DESENVOLVIMENTO DE MEIOS DE PROGRAMAÇÃO DA LÓGICA DO AGENTE PARA VIGILÂNCIA DE VÍDEO INTELIGENTE MULTICHANEL

Aleksey A. Morozov, Olga S. Sushkova, Nadezhda G. Petrova

Kotelnikov Instituto de Radioengenharia e Eletrônica de RAS, http://www.cplire.ru
Moscou 125009, Federação Russa

Margarita N. Khokhlova, Cyrille Mignot

Universidade da Borgonha Franche-Comte, http://www.ubfc.fr

Dijon 21000, França

morozov@cplire.ru, o.sushkova@mail.ru, margokhokhlova@gmail.com, cyrille.migniot@u-bourgogne.fr, petrova@cplire.ru

Resumo. **Este artigo propõe os meios experimentais desenvolvidos na linguagem de lógica paralela orientada a objetos Actor Prolog para implementação de sistemas de vigilância visual inteligente multicanal heterogêneo. Esses meios são examinados pela instância de um programa lógico para monitoramento permanente da temperatura de partes do corpo das pessoas na área de vigilância visual. O programa lógico implementa uma fusão de dados heterogêneos adquiridos por dois dispositivos: (1) as coordenadas 3D do corpo humano são medidas usando uma câmera de tempo de voo (ToF), as coordenadas 3D do esqueleto do corpo humano são calculadas na base de essas coordenadas 3D do corpo; (2) um vídeo térmico é adquirido usando uma câmera de imagem térmica. Classes especiais incorporadas são desenvolvidas na linguagem Actor Prolog para a aquisição e análise de dados de vídeo 3D e 2D: as classes Kinect e** *KinectBuffer* **implementam a interação do programa lógico com a câmera ToF do dispositivo Kinect 2; a** classe *BufferedScene* **implementa o armazenamento e transferência de imagens 3D; a classe Canvas3D implementa gráficos 3D baseados na biblioteca de código aberto Java3D. No exemplo em consideração, o vídeo térmico é projetado na superfície 3D do corpo humano; então a temperatura do corpo humano é projetada para os vértices e bordas do esqueleto. Um agente lógico especial (isto é, o programa lógico escrito em Actor Prolog) implementa essas operações em tempo real e transfere os dados para outro agente lógico. O último agente implementa uma média temporal da temperatura dos esqueletos humanos e exibe imagens 3D coloridas dos esqueletos; a temperatura média dos vértices e arestas dos esqueletos é representada pelas cores. Um método de chamadas remotas de predicados é utilizado para a interação dos agentes lógicos; este método foi desenvolvido e implementado no Actor Prolog para suportar o paradigma de programação de lógica de agente. Os meios de programação lógica em consideração são desenvolvidos com o objetivo de implementar a análise lógica da semântica das cenas de vídeo nos sistemas inteligentes de vigilância visual.**

Palavras-chave: **videovigilância inteligente; imagem térmica, Kinect, visão tridimensional; programação lógica orientada a objetos; a linguagem Actor Prolog; esqueletos; reconhecimento de eventos complexos; visão de máquina; visão técnica**

UDC 510.663; 519,68: 007,5; 519,68: 681.513.7; 681.3.06

Bibliografia - 44

referências RENSIT, 2018, 10(1):101-116

Recebido em 03.06.2018

DOI: 10.17725 / rensit.2018.10.101

Conteúdo

1. INTRODUÇÃO

Este artigo descreve o desenvolvimento de uma plataforma de software para distribuição/ análise lógica descentralizada de dados multicanais heterogêneos nos sistemas inteligentes de vigilância por vídeo. A plataforma é baseada na linguagem lógica orientada a objetos Actor Prolog. Esta plataforma é desenvolvida com o objetivo de implementar a análise lógica da semântica das cenas de vídeo. A ideia da análise semântica das cenas de vídeo sugere que um sistema de videovigilância inteligente não é apenas capaz de reconhecer, mas também de estimar a semântica de determinados objetos, ações e eventos no contexto de uma determinada cena de vídeo. Exemplos de análise semântica de vídeo são a análise do comportamento das pessoas e, em particular, a análise das interações sociais. A análise do comportamento das pessoas implica, ao contrário da análise da ação das pessoas, a análise do contexto, ou seja, circunstâncias suplementares ligadas ao lugar, ao tempo, eventos anteriores e posteriores, ações de outras pessoas, presença e ausência de objetos, etc.

A análise da semântica de cenas de vídeo tem óbvias áreas de aplicação na medicina, educação e arte, mas na opinião dos autores, a questão do dia é o reconhecimento de comportamentos suspeitos de pessoas pelos sistemas inteligentes de vigilância visual. Por exemplo, a análise das comunicações sociais entre as pessoas (apertos de mão, comportamento agressivo, etc.) é necessária para reconstruir o quadro completo do que acontece em um determinado edifício ou território que pode ajudar a reconhecer pessoas não autorizadas e/ou infratores. O reconhecimento do status social e dos papéis das pessoas pela aparência (por exemplo, detecção de funcionários, policiais, militares, etc.) ações (por exemplo, para reconhecer criminosos ou terroristas mascarados pelo uniforme da polícia).

Lógica matemática e programação lógica são meios convenientes para descrição e análise do contexto de eventos observados.

O problema de análise das cenas de vídeo é abordado em diversas áreas da matemática e informática, como lógica temporal e espacial, prova automática de teoremas em tempo real, inferência lógica distribuída e paralela, programação lógica orientada a objetos, etc. Com todas essas áreas de pesquisa, a possibilidade de uso prático desses meios é determinada por a presença de informações de origem que são suficientes para inferir conclusões lógicas úteis. Por exemplo, o aparecimento de sensores para aquisição de dados 3D (como câmeras de tempo de voo, câmeras estéreo, flash LIDARS, etc.) objetos. É óbvio que mais progressos na área estarão ligados ao uso de redes neurais para o reconhecimento de poses e objetos humanos. No entanto, os autores consideram a lógica matemática como a linguagem matemática mais adequada para a descrição declarativa do comportamento de pessoas complexas e outros eventos com base na combinação das descrições de eventos e ações elementares. Além disso, consideramos a programação lógica como a ferramenta mais útil, adaptável e relevante para a prática para a implementação da inferência lógica em tempo real.

Observe que a programação lógica não é uma abordagem única para a descrição lógica e análise da semântica das cenas de vídeo; existem sistemas inteligentes de vigilância visual baseados em regras, inferência lógica difusa, Markov

redes lógicas, etc. Uma característica distintiva da abordagem baseada em programação lógica para o desenvolvimento do sistema de vigilância por vídeo inteligente é que o programa lógico não é apenas um conjunto de regras/fórmulas lógicas, mas também instruções de processamento de dados. Em outras palavras, o programa lógico possui não apenas semântica declarativa, mas também operacional. Isso significa que o programador lógico tem a possibilidade de controlar a inferência lógica durante a análise do comportamento de determinada pessoa e/ou evento complexo para levar em conta a velocidade de processamento de dados e o tamanho dos dados a serem processados. Assim, a aplicação de uma linguagem lógica em vez de um conjunto simples de regras lógicas abre a possibilidade de usar inferência lógica de dados multicanal de forma eficiente na estrutura de sistemas inteligentes de vigilância por vídeo.

