МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

"Национальный исследовательский ядерный университет МИФИ"

НИЯУ МИФИ

Факультет управления и экономики высоких технологий

Кафедра стратегического планирования и методологии управления

УДК 681.58

«ДОПУЩЕНА К ЗАЩИТЕ»

Зам. зав.
кафедрой _____(Королёв А.С.)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Исследование и разработка метода логического программирования интеллектуального видеонаблюдения за действиями персонала АЭС

Руководители ВКР:	Доцент, к.т.н.	_____	(Королёв А.С.)
	К.ф.-м.н.	_____	(Морозов А.А.)
Исполнитель:	студентка гр. У08-82	_____	(Тукмачёва Ю.А.)

Москва – 2016

МИНОБРНАУКИ РОССИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

"Национальный исследовательский ядерный университет МИФИ"

НИЯУ МИФИ

Факультет управления и экономики высоких технологий

Кафедра стратегического планирования и методологии управления

УТВЕРЖДАЮ	СОГЛАСОВАНО
Декан факультета «У» _____ А.В. Путилов	Зам. зав. кафедрой _____ А.С. Королёв
«____» _____201__г.	«____» _____201__г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

Студент: Тукмачёва Ю.А.
Шифр:
Направление: Системный анализ и управление
Группа: У08-82

1. Тема выпускной квалификационной работы

Исследование и разработка метода логического программирования интеллектуального видеонаблюдения за действиями персонала АЭС

2. Техническое задание на выполнение выпускной квалификационной работы

1. Исследовать предметную область и выработать перечень задач интеллектуального видеонаблюдения за действиями персонала АЭС.
2. Провести экспериментальное исследование методов низкоуровневого анализа видеоизображений в Акторном Прологе, в том числе, метода вычитания фона и метода оценки скорости движения объектов.
3. Подготовить экспериментальные видеоклипы для исследования методов низкоуровневого анализа видео в Акторном Прологе, в том числе, примеры поведения «ходьба», «бег», «человек заснул на рабочем месте», «человек покинул рабочее место», а также примеры аномального поведения сотрудников АЭС.

4. Разработать и описать пример логической программы, выявляющей выбранный тип аномального поведения персонала АЭС.

3. Планируемые результаты выполнения выпускной квалификационной работы

1. Экспериментальные видеоклипы для исследования алгоритмов интеллектуального видеонаблюдения.
2. Пример логической программы, выявляющей заданный тип аномального поведения персонала АЭС.

4. Этапы выполнения выпускной квалификационной работы

№ этапа	Содержание этапа работы	Результаты выполнения этапа	Срок выполнения
1	Исследование предметной области и выработка перечня задач интеллектуального видеонаблюдения за действиями персонала АЭС	Перечень задач интеллектуального видеонаблюдения за действиями персонала АЭС	30.06.2016
2	Подготовка экспериментальных видеоклипов для исследования методов низкоуровневого анализа видео в Акторном Прологе	Набор экспериментальных видеоклипов для исследования алгоритмов интеллектуального видеонаблюдения	03.06.2016
3	Экспериментальное исследование методов низкоуровневого анализа видеоизображений в Акторном Прологе, в том числе, метода вычитания фона и метода оценки скорости движения объектов	Результаты статистической оценки качества работы алгоритмов низкоуровневого анализа изображений Акторного Пролога	13.06.2016
4	Разработка примера логической программы, выявляющей выбранный тип аномального поведения персонала АЭС	Логическая программа, выявляющая выбранный тип аномального поведения персонала АЭС	24.06.2016

5. Перечень разрабатываемых документов и графических материалов

1. Экспериментальные видеоклипы.
2. Пример логической программы.

6. Руководители и консультанты выпускной квалификационной работы

Функциональные обязанности	**Должность в НИЯУ МИФИ**	**Ф.И.О.**	**Подпись**
Руководитель выпускной работы	Заместитель заведующего кафедрой № 82	Королёв Антон Сергеевич	
Руководитель выпускной работы	К.ф.-м.н., с.н.с. ИРЭ им. В.А. Котельникова РАН	Морозов Алексей Александрович	

7. Мониторинг выполнения выпускной квалификационной работы

Этап, №	Дата представления результатов руководителю и консультантам	Отметка руководителя и консультантов о выполнении работы
1		
2		
3		
4		

Задание выдал
Руководитель
_____

«___»_____201__г.

Задание принял к исполнению

Студент _____

«___»_____201__г.

