Development of Concurrent Object-Oriented Logic Programming Platform for the Intelligent Monitoring of Anomalous Human Activities

Alexei A. Morozov[1,4(✉)], Abhishek Vaish[2], Alexander F. Polupanov[1,4], Vyacheslav E. Antciperov[1], Igor I. Lychkov[3], Aleksandr N. Alfimtsev[3], and Vladimir V. Deviatkov[3]

[1] Kotel'nikov Institute of Radio Engineering and Electronics of RAS, Moscow, Russia
[2] Indian Institute of Information Technology, Allahabad, India
[3] Bauman Moscow State Technical University, Moscow, Russia
[4] Moscow State University of Psychology & Education, Moscow, Russia
morozov@cplire.ru

Abstract. The logic programming approach to the intelligent monitoring of anomalous human activity is considered. The main idea of this approach is to use first order logic for describing abstract concepts of anomalous human activity, i.e. brawl, sudden attack, armed attack, leaving object, loitering, pickpocketing, personal theft, immobile person, etc. We have created a research led software platform based on the Actor Prolog concurrent object-oriented logic language and a state-of-the-art Prolog-to-Java translator for examining the intelligent visual surveillance. A method of logical rules creation is considered in relation to the analysis of anomalous human behavior. The problem of creation of special built-in classes of Actor Prolog for the low-level video processing is discussed.

Keywords: Anomalous human activity · Intelligent visual surveillance · Object-oriented concurrent logic programming · Actor Prolog

1 Introduction

Human activity recognition is a rapid growing research area with important application domains including security and anti-terrorist issues [1,6,7]. Recently logic programming was recognized as a promising approach for dynamic visual scenes analysis [4,8,17–19]. The idea of the logic programming approach is in usage of logical rules for description and analysis of people activities. To approach the problem, knowledge about object co-ordinates and properties, scene geometry, and human body constraints is encoded in the form of certain rules in a logic programming language and is applied to the output of low-level object/feature detectors. There are several studies based on this idea. In [4] a system was designed for recognition of so-called long-term activities (such as fighting and meeting) as temporal combinations of short-term activities (walking, running, inactive, etc.) using a logic programming implementation of the Event Calculus.

G. Plantier et al. (Eds.): BIOSTEC 2014, CCIS 511, pp. 82–97, 2015.
DOI: 10.1007/978-3-319-26129-4_6

The ProbLog state-of-the-art probabilistic logic programming language was used to handle the uncertainty that occurs in human activity recognition. In [19] an extension of predicate logic with the bilattice formalism that permits processing of uncertainty in the reasoning was proposed. The VidMAP visual surveillance system that combines real time computer vision algorithms with the Prolog based logic programming had been proposed by the same team. S. O'Hara [18] communicated the VERSA general-purpose framework for defining and recognizing events in live or recorded surveillance video streams. According to [18], VERSA ensures more advanced spatial and temporal reasoning than VidMAP and is based on SWI-Prolog. F.A. Machot et al. [8] have proposed real time complex audio-video event detection based on the Answer Set Programming approach. The results indicate that this solution is robust and can easily be run on a chip.

Research indicates that conventional approaches to human behavior recognition include low-level and high-level stages of video processing. In this paper, we addressed the problem of the high-level semantic analysis of people activity. We have created a research led software platform based on the Actor Prolog concurrent object-oriented logic language [9–14] and a state-of-the-art Prolog-to-Java translator [15] for implementation of the logical inference on video scenes. The Prolog-to-Java translator provides means for a high-level concurrent programming and a direct access to the low-level processing procedures written in Java.

In the case of simple human behavior, a set of logic program rules can be created manually on the basis of a priori knowledge of the particular behavior features, for example, speed of moving, but in the case of complex spatio-temporal behavior, special methods of automatic logical rules creation are to be developed.

We have described our first experiments in the area of human activity recognition in Sect. 2. The problem of creation of special built-in classes of the Actor Prolog logic language for the low-level video processing is discussed in Sect. 3. A method of logical rules creation based on a hierarchy of fuzzy finite state automata is briefly considered in Sect. 4.

2 Logical Analysis of Manually Marked Videos

On the first stage of the research, we have performed several experiments on analysis of manually marked videos that is traditional approach in the area. The CAVIAR data sets [5] were used. The CAVIAR data sets are annotated using the XML-based Computer Vision Markup Language (CVML). The structure of CVML is simple enough, so we read it using the '*WebReceptor*' built-in class of the Actor Prolog for XML/HTML parsing. The CVML annotations contain information about co-ordinates of separate persons and groups of persons in videos. So, our experiments have pursued the following goals:

1. To check if the Actor Prolog system is fast enough to process videos in real time even without performing low-level analysis.
2. To check if there is enough information about the positions of persons for accurate estimation of the velocity and the acceleration of separate personages in the video scene.

