Vol. 27, No. 1/4, 2018

Machine GRAPHICS & VISION

International Journal

Published by
The Faculty of Applied Informatics and Mathematics – WZIM
Warsaw University of Life Sciences – SGGW
Nowoursynowska 159, 02-776 Warsaw, Poland
in cooperation with
The Association for Image Processing, Poland – TPO

Interpreted Graphs and $ETPR(k)$ Graph Grammar Parsing for Syntactic Pattern Recognition

Mariusz Flasiński
IT Systems Department, Jagiellonian University
ul. St. Łojasiewicza 4, 30-384 Cracow, Poland

Abstract. Further results of research into graph grammar parsing for syntactic pattern recognition (*Pattern Recognit.* 21:623–629, 1988; 23:765–774, 1990; 24:1223–1224, 1991; 26:1–16, 1993; 43:249–2264, 2010; *Comput. Vision Graph. Image Process.* 47:1–21, 1989; *Fundam. Inform.* 80:379–413, 2007; *Theoret. Comp. Sci.* 201:189–231, 1998) are presented in the paper. The notion of interpreted graphs based on Tarski's model theory is introduced. The bottom-up parsing algorithm for $ETPR(k)$ graph grammars is defined.

Key words: syntactic pattern recognition, graph grammar, parsing, interpreted graph, model theory

1. Introduction

Syntactic pattern recognition consists in representing patterns with string, tree or graph structures, defining generative grammars for sets of such structures, and using corresponding automata/syntax analyzers for classifying unknown structural patterns [1, 2, 3, 4]. Graph grammars are the strongest formalism generating structural patterns and they are used in machine graphics and vision for this purpose for more than 40 years [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19].

Although the extensive research into parsing of graph languages for syntactic pattern recognition has been carried out, only a few syntax analyzers for graph-based patterns have been developed. The efficient parser for expansive graph grammars was defined by Fu and Shi in 1983 [20]. In 1990 parsing algorithms for plex grammars were constructed by Bunke and Haller [21], and Peng, Yamamoto and Aoki [22]. Then, syntax analyzers for relational grammars were proposed by: Wittenburg, Weitzman and Talley in 1991 [23], and Ferruci, Tortora, Tucci and Vitiello in 1994 [24]. The parsing algorithm for context-sensitive layered grammars was presented by Rekers and Schürr in [25]. The parsing method for reserved graph grammars was defined by Zhang, Zhang and Cao in 2001 [26].

Till the first half of 1990s three efficient parsing algorithms, $O(n^2)$, were defined for subclasses of classical *Node Label Controlled* (*NLC*) graph grammars introduced in [27]. Firstly, the syntax analyzer for the regular $ETL(1)$ subclass of *edNLC* languages was proposed in [28, 29], then its error-correcting extension was defined [30, 31], and finally the parser for the context-free $ETPL(k)$ subclass of *edNLC* languages was

constructed [32]. Moreover, formal power properties of $ETPL(k)$ graph languages were characterized in [33] and the inference algorithm for these languages was defined [34]. The $ETPL(k)$ parser was successfully used in a variety of practical applications like: scene analysis in robotics [28], software allocation in distributed systems [35], CAD/CAM integration [36, 37], reasoning in real-time expert system [38], mesh refinement (finite element method, FEM) in CAE system [39], reasoning with semantic networks in AI systems [40], sign language recognition [41, 42].

The successful use of the $ETPL(k)$ model in aforementioned applications results, among others, from the linear ordering of graphs representing visual objects, components of networks, etc. In turn, the effective definition of this ordering is the result of constructing the graphs on the basis of semantic features of represented phenomena. Formulating general preconditions allowing one to construct such graphs, called here *interpreted graphs* is the first goal of the paper. The complete formalization of the bottom-up parsable version of the deterministic subclass of *edNLC* graph grammars analogous to $ETPL(k)$ grammars[1] and presenting the $ETPR(k)$ parsing algorithm analogous to the one introduced in [32] is the second goal of the paper.

Definitions relating to *edNLC* graph grammars are presented in Section 2. Notions of interpreted graphs and (reversely) indexed edge-unambiguous graphs that allow us to introduce linear ordering on graphs used for representing patterns are included in Section 3. Definitions of bottom-up parsable $ETPR(k)$ graph grammars are contained in Section 4, whereas in Section 5 the $ETPR(k)$ parsing algorithm is presented. The concluding remarks are included in the last section.

2. Preliminaries

In this section we present basic definitions of: *EDG* graph, *edNLC* graph grammar and *edNLC* graph language [27].

Definition 2.1. A *directed node- and edge-labelled graph*, *EDG* graph, over Σ and Γ is a quintuple

$$H = (V, E, \Sigma, \Gamma, \phi) \text{ , where}$$

V is a finite, non-empty set of nodes,
Σ is a finite, non-empty set of node labels,
Γ is a finite, non-empty set of edge labels,
E is a set of edges of the form (v, λ, w), where $v, w \in V, \lambda \in \Gamma$,
$\phi : V \longrightarrow \Sigma$ is a node-labelling function.

The *family of all the EDG graphs over* Σ *and* Γ is denoted by $EDG_{\Sigma,\Gamma}$. The components V, E, ϕ of a graph H are sometimes denoted with V_H, E_H, ϕ_H.

[1] The preliminary formalization was presented in [37].

Let $A = (V_A, E_A, \Sigma, \Gamma, \phi_A)$, $B = (V_B, E_B, \Sigma, \Gamma, \phi_B)$ and $C = (V_C, E_C, \Sigma, \Gamma, \phi_C)$ be *EDG* graphs. An *isomorphism from A onto B* is a bijective function h from V_A onto V_B such that

$$\phi_B \circ h = \phi_A \text{ and } E_B = \{(h(v), \lambda, h(w)) : (v, \lambda, w) \in E_A\} .$$

We say that A *is isomorphic to* B, and denote it with $A \stackrel{\text{isom}}{=} B$.
A graph C is a *(full) subgraph* of B iff $V_C \subseteq V_B, E_C = \{(v, \lambda, w) \in E_B : v, w \in V_C\}$ and ϕ_C is the restriction to V_C of ϕ_B.

Definition 2.2. An *edge-labelled directed Node Label Controlled*, *edNLC*, *graph grammar* is a quintuple

$$G = (\Sigma, \Delta, \Gamma, P, Z), \text{where}$$

Σ is a finite, non-empty set of node labels,
$\Delta \subseteq \Sigma$ is a set of terminal node labels,
Γ is a finite, non-empty set of edge labels,
P is a finite set of productions of the form (l, D, C), in which
$l \in \Sigma \backslash \Delta, D \in EDG_{\Sigma,\Gamma}, C : \Gamma \times \{\text{in}, \text{out}\} \longrightarrow 2^{\Sigma \times \Sigma \times \Gamma \times \{\text{in,out}\}}$ is the embedding transformation,
$Z \in EDG_{\Sigma,\Gamma}$ is the starting graph called the *axiom*.

Definition 2.3. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be an *edNLC* graph grammar.

1. Let $H, \overline{H} \in EDG_{\Sigma,\Gamma}$. Then H *directly derives* $\overline{H}$ in G, denoted by $H \underset{G}{\Longrightarrow} \overline{H}$, if there exists a node $v \in V_H$ and a production (l, D, C) in P such that the following holds.
   (a) $l = \phi_H(v)$.
   (b) There exists an isomorphism from $\overline{H}$ onto the graph X in $EDG_{\Sigma,\Gamma}$ constructed as follows. Let $\overline{D}$ be a graph isomorphic to D such that $V_H \cap V_{\overline{D}} = \emptyset$ and let h be an isomorphism from D onto $\overline{D}$. Then

$$X = (V_X, E_X, \Sigma, \Gamma, \phi_X) ,$$

where

$$\begin{aligned} V_X &= (V_H \setminus \{v\}) \cup V_{\overline{D}} , \\ \phi_X(y) &= \begin{cases} \phi_H(y) & \text{if} \quad y \in V_H \setminus \{v\} , \\ \phi_{\overline{D}}(y) & \text{if} \quad y \in V_{\overline{D}} , \end{cases} \end{aligned}$$

$$
\begin{aligned}
E_X \;=\; & (E_H \setminus \{(n,\gamma,m) : n = v \text{ or } m = v\}) \cup E_{\overline{D}} \cup \\
& \{\,(n,\gamma,m) : n \in V_{\overline{D}}\,, m \in V_{X\setminus\overline{D}} \text{ and there exists an edge } (m,\lambda,v) \in E_H \\
& \quad \text{such that} (\phi_X(n), \phi_X(m), \gamma, \text{out}) \in C(\lambda, in)\,\} \cup \\
& \{\,(m,\gamma,n) : n \in V_{\overline{D}}\,, m \in V_{X\setminus\overline{D}} \text{ and there exists an edge } (m,\lambda,v) \in E_H \\
& \quad \text{such that} (\phi_X(n), \phi_X(m), \gamma, \text{in}) \in C(\lambda, \text{in})\,\} \cup \\
& \{\,(n,\gamma,m) : n \in V_{\overline{D}}\,, m \in V_{X\setminus\overline{D}} \text{ and there exists an edge } (v,\lambda,m) \in E_H \\
& \quad \text{such that} (\phi_X(n), \phi_X(m), \gamma, \text{out}) \in C(\lambda, \text{out})\,\} \cup \\
& \{\,(m,\gamma,n) : n \in V_{\overline{D}}\,, m \in V_{X\setminus\overline{D}} \text{ and there exists an edge } (v,\lambda,m) \in E_H \\
& \quad \text{such that} (\phi_X(n), \phi_X(m), \gamma, \text{in}) \in C(\lambda, \text{out})\,\}\,.
\end{aligned}
$$

2. By $\overset{*}{\underset{G}{\Longrightarrow}}$ we denote the transitive and reflexive closure of $\underset{G}{\Longrightarrow}$.

3. The language of G, denoted $L(G)$, is the set

$$L(G) = \{H : Z \overset{*}{\underset{G}{\Longrightarrow}} H \;\; and \; H \in EDG_{\Delta,\Gamma}\}\,.$$

An example of a derivation step is shown in Fig. 1. The graph h which will be transformed is shown in Fig. 1a. The left-hand side $l = A$ and the right-hand side D of a production are shown in Fig. 1b. Let us assume that the embedding transformation is defined in the following way.

$$\begin{aligned}
&\textbf{(i)} \quad C(p, \text{in}) \;\; = \{(D, B, v, \text{out})\}\,, \\
&\textbf{(ii)} \quad C(v, \text{out}) = \{(b, a, v, \text{out})\}\,.
\end{aligned}$$

The derivation step is performed in two stages. Firstly, the node labelled with A of the graph h is removed and the graph of the right-hand side D is placed instead of this node. The transformed graph after removing the node is called the rest graph. During the second stage the embedding transformation is used in order to connect certain nodes of the graph D with the rest graph. The item 2 is interpreted in the following way.

1. Each edge labelled with p and coming *in* the node corresponding to the left-hand side of a production, i.e. A, should be replaced by
2. the edge:
   (a) connecting the node of the graph of the right-hand side of the production and labelled with D with the node of the rest graph and labelled with B,
   (b) labelled with v,
   (c) and going *out* from the node D.

So, the item 2 generates the edge of the graph $\overline{h}$, shown in Fig. 1c, which is labelled with v and connects nodes labelled with D and B on the basis of the edge of the graph h labelled with p and connecting nodes labelled with A and B. Let us notice that the application of the item 2 preserves the edge labelled with v of the graph h.

3. Interpreted graphs and indexed edge-unambiguous graphs

As it has been discussed in [33], there are two main reasons of difficulties with constructing efficient parsing algorithms for graph languages (comparing with such algorithms for string and tree languages): the lack of ordering of graph structure and the complexity of embedding transformation. In this section we consider the first problem.

Let is notice that the main idea of any syntax analysis algorithm consists in repetitive tearing off succeeding subphrases/substructures (*handles*) from an analyzed sentence/structure and matching them against phrases/structures which are defined on the basis of right-hand sides of productions. In case of a graph structure it means looking for a subgraph (a handle) which is isomorphic to a given graph, i.e. resolving the subgraph isomorphism problem known to be NP-complete. To resolve this problem we have introduced the subclass of *EDG* graphs called *indexed edge-unambiguous graphs, IE graphs* [28, 33] in which the linear order has been defined. In spite of the fact that

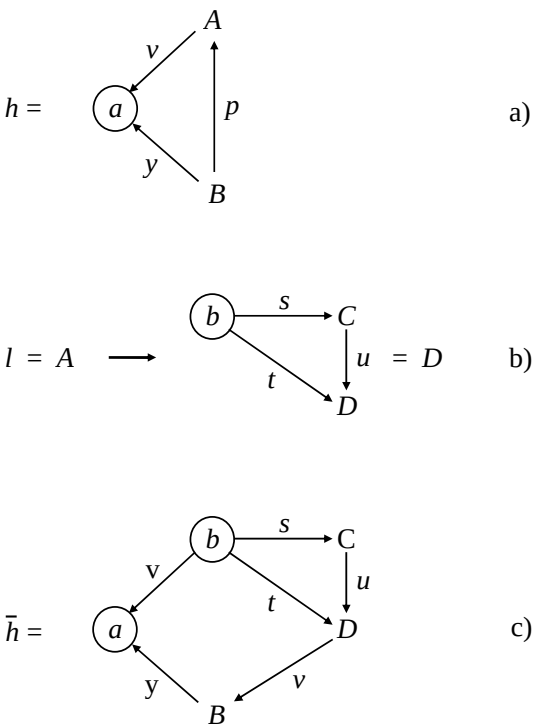


Fig. 1. An example of a derivation step in an *edNLC* graph grammar.

formal restrictions have been imposed on the class of *IE* graphs, it has turned out that they do not limit its descriptive power in practice. The *IE* graphs have been used as a descriptive formalism for representing: combinations of objects of scenes analyzed by industrial robots [28], configurations of hardware/software components analyzed by distributed software allocators [35], structures consisting of geometrical/topological features of machined parts in CAD/CAM integration systems [36, 37], frames in real-time expert systems [38], grids analyzed with FEA methods in CAE systems [39], semantic networks in AI systems [40] hand postures analyzed by sign language recognition systems [41, 42].

Although specific preconditions which have to be fulfilled for the effective use of *IE* graphs have been determined and discussed for each of these applications, they have not been defined formally in a general case till now. Only in [28, 33] it has been pointed out intuitively that one has to refer to a *semantic aspect* of a graph representation. Indeed, in order to formulate general conditions for the effective use of the class of *IE* graphs we have to refer to *Tarski's (semantic) model theory*. We will use the model theory approach to define the class of *interpreted EDG graphs*.

Graphs are used in computer science for representing *structures* which consist of objects and relations among them. These objects, corresponding to *individual objects* in logic, can represent physical entities/phenomena (e.g. Albert Einstein, Hurricane Katrina), sets (groups) of entities (e.g. my family), social/cultural/political constructs (e.g. UNESCO, USA), (theoretical) concepts (e.g. the set of natural numbers, triangle, animal). Individual objects are represented with graph nodes, whereas relations among these objects are represented with graph edges. Hereinafter structures represented with graphs will be called *relational structures* (in order to distinguish them from (abstract) structures defined in model theory)[2] Then we will assume that a graph node is characterized with *node attributes* and a *node label* (which in fact is a kind of the distinguished attribute). Usually for a graph node representing a *unique object* (e.g. (this) Albert Einstein) the node label corresponds to the unique "identifier" of the object, and for a graph node representing a *concept object (class, category)* (e.g. tree) the node label corresponds to the name of the concept. A graph edge between nodes v and w is characterized with an *edge label* which defines the kind of the relation between objects (of a relational structure) which are represented with the nodes v and w. Since we construct our formalism for *EDG* graphs we assume that relations are binary and there are no multiple relations between objects (cf. [27]). For graph edges, similarly as for nodes, attributes can be also defined.

Let us formalize our considerations. Firstly, we introduce the definition of *relational structure*.

Definition 3.1. Let $\mathcal{U}$ be a finite set of individual objects called *universe*, $\mathcal{N}_{\mathcal{U}}$ be a set of their names, $\mathcal{A}_{\mathcal{U}}$ be a set of their attributes.

[2]That is, in our terminology *graphs* (treated as a representative formalism) represent *relational structures* that are constituted by objects and relations among them.

Let each object $o^k, k = 1, \ldots K$ of $\mathcal{U}$ be represented by its name $n_u^k \in \mathcal{N}_\mathcal{U}$ and the set of its attributes[3] $a_u^k \in 2^{\mathcal{A}_\mathcal{U}}$.
Let $\mathcal{R} \subset 2^{\mathcal{U}\times\mathcal{U}}$ be a set of binary relations such that for a pair of objects at most one relation is established, $\mathcal{N}_\mathcal{R}$ be a set of their names, $\mathcal{A}_\mathcal{R}$ be a set of their attributes, $\mathcal{R} = \{(n_r, a_r) \mid n_r \in \mathcal{N}_\mathcal{R}, a_r \in 2^{\mathcal{A}_\mathcal{R}}\}$.
A *relational structure* is a sextuple $\mathfrak{S} = (\mathcal{U}, \mathcal{R}, \mathcal{N}_\mathcal{U}, \mathcal{N}_\mathcal{R}, \mathcal{A}_\mathcal{U}, \mathcal{A}_\mathcal{R})$.

Now we can define the *interpretation of EDG graph over relational structure* and the *interpreted EDG graph*.

Definition 3.2. Let $H = (V, E, \Sigma, \Gamma, \phi)$ be an *EDG* graph over Σ and Γ, $\mathfrak{S}$ be a relational structure defined as in Definition 3.1, $\Sigma \subset \mathcal{N}_\mathcal{U}, \Gamma \subset \mathcal{N}_\mathcal{R}$.
An *interpretation* $\mathcal{I}$ of the graph H over the structure $\mathfrak{S}$ is a pair

$$\mathcal{I} = (\mathfrak{S}, \mathcal{F}) \text{ , where}$$

$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2)$ is the *denotation function* defined in the following way.

- $\mathcal{F}_1$ assigns an object $u \in \mathcal{U}$ having a name $a \in \mathcal{N}_\mathcal{U}$ to each graph node $v \in V, \phi(v) = a, a \in \Sigma$,
- $\mathcal{F}_2$ assigns a pair of objects $(u', u'') \in r, r \in \mathcal{R}$ to each graph edge $(v, \lambda, w) \in E$, $v, w \in V, \lambda \in \Gamma$ such that $\mathcal{F}_1(v) = u'$, $\mathcal{F}_1(w) = u''$ and r has the name λ.

Definition 3.3. Let H be an *EDG* graph over Σ and Γ, $\mathfrak{S}$ be a relational structure, $\mathcal{I}$ be the interpretation of H over $\mathfrak{S}$ defined as in Definition 3.2. An *interpreted EDG graph* is a triple $H^\mathcal{I} = (\mathfrak{S}, H, \mathcal{I})$.

The *family of all the EDG graphs over* Σ *and* Γ *interpreted by* $\mathcal{I}$, shortly *interpreted EDG graphs*, is denoted by $EDG^\mathcal{I}_{\Sigma,\Gamma}$.

The examples of defining interpreted graphs for representing machined parts in the vision subsystem of the CAD/CAM system [36] and hand gestures in the Polish Sign Languages recognition system [41] are shown in Figs. 2a and 2b, respectively.

As we have mentioned above, we have been able to define the linear order for *EDG* graphs in various application areas because, in fact, we have considered *interpreted EDG* graphs. In all mentioned applications of *edNLC* grammars, the linear order has been introduced on the basis of semantics (i.e. attributes) of graphs representing relational structures.

