Real-time Video Vectorization Method for the Purpose of Surroundings Recognition and Mobile Robot Navigation

Received 4th October; accepted 16th November.

Marek Zachara, Ryszard Tadeusiewicz

Abstract:

In this paper a very simple method of the visual information interpretation for recognition and navigation purposes is presented and discussed. The proposed method consists of six steps: image acquisition, edge detection, fast edge vectorization using a high number of short preliminary vectors, aggregation of the preliminary vectors into the form of final vectors. The next stages of the visual information interpretation for recognition and navigation purposes will be description of the objects' shapes by means of the final vectors, object recognition and/or robot navigation on the base of comparison between actual shape description and templates memorized during the programming/training process, but they are not discussed in this paper. The main advantage of the proposed method is a simple and time-effective algorithm, which can be performed in real time also by a simple and cheap processor, working as a "brain" of the considered robot.

Keywords: *robot vision, surroundings recognition, real time navigation, edge detection, raster image vectorization*

1. Introduction and reasoning behind the robotic navigation

For decades, autonomous navigation has been one of the primary objectives of robotics. A few samples of surveys of the area available could be [4], [11] and [12]. This is no wonder considering the fact how large portion of our time is taken by simple tasks of going to a certain place to perform some tasks there. Delivering items from one place to another is probably the most common task performed by people all around the world.

1.1. Robot navigation and navigation with vision

Of course, mobile robots have been around for quite a while, almost from the beginning of robotics. Robots are well prepared to carry out strict orders (e.g. go forward 10 meters, then turn left 60 degrees and continue forward another 5 meters). Such precisely defined tasks, they can carry out very well, usually even much better than humans. Unfortunately this simple scheme of the mobile robot activity is possible only in stable environment, without any moving obstacles or topography changes. When we consider more casual order (e.g. take this bag, go outside turn left after a red bin and drop it at a pile near the bench there) we will come to the conclusion that this is near/almost impossible to be performed by any given robot available today.

To find out why the abovementioned order would be so difficult to carry out by a machine (while no human would have much problems with it) lets analyse it.

The first part ('take this bag') does not seem to pose real problem as long as the machine can understand that 'this' means an object of certain type that is placed most probably between it and the speaker. The second part ('go outside') is however probably the most difficult to understand. This requires that the machine understands human concept of buildings and places, some being external to another. Outside may, depending on context, mean: "go outside the room" or "go outside the building", possibly some other options as well. Assuming the machine can understand this concept, it must also understand how to get outside which usually includes finding a door and opening it. Then, to continue with carrying out the order, the machine would need to understand how 'a bin' look like, especially 'a red bin' and how to identify 'a pile'. Generally, to put it in a short phrase, the machine needs video-feedback and must understand what it sees. Full description of the meaning of term "Automatic understanding of the images" is given in book [17] and paper [18].

Now, one could say that there is no need for such high level understanding that it would be sufficient to enter a map of surrounding environment into the robot's memory and just identify a few important landmarks that will let the machine correctly calculate its current physical location [2], [3]. This is true such applications already exist. But they are not really feasible for general use because of several reasons. One is the amount of work and skills required to produce such map and configure it with the machine. The second issue is the inflexibility of such map - whenever the environment changes (e.g. a path is blocked or a landmark is removed) it has to be manually updated to reflect the changes. Last, but not least, such maps are only useable for the specific place, and do not allow the robot to be moved to e.g. another office.

1.2. Object recognition

Therefore, to construct a versatile robot, able to navigate in common environment, it seems necessary to introduce a method that will let the machine understand its surroundings - and this means correctly recognize objects and their relative placement in the observed area. A good starting point for investigating object modelling could be [1]. The key to the success is the ability to identify these objects regardless of their observed size and position. This ability is usually referred to as scale and view port invariance, and has been drawing attention of researches for many years now [8], [15]. To add difficulty to the task, the observed objects can be partially occluded and yet the recognition process must be able to

deal with them. This usually involves analysing the scene from different angles [6].

The current approach to object classification and scene understanding usually utilizes a scene segmentation (with a good comparison of recent methods available in [14]), followed by an attempt to reconstruct the shape of the observed objects [7].