Fig. 1. An example of CAVIAR video with a case of abrupt motions.

Fig. 2. The logic program has recognized that two persons were fighting.

The latter issue is important because the accurate estimation of the velocity and/or acceleration opens a way for the recognition of so-called abrupt motions of objects [4]. This kind of motions is necessary for recognition of several long-term activities (such as fighting or sudden attack), though recognition of abrupt motions is not usually provided by standard low-level analyzing procedures. The abrupt motions are not marked in the CAVIAR annotations as well.

An example of abrupt motion recognition is shown in Figs. 1 and 2. A program written in Actor Prolog uses given co-ordinates of two persons to estimate the distance between them and the 2-nd derivative of the co-ordinates to detect abrupt motions.

A logical rule describes an abnormal behavior (fighting) as a conjunction of two conditions:

1. Several persons have met sometime and somewhere.
2. After that they implement abrupt motions.

The text of the logic program is not given here for brevity. After recognition of these two conditions, the logic program has decided that there was a case of a scuffle and has indicated the fighting persons by a red rectangle (see Fig. 2).

This example demonstrates a possibility of recognition of video scenes semantics using the logical inference on results of the low-level recognition of separate objects; however one can see the following bottle-neck of the approach.

Manually defined co-ordinates of the objects were used for estimation of their acceleration and nobody can guarantee that automatic low-level procedures will provide exact values of co-ordinates that are good enough for numerical differentiation. So, the discussion on the high-level recognition procedures is impossible without consideration of underlying low-level recognition methods.

The second issue of this example is whether it is useful to separate the recognition process into concurrent sub-processes implementing different stages of the high-level logical inference. Working intensities of different sub-processes are different. For example, the differentiation of co-ordinates requires more computational resources and another sub-process that implements recognition of people behavior could wait for the results of differentiation.

3 Advanced Logic Analysis of Video Scenes

On the next stage of the research, we have implemented experiments on video analysis based on the automatically extracted information about co-ordinates and velocity of blobs in video scenes.

3.1 Implementation of Base Low-Level Video Processing Procedures

A promising approach for implementation of the low-level recognition procedures in a logic language is usage of the OpenCV computer vision library and we are planning to link Actor Prolog with the JavaCV library that is a Java interface to OpenCV. Nevertheless, Java has enough standard tools to solve simple image processing/recognition problems and we have started our experiments with pure Java.

We have created low-level Java procedures [16] that implement several basic recognition tasks:

1. Background subtraction;
2. Discrimination of foreground blobs;

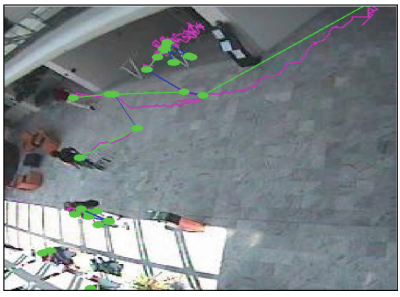

Fig. 3. A low-level procedure discriminates trajectories (violet lines) of objects and moments of their interactions (green circle marks and blue links).

3. Tracking of the foreground blobs over time;
4. Detection of interactions between the blobs.

The first experiments have demonstrated clearly that the exact estimation of an object velocity was impossible without taking into account the interactions of objects (see Fig. 3), because of edge effects of differentiation in the interaction points.

After implementation of the object interactions check, we have got tracks that were accurate enough to determine whether a person is walking or running. In the next section, we will describe an approach to lower boundary estimation of blob velocity and discuss its possible application to the detection of anomalous behavior of people.

3.2 A Fast Algorithm for Estimation of Object Velocity

At this stage of research, we use standard method of recovering physical co-ordinates of objects in a scene, based on computing inverse matrix of projective transformation by co-ordinates of four defining points. A well-known disadvantage of this method is so-called ground plane assumption, that is, one cannot compute co-ordinates of body parts that are situated outside from a pre-defined plane. Usually, this pre-defined plane is a ground one and we can estimate properly the co-ordinates of person's shoes only. Generally speaking, this problem cannot be avoided in the framework of single camera approach, nevertheless, our idea is in usage of object velocity (but not co-ordinates) for the anomalous behavior detection and this point is exploited in the following algorithm.

We consider simplified rectangle blobs describing moving objects in the scene (see example in Fig. 5). Co-ordinates of every corner of the blob are recovered using the inverse matrix of the projective transformation. Then, one compares the co-ordinates of corresponding corners of the blob in consecutive frames and calculates the first derivative of their co-ordinates. The idea is that only the corners situated in the ground plane give realistic estimations of velocity and other corners give greater values because upper parts of body visually correspond to more distant points in the ground plane. So, we exploit this property of projective transformation and accept the lower boundary estimation of object velocity as a minimal value of velocities (V_{11}, V_{12}, V_{21}, and V_{22}) of four blob corners:

$$V \approx min(abs(V_{11}), abs(V_{12}), abs(V_{21}), abs(V_{22}))$$

Note, that the algorithm does not recover the direction of blob movement. The precision of the estimation of the blob velocity is not very high too, because of the approximate nature of the algorithm. Moreover, the automatic detection of blob shapes often produce illegal co-ordinates of blob corners because of common problems with shades, obstacles, digital noise etc., and this issue is an additional source of errors in the velocity estimation.

