A line detection and description algorithm based on Swarm Intelligence

Ulrich Kirchmaier, Simon Hawe and Klaus Diepold

Technische Universität München, Arcisstrasse 21, 80290 Munich, Germany

ABSTRACT

In this work, we use the principles of Swarm Intelligence to establish a novel algorithm for detecting and describing straight edges in images. The algorithm uses a set of individual mobile agents with limited cognitive possibilities. Using their memory and communication abilities, the agents can establish fast and robust solutions. The agents initially move randomly in a two dimensional space defined by an arbitrary input image or image sequence. In every time step, each agent calculates the derivative values in x and y direction at its current position and thresholds these values subsequently. If an agent discovers an edge or respectively a straight edge, it follows this straight edge and stores its start point. When it reaches the straight edge's end, it marks its last position as its stop point. As a kind of indirect communication between the agents, each of them leaves important information at each new position discovered. Thus each agent can benefit from the calculations any other agent has done before, which speeds up the algorithm. This new approach is a fast alternative to classical line finding operation like e.g. the Hough Transform.

Keywords: image processing, gradient orientation, straight edge finding, swarm intelligence

1. INTRODUCTION

Finding straight edges, or lines in images is a well known and important problem in image processing. In a grayscale image, an edge generally represents the border between two blocks of different intensity. A line can be defined as a number of pixels of a different intensity on an otherwise unchanged background, which is leading to one edge at each side of the line.

Knowing the existence, positions and orientation of lines inside an image is a major topic in image processing. As straight lines usually represent significant information of an inspected scenario, they are a prerequisite for a huge amount of computer vision tasks like object recognition and scene analysis. Consequently, there has been an amount of research to robust line detection for the last decades, leading to diversified approaches. Statistical methods have been applied to the problem. In,[1] hypotheses were created of statistical models of line segments, which were then tested and verified in images in order. Other approaches analyze the information of the gradient orientation and magnitude of the pixels,[2] which is also the starting point of this work. In,[3] line support regions were established on the basis of the gradient orientation, which enabled the determination of location and the properties of an edge. Furthermore, subspace techniques have been applied. In,[4] the edge image was windowed and the correlation matrix of each window was examined. If the small eigenvalue was below a certain threshold, the windowed area represented a line. This method was shown to be robust to image transformations, while leading to a computation time comparable to fast Hough Transform implementations. However, unsuitable sizes of the window or excessive noise pixels in the edge image can downgrade this method's performance. A similar approach using the Principal Component Analysis was introduced in.[5] The most widespread and well known technique is the Hough-Transform (HT),[6] which transfers an edge image in a parameter space, that is quantized in cells, which each edge pixel is assigned to. When detecting lines, the parameter space is dimensioned by the parameters of a line, e.g. the slope and the intercept. Each cell holding a high accumulation of pixels then corresponds to a straight line. While being generally a very robust approach, the HT's efficiency is dependent on the quality of its input data, i.e. the edge image. Furthermore, as the standard HT is a mandatory grouping method for collinear lines, the actual distribution of edges, the length of the lines is not known. There is a considerable amount of variations of the Hough Transform, we will focus on the Probabilistic Hough Transform (PHT),[7] as it is capable of determining start- and endpoints of lines, which makes its output better comparable to the algorithm presented in this paper. The PHT needs, besides the parameters of the standard HT (the quantization of the parameter space), an initial estimation of the number of lines appearing in the regarded image. The amount of transformation operations make the HT generally computationally intensive, whereas the accuracy as well as the efficiency depend on the quantization. Real time computer vision applications therefore normally need special implementations of the Hough Transform, such as GPU-based or parallelized multi-core implementations.[8]

