640x480 pixels with 7 fixed objects in the field and with the moving object also being a triangle. An example is shown in Figure 4. Of the 307200 pixels there are about 47811 object pixels of which 2654 are border pixels.

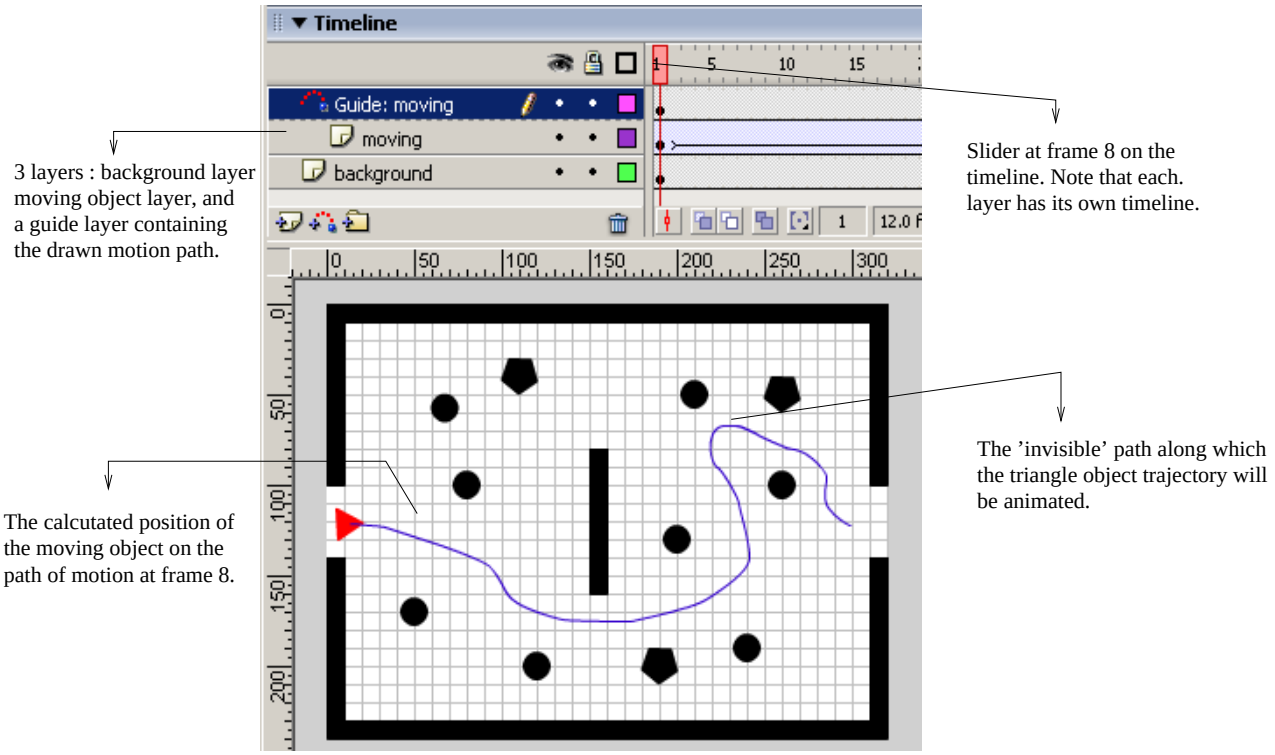

Figure 6. The dynamic test environment for robot navigation purposes with its most important compounds are labeled. The line drawn from left to right denotes the robot's trajectory.

The size were chosen because they are common sizes in camera's. The position of fixed objects were chosen to provide an average difficulty for FEED to find bisection lines (see Section 2). Note that these video sequences are used to compare performances, many more images were used to check the correct functioning of all the implementations.

The frames of the sequence were R,G,B images with white representing a background pixel, black a fixed object pixel, and red a moving object pixel. Additional moving objects can then be given other colors. The used test programs convert the frame to a byte image with white representing a background pixel and black a fixed object pixel while certain gray levels are used to denote moving object pixels. Note that the two sets of 10 images used in the previous section, were equidistant samples from the two video sequences. The sequences themselves are used in the next section.

5. TFEED FOR VIDEO

For tFEED we look to the situation of a sequence of frames (or images) showing fixed objects and one moving objects. Please not that methods for more moving objects can be developed based on the methods for one moving object. A simple possibility is to run the method for the moving object twice with different tests for determining the various kind of pixels. Further certain operations can then be combined or interleaved to gain some speed.