We have applied a median filtering to eliminate outliers in the velocity function. For instance, in the example in Fig. 4, the seven point median filter ensures

an estimation of blob velocity that is good enough for discrimination of running and walking persons in the scene.

We have implemented this algorithm of velocity estimation in the library [16] of low-level methods of image analysis of the Actor Prolog system and use it in our experiments with the intelligent visual surveillance.

3.3 Creation of a Built-In Class of Actor Prolog

We have developed a special built-in class of the Actor Prolog language that uses formerly described low-level recognition procedures. The '*ImageSubtractor*' class of Actor Prolog implements the following tasks:

1. Video frames pre-processing including 2D-gaussian filtering, 2D-rank filtering, and background subtraction.
2. Recognition of moving blobs and creation of Prolog data structures describing the co-ordinates of the blobs in each moment.
3. Recognition of tracks of blob motions and creation of Prolog data structures describing the co-ordinates and the velocity of the blobs. The tracks are divided into separate segments; there are points of interaction between the blobs at the ends of a segment.
4. Recognition and ejection of immovable and slowly moving objects. This feature is based on a simple fuzzy inference on the attributes of the tracks (the co-ordinates of the tracks and the average velocities of the blobs are considered).
5. Recognition of connected graphs of linked tracks of blob motions and creation of Prolog data structures describing the co-ordinates and the velocity of the blobs.

We consider two tracks as linked if there are interactions between the blobs of these tracks. In some applications, it is useful to eject tracks of immovable and slowly moving objects from the graphs before further processing of the video scenes.

3.4 An Example of Anomalous Behavior Detection

Let us consider an example of logical inference on video. The input of the logic program written in Actor Prolog is the *Fight_RunAway*1 CAVIAR [5] dataset sample (the sequence of JPEG files is used). The program will use no additional information about the content of the video scene, but only co-ordinates of four defining points in the ground plane (the points are provided by CAVIAR). The total text of the logic program is not given here for brevity; we will discuss only the program structure and main stages of data analysis.

The logic program creates two concurrent processes with different priorities (see [12] for details about Actor Prolog model of asynchronous concurrent computations). The first process has higher priority and implements video data gathering. This process reads JPEG files and sends them to the instance of the

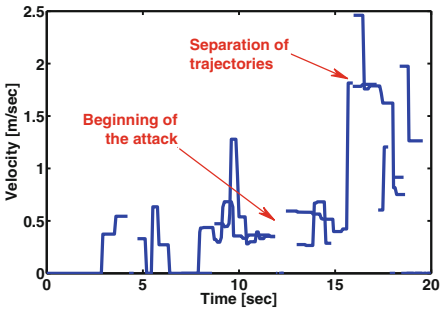

Fig. 4. An example of estimation of velocities of blobs in a visual scene (see Fig. 3). The X-axis denotes time in seconds and the Y-axis denotes lower boundary estimation of blob velocities (m/sec). One can recognize walking persons (before the beginning of the attack) and running persons (after the separation of the trajectories of persons) in the diagram.

'*ImageSubtractor*' predefined class that implements all low-level processing of video frames. The sampling rate of the video is 25 frames per second, so the process loads a new JPEG file every 40 ms.

The second concurrent process implements logical analysis of collected information and outputs results of the analysis. The analysis of video frames requires more computational resources, but it does not suspend the low-level analysis, because the second process has less priority. The analysis includes extraction of blobs, tracking of the blobs over time, detection of interactions between the blobs, creation of connected graphs of linked tracks of blobs, and estimation of average velocity of blobs in separate segments of tracks (see Fig. 4). This information is received by the logic program in a form of Prolog terms describing the list of connected graphs.