1.3. Considerations for the processing of video data

Having decided on the desirability of the vision based on navigation it is now time to look for methods that could help get us closer to this goal. The primary advantage, and yet a major problem is the amount of data available. Video data streams are extremely large compared to other sensor inputs. The number of distinguishable points (i.e. pixels) in each data frame can mount up to a million or more and the real-time processing requires to process at the very least a few frames per second. So there are a few megabytes per second of data to handle. When we consider the task of analysing the picture pixel-by-pixel, comparing them to the past data and the other parts of the picture and reconstructing the objects present in the scene, this is the task that will exceed the capacity of CPUs for yet many years to come.

Therefore, to approach the target we set, which is autonomous navigation, the stream of processed data must be reduced to the level that will allow effective computing.

One of such approaches, that seems attractive is the vectorization; i.e. transformation of the observed scene into a set of discrete lines, and then using these lines to identify the objects they represent.

Actually, the vectorized models are already quite commonly used for computer graphics and animation. Vector models are used for rendering, giving us movies and computer games. Of course it's much easier to build a raster type image from vector objects and textures, than guessing the spatial model of the objects from the observed scene. Nevertheless, if this is achieved, it would help greatly in getting us closer to the goal - which is the ability to identify correctly all objects and their location within the observed scene.

Table 1. Vector versus raster images comparison (for the purpose of recognition).

	raster images	**vector images**
object shape	hard to determine	easy to determine
object textures	present	usually removed
detail level	high	usually small details are removed/omitted,
amount of data	large	small

It must be mentioned, that the vectorization of raster images is not a novel subject. It has been used widely for various purposes, especially for storing maps, blueprints and technical documentation that was only available in paper form. Vector images are scalable and require less space than their raster equivalents, it is therefore desirable to store documents in the vector form if possible. However the algorithms that have been developed for such uses are focused on quality of the representation while the speed of processing is not an important factor.

A vectorization of an image done in a few seconds is considered as a very good result while some algorithms tend to take minutes to complete. This is acceptable for an off-line document handling, but by no means is suitable for analysing of a video stream. While processing the video stream, the quality of a single vectorization process has to be sacrificed in favour of its speed. Because the usual video processing involves a series of frames that tend to be very similar one to the other, comparing the results from several consecutive operations can compensate the relatively lower quality of a single vectorization process. Then, by using statistical means a very good representation of the reality can be achieved.

1.4. The vectorization approach

Having decided on the benefits of vector approach to video processing, we have put that idea to a test. Our long-term goal is to build a robot that will be able to navigate in an unknown environment and identify the objects around. Of course, identifying objects may not always be possible, due to known issues like occlusion, therefore the robot is supposed to move around, trying to get a view on the objects that will let it match them with the internal database and allow recognition with an acceptable level of certainty.

The whole recognition process has been split into the following major phases:

- vectorization of the retrieved video stream
- shape extraction and matching with database
- move action

During the vectorization phase each incoming frame is translated into a set of vectors that represent the edges found in the frame. These vectors are then grouped into shapes that are matched against definitions of objects known to the robot (stored in its local database). The matching algorithm then has to decide whether some objects have been recognized with acceptable certainty and identify objects (or parts of the edges) that the robot is not able to identify. At the last stage the robot is supposed to take an action by moving around to get a better view on the objects that were not yet identified and in the end to be able to reconstruct a spatial representation of the observed scene.

For testing of the scenario described above, we have built a simple mobile platform with a camera, connected via radio link to the backing machine which process the incoming data and controls the robot's movements. A photo of this device is presented below in Figure 1:

Fig. 1. The mobile robot used for experiments.

This paper will present the first phase of the proposed method; i.e. the fast vectorization algorithm that can be used to detect shapes - and ultimately to reconstruct a virtual model of the observed scene.