Este artigo aborda o desenvolvimento dos meios de programação orientada a objetos e lógica de agente no contexto do problema da análise semântica das cenas de vídeo. A experiência da aplicação da lógica de programação para a análise de dados de vídeo indica que esta atividade de pesquisa traz resultados perspicazes na área de análise de dados de vídeo e, ao mesmo tempo, dá novas ideias e meios na área de programação lógica. A questão é que a programação lógica e a primeira linguagem lógica Prolog foram projetadas e desenvolvidas como meio de processamento simbólico no âmbito da inteligência artificial.

Assim, o processamento de dados em tempo real e o processamento de matrizes de big data (incluindo imagens, vídeos e dados multimídia) eram tradicionalmente considerados áreas inadequadas para a aplicação das linguagens lógicas. Este fato deixou uma marca na sintaxe das linguagens lógicas, na implementação de compiladores e bibliotecas embutidas, etc. A experiência da aplicação da programação lógica em a área de videovigilância inteligente garantiu a possibilidade de descobrir gargalos nas linguagens lógicas e compiladores que abrem novas perspectivas para o desenvolvimento da programação orientada a objetos/lógica de agente e tradutores de Prolog para linguagens imperativas.

A programação lógica orientada a objetos é uma área de pesquisa que aborda o desenvolvimento de linguagens de programação orientadas a objetos e lógica simultaneamente. Esta direção de pesquisa é continuamente desenvolvida durante as últimas décadas, mas infelizmente ainda é uma direção exótica na estrutura da programação lógica; basta dizer que vários pesquisadores usam os termos objeto, classe e herança de maneiras diferentes. Essa prática provavelmente está implícita no fato de que tanto os matemáticos puros quanto os cientistas da computação estão envolvidos na área de programação lógica; os matemáticos muitas vezes consideram os recursos orientados a objetos como um aspecto não suficiente das linguagens lógicas e os programadores muitas vezes ignoram a pesquisa na área da semântica da teoria dos modelos das linguagens lógicas. Hoje a experiência da aplicação da programação lógica para a análise de dados multimídia estimula a pesquisa na área da programação lógica orientada a objetos.

A necessidade urgente de desenvolvimento e aplicação de instrumentos lógicos orientados a objetos foi causada pelo problema de processamento eficiente de grandes volumes heterogêneos/ arrays de dados multimídia em linguagens declarativas. A linguagem Prolog usa listas e estruturas para este propósito porque esses itens de dados têm uma correspondência um a um com os elementos da lógica de primeira ordem (constantes de Skolem, números, variáveis, etc.). Este fato não diminui o poder descritivo da linguagem Prolog, mas torna

uma ferramenta inconveniente para processamento de big data. Os meios orientados a objetos dão uma solução para este problema porque se pode encapsular os grandes arrays de dados nas instâncias de classes embutidas especializadas sem violação da semântica declarativa dos programas lógicos.

Neste artigo, as ideias básicas do uso da linguagem lógica orientada a objetos Actor Prolog para a análise de dados multicanal/ multimídia são descritas pelo exemplo de processamento de dados de vídeo 3D adquiridos usando o dispositivo Kinect 2 (Microsoft, Inc.) vídeo de imagem adquirido usando a câmera Thermal Expert V1 (i3system, Inc.). Os meios de programação lógica distribuída da linguagem Actor Prolog são discutidos pelo exemplo dos agentes lógicos comunicantes que analisam dados de vídeo 3D e implementam uma fusão desses dados com o vídeo térmico.

A fusão do vídeo de imagem térmica com dados de diferentes modalidades (incluindo os dados de vídeo 3D) é uma área de pesquisa em rápido desenvolvimento [1-4]. As principais aplicações deste conceito são a detecção, reconhecimento e reidentificação de pessoas nos sistemas inteligentes de videovigilância [5-11], aplicações médicas, incluindo detecção não invasiva de doenças perigosas no aeroporto [12-15], equipe de resgate suporte [16], interação humano-computador [17], agricultura [18, 19] e verificação automática de equipamentos e odometria [20-22]. Este artigo considera o problema de medição remota da dinâmica da temperatura de partes do corpo humano na área de videovigilância.

Na próxima seção, a arquitetura e os princípios básicos do sistema de programação lógica Actor Prolog são descritos. Na terceira seção, um conjunto de aulas desenvolvidas pelos autores para a linguagem Actor Prolog são propostos para a aquisição e análise de dados de vídeo 3D. A quarta seção dá um exemplo de um agente lógico que insere dados 3D da superfície corporal das pessoas sob vigilância por vídeo e implementa uma fusão desses dados com o vídeo de imagem térmica. Na quinta seção, os princípios básicos e meios para a comunicação da agentes lógicos na linguagem Actor Prolog são considerados.

2. A ARQUITETURA DO SISTEMA DE PROGRAMAÇÃO ATOR PROLOG LOGIC

Actor Prolog é uma linguagem de programação lógica desenvolvida no Instituto Kotel'nikov de Engenharia de Rádio e Eletrônica da Academia Russa de Ciências [23-27]. O Actor Prolog foi projetado inicialmente como uma linguagem orientada a objetos e lógica simultaneamente, ou seja, a linguagem implementa classes, instâncias e herança; ao mesmo tempo, os programas lógicos orientados a objetos têm semântica de teoria de modelo. A linguagem Actor Prolog suporta meios para definição de tipos de dados (os chamados domínios), a determinação de predicados e a direção da transferência de dados nos argumentos da sub-rotina [28, 29]. Esses meios são vitais para as aplicações industriais da programação lógica porque a experiência demonstrou que é muito difícil suportar e depurar programas lógicos grandes e complexos sem esses meios.
Actor Prolog é uma linguagem paralela; existem meios de sintaxe que suportam a criação e o controle de processos paralelos de comunicação. Esses meios de sintaxe da linguagem também fornecem a semântica da teoria do modelo dos programas lógicos, mas apenas quando certas restrições são impostas à sintaxe e estrutura dos programas [25, 26].

Uma característica distintiva do sistema de programação lógica Actor Prolog é o fato de que o

programas lógicos são traduzidos em código Java e executados pela máquina virtual Java padrão [30, 31]. Este esquema lógico de execução de programas foi desenvolvido, principalmente, para garantir a estabilidade dos programas e prevenir possíveis problemas com o gerenciamento de memória. Outra característica importante deste esquema é o fato de que ele garante alta extensibilidade da linguagem lógica: pode-se adicionar facilmente as classes internas necessárias à linguagem.