The term Swarm Intelligence (SI) was first introduced in[9] in the context of Cellular automata, describing the collective behavior of decentralized and self-organized systems. Since then, a variety of different algorithms have been established, each of them basing on the swarm intelligence-concepts inspired by nature, however, each of them interpreting them discriminatively. Examples of very successful implementations are the Particle Swarm Optimization (PSO),[10] the Ant Colony Optimization (ACO)[11] and the Stochastic Diffusion Search (SDS).[12] In recent years, these algorithms have been successfully applied to a variety of research fields, ranging from data mining to robotics.[13, 14] Furthermore a variety of tasks descending from the field of image processing were already effectively approached using SI Concepts. Examples range from from object recognition tasks[12] and segmentation,[15] and tracking,[16] to feature selection,[17] Furthermore, different approaches in edge detection using SI exist.[18, 19]

The Rest of this paper is organized as follows. Section 2 clarifies the terms of lines used in this paper. An introduction to the basic principles of SI follows in section 3. In section 4, the concept of the SI-based straight edge finding-algorithm is explained in detail. Afterwards, results of the algorithm are displayed and compared to the Hough Transform in section 5, which is followed by a conclusion in section 6.

2. TERMINOLOGY

In a grayscale image, an edge generally represents the border between two blocks of different intensity. A straight edge then means a border which follows a (approximately) linear slope between two endpoints. A line can be defined as a number of pixels of a different intensity on an otherwise unchanged background, which is leading to one edge at each side of the line. Strictly speaking, the algorithm in this work therefore detects and describes straight edges instead of lines.

3. SWARM INTELLIGENCE PRINCIPLES

Swarm intelligence is a type of artificial intelligence where the a swarm of individuals, so-called agents interact with each other as well as their environment, each of them following simple rules without any centralized control instance. After some iterations, this collective behavior leads to the emergent phenomenon of an 'intelligent' global behavior, in terms of a globally optimal or near optimal solution, discovered in a defined multidimensional search space and regarding a defined global fitness or function, which has to be minimized. The basic principles of SI concur with the basic skills of each agent, which are its ability to move and to analyze its current position in within a search space, furthermore its memory and its ability to communicate with other agents. There is a variety of implementations of Swarm Intelligence, each of them carrying out these principles diverse, often depending on the biological observation it is descending. For example, the well known ACO simulates the ants of a colony searching for food. Each agent, in this algorithm called ant, that finds food carries a package of it to the anthill, leaving a pheromone trail on its way, which describes the way to the food place for all the other ants of the colony. With several ants finding the same of on different paths, alternative ways will emerge and different ants will take different directions following pheromone trails. The higher a the pheromone-concentration on a trail, the more ants will follow this trail, which will again raise the concentration. As the pheromone evaporates over time, the algorithm successively favors successful paths with short distances, therefore finding the best solution in a search space in terms of a shortest path. In ACO, the ants thus communicate indirectly, via local pheromone amounts. The pheromone concentration controls the ants decision, in the same time the ants decision affects the pheromone amount. Another popular SI algorithm is the PSO, in which a swarm of particles flies through the search space. In every iteration, each particle tests its current solution, respectively its current position using a generally valid fitness function. The algorithm's emergent strength, the robust discovery of a global, or near global best solution is achieved by the rules for every particle's movement. It is influenced by three factors. These are a sort of inertia, which is considering the particles movement up to this time step, the position of the best solution the current particle has discovered so far, which represents a memory ability, and the position of the best solution any particle of the swarm has obtained. The latter term constitutes the particles communication ability. In contrast to the ACO, the PSO's communication can be described as a broadcast communication, as only the currently best solution of all is conveyed to each particle. A third example, the SDS presumes a global cost function that is decomposable into multiple independent partial boolean functions. In every step, each agent tests the hypothesis it maintains on a randomly chosen partial solution. Then the agents, with failed partial solutions randomly chose another agent, asking for its partial solution. If the other agent has a successful partial hypothesis, the failed agent will change its hypothesis according to it. If not, the failed agent will randomly chose a new one. This positive feedback mechanism leads to an agglomeration of agents at the global best solution after some iterations. The SDS uses a third communication mechanism, which can be described as a direct one-on-one communication. These are three examples of differing realizations of the principles of memory, communication and movement of the agents in SI based concepts, which all have their strengths in tackling specific tasks. The implementation that is presented in this paper, in our view, suited for the problem of finding image primitives.

