Gheorghe Lazea - Alin Mureşan - Emil Lupu *

ALGORITMY SPRACOVANIA OBRAZU VYUŽÍVANÉ V APLIKÁCIÁCH ROBOTIKY

IMAGE PROCESSING ALGORITHMS USED IN ROBOTICS APPLICATIONS

Štúdia predstavuje výsledky získané realizovaním algoritmov pre spracovanie obrazu (detekcia hrán, dokončovanie obrysov, vytvorenie kostry). Spolu s uplatnenými príkladmi je uvedených niekoľko klasických ako aj dva originálne algoritmy. Spracovanie obrazu pre aplikácie robotiky si vyžaduje rýchly čas spracovania na počítači. Je skúmaná možnosť zlepšenia času spracovania klasického algoritmu vytvárania kostry (Nacacheho a Shingalova metóda) využívajúceho vopred nacvičenú neurónovú sieť. Do úvahy sa berú dva prístupy: dvojvrstvová neurónová sieť a zhluk 12 jednovrstvových neurónových sietí. Je prezentovaná a okomentovaná dosiahnutá doba spracovania daného problému.

The paper presents the results obtained in implementing image-processing algorithms (edge detection, contour closing, skeletonizing). Several classical as well as two original algorithms are presented, together with sample images from the implementation. Image processing for robotic applications requests fast computation times. We investigate the possibility to improve the processing time of a classical skeletonizing algorithm (Nacache and Shingal method) using a previously trained neural network. Two approaches are considered: a two layers neural network and a cluster of 12 single layer neural network. The processing times obtained are presented and commented.

1. Introduction

During the last five years the main field of interest of the research group within the Robotics Laboratory at the Automation Department in Technical University of Cluj-Napoca was the development of a wheeled mobile robot for indoor use. The work done until now has been supported by several grants from the Romanian Ministry of Education (see [8]) the main achievements being a mobile platform actuated by DC motors (see [5] and [6]), a stepper motor actuated micro-robot, path planning programs (see [9]) and a low-cost, intelligent ultrasonic range finding sensor (see [7]). The experience gained through these projects has shown that the sensor system is perhaps the most important in building an autonomous, useful mobile robot. This opinion is also shared by other authors, ex. [10], [4], and has led us to the decision to incorporate vision systems in our future mobile robots. In the early times of mobile (and industrial) robot research, vision could not be used as a practical approach for sensor driven robots in real time (see ex. [13]) due to the lengthy processing of the huge amount of data yielded by the vision system. But with the advent of today's new computers this is now possible. Figure 1 briefly sketches a standard architecture for a vision driven (mobile) robot.

Our first steps in implementing this architecture were to develop the "Preprocessing" and, partly, the "Recognition" modules in the figure bellow. Some of the results obtained in edge detection, skeletonizing and objects localisation within 2D images, were presented in [11]. Further algorithms can be found in [15], [3], [12], [2] and [14].

2. Object localisation

In the sequel we present an original approach to object localisation for specific objects (objects with holes - see Fig. 2). Such objects can be found quite frequently in assembly operations (peg in hole). The algorithms may also be applied to natural and artificial landmark recognition for mobile robot navigation. The image processing is done in two steps: first edge detection and contour closing followed by hole detection and localisation. We have chosen to work with binary (monochrome) bitmap images coded at byte (not bit) level. This increases the memory usage but considerably simplifies the algorithms and, consequently, yields faster processing.

2.1. Edge detection and contour closing

An edge is a boundary between two regions with relatively distinct grey levels [14]. The most used algorithms for edge detection employ local, gradient-based methods. The gradient can be computed using convolution products between a matrix defining the neighbourhood for each pixel and a gradient operator.

The edge detection and contour closing algorithm uses differential, local methods and consists of the following steps:

* **Prof. Dr. Eng. LAZEA, Ass. Alin MUREŞAN, Ass. Emil LUPU**
Automation Department, Faculty of Automation and Computer Science, TU of Cluj-Napoca, Daicoviciu street, no. 15, Cluj-Napoca, România,
Tel.: +40-64-199873, E-mail: Gheorghe.Lazea@aut.utcluj.ro; Alin.Mureşan@aut.utcluj.ro, Emil.Lupu@aut.utcluj.ro