Список сокращений

АЭС – Атомная электростанция

БД – База данных

ВКР – Выпускная квалификационная работа

ИРТ – Исследовательский реактор типовой

МФ – Медианная фильтрация

СПТ – Система промышленного телевидения

Реферат

Пояснительная записка 70 с., 4 ч., 13 рис., 5 табл., 8 источников, 3 прил.

Тема: Исследование и разработка метода логического программирования интеллектуального видеонаблюдения за действиями персонала АЭС.

Объектом исследования и разработки является метод интеллектуального видеомониторинга аномального поведения людей, базирующегося на объектно-ориентированном логическом языке Акторный Пролог.

Цель работы – повысить уровень безопасности эксплуатации АЭС за счёт автоматизации процесса контроля соблюдения правил техники безопасности сотрудниками АЭС. Процесс автоматизации осуществить с помощью метода интеллектуального видеонаблюдения за действиями персонала АЭС, используя логический язык программирования Акторный Пролог.

В процессе работы было проведено исследование предметной области и выработан перечень задач интеллектуального видеонаблюдения за действиями персонала АЭС.

Помимо изучения правил техники безопасности на АЭС, был проведён анализ существующих методов видеонаблюдения, разработанных целенаправленно для АЭС. Этот анализ позволил дать оценку развития данного направления – существующие методы находятся на начальной стадии развития, в настоящее время разработчики осуществляют лишь низкоуровневую обработку видео.

Для исследования методов низкоуровневого анализа видео, а также для реализации метода, выполняющего высокоуровневую обработку, были подготовлены экспериментальные видеоклипы:

1. Видео пробегающей девушки на открытом пространстве.
2. Видео пробегающей и идущей спокойным шагом девушки в закрытом пространстве.

3. Пультовая комната, оставленная без присмотра:

3.1. Диспетчер уснул или потерял сознание в пультовой комнате во время смены.

3.2. Диспетчер покинул пультовую комнату.

Для оценки правильности работы алгоритма, выполняющего низкоуровневую обработку, было проведено экспериментальное исследование. По итогам эксперимента были вычислены количественные показатели, характеризующие работу алгоритма, а также выявлены проблемы – условия, при которых алгоритм не справлялся с работой.

В качестве примера аномального поведения на АЭС была выбрана ситуация, в которой пультовая комната остаётся без присмотра диспетчера. Для выбранной ситуации был разработан пример логической программы на языке Акторный Пролог, выявляющей данный тип аномального поведения персонала АЭС.

После разработки алгоритма было проведено его экспериментальное тестирование. По результатам эксперимента выявлено, что среди 30 тестовых запусков, программа ни разу не дала сбоя, т.е. вовремя и верно регистрировала пультовую комнату, оставленную без присмотра.

Оглавление

Введение

В настоящее время системы интеллектуального видеонаблюдения становятся всё более популярными. Теперь распознаванием аномального поведения человека может заниматься не только диспетчер или охранник, сидящий у камеры, но и техника, запрограммированная для распознавания движения на видео, а также для последующего его анализа. С помощью такой техники можно обнаруживать драки, кражи, грабежи, разбои, теракты и многое другое, что может оказаться опасным для жизнедеятельности. Это позволит человеку, сидящему у камеры, своевременно отреагировать и принять меры по предотвращению угрозы.

Атомные электростанции являются отраслью с повышенным уровнем опасности. Соблюдение правил техники безопасности, правил эксплуатации технического оборудования, правил поведения в зонах с высоким уровнем радиации и т.д. оказывают огромное влияние на безопасную, как для персонала, так и для окружающей среды работу АЭС. Применение систем видеонаблюдения с вышеописанными функциями позволит контролировать процесс работы АЭС и тем самым увеличит безопасность её эксплуатации.

Целью настоящей работы является повысить уровень безопасности эксплуатации АЭС за счёт автоматизации процесса контроля соблюдения персоналом АЭС правил техники безопасности. Автоматизировать процесс видеонаблюдения с помощью метода интеллектуального анализа аномального поведения людей, используя логический язык программирования Акторный Пролог.

Акторный Пролог – язык программирования, который воплощает новый подход к объединению логического и объектно-ориентированного программирования и обладает следующими достоинствами:

- В основе подхода лежит использование классической логики (логики предикатов первого порядка).

- Центральной идеей и сущностью подхода является обнаружение и устранение логических противоречий, возникающих в процессе взаимодействия объектов.
- Разработанный подход позволил математически корректным образом ввести в логический язык разрушающее присваивание и параллельные процессы.