4. CONCEPT OF THE SI-BASED STRAIGHT EDGE FINDING-ALGORITHM

The task of the algorithm is to detect and describe all straight edges within a grayscale input image. For this purpose, a set A agents a with $a = 1 \ldots A$ move and act directly in the two dimensional image space for a user-defined maximum number of iterations. The detected straight edges are represented by the image coordinates of their two Endpoints. The algorithm furthermore delivers a straight edge's orientation with regard to the image origin. The algorithm works as follows:

4.1 Agent behavior

The different situations and the decisions, depending on the local environment, or respectively the communication and the current memory, each agent has to execute require internal states of the agents. This leads to a finite state machine determining the agents' activities, which represents the agents' intelligence. All agents posses four different activity states called *state* $0 \ldots$ *state* 4, as shown in figure 1, which denote following situations as well as the reactions to these situations:

4.1.1 State 0 - agent is currently searching a line

After being initialized, as explained in section 4.2, an agent a checks the positions inside a $n \times n$ - window W, with n being odd and $n > 1$ placed at its current position $p_a(i)$ at iteration i for the existence of an edge, as shown in section 4.3. While no edge is discovered, the agent moves over the image by the speed set randomly during the initialization and continues to check for an edge inside its window, updating its search-time st at each iteration. If it reaches a border of the image, or if it exceeds a certain number of iterations, i.e.

$$st \geq thr\_st, \tag{1}$$

with thr_st being a user-defined threshold, it will switch to *state* 3. If an edge is detected, the agent calculates the edge's orientation and changes its movement direction according to it, as pointed out in section 4.4. Then it sets its current position $p_a(i)$ to the position inside the window, where the edge was detected, sets a temporary first endpoint of a straight edge at this position, and switches to *state* 1 to track the edge.

4.1.2 State 1 - agent is currently tracking a discovered line

While the agent is tracking the straight edge, it moves along the edge by adding its mean direction $mean\_V(i)$ to its current position, i.e.

$$p_a(i+1) = mean\_V(i) + p_a(i), \tag{2}$$

where $mean\_V(i)$ indicates the edge's orientation, see section 4.4. In each iteration, it thereby encounters a new position, at which it at first again checks each position inside W for the existence of an edge, according to section 4.3. Then the agent checks, if the orientation of at least one of the edge positions inside W resembles its current edge orientation, using the orientation condition. While this condition is fulfilled, the agent maintains *state* 1 and continues tracking, updating $mean\_V(i)$ as well as the track time tt in each iteration and placing the currently tracked straight edge's second endpoint at each new found position, thereby exploring its length. A straight edge is only successfully tracked in a position, if the magnitude and the orientation condition ($EdgeCon$ and $OrCon$) are fulfilled. Else, the agent switches to *state* 2.

4.1.3 State 2 - agent is has currently lost track of a line

When an agent that was currently tracking a straight edge encounters a gap or the straight edge's end, it will switch to *state* 2. In this state, it searches for the edge by moving its position according to equation 2, but without updating $mean\_V(i)$. The search stops either, if the agent rediscovers its current line, due to the edge's magnitude and orientation condition, or, if the following condition

$$lt \geq thr\_lt, \tag{3}$$

is reached, with lt being the agent's lost-time, that is recurrently updated, and thr_lt being the maximum lost-time, a user defined threshold. In the first case, the agent will switch back to *state* 1 and continue the tracking. When the second case is reached for the first time, the agent is likely to have moved to the first end of the line. In this situation, it jumps back to first endpoint, where the line was detected, and explores the line in the other direction, by changing the factor f in the equation 5 from 1 to -1 and the signs $mean\_V(i)$. This happens, because an agent will generally more likely discover a straight edge somewhere in the middle than directly at one of its endpoints. In this way, the true first endpoint is detected. When the condition of a maximum lost-time is reached a second time, the agent switches to *state* 3.