Pode-se adicionar uma nova classe interna ao Linguagem Actor Prolog da seguinte maneira. Durante a tradução do programa lógico, ele é convertido no conjunto de classes Java. Existem meios de sintaxe especiais para declarar algumas classes Java geradas automaticamente como descendentes de classes Java externas que foram criadas manualmente pelo programador. Assim, basta implementar uma nova classe embutida em Java e vinculá-la ao programa lógico no decorrer da tradução para tornar essa classe a classe interna do Actor Prolog. Atualmente, um conjunto de classes internas do Actor Prolog são implementados totalmente em Java puro; outras classes incorporadas são interfaces Java com bibliotecas de código aberto implementadas em C++. Os exemplos das classes anteriores são: a classe *Database* que implementa um sistema simples de gerenciamento de dados; a classe *File* que suporta leitura e gravação de arquivos; a classe *WebResource* que implementa a aquisição de dados da Web. Os exemplos dessas últimas classes são: a classe *FFmpeg* que vincula o Actor Prolog com a biblioteca de código aberto FFmpeg para leitura e gravação de vídeo; a classe *Java3D* que implementa gráficos 3D; a classe *Webcam* que suporta aquisição de dados de vídeo. Os autores consideram a tradução para Java como uma solução arquitetural que ajuda a desenvolver e depurar rapidamente novas classes incorporadas. A aceleração do ciclo de vida do software é causada pelo fato que a linguagem Java (em comparação com a linguagem C++) evita o aparecimento de bugs associados ao acesso incorreto à memória e ao acesso ao array fora do intervalo que pode ser muito difícil de detectar.

3. AULAS INTEGRADAS DE APOIO À VIGILÂNCIA VISUAL INTELIGENTE

Um conjunto de classes integradas para vídeo 2D e 3D aquisição e análise é implementado no sistema de programação lógica Actor Prolog. Essas classes integradas foram desenvolvidas principalmente durante a experimentação dos métodos de vigilância por vídeo inteligente.

Em primeiro lugar, a classe integrada *ImageSubtractor* foi desenvolvida no Actor Prolog que implementa um conjunto de operações padrão de processamento de imagem de baixo nível: subtração de fundo, filtragem gaussiana e de classificação, extração de blobs, estimativa de tamanhos e velocidades dos blobs, computação de recursos estatísticos adicionais que descrevem a uniformidade dos movimentos, etc. Todas as matrizes de dados utilizadas em várias etapas do processamento da imagem são encapsuladas nas instâncias da classe ImageSubtractor, que garante alto desempenho de processamento de vídeo. As regras lógicas escritas no Actor Prolog manipulam apenas os resultados do processamento de vídeo de baixo nível, ou seja, gráficos que descrevem as trajetórias, velocidades e outras características dos blobs. Pode-se ver uma descrição detalhada da classe *interna ImageSubtractor* e exemplos de sua aplicação para o processamento de imagens de baixo e alto nível em [30, 32-37].

A próxima etapa do desenvolvimento dos meios para a análise de vídeo de baixo nível no Actor Prolog foi a criação da máquina virtual de processamento de vídeo [38, 39] que foi implementada no *VideoProcessingMachine*

classe embutida. Esta máquina virtual foi desenvolvido no decurso da experimentação com a análise dos vídeos biomédicos, nomeadamente, a análise do comportamento dos roedores de laboratório. A característica distintiva desses vídeos é a presença de blobs de vários tipos na imagem; blobs diferentes requerem algoritmos diferentes de processamento de imagem de baixo nível [38, 40]. O programador pode definir uma sequência de operações de processamento de vídeo de baixo nível e fazer o download para a instância da máquina virtual; esta sequência será repetida automaticamente para cada quadro de entrada do vídeo. As operações típicas são: processar a imagem em formato de pixel; seleção e processamento de pixels de primeiro plano; extração e rastreamento de blobs no vídeo; etc. Todas as matrizes de dados intermediárias são encapsuladas dentro da instância da classe *VideoProcessingMachine* da mesma forma que na classe *ImageSubtractor* .

Neste artigo, são descritos novos meios da linguagem Actor Prolog que foram desenvolvidos para aquisição e análise de dados 3D usando o dispositivo Kinect 2. Esses meios e classes integradas especializadas do Actor Prolog foram criados para a experimentação da vigilância visual inteligente 3D [41, 42].

Os meios desenvolvidos baseiam-se nas mesmas ideias que os meios para a análise de vídeo 2D:

1. Os estágios de processamento de vídeo de baixo e alto nível são separados.
2. O estágio de processamento de vídeo de baixo nível é implementado em classes especiais embutidas que encapsulam todas as matrizes de dados intermediárias.
3. O estágio de processamento de vídeo de alto nível é implementado pelas regras lógicas; os dados são processados na forma de gráficos, listas e outros termos da linguagem lógica.

O esquema de processamento de dados foi adaptado às seguintes propriedades de dados de vídeo 3D:

1. Os dados são heterogêneos (multimodais). Por exemplo, o dispositivo Kinect 2 fornece vários fluxos de dados simultaneamente. São os quadros que descrevem nuvens de pontos 3D, os quadros de imagem infravermelha, os quadros convencionais coloridos (RGB), os quadros que descrevem as coordenadas de esqueletos humanos, etc.
2. 2O tamanho dos dados de vídeo 3D geralmente é enorme. Um computador pessoal típico não é poderoso o suficiente para armazenar em tempo real todos os dados brutos do Kinect 2 no disco rígido. Assim, um esquema preferível de processamento de dados 3D inclui análise preliminar em tempo real dos dados e armazenamento/ rede os resultados intermediários da análise.

Existem duas classes integradas no Actor Prolog que suportam a aquisição e análise de vídeo 3D: *Kinect e KinectBuffer.* A primeira classe implementa a comunicação entre o programa lógico e o dispositivo Kinect 2.

A última classe implementa a análise de baixo nível, bem como a leitura e gravação de arquivos de dados 3D. As definições dessas classes, incluindo as definições de tipos de dados (domínios) e predicados, são colocadas no arquivo "Morozov/ Kinect" do Actor Prolog.

A classe *KinectBuffer* é o elemento mais importante no esquema de processamento de dados 3D.

A instância da classe *KinectBuffer* pode ser utilizada nos seguintes modos: aquisição de dados do Kinect 2; lendo dados do arquivo; reproduzir arquivo de vídeo 3D; gravando dados no arquivo.

Os modos de reprodução de arquivo de vídeo 3D e leitura do arquivo diferem do anterior mode a classe *KinectBuffer* lê os dados e os transfere para o programa lógico em tempo real; no modo de leitura do arquivo, o programador deve controlar a leitura de cada quadro do vídeo.