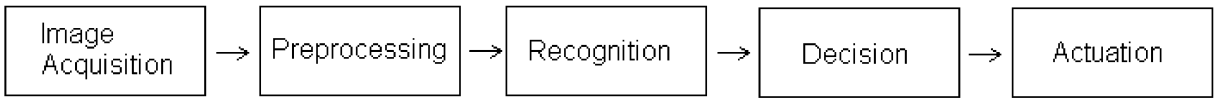

Fig. 1. Architecture for a vision-driven robot

a) Two threshold values are chosen: P1 - the minimum gradient value for a pixel for which this still is considered to belong to an edge, and P2 - the minimum grey level, such that the background grey level is bigger than P2. At this point is noteworthy mentioning that in bitmap files the standard coding is 255 for the white grey level and 0 for the black grey level. In this paper the images are presented inverted for readability reasons.
b) Using one of the following operators: Roberts with 4 or 9 points neighbourhood, Mero-Vassy, Sobel or Prewitt the gradients for each image pixel is computed (see [11] for details).
c) For each pixel a nine-point neighbourhood is built and the maximum grey level is computed.
d) Each point with the gradient bigger than P1 and having a neighbourhood with maximum grey level bigger than P2 is flagged as belonging to the edge (it gets the 255 value for the grey level).

This procedure assures reliable edge detection with strong rejection of false edges (see figure 3).

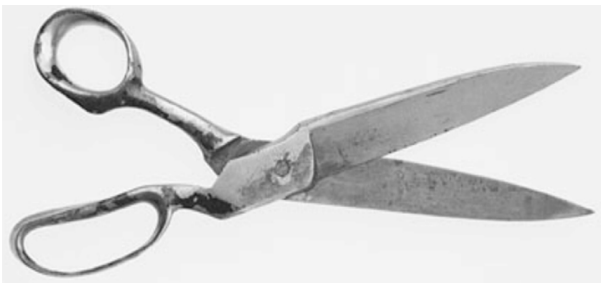

Fig. 2. Original test image

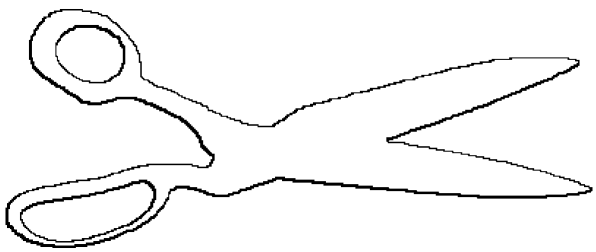

Fig. 3. Edge detection and contour closing

2.2. Hole detection and localisation

The algorithm consists of the following steps:

a) The image is scanned line by line until the first point belonging to an edge (255 grey level value) is reached. This point is being given the value of 254 for the grey level (the point belongs to an exterior contour) and the same value is written in a flag variable.
b) The image scanning continues until the next 255 level point is being hit. For this point the 5 points neighbourhood depicted in Fig. 4 is being built.

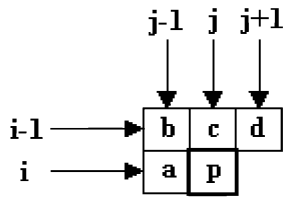

Fig. 4. 5 points neighbourhood

c) Compute max grey levels (a, b, c, d). If it is 0 this means the current point **p** is on a new contour. The flag is decremented and point **p** gets its value as the grey level. Otherwise the point p gets the value max(a, b, c, d).
d) For each of the points a, b, c, d with grey levels smaller than max(a, b, c, d) (namely x) the scanning is resumed from the point reached at step a) and each point with x grey level is being given the value max(a, b, c, d). All grey levels smaller than x are incremented. The flag is incremented.
e) Go to step b) until no point has a 255 value grey level.

At algorithm completion, the flag will yield the smallest grey level allocated, information used for finding the interior contours number. For example, if the flag equals 252, all the points with 254 grey level are exterior contour points, so there will be $254 - 252 = 2$ interior contours (that is, two holes).

The processing continues with the interior contours filling algorithm and mass centre computation (for details see [11]). The results obtained for the test image are presented in Fig. 5.

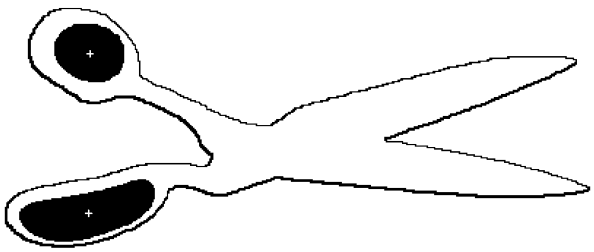

Fig. 5. Hole detection and localisation

3. Skeletonizing

The objective of skeletonizing is to reduce the representation of a region to a chain of single pixel width while preserving all

other relevant features [14]. Skeletonizing is also known as thinning and the image obtained can be interpreted as a Voronoi graph with applications in path planning.