4.1.4 State 3 - agent has stopped the search and will be reinitialized

When an agent has reached *state* 3, it resets all the knowledge it has gathered so far. If it has found a straight edge, it will store it into a global line list, under the condition that the line exceeds a user-defined minimum length. As the straight edge's two endpoints are only set during *state* 1, the track-time tt explicitly represents the straight edge's length. Then the variables st, tt and lt are set to zero and the agent is ready to be reinitialized and switched to *state* 0.

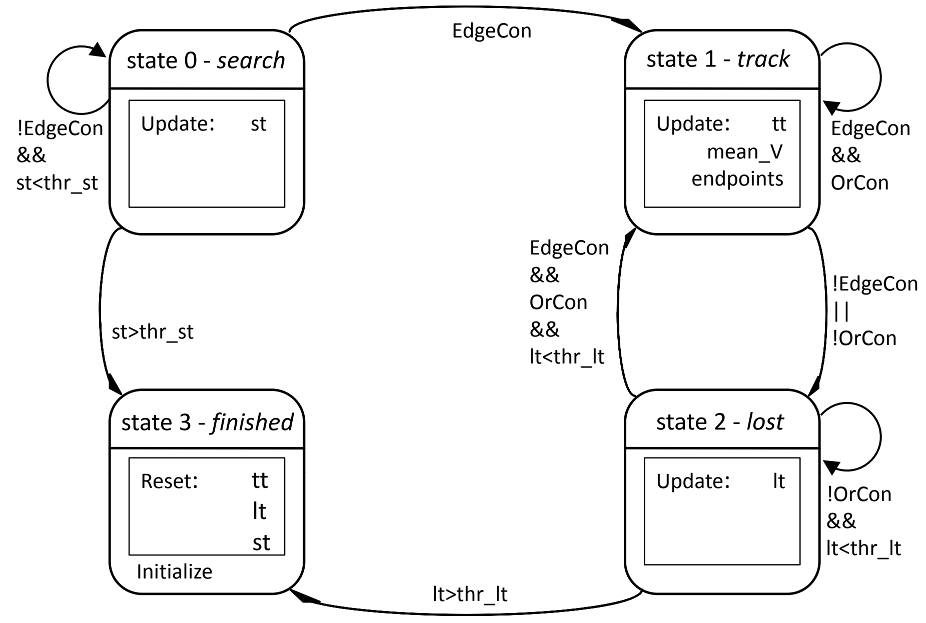


Figure 1. Scheme of the finite state machine within each agent.

4.2 Initialization

In the beginning of the algorithm, but also every time, an agent reaches an image border or *state* 3, as explained in section 4.1, the agent will be reinitialized. Therefore, agents are placed normally distributed over the entire image at positions that have not yet been encountered, in order to explore the entire image. Besides the position, also a movement direction used during the searching process is initialized randomly.

4.3 Edge detection

In every iteration i, an agent encounters a new position $p_a(i)$ and centers the window W around it, within which it will gather its local information. First it checks, if any position inside W was already discovered by another agent. If so, it uses the local information gathered by the other agent. If not, it calculates the image gradients in x and y direction (dx and dy) for each pixel position inside W using a 3×3 - edge detection filter mask, like Sobel, Scharr, or Robert, etc. In the next step it will test if any position inside W represents an edge by thresholding the value of the gradient magnitude. If this condition is fulfilled at at least one position, an edge is presumed at this position, i.e.

$$bool\ EdgeCon = |dx_c| + |dy_r| > thr\_Edge, \tag{4}$$

with $c = x_{p_a(i)} - \frac{n-1}{2} \dots x_{p_a(i)} + \frac{n-1}{2}$ and $r = y_{p_a(i)} - \frac{n-1}{2} \dots y_{p_a(i)} + \frac{n-1}{2}$ being each position inside window W, and thr_Edge being a user-defined threshold.