According to the definition of the DT, the distance maps (see Equation 1), for the fixed objects (D_{fixed}) and for the moving object (D_{moving}) can be calculated separately and then be combined using the minimum operator to obtain the distance map for the total frame:

$$D_{fixed+moving}(p) \;=\; min\{D_{fixed}(p)\;,\; D_{moving}(p)\} \tag{3}$$

To speedup the method, the min operator is applied during the process (instead of afterward) and so takes the influence of the moving objects into account. Subsequently, the following algorithm can be applied for the calculation of the exact

ED for a sequence of frames (tFEED):

$$
\begin{aligned}
&fixedFEED:\ D_{fixed}(p)\ =\ FEED\ with\ O_{fixed}\\
&movingFEED:\\
&\quad initialize\ D_{fixed+moving}(p)\ =\ D_{fixed}(p)\\
&\quad foreach\ q \in O_{moving}\\
&\qquad foreach\ p\\
&\qquad\quad update:\ D_{fixed+moving}(p)\ =\ min\{D_{fixed+moving}(p)\ ,\ ED(q,p)\},
\end{aligned}
\tag{4}
$$

where O_{fixed} and O_{moving} denote the object pixels of respectively the fixed and the moving objects.

Here FEED is the method as described in Sections 2. The same speedups are applied to movingFEED as were to FEED. In addition, the maximum distance d_{max} in $D_{fixed}(p)$ is determined because each border pixel in O_{moving} needs to feed its ED only up to a distance d_{max}. This is implemented by using d_{max} in the initialization of the maximum x and y values per quadrant, as defined in Section 2.

Note that this optimization can also be applied to FEED directly when the maximum distance in the image is somehow known a priori or when one is only interested in distances up to a certain maximum. For example, in a robot navigation problem where only smaller distances give navigation limitations.

The four methods to which we want to compare tFEED with and which are indicated in Section 3, were also adapted to the situation of frames with fixed objects and a moving object. First the original method is applied to an image with only the fixed objects producing D_{fixed}. Then, for each frame the bounding box of the moving object is determined. As we can assume an 8-connected moving object, otherwise it would be two objects, this can be done by searching for a pixel of the moving object using a refining sequence of coarse scans over the image. When a pixel is found, neighborhood scans are used to locate the bounding box. Next, the bounding box is extended in all directions by the maximum distance d_{max} occurring in D_{fixed} in order to define a rectangular area of the frame to which the original method is applied. This produces a local $D_{fixed+moving}$ of the rectangular area. Since the moving object has no influence outside this rectangular area, copying the local $D_{fixed+moving}$ into D_{fixed} results in $D_{fixed+moving}$ of the full frame.

As input video sequences, two sets of images were used, as described in Section 3. The input image was split in two images. One for the program parts handling the fixed objects with 0 indicating a fixed object pixels and 255 the background. The other for the program parts handling the moving object with 0 indicating a fixed object pixel, 127 indicating a pixel from the moving object, and 255 for the background.

format	truncated integer				floating point			
images	320 x 240		640 x 480		320 x 240		640 x 480	
method	full	partial	full	partial	full	partial	full	partial
(t)FEED	4.5	0.7	17.3	3.2	5.2	0.7	18.5	3.3
(t)EDT2	8.5	2.6	36.5	10.2	8.9	2.9	37.3	10.8
(t)EDT4	14.5	4.4	62.0	17.0	14.3	4.2	61.6	16.4
(t)CH3,4	2.2	1.0	11.3	4.0	3.4	1.5	16.3	5.3
(t)CH1,1	1.3	1.0	7.2	5.0	2.1	1.2	10.2	5.9

Table 2. Timing results in ms. "full" gives the time when applying the original DT methods on the full image. "partial" gives the time for the distance map per image using the adapted methods per frame.

Table 2 shows the obtained timing results using the same hardware and software configuration as described in Section 2.2 and the same measurement condition as given in Section 3. "full" provides the processing time (in ms), when applying the original DT methods on the full images, considering pixels from fixed and moving objects as the object pixels to which the distance of the other pixels are calculated. They should be equal to the times given in Table 1 within the stated repeatability of 0.1 ms and within a small variation caused by the fact that the measurements are over a different number of images. The times are indeed equal to within 0.1 ms. "partial" denotes the processing time (in ms) for obtaining the full distance map by generating a partial map for the moving object and combining that with the map for the fixed object. The timings (in ms) are the averages over the 60 images in the 320 x240 sequence and the 120 images of the 640x480 sequence.