The key point in using vision with robotics is the real time implementation [16]. This became possible only in the last few years based on the newer and faster processors. Faster algorithms are also a key issue. In the present paper we investigate several ways to implement a skeletonizing algorithm. A classical implementation is presented versus two other using neural networks. The idea is to introduce the heavy computations in the training stage obtaining a fast neural network on the exploitation stage.

3.1. Modified Nacache and Shinghal's Method

Naccache and Shingal [1984] proposed a skeletonizing algorithm. After several studies, we came to the conclusion that the original algorithm must be a little bit modified in order to obtain better results. First we study 8 neighbours of the interest pixel, as shown in Fig. 6:

n3	n2	N1
n4	p	N0
n5	n6	N7

Fig. 6. Notation for the neighbours of p

An edge point is flagged if it is not an endpoint or breakpoint, or if its deletion would cause excessive erosion. Comparing the 8 neighbourhood against the windows in Fig. 7 carries out the test of these conditions.

*		d	d	d	d	d			d	d	d
	p	d		p	d		p	*		p	
d	d	d	*		d	d			e	e	e

Fig. 7. The "" denotes a dark point, and d and e can be either dark or light*

Testing the 8 neighbors for identification of the above situations is done with simple boolean expressions:

$B_4 = n_0 \cdot (n_1 + n_2 + n_6 + n_7) \cdot (n_2 + \overline{n_3}) \cdot (\overline{n_5} + n_6)$
for left edge points;
$B_0 = n_4 \cdot (n_2 + n_3 + n_5 + n_6) \cdot (n_6 + \overline{n_7}) \cdot (\overline{n_1} + n_2)$
for right edge points;
$B_2 = n_6 \cdot (n_0 + n_4 + n_5 + n_7) \cdot (n_0 + \overline{n_1}) \cdot (\overline{n_3} + n_4)$
for top edge points;
$B_6 = n_2 \cdot (n_0 + n_1 + n_3 + n_4) \cdot (n_4 + \overline{n_5}) \cdot (\overline{n_7} + n_0)$
for bottom edge points.

where a dark pixel is "TRUE" and a light pixel is "FALSE". For the thinning algorithm, we use two matrices (the first is the original image, the second is a mirror of the image). The algorithm is:

1. scan the original image (along the rows or the columns) and calculate the expressions B_0, B_2, B_4, B_6 if the pixel is dark;
2. if one or two of the expressions are "FALSE" then the pixel in the second matrix is set to "FALSE", else is set to "TRUE"
3. the second matrix becomes the original image;
4. if no new edge points were flagged during the scan, the algorithm stops.

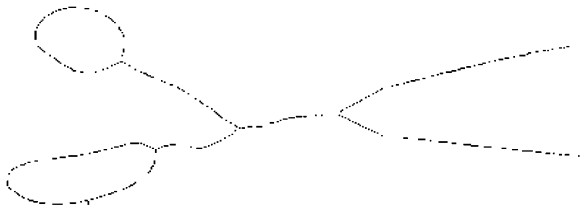

Fig. 8. Skeleton obtained using Naccache method

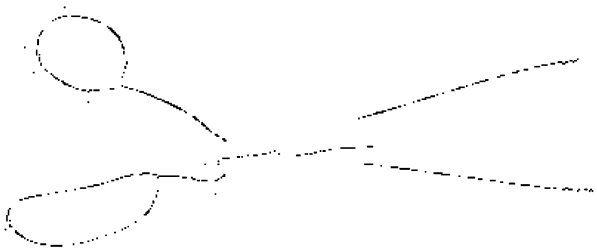

Fig. 9. Skeleton obtained using original method

We have also developed and implemented an original skeletonizing method for binary images. It consists of successive scanning of the original image on several directions and marking the centre point of the object segment as belonging to the skeleton (see Fig. 9).

3.2. Two layers neural network

Let us consider a classical feed-forward neural network using back-propagation for learning. We want to design this neural network to be able to implement the algorithm presented in 4.1. The natural solution is to choose a 2 layers neural network (figure 10). The input vector contains the eight neighbours of the pixel p and the output vector contains the desired value of p in order to skeletonize the object. Successive tests have shown that a number of eight hidden neurones is optimal.