Before we introduce the family of indexed edge-unambiguous graphs (defined for top-down parsable *edNLC* languages) and the family of reversely indexed edge-unambiguous graphs (defined for bottom-up parsable *edNLC* languages), let us define the string-like graph representation of *EDG* graphs as in [28, 32]. (This kind of the representation was originally defined for Ω graphs in [20].)

Definition 3.4. Let $k \in V$ be the node having the index k of the *EDG* graph $H = (V, E, \Sigma, \Gamma, \phi)$. A *characteristic description* of the node k is the quadruple

[3] The set of attributes for an object can be the empty set.

$$n(k), r, (e_1 \dots e_r), (i_1 \dots i_r) \ , \text{ where}$$

n is the label of the node k, i.e. $\phi(k) = n$,
r is the out-degree of k (out-degree of the node designates the number of edges going out from this node),
$(i_1 \dots i_r)$ is the string of node indices to which edges going out from k come (in increasing order),
$(e_1 \dots e_r)$ is the string of edge labels ordered in such a way that the edge having the label e_x comes into the node having the index i_x.

If nodes of the graph h from Fig. 1a labelled with: a, B, A are indexed with: $1, 2, 3$, respectively, then:

$$B(2), \quad 2, \quad (y\ p), \quad (1\ 3)$$

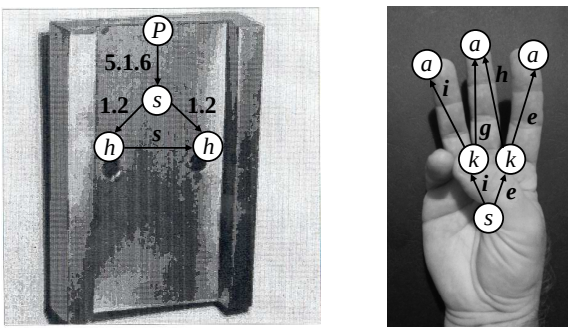
is the characteristic description of the node indexed with 2.

Definition 3.5. Let $H = (V, E, \Sigma, \Gamma, \phi)$ be an *IE* graph, where $V = \{1, \dots, k\}$ is the set of its nodes, $I(i), i = 1, \dots, k$ is the characteristic description of the form of the node i. A string $I(1) \dots I(k)$ is called a *characteristic description* of the graph H.

Assuming a way of indexing of the graph h from Fig. 1a as it has been defined above, we receive the following characteristic description of this graph

$a(1)$	$B(2)$	$A(3)$
0	2	1
–	$y\ p$	v
–	1 3	1

.



a) b)

Fig. 2. The application of interpreted graphs for representing (a) machined parts in the vision subsystem of the CAD/CAM system [36] and (b) hand gestures in the Polish Sign Language recognition system [41].

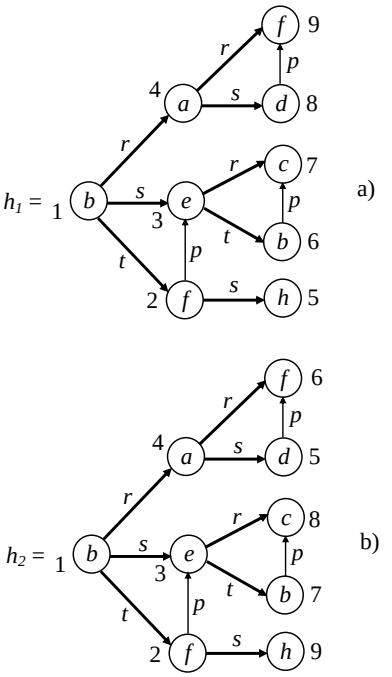


Fig. 3. An example of an *IE* graph (a) and an *rIE* graph (b).

On the basis of semantic features of *EDG* graphs we have constructed the so-called *IE* graphs used in the top-down *ETPL*(k) parsing scheme [28, 32]. Let us define them formally on the basis of the concept of *interpreted graphs* introduced above in Definition 3.3.

Definition 3.6. Let $H^{\mathcal{I}} = (\mathfrak{S}, H, \mathcal{I})$ be an interpreted *EDG* graph over Σ and Γ. An *indexed edge-unambiguous graph*, *IE* graph, over Σ and Γ defined on the basis of the graph $H^{\mathcal{I}}$ is an *EDG* graph $G = (V, E, \Sigma, \Gamma, \phi)$ which is isomorphic to H up to direction of edges[4], such that the following conditions are fulfilled.

1. G contains a directed spanning tree T such that nodes of T have been indexed due to the Level Order Tree Traversal (LOTT)[5].
2. Nodes of G are indexed in the same way as nodes of T.

[4]That is, (some) edges of G can be re-directed with respect to their counterparts in H.

[5]Let us recall that LOTT means that for each node firstly the node is visited, then its child nodes are put into the FIFO queue. This type of a tree traversal is also known as the Breadth First Search (BFS) scheme.

3. Every edge in G is directed from the node having a smaller index to the node having a greater index.

The *family of all the IE graphs over* Σ *and* Γ is denoted by $IE_{\Sigma,\Gamma}$.
The exemplary IE graph h_1 is shown in Fig. 3a. The indices are ascribed to the graph nodes according to LOTT. The edges of the spanning tree T are thickened.

Since in the paper we characterize formally bottom-up parsable $ETPR(k)$ graph grammars, we will introduce the class of graphs which are generated by these grammars. These graphs should be indexed according to the scheme allowing us to apply a reductive parsing. During such a parsing a syntax analyzer produces the rightmost derivation in reverse. (As it is performed for Knuth's (string) $LR(k)$ parsers [43].) The class of such graphs has been introduced informally in [37]. We define them on the basis of interpreted graphs using the new traversal scheme called the *Reverse Level Order Tree Traversal (RLOTT)*. This scheme is analogous to the LOTT scheme used above for *IE* graphs, however it uses the LIFO queue, i.e. the stack, instead of the FIFO queue.

Definition 3.7. Let $H^{\mathcal{I}} = (\mathfrak{S}, H, \mathcal{I})$ be an interpreted *EDG* graph over Σ and Γ. A *reversely indexed edge-unambiguous graph*, *rIE* graph, over Σ and Γ defined on the basis of the graph $H^{\mathcal{I}}$ is an *EDG* graph $G = (V, E, \Sigma, \Gamma, \phi)$ which is isomorphic to H up to direction of edges, such that the following conditions are fulfilled.

1. G contains a directed spanning tree T such that nodes of T have been indexed due to the Reverse Level Order Tree Traversal (RLOTT).
2. Nodes of G are indexed in the same way as nodes of T.
3. Every edge in G is directed from the node having a smaller index to the node having a greater index.

The *family of all the rIE graphs over* Σ *and* Γ is denoted by $rIE_{\Sigma,\Gamma}$.

The exemplary IE graph h_2 is shown in Fig. 3b.

At the end of this section we introduce the notion of node level. We say that the node v of the *IE* (*rIE*) graph is of the n level, if v is at the n level of the spanned tree T constructed as in Definition 3.6 (Definition 3.7).

4. Formal properties of $ETPR(k)$ graph grammars

The formal properties of the $ETPR(k)$ bottom-up parsable subclass of *edNLC* graph grammars are presented in this section. Some of them are analogous to those imposed of the $ETPL(k)$ top-down parsable graph grammars [28, 29, 32].

Firstly, let us impose the constraint on the form of the right-hand side graphs in order to reduce the computational complexity of a single derivation step.

Definition 4.1. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be an *edNLC* graph grammar. The grammar G is called a *TLP graph grammar*, abbrev. from *Two-Level Production*, if the following conditions are fulfilled.

1. P is a finite set of productions of the form (l, D, C), where:
   (a) $l \in \Sigma$,
   (b) D is the *rIE* graph having the characteristic description:

$n_1(1)$	$n_2(2)$	$\ldots$	$n_m(m)$	or	$n_1(1)$,	where	$n_i(i)$
r_1	r_2	$\ldots$	r_m		0		r_i
E_1	E_2	$\ldots$	E_m		$-$		E_i
I_1	I_2	$\ldots$	I_m		$-$		I_i

   is a characteristic description of the node $i, i = 1, \ldots, m, n_1 \in \Delta$ (i.e. n_1 is a terminal label), $i, i = 2, \ldots, m$ is the node of the second level,
   (c) $C : \Gamma \times \{\text{in}, \text{out}\} \longrightarrow 2^{\Sigma \times \Sigma \times \Gamma \times \{\text{in},\text{out}\}}$ is the embedding transformation.
2. Z is an *rIE* graph such that its characteristic description satisfies the condition defined in point 1(b).

Let us require a derivation process to be performed according to the linear ordering imposed on *rIE* graphs.

Definition 4.2. A *TLP* graph grammar G is called a *closed rTLP graph grammar* G if for each derivation of this grammar

$$Z = g_0 \underset{G}{\Longrightarrow} g_1 \underset{G}{\Longrightarrow} \ldots \underset{G}{\Longrightarrow} g_n$$

a graph $g_i, i = 0, \ldots, n$ is an *rIE* graph.

Definition 4.3. Let there be given a derivation of a closed *rTLP* graph grammar G:

$$Z = g_0 \underset{G}{\Longrightarrow} g_1 \underset{G}{\Longrightarrow} \ldots \underset{G}{\Longrightarrow} g_n \ .$$

This derivation is called a *regular right-hand side derivation*, denoted $\underset{\text{rr}(G)}{\Longrightarrow}$ if:

1. for each $i = 0, \ldots, n-1$ we apply a production for a node having the greatest) index in a graph g_i,
2. node indices do not change during a derivation.

A closed *rTLP* graph grammar rewriting graphs according to the regular right-hand side derivation is called a *closed rTLPO graph grammar*, abbrev. from *reverse Two-Level Production-Ordered*.

Now we introduce notions used for extracting handles in analyzed graphs which are matched against right-hand sides of productions during graph parsing.

Definition 4.4. Let g be an *rIE* graph, l some node of g defined by a characteristic description $n(l), r, e_1 \ldots e_r, i_1 \ldots i_r$. A subgraph h of the graph g consisting of node l, nodes having indices $i_{a+1}, i_{a+2}, \ldots, i_{a+m}, a \geq 0, a + m \leq r$, and edges connecting the nodes: $l, i_{a+1}, i_{a+2}, \ldots, i_{a+m}$ is called an *m-successors two-level graph originated in the node l and beginning with the* (i_{a+1})th *successor*. The subgraph h is denoted $h = m - TL(g, l, i_{a+1})$. By $0 - TL(g, l, -)$ we denote the subgraph of g consisting only of node l.

Definition 4.5. Let g be an *rIE* graph, l some node defined by a characteristic description $n(l), r, e_1 \dots e_r, i_1 \dots i_r$. A subgraph h of graph g consisting of node l, nodes having indices $i_{a+1}, i_{a+2}, \dots, i_r, a \geq 0$, and edges connecting the nodes $l, i_{a+1}, i_{a+2}, \dots, i_r$ is called a *complete two-level graph originated in node* l *and beginning with the* (i_{a+1})th *successor*. The subgraph h is denoted

$$h = CTL(g, l, i_{a+1}).$$

Let us impose the fundamental constraint which is analogous to that used in the definition of string Knuth's $LR(k)$ grammars [43]. It allows us to construct the efficient non-backtracking bottom-up parsing scheme.

Definition 4.6. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be a closed *rTLPO* graph grammar. The grammar G is called a $PR(k)$ abbrev. *Production-ordered k-Right nodes unambiguous, graph grammar* if the following condition is fulfilled. Let

$$Z \overset{*}{\underset{\mathrm{rr}(G)}{\Longrightarrow}} X_1 A X_2 \underset{\mathrm{rr}(G)}{\Longrightarrow} X_1 g X_2 \ ,$$

$$Z \overset{*}{\underset{\mathrm{rr}(G)}{\Longrightarrow}} X_3 B X_4 \underset{\mathrm{rr}(G)}{\Longrightarrow} X_1 g X_5 \ ,$$

and

$$k - TL(X_2, 1, 2) \overset{\mathrm{isom}}{=} k - TL(X_5, 1, 2) \ ,$$

where $\overset{*}{\underset{\mathrm{rr}(G)}{\Longrightarrow}}$ is the transitive and reflexive closure of $\underset{\mathrm{rr}(G)}{\Longrightarrow}$, A, B are characteristic descriptions of certain nodes, X_1, X_2, X_3, X_4, X_5 are substrings of characteristic descriptions, g is the right-hand side of a production: $A \longrightarrow g$.
Then:

$$X_1 = X_3, \ A = B, \ X_4 = X_5 \ .$$

The last restriction concerns the embedding transformation. The *edNLC* embedding transformation operates at the border between the production and its context. So, we do not have the context freeness property stated that reordering of derivation steps does not influence the result of a derivation. The lack of the order independence property, related to the finite Church-Rosser, fCR, property, results in the intractability of parsing. Thus, we have to restrict the power of the embedding transformation in order to obtain fCR and to guarantee the parsing efficiency. We make it by preserving the part of the production context unchanged during a derivation step.

Definition 4.7. Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be a $PL(k)$ ($PR(k)$) graph grammar. A pair $(b, x), b \in \Delta, x \in \Gamma$, is called a *potential previous context* for a node label $a \in \Sigma$, if there exists the *rIE* graph $g = (V, E, \Sigma, \Gamma, \phi)$ belonging to a certain regular left-hand (right-hand) side derivation in G that: $(k, x, l) \in E, \phi(k) = b$, and $\phi(l) = a$.

Definition 4.8. A $PR(k)$ graph grammar G is called an $ETPR(k)$, abbrev. from *Embedding Transformation-preserving Production-ordered k-Right nodes unambiguous, graph grammar*, if: for each production of the form

$$(l) \quad A \longrightarrow \begin{array}{cccc} X_1(1) & X_2(2) & \dots & X_m(m) \\ r_1 & r_2 & \dots & r_m \\ E_1 & E_2 & \dots & E_m \\ I_1 & I_2 & \dots & I_m \end{array}$$

$$\text{where } X_a \neq X_b, a, b = 1, \dots, m \ .$$

If (b, y) is a potential previous context for A, then there exists only one $(X_i, b, z, in) \in C_l(y, in), i \in \{1, \dots, m\}$, where C_l is the embedding transformation of the lth production. If $i = 1$, then $z = y$, i.e. $(X_1, b, y, in) \in C_l(y, in)$.

Let $G = (\Sigma, \Delta, \Gamma, P, Z)$ be the $ETPR(k)$ graph grammar. The language of G denoted $L(G)$ is the set

$$L(G) = \{H : Z \underset{\mathrm{rr}(G)}{\overset{*}{\Longrightarrow}} H \ \text{ and } H \in rIE_{\Delta,\Gamma}\} \ .$$

5. Parsing algorithm of $ETPR(k)$ graph languages

The general scheme of parsing for $ETPR(k)$ graph grammars [37] is a slight modification of the parsing schemes for $ETL(1)$ [28] and $ETPL(k)$ [32] graph grammars. It consists in a succeeding identification of a handle constructed on the basis of the property of an unambiguous choice of a production in the regular right-hand side derivation[6] according to Definition 4.6.

Let us introduce the following denotations and functions.

- G – an *rIE* graph to be analyzed represented by its characteristic description.
- H – the *rIE* graph represented by its characteristic description, which is being constructed (with succeeding reductions) during parsing on the basis of the graph G.
- *give_index*() – the function gives the succeeding index from the stack of node indices constructed according to Definition 3.7.
- *nonempty_indices_stack*() – the Boolean function gives *true* if the stack of node indices is nonempty.
- *give_handle*(H, i) – the function extracts the handle (in the form of its characteristic description) originated in the node indexed with i from the graph H according to Definition 4.6.
- *choose_production*(*handle*) – the function, on the basis of *handle*, identifies the proper production to be used for a reduction according to Definition 4.6.
- *reduction*(H, i, k) – the function performs the reduction to the node indexed with i in the graph H according to the production k.

We assume that the right-hand side graphs of the grammar are stored in the form of their characteristic descriptions.

Now, we can define the parsing algorithm 5.1.

[6] In $ETL(1)$ and $ETPL(k)$ graph grammars the corresponding property concerns the regular left-hand side derivation.

Algorithm 5.1 The parsing algorithm for $ETPR(k)$ graph grammar

$H := G$;
$err := 0$;
`while` $err = 0$ `and` $nonempty\_indices\_stack()$ `do`
`begin`
    $i := give\_index()$;
    $handle := give\_handle(H, i)$;
    $k := choose\_production(handle)$;
    `if` $k = 0$ `then` $err := 1$ `else` $reduction(H, i, k)$;
`end;`

In order to evaluate the time complexity of Algorithm 5.1 let us analyze the running times of its functions. Let n be the number of nodes of the graph G. The running time of the function $give\_index()$ operating on the stack of indices is $\sim n$. Extracting the handle with the help of the function $give\_handle(H, i)$, if we assume that H is represented with its characteristic description, is $\sim n$, as well. The running time of the function $choose\_production(handle)$ is bounded by the constant c which depends on the size of the grammar, i.e. the number of its productions and the maximum size of the right-hand size graph. (Thus, c does not depend on the size of the input graph G.) The function $reduction(H, i, k)$ is analogous to the function $production(H, i, k)$ of the $ETPL(k)$ parser introduced in [32]. Its running time is $\sim n$.

Now, we can formulate the following theorem.

Theorem 5.1. The running time of the parsing algorithm for $ETPR(k)$ graph grammar (Algorithm 5.1) is $O(n^2)$, where n is the number of the nodes of the analyzed *rIE* graph.

Proof. The `while` loop of Algorithm 5.1 is performed at most n times. Since the running times of all the functions inside the loop are bounded either by a constant or by n, the running time of the algorithm is $O(n^2)$.

6. Concluding remarks

The deterministic subclasses of *Node Label Controlled (NLC)* graph grammars for syntactic pattern recognition and computer vision have been studied for thirty years. They have been used in a variety of the real-world applications. The possibility of imposing the linear ordering on *EDG* graphs is one of two crucial factors resulting in the high efficiency of the model. (The balanced restrictions imposed on the embedding transformation of *edNLC* grammars is the second key factor.) In all the aforementioned applications of the model the linear ordering has been defined on the basis of semantic features of patterns

considered. However, the theoretical foundations of the scheme allowing us to index graph nodes based on pattern semantics have not been formulated till now. We have introduced them in the paper defining concepts of: relational structure, interpretation of *EDG* graphs over relational structure and interpreted *EDG* graph. These notions allow us to introduce the concept of (reverse) indexed edge-unambiguous graphs (*IE* and *rIE* graphs) in a formalized way.

The presentation of formal properties of $ETPR(k)$ graph grammars introduced preliminarily in [37] and their parsing algorithm has been the second goal of the paper. The $ETPR(k)$ parsing scheme is analogous to the $ETPL(k)$ one [32]. However, our experience with practical applications has revealed that there are some graph languages that cannot be generated by $ETPL(k)$ grammars and can be generated by $ETPR(k)$. This problem is worth further studying. Therefore, similarly as in case of $ETPL(k)$ grammars which are characterized from the point of view of descriptive power [33], the research into power properties of $ETPR(k)$ graph grammars will be carried out and the results obtained will be the subject of further publications.

References

[1] T. Pavlidis. *Structural Pattern Recognition*. Springer, New York, 1977.

[2] R.C. Gonzales and M.G. Thomason. *Syntactic Pattern Recognition: An Introduction*. Addison-Wesley, Reading, 1978.

[3] K.S. Fu. *Syntactic Pattern Recognition and Applications*. Prentice Hall, Englewood Cliffs, 1982.

[4] H. Bunke and A. Sanfeliu (eds.). *Syntactic and Structural Pattern Recognition – Theory and Applications*. World Scientific, Singapore, 1990.