4.4 Movement direction

While no edge is discovered, the agent continues moving on the image due to the direction set at initialization, leaving the discovered information at all past position. But if an edge is detected at a position inside W, the agent will change his movement direction according to the gradient orientation, as shown in figure 2, in order to fly over the straight edge. The agent's movement direction cur_V is calculated by

$$cur\_V = \begin{pmatrix} x_{cur\_V} \\ y_{cur\_V} \end{pmatrix} = \begin{pmatrix} -f \cdot \frac{dy}{\max(dx,dy)} \\ f \cdot \frac{dx}{\max(dx,dy)} \end{pmatrix}, \tag{5}$$

with $f \neq 0$ being an integer movement scale factor, which annotates the movement speed in pixels. As the gradient vector $(dx\ dy)^T$ is always perpendicular to the edge, swapping of the gradient elements dx and dy and negating one of them, as shown in equation 5, causes the agent to fly in a direction parallel to the edge, or respectively right over it. The direction cur_V is now the direction the agent assumes a possible straight edge to run. In the next iteration, the agent will move into this direction for a small amount and analyze its next position $p(i+1)$.

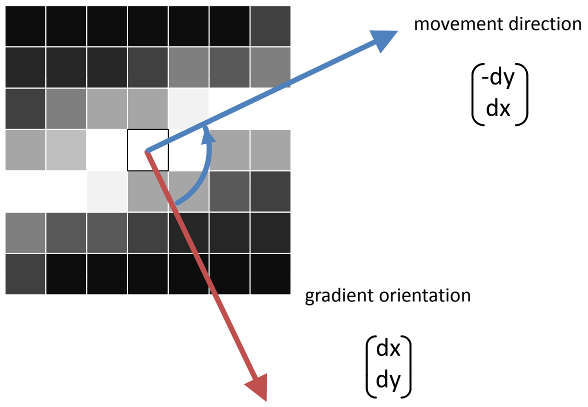

movement direction
$\begin{pmatrix} -dy \\ dx \end{pmatrix}$
gradient orientation
$\begin{pmatrix} dx \\ dy \end{pmatrix}$

Figure 2. Relation between the gradient orientation of a pixel and the movement direction of an agent analyzing its position.

4.5 Edge orientation condition

In each new position, when an agent with *state* 1 or *state* 2 has encountered an edge (i.e. $EdgeCon == true$), it has to decide, wether the discovered edge is valid, i.e. resembles his current orientation. As the movement direction directly represents the orientation of the tracked straight edge, the verification of the current segment can be done by a simple threshold test,

$$bool\ OrCon = |x_{mean\_V(i)} - x_{cur\_V}| < thr\_OrCon \ \&\& \ |y_{mean\_V(i)} - y_{cur\_V}| < thr\_OrCon, \tag{6}$$

with thr_OrCon being a user-defined threshold. When $OrCon = true$, the agent recursively calculates its mean direction $mean\_V(i)$ using

$$mean\_V(i) = \begin{pmatrix} x_{cur\_V} \\ y_{cur\_V} \end{pmatrix} = ((tt-1) * mean\_V(i-1) * cur\_V)/tt. \tag{7}$$

4.6 Memory

While moving over the image and analyzing its positions, each agent gathers some information about its past, which he stores as memory and updates in each iteration. The agent uses several different iteration counters, that evince the number of iterations, the agent is either searching (st), tracking (tt) or has lost (lt) a Line. Furthermore, when an agent is currently tracking a line, he stores the temporary positions of the lines two endpoints in each iteration as well as the mean direction of the line, which can directly express the lines angle with respect to the image origin.

4.7 Communication

In the specific task of this algorithm is to find the straight edges inside the *entire* image, there is no single global solution or optimum, to which the agents will concentrate over iterations, but arbitrarily many solution, i.e. edges, that are more or less independently distributed over the image. Therefore, the agents' movement in this method must not focus on the positions of the best solutions, or respectively partial solution, or on the past path in a search space, the way it is in other SI based optimization algorithms (as described in section 3). Instead, this task demand widely individual movement of the agents, especially during the process of tracking a straight edge.