2. The vectorization process

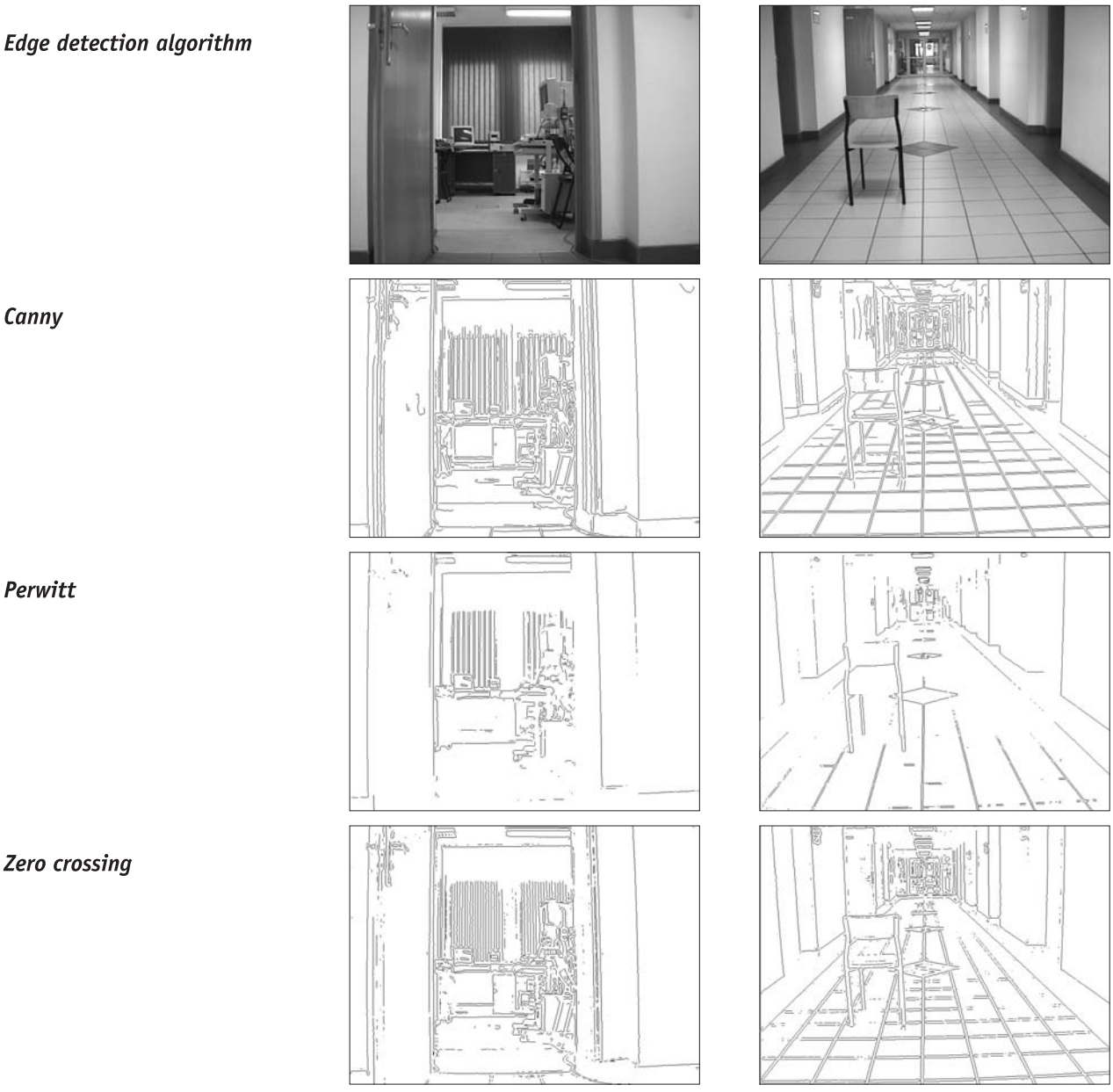
As mentioned before, the process of transforming raster data into a set of vectors must be as fast as possible to keep up with a real-time performance requirements. To facilitate that, the process must be streamlined, preferably based on single-pass algorithms at the stages when large quantity of data is processed. There are also a few arbitrary constrains introduced. These are related to the type of data we are extracting from the image. Because of its nature, vectorized graphics are best suited to represent the boundaries (also referred to as edges) of the objects. Although the information about texture may also provide valuable data, the scope of this paper is focused on the exploitation of the objects shapes and therefore is limited to processing the edges observed in the scene. Therefore the whole process of vectorization is divided into three stages: edge detection, vector creation and vector merging. These will be described in detail below.