The '*ImageSubtractor*' class uses the following data structures for describing connected graphs of tracks (note, that the DOMAINS, the PREDICATES, and the CLAUSES program sections in Actor Prolog have traditional meaning developed in the Turbo/PDC Prolog systems):

```
DOMAINS:
ConnectedGraph  = GraphEdge*.
GraphEdge   = {
       frame1: INTEGER,
       x1: INTEGER,
       y1: INTEGER,
       frame2: INTEGER,
       x2: INTEGER,
       y2: INTEGER,
       inputs: EdgeNumbers,
       outputs: EdgeNumbers,
       identifier: INTEGER,
```

```
        coordinates: TrackOfBlob,
        mean_velocity: REAL
        }.
EdgeNumbers = EdgeNumber*.
EdgeNumber  = INTEGER.
TrackOfBlob = BlobCoordinates*.
BlobCoordinates = {
        frame: FrameNumber,
        x: INTEGER,
        y: INTEGER,
        width: INTEGER,
        height: INTEGER,
        velocity: REAL
        }.
```

That is, connected graph is a list of underdetermined sets [9] denoting separate edges of the graph. The nodes of the graph correspond to points where tracks cross, and the edges are pieces of tracks between such points. Every edge is directed and has the following attributes: numbers of first and last frames ($frame1$, $frame2$), co-ordinates of first and last points ($x1$, $y1$, $x2$, and $y2$), a list of edge numbers that are predecessors of the edge ($inputs$), a list of edge numbers that are followers of the edge ($outputs$), the identifier of corresponding blob (an integer $identifier$), a list of sets describing the co-ordinates and the velocity of the blob in different moments of time ($coordinates$), and an average velocity of the blob in this edge of the graph ($mean\_velocity$).

Fig. 5. A logical inference has found a possible case of a sudden attack in the graph of blob trajectories. Rectangle blobs are depicted by yellow lines, blob trajectories are depicted by red lined, moments of interactions between blobs are depicted by green circles and blue links (Color figure online).

The logic program checks the graph and looks for the following pattern of interaction among several persons: if two or more persons met somewhere in the scene, and one of them has walked (not run) before this meeting, and one of them has run (not walked) after this meeting, the program considers this scenario as a

kind of a running away and a probable case of a sudden attack or a theft. So, the program alarms if this kind of sub-graph is detected in the total connected graph of tracks. In this case, the program draws all tracks of the inspected graph in red and outputs the "Attention!" warning in the middle of the screen (see Fig. 5).

One can describe formally the concept of a running away using defined connected graph data type.

```
PREDICATES:
is_a_running_away(
  ConnectedGraph,
  ConnectedGraph,
  ConnectedGraphEdge,
  ConnectedGraphEdge,
  ConnectedGraphEdge) - (i,i,o,o,o);
```

We will define the $is\_a\_running\_away(G, G, P1, E, P2)$ predicate with the following arguments: G is a graph to be analyzed (the same data structure is used in the first and the second arguments), E is an edge of the graph corresponding to a probable incident, $P1$ is an edge of the graph that is a predecessor of E, $P2$ is an edge that is a follower of E. Note that G is an input argument of the predicate and $P1$, E, and $P2$ are output ones. Here is an Actor Prolog program code with brief explanations:

```
CLAUSES:
is_a_running_away([E|_],G,P1,E,P2):-
  E == {inputs:I,outputs:O|_},
  O == [_,_|_],
  walking_person(I,G,P1),
  running_person(O,G,P2),!.
is_a_running_away([_|Rest],G,P1,E,P2):-
  is_a_running_away(Rest,G,P1,E,P2).
walking_person([N|_],G,P):-
  get_edge(N,G,E),
  is_a_walking_person(E,G,P),!.
walking_person([_|Rest],G,P):-
  walking_person(Rest,G,P).
running_person([N|_],G,P):-
  get_edge(N,G,E),
  is_a_running_person(E,G,P),!.
running_person([_|Rest],G,P):-
  running_person(Rest,G,P).
get_edge(1,[Edge|_],Edge):-!.
get_edge(N,[_|Rest],Edge):-
  N > 0,
  get_edge(N-1,Rest,Edge).
```

In other words, the graph contains a case of a running away if there is an edge E in the graph that has a predecessor $P1$ corresponding to a walking person and

a follower $P2$ that corresponds to a running person. It is expected also that E has more than one follower (it is a case of a branching in the graph)[1].

```
is_a_walking_person(E,_,E):-
  E == {mean_velocity:V|_},
  V <= 0.5,!.
is_a_walking_person(E,G,P):-
  E == {inputs:I|_},
  walking_person(I,G,P).
```

That is, the graph edge corresponds to a walking person if the average blob velocity in this edge is less or equal to 0.5 m/s, or the edge has a predecessor that corresponds to a walking person.

```
is_a_running_person(E,_,E):-
  E == {mean_velocity:V|_},
  V >= 1.0,!.
is_a_running_person(E,G,P):-
  E == {outputs:O|_},
  running_person(O,G,P).
```

The graph edge corresponds to a running person if the average velocity in this edge is more or equal to 1 m/s, or the edge has a follower corresponding to a running person.

Note that aforementioned rules use plain numerical thresholds to discriminate walking and running persons for brevity. Better discrimination could be ensured by a kind of a fuzzy check, which can be easily implemented using arithmetical means of standard Prolog.

This example illustrates the possible scheme of a logic program implementing all necessary stages of video processing including video information gathering, low-level image analysis, high-level logical inference on the video scene, and reporting the results of the intelligent visual surveillance.