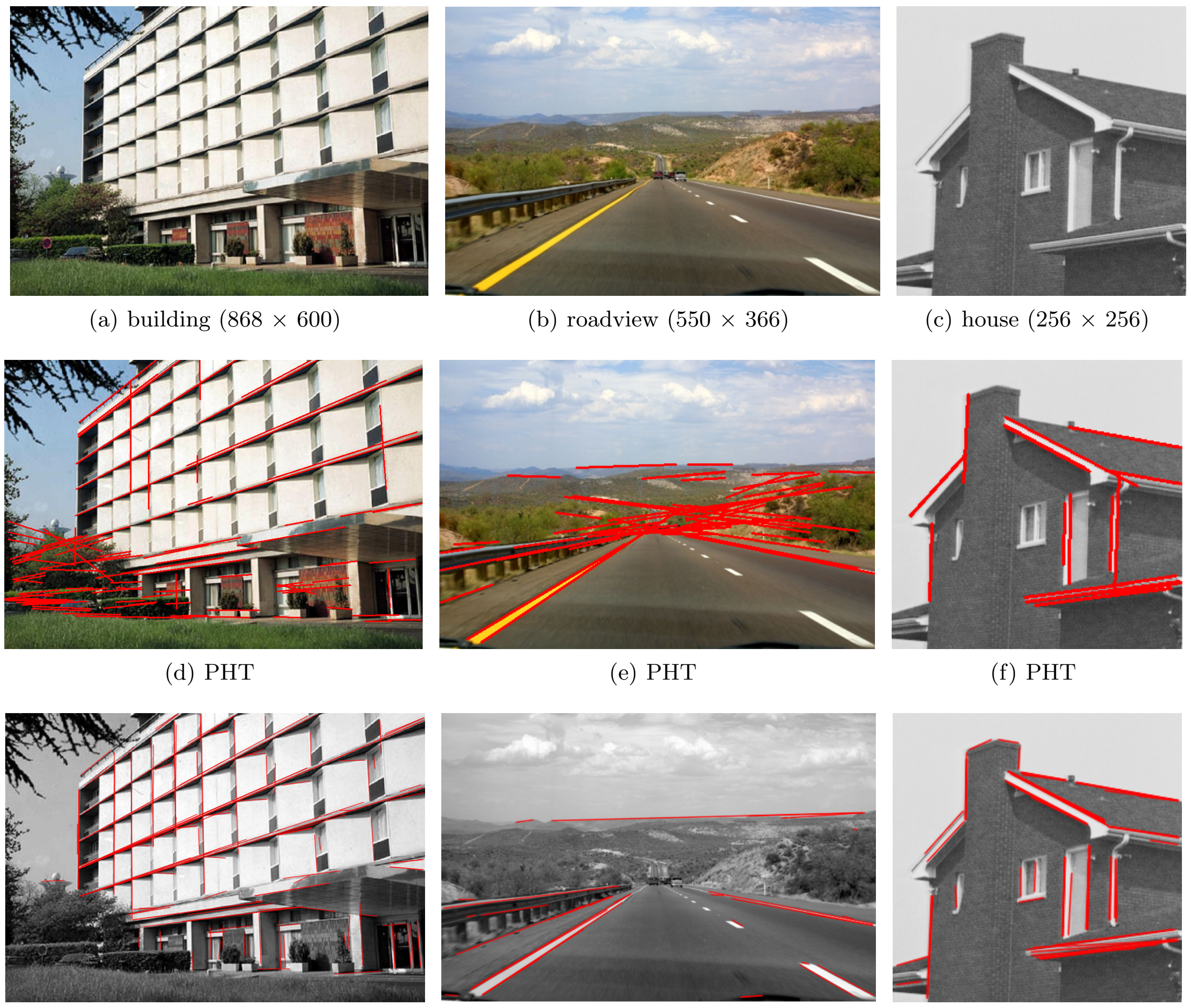
The prerequisite of individual movement generally calls for an indirect kind of communication between the agents that is locally accessible. While they move independently on the image, it is unfavorable for the system's performance, if the information at a specific position is calculated multiple times by different agents encountering this position. Instead, every agent leaves the information it calculated at its current position. Whenever another agent hits this position, it can access all the left information instead of recalculating. The main information each agent can find at each position are

- has the position already been discovered? If so,
- does an edge exist at this position? If so,
- what are the values for dx and dy inside W?