The time needed to calculate $D_{fixed}(p)$ once per sequence is not shown, but is equal to the "full" time within a few tenths of a ms.

As shown in Table 2, tFEED is about a factor 6 faster than FEED. However, also for the other adapted methods a considerable speedup is observed but to a lesser extend than for tFEED. This results in tFEED being a factor 3 to 4 faster than the adapted EDT2 (tEDT2) method, for obtaining ED maps for a moving object between fixed objects. This holds even without taking the time needed for correcting the results of the latter into account. Hence, tFEED is the fastest available method for obtaining ED maps for video sequences.

Note that tFEED is even 20% to 50% faster than the adapted chamfer 3,4 (tCH3,4) and the adapted city-block distance (tCH1,1). In case that one wants these distances, tFEED can be adapted to provide them. This means that the bisection lines can only be used for the horizontal and vertical directions and that the matrix M (see Section 2) must be calculated to give the appropriate distance, or that the distance is directly calculated for each update as that might be faster. We did this for the city-block distance, obtaining 0.7 and 3.4 ms for the small and large images in the integer output format. Here we recalculated the distances as that was faster. ¿From Table 2 it can be seen that using tFEED to obtain the city-block distance is thus substantially faster than doing it directly.

Note that the chamfer 3,4 distance is often faster than the city-block distance. This can be explained by the fact that the city-block distance largely overestimates distances toward the 45° directions. This makes the needed rectangular area around the moving object larger for the city-block distance than for the chamfer 3,4 distance.

6. DISCUSSION

Schouten and Van den Broek[11] have developed a Fast Exact Euclidean Distance (FEED) transformation, starting from the inverse of the DT: each object pixel feeds its distance to all background pixels. The prohibitive computational cost of this naive implementation of traditional ED transformations, was tackled by three operations: restriction of both the number of object pixels and the number of background pixels taken into consideration as well as the pre-computation of the ED. In the first part of this paper, the original FEED algorithm is briefly discussed as well as additional speedups. In Section 3, FEED was compared with two other fast EDTs, showing that FEED is about a factor 2 faster than the fastest of them. It also shows than FEED compares favorable with chamfer distances, which provide an approximation of the ED.

In Schouten and Van den Broek[11] it was stated that it would be feasible to adapt FEED so that ED maps could be obtained of video sequences. With the introduction of timed FEED (tFEED) (in Section 5) this claim proved to be justified. In Section 5, the algorithms tested in Section 3 were also adapted to handle video sequences and compared with tFEED. The obtained speedup is larger for tFEED than for the other methods. This makes tFEED by a factor 3 to 4 the fastest method for obtaining ED maps for video sequences. It is then even 20% to 50% faster than the adapted chamfer 3,4 and the city-block distances.

Note that there are some further possibilities to increase the speed of tFEED and the adaptations of the other methods. The fixed objects could be encoded in a suitable form, for instance by run length encoding, to speed up the searching in $movingFEED$ (see Equation 4) and the location of the moving object in all methods. It can be expected that this speedups tFEED more than the other methods. Further, the position of the moving object can be predicted from previous frame, resulting in a faster location of it in average sense.

Last, note that FEED is essentially parallel so that parallel implementations can be developed if the need arises. Parallel implementations of the EDT are in particular required for real world applications involving large amounts of data and/or real-time execution. The use of parallel methods for DT was previously addressed in[4, 7], where a Single Instruction Multiple Data (SIMD) system was adopted. Alternatively, Multiple Instruction Multiple Data (MIMD) systems can be used, which have a superior performance when compared to SIMD counterparts.

The dynamic test environment (see Section 4) provides the means to generate, in principle every possible test environment for EDT methods. So, future releases of FEED and tFEED can be thoroughly tested varying several parameters (e.g., number, size, position of static and/or moving objects). Moreover, we aim in narrowing the gap between the conditions generated in the test environment and those in real world robot navigation. For example, slightly changing background can be included in the test environment, the effects of a light source can be mimicked (e.g., shadows can be generated), and the shape of objects can be manipulated through time.

So, with tFEED no approximations of ED transformations are needed due to its computational burden, but both Fast and Exact ED transformations can be done on video sequences containing objects. With that a new real time video processing algorithm is launched, important for many applications in image and video analysis.