4 A Method of Logical Rules Creation

The logical rules considered in the previous section were created manually on the basis of a priori knowledge of the particular behavior, but we would like to create logical rules automatically in cases of complex spatio-temporal behavior. In this section, we describe the method of logical rules creation [3] based on a hierarchy of fuzzy finite state automata.

Let $T = \{t_i | t_i \in N\}$ be a discrete set of time instances with constant intervals $\Delta t = t_{i+1} - t_i$ between consecutive time instances, where $[t_s, t_e] = \{t | t_s \leq t \leq t_e\}$ is a time interval T. Suppose that each 0^{th} level feature (a speed or a position)

[1] Note, that in the Actor Prolog language, the operator == corresponds to the ordinary equality = of the standard Prolog.

of each moving object θ from a set $\{\theta^1, \theta^2, \ldots, \theta^l\}$ at a time instance t equals $y_{i_0}(\theta_t), i_0 \in \{1, \ldots, m_0\}$, that we call a feature sample. Samples $Y_{i_0}[\theta_{t_s}, \theta_{t_e}] = \langle y_{i_0}(\theta_{t_s}), \ldots, y_{i_0}(\theta_{t_e})\rangle, i_0 \in \{1, \ldots, m_0\}$ of a single 0th level feature at several consecutive time instances $t_s, \ldots, t_e$ during a $[t_s, t_e]$ time interval are called a trend.

Let us consider the following situation. Two persons walk alongside two roads that are perpendicularly directed towards their meeting point (intersection). While person A is far from the intersection, person B slows down waiting for person A. When person A enters the intersection, person B accelerates and runs into person A.

Let us formalize the persons' behavior. Let persons A and B walk along perpendicular lines with the intersection point O. Let xOy be a rectangular co-ordinate system such that the Ox axis corresponds to the A person and the Oy axis corresponds to the B person (Fig. 6). Let us consider each person as a rectangle and the co-ordinates of the centroid of the rectangle as the co-ordinates of the person. Suppose that persons move strictly along the co-ordinate axes and current positions of persons A and B can be determined by single co-ordinates $y_s(\theta_t^A)$ and $y_s(\theta_t^B)$ respectively. The $y_s(\theta_t^A)$, $y_s(\theta_t^B)$ co-ordinates of the persons A and B are considered as first features. The $y_v(\theta_t^A)$, $y_v(\theta_t^B)$ speed values of the persons are considered as second features.

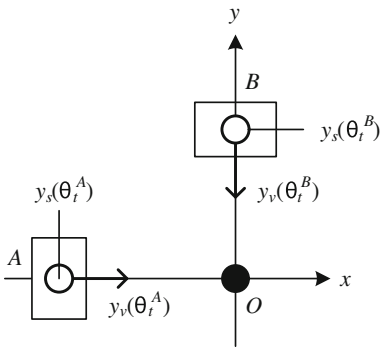

Fig. 6. Two persons in the rectangular co-ordinate system.

In order to estimate velocities one should specify an observation time interval $[t_s, t_e] = \{t | t_s \leq t \leq t_e\}$ and time instances $t_i \in [t_s, t_e]$. Linguistic variables are specified and behavior template models are defined as in [3]. Let $position(\theta^A)$, $speed(\theta^A)$ and $position(\theta^B)$, $speed(\theta^B)$ be linguistic variables that describe positions and speed values of persons A and B. The $position(\theta^A)$ and $position(\theta^B)$ linguistic variables assume linguistic values $far(\theta^X)$, $near(\theta^X)$, and $inside(\theta^X)$. The $speed(\theta^A)$ and $speed(\theta^B)$ linguistic variables assume linguistic values $high(\theta^X)$ and $low(\theta^X)$.

Fuzzy sets corresponding to linguistic values $far(\theta^X)$, $near(\theta^X)$, $inside(\theta^X)$ and $high(\theta^X)$, $low(\theta^X)$ are shown in Fig. 7. Fuzzy sets shown in Fig. 7 are used

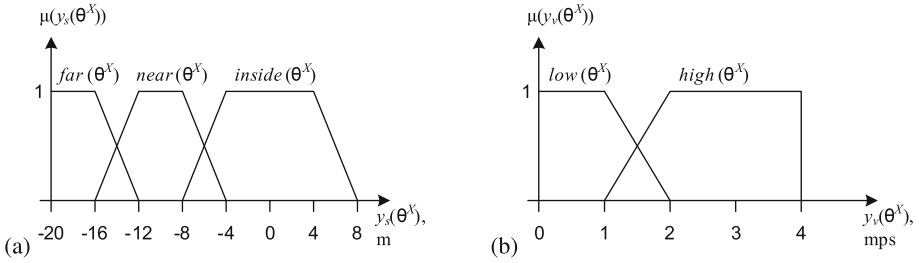

Fig. 7. (a) Fuzzy sets corresponding to linguistic values $far(\theta^X)$, $near(\theta^X)$, and $inside(\theta^X)$; (b) Fuzzy sets corresponding to linguistic values $low(\theta^X)$ and $high(\theta^X)$.