This information also helps the agent to quickly decide its next actions.

5. RESULTS AND COMPARISON

The algorithm was tested on several input images with different resolutions. Due to the lack of objective ground truth material for straight edges or lines in images, the algorithm's qualitative results are compared by inspection. We used a variety of input images, all of which yielded to an output similar to the images shown below. For comparison of the output as well as the computation time, we used the OpenCV-based implementation of the Probabilistic Hough Transform (PHT), for the following reasons. The Hough Transform is generally one of the most widely used methods for line detection and OpenCV is known to work fast and robust. The PHT, which demands a computation time equal to the standard Hough Transform, is in opposite to it capable of determining the lines' endpoints. The OpenCV-based implementation of a Canny-Filter served as input for the PHT. These two modules represent our reference system. The reference system and our algorithm were tested on a 2.6 GHz 3GB RAM QuadCore PC using MutanT,[20] a multi-threaded generic C++ development and testing environment. The following input images serve are shown examples for the algorithms' ouptut. Apparently the striking lines,


(a) building (868 × 600) (b) roadview (550 × 366) (c) house (256 × 256)

(d) PHT (e) PHT (f) PHT

(g) our algorithm (h) our algorithm (i) our algorithm

Figure 3. Example Input images a)-c), and their resulting Output of the Probabilistic Hough Transform (PHT) d)-f), and of the method proposed in this paper g)-i) .

regarding length and saliency, are extracted in both algorithms. However, one can see that, besides detecting the important lines inside the images correctly, the PHT additionally creates several false positives, for example on the lower left part of image 3(d).

One restriction of the HT is, that its parameters have to be well tuned to the input image, like the angle resolution etc. As the HT accumulates the edge pixels a predefined number of cells, each of which represents a certain line, the HT can reach only a maximum accuracy describing the image lines, which depends on the number of cells. Increasing the number of cells leads to an improved accuracy, but concurrently also to a growth of computation time, whereas the angle resolution in our algorithm is only limited by the image resolution, not by the algorithm itself. The need of estimating the number of lines appearing in the input image when using PHT, is another limitation that is overcome by the method proposed in this paper, as the number of lines there is completely arbitrary.

The most prominent feature of our approach is its comparably low computation time. Benefitting from the SI principles, the algorithm works twice as fast as the sum of the computation time of the Canny-Filter plus the PHT with small sized images. With bigger input images, this speed advantage increases further, so that the computation time of our algorithm is only about 25% of the PHT and the Canny-Filter when considering the 'building' image. The mean computation times over several runs for the PHT plus the Canny-Filter and our algorithm on the tested images are shown in table 5.

Input Image	Resolution	Canny (s)	PHT (s)	sum (s)	our algorithm (s)
Building	868×600	0.0165	0.0779	0.1285	0.0319
Roadview	550×366	0.0067	0.0237	0.0304	0.0155
House	256×256	0.0032	0.0090	0.0122	0.0061

Table 1. Mean computation times in seconds (s) of the Canny-Filter, the Probabilistic Hough Transform, the sum of both, and the method proposed in this paper, on different input images.

6. CONCLUSION

In this paper, we presented a novel algorithm for detecting and describing straight edges in images. It creates a set of individual agents and uses the principles of Swarm Intelligence, like memory and communication, to efficiently calculate and compare local image gradients and seek for lines. The algorithm was shown to be faster than the most widely used line detection method, the Hough Transform, while delivering equal qualitative results. The arbitrary input parameters of the number of agents and the number of iterations used makes this method adjustable to trade offs between accuracy and speed, depending on the demanded speed of the application. The mechanisms of the track-time and the lost-time make it robust to gaps or partial occlusions of the straight edges, as well as to misleading noise pixels.

In future work, research will be spent to increase the robustness of the straight edge detection in sense of repeatability, which until now shows dependance on the number of agents and iterations, which causes the detected lines to slightly vary in length and assumed position. Furthermore, this agent-based approach for is to be extended to other other image processing tasks. Slight variations of the mechanism are supposed to be well suited for detecting and describing other image primitives, like corners, polygons, circles or ellipses.