The training vectors were obtained using the modified Nacache and Shinghal method.

In the training stage, we used a sigmoid activation function:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad (1)$$

In the processing stage, for minimising the processing time, the function in equation (1) was replaced by that in equation (2), but the recognition rate was maintained at 100 %.

$$f(x) = \begin{cases} 0.166 \cdot x + 0.5 & , if -3 \le x \le 3 \\ 1 & , if\ x > 3 \\ 0 & , if\ x < -3 \end{cases} \quad (2)$$

Table 1 presents the weights and biases obtained after training. The output value is given by:

$$out = f\left(\sum_{j=0}^{7} W_{1j} \cdot out_j + B\right)$$

$$out_j = f\left(\sum_{i=0}^{7} W_{ji} \cdot n_i + B_j\right) \quad (3)$$

where the f function is given in equation (1) or (2). The output is a float number, so it is necessary to apply a step function:

$$f(out) = \begin{cases} 1, if\ out \ge 0.5 \\ 0, if\ out < 0.5 \end{cases} \quad (4)$$

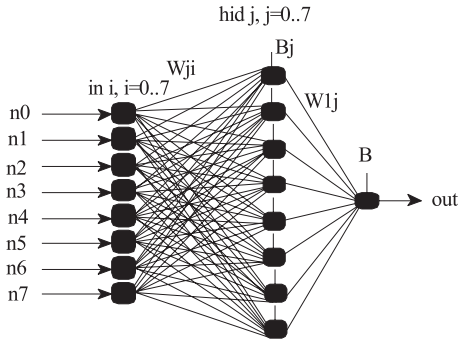

Fig. 10. Two layers neural network

Weights and biases of two layers full connected net Table 1

	W0	W1	W2	W3	W4	W5	W6	W7	B
Hid0	7.18	0.56	2.16	-3.48	13.69	-3.62	2.02	0.54	-6.25
Hid1	-2.32	-0.68	-6.45	-0.64	-2.26	4.54	10.04	4.49	4.88
Hid2	-1.92	5.1	-12.08	2.36	1.34	-0.35	-6.11	-0.71	1.34
Hid3	-8.38	0.21	5.35	1.45	6.78	2.46	-2.52	-0.92	1.45
Hid4	9.19	-3.36	0.93	0.58	3.45	0.2	2.05	-4.7	-0.65
Hid5	6.7	0.37	5.54	1.33	-2.06	-1.23	-2.78	1.81	-4.38
Hid6	3.31	2.47	0.55	-0.99	7.86	0.89	-4.73	-0.08	-4.75
Hid7	-1.11	0.72	6.8	-1.31	2.96	2.56	2.03	1.94	-3.09
Out	-19.88	18.22	14.24	11.98	-13.4	15.2	8.96	-11.04	-0.75

3.3. Single Layer Neural Network

Another method might be the training of several single layer nets (Fig. 11) with sets of training vectors.

For each input vector a descriptor is computed:

$$k = \sum_{i=0}^{7} n_j \cdot 2^i \quad (5)$$

The nets are trained using training vectors with consecutive descriptors. This property is used in the processing stage too, the descriptor pointing to the net that memorised the input vector.

In the processing stage, the descriptor k was computed for each input vector, in order to identify the net. The net's output is given by the following equation:

$$out = f\left(\sum_{i=0}^{7} n_i \cdot W_i + B\right) \quad (6)$$

where f(x) is a simple step function:

$$f(x) = \begin{cases} 1, if\ x \ge 0 \\ 0, if\ x < 0 \end{cases} \quad (7)$$

The algorithms were implemented in C++ under Linux.

4. Results

We are interested in obtaining faster algorithms. A clue indicating the speed is the number of operations. Analytically we have found that the modified Nacache and Shinghal method implies 40 logical plus 3 arithmetic operations/pixel, the two layers neural network implies more than 150 arithmetic operations and the single layer neural network implies 39 arithmetic operations. We have tested all three implementations on a 510 x 412 test image on a Pentium II MMX 350 MHz processor.

The computation times are presented in table 2.