O atributo operating_mode da *classe KinectBuffer* deve ser usado para selecionar o modo de operação da instância da classe: *'LISTENING', 'READING', 'PLAYING'* ou *'RECORDING'* correspondentemente. A classe *KinectBuffer* pode ser usada sozinha para ler/escrever vídeos 3D; no entanto, é necessário usá-lo em conexão com a classe *Kinect* para adquirir dados do dispositivo Kinect 2. Neste modo, deve-se criar uma instância da classe *Kinect* e transmiti-la ao construtor de a instância da classe *KinectBuffer ;* o atributo *input_device* deve ser usado como argumento do construtor. Um exemplo do programa lógico que lê e processa dados de vídeo 3D do arquivo é considerado na próxima seção.

4. AQUISIÇÃO E FUSÃO DE IMAGEM 3D E TÉRMICA DADOS DE VÍDEO

j) da matriz lógica contém as coordenadas *xey* de um programa que lê *e* implementa uma imagem análise de dados de vídeo 3D que foram adquiridos *T é* a imagem térmica. Assim, a imagem *T* 2. a serem projetados em superfícies 3D investigadas Fragmentos de código-fonte escritos em pesquisa 3D Prolog será demonstrada abaixo com também uma matriz, mas a célula desta matriz coeficientes polinomiais quadráticos reduzidos, pois o objetivo do exemplo que é necessário para dos dados as coordenadas na imagem *T ,* mas não o processamento usando o Kinect *e KinectBuffer* classes.

definir o 3DVideoSupplier e *q3 .* *KinectBuffer T* são calculadas usando a classe quadrática. O atributo operating_mode tem o *PLAYING* valor que indica que os dados da distância *d(i,j)* em metros entre o devem ser lidos do superfície do objeto para o modo de reprodução de arquivo de vídeo 3D.

```
classe '3DVideoSupplier' (especializado
 'KinectBuffer'):
nome                    = "Meu3DVídeo";
modo operacional        = 'JOGANDO';
```

No decorrer da criação do *3DVideoSupplier ,* ele baixa uma tabela de pesquisa do arquivo "MyLookUpTable. txt". Essa tabela de consulta estabelece a correspondência entre as coordenadas 3D medidas pela câmera ToF e as coordenadas 2D no vídeo de imagem térmica 2D. Depois disso, a leitura do arquivo é ativada usando o predicado *inicial* da classe *KinectBuffer .*

```
meta:-!,
    set_lookup_table("MyLookUpTable.txt"),
    começar.
```

A classe *KinectBuffer* suporta tabelas de pesquisa 2D e 3D. A tabela de pesquisa deve ser calculada e armazenada no arquivo de texto antecipadamente durante a calibração do sistema de aquisição de vídeo. A tabela de pesquisa 2D é uma matriz K do mesmo tamanho que o K quadro de vídeo infravermelho. Cada célula *(i,* simples T, no exemplo em consideração, a está usando a câmera ToF do dispositivo Kinect Actor usando a câmera ToF. A tabela de comenta; naturalmente, o código fonte contém o cálculo de é apenas demonstrar o esquema *(x,* y) se coordenam. Cada célula *(ou seja,*

j) da matriz contém seis coeficientes *p1 ,* Vamos *p2 , p3 , q1 , q2 ,* que é descendente da imagem polinômio dependendo do valor inverso arquivo "My3DVideo" na câmera ToF e a As coordenadas na classe ser investigado:

$$S = \begin{matrix} x = p1\,(1/d)2 + p2\,(1/d) + p3\,, \\ q1\,(1/d)2 + q2\,(1/d) + q3\,. \end{matrix}$$

No exemplo em consideração, a imagem térmica é projetada na superfície 3D

durante o processamento de cada quadro de vídeo 3D. O predicado *frame_obtained* é invocado automaticamente na instância da classe *KinectBuffer* quando um novo quadro de vídeo 3D é lido do arquivo. O programador utiliza o predicado *commit* para informar à classe *KinectBuffer* que ele irá processar este quadro. Depois disso, todos os predicados da classe *KinectBuffer* operam com o conteúdo desse quadro específico até que o predicado de *confirmação* seja chamado novamente. O programa lógico obtém o tempo *Time1* do quadro em milissegundos usando o *predicado get_recent_frame_time.* Em seguida, o número de quadros de imagem térmica correspondente é calculado usando essas informações. O atributo *texture_time_shift* contém um valor do deslocamento temporal entre os vídeos 3D e térmicos. O *texture_frame_rate* O atributo contém a taxa de quadros do vídeo de imagem térmica.

```
frame_obtido:-
        comprometer-se,!,
        get_recent_frame_time(Time1),
        Time2== Time1 - texture_time_shift,
        FileNumber== texture_frame_rate * Time2 / 1000,
```

Suponha que o vídeo de imagem térmica seja convertido em quadros separados. O predicado *frame_obtained* calcula o nome do arquivo JPEG correspondente usando o número do quadro. Então ele usa a carga predicado para ler o quadro e o armazena na instância de *imagem* da classe *interna BufferedImage .* O predicado get_recent_scene da *classe KinectBuffer* é chamado; este predicado cria uma superfície 3D na base do ToF dados da câmera e projetos deram textura a esta superfície. A textura é transferida para o predicado pelo segundo argumento *image* que contém uma instância da classe *BufferedImage .* A tabela de pesquisa carregada acima é usada para a implementação da projeção de textura.

A superfície 3D criada é retornada de o predicado *get_recent_scene* por meio do primeiro argumento. Esse argumento deve conter uma instância da classe *interna BufferedScene .* A classe *interna BufferedScene* implementa o armazenamento e a transferência dos dados 3D. Em particular, o conteúdo da instância da classe *BufferedScene* pode ser inserido na cena 3D exibida por meio da biblioteca de gráficos Java3D. No exemplo considerado, o predicado *set_node* é usado para esta finalidade. Ele substitui o nó "MyLabel" da imagem 3D criada usando a instância *graphics_window* da classe integrada *Java3D .*

```
ImageToBeLoaded == text?format(
    "%08d.jpeg",?round(FileNumber)),
imagem? load(ImageToBeLoaded),
get_recent_scene(buffer3D,imagem),
graphic_window ? set_node(
        "MeuRótulo",
        'Grupo de Filiais'({
                rótulo: "MeuRótulo",
                allowDetach: 'sim',
                compilar: 'sim',
                ramos: [buffer3D]
        })),
```

O predicado *get_recent_mapping* da classe *interna KinectBuffer* opera aproximadamente da mesma maneira que o *get_* predicado *recent_scene .* A diferença é a seguinte. Não cria uma superfície 3D e os resultados da projeção da textura para a superfície 3D são retornados em um formulário de uma imagem 2D conveniente. Neste exemplo, o predicado *get_recent_mapping* armazena a imagem criada na instância *buffer2D* da classe interna *BufferedImage .* Os *get_skele* O predicado da classe *KinectBuffer* retorna uma lista dos esqueletos detectados no quadro atual. Os esqueletos são gráficos que contêm informações sobre as coordenadas do corpo humano, cabeça, braços e pernas. No programa lógico, os gráficos são descritos usando o padrão simples e composto

termos: listas, estruturas, conjuntos subdeterminados, símbolos e números [41, 42]. No exemplo, os esqueletos e as imagens térmicas são transferidos para outro agente lógico que implementa uma análise e fusão adicionais dos dados. A rotina de transferência de dados

entre os agentes lógicos serão considerados na próxima seção. Observe que a imagem a ser transferida da instância *buffer2D* da classe *BufferedImage* é convertida para o termo do tipo *BINARY* . O *get_binary* predicado da classe *BufferedImage* é usado para esta finalidade.

```
get_recent_mapping(buffer2D,imagem),
get_skeletons(Esqueletos),
comunicador? notify_all_consumers(
        Esqueletos,buffer2D?get_binary()).
```