2.1. Edge detection

This stage of the processing is responsible for extracting the edges from the image. This is a task that has been given a considerable attention by researches already, as edge detection has been used for other various tasks in image processing and recognition.

For the purpose of proposed vectorization the Canny algorithm has been selected as it provides reasonable output, suitable for further processing with a relatively low number of artefacts in the resultant picture and continuous lines. For a comparison, a few algorithms with the results of their run have been presented in the Figure 2 below. Further information and performance comparison of edge detection algorithms can be found in [9].

Fig. 2. Sample results of a few edge-detector algorithms.



As it can be observed in Figure 2, the Canny algorithm produces a good quality output, with least edges undetected and lower number of 'artefacts'. The number of artefacts (or a 'noise') in the resulting picture has much impact on the performance of the vectorization process, so the processing parameters must be adjusted to limit their number, even at the cost of some real edges being omitted in the resultant image. This image is then used for the second phase of the process.

2.2. Vector creation

Creating vectors even from an image that consists of lines is not a straightforward task. This is because there is no deterministic algorithm that would describe how to translate set of pixels into a set of vectors (on the contrary, translating set of vectors into a set of pixels is a simple, deterministic task).

The same set of pixels can be translated into many equivalent sets of vectors, but of these the most attractive for further processing are these that contain less in number, but longer vectors. Unfortunately, creating quality, long vectors requires a lot of computations, especially when taking into the account the fact that lines on the processed image can be slightly distorted or damaged.

To counter this problem, a two-phase method has been introduced. In the first phase, the image is searched for straight, continuous lines. This constrain allows usage of a single pass algorithm which is an important advantage. In the second stage, the algorithm analyses the vectors found and tries to merge them together to form longer lines.

The algorithm that creates initial shorter vectors starts by making a copy of the initial image. The purpose of this copy is to hold the information about pixels that has not yet been assigned to any vector. Any vector, that will be find later on, will have all the pictures that constitute it erased from this image copy.

The algorithm then starts its cyclic routine with locating in the image copy the first available pixel in the 'on' state. This pixel will be a base for vector search. An initial vector is constructed with the following parameters:

$$\vec{v}_0 = \begin{bmatrix} x_0, y_0 \\ x_0, y_0 \end{bmatrix} \tag{1}$$

As a next step, the surrounding of the picture is searched for other 'lit' pixels, and if found, a number of hypothetical vectors $\vec{v}_h : \vec{v}_{h0},...\vec{v}_{hn}$ are constructed. These all vectors start in the base point but have different ending points.

The algorithm then repeats recursively the search, trying to extend each of these vectors at their ends. For each path, a complete tracking path is kept, described by all merged pixels (2), with new pixels being added at the end. Each vector $\vec{v}_{hn}$ is constructed with a number of pixels; therefore its construction path $p_{\vec{v}_{hn}}$ will consist of a number (i) of pixels:

$$p_{\vec{v}_{hn}} = [(x_0, y_0),(x_{n1}, y_{n1}),...,(x_{ni}, y_{ni})] \tag{2}$$

Each time the new point is added to the hypothetical vector, there is a check made against all pixels that have been added so far - to find out if they altogether still constitute an acceptably straight line. This is done by measuring the distance between the hypothetical vector and each pixel in the tracking path (3)

$$\forall k = 0...i \tag{3}$$

$$d_k = \begin{Bmatrix} |y_0 - y_i x_0 - x_i * (p_{xk} - p_{x0}) + p_{y0} - \\ - p_{yk}|; if |y_0 - y_i x_0 - x_i| < 1 \\ |x_0 - x_i y_0 - y_i * (p_{yk} - p_{y0}) + p_{x0} - \\ - p_{xk}|; if |y_0 - y_i x_0 - x_i| \geq 1 \end{Bmatrix}$$

If the distance between the created hypothetical vector and one of the pixels constituting it is larger than the predefined constrain, the last pixel added is removed and the building of this particular vector ends at this point. Figure 3 presents a sample pixel layout and the vectors that will be build (solid black line) together with vectors that will be rejected due to too large distance to one of its pixels (dashed line):