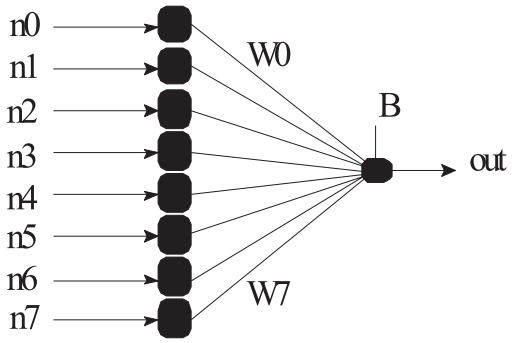

Fig. 11. Single layer neural network

Processing times Table 2

Algorithm	Processing times
Hole detection and localisation	0.67 seconds
Skeletonizing using the original method	0.31 seconds
Skeletonizing using Nacache and Shinghal's method	0.73 seconds
Skeletonizing using two layers net and sigmoid activation function	5.11 seconds
Skeletonizing using two layers net and linear threshold activation function (equation (2))	3.54 seconds
Skeletonizing using single layer net and step activation function	0.89 seconds

5. Conclusion

As expected the two layers neural network is the slowest method due to the large number of operations.

Although the single layer neural network is optimal with respect to the number of operations, these are floating point operations and overall the computation time is longer than using the classical method. Integer approximation of weights and biases doesn't reduce processing time (both floating and integer numbers are kept on 4 bytes), but reduces recognition rate. These results show that the use of neural networks does not improve computation time over the classical methods.

Reviewed by: S. Legutko, J. Pilc

6. References

[1] BORENSTEIN, J., EVERETT, H. R., FENG, L.: Where am I? Sensors and Methods for Mobile Robot Positioning. In: The University of Michigan, 1996.
[2] FAIRHURST, M.: Computer vision for robotic systems: an introduction. In: New York, Prentice Hall, 1988.
[3] FU, K. S., GONZALEZ, R. C., LEE, C. S. G.: ROBOTICS - Control, Sensing, Vision and Intelligence. In: McGraw-Hill, 1987.
[4] JONES, J. L., FLYNN, A. M.: Mobile Robots, Inspiration to Implementation. In: A K Peters, Wellesley, Massachusetts, 1993.
[5] LAZEA, Gh., LUPU, E., PATKO, M.: Aspects on Kinematic Modelling and Simulation of Wheeled Robots. In: International Symposium on Systems Theory, Robotics, Computers and Process Informatics, SINTES 8, Craiova, June 1996, p. 150.
[6] LAZEA, Gh., LUPU, E., PATKO, M.: The Architecture of MAURO Autonomous Mobile Robot. Path Planning and Obstacle Avoidance Modules. In: Proceedings of the 8-th International DAAAM Symposium. 23-25 October 1997, Dubrovnik, Croatia. p.199-200.
[7] LAZEA, Gh., LUPU, E., FOLEA, S.: Ultrasonic Range Finding Sensor. In: A&Q '98, Cluj-Napoca, May 28-29, Ed. Mediamira, 1998. p.337-341.
[8] LAZEA, Gh., LUPU, E., PATKO, M.: MAURO Research Report, Technical University of Cluj-Napoca 1996.
[9] LAZEA, Gh., LUPU, E.: Aspects on Path Planning for Mobile Robots. In: TEMPUS M-JEP 11467 Intensive Course on Computer Aided Engineering in Flexible Manufacturing, Bucharest, May 19-23, 1997.
[10] EVERETT, H. R.: Sensors for Mobile Robots. In: A K Peters, Wellesley, Massachusetts, 1995.
[11] LAZEA, Gh., MUREŞAN, A., LUPU, E., DOBRA, P.: Aspects Concerning Object Localization and Recognition using Video Information for Mobile Robots. In: Proceedings of microCAD '99 International Computer Science Conference, Patko G. (Ed.), pp. 89-94, ISBN 963 661 355 9, Miskolc, February 1999, Miskolci Egyetem, Miskolc
[12] McKERROW, J.: Introduction to Robotics, Addison Wesley, 1996.
[13] NILSSON, N.: Shakey the Robot. In: Artificial Intelligence Center, SRI International Technical Note 323. Menlo Park, CA.
[14] PANDYA, A. S., MACY, R. B.: Pattern recognition with neural networks in C++, Boca Raton, CRC Press, 1996.
[15] POPESCU, D.: Senzorişi sisteme de percepţie artificială (curs), Litografia UPB, Bucuresti, 1991.
[16] ARKIN, C. Ronald: Behavior-based robotic. In: The MIT Press, ISBN 0-262-01165-4, Cambridge (1998)