Os resultados da fusão de dados 3D e de imagem térmica implementados pelo agente lógico em consideração são demonstrados na **Figura 1.**

Na próxima seção, um esquema de comunicação entre os programas lógicos (agentes lógicos) com base nas classes internas *Database* e *DataStore* e o mecanismo

das chamadas de predicado remotas são discutidas.

5. INICIALIZAÇÃO DO LINK E COMUNICAÇÃO ENTRE OS AGENTES LÓGICOS

O mecanismo de chamadas remotas de predicado é um recurso especial da linguagem Actor Prolog que foi desenvolvida para suportar programação lógica distribuída/descentralizada. A ideia desse mecanismo é que qualquer objeto do programa lógico (agente lógico) pode ser transferido para outro agente lógico; depois disso, o novo proprietário do objeto pode invocar remotamente e de forma assíncrona os predicados do objeto [28, 29].

Observe que em termos de Actor Prolog, o objeto é sinônimo de mundo e o

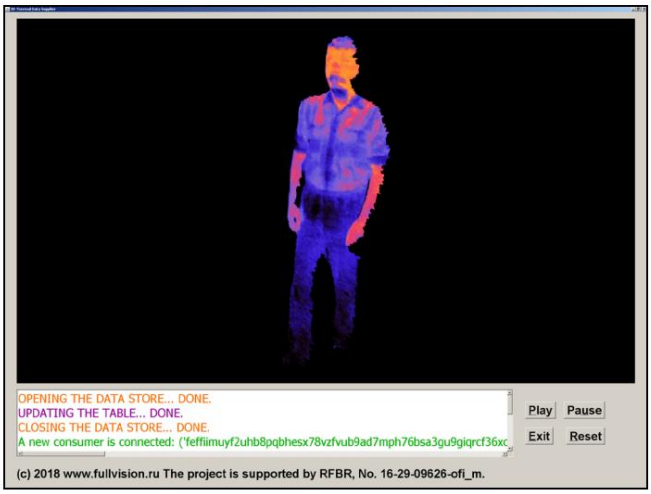

Fig. 1. *Um exemplo de um agente lógico que coleta e transfere dados multicanais heterogêneos (dados de vídeo 3D e dados de imagem térmica).*

instância de uma classe. A complicação do desenvolvimento desse mecanismo foi que o Actor Prolog possui um sistema de tipos forte e, portanto, o sistema de tipos da linguagem não fornece a possibilidade de os agentes se conectarem e se comunicarem dinamicamente sem uma troca preliminar das informações sobre os tipos dos dados a serem transferidos. Normalmente, as linguagens com sistemas de tipos fortes exigem uma troca de definições de tipos de dados na etapa de compilação dos agentes, mas na linguagem Actor Prolog, outra solução foi elaborada. Na versão distribuída do Actor Prolog, foi desenvolvido um sistema de tipos combinados, ou seja, o sistema de tipos fortes foi parcialmente suavizado no caso em que a troca de dados entre agentes é realizada.

Para ser mais preciso, os tipos de argumentos nas chamadas de predicados remotos são comparados por estrutura, mas não por nomes; além disso, a verificação de que o objeto externo implementa um predicado necessário é adiada até que essa chamada de predicado seja realmente executada. Em todos os outros casos, a verificação de tipo estático padrão é implementada na linguagem. O sistema de tipo combinado

une as vantagens do sistema de tipo forte que é usado para geração de código executável rápido e confiável e a possibilidade de verificação de tipo dinâmico durante a troca de dados entre agentes.

Uma instância de uma classe do agente lógico deve ser transferida para outros agentes de alguma forma para estabelecer uma conexão entre os agentes. A instância da classe pode ser transferida por meio de um arquivo do sistema operacional, um banco de dados compartilhado ou apenas em forma de texto por e-mail. No exemplo em consideração, o sistema de gerenciamento de banco de dados integrado do Actor Prolog é usado para essa finalidade. Este sistema é implementado pelo Banco de *Dados* e *DataStore* classes embutidas da linguagem.

Vamos definir a classe *Main* que irá implementar dois papéis no programa lógico: a execução do programa lógico começa com a criação de uma instância da classe *Main* de acordo com a definição da linguagem; a instância da classe deve ser transferida para outro agente para estabelecer um link e para a comunicação no

exemplo em consideração.

A classe *Main* contém vários slots (ou seja, as variáveis de instância da classe). O slot do *armazenamento* de dados contém uma instância do Classe *interna DataStore .* O slot do *banco* de dados contém uma instância do *3DDataSources* classe que é descendente do banco de *dados* classe embutida. O slot *video_supplier* contém uma instância da classe *3DVideoSupplier* que foi considerada na seção anterior. O slot de *consumidores* contém uma instância da classe *ConsumersList* que é descendente da classe interna *Database ;* o banco de dados de *consumidores* mantém uma lista de agentes lógicos externos que solicitaram informações do agente em consideração.

```
classe 'Principal' (especializada 'Alfa'):
    datastore= ('DataStore',
                name="AgenteBlackboard.db",
                sharing_mode='shared_access',
                access_mode='modificando');
    banco de dados= ('3DDataSources',
                lugar= compartilhado(
                        banco de dados,
                        "3DDataSources"));
    video_supplier = ('3DVideoSupplier',
                comunicador=eu);
    consumidores = ('Lista de Consumidores');
```

A classe interna *Database* implementa um sistema de gerenciamento de banco de dados simples que fornece armazenamento e pesquisa de dados de estrutura arbitrária; as operações desse tipo são típicas das linguagens lógicas do tipo Prolog. Pode-se dizer que os bancos de dados relacionais convenientes são um caso especial dos bancos de dados Prolog quando o banco de dados é utilizado apenas para armazenar os dados do tipo estrutura, ou seja, os registros com nome e lista de argumentos. Observe que a classe *Database* foi projetada para armazenar dados na memória principal do computador, ou seja, para o gerenciamento dos dados temporários. Os dados temporários podem ser armazenados no arquivo ou carregados do arquivo, se necessário.