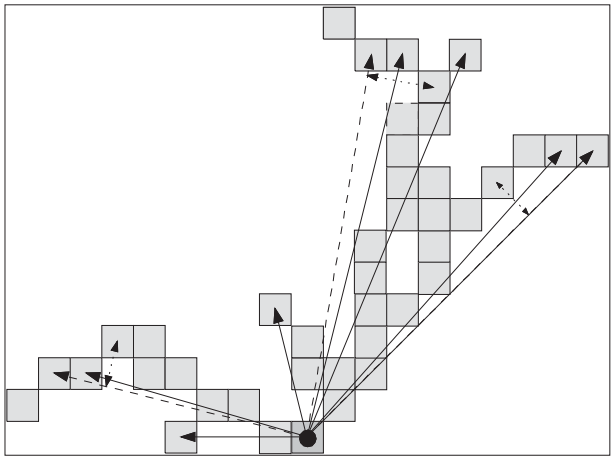


Fig. 3. Sample results of vector construction with the described algorithm.

The value of the maximum allowed distance (d_{max}) between the constructed vector and the pixels that constitute it has a great impact on the vectorization process. Depending on its value, different results can be expected (based on experimental results):

Table 2. Results of using different max distance parameter.

max distance d_{max}	time required for vectorization	vectors characteristic
< 0,5	short	short, usually a few pixels only
0,5-0,8	average	average length
> 0,8	long, strongly increasing with d_{max}	long, over 10 pixels

To achieve a short processing time, while not sacrificing the quality of the results, is the best is to adjust dynamically the d_{max} value while constructing the vector. Starting with a small value of 0.5 and then increasing it to 0,85 as the vector length grows over 4 pixels. As a final step, an attempt is also made to extend the vector in the direction of the starting point. This is done by repeating the procedure described above, but with the ending and staring points swapped:

All pixels that belong to the tracking path of the constructed vector are then removed from the copy of the image, in order to prevent them from being used as a starting point for another vector search (because this will most likely result in the same vector being constructed again).

2.3. Vector aggregation

As it was mentioned before, the first phase results in creating a large number of relatively short vectors. Because of the nature of raster images and edge detection algorithms, artefacts like missing or misplaced pixels are to be expected. The algorithm described above does not handle misplaced pixels well, it stops on missing pixels, so naturally the result of this stage of processing is far from what one would expect from a quality vectorization.

To cope with missing pixels, the algorithm would need to extend the search for the constructed vector, which would lead to extensive computation increase. Instead another approach has been taken - the short vectors are merged together in the second stage of the vectorization. This solution generates much less computational overhead, as it requires only to perform computations on a (usually) few thousands of records that were the result of the first stage. The algorithm that will be presented also provides other means to further narrow down the number of vectors processed, thus greatly improving the overall performance.

The aggregation algorithm picks up vectors from the source list and tries to merge or join them with another one on the list. To reduce the number of computations, initially the vectors in the source list are transformed into another form (4).

$$\vec{v} = \begin{bmatrix} x_0, y_0 \\ x_e, y_e \end{bmatrix} \Rightarrow [x_c, y_c, \alpha, l] \tag{4}$$

So vectors are now defined by their centres, angle and length (instead of position of beginning and ending pixels). The angle is calculated against the vertical line going through the centre of the vector (as in Figure 4).

The algorithm picks up the vectors from the initial lists, starting with the longest vector available. This approach assures better quality of the resultant vector, as longest ones are most likely to represent real edges of the objects in the image and not the artefacts. Then the aggregation is performed by two means: joining another vector (which results in extension of the original one) or by merging another vector which is small enough but matches the constructed vector position and direction so well that it can be discarded without real loss of information. Both of these processes start from a common stage, which is defining the search range for the vector v_0 as explained in Figure 4:

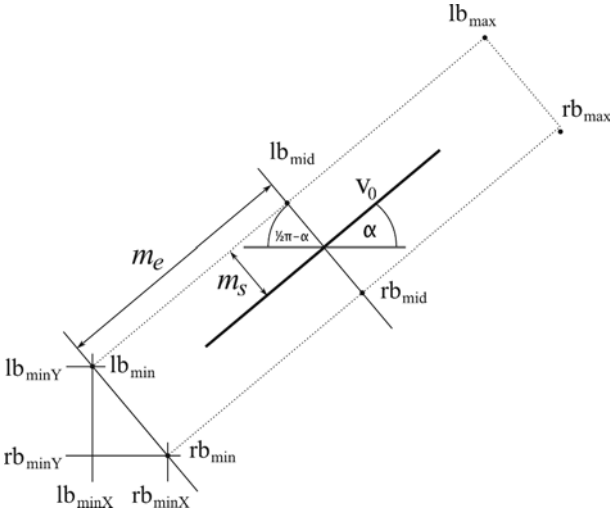


Fig. 4. Search region for vector aggregation.