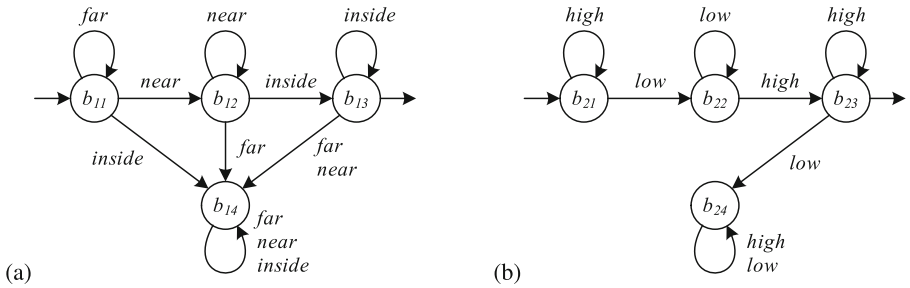

Fig. 8. (a) The $M_{pos(\theta^X)}$ first level automaton; (b) The $M_{speed(\theta^A)}$ first level automaton.

to define first level template automata $M_{pos(\theta^X)}$, $M_{speed(\theta^A)}$, and $M_{speed(\theta^B)}$ that describe position and speed of persons A and B.

An automaton $M_{pos(\theta^X)}$ shown in Fig. 8(a) determines a sequence of the linguistic values $[far(\theta^X), near(\theta^X), inside(\theta^X)]$ of the variable $position(\theta^X)$. The automaton graph is based on a chain of allowed states $b_{11} - b_{12} - b_{13}$ corresponding to the values of the determined sequence. b_{11} is the initial state (marked by the input arrow in Fig. 8(a)) and b_{13} is the final state of the automaton (the output arrow in Fig. 8(a)). State transitions are specified below. If the input linguistic value corresponds to the current state, then automaton retains its current state. If the input linguistic value corresponds to the next allowed state, then the automaton moves to that state. Automaton moves to the b_{14} denied state if an input linguistic value violates the allowed linguistic values sequence of the automaton. Note that the automaton cannot leave the denied state.

Automata $M_{speed(\theta^A)}$ and $M_{speed(\theta^B)}$ are presented in Figs. 8(b) and 9(a).

Let us define the second level template automaton that describes the joint persons' behavior. Let $condition(\theta^A, \theta^B)$ be a linguistic variable that assumes linguistic values $safe(\theta^A, \theta^B)$, $warning(\theta^A, \theta^B)$, and $unsafe(\theta^A, \theta^B)$, where

$$safe(\theta_t^A, \theta_t^B) = [pos(\theta_t^A) = far(\theta^A)] \wedge [pos(\theta_t^B) = far(\theta^B)] \quad (1)$$

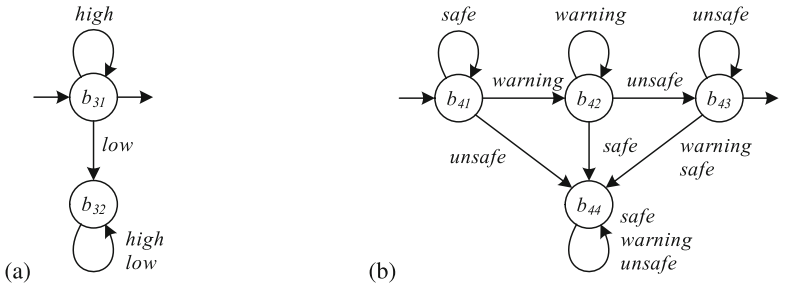

Fig. 9. (a) The $M_{speed(\theta^B)}$ first level automaton; (b) The $M_{condition(\theta^A,\theta^B)}$ second level automaton.

$$warning(\theta_t^A, \theta_t^B) = ([pos(\theta_t^A) = near(\theta^A)] \wedge [speed(\theta_t^B) = high(\theta^B)]) \vee ([speed(\theta_t^A) = high(\theta^A)] \wedge [pos(\theta_t^B) = near(\theta^B)]) \quad (2)$$

$$unsafe(\theta_t^A, \theta_t^B) = [pos(\theta_t^A) = inside(\theta^A)] \wedge [pos(\theta_t^B) = inside(\theta^B)] \quad (3)$$

Each linguistic value of the $condition(\theta^A, \theta^B)$ linguistic variable corresponds to a composite fuzzy set. The multidimensional domain of the composite fuzzy set is a Cartesian product of the domains of the corresponding fuzzy sets [3]. According to Eq. (1), the domain of the $safe(\theta_t^A, \theta_t^B)$ linguistic value is defined as follows:

$$\mathrm{dom}[safe(\theta_t^A, \theta_t^B)] = \mathrm{dom}[far(\theta^A)] \times \mathrm{dom}[far(\theta^B)],$$

where $\mathrm{dom}[E]$ is the domain of a fuzzy set E and $\times$ is the Cartesian product.