Um mecanismo de gerenciamento de banco de dados de nível mais alto deve ser usado para controlar os dados que são compartilhados entre vários programas lógicos. Esse mecanismo é implementado na classe *interna DataStore .* A classe *DataStore* pode coordenar e controlar a operação de uma ou várias instâncias da classe *Database .* Por exemplo, pode-se ler ou gravar no arquivo o conteúdo de várias instâncias da classe *Database* de uma só vez. Outro mecanismo útil da classe *DataStore* é o suporte ao acesso compartilhado aos dados, ou seja, pode vincular as instâncias da classe *Database* com os arquivos do sistema operacional e transferir automaticamente as atualizações dos dados na memória de um

programa lógico para a memória de outros programas lógicos. A integridade dos dados é garantida pelo mecanismo padrão de transações. Esses instrumentos da classe *DataStore* são usados no exemplo em consideração.

No código acima, o construtor da instância da classe *DataStore* aceita os seguintes argumentos de entrada: o atributo name que contém o nome do arquivo "AgentBlackboard.db" que deve ser usado para o armazenamento de dados compartilhados; o atributo *sharing_mode* que atribui o modo compartilhado de acesso aos dados (o modo *shared_access);* o atributo *access_mode* que indica que o programa lógico exige os privilégios para modificação de dados compartilhados (o modo de *modificação).*

O construtor do *3DDataSources* class aceita o argumento *place* que contém o *shared(datastore,"3DDataSou rces")* com dois argumentos internos. Este argumento indica que a instância do banco de dados *3DDataSources* operará sob o controle da classe *DataStore* e o conteúdo do banco de dados possui o identificador exclusivo "3DDataSources" no namespace da instância da classe *DataStore .* Usando a analogia com os bancos de dados relacionais, "3DDataSources" é o nome de uma tabela relacional generalizada no banco de dados compartilhado "AgentBlackboard.db".

No decorrer da criação da instância da classe *Main ,* uma sequência de operações nos dados compartilhados será executada. O predicado *aberto* inicia o acesso ao arquivo de dados compartilhado "AgentBlackboard.db". Depois disso, uma transação será aberta com os privilégios de modificação de dados. Todos os registros no banco de dados "3DDataSources" serão excluídos e a instância da classe *Main* se armazenará no banco de dados. Em seguida, a transação será concluída pelo predicado *end_transaction* e o acesso ao banco de dados "AgentBlackboard.db" será encerrado pelo predicado close.

```
meta:-
    banco de dados ? abrir,
    base de dados ? begin_transaction('modificando'),!,
    base de dados ? retrair_tudo(),
    base de dados ? inserir (auto),
    base de dados ? end_transaction,
    banco de dados ? Fechar.
meta:-!.
```

A instância da classe *Main* fica disponível para outros agentes lógicos após o armazenamento no banco de dados sombreado. Em particular, um agente externo pode ler esta instância do banco de dados e invocar o comando *register_ consumidor* predicado no mundo para receber as informações sobre a temperatura das pessoas no âmbito do sistema de vigilância visual. O agente externo tem que se enviar como argumento do *register_consumer* predicado. O predicado *register_consumer* do ex-agente irá armazená-lo para os *consumidores* banco de dados interno.

```
register_consumer(ExternalAgent):-
        consumidores? insert(ExternalAgent).
```

Durante cada chamada do *frame_obtained* predicado da classe *3DVideoSupplier ,* o predicado *notify_all_consumers* é chamado. Este predicado utiliza a busca com retrocesso para extrair um a um os receptores do banco de dados de *consumidores* e enviar a eles o conjunto de dados.

```
notify_all_consumers(Skeletons,Image):-
    consumidores? find(ExternalAgent),
      notificar_consumidor(
        Agente Externo, Esqueletos, Imagem),
      falhou.
notify_all_consumers(_,_).
```

O predicado *notify_consumer* implementa a chamada remota do predicado *new_frame* no mundo externo *ExternalAgent .*

```
notify_consumer(ExternalAgent,Skeletons,Image):-
    [ExternalAgent] [<<] new_frame(Skeletons,Image).
```

A notação de sintaxe desta cláusula tem a seguinte semântica. O símbolo << indica que a chamada mensagem direta informativa deve ser usada para a transferência de dados. A mensagem direta informativa é um tipo de mensagem assíncrona suportada pela linguagem Actor Prolog [25, 26]. Os colchetes na expressão [<<] indicam que esta mensagem não deve ser armazenada em buffer, ou seja, a mensagem deve ser descartada se o receptor não a tiver processado antes do recebimento da próxima mensagem. Os colchetes na expressão *[ExternalAgent]* indicam que a chamada remota do *new_frame* predicado deve ser executado imediatamente durante a execução do comando em consideração; caso contrário, a chamada remota será suspensa e posteriormente cancelada durante o retrocesso do predicado *notify_all_consumers .*

O agente lógico receptor aceitará e processará a *mensagem new_frame(Skeletons,Image) .* No exemplo, o agente receptor deve analisar o gráfico de Esqueletos descrevendo os esqueletos das pessoas observadas pelo sistema de videovigilância inteligente. Os vértices e arestas do gráfico com o status *TRACKED* (o que significa que a câmera ToF os observa diretamente) serão selecionados e comparados com o quadro de *imagem* térmica da imagem. As intensidades dos pixels na imagem térmica que correspondem às coordenadas das bordas do gráfico são calculadas em média. As intensidades dos vértices e as intensidades médias das arestas são armazenadas em uma tabela especial. No decorrer do processamento de novos quadros, será implementada uma média temporal das intensidades dos vértices e arestas armazenadas na tabela. O propósito disto processamento é a estimativa da temperatura média das partes do corpo humano durante os movimentos irrestritos das pessoas na cena do vídeo (veja a **Figura** 2).

Os princípios de análise e fusão de dados de vídeo multicanal heterogêneos por meio da programação lógica orientada a objetos foram discutidos usando o exemplo concreto. Foi considerado um conjunto de classes embutidas da linguagem Actor Prolog para o gerenciamento do banco de dados, bem como aquisição e processamento distribuído de dados de vídeo 2D e 3D. O código completo do exemplo considerado neste artigo está publicado no pacote de instalação do sistema Actor Prolog que está disponível gratuitamente no Web Site [43]. Os códigos-fonte das classes internas consideradas no artigo estão disponíveis no GitHub [44].

6. CONCLUSÃO

No projeto Actor Prolog, são desenvolvidos e implementados os meios e ferramentas para aquisição e processamento de dados de vídeo 2D e 3D, incluindo os meios para a fusão de dados 3D coletados usando uma câmera ToF e uma câmera termográfica.