As marked on the picture, the area used for searching of neighbourhood vectors is defined as a rectangle centred with the analysed vector, with size based on two parameters:

m_s - the distance orthogonal to the vector axis, usually the size of a few pixels and,

m_e - which defines the maximum margin aligned with the vector axis. This parameter has a value calculated as half of the length of the analysed vector plus a few pixels.

Because (as it was stated before) the aggregation of the vectors is done by descending order on a length-sorted table of initial vectors, there is a guarantee that the joined vector will not be longer than the vector v_0.

For the performance reasons, two additional sorted sets are created before starting the aggregation procedure. These are lists of source vectors sorted by the position of their centres (x and y respectively). By utilizing these two lists, the algorithm can quickly locate the potential candidates that fit into the defined region.

The algorithm now gets into the stage when it looks for best candidates to extend the analysed vector. For this purpose, a quality function Q is constructed which describes how well the proposed candidate matches the analysed vector. There are three key factors taken into account when calculating this function:

- The distance between the end points of the candidate and analysed vector. The closer they are to each other, the higher chance they in reality were parts of the same line, so the Q function is adjusted accordingly.
- The length of the vector candidate. It's better if two longer vectors can be joined so this must be reflected in Q function.
- The difference in angles between vectors. This relation actually depends on the length of the vector candidate. A two pixel vector can be even orthogonal to the analysed vector and yet be joined without much error, but the longer the vector candidate, the more the angles (between the vector and the horizontal line) must match each other.

As a result, the following function Q_d to measure quality of matching has been proposed (5). The function value is proportional to the length of the vector candidate and the distance between the ends of the vectors, but includes an exponential negative punishment for the difference in vectors' angles. This results in longer and close to each other vectors being promoted for merge, but with a very strong emphasis on forming a common line.

$$Q_d = l_q - a * \left| v_{dist} - \frac{l_q}{2} - \frac{l_0}{2} \right| - b * 2^{(l_0 - 1) * \left(|\alpha_v - \alpha_0| + |\alpha_q - \alpha_0| \right)} \tag{5}$$

where:

l_q, l_0 length of the vector candidate (v_q) and base (v_0) vector,
a_q, a_0 angle of the v_c and v_0,
v_{dist} distance between the centres of the vectors,
a_v angle of the line that goes through both vector centres,
a, b experimentally adjusted parameters.

After the value of Q is calculated for all the vector candidates, the one with the highest value is chosen for join, but only if the value of Q exceeds a predefined value $Qmax$. This last constrain is necessary to eliminate join actions on formally best matching pair if the matching quality is so low that they should not really be joined.

The joining of the vectors is performed by creating a new vector with end points in the end points of the joined vectors that are not close to each other. Both vectors that have been joined are in turn removed from the list.

As a last step of the aggregation process, the algorithm looks for vectors that could be merged with the newly constructed one. These are short vectors that are in line with the constructed vector and are in most part covered by it.

To find out if the vector can be matched, again a quality function is calculated. First, the distance between the centres of the constructed vector and the candidate for merge are calculated as noted in (6):

$$m_d = \sqrt{\left(v_{0x} - v_{tx}\right)^2 + \left(v_{0y} - v_{ty}\right)^2} * \sin(\beta) \tag{6}$$

then, the Q_m function which is de-facto the maximum distance between the vectors (7):

$$Q_m = m_d + \left| \frac{1}{2} * l_t * \sin(\alpha) \right| \tag{7}$$

Finally, if the value of Q_m is less than some predefined value (e.g. 2 pixels), the vector is merged which means it's just deleted from the list (as it is assumed that it does not carry valuable new information).