Основной идеей метода распознавания аномального поведения людей, реализованного с помощью языка Акторный Пролог, является описание искомого сценария поведения объекта и последующий его анализ с использованием логики предикатов первого порядка. Отличительными особенностями такого подхода представляются использование именно объектно-ориентированного логического языка и трансляция его в язык Java. Это позволяет разделить программу на взаимодействующие параллельные процессы, которые реализуют различные этапы обработки видеоизображения и анализа сцены. Трансляция в язык Java гарантирует необходимую надёжность, переносимость и открытость программного обеспечения интеллектуального видеонаблюдения.

В качестве параллельных взаимодействующих процессов могут быть реализованы два этапа обработки видеоизображения – низкий и высокий уровень. Низкий уровень осуществляет начальные этапы обработки – вычитание фона, распознавание движущихся объектов, отслеживание объектов, вычисление их скорости и точек взаимодействия. Высокий уровень анализирует данные, полученные на этапе низкоуровневой обработки (распознаёт аномальное поведение), и выводит результаты на экран.

Реализация метода обнаружения и анализа поведения объектов на языке Акторный Пролог является новым направлением в области систем интеллектуального видеонаблюдения. В результате изучения описанных в литературе подходов к интеллектуальному видеонаблюдению и распознаванию аномальных ситуаций на АЭС, было обнаружено, что в настоящее время существуют системы, выполняющие только начальный этап

обработки (низкий уровень), в области анализа высокого уровня идут лишь исследования и разработка программ. Этот факт подчёркивает новизну данной работы, т.к. метод, разработанный на базе логического языка Акторный Пролог, наряду с выполнением низкоуровневой обработки, будет осуществлять и высокоуровневую обработку на конкретной смоделированной ситуации.

Актуальность настоящей работы можно обосновать с помощью следующих фактов:

- Системы видеонаблюдения позволяют осуществлять непрерывный мониторинг в режиме реального времени, тем самым диспетчер может непрерывно контролировать критические области.
- С помощью такой системы можно анализировать обстановку в труднодоступных местах или в местах с повышенным уровнем радиационного излучения.
- Распознавание аномальных ситуаций и нарушений правил техники безопасности АЭС позволит обеспечить своевременную реакцию диспетчера на угрозу и принятие им мер по её предотвращению.

1 Обзор литературы

В настоящее время системы видеонаблюдения на объектах с повышенным уровнем опасности, таких как АЭС, играют немаловажную роль. С каждым годом уровень их развития повышается, т.к. с помощью них можно решить задачи, отвечающие за безопасную эксплуатацию АЭС. К таким задачам можно отнести контроль за соблюдением персоналом правил техники безопасности, правил эксплуатации оборудования, правил поведения в зонах с высоким уровнем радиации, правил по защите окружающей среды и т.д. Существует несколько систем, которые позволяют контролировать указанные выше процессы.

Система видеонаблюдения, предложенная исследователями из Индии Gaurav Kumar Singh, Vipin Shukla, Swapnil Patil и Pratik Shah используется для контроля безопасности «чувствительных» областей АЭС [**1**]. К чувствительной области АЭС относятся здание реактора, заводы по производству и обогащению топлива, хранилища переработанного топлива и т.д. Система анализирует видеофрагменты для того, чтобы выявить аномальные и нештатные ситуации. Конечной целью создания системы является автоматическое формирование сигнала тревоги для оператора, чтобы он в режиме реального времени мог предпринять какие-то действия по её предотвращению.

Неотъемлемой частью ядерной безопасности является физическая защита. Визуальный контроль ядерных установок и ядерных материалов также является частью системы видеонаблюдения. Работа направлена на предотвращение несанкционированного проникновения или доступа к ядерным установкам и материалам. Система также предотвращает радиологический саботаж ядерных объектов и хищение ядерных материалов.

Данный проект в будущем превратится в полностью автоматическую систему видеонаблюдения, которая будет выполнять низкоуровневую обработку, подразумевающую обнаружение движения, отслеживание

движущихся объектов, а также задачи высокого уровня – распознавание аномальных ситуаций, представляющих угрозу для нормальной эксплуатации АЭС.

На сегодняшний день данная система выполняет лишь низкий уровень обработки. Программная среда, в которой она реализована – MATLAB®, Simulink®. Конвертируя цветное изображение в интенсивности и используя Гауссовские смеси, алгоритм вычисляет передний план и обнаруживает блобы. Результаты работы алгоритма представлены на рисунке 1, где белая область обозначает обнаруженный блоб.