Na opinião dos autores, o principal resultado do projeto é a linguagem lógica adotada para o processamento efetivo das grandes matrizes de dados heterogêneos. Isso é fornecido graças à arquitetura orientada a objetos do sistema de programação lógica Actor Prolog que fornece o encapsulamento dos arrays de big data nas instâncias de classes integradas especializadas. Os instrumentos lógicos desenvolvidos abrem novas perspectivas na área da vigilância visual inteligente, nomeadamente, permitem implementar a análise semântica das cenas de vídeo, ou seja, realizar inferência lógica com base em dados heterogéneos que descrevem o contexto das ações humanas, objetos ,

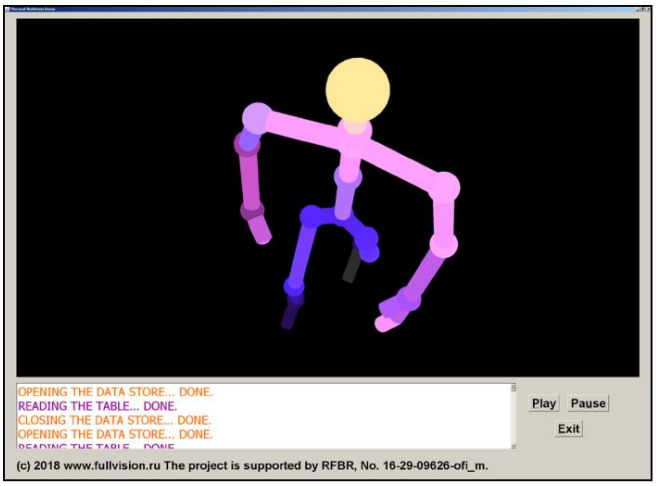

Fig. 2. *Um agente lógico que estima a temperatura média das partes do corpo humano durante os movimentos irrestritos das pessoas na cena do vídeo.*

e eventos observados pelo sistema de videovigilância inteligente.

RECONHECIMENTOS

Esta pesquisa foi apoiada pela Fundação Russa para Pesquisa Básica (concessão número 16-29-09626-ofi_m).

REFERÊNCIAS

1. Rikke G, Moeslund TB. Câmeras térmicas e aplicações: uma pesquisa. *Visão de Máquina e Aplicações,* 2014, 25:245-262.
2. Nakagawa W, Matsumoto K, de Sorbier F, Sugimoto M, Saito H, Senda S, Shibata T, Iketani A. Visualização da mudança de temperatura usando câmera RGB-D e câmera térmica. *Conferência Europeia sobre Visão Computacional-ECCV 2014 Workshops,* 2015, LNCS 8925:386-400.
3. Rangel J, Soldan S, Kroll A. Imagens térmicas 3D: fusão de termografia e câmeras de profundidade. *Proc. Estagiário Conf. em Termografia Infravermelha Quantitativa,* 2014; DOI: 10.21611/qirt.2014.035.
4. Han S, Gu X, Gu X. Um método de calibração preciso de um sistema multicâmera. Proc. Estagiário *Conf. em Modelagem e Simulação do Sistema de Vida,* Cingapura, Springer, 2017:491-501.
5. Dickens JS, van Wyk MA, Green JJ. Detecção de pedestres para veículos de minas subterrâneas usando imagens térmicas. *AFRICÃO,* 2011:1-6; DOI: 10.1109/AFRCON.2011.6072167.
6. Møgelmose A, Bahnsen C, Moeslund TB, Clapés A, Escalera S. Reidentificação de pessoa trimodal com RGB, profundidade e características térmicas. *Proc. Conferência IEEE sobre Visão Computacional e Workshops de Reconhecimento de Padrões,* 2013:301-307.
7. Susperregi L, Arruti A, Jauregi E, Sierra B, Martínez-Otzeta JM, Lazkano E, Ansuategui A. Fusão de múltiplas transformações de imagem e um sensor térmico com Kinect para melhorar a capacidade de detecção de pessoas. *Aplicações de Engenharia de Inteligência Artificial,* 2013, 26 (8): 1980-1991.
8. Kristoffersen MS, Dueholm JV, Gade R, Moeslund TB. Contagem de pedestres com manuseio de oclusão usando câmeras térmicas estéreo. *Sensores,* 2016, 16(1):62.
9. Spremolla IR, Antunes M, Aouada D, Ottersten B. RGB-D e fusão de sensores térmicos: aplicação em rastreamento de pessoas. *Estagiário Conf. em Teoria e Aplicações de Visão Computacional:* 610-617; doi:10.5220/0005717706100617.
10. Corneanu CA, Oliu M, Cohn JF, Escalera S. Pesquisa sobre RGB, 3D, térmica e abordagens multimodais para reconhecimento de expressão facial: história, tendências e aplicações relacionadas ao afeto. *Transações IEEE em análise de padrões e inteligência de máquina,* 2016, 38(8):1548-1568.
11. Berretti S, Daoudi M, Turaga P, Basu A. Representação, análise e reconhecimento de humanos 3D: uma pesquisa. *Transações ACM em Computação, Comunicações e Aplicativos Multimídia,* 2018, 14(1s):1-35.
12. Skala K, Lipic T, Sovic I, Gjenero L, Grubisic I. Sistema de imagem térmica 4D para

aplicações médicas. *Periodicum Biologorum,* 2011, 113(4):407-416.

13.Procházka A, Charvátová H, Vyšata O, Kopal J, Chambers J. Análise da respiração usando registros de vídeo de câmera de imagem térmica e de profundidade. *Sensores,* 2017, 17(6):1408.

14. Sun G, Matsui T, Kirimoto T, Yao Y, Abe S. Aplicações de termografia infravermelha para triagem em massa sem contato e não invasiva de viajantes internacionais febris em estações de quarentena de aeroportos. *Aplicação do Infravermelho às Ciências Biomédicas,* 2017:347-358

15. Chernov G, Chernov V, Flores MB. Sistema de termografia dinâmica 3D para aplicações biomédicas. *Aplicação do Infravermelho às Ciências Biomédicas,* 2017:517-545.

16. Schönauer C, Vonach E, Gerstweiler G, Kaufmann H. Reconstrução 3D de edifícios e mapeamento térmico em operações de bombeiros. *Proc. 4ª Conferência Internacional Humana Aumentada, ACM,* 2013:202-205.

17.Worch JH, Bálint-Benczédi F, Beetz M. Percepção para a interação diária do robô humano. *AI – Inteligência Artificial,* 2016, 30(1):21-27.

18.Narváez FJY, del Pedregal JS, Prieto PA, Torres-Torriti M, Cheein FAA LiDAR e fusão de imagens térmicas para caracterização 3D terrestre de árvores frutíferas. *Engenharia de Biossistemas,* 2016, 151:479-494.

19.Kawasue K, Win KD, Yoshida K, Tokunaga T. Forma do corpo do gado preto e medição de temperatura usando termografia e sensor Kinect. *Vida Artificial e Robótica,* 2017, 22(4):464-470.

20. Borrmann D, Nüchter A, ÿakuloviÿ M, Maroviÿ I, Petroviÿ I, Osmankoviÿ D, Velagiÿ J. Um sistema baseado em robô móvel para mapeamento 3D térmico totalmente automatizado. *Informática de Engenharia Avançada,* 2014, 28(4):425-440.