A membership function is expressed as follows:

$$a = b \wedge c \Longrightarrow R_a(y_b, y_c) = \min\{R_b(y_b), R_c(y_c)\} \quad (4)$$

$$a = b \vee c \Longrightarrow R_a(y_b, y_c) = \max\{R_b(y_b), R_c(y_c)\} \quad (5)$$

$$a = \neg b \Longrightarrow R_a(y_b) = 1 - R_b(y_b) \quad (6)$$

where a, b, and c are fuzzy sets; R_E is a membership function of a fuzzy set E, determined on the $\mathrm{dom}[E]$ domain; $y_b \in \mathrm{dom}[b]$ and $y_c \in \mathrm{dom}[c]$ are feature values from corresponding domains.

According to Eq. (4), the membership function of a composite fuzzy set specified on the conjunction of two fuzzy sets is equal to minimum value of the membership functions of these fuzzy sets. The second level template automaton $M_{condition(\theta^A,\theta^B)}$ that describes the joint persons' behavior is shown in Fig. 9(b).

The computation scheme of the recognition process is presented in Fig. 10. It includes five units for evaluation and processing of linguistic variables arranged in two levels. Each unit computes value of corresponding linguistic variable and inputs it to the corresponding automaton.

The recognition of the situation is implemented in the following way. Initially, all first and second level automata have initial states. Then feature samples for

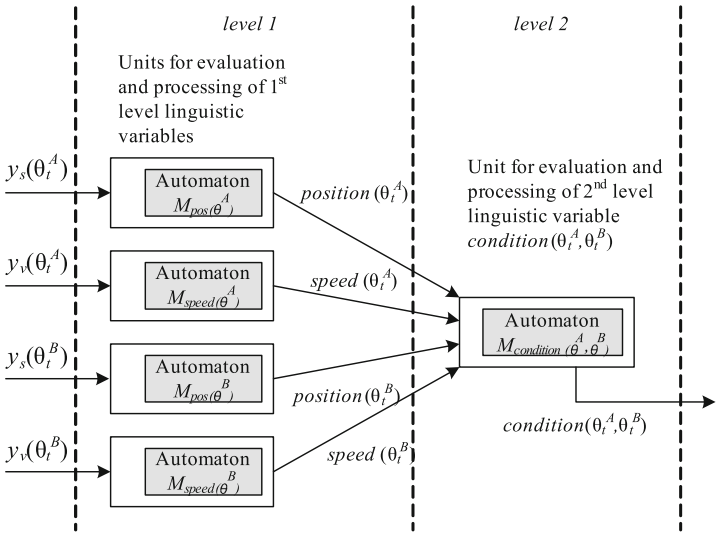

Fig. 10. The computation scheme of recognition.

consecutive time instances $t_i \in [t_s, t_e]$ are passed by turn into the first level units for evaluation and processing of the first level linguistic variables. The first level units compute values of linguistic variables and pass them into the second level unit for evaluation and processing of the second level linguistic variable $condition(\theta^A, \theta^B)$. The first and the second level automata may change their states during the operation. Situation is recognized if all first and second level automata have moved to their final states after the end of the processing of all feature samples. Situation is not recognized if one automaton has not moved to the final state at least.

The computation scheme based on the fuzzy finite automata can easily be converted to a logic program; one can use standard techniques of transforming finite state machines into the logic programs [2].

5 Conclusions

We have created a research led software platform based on the Actor Prolog concurrent object-oriented logic language and a state-of-the-art Prolog-to-Java translator for examining the intelligent visual surveillance. The platform includes the Actor Prolog logic programming system and an open source Java library of Actor Prolog built-in classes [16]. It is supposed to be complete for facilitation of research in the field of intelligent monitoring of anomalous people activity and studying logical description and analysis of people behavior.

Our study has demonstrated that translation from a concurrent object-oriented logic language to Java is a promising approach for application of the

logic programming to the problem of intelligent monitoring of people activity; the Actor Prolog logic programming system is suitable for this purpose and ensures essential separation of the recognition process into concurrent subprocesses implementing different stages of high-level analysis.

In the paper, a specialized built-in class of the Actor Prolog language implementing simple pre-processing of video data and low-level analysis of video scenes concerning the problem of intelligent monitoring of people activity was demonstrated. We have implemented a simple analysis of videos based on automatically extracted information on the co-ordinates and velocities of blobs in the video scene. It was shown that robust recognition of abrupt motions is impossible without accurate low-level recognition of body parts (face, hands). This is a subject of further studies.