[5] V. Claus, H. Ehrig and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science and Biology. *Lecture Notes in Computer Science* 73, 1979. doi:10.1007/BFb0025713.

[6] H. Ehrig, M. Nagl and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 153, 1983. doi:10.1007/BFb0000094.

[7] H. Ehrig, M. Nagl and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 291, 1987. doi:10.1007/3-540-18771-5.

[8] H. Ehrig, H.-J. Kreowski and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 532, 1991. doi:10.1007/BFb0017372.

[9] J. Cuny, H. Ehrig, G. Engels and G. Rozenberg (eds.). Graph Grammars and Their Application to Computer Science. *Lecture Notes in Computer Science* 1073, 1996. doi:10.1007/3-540-61228-9.

[10] H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg (eds.). Theory and Application of Graph Transformations. *Lecture Notes in Computer Science* 1764, 2000. doi:10.1007/b75045.

[11] A. Corradini, H. Ehrig, H.-J. Kreowski and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 2505, 2002. doi:10.1007/3-540-45832-8.

[12] H. Ehrig, G. Engels, F. Parisi-Presicce and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 3256, 2004. doi:10.1007/b100934.

[13] A. Corradini, H. Ehrig, U. Montanari, L. Ribeiro and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 4178, 2006. doi:10.1007/11841883.

[14] H. Ehrig, R. Heckel, G. Rozenberg and G. Taentzer (eds.). Graph Transformations. *Lecture Notes in Computer Science* 5214, 2008. doi:10.1007/978-3-540-87405-8.

[15] H. Ehrig, A. Rensink, G. Rozenberg and A. Schürr (eds.). Graph Transformations. *Lecture Notes in Computer Science* 6372, 2010. doi:10.1007/978-3-642-15928-2.

[16] H. Ehrig, G. Engels, H.-J. Kreowski and G. Rozenberg (eds.). Graph Transformations. *Lecture Notes in Computer Science* 7562, 2012. doi:10.1007/978-3-642-33654-6.

[17] H. Giese and B. König (eds.). Graph Transformations. *Lecture Notes in Computer Science* 8571, 2014. doi:10.1007/978-3-319-09108-2.

[18] F. Parisi-Presicce and B. Westfechtel (eds.). Graph Transformations. *Lecture Notes in Computer Science* 9151, 2015. doi:10.1007/978-3-319-21145-9.

[19] R. Echahed and M. Minas (eds.). Graph Transformations. *Lecture Notes in Computer Science* 9761, 2016. doi:10.1007/978-3-319-40530-8.

[20] Q.Y. Shi and K.S. Fu. Parsing and translation of attributed expansive graph languages for scene analysis. *IEEE Trans. Pattern Analysis Mach. Intell.*, 5:472–485, 1983. doi:10.1109/TPAMI.1983.4767426.

[21] H.O. Bunke and B. Haller. A parser for context free plex grammars. *Lecture Notes in Computer Science*, 411:136–150, 1990. doi:10.1007/3-540-52292-1_10.

[22] K.J. Peng, T. Yamamoto and Y. Aoki. A new parsing scheme for plex grammars. *Pattern Recognition*, 23:393–402, 1990. doi:10.1016/0031-3203(90)90026-H.

[23] K. Wittenburg, L. Weitzman and J. Talley J. Unification-based grammars and tabular parsing for graphical languages. *Journal Visual Languages Computing*, 2:347–370, 1991. doi:10.1016/S1045-926X(05)80004-7.

[24] F. Ferruci, G. Tortora, M. Tucci and G. Vitiello. A predictive parser for visual languages specified by relational grammars. In *Proc. IEEE Symp. Visual Lang. VL'94*, 245-252. doi:10.1109/VL.1994.363611.

[25] J. Rekers and A. Schürr. Defining and parsing visual languages with layered graph grammars. *Journal Visual Languages Computing*, 8:27–55, 1997. doi:10.1006/jvlc.1996.0027.

[26] D.Q. Zhang, K. Zhang and J. Cao. A context-sensitive graph grammar formalism for the specification of visual languages. *The Computer Jornal*, 44:186–200, 2001. doi:10.1093/comjnl/44.3.186.

[27] D. Janssens and G. Rozenberg. On the structure of node-label-controlled graph languages. *Information Sciences*, 20:191–216, 1980. doi:10.1016/0020-0255(80)90038-9.

[28] M. Flasiński. Parsing of edNLC-graph grammars for scene analysis. *Pattern Recognition*, 21:623–629, 1988. doi:10.1016/0031-3203(88)90034-9.

[29] M. Flasiński. Characteristics of edNLC-graph grammars for syntactic pattern recognition. *Computer Vision Graphics Image Processing*, 47:1–21, 1989. doi:10.1016/0734-189X(89)90050-9.

[30] M. Flasiński. Distorted pattern analysis with the help of Nodel Label Controlled graph languages. *Pattern Recognition*, 23:765–774, 1990. doi:10.1016/0031-3203(90)90099-7.

[31] M. Flasiński. Some notes on a problem of constructing the best matched graph. *Pattern Recognition*, 24:1223–1224, 1991. doi:10.1016/0031-3203(91)90147-W.

[32] M. Flasiński. On the parsing of deterministic graph languages for syntactic pattern recognition. *Pattern Recognition*, 26:1–16, 1993. doi:10.1016/0031-3203(93)90083-9.

[33] M. Flasiński. Power properties of NLC graph grammars with a polynomial membership problem. *Theoretical Computer Science*, 201:189–231, 1998. doi:10.1016/S0304-3975(97)00212-0.

[34] M. Flasiński. Inference of parsable graph grammars for syntactic pattern recognition. *Fundamenta Informaticae*, 80:379–413, 2007.

[35] M. Flasiński and L. Kotulski. On the use of graph grammars for the control of a distributed software allocation. *The Computer Journal*, 35:A165–A175, 1992.

[36] M. Flasiński. Use of graph grammars for the description of mechanical parts. *Computer-Aided Design*, 27:403–433, 1995. doi:10.1016/0010-4485(94)00015-6.

[37] M. Flasiński and Z. Flasińska. Characteristics of bottom-up parsable edNLC graph languages for syntactic pattern recognition. In L.J. Chmielewski et al., editors, *Computer Vision and Graphics: Proc. Int. Conf. ICCVG 2014*, volume 8671 of *Lecture Notes in Computer Science*, pages 195–202, Warsaw, Poland, September 2014. Springer, Heidelberg. doi:10.1007/978-3-319-11331-9_24.

[38] U. Behrens, M. Flasiński, L. Hagge and K. Ohrenberg. ZEX – an expert system for ZEUS. *IEEE Trans. Nuclear Science*, 41:152–156, 1994. doi:10.1109/23.281478.

[39] M. Flasiński, R. Schaefer and W. Toporkiewicz. Supporting CAE parallel computations with IE-graph solid representation. *J. Geometry Graphics*, 1:23–29, 1997.

[40] M. Flasiński. *Introduction to Artificial Intelligence*. Springer International, Switzerland, 2016. doi:10.1007/978-3-319-40022-8.

[41] M. Flasiński and S. Myśliński. On the use of graph parsing for recognition of isolated hand postures of Polish Sign Language. *Pattern Recognition* 43:2249–2264, 2010. doi:10.1016/j.patcog.2010.01.004.

[42] M. Flasiński. Syntactic pattern recognition: paradigm issues and open problems. In C.H. Chen (ed.), Handbook of Pattern Recognition and Computer Vision, World Scientific, New Jersey – London – Singapore, 2016, Chapt. 1, pp. 3-25.

[43] D.E. Knuth. On the translation of languages from left to right. *Information Control* 8:607–639, 1965. doi:10.1016/S0019-9958(65)90426-2.

Isocontouring with Sharp Corner Features

Sui Gong, Timothy S. Newman
Department of Computer Science, University of Alabama in Huntsville, Huntsville, USA
sg0010@uah.edu, tnewman@cs.uah.edu

Abstract. A method that achieves closed boundary finding in images (including slice images) with sub-pixel precision while enabling expression of sharp corners in that boundary is described. The method is a new extension to the well-known Marching Squares (MS) 2D isocontouring method that recovers sharp corner features that MS usually recovers as chamfered. The method has two major components: (1) detection of areas in the input image likely to contain sharp corner features, and (2) examination of image locations directly adjacent to the area with likely corners. Results of applying the new method, as well as its performance analysis, are also shown.

Key words: marching squares, feature preservation, corner recovery, contour finding, isocontours.

1. Introduction

In this paper, we describe an improvement for a popular means to find a closed boundary of a region of constant value (i.e., intensity or activity) in an image. This improvement can be considered to be an extension of the popular Marching Squares (MS) isocontouring method for 2D scalar fields (N.B. We have previously briefly described this new extension's key features in a conference report [9]). An isocontour can be defined as follows. Given a scalar field $f(x, y)$ (for example, f may represent the abstract function describing the densities captured in an X-ray), the isocontour is the collection of locations in the field having a particular scalar value α (e.g. the (x, y) locations where $f(x, y) = \alpha$). The scalar value α is the isovalue associated with (i.e., giving rise to) that isocontour. Isocontouring is a strategy often employed in processing or analyzing many types of data organized on grids. The most prevalent class of scalar 2D grid data is probably data arranged on rectilinear grids (such as X-ray images, slice images of CT datasets, intensity images, some planar simulation outputs, etc.). Other popular grid types include triangular and hybrid/adaptive grids [18, 29]. Here, our focus is on isocontouring for scalar data arranged on a rectilinear grid. Isocontouring is a very valuable technique for such data as it can, in one integrated process, define a closed boundary, with sub-pixel precision, of an area associated with some fixed level of activity (e.g. density).

There are a number of algorithms for finding isocontours in datasets of 2, 3 and more dimensions arranged on a grid. In determining the isocontour on these types of datasets, the existing algorithms treat the data values as samples from an underlying scalar field; such isocontouring algorithms form an isocontour in consideration of the samples. It is

often useful to find an approximate description of the boundaries of phenomena or structures in image processing and analysis applications. For example, boundary delineation can be useful for location-finding tasks, such as defect detection and front tracking. Also, isocontouring can be used to enable estimation of boundary or region properties (e.g. perimeter or area). In such uses, especially where a structure boundary is desired, the produced isocontour is sometimes said to be a reconstruction [17]. Other uses include as components of feature extraction [27] and segmentation [15] methods. Isocontouring can also be integrated into contour extraction frameworks [26]. Isocontours also have other uses, including discovery in terrain data [1, 12], simulation analysis [6] and fluid surface tracking [19].

The MS algorithm has been widely applied in 2D rectilinear grid data isocontouring. MS produces a piecewise linear approximation of the isocontour, as described in detail in Section 2. However, when applied in cases where the actual boundary has sharp corners, MS instead produces mostly blunted corners. This behavior of MS creates a problem for contour recovery when sharp corners are present. Producing a contour that recovers sharp corner features can improve renderings as well as assist in registration and pattern recognition. Thus, it is important for isocontouring to recover them correctly. However, previously there has not been a 2D isocontouring method that is able to correctly recover sharp corners. In this paper, we describe our extension to the Marching Squares that improves on that situation; our approach allows producing an isocontour that has sharp corner features. This extension enables better expression of actual boundary shape for boundaries containing such features.

This paper is organized as follows. Background is discussed in Section 2. Related work is described in Section 3. In Section 4, we present our extension to Marching Squares that enables construction of an isocontour that includes sharp corners. In Section 5, we provide results from applying the extended algorithm and comparisons with standard MS. The conclusion and future work are presented in Section 6.

2. Background

Marching Squares takes a 2D rectilinear grid of scalar data as its input. Such inputs could be X-ray images, individual slice images of volumetric datasets, infrared (IR) images, etc. The rectilinear grid is treated as a collection of grid cells by MS. In the case of images, each pixel value is treated as the data value at a grid point. The MS algorithm produces an isocontour with sub-cell accuracy by *marching* through the dataset in sequential order, processing one grid cell at a time until all grid cells have been processed. In each of the cells, the algorithm's processing involves a series of steps that determine if any isocontour segments need to be generated within the cell.

The first step MS follows in each cell is to compare each grid point's data value with the isovalue α. MS marks each data point with a value greater than or equal to α

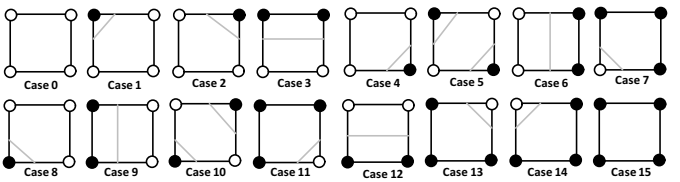


Fig. 1. 16 topological cases of cells in Marching Squares.

with a "1." It marks the others with a "0." Grid edges marked with a "1" at one end point and a "0" at the other are intersected by the isocontour.

There are $2^4 = 16$ possible markings for a cell since each grid cell has four grid points. Each unique marking type is called a topological case and defines a particular isocontour topology. The 16 cases and the general form of the contour segment(s) produced by MS for each case are shown in Fig. 1. In the figure, data points with a "1" marking are indicated by filled (black) grid points. Data points with a "0" marking are indicated by hollow (white) grid points. Typically, MS encodes the 16 possible cell cases in a look-up table before processing the dataset. For each case, that table records the identity of intersected edges and how the intersected edges are connected by the isocontour segments. When a look-up table is used, the second step of processing in the cell is to use the markings to determine the topological case for the cell.

Next, for each cell MS calculates positions of any isocontour intersections with the cell's grid edges. If a look-up table is used, the identities of intersected edges are retrieved directly from the table. For each such edge, linear interpolation on the values at the end points of the edge is used to find the intersection location.

Finally, in each cell, the isocontour is formed. In this step, MS connects each pair of intersection locations by a line segment. If a look-up table is used, the identities of the edges that are to be connected are found from it. The collection of such line segments defines the isocontour.

We note that two of the MS cases (i.e., Cases 5 and 10 in Fig. 1) can be contoured in a variant way from what is shown in Figure 1 (since there are two ways to connect each set of four intersections such that the contour does not self-intersect). Since triangular cells do not have this issue of variant contours (due to there being only one choice for each cell in a triangular mesh), in some cases isocontouring of rectilinear grid data is done by an analogue of MS that triangulates the cells before isocontouring (i.e., by dividing each cell into two triangles and then performing isocontouring in each triangle). This analogue can be called *Marching Diagonally Divided Squares* (MDDS).

One of MS's potential problems is that when sharp corner features occur in an actual boundary (e.g. of a structure), the isocontour that MS produces usually has chamfered corners at the places where the actual boundary has sharp corners, unless the location

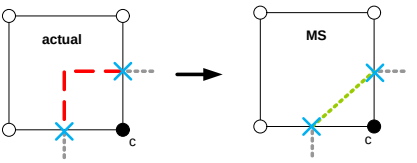


Fig. 2. A sharp corner of a boundary (**left**) that MS chamfers (**right**).

of the actual corner is very close to the edge of a grid cell. Fig. 2 shows an example of the problem. In the figure, a grid cell with three grid points whose values are less than the isovalue (i.e., denoted as hollow (white) circles in the figure) and one grid point whose value is greater than the isovalue (i.e., denoted as a filled (black) circle in the figure) is shown. The contour shown at the left (as a dashed line) represents the actual phenomenon boundary, which has a sharp corner feature. Such cells are recognized by the standard MS as Case 2 cells (using the numbering as in Fig. 1). The Case 2 topology specifies two grid edge intersections, and MS performs linear interpolation on these edges to find the intersection locations. For the Fig. 2 example, the $\times$ marks show the locations. MS connects the two intersections with a single line segment, as shown in the contour on the right (as a dotted line) of Fig. 2. The *corner* produced here by MS does not have the sharp shape of the actual boundary; instead, MS produces a chamfered corner.

The analogue of MS that triangulates each rectilinear grid cell before isocontouring in that triangulation can reduce the degree of chamfering for some corners. However, there are several costs for that benefit. First, the upfront triangulation of cells imposes an additional computational burden. Second, for an actual contour with a long, straight run that is diagonally-oriented, the contour produced for that *run* can be staircased. Third, the contour intersection locations on the diagonal edge segments of the triangulation differ from segment positions in MS since the analogue uses a different interpolation than the axis-aligned bilinear interpolant used by MS on the underlying rectilinear grid. Fourth, the total number of segments is usually significantly increased.

MS is often applied on many types of 2D grayscale images as a boundary-finding method (e.g. for segmentation [2, 29] or silhouette determination [11]). Compared to using contours or boundaries based on traditional, popular edge detectors, like the Sobel edge detector [10] (that typically produce edges that pass through pixel locations where the highest gradient changes are detected), using MS for boundary-finding has certain advantages. First, traditional edge detectors can produce a boundary that is not continuous (i.e., some individual edge pixels can be isolated (i.e., disconnected) from others), rather than a guaranteed continuous water-tight contour, as MS does. While edge detectors can be coupled with edge linking algorithms (e.g. such as [3, 17]) on the disconnected edge segments, the boundaries may still not be continuous. If the area needing the boundary description is known, one historically popular alternative approach to

determine a boundary description has been chain coding [7] (or its variants). However, in chain coding, the segment endpoints have pixel-level precision only. In comparison to traditional edge detection with linkage or chain coding, MS produces a contour that has sub-pixel-level precision. In addition, the MS contour segments have many possible orientations (limited only by the finite precision of machine floating-point arithmetic). In contrast, the elements of a chain-coded boundary can have only 8 possible orientations. Finally, the MS boundary is associated with a specific activity (or intensity) level whereas boundaries from edge linkages may not be associated with one activity (or intensity) level.

The boundary MS produces has been used in many applications, such as a part of processing to find the contour of a pelvic region in CT data [8], for generating toolpaths in a 3D printing application [28], and in a set relations visualization scheme [4]. Other than boundary finding, MS has been combined with methods for image segmentation [3] or with methods based on Minkowski functionals for image analysis [16]. In addition, MS has been used in a color distinguishability study to extract features such as stripe outlines from images [23]. MS can also be used in contour length estimation [3], since the contour segment produced for each cell is easy to calculate. A parallelization scheme for MS has also been described [20].

3. Related work in feature preserving isocontouring

Isocontouring that maintains certain shape features in the boundary it produces is called feature-preserving isocontouring. Next, some such existing approaches are described. These existing approaches are aimed at true 3D isocontouring (on volumetric data), usually by extension of the Marching Cubes (MC) Algorithm. MC can be viewed as a generalization of MS to volumetric data. Its isocontour is a surface called an *isosurface*. MC is very well-known [21, 22]. Like MS, MC can produce an isocontour that lacks certain "sharp" features, even if the isocontour represents the boundary of an underlying phenomenon or structure that does have such features.

One approach to enable certain sharp features to be preserved in an isosurface was described by Kobbelt et al. [14]. In their approach, after the isovalue is determined, the original dataset is converted into a directed distance field that stores three distance values at each grid point, namely the shortest distances in the x, y and z directions from the point to an isosurface component.

Then, Kobbelt et al.'s approach performs MC-like processing on the distance field as follows. First, a basic octree is built whose leaf nodes each store one 3D grid cell. The octree also records which cells are candidates for containing sharp features, as described later. The octree building starts from the leaf nodes. A merging process forms higher levels of the tree by merging some leaf nodes with their parent (if certain rules are satisfied). One such rule, for example, involves considering the current isocontour

position in lower level nodes versus the isocontour position in the next higher level node (i.e., formed by merging the lower level nodes). If the difference between these positions is too large, no merging takes place. Merging of non-leaf nodes with their parents occurs if they have particular values and all their children can also be merged. Following the merging process, nodes that haven't been merged are the candidate nodes (for containing sharp isocontour features). Grid cells within merged nodes are processed by standard Marching Cubes (since no sharp corner is likely present). In all cells of candidate nodes, the algorithm produces a different isocontour than MC; it creates additional triangular facets that could allow a sharp corner shape to be produced. These facets are connected to the ones generated in the nearby cells to avoid "holes" occurring in the isocontour.