21.Vidas S, Moghadam P, Bosse M. Mapeamento térmico 3D de interiores de edifícios usando RGB-D e câmera térmica. *IEEE International Conference on Robotics and Automation (ICRA),* 2013:2311-2318.

22. Yamaguchi M, Truong TP, Mori S, Nozick V, Saito H, Yachida S, Sato H. Sobreposição de dados infravermelhos térmicos em estrutura 3D reconstruída por odometria visual RGB. *Transações IEICE em Informações e Sistemas,* 2018, E101D(5):1296-1307.

23. Morozov A.A. Aktorny Prolog [Prólogo do Ator]. *Programmirovanie,* 1994, 5:66-78, (na Rússia).

24.Morozov AA Actor Prolog: uma linguagem orientada a objetos com a semântica declarativa clássica. *Proc. Estagiário Workshop sobre Implementação de Linguagens Declarativas* (IDL 1999), Paris, França, 1999:39-53.

25.Morozov A.A. Ob odnom podkhode k logicheskomu programmirovaniyu intellektualnykh agentesov dlya poiska i raspoznavaniya informatsii v Internet [Em uma abordagem para a programação lógica de agentes inteligentes para pesquisar e reconhecer informações na Internet]. *Zhurnal radioeletrônica,* 2003; http://jre.cplire.ru/jre/nov03/1/text.html.

26.Morozov AA. Modelo lógico orientado a objetos de computações simultâneas assíncronas. *Reconhecimento de Padrões e Análise de Imagem,* 2003, 13(4):640-649.

27.Morozov AA. Abordagem operacional do raciocínio modificado, baseada no conceito de provas repetidas e atores lógicos. *CICLOPS,* 2007, 7:1-15.

28. Morozov AA, Sushkova OS, Polupanov AF. O probleme vvedeniya sredstv raspredelennogo mnogoagentnogo programmirovaniya v logicheskiy yazyk so stroginy tipizatsiey [Sobre o problema de introduzir as ferramentas de distribuição

programação multiagente em uma linguagem lógica com tipagem forte]. *Zhurnal radioelektroniki,* 2016, 7; http://jre.cplire.ru/jre/jul16/9/text.pdf.

29. Morozov AA, Sushkova OS, Polupanov AF. Rumo à programação lógica distribuída de aplicativos de vigilância visual inteligente. Avanços em Soft Computing: *Proc. 15º Estagiário Mexicano. Conferência sobre Inteligência Artificial, MICAI* 2016, Cancun, México, 2017:42-53.

30. Morozov AA, Polupanov AF. Programação lógica de vigilância visual inteligente: problemas de implementação. *CICLOPS-WLPE,* 2014, AIB:31-45.

31. Morozov AA, Sushkova OS, Polupanov AF. Um tradutor do Actor Prolog para Java. *RegraML,* 2015, 8.

32. Morozov AA, Vaish A, Polupanov AF, Antciperov VE, Lychkov II, Alfimtsev AN, Deviatkov VV. Desenvolvimento de sistema de programação lógica orientada a objetos concorrente para monitoramento inteligente de atividades humanas anômalas. *Proc. 7º Estagiário. conf. eletrônica e dispositivos biomédicos,* Espanha, 2014:53-62.

33. Morozov AA. Desenvolvimento de um método para monitoramento de vídeo inteligente de comportamento anormal de pessoas baseado em programação em lógica paralela orientada a objetos. *Reconhecimento de padrões e análise de imagens,* 2015, 25(3):481-492.

34. Morozov AA, Polupanov AF. Desenvolvimento da abordagem de programação lógica para o monitoramento inteligente do comportamento humano anômalo. *Proc. 9º Workshop Aberto Alemão-Russo sobre Reconhecimento de Padrões e Compreensão de Imagem (OGRW* 2014), Koblenz, Universidade de Koblenz Landau, 2015, 5:82-85.

35. Morozov AA, Sushkova OS, Polupanov AF. Uma abordagem para o inteligente monitoramento de comportamento humano anômalo baseado na linguagem lógica orientada a objetos Actor Prolog. *RuleML,* 2015, DC e Challenge, 2015, 8.

36. Morozov AA, Vaish A, Polupanov AF, Antciperov VE, Lychkov II, Alfimtsev AN, Deviatkov VV. Desenvolvimento de plataforma de programação lógica orientada a objetos concorrente para o monitoramento inteligente de atividades humanas anômalas. *Proc. Estagiário Conf. em Sistemas e Tecnologias de Engenharia Biomédica,* Heidelberg, Springer, 2015, 511:82-97.

37. Morozov AA, Sushkova OS. Análise em tempo real do vídeo por meio da linguagem Actor Prolog. *Computer Optics,* 2016, 40(6):947-957.

38. Morozov AA, Sushkova OS, Vaniya SM. Desenvolvimento de métodos e algoritmos baseados em programação lógica orientada a objetos para monitoramento de vídeo de roedores de laboratório. *13º Estagiário. Conf. sobre Tecnologia de Imagem de Sinal e Sistemas Baseados na Internet, IEEE,* 2017:311-318; DOI: 10.1109/SITIS.2017.59.

39. Morozov AA, Sushkova OS. Virtualnaya mashina nizkourovnevoy obrabotki videoizobrazheniy v Aktornom Prologe [Máquina virtual para processamento de vídeo de baixo nível no Actor Prolog]. *Sab Works IV Int. conf. e escola para jovens "Tecnologia da informação e nanotecnologia" (ITNT-2018),* Samara, Samara Univercity, 2018:1275-1285.

40. Morozov AA, Sushkova OS. Sobre o desenvolvimento de métodos e algoritmos baseados em programação lógica orientada a objetos para monitoramento de vídeo de

ratos de laboratório]. *Sab Works IV Int. conf. e escola para jovens "Tecnologia da informação e nanotecnologia» (ITNT-2018),* Samara, Samara Univercity, 2018:1182-1192.

41. Morozov AA, Sushkova OS, Polupanov AF. Programação lógica orientada a objetos de vigilância por vídeo inteligente 3D: A declaração do problema. *IEEE 26º Estagiário. Simpósio de Eletrônica Industrial (ISIE), Biblioteca Digital IEEE Xplore,* 2017:1631-1636.

42. Morozov AA, Sushkova OS, Polupanov AF. Programação lógica orientada a objetos de Sistemas 3D de Vigilância Intelectual por Vídeo: Enunciado do Problema. *radioeletrônica. Nanossistemas. Tecnologias de Informação (RENSIT),* 2017, 9(2):205-214; DOI: 10.17725/rensit.2017.09.205.

43. Morozov AA, Sushkova OS. O site de programação de lógica de vigilância visual inteligente. *Recuperado em 20 de maio de 2018, de http:// www.fullvision.ru.*

44. Morozov AA. Um repositório GitHub contendo códigos-fonte de classes internas do Actor Prolog. *Recuperado em 20 de maio de 2018, de https:// github.com/ Morozov2012/ ator-prolog-java-library.*