REFERENCES

[1] Mansouri, A.-R., Malowany, A. S., and Levine, M. D., "Line detection in digital pictures: a hypothesis prediction/verification pardigm," *Comput. Vision Graph. Image Process.* **40**(1), 95–114 (1987).

[2] Nelson, R. C., "Finding line segments by stick growing," *IEEE Trans. Pattern Anal. Mach. Intell.* **16**(5), 519–523 (1994).

[3] Burns, J. B., Hanson, A. R., and Riseman, E. M., "Extracting straight lines," *IEEE Trans. Pattern Anal. Mach. Intell.* **8**(4), 425–455 (1986).

[4] Guru, D. S., Shekar, B. H., and Nagabhushan, P., "A simple and robust line detection algorithm based on small eigenvalue analysis," *Pattern Recogn. Lett.* **25**(1), 1–13 (2004).

[5] Lee, Y.-S., Koo, H.-S., and Jeong, C.-S., "A straight line detection using principal component analysis," *Pattern Recognition Letters* **27**(14), 1744 – 1754 (2006).

[6] Hough, P., "Method and means for recognizing complex patterns," (1962).

[7] Kiryati, N., Eldar, Y., and Bruckstein, A. M., "A probabilistic hough transform," *Pattern Recogn.* **24**(4), 303–316 (1991).

[8] Chen, Y.-K., Li, W., Li, J., and Wang, T., "Novel parallel hough transform on multi-core processors," in [*Acoustics, Speech and Signal Processing. (ICASSP 2008)*], 1457–1460 (2008).

[9] Beni, G. and Wang, J., "Swarm intelligence in cellular robotic systems," in [*NATO Advanced Workshop on Robots and Biological Systems*], 26–30 (July 1989).

[10] Kennedy, J. and Eberhart, R., "Particle swarm optimization," in [*IEEE International Conference on Neural Networks*], 1942–1948 (1995).

[11] Colorni, A., Dorigo, M., and Maniezz, V., "Distributed optimization by ant colonies," in [*European Conference on Artificial Life (ECAL)*], Publishing, E., ed., 134–142 (1991).

[12] Bishop, J., "Stochastic searching networks," in [*Artificial Neural Networks, 1989., First IEE International Conference on (Conf. Publ. No. 313)*], 329–331 (Oct 1989).

[13] Abraham, A., Grosan, C., and Ramos, V., [*Swarm Intelligence in Data Mining*], vol. 34 of *Studies in Computational Intelligence*, Springer (2006).

[14] Pini, G. and Tuci, E., "On the design of neuro-controllers for individual and social learning behaviour in autonomous robots: an evolutionary approach," *Connection Science* **20**, 211–230 (June-September 2008).

[15] Cagnoni, S., Mordonini, M., and Sartori, J., "Particle swarm optimization for object detection and segmentation," in [*EvoWorkshops*], 241–250 (2007).

[16] Keyrouz, F., Kirchmaier, U., and Diepold, K., "Three dimensional object tracking based on audiovisual fusion using particle swarm optimization," in [*11th International Conference on Information Fusion*], 611–615 (July 2008).

[17] Kanan, H., Faez, K., and Taheri, S., "Feature selection using ant colony optimization (aco): A new method and comparative study in the application of face recognition system," in [*7th Industrial Conference on Data Mining (ICDM)*], 63–76 (July 2007).

[18] Tian, J., Yu, W., and Xie, S., "An ant colony optimization algorithm for image edge detection," in [*Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on*], 751–756 (June 2008).

[19] Rezaee, A., "Extracting edge of images with ant colony," *Journal of Electrical Engineering* **59**(1), 57–59 (2008).

[20] Hawe, S., Kirchmaier, U., and Diepold, K., "Mutant: A modular and generic tool for multi-sensor data processing," in [*12th International Conference on Information Fusion*], 1304–1309 (July 2009).