3. Shape estimation

All steps of the image analysis and processing, described in previous chapters, leads to estimation of shapes for all-important objects presented on the processed images. This is currently a work in progress, so just a general overview will be given here. On the contrary to object recognition based on object features (as for example proposed in [13] and [16]), vector based images are better suited for shape analysis. This is obvious considering the fact vectors composing an image usually represent objects edges.

A technical difficulty arises however when analysing data from consecutive frames. Even if the camera and the scene is still, the set of vectors generated for each frame will be different due to the noise and slight differences in illumination. The same line may be represented by different number of vectors on two consecutive frames.

This issue can be overcome by utilizing probabilistic methods, which calculate the probability of certain line presence in certain place over a range of data from consecutive frames. Similar techniques are already used [10], and there are several methods (e.g. particle filtering [15]) that are widely used. The algorithms available are however designed so far for raster data, thus the vector approach needs its own set of methods that are currently being implemented.

The next step planned is to calculate the relative displacement of the lines present in the image, and by this retrieve information about relative movement of the camera against the scene. This information can also be used for a second pass of probabilistic model update, increasing the certainty of lines estimation.

Finally lines with their displacement when compared against the movement of the camera will allow for the reconstruction of the third dimension and can be used to match with the objects stored in the database to identify them.

4. Concluding remarks

The presented method of fast vectorization has been tested on some representative examples and it was proven that the proposed algorithm works. Figures 5 and 6 demonstrate the results of the proposed vectorization algorithm.

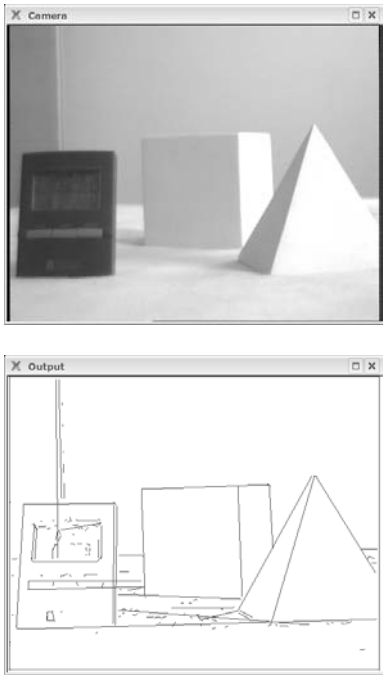


Fig. 5. Example of the vectorization of an artificial scene.

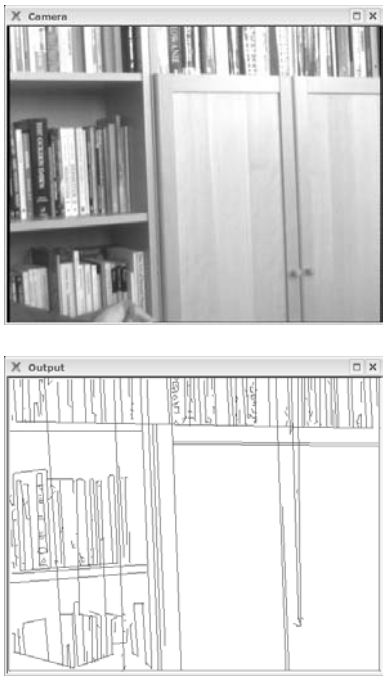


Fig. 6. Vectorization results of a real scene.

The algorithm was proven to be effective, with the results of only 141 vectors created for the scene in Figure 5 and 345 vectors for the scene presented in Figure 6. Moreover, the algorithm has proven to be really fast in the real experiments. On a recent PC-type machine with a 2GHz processor the vectorization time of one frame (512x384 pixels) is less than 1 second. This is quite enough to achieve real-time actions of the mobile robot in a real environment of course when the robot does not move too fast. Considering the fact, that this is only a reference implementation, without optimisations, and a lot of overhead included for clarity and manageability purpose, it can be assumed, the performance results can be improved further. Moreover when all the algorithms mentioned above are tested and proved properly using software implementation described here, it will be possible to made hardware implementation of such algorithms using e.g. FPGA technology.