REFERENCES

1. A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM* **13**(4), pp. 471–494, 1966.
2. A. Rosenfeld and J. L. Pfaltz, "Distance functions on digital pictures," *Pattern Recognition* **1**, pp. 33–61, 1968.
3. G. Borgefors, "Distance transformations in digital images," *Computer Vision, Graphics, and Image Processing: an international journal* **34**, pp. 344–371, 1986.
4. L. Chen and H. Y. H. Chuang, "An efficient algorithm for complete Euclidean distance transform on mesh-connected SIMD," *Parallel Computing* **21**, pp. 841–852, 1995.
5. D. Crookes and J. Brown, "I-BOL: A tool for image processing on transputers," *Transputer Applications and Systems* **93**, pp. 712–727, 1993.
6. Y. Lee, S. Horng, T. Kao, and Y. Chen, "Parallel computation of the Euclidean distance transform on themesh of trees and the hypercube computer," *Computer Vision and Image Understanding* **68**(1), pp. 109–119, 1997.
7. J. H. Takala and J. O. Viitanen, "Distance transform algorithm for Bit-Serial SIMD architectures," *Computer Vision and Image Understanding* **74**(2), pp. 150–161, 1999.
8. F. Y. Shih and J. J. Liu, "Size-invariant four-scan euclidean distance transformation," *Pattern Recognition* **31**(11), pp. 1761–1766, 1998.
9. L. F. Costa, "Multidimensional scale-space shape analysis," in *Proceedings of the International Workshop on Synthetic-Natural Hybrid Coding and Three Dimensional Imaging*, pp. 214–217, 2001.
10. F. Y. Shih and Y.-T. Wu, "Fast euclidean distance transformation in two scans using a 3 3 neighborhood," *Computer Vision and Image Understanding* **93**(2), pp. 195–205, 2004.
11. T. E. Schouten and E. L. van den Broek, "Fast Exact Euclidean Distance (FEED) Transformation," in *Proceedings of the 17th International Conference on Pattern Recognition (ICPR 2004)*, J. Kittler, M. Petrou, and M. Nixon, eds., **3**, pp. 594–597, IEEE Computer Society, (Cambridge, United Kingdom), 2004.
12. O. Cuisenaire and B. Macq, "Fast euclidean transformation by propagation using multiple neighborhoods," *Computer Vision and Image Understanding* **76**(2), pp. 163–172, 1999.
13. R. Kimmel, N. Kiryati, and A. M. Bruckstein, "Multivalued distance maps for motion planning on surfaces with moving obstacles," *IEEE Transactions on Robotics and Automation* **14**(3), pp. 427–436, 1998.
14. A. Zelinsky, "A mobile robot navigation exploration algorithm," *IEEE Transactions of Robotics and Automation* **8**(6), pp. 707–717, 1992.
15. R. Kimmel, D. Shaked, N. Kiryati, and A. M. Bruckstein, "Skeletonization via distance maps and level sets," *Computer Vision and Image Understanding* **62**(3), pp. 382–391, 1995.
16. W. Guan and S. Ma, "A list-processing approach to compute Voronoi diagrams and the Euclidean distance transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **20**(7), pp. 757–761, 1998.
17. Y. Lu, T. Jiang, and Y. Zang, "Region growing method for the analysis of functional mri data," *NeuroImage* **20**(1), pp. 455–465, 2003.
18. L. F. Costa, E. T. M. Manoel, F. Faucereau, J. van Pelt, and G. Ramakers, "A shape analysis framework for neuromorphometry," *Network: Computation in Neural Systems* **13**(3), pp. 283–310, 2002.
19. S. F. F. Gibson, "Calculating the distance map for binary sampled data," Tech. Rep. TR99-26, Mitsubishi Electric Research Laboratories, 1999.
20. L. F. Costa and R. M. C. Jr., *Shape Analysis and Classification*, CRC Press, 2001.
21. F. Meyer, "Topographic distance and watershed lines," *Signal Processing* **38**, pp. 113–125, 1994.
22. E. Wolfson and E. L. Schwartz, "Computing minimal distances on polyhedral surfaces," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **11**, pp. 1001–1005, 1989.
23. E. Adin and A. M. Bruckstein, "Navigation in a dynamic environment," cis #9101, Center of Intelligent Systems, Technion, Haifa, Israel, January 1991.
24. N. Kiryati and G. Székely, "Estimating shortest paths and minimal distances on digitized three dimensional surfaces," *Pattern Recognition* **26**(11), pp. 1623–1637, 1993.
25. J. C. Latombe, *Robot Motion Planning*, Boston, M.A.: Kluwer, 1991.