Another feature preserving approach for 3D data is the fine feature recovery approach of Kaneko et al. [13]. By recovering fine features, they mean that the isocontour includes the thin or narrow parts of the actual underlying object or structure in a volumetric dataset. In the isocontour produced by standard MC, these types of features could be smaller or larger than they actually are. The Kaneko et al. approach first produces the standard MC contour. Then, using this contour, it estimates what the dataset values at each grid point location would need to be if the produced isocontour was the actual boundary. At each grid point, it then compares these estimates with the actual data value. Whenever such values differ significantly, the approach considers the recovered isocontour segments near this grid point to be in need of adjustment. It follows a two-step process to do that. First, it adds or subtracts a constant to the value at that grid point, producing an *adjusted dataset*. Then it produces an isocontour on the adjusted dataset using standard MC. The isocontour produced from the adjusted dataset encloses a region that has a volume that better resembles the volume enclosed by the actual boundary [13].

These two methods, however, are extensions of 3D isocontouring. They have not been extended to address sharp corner recovery in 2D isocontouring. In addition, while inspiring, the Kobbelt method has some overhead (i.e., it must produce the distance field prior to isocontour recovery), which can be time consuming. Kaneko's method, on the other hand, has a primary focus on fine feature preservation instead of sharp corner preservation. While sharp corners can sometimes also be fine features (e.g. when they are narrow and long), in most cases they are not fine enough for Kaneko's method to recover.

Our method described here extends the state-of-the-art for 2D isocontouring by allowing sharp corner recovery in 2D isocontouring.

4. Methods: Corner Feature Expressive Marching Squares

Our new algorithm is a feature-preserving approach that extends the Marching Squares to allow expression of sharp corner features. In this section, a step-by-step elaboration of our algorithm and an enumeration of its topological cases are described.

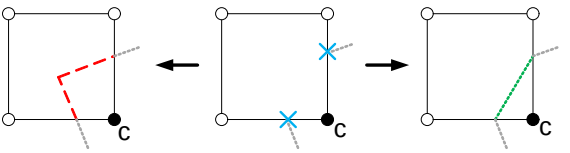


Fig. 3. Two possible contours. (**Left**) contour with a sharp corner feature; (**right**) contour with a chamfered corner feature; (**middle**) with the same intersection locations.

4.1. Corner Feature Expressive Marching Squares: motivation

Our corner feature expressive algorithm can produce a contour with sharp corners using information from small, local regions surrounding the corner. We motivate the algorithm's approach next. Later in the section, the specific, step-by-step processing is laid out.

As discussed in Section 2, the standard MS has, for each topological case, a single predefined way to connect the points of intersections of the isocontour with the grid lines. Thus, no matter what type of contour shape is actually present in a given cell, the predefined contour shape for that cell's topological case is the only type of shape that can be produced for the isocontour for the cell. An example of this situation is shown for one cell (of topological Case 4) in Fig. 3. For the case of this figure, the isocontour has a segment below the cell, which starts from the middle of the lower edge and extends down and to the right, and a segment on the right of the cell, which starts in the middle of the right edge and extends above and to the right (i.e., these parts of the isocontour are shown as dotted lines in the center part of the figure). The lower right lattice point of this cell is labeled as C. The value at that lattice point is used in determining both of the isocontour's intersection locations with the cell. Here, those points are marked with "$\times$" marks in the center part of the figure. The contour MS produces is shown as a dotted line on the right. This contour has a chamfered corner, even though the true contour shape could include a sharp corner, like the one shown on the left of this figure. Using only the locations of the isocontour intersections with the grid lines bounding the cell, it is not possible to tell whether the shape of the contours inside the grid cell contains a chamfered corner shape or a square corner shape.

Our algorithm exploits the contour segment orientations in the cells adjacent to possible corners to estimate contour shape and corner position in cells possibly containing a corner. In particular, it first considers if the contour segments in the adjacent cells provide clues suggesting if there could be a sharp corner.

An example of two sorts of clues that neighboring cells can provide is shown in Fig. 4. In the middle part of this Figure, two types of contours with different corner shapes are shown for one cell. Beside that, two possible arrangements of cells neighboring the one in

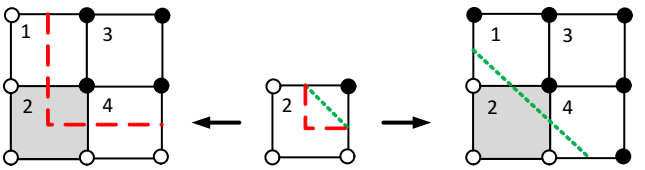


Fig. 4. Two possible arrangements of cells neighboring one grid cell.

question are shown. The cell in question is shaded and numbered "2." The neighboring cells in this example are labelled "1," "3" and "4." The actual contour shapes are also shown for each arrangement. In both arrangements shown here, the cell in question has the Case 2 topology. One arrangement features a diagonal edge (perhaps part of a chamfered corner), as shown by the dotted line. The other one features a sharp corner, as shown by the dashed segments. Since the grid cell in question has a Case 2 topology, the isocontour in it will be produced as a diagonal edge in standard MS, regardless if there is in fact a chamfered corner there. However, here the neighboring cells can provide some clue about the likely corner type in the cell. For the arrangement shown at the left, the dashed contour in the cells labeled "1" and "4" provides a clue that there may be a sharp corner feature in the shaded cell. For that arrangement, cells "4" and "1" have the Case 3 and Case 6 topologies, respectively. For such scenarios, a contour similar to the square corner shown in the left arrangement seems more credible to many human observers. In contrast, for the arrangement shown at the right, the dotted contour in the cells labeled "1" and "4" does not provide such a clue. For that arrangement, cells "1" and "4" have Case 7 topologies, and the contour produced by MS has a suitable shape for the cell in question.

4.2. Corner Feature Expressive Marching Squares: elaboration

In keeping with such reasoning, our extended Marching Squares considers cells adjacent to potential corner-containing grid cells to determine situations for which sharp corners are credible. Our approach follows processing similar to the illustration shown for the situation in Fig. 4; cells likely to contain a part of the boundary that has a sharp corner are found by considering the neighborhood about the cell. We call such possible corner-containing cells *candidate corner cells*, and we call neighborhoods about the corner candidates *grid cell groups*. Each grid cell group consists of the candidate corner cell and two cells adjacent (i.e., that are 4-neighbors) to that one. Specifically, the two adjacent cells considered are the ones that the contour enters from the possible corner-containing cell. (Due to the way MS finds isocontour-cell intersections, the adjacent cells can only be horizontally or vertically adjacent to the candidate corner cell.) We call those two adjacent cells the adjacent neighbor cells.

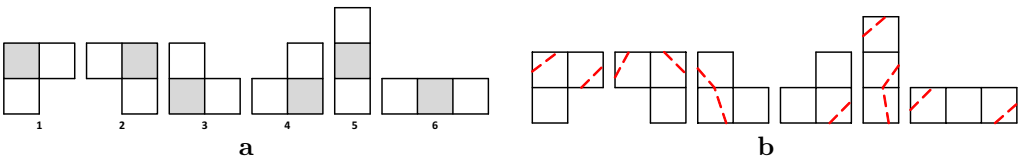


Fig. 5. Grid cell group layouts and insufficient groups. (**a**) Enumeration of grid cell group. (**b**) Insufficient (grid cell) groups.

There are 6 possible layouts (forms) for each grid cell group, as shown in Fig. 5a. We call each one a grid cell group layout, and we label them as Layouts 1 through 6. In Fig. 5a, the candidate corner cell for each layout is shaded. The other two grid cells that are not shaded are its adjacent neighbor cells. Each grid cell group has a total of eight grid points and ten unique edges. Since each grid point can either be marked as "1" or "0", there are $2^8 = 256$ possible different combinations of markings. However, in corner detection processing, it is not necessary to consider such a number of combinations since some of the combinations do not contain a corner (e.g. the combinations with all 0 or all 1 markings have no cells intersected by the isocontour).

Grid cell groups that include diagonally-adjacent cells could potentially be used, but we do not consider them here, since by themselves they often do not provide additional credible evidence of corners. But, if a larger set of nearby cells was considered, these larger groups of grid cells could be useful for corner detection. We leave corner detection and sharp corner contour production for such groups to future work; we focus here on a methodology that is applicable to grid cell groups of size 3. In the work here, evidence from size 3 grid cell groups is used to detect where sharp corner features are likely and then to produce contours containing such features. We also focus only on grid cell groups that have sufficient neighbor information.

Fig. 5b shows six example grid cell groups with insufficient neighbor information. In it, each group contains at least one part of a contour that might in fact be a sharp corner but the candidate corner cells in the group do not have two adjacent neighbors that both contain continuations of the contour segment from the candidate corner cell. We term such grid cell groups without enough neighboring information as *insufficient groups*. We call grid cell groups that have enough neighboring information *sufficient groups*.

Some sharp corners in insufficient groups are in fact detected and produced correctly by our method. This detection occurs due to a nearby sufficient group that includes some grid cells that are also in the insufficient group. An example of such a situation is shown in Fig. 6b. In this figure, there are three grid cell groups that are highlighted (using an outline feature or by gray shading). One of them (the gray shaded one) can be considered as an insufficient group. The other two (shown as the solid outline and dashed outline) are sufficient groups. The insufficient grid cell group contains two corner

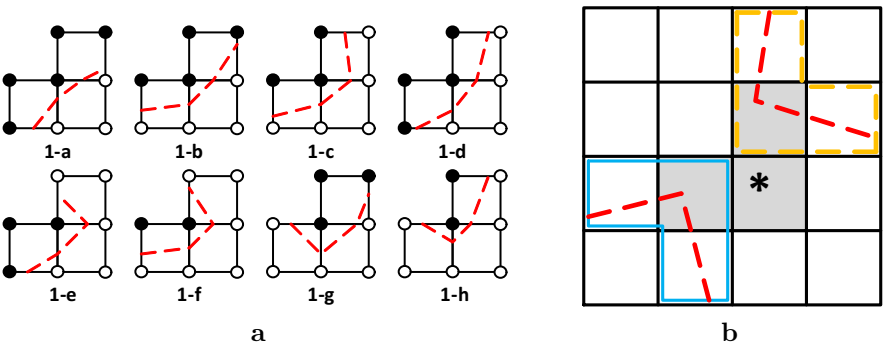


Fig. 6. Examples of group topology and insufficient/sufficient groups. (**a**) Group topology list for Case 1 candidate corner cell. (**b**) Corner features detectable from the sufficient groups (dashed outlined and solid outlined) but not in the insufficient group (shaded).

features, but neither occurs at its candidate corner cell (marked by "$*$"). However, each corner feature can be detected via one of the two (outlined) sufficient grid cell groups since they offer credible evidence that a corner could be present.

We examined all of the 16 topological cases for MS and determined the grid cell group topologies that could contain sharp corners. Our algorithm stores such cases in a group topology list. In this list, each group contains the three grid cells making up a grid cell group. These are the candidate corner cell and its opposing neighbor cells. Each MS topological case, except Cases 0 and 15, can give rise to a candidate corner cell. Thus, for the Case 1 through 14 topologies, we examined all of the 4-neighbors of candidate corner cells and the contour segments in them.

We list some examples of the group topologies in Fig. 6a. In this figure, each group topology is labelled with a combination of a number and a letter. The number indicates the case topology of its candidate corner cell in the original MS look-up table (from Fig. 1), and the letter indicates our subcase identifier. For example, in Fig. 6a, we list all 8 subcases for candidate corner cells with Case 1, labeled as 1-a through 1-h. The complete group topology listing for all the cases is shown in Appendix A, and that listing is labelled with the same scheme as in Fig. 6a.

While our method can produce a contour in the candidate corner cell that differs from what standard MS produces for that, it does not vary the contour shape in the other cells of the grid cell group. To detect if the contour in a candidate corner cell is likely to contain a sharp corner, we first determine the orientations of the contour segments in the adjacent neighbor cells of its grid cell group. Then the angular difference between these orientations is calculated. If that difference is close to 90 degrees, our method marks the

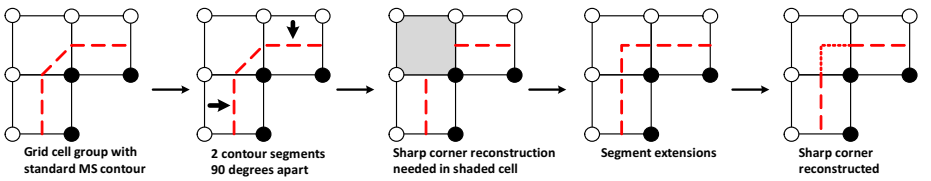


Fig. 7. Corner detection and production process.

cell as one which likely has a sharp corner feature. Otherwise, when the angle between two line segments in the neighbors is not sharp, we classify the cell as a non-corner cell and use the standard MS rules to produce the isocontour segments. In addition, all other non-corner cells are produced according to the standard MS rules.

For candidate corner cells that likely have a sharp corner, a two-step process is used to produce the corner. First the corner location is determined as the point where the contour segments in the adjacent neighbor cells intersect. Then, contour segments are formed connecting this point to the locations where the contours intersect cell edges.

Fig. 7 shows an example of the corner construction process for a grid cell group with a Case 4 candidate corner cell. In the adjacent neighbor cells, the two contour segments form an angle of 90 degrees. In this case, due to the size of the angular difference, our method determines that the candidate corner cell contains a sharp corner feature (i.e., thus, for that cell, we do *not* use standard MS rules to produce its contour). Once a likely sharp corner has been determined, contour segments in opposing neighbor cells are extended, as shown at the right part of the figure. The final result is the contour with the sharp corner shown in the last panel of the figure.

4.3. Corner Feature Expressive Marching Squares: algorithm

A step-by-step elaboration of our algorithm is shown in the listing labelled Algorithm 4.1.

5. Results and discussion

Next, some results of applying our algorithm in images are presented. Results of synthetic images are presented first, followed by images from volumetric datasets, and then X-ray images at last. For comparison, results for standard MS are also exhibited.

Our synthetic images are density/occupancy images of objects with density 100 in a 16×16 rectilinear grid. The background density is 0. At each grid lattice point, the data value is the aggregate density over a unit-sized square region centered at that lattice point. Thus, any lattice point representing a fully occupied unit-sized region in space

Algorithm 4.1 Corner Feature Expressive Marching Squares.

```
for each grid cell in the dataset do
    determine its topological case
    if it is not Case 0 or Case 15 then
        locate its two adjacent neighbor cells according to the group topology list
        calculate the contour segment orientations in the neighboring cells
        if they form a sharp angle then
            extend the contour segments of the adjacent neighbor cells into this cell to
produce a sharp corner
        else
            apply standard MS on the cell
        end if
    else
        apply standard MS on the cell
    end if
end for
```

has a data value of 100 associated with it. The objects are axis-aligned in some of the images, but not in others. We can determine the error in an isocontouring method's result using these images because the actual boundaries in each of them are known.

A comparison of results from standard MS and our algorithm for synthetic images of an L-shaped block, star-shaped object (in two orientations), and a ninja star-shaped object is shown in Fig. 8. In this figure, the dashed contours are the results produced by our algorithm and the dotted contours are the results produced by standard MS. The solid black contours are the actual object boundaries. The parts of the boundary that lack sharp corner features yield overlapping contours from standard MS and our algorithm.

We also show a result from standard MS, our algorithm, and MDDS in Fig. 9b. In this figure – and for Fig. 10, 11 and 12 – we use the same contour drawing patterns as used in Fig. 8, except we periodically overlay circle glyphs on the dashed results (of our approach). In addition, the result of MDDS is drawn in a dash-dot pattern in the Fig. 9b. Our approach recovered four sharp corners while MDDS recovered none. MDDS also recovered some of the straight edges as wavy edges. Since MDDS does not produce a competitive result for sharp corner recovery, it is not shown in other result images in this paper.

Our new algorithm recovered many, but not all, of the sharp corners in these images. In particular, if a corner feature spans a few cells, the new algorithm can have difficulty to fully recover all corners. Expanding the size of the grid cell groups may improve matters, although possibly at a cost of false positives.

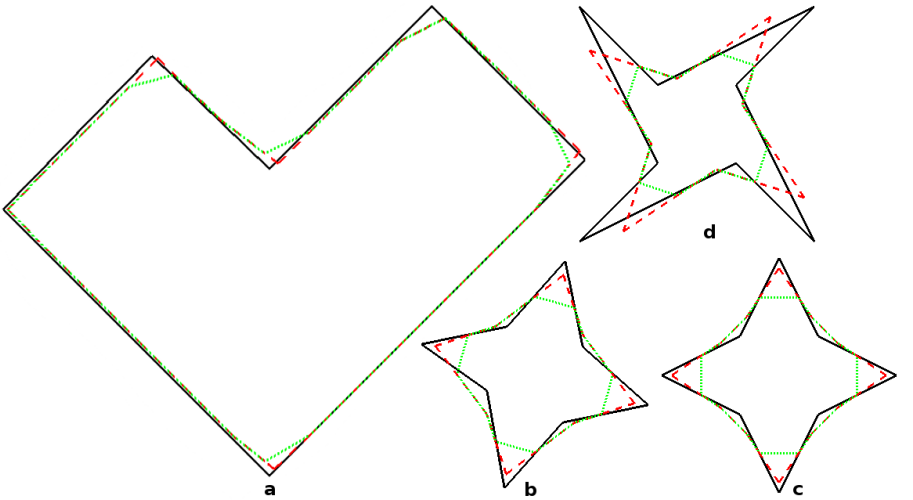


Fig. 8. Synthetic dataset of object isocontouring results for two approaches. (**a**) L-shape oriented at 45 degree angle with the x-axis. (**b**) A star object oriented at 15 degree angle with the x-axis. (**c**) A Star object oriented at 0 degree angle with the x-axis. (**d**) A ninja star object.

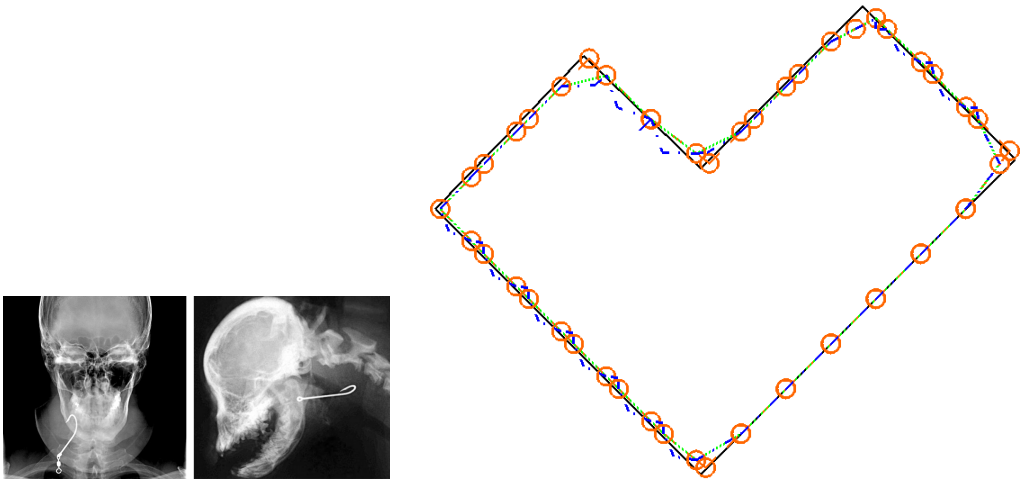


Fig. 9. Some applications. (**a**) X-ray images of: (left) hook embedded in a human face, (right) hook caught on a dog. (**b**) Example results.

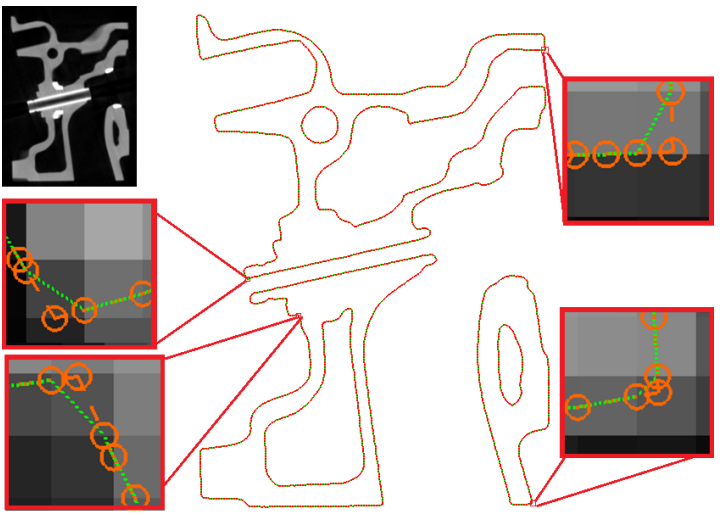


Fig. 10. Engine Block dataset slice image isocontouring results. (**Upper left corner**) the slice image.

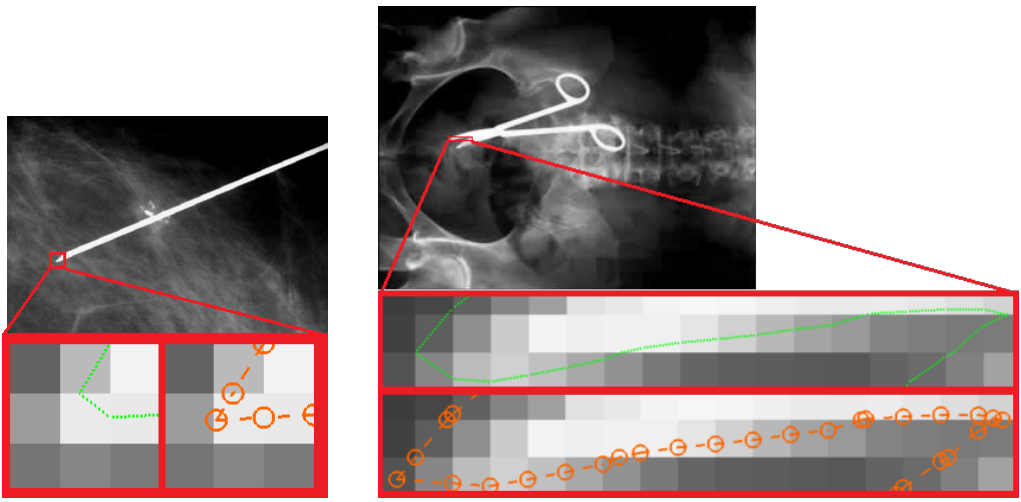
Next, we describe results from applying our algorithm on real images. First, results for one slice image (size 256×256) of the well-known Engine Block dataset (from the Volume Library [25]) are shown in Fig. 10 using the line marking pattern described earlier. In this figure, zoomed-in call-outs are shown for several corners. The gray scale image of this slice of data is also shown at the upper left of the figure as a reference.

We applied our algorithm and standard MS on 10 of the slices of the Engine Block dataset and counted the number of sharp corners recovered by each algorithm. The corner counts are presented in Tab. 1. On average, our algorithm recovered 10 sharp corners more than standard MS did.

Results for applications to some X-ray images are shown in Fig. 11, which includes comparison results for standard MS versus our algorithm for two images containing objects with sharp corners (from radiopaedia.org [5]). The Fig. 11a image is of a biopsy needle is size 220×320. The Fig. 11b image is of scissors lodged in a human and is size 320×240. We call these images Needle and Scissors, respectively. For each image,

Tab. 1. Number of sharp corners recovered by our algorithm but not by standard MS.

Image No.	1	2	3	4	5	6	7	8	9	10
Sharp Corners	13	12	10	15	10	9	14	7	4	6



a **b**

Fig. 11. X-ray image and isocontour comparison results: (**a**) biopsy needle, (**b**) scissors lodged inside human.

a zoomed-in call-out of an area containing a sharp corner feature is shown, with the result of standard MS (in dotted segments) side-by-side with our algorithm result (in dashed segments). In these images, the new algorithm has produced a sharper *point* for the needle and scissors objects than standard MS has. We also applied both algorithms on four images (labelled Hook1, Hook2, Hook3, and Hook4) of hooks embedded in tissue, two of which are shown in Fig. 9a.

Some results for applying our algorithm to three range images of block or box objects containing sharp corners are shown in Fig. 12. These images are from the OSU (MSU/WSU) range image database [24] and include a set of rectangle blocks (called Blox2), a set of cylindrical blocks (called Blox3), and three stacked boxes (called Stack). The Blox2 and Blox3 images are size 240×240. The Stack image is size 128×128. For each image, we exhibit several zoomed-in call-outs of areas containing sharp corner features, showing both the result of standard MS (in dotted segments) and our algorithm (in dashed segments) shown. In these images, the new algorithm has produced many sharp corners while MS has failed to produce them.

5.1. Empirical Error Studies: Our Algorithm vs. Standard MS

We have also used synthetic images in empirical tests of our algorithm and standard Marching Squares to study contour accuracy in a quantifiable way. In these images, there

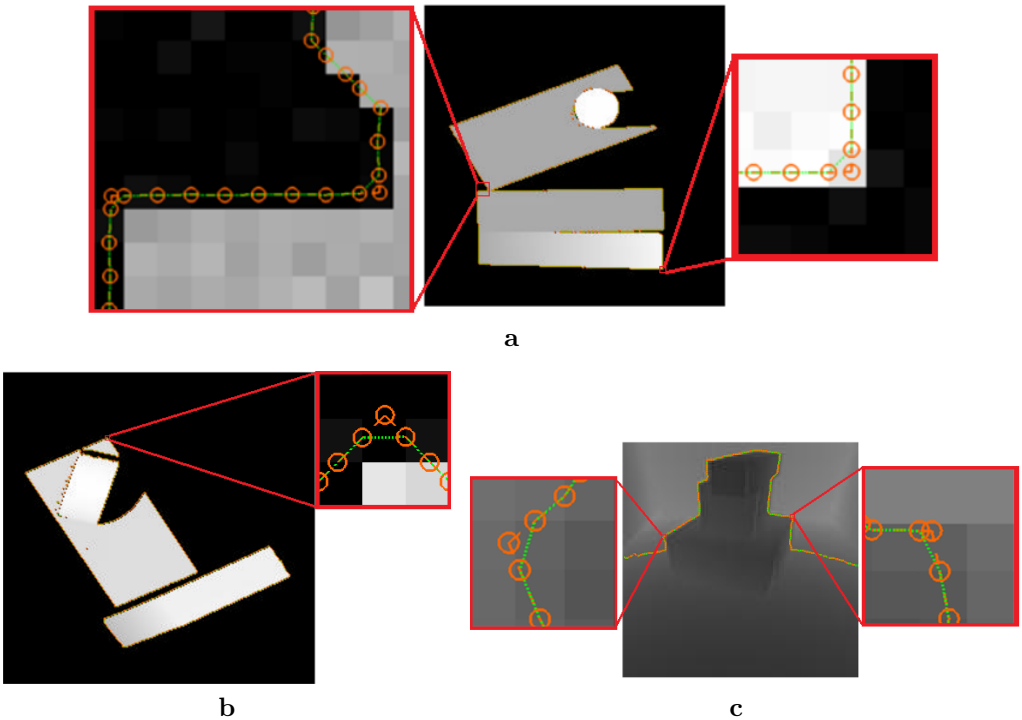


Fig. 12. Range images and isocontour comparison results. (**a**) Blox2. (**b**) Blox3. (**c**) Stack.

are sharp corners that have an internal angle very close to 90 degrees. In determining accuracy for the tests, the lower left grid point of each cell was taken as a local coordinate system origin of a cell assumed to be size 1×1. A number of test scenarios were used in the experiments, with the sharp corner locations varied in each scenario. Four methods were used to estimate the error in the contours. Those methods and outcomes from using them are presented next.

5.1.1. Average Closest Euclidean Distance Error Measure

For each test case, we determined the smallest Euclidean distance to the actual boundary from points sampled from the recovered contour. The average of these distances was taken as the error E_D in the recovered contour for that test case:

$$E_D = \frac{1}{N} \sum D(i)\,, \tag{1}$$

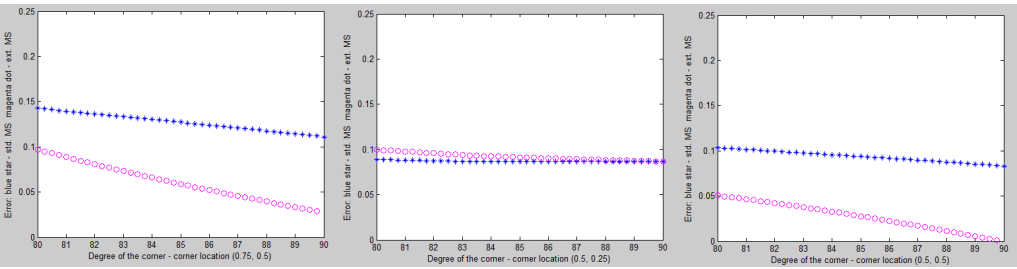


Fig. 13. Average closest Euclidean distance error measure, E_D, for new algorithm (circles) and standard MS (stars).

where N is the number of sample points, and $D(i)$ is the closest distance from the i-th sample point on the recovered contour to the actual boundary.

Plots of such errors for a range of test cases are shown in Fig. 13. These plots consider test cases for three distinct corner locations (the center of the cell and two other points on the diagonal of a cell, as indicated in the plot captions). For each plot, the x axis indicates the degree of the corner, and the y axis indicates the error for each case. The stars show error for standard MS, and the circles show error for the new algorithm. For most cells with sharp corners, the new algorithm yielded the lower error. Its improvement appears to be best when corner angles are close to 90 degrees.

5.1.2. Corresponding Points Error Measure

Another error measurement we considered was E_{DC}, an average distance measure computed by averaging distances between corresponding points on two contours. Since standard MS does not usually produce a sharp corner feature, we determined the distance measure's component for the corner by taking the mid point of MS's contour as the point corresponding to the corner of the actual contour. We then divided it into two pieces at its mid point, with each piece corresponding to one portion of the contour that forms the sharp corner. Then, evenly-spaced point samples were taken on each part of the contour. The same number of point samples were also taken on the actual corner boundary, and they were also evenly-spaced. The correspondences of the two sets of points were then found one-for-one (e.g. the first point on the recovered boundary corresponded to the first point on the actual boundary, etc.)

We tested nine cases of distinct corner locations inside a cell using this error measure. Results are shown in plots in Fig. 14. In each test case, the location of the corner was fixed as indicated in each plot (e.g. in the first plot, the corner location is $(0.5, 0.5)$ – the center of the cell). For each location, the degree of the corner varied from 80

to 90 degrees. The stars show error for standard MS and the circles show error for the extended method. For the cases shown in Fig. 14, the isocontour produced by our algorithm had a lower error than the isocontour produced by standard MS.

5.1.3. Contour Length Ratio Error Measure

A third measure we considered was the length of the produced contours versus the length of the actual boundary. We call this measure the *contour length ratio error measure*. The measure uses the ratio of the length of the recovered isocontour, L_r, to the length of the actual boundary, L_a. The error of this measure, E_L, is defined as:

$$E_L = L_r / L_a \,. \tag{2}$$

Thus, values closer to 1 represent better contours. We used the same test cases described at the beginning of Section 5.1 to find this error measure, and we show several plots of error in Fig. 15, with the circles denoting the error of our method and the stars denoting the error of standard MS. The average value of E_L for standard MS is 0.55 (45% underestimation) while the average value of E_L for our algorithm is 0.77 (23% underestimation). These experiments suggest that the extended method produces a contour that has a length closer to the actual boundary than does standard MS.

5.1.4. Error between Actual Corner Location and Recovered Corner Location

A fourth error measure we considered is the discrepancy (error) between the actual corner location and the recovered corner location. We consider such error versus the corner's relative distance to a reference location, L_{AC}, on the grid cell. Since standard MS does not recover a sharp corner, we considered the midpoint of the contour as the recovered corner location, L_{RC}, to measure the discrepancy. The error of this measure, E_{CL}, is defined as:

$$E_{CL} = |L_{AC} - L_{RC}| \,. \tag{3}$$

To evaluate this error, we used synthetic images of an object with a 90-degree sharp corner inside one grid cell. We varied the corner location in each test image.

To determine E_{CL}, we used a grid cell *vertex* as the reference location. Specifically, we used the vertex that is closest to the recovered corner. A plot of error measured this way is shown in Fig. 16a. In this figure, the errors of our algorithm are shown as circles, and the errors of standard MS are shown as stars. Our algorithm was found to produce an error free contour when the actual location is at the center of the grid cell (at that location, the distance to the reference location is 0.71).

We also computed E_{CL} using a location of the grid cell *edge* closest to the recovered corner as the reference location. A plot of error measured this way is shown in Fig. 16b. This plot uses the same color coding as Fig. 16a does. The error free corner case can also be observed in this figure. These two experiments suggest that our algorithm produces

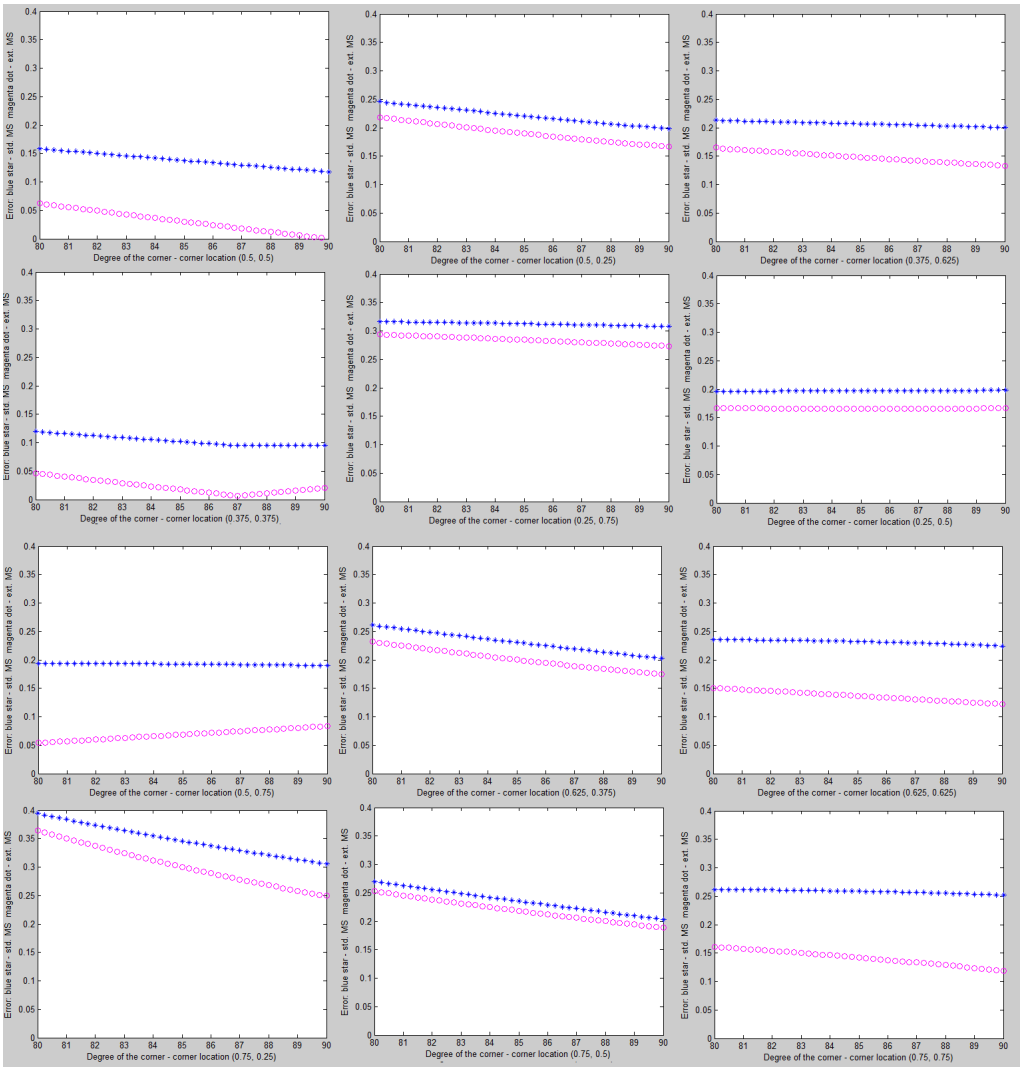


Fig. 14. Average distance between corresponding points error measure, E_{DC}, for new algorithm (circles) and standard MS (stars).

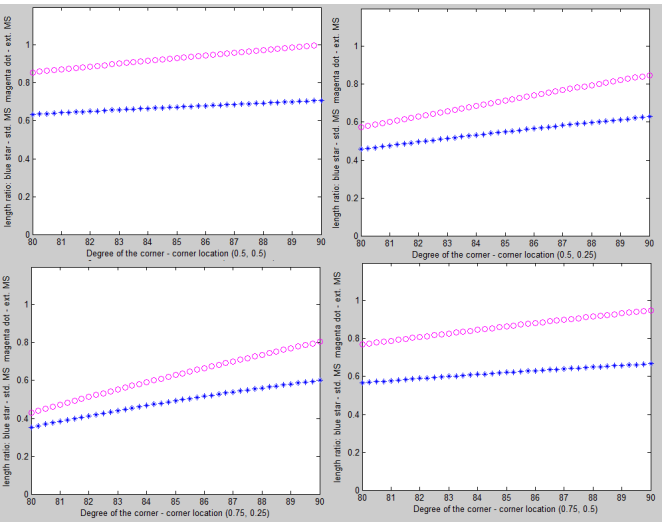


Fig. 15. Contour length ratio error, E_L, measure for new algorithm (circles) and standard MS (stars).

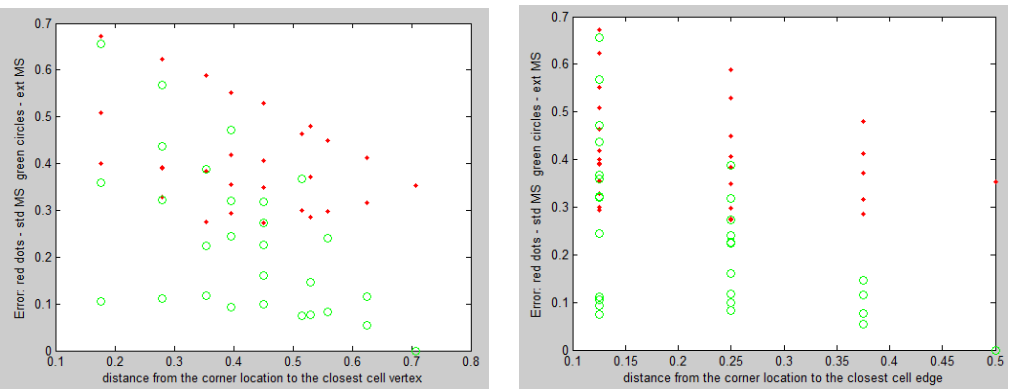


Fig. 16. The distance between recovered corner and the actual corner, E_{CL}, vs. the distance between the actual corner location and two different reference locations. (**a**) Reference location: the closest grid cell *vertex*. (**b**) Reference location: the closest grid cell *edge*.

a corner that is located closer to, and sometimes at the exact location of, the actual corner, while standard MS invariably exhibited error. Our algorithm also tends to produce corners with error smaller than the contours produced by standard MS.

5.2. Processing Time

Next, we report on processing times for the new algorithm and the standard MS. Timings were done on a machine with an Intel i7-3770 quad core processor and 12GB of memory. The datasets used for testing were the Hook1, Hook2, Needle, and Scissors images described earlier and the synthetic L-shaped axis aligned block used in Fig. 8 (which we call Synthetic here). Results are presented in Tab. 2. In these tests, our method had slightly longer processing times than the standard MS due to its additional processing steps for the corner-containing grid cells, but the amount of overhead associated with sharp corner production is quite small (averaging 3.3% overhead).

6. Conclusions

We have presented a new algorithm that allows for production of an isocontour with sharp corner features. The method is an extension to the Marching Squares algorithm. It exploits contour information from neighboring cells to determine likely locations of sharp corners and then creates a contour with sharp corners there. Applications of the new algorithm on synthetic datasets and X-ray images suggest that when sharp corner features occur in the actual boundaries, the new algorithm produces contours closer to the actual boundaries than standard MS does. Error studies show that our approach exhibits smaller error than standard MS in (1) two out of three cases using average closest Euclidean distance, (2) all twelve cases using average distance between corresponding points, and (3) all four cases using the contour length ratio measure. The new algorithm also produces very small to no error when the angle of the sharp corner is close to 90 degrees.

In future work, we hope to develop further extensions that can produce other contour feature types.

Tab. 2. Processing times.

	Processing times [ms]				Processing times (in ms)		
Dataset	Std. MS	Our alg.	Overhead	Dataset	Std. MS	Our alg.	Overhead
Synthetic	0.132	0.143	7.7%	Hook4	0.138	0.144	4.2%
Hook1	0.142	0.147	3.4%	Needle	0.131	0.135	3.0%
Hook2	0.143	0.145	1.4%	Scissors	0.138	0.141	2.1%
Hook3	0.142	0.144	1.4%				

A. Grid Cell Group Topologies

Figs. 17 and 18 show the grid cell group topologies keyed to the Cases 1 to 14 for Marching Squares.

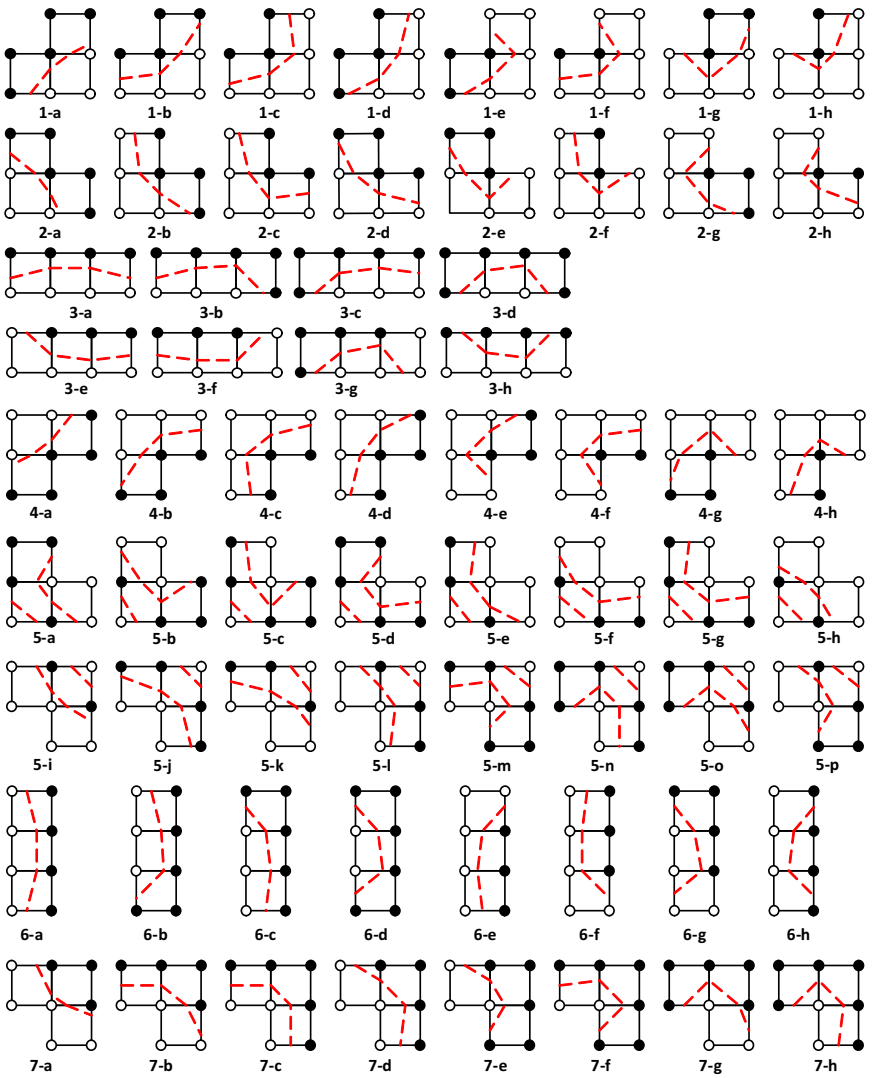


Fig. 17. Grid cell group look-up table – part I.

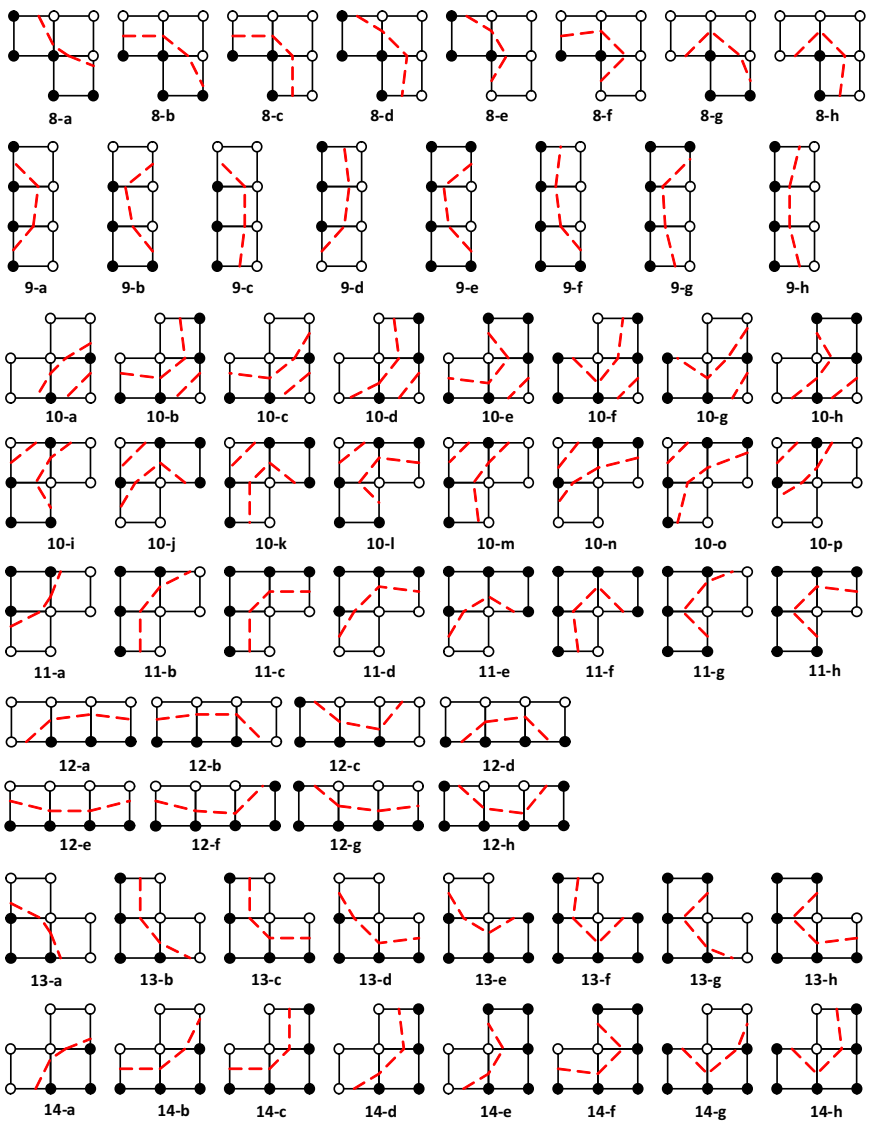


Fig. 18. Grid cell group look-up table – part II.

References

[1] M. Cammarano. Depicting terrain with shaded relief maps. Stanford Univ. Class Report, 2004. http://graphics.stanford.edu/~mcammara/vis2004/paper.pdf. Accessed: Sept 22, 2015.

[2] P. B. Chamberlain and C. L. Thomas. Direct thick layer rapid prototyping from medical images. In *Proc. 10th Solid Freeform Fabrication Symp. SFF '99*, pages 599–605, Austin, TX, August 9-11, 1999. http://sffsymposium.engr.utexas.edu/Manuscripts/1999/1999-069-Chamberlain.pdf.

[3] M. P. Cipolletti, C. A. Delrieux, G Perillo, and M. C. Piccolo. Superresolution border segmentation and measurement in remote sensing images. *Computers & Geosciences*, 40:87–96, March 2012. doi:10.1016/j.cageo.2011.07.015.

[4] C. Collins, G. Penn, and S. Carpendale. Bubble sets: Revealing set relations with isocontours over existing visualizations. *IEEE Trans. Vis. and Comp. Graphics*, 15(6):1009–1016, November 2009. doi:10.1109/TVCG.2009.122.

[5] Fishing accident. http://radiopaedia.org/cases/fishing-accident. Accessed: September 22, 2015.

[6] A. Fofonov, V. Molchanov, and L. Linsen. Visual analysis of multi-run spatio-temporal simulations using isocontour similarity for projected views. *to appear in IEEE Trans. Vis. and Comp. Graphics*, pages 2037–2050, 2016. doi:10.1109/TVCG.2015.2498554.

[7] H. Freeman. Computer processing of line-drawing images. *ACM Comput. Surv.*, 6(1):57–97, March 1974. doi:10.1145/356625.356627.

[8] Q. Gao, S. M. Ali, and P. Edwards. Automated atlas-based pelvimetry using hybrid registration. In *Proc. IEEE 10th Int. Symp. Biomedical Imaging ISBI 2013*, pages 1292–1295, San Francisco, USA, April 2013. doi:10.1109/ISBI.2013.6556768.

[9] S. Gong and T. S. Newman. A corner feature sensitive marching squares. In *Proc. IEEE Southeastcon SECON 2013*, pages 1–6, Jacksonville, USA, April 2013. doi:10.1109/SECON.2013.6567363.

[10] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Prentice Hall, 3 edition, 2007.

[11] T. Igarashi, T. Moscovich, and J. F. Hughes. As-rigid-as-possible shape manipulation. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 1134–1141, Los Angeles, CA, USA, 2005. ACM. doi:10.1145/1186822.1073323.

[12] L. S. Johnson. Progressive transmission of surfaces with geometric constraints. Master's thesis, Univ. of South Carolina, 2004.

[13] T. Kaneko and Y. Yamamoto. Volume-preserving surface reconstruction from volume data. In *Proc. Int. Conf. Image Processing ICIP '97*, volume 1, pages 145–148, Santa Barbara, USA, October 1997. doi:10.1109/ICIP.1997.647405.

[14] L. P. Kobbelt, M. Botsch, U. Schwanecke, and H-P. Seidel. Feature sensitive surface extraction from volume data. In *Proc. 28th Ann. Conf. Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 57–66, New York, NY, USA, 2001. ACM. doi:10.1145/383259.383265.

[15] K. R. Krishnan and S. Radhakrishnan. Focal and diffused liver disease classification from ultrasound images based on isocontour segmentation. *IET Image Proc.*, 9(4):261–270, 2015. doi:10.1049/iet-ipr.2014.0202.

[16] H. Mantz, K. Jacobs, and K. Mecke. Utilizing Minkowski functionals for image analysis: A marching square algorithm. *J. Statistical Mechanics: Theory and Experiment*, 2008(12):P12105, 2008. doi:10.1088/1742-5468/2008/12/P12015.

[17] C. Maple. Geometric design and space planning using the marching squares and marching cube algorithms. In *Proc. Int. Conf. Geometric Modeling and Graphics GMAG 2003*, pages 90–95, July 2003. doi:10.1109/GMAG.2003.1219671.

[18] P. Moinier, J-D. Müller, and M. B. Giles. Edge-based multigrid and preconditioning for hybrid grids. *AIAA Journal*, 40(10):1954–1960, 2002. doi:10.2514/2.1556.

[19] M. Müller. Fast and robust tracking of fluid surfaces. In *Proc. 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, pages 237–245, New Orleans, Louisiana, USA, 2009. ACM. doi:10.1145/1599470.1599501.

[20] A. Murthy, E. Bartocci, F. Fento, et al. Curvature analysis of cardiac excitation wavefronts. *IEEE Trans. Computational Biology and Bioinformatics*, 10(2):323–336, 2013. doi:10.1109/TCBB.2012.125.

[21] T. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers and Graphics*, 30(5):854–879, 2006. doi:10.1016/j.cag.2006.07.021.

[22] G. M. Nielson. On marching cubes. *IEEE Trans. Vis. and Comp. Graphics*, 9(3):283–297, July 2003. doi:10.1109/TVCG.2003.1207437.

[23] Y. Omori, T. Murakami, and T. Ikeda. Color universal design without restricting colors and their combinations using lightness contrast dithering. In *Proc. 5th Int. Congress of Int. Assoc. of Societies of Design Res. IASDR 2013*, 2013. Paper No. 2227-1. http://design-cu.jp/iasdr2013/papers/2227-1b.pdf.

[24] OSU (MSU/WSU) range image database. http://web.archive.org/web/19991008150305/http://eewww.eng.ohio-state.edu/~flynn/3DDB/RID/. Accessed: October 23, 2016.

[25] S. Roettger. The Volume Library. http://schorsch.efi.fh-nuernberg.de/data/volume/. Accessed: September 22, 2015.

[26] B. Schlei. A new computational framework for 2D shape-enclosing contours. *Image and Vision Computing*, 27(6):637–647, May 2009. doi:10.1016/j.imavis.2008.06.014.

[27] D. Siedhoff, F. Weichert, P. Libuschewski, and C. Timm. Detection and classification of nano-objects in biosensor data. In *Proc. 6th Int. Workshop on Microscopic Image Analysis with Applications in Biology MIAAB 2011*, Heidelberg, Germany, September 2011.

[28] Z. Wang, J. K. Min, and G. Xiong. Robotics-driven printing of curved 3D structures for manufacturing cardiac therapeutic devices. In *Proc. IEEE Int. Conf. Robotics and Biomimetics ROBIO 2015*, pages 2318–2323, December 2015. doi:10.1109/ROBIO.2015.7419120.

[29] D. Wu, H. Tian, G. Hao, et al. Design and realization of an interactive medical images three dimension visualization system. In *Proc. 3rd Int. Conf. Biomedical Engineering and Informatics BMEI 2010*, volume 1, pages 189–193, Oct 2010. doi:10.1109/BMEI.2010.5639435.

Extraction of Image Parking Spaces in Intelligent Video Surveillance Systems

Rykhard Bohush[1], Pavel Yarashevich[1], Sergey Ablameyko[2], Tatiana Kalganova[3]
[1] *Polotsk State University, Polotsk, Belarus*
[2] *Belarusian State University, Minsk, Belarus*
[3] *Brunel University, London, UK*
r.bogush@psu.by

Abstract. This paper discusses the algorithmic framework for image parking lot localization and classification for the video intelligent parking system. Perspective transformation, adaptive Otsu's binarization, mathematical morphology operations, representation of horizontal lines as vectors, creating and filtering vertical lines, and parking space coordinates determination are used for the localization of parking spaces in a video frame. The algorithm for classification of parking spaces is based on the Histogram of Oriented Descriptors (HOG) and the Support Vector Machine (SVM) classifier. Parking lot descriptors are extracted based on HOG. The overall algorithmic framework consists of the following steps: vertical and horizontal gradient calculation for the image of the parking lot, gradient module vector and orientation calculation, power gradient accumulation in accordance with cell orientations, blocking of cells, second norm calculations, and normalization of cell orientation in blocks. The parameters of the descriptor have been optimized experimentally. The results demonstrate the improved classification accuracy over the class of similar algorithms and the proposed framework performs the best among the algorithms proposed earlier to solve the parking recognition problem.

Key words: parking space, localization, Histogram of Oriented Descriptors, classification, Support Vector Machine.

1. Introduction

The importance of video surveillance systems become more and more important in different types of human activity. The development in computer vision technologies made it possible thanks for recent development of video surveillance systems with intelligent processing of input video data for various applications, including car parking control systems.

Over the last decade the numerous intelligent systems have been developed for parking space detection with different lighting conditions and restrictions [1, 2, 6, 14, 24, 25]. For example, a system for management and monitoring of a parking lot by a video camera for simple indoor parking garage proposed in [6] uses the edge detection method for identification of the free parking places within indoor scenario only with a permanent lightning source. Parking system described in the paper [25] uses the brown rounded

image drawn at parking spaces and produces the information of empty spaces. A display shows the number of currently available parking lots. This system was tested and results were presented only for 8 parking spaces. A video-based system for vacant parking space detection based on colour histograms and difference of Gaussian features, SVM classifier, and exponential smoothing for temporal integration is presented in [24]. This system can be adopted by a car-park routing system to navigate drivers to a comfortable parking space. Authors proposed to use Raspberry Pi in a video camera in [2]. That work is based on deep Convolution Neural Network (CNN) architecture to classify images of parking spaces as occupied or vacant and exhibits a very high accuracy. However, such approach is computationally expensive.

The systems like those just described can be integrated in a large system known as Smart City and some optional functionality for both parking owners and parking users can be implemented [1, 2, 18]. For example, for parking owner, an important additional information is the number of automobiles in the parking lot, the automobile types, the human activity. Car owners need background information about vacant parking space location, car surveillance with a smartphone, alarming via smartphone in case of an extraordinary situation [16]. In such systems the advanced, efficient video data processing algorithms are necessary, which need large computational resources. They should operate in the presence of various noise factors: shadows, light spots in sunny weather, changes in the overall illumination of the automotive parking during the day, changing weather conditions, etc.

The methods that determine the occupancy of parking spaces on video images can be divided into three groups. The first group includes methods based on detection of a car in a parking lot [5, 22], the second group contains methods based on a comparison of the processed parking space with the reference vacant space [20] and the third group includes from combined methods [12, 15].

The methods from the first group are unstable due to classification errors, which arise because of the overlap of cars in video images. The second group of methods are based on the model of reference vacant parking space, so the probability of false classification increases with the appearance of natural noise, the presence of people and other objects in an image, the local change in illumination, etc. In the combined methods, as a rule, the positive properties of the component methods are emphasized and their negative sides are avoided, but computing cost of these methods are usually large.

Among the features to be chosen to form vectors of image characteristics, HOG [7] is a good candidate. This is due to that it has a number of desirable characteristics, like the invariance to image rotation and scale changing, and the stability to noise and illumination variations. Recently, a range of methods and algorithms using HOG gradients have been proposed [10, 11, 12, 13]. In the method [11] the HOG descriptor and Bayesian classifier was used, and the correct classification probability was 0.9945 when the images were taken in typical lighting conditions. In this method it is assumed that

a structural 3D model of each parking place is built, so the computation time is long. The method described in [12] is more effective than those previously mentioned and attains an increased probability of correct classification in case of less controlled shooting conditions. An improvement of this algorithm by introducing the SVM classification method is presented in [13]. As a result, the probability of correct classification became 0.9955. In [10], the parking lots classification algorithm based on HOG and SVM is compared with the algorithm based on Haar-like features and AdaBoost method. For the first algorithm, the probability of correct classification is equal to 0.691, while for the second method this probability is 0.95. Hence, despite of good characteristics of the both methods the problem which is our object of interest has not being solved yet. Therefore, this area of research is still of great interest.

This paper is organized as follows. In Section 2, an intelligent parking lot inspection system, suitable for scaling and having some optional service functionalities is illustrated. In Section 3, the algorithm for locating the parking spaces in the parking lot image and the algorithm for space classification are presented. In Section 4, experimental results of the search for more effective descriptor parameters and for better class separating functions in the classifier are performed. The correct classification probability of parking spaces of the present algorithm is compared to those of the other approaches. Finally, a conclusion is provided in Section 5.

2. Intelligent Parking Lot Control System

Intelligent parking lot control system is shown in Fig. 1. The system consists of video camera, video processing module, service functions module, information representation device or display and cloud platform for integration with the Smart City. Video information processing module includes subsystems for parking space localization and subsystems for parking space classification.

Parking space localization subsystem is intended to locate parking spaces in video frame and save these positions in Local Storage. This is required when the point of view is changing after the installation of a new system or camera. Input image perspective transformation is used to facilitate possibility of describing parking space by rectangle, to detect separating parking lines and should be found once for a given car park and camera location. For parking space segmentation is mandatory a specialized processing algorithm. In the last step, coordinates of parking spaces are saved in local storage. This data will be used by subsystem of parking spaces classification.

Parking space classification subsystem captures a video, processes it frame-by-frame and present the result for the current frame. This subsystem is used to dedicate following contributions: coordinates of parking space extraction from local storage; feature extraction for each image of parking space; parking spaces classification to “vacant” and “occupied” based on features; merging results from the previous step with input video

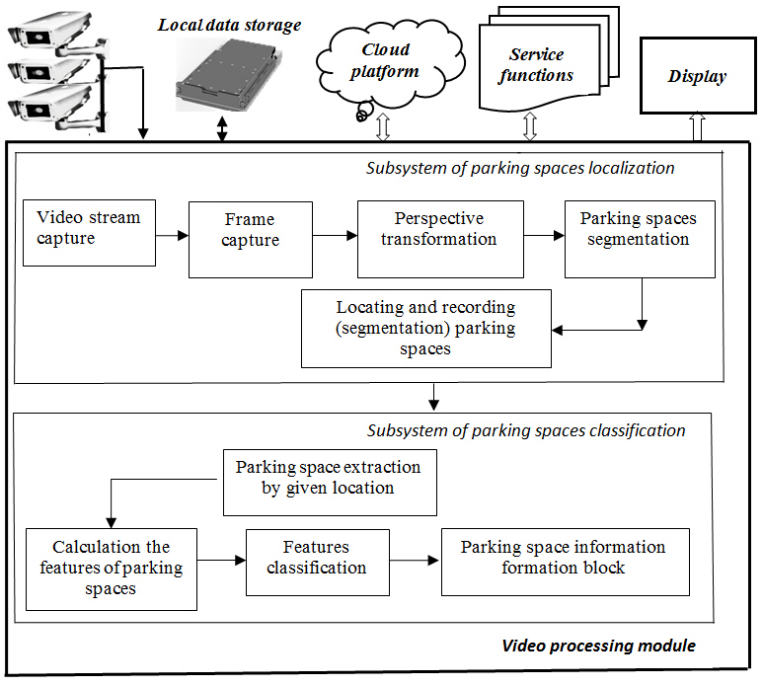


Fig. 1. The structure of the intelligent parking lot control system.

frame for visualization on a display of occupied and vacant parking spaces. System capability can be escalated by optional module "Service functions". This module can consist of car counter on parking lot, car types recognition, map of vacant parking spots, car surveillance with smart phone, etc. This approach can help to solve an important problem of huge negative environmental impact by minimizing it. The level of car emissions will significantly decrease, as a driver can quickly find a vacant parking spot in a large parking lot.

3. Parking space extraction algorithms

3.1. Parking spaces localization

The system determine locations of parking spaces when a new system is installed or the camera's point of view is changed. Parking space location is describes by four coordinates in image of parking lot. Locations of Parking spaces stored in Local Storage. Algorithm of parking space localization includes the following steps:

1. Perspective transformation of parking lot image. The perspective transformation is: $\left(x^{'}, y^{'}, 1\right) = H \times (x, y, 1)^T$, where $\left(x, y, 1\right)$ are homogeneous pixel coordinates in the image of parking lot, $\left(x^{'}, y^{'}, 1\right)$ are corresponding homogeneous pixel coordinates in the output image, H – homography matrix. The pixels coordinates $\left(x_p, y_p\right), p = \overline{1,4}$ on the image of parking lot and corresponding pixels coordinates $\left(x_p^{'}, y_p^{'}\right), p = \overline{1,4}$ on the output image are used to compute homography matrix. Pixels coordinates $\left(x_p, y_p\right)$ are defined manually, when perspective transformation is applying the first time. These coordinates are the corners of the closest parking row. Pixels coordinates $\left(x_p^{'}, y_p^{'}\right)$ describe a rectangular area. The output image after perspective transformation is larger than the original image. Now, each parking space represents a rectangle area, and its sides are parallel to the axes.
2. Otsu method is used to binarize the image.
3. A closing operation of mathematical morphology is used to remove gaps on the lines that separate parking spaces from each other:

$$A \bullet B = \left(A \oplus B\right) \ominus B \tag{1}$$

where $\left(A \oplus B\right)$ is dilation of image A by the structuring element B of 7×7 and $\ominus$ is erosion. An opening operation of mathematical morphology is used to shrink areas of small noise and unrelated elements:

$$A \circ C = \left(A \ominus C\right) \oplus C \tag{2}$$

where C – the structuring element of 5×5.
4. Each inseparable horizontal sequence of pixels is represented by horizontal line $V_i\left(x_i^{'hl}, y_i^{'hl}, l_i\right)$, $i = \overline{0, nov-1}$, where $\left(x_i^{'hl}, y_i^{'hl}\right)$ is a start pixel of horizontal line, l is a length of horizontal line, nov is a total number of lines. The one pixel represents as $\left(x^{'hl}, y^{'hl}, 1\right)$.
5. Horizontal lines $V_i\left(x_i^{'hl}, y_i^{'hl}, l_i\right)$ are stacking into vertical lines $L_j\left(\bar{V}\right)$, $j = \overline{0, nol-1}$, nol – total number of vertical lines. Vectors form vertical line and they should be one under another and should not exceed neighboring vectors by length. They stacked into vertical line $L_j\left(x_j^{'vl}, y_j^{'vl}, w_j^{'vl}, h_j^{'vl}\right)$, where $\left(x_j^{'vl}, y_j^{'vl}, \right)$ is a start pixel of vertical line, w_j, h_j are width and height of vertical line. Each vertical line L_j covers every horizontal line over a rectangle area $x_j^{'vl}, y_j^{'vl}, x_j^{'vl} + w_j^{'vl}, y_j^{'vl} + h_j^{'vl}$.
6. The lines are sorted according to the initial coordinates: from the left to the right, from the top to the bottom. Then they are combined into parking spaces $S_k\left(L_k^l, L_k^r\right)$, $k = \overline{0, nos-1}$, where L_k^l, L_k^r are the left and the right dividing strips respectively, nos is a number of parking spaces.
7. Parking spaces $S_k\left(x_k^{'s}, y_k^{'s}, w_k^{'s}, h_k^{'s}\right)$, $k = \overline{0, nos-1}$ are described by its initial coordinates $\left(x^{'s}, y^{'s}\right)$, its width $w^{'s}$ and height $h^{'s}$.

3.2. Feature set computation

1. The subimage of parking space $I\left(w^{'s}, h^{'s}\right)$ is extracted according to its location $S\left(x^{'s}, y^{'s}, w^{'s}, h^{'s}\right)$. Then it is scaled into subimage $I\left(w, h\right)$, where $\left(w, h\right)$ is the size of the subimage.
2. Magnitude of gradient G and orientation of gradient θ is calculated for grayscaled subimage of parking space I as:

$$G_{x,y} = \sqrt{{G_{x,y}^{x}}^2 + {G_{x,y}^{y}}^2} \tag{3}$$

$$\theta_{x,y} = \arctan\left(\frac{G_{x,y}^{x}}{G_{x,y}^{y}}\right) \tag{4}$$

where $G_{x,y}^{x}$ and $G_{x,y}^{y}$ – horizontal and vertical gradients:

$$G_{x,y}^{x} = I_{x,y} \cdot M^{x} \tag{5}$$

$$G_{x,y}^{y} = I_{x,y} \cdot M^{y} \tag{6}$$

where $0 \leq x < w$, $0 \leq y < h$, M^{x} and M^{x} – derivative kernels $\left(M^{x} = M^{yT} = [-1, 0, +1]\right)$.

3. Magnitude of gradients G is divided into cells $C_{m,n}$ with size $C_w \times C_h$ and power gradient accumulation in accordance with orientations θ for each cell:

$$C_{m,n}^{l} = \sum_{\substack{j=n\cdot ch \\ i=m\cdot cw}}^{\substack{(n+1)\cdot ch-1 \\ (m+1)\cdot cw-1}} \begin{cases} G_{i,j}, \ \frac{l\cdot 2\cdot \pi}{b} \leq \theta_{i,j} < \frac{(l+1)\cdot 2\cdot \pi}{b} \\ 0, \ \frac{l\cdot 2\cdot \pi}{b} > \theta_{i,j} \bigcup \theta_{i,j} \geq \frac{(l+1)\cdot 2\cdot \pi}{b} \end{cases} \tag{7}$$

where b – a number of orientation bins in cell, $0 \leq l < b$, $0 \leq m < cw$, $0 \leq n < ch$, $cw = \frac{w}{C_w}$, $ch = \frac{h}{C_h}$.

4. Cells are united into overlapping blocks $B_{f,g}$ with size $B_w \times B_h$:

$$B_{f,g} = \begin{bmatrix} C_{f,g} & \cdots & C_{f+B_w-1,g} \\ \vdots & \ddots & \vdots \\ C_{f,g+B_h-1} & \cdots & C_{f+B_w-1,g+B_h-1} \end{bmatrix} \tag{8}$$

where $0 \leq m < bw$, $0 \leq g < bh$, $bw = cw - B_w + 1$, $bh = ch - B_h + 1$.

5. The L2-norm block normalization scheme is used. Some cells present in several overlapping blocks, and each of them normalized with respect to the block it belongs to:

$$C_{i,j}^{l\ f,g} = \frac{C_{i,j}^{l}}{||B_{f,g}||_2} = \frac{C_{i,j}^{l}}{\sqrt{\sum_{i=f}^{f+B_w-1} \quad \sum_{j=g}^{g+B_h-1} \quad \sum_{l=0}^{b-1} {C_{i,j}^{l}}^2 + \epsilon^2}} \tag{9}$$

where ϵ is a small constant, prevents from division by zero.

6. Normalized blocks are sequentially added to the descriptor. Values of cells $C_{i,j}^{l\ f,g}$ in blocks $B_{f,g}$ are grouped together to construct a feature vector:

$$d_{b\cdot B_w\cdot B_h\cdot(bw\cdot g+f)+b\cdot(B_w\cdot(j-g)+(i-f))+l} = C_{i,j}^{l\ f,g} \tag{10}$$

Vector size of d depends on size $w \times h$ of parking space subimage, size of cell $C_w \times C_h$, numbers of orientations in cell b and size of block $B_w \times B_h$:

$$D_S = b \cdot B_w \cdot B_h \cdot \Big(\frac{w}{C_w} - 1\Big) \cdot \Big(\frac{h}{C_h} - 1\Big) \tag{11}$$

3.3. Feature set classification

We use support vector machine to classify computed descriptors of parking space [21]. Support Vector Machine is a linear classifier. To perform a non-linear classification kernel trick is used. Support Vector Machine is used for binary classification to distinguish vacant parking spaces from occupied:

$$a(d) = \text{sign}\Big(\sum_{i=0}^{n} \lambda_i y_i K\big(d_i, d\big) - w_0\Big) \tag{12}$$

where $a(d)$ – returns +1 if parking space is vacant, and -1 if parking space is occupied; $\lambda\big(\lambda_0, \lambda_1, ..., \lambda_n\big)$ – a vector of dual variables; d_i, $i = 1, n$ – descriptors of support vectors; $y\big(y_0, y_1, ..., y_n\big)$ – an array of class labels; w_0 – a threshold value; d – a descriptor of parking space subimage to be classified. Classification efficiency depends on kernel functions. For parking lot classification, the following kernel functions can be used:

- linear:

$$K_{\text{lin}}(d_a, d_b) = d_a \cdot d_b \tag{13}$$

where d_a and d_b – descriptors.

- radial basis function:

$$K_{\text{rbf}}(d_a, d_b) = \exp\left(-\gamma \cdot ||d_a - d_b||^2\right) \tag{14}$$

- polynomial:

$$K_{\text{poly}}(d_a, d_b) = \big(\gamma \cdot K(d_a, d_b) + c\big)^{\delta} \tag{15}$$

γ, δ, c – parameters of corresponding kernel functions.

- histogram intersection:

$$K_{\text{hist}}(d_a, d_b) = \sum \min\left(|d_{ai}|, |d_{bi}|\right) \tag{16}$$

To achieve good results of classification the parameters of Histogram of Oriented Gradients and Support Vector Machine should be chosen attentively. Relevant experiments are described in the next section.

Tab. 1. Main characteristics of parking spaces subimages.

Weather conditions	**Number of days/ number of images**	**Number of subimages of parking spaces**	
		vacant	**occupied**
Mist	6/664	61406	28898
Rain	4/416	18670	37906
Shiny	12/1073	78801	67127

4. Experimental results and discussion

4.1. Evaluation metrics definition

For our experiments, we used the PKLot dataset [8]. PKLot dataset contains images of parking lot taken in different weather conditions, shadow effects and illumination. Pictures from this dataset taken by a camera, that is installed high enough and located in the middle of the right parking row, and rows behind it. So the perspective transformation could be applied to receive the top view of parking lot image, without overlapping between vehicles.

Experiments on described algorithms and evaluation of research result was realized using programming language Java, Eclipse development environment, computer vision libraries OpenCV 3.0.0 and machine learning library jlibsvm.

Fig. 2 shows parking lot image (Fig. 2a) and results for some steps (Fig. 2b-f) of parking space localization algorithm. Parking lot image after perspective transformation, binarization and applying morphological operations is presented in Fig. 2b. Fig. 2c is a subimage from Fig. 2b. Fig. 2d shows a result after horizontal and vertical lines definition on the subimage (Fig. 2c). Results parking spases localization for subimage and all parking lot are presented in Fig. 2e and Fig. 2f correspondingly. We use 2135 images of parking lot from PKLot dataset. The total number of subimages of parking spaces is 292808. Fig. 3 shows some examples for vacant and occupied parking spaces. Main characteristics of parking images for different weather conditions are presented in Tab. 1. Descriptor efficiency is evaluated by correctly parking space images classification probability RR that can be represented as:

$$RR = \frac{TP + TN}{TP + FP + TN + FN} \tag{17}$$

where TP – correctly classified vacant parking spaces; FP – incorrectly classified vacant parking spaces; TN – correctly classified occupied parking spaces; FN – incorrectly classified occupied parking spaces. The probability of falsely vacant parking space subimage classification is calculated as follows:

$$FPR = \frac{FP}{TP + FP} \tag{18}$$

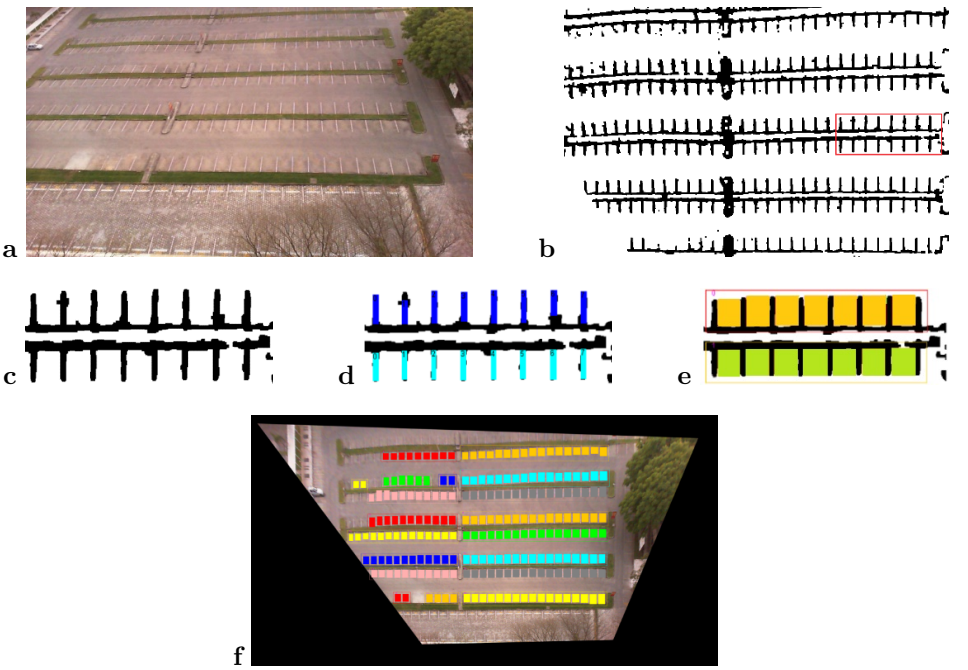


Fig. 2. Example of parking spaces localization algorithm. (**a**) The image of parking lot; (**b**) the result of applying the first, the second and the third step; (**c**) the subimage of Fig. 2b; (**d**) the result of applying the fourth and the fifth step; (**e**) after applying the sixth and the seventh step; (**f**) localized parking spaces.

Fig. 3. Examples for image of parking spaces: (**a**) vacant; (**b**) occupied.

The probability of falsely occupied parking space subimage classification is defined as

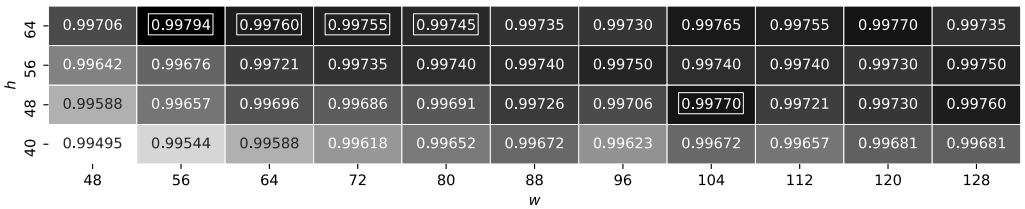


Fig. 4. Probability of correct classification for different size of parking space subimage.

equation:

$$FNR = \frac{FN}{TN + FN} \tag{19}$$

4.2. Experiments and its results

Proposed model for parking lot classification is consists of HOG descriptor and SVM classifier. Parameters of descriptor and classifier should be optimized to receive high efficiency model. HOG includes a large number of parameters to optimize: the size of parking space subimage, the size of cell, the number of orientation bins, the size of block. It is very hard problem to find an optimal set for all the possible combinations of parameters. Therefor, fixed sequence of parameters would be used to optimize: w, h, C_w, C_h, b, where C_w, C_h are width and height of cell; b is number of orientations in cell. At the first step, dual-dimension parameters (the size of parking space subimage, the size of cell) would be determined. After that, the one-dimensional parameter (number of orientation cells) should be determined. At each step, parameters will be selected to reduce the descriptor size. This approach would reduce computational cost for parking space classification subsystem.

The highest probability of correct classification was achieved by using the following sizes of parking space subimage: 64×56, 64×64, 64×72, 64×80, 48×104. We used 8×8 cell and 9 bins cell histogram, which are effective for other practical tasks, for example, pedestrian detection The experimental results are shown in Fig. 4. Figures 4-6 are the heatmap diagrams. Such a diagram represents a value by a color, a minimum and maximum values are represented by white and black colors correspondingly. White rectangles show parameter values that are selected for next step. Next, highest probability of correct classification was achieved for parking spaces subimage size and cells size $\left(w, h, C_w, C_h\right)$: (64, 56, 8, 8), (64, 64, 8, 8), (64, 72, 8, 6), (64, 72, 8, 8), (64, 72, 8, 9), (64, 72, 8, 12), (64, 80, 8, 5), (64, 80, 8, 8), (64, 80, 8, 16), (48, 104, 8, 8). The experimental results are shown in Fig. 5. At last step, we find the orientation numbers in cells that provide maximal correct detection probability on the

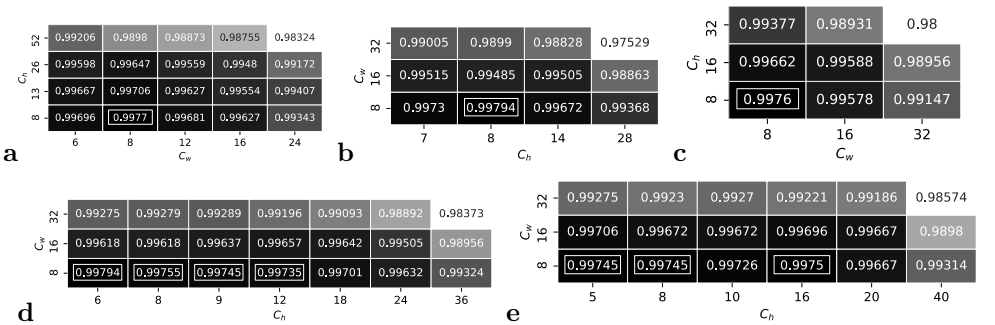


Fig. 5. Correct classification probability for different cell's size with size of parking space subimage. (**a**) 64×56; (**b**) 64×64; (**c**) 64×72; (**d**) 64×80; (**e**) 48×104.

Tab. 2. Efficiency comparison of different kernel function.

KF	SSW	SVN	RR	FPR	FNR
Lin.	4.366	161	0.99431	2.91e-3	9.16e-3
H. I.	2.387	262	0.99652	2.12e-3	5.19e-3
R. B. F.	1.371	330	0.99706	1.41e-3	4.86e-3
Poly.	0.752	479	0.99740	1.58e-3	3.86e-3

interval from 4 to 18 bins histogram. Experimental results for the number of orientations in the cell showed that for the interval from 4 to 8 bin, values of right detecting probability are not acceptable. Therefore, experimental results from 9 to 18 are shown in fig. 6.The highest probability of correct classification was achieved by using for following parameters $\big(w, h, C_w, C_h, b\big)$: (64, 72, 8, 6, 14), (64, 72, 8, 6, 16), (64, 72, 8, 8, 16). The feature set length of obtained descriptors are $D_s\big(64, 72, 8, 6, 14\big) = 4928$, $D_s\big(64, 72, 8, 6, 16\big) = 5638$, $D_s\big(64, 72, 8, 8, 16\big) = 3584$. So, the most effective descriptor given reduce computation cost has the following parameters: parking space size is 64×72, cell size is 8×8, and orientations number in cell is 16.

SVM classification result efficiency is depends on the type of kernel function. Therefore, experiments were conducted to evaluate the effectiveness of SVM classification of parking space characteristics based on HOG using various kernel functions Tab. 2 (where KF – kernel function, SSW – separating strips wide, SVN – support vector number, Lin. – linear kernel, H. I. – histogram intersection kernel, R. B. F. – Radial Basis Function, Poly – polynomial kernel). The polynomial kernel shows the highest RR, but we choose histogram intersection kernel because it classifies much more faster then radial basic function kernel or polynomial kernel.

Fig. 7 shows the example of parking space classification using the developed algorithm

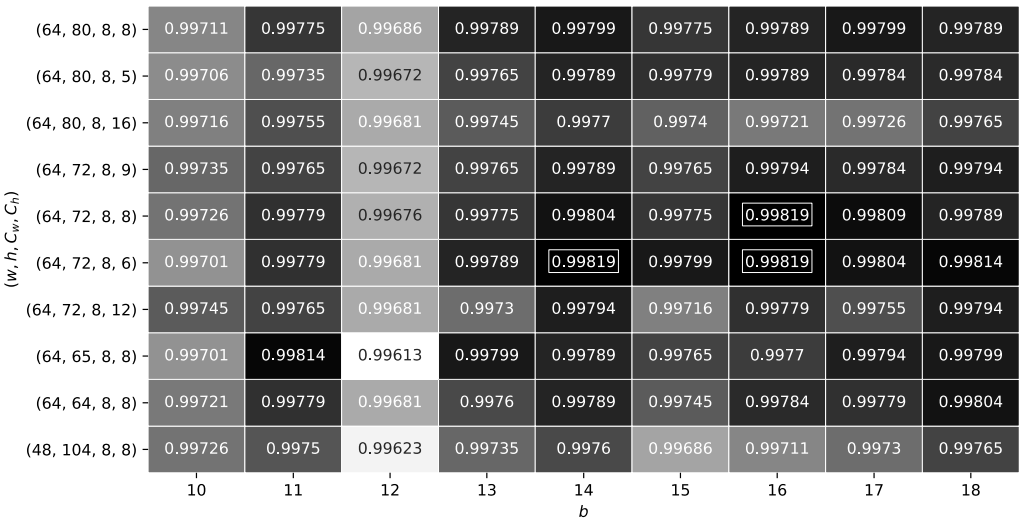


Fig. 6. Probability of correct classification for different orientations numbers in cells: (64, 56, 8, 8), (64, 64, 8, 8) (64, 72, 8, 6), (64, 72, 8, 8), (64, 72, 8, 9), (64, 72, 8, 12), (64, 80, 8, 5), (64, 80, 8, 8), (64, 80, 8, 16), (48, 104, 8, 8).

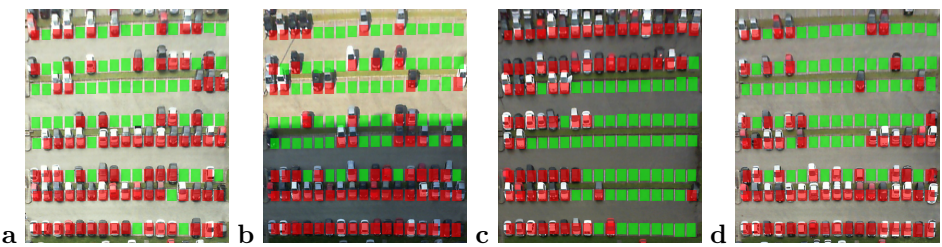


Fig. 7. The result of parking spaces classification under different weather conditions. (**a**) Sunny; (**b**) Shadow cast; (**c**) Rainy; (**d**) Cloudy.

with defined parameters under different weather conditions. Every image of parking space has been correctly classified. Comparison of the proposed algorithm with other classification of parking space image algorithms is shown in Tab. 3.

These results show that correct classification of image parking spaces based on proposed descriptor parameters is more attractive with other descriptor parameters.

Comparison of classification algorithms for PKLot dataset is presented in Tab. 4. As

Tab. 3. Characteristics comparison of classification algorithms for parking spaces.

Algorithm	**Descriptor**	**Classifier**	**No. of parking lot images**	**No. of parking spaces**	**RR**
C. C. Huang [11]	HOG (64, 32, 16, 16, 8)	Bayes	955	72	0.9945
C. C. Huang [12]	HOG (64, 32, 16, 16, 10)	Bayes	825	72	0.9939
C. C. Huang [13]	HOG (96, 48, -, -, -)	SVM	1,564	72	0.9955
R. Fusek [10]	HOG (96, 96, 8, 8, 4)	SVM	-	57	0.6910
R. Fusek [10]	Haar	Adaboots	-	57	0.9500
M. Tschentscher [23]	Histogram, HSV	k-NN	1,010	36	0.9655
	DoG				0.9358
	Histogram, RGB	SVM			0.9712
	DoG				0.9413
L. Baroffio [4]	Histogram, HSV	SVM	3,791	28	0.9600
			4,152	37	0.9300
			4,474	100	0.8700
Proposed	HOG(64, 72, 8, 8, 16)	SVM	2,153	136	**0.9970**

Tab. 4. Comparison of classification algorithms for PKLot dataset.

Author	**Classifier**	**Descriptor**	**RR**
L. Baroffio et al. [4]	SVM	Color Hist.	0.960
G. Amat et al. [3]	mAlexNet	-	0.904
D. D. Mauro et al. [9]	mAlexNet	-	0.990
G. Amato et al. [2]	mAlexNet	-	0.996
X. Li et al. [17]	GAN	-	0.957
S. Nurullayev et al. [19]	CarNet	-	0.982
Proposed	SVM	HOG	**0.997**

we can see in this table, the approach is the most effective among other approaches, which using PKLot dataset for testing.

5. Conclusion

The intelligent parking lot control system is proposed. It includes the video camera, the video processing module, the cloud platform, the service functions module, and the information representation device or display. In this system, efficient algorithms for parking space localization and classification have been proposed. The algorithm for localization of parking spaces in the parking lot image consists of the following steps: perspective transformation, Otsu's binarization, operations of mathematical morphology, lines construction and coordinates of parking spaces determination. Histogram of Oriented Descriptors and Support Vector Machine are used to classify the parking spaces. The

algorithm for HOG computation consists of the following steps: vertical and horizontal gradient calculation for image of parking lot, gradient module vector and orientation calculation, power gradient accumulation in accordance to the cell orientations, blocking of cells, second norm calculations, and normalization of cell orientation in blocks. The kernel function based on histogram intersection is the most effective to classify the computed HOG feature sets for car parking spaces. The most effective parameters of the descriptor were found to be as follows: 64×72 – the size of parking space subimage, 8×8 – the size of cell, 16 – the number of orientation bins. The correct classification rate is 0.997. The experimental results demonstrate the improved classification accuracy.

References

[1] M. Alam, D. Moroni, G. Pieri, M. Tampucci, M. Gomes, J. Fonseca, J. Ferreira, and G. R. Leone. Real-time smart parking systems integration in distributed ITS for smart cities. *Journal of Advanced Transportation*, 2018. Article ID 1485652. doi:10.1155/2018/1485652.

[2] G. Amato, F. Carrara, F. Falchi, C. Gennaro, C. Meghini, and C. Vairo. Deep learning for decentralized parking lot occupancy detection. *Expert Systems with Applications*, 72:327–334, 2017. doi:10.1016/j.eswa.2016.10.055.

[3] G. Amato, F. Carrara, F. Falchi, C. Gennaro, and C. Vairo. Car parking occupancy detection using smart camera networks and deep learning. In *Proc. IEEE Symp. on Computers and Communication (ISCC 2016)*, pages 1212–1217. IEEE, 2016. doi:10.1109/iscc.2016.7543901.

[4] L. Baroffio, L. Bondi, M. Cesana, A. E. Redondi, and M. Tagliasacchi. A visual sensor network for parking lot occupancy detection in smart cities. In *Proc. IEEE 2nd World Forum on Internet of Things (WF-IoT 2015)*, pages 745–750, 2015. doi:10.1109/WF-IoT.2015.7389147.

[5] D. B. L. Bong, K. C. Ting, and N. Rajaee. Car-park occupancy information system. In *Proc. 3rd Real-Time Technology and Applications Symposium (RENTAS 2006)*, pages 65–70, Serdang, Selangor, Malaysia, 2006.

[6] T. Čaklović, I. Aleksi, and Ž. Hocenski. Managing and monitoring of a parking lot by a video camera. In *Proc. of Automation in Transportation*, Zagreb, Croatia, 2010. http://bib.irb.hr/datoteka/509872.Automatizacija_u_prometu_2010.pdf.

[7] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *Proc. Int. Conf. on Computer Vision & Pattern Recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005. doi:10.1109/CVPR.2005.177.

[8] P. R. L. De Almeida, L. S. Oliveira, A. S. Britto, E. J. Silva, and A. L. Koerich. PKLot – a robust dataset for parking lot classification. *Expert Systems with Applications*, 42(11):4937–4949, 2015. doi:10.1016/j.eswa.2015.02.009.

[9] D. Di Mauro, S. Battiato, G. Patanè, M. Leotta, D. Maio, and G. M. Farinella. Learning approaches for parking lots classification. In *Proc. Int. Conf. on Advanced Concepts for Intelligent Vision Systems (ACIVS 2016)*, volume 10016 of *Lecture Notes in Computer Science*, pages 410–418. Springer, 2016. doi:10.1007/978-3-319-48680-2_36.

[10] R. Fusek, K. Mozdřeň, M. Šurkala, and E. Sojka. Adaboost for parking lot occupation detection. In *Proc. 8th Int. Conf. on Computer Recognition Systems CORES 2013*, volume 226 of *Advances in Intelligent Systems and Computing*, pages 681–690. Springer, 2013. doi:10.1007/978-3-319-00969-8_67.

[11] C. C. Huang, Y. S. Dai, and S. J. Wang. A surface-based vacant space detection for an intelligent parking lot. In *Proc. 12th Int. Conf. on ITS Telecommunications*, pages 284–288. IEEE, 2012. doi:10.1109/ITST.2012.6425183.

[12] C. C. Huang, Y. S. Tai, and S. J. Wang. Vacant parking space detection based on plane-based bayesian hierarchical framework. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(9):1598–1610, 2013. doi:10.1109/TCSVT.2013.2254961.

[13] C. C. Huang, H. T. Vu, and Y. R. Chen. A multiclass boosting approach for integrating weak classifiers in parking space detection. In *Proc. 2015 IEEE Int. Conf. on Consumer Electronics – Taiwan*, pages 314–315. IEEE, 2015. doi:10.1109/ICCE-TW.2015.7216918.

[14] M. Y. I. Idris, Y. Y. Leng, E. M. Tamil, N. M. Noor, and Z. Razak. Car park system a review of smart parking system and its technology. *Information Technology Journal*, 8(2):101–113, 2009. doi:10.3923/itj.2009.101.113.

[15] J. Jermsurawong, M. U. Ahsan, A. Haidar, H. Dong, and N. Mavridis. Car parking vacancy detection and its application in 24-hour statistical analysis. In *Proc. 10th Int. Conf. on Frontiers of Information Technology (FIT 2012)*, pages 84–90. IEEE, 2012. doi:10.1109/FIT.2012.24.

[16] J. Lanza, L. Sánchez, V. Gutiérrez, J. A. Galache, J. R. Santana, P. Sotres, and L. Muñoz. Smart city services over a future internet platform based on internet of things and cloud: The smart parking case. *Energies*, 9(9):719, 2016. doi:10.3390/en9090719.

[17] X. Li, M. C. Chuah, and S. Bhattacharya. Uav assisted smart parking solution. In *Proc. Int. Conf. on Unmanned Aircraft Systems (ICUAS 2017)*, pages 1006–1013. IEEE, 2017. doi:10.1109/icuas.2017.7991353.

[18] R. Novotnỳ, R. Kuchta, and J. Kadlec. Smart city concept, applications and services. *Journal of Telecommunications System & Management*, 3(2):1, 2014. doi:10.4172/2167-0919.1000117.

[19] S. Nurullayev and S. W. Lee. Generalized parking occupancy analysis based on dilated convolutional neural network. *Sensors*, 19(2):277, 2019. doi:10.3390/s19020277.

[20] R. J. L. Sastre, P. G. Jimenez, F. J. Acevedo, and S. M. Bascon. Computer algebra algorithms applied to computer vision in a parking management system. In *Proc. IEEE Int. Symp. on Industrial Electronics (ISIE 2007)*, pages 1675–1680. IEEE, 2007. doi:10.1109/ISIE.2007.4374856.

[21] M. O. Stitson, J. A. E. Weston, A. Gammerman, V. Vovk, and V. Vapnik. Theory of support vector machines. Technical Report CSD-TR-96-17, Department of Computer Science, Royal Holloway University of London, 1996.

[22] N. True. Vacant parking space detection in static images. Report, course CSE 190-A: Projects in Vision & Learning, University of California, San Diego, 2007. http://cseweb.ucsd.edu/classes/wi07/cse190-a/.

[23] M. Tschentscher, C. Koch, M. König, J. Salmen, and M. Schlipsing. Scalable real-time parking lot classification: An evaluation of image features and supervised learning algorithms. In *Proc. Int. Joint Conf. on Neural Networks (IJCNN 2015)*, pages 1–8. IEEE, 2015. doi:10.1109/IJCNN.2015.7280319.

[24] M. Tschentscher and M. Neuhausen. Video-based parking space detection. In *Proc. Forum Bauinformatik*, pages 159–166, 2012.

[25] R. Yusnita, F. Norbaya, and N. Basharuddin. Intelligent parking space detection system based on image processing. *International Journal of Innovation, Management and Technology*, 3(3):232–235, 2012. doi:10.7763/IJIMT.2012.V3.228. http://www.ijimt.org/show-37-455-1.html.

Rykhard Bohush Graduated from Polotsk State University in 1997. In 2002 he got his PhD in the field of Information Processing at the Institute of Engineering Cybernetics, the National Academy of Sciences of Belarus. Head of Computer Systems and Networks Department of Polotsk State University. His scientific interests include image and video processing, object representation and recognition, intelligent systems, digital steganography.

Pavel Yarashevich Born in 1991. In 2016 he got his the master degree in the field of Mathematical Modelling, Numerical Methods and Complexes of Programs. Currently he works as assistant of Computer Systems and Networks Department in PSU. Research interests are computer graphics processing, cryptography and programming.

Sergey Ablameyko DipMath in 1978, PhD in 1984, DSc in 1990, Prof. in 1992. Professor of Belarusian State University. His scientific interests are: image analysis, pattern recognition, digital geometry, knowledge based systems, geographical information systems, medical imaging. He is in Editorial Board of Pattern Recognition Letters, Pattern Recognition and Image Analysis and many other international and national journals. He is a senior member of IEEE, Fellow of IAPR, Fellow of Belarusian Engineering Academy, Academician of National Academy of Sciences of Belarus, Academician of the European Academy, and others. He was a First Vice-President of International Association for Pattern Recognition IAPR (2006-2008), President of Belarusian Association for Image Analysis and Recognition.

Tatiana Kalganova Graduated from Belarusian State University of Informatics and Radio-electronics. In 2000 she got her PhD in the field of Information Processing at the Napier University. Lecturer at Brunel University London. The main research fields of Dr. Kalganova are in Artificial Intelligence and its real-life applications: Evolutionary Design and Optimisation, Evolvable hardware, Modelling and optimisation of Large Systems, Operational research, Robotics, Swarm optimisation.