Right now however our priority is to improve the efficiency of the shape extraction and matching algorithm, which proved to be a bottleneck for real-time processing. We hope that we will be able to overcome them and publish the results of these shortly.

ACKNOWLEDGMENTS

This work was supported by the AGH University of Science and Technology under Grant No. 11.11.120.612.

AUTHORS

Marek Zachara* - AGH University of Science and Technology, al. Mickiewicza 30, Kraków, Poland, tel: +48-12-6173924, e-mail: mzachara@agh.edu.pl.

Ryszard Tadeusiewicz - Head of Chair of Automation at the AGH University of Science and Technology, al. Mickiewicza 30, Kraków, Poland, tel: +48-12-6172830, e-mail:rtad@agh.edu.pl.

* Corresponding author

References

[1] D. Burshka, D. Cobzas, Z. Dodds, G. Hager, M. Jagers, K. Yerex, "Recent methods for image based modeling and rendering". In: *IEEE Conference on Virtual Reality*, Baltimore Hotel, Los Angeles, 2003, pp. 55-66.

[2] A. J. Davison, *Mobile Robot Navigation Using Active Vision,* PhD thesis, Department of Engineering Science, University of Oxford, UK, 1998.

[3] A. J. Davison and D. W. Murray, "Simultaneous localization and map-building using active vision". In: *Proceedings of IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002, pp. 865-880.

[4] G. N. DeSouza and A. C. Kak, "Vision for mobile robot navigation: A survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2002, pp. 237-267.

[5] A. Doucet, C. Andrieu and S. Godsill, On Sequential Monte Carlo Sampling Methods for Bayesian Filtering, *Statistics and Computing*, vol. 10, no. 3, 2000, pp. 197-208.

[6] M. Han, T. Kanade, *Scene reconstruction from multiple uncalibrated views. Technical Report CMU-RI-TR-00-09*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2000.

[7] G. Elidan, G. Heitz, D. Kolle, "Learning object shape: From drawings to images". In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 2064-2071.

[8] I. Gordon, D. G. Lowe, "Scene modeling, recognition and tracing with invariant image features". In: *International Symposium on Mixed and Augmented Reality*, Arlington, VA, 2004, pp. 110-119.

[9] M. Heath, S. Sarkar, T. Sanocki, K.W. Bowyer, "A Robust Visual Method for Assessing the Relative Performance of Edge-Detection Algorithms", In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, 1997, pp. 1338-1359.

[10] Y. Jin and S. Geman. Context and hierarchy in a probabilistic image model. In: *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 2145-2152.

[11] J. Knight, *Towards Fully Autonomous Visual Navigation*. PhD thesis, Robotics Research Group, Department of Engineering Science, University of Oxford, UK, 2002.

[12] E. Malis, *Survey of vision-based robot control*, In: *Proceedings of European Naval Ship Design*, Captain Computer Forum, ENSIETA, Brest, France, 2002.

[13] K. Mikolajczyk, B. Leibe, B. Shiele, "Local features for object class recognition". In: *Proceedings of IEEE International Conference on Computer Vision*, vol. 2, 2005, pp. 1792-1799.

[14] C. Pantofaru, M. Hebert, *A comparison of image segmentation algorithms. Technical Report CMU-RI-TR-05-40*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[15] S. Se, D. G. Lowe, J. Little, "Vision-based mobile robot localization and mapping using scale-invariant features". In: *Proceedings of IEEE International Conference on Robotics and Automation*, Seoul, Korea, 2001, pp. 2051-2058.

[16] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, W. T. Freeman, "Discovering objects and their location in images". In: *Proceedings of IEEE International Conference on Computer Vision*, vol. 1, 2005, pp. 370-377.

[17] R. Tadeusiewicz, M.R. Ogiela, *Medical Image Understanding Technology*, Series: Studies in Fuzziness and Soft Computing, vol. 156, Springer-Verlag, Berlin Heidelberg New York, 2004.

[18] R. Tadeusiewicz, M.R. Ogiela, "Why Automatic Understanding?". In: B. Beliczynski *et al.* (Eds.): *ICANNGA 2007, Part II, Lecture Notes on Computer Science*, vol. 4432, Springer-Verlag, Berlin Heidelberg New York, 2007, pp. 477-491.