A method of logical rules creation is proposed for situation analysis in the environment of moving objects. A formal method for representing situations using hierarchy of fuzzy finite state automata was considered. Future work will include comprehensive testing of the proposed methods on massive datasets and development of fully automatic method for situation representation using real feature trends.

Acknowledgements. We acknowledge a partial financial support from the Russian Foundation for Basic Research, grant No 13-07-92694, and Department of Science and Technology, Govt. of India, grant No DST-RFBR P-159.

References

1. Aggarwal, J., Ryoo, M.: Human activity analysis: a review. ACM Comput. Surv. (CSUR) **43**(3), 16:1–16:43 (2011)
2. Bratko, I.: Prolog Programming for Artificial Intelligence. Addison-Wesley, Boston (1986)
3. Devyatkov, V.: Multiagent hierarchical recognition on the basis of fuzzy situation calculus. Vestnik, Journal of the Bauman Moscow State Technical University, Natural Science & Engineering, pp. 129–152. Vestnik MGTU, Moscow (2005)
4. Filippou, J., Artikis, A., Skarlatidis, A., Paliouras, G.: A probabilistic logic programming event calculus (2012). http://arxiv.org/abs/1204.1851
5. Fisher, R.: CAVIAR Test Case Scenarios. The EC funded project IST 2001 37540 (2007). http://homepages.inf.ed.ac.uk/rbf/CAVIAR/
6. Junior, J., Musse, S., Jung, C.: Crowd analysis using computer vision techniques. A survey. IEEE Signal Process. Mag. **27**(5), 66–77 (2010)
7. Kim, I., Choi, H., Yi, K., Choi, J., Kong, S.: Intelligent visual surveillance–a survey. Int. J. Control Autom. Syst. **8**(5), 926–939 (2010)
8. Machot, F., Kyamakya, K., Dieber, B., Rinner, B.: Real time complex event detection for resource-limited multimedia sensor networks. In: Workshop on Activity Monitoring by Multi-camera Surveillance Systems (AMMCSS), pp. 468–473 (2011)
9. Morozov, A.A.: Actor Prolog: an object-oriented language with the classical declarative semantics. In: Sagonas, K., Tarau, P. (eds.) IDL 1999, pp. 39–53. Paris, France (1999). http://www.cplire.ru/Lab144/paris.pdf

10. Morozov, A.A.: On semantic link between logic, object-oriented, functional, and constraint programming. In: MultiCPL 2002. Ithaca (2002). http://www.cplire.ru/Lab144/multicpl.pdf
11. Morozov, A.A.: Development and application of logical actors mathematical apparatus for logic programming of web agents. In: Palamidessi, C. (ed.) ICLP 2003. LNCS, vol. 2916, pp. 494–495. Springer, Heidelberg (2003)
12. Morozov, A.A.: Logic object-oriented model of asynchronous concurrent computations. Pattern Recognit. Image Anal. **13**(4), 640–649 (2003). http://www.cplire.ru/Lab144/pria640.pdf
13. Morozov, A.A.: Operational approach to the modified reasoning, based on the concept of repeated proving and logical actors. In: Salvador Abreu, V.S.C. (ed.) CICLOPS 2007, pp. 1–15. Porto, (2007). http://www.cplire.ru/Lab144/ciclops07.pdf
14. Morozov, A.A.: Visual logic programming method based on structural analysis and design technique. In: Dahl, V., Niemelä, I. (eds.) ICLP 2007. LNCS, vol. 4670, pp. 436–437. Springer, Heidelberg (2007)
15. Morozov, A.A.: Actor Prolog to Java translation (in Russian). IIP-9, pp. 696–698. Torus Press Moscow, Budva (2012)
16. Morozov, A.A.: A GitHub repository containing source codes of Actor Prolog built-in classes (including the Vision package) (2014). https://github.com/Morozov2012/actor-prolog-java-library
17. Morozov, A.A., Vaish, A., Polupanov, A.F, Antciperov, V.E., Lychkov, I.I., Alfimtsev, A.N., Deviatkov, V.V.: Development of concurrent object-oriented logic programming system to intelligent monitoring of anomalous human activities. In: Jr., A.C., Plantier, G., Schultz, T., Fred, A., Gamboa, H. (eds.) BIODEVICES 2014, pp. 53–62. SCITEPRESS (2014). http://www.cplire.ru/Lab144/biodevices2014.pdf
18. O'Hara, S.: VERSA–video event recognition for surveillance applications. M.S. thesis, University of Nebraska at Omaha (2008)
19. Shet, V., Singh, M., Bahlmann, C., Ramesh, V., Neumann, J., Davis, L.: Predicate logic based image grammars for complex pattern recognition. Int. J. Comput. Vis. **93**(2), 141–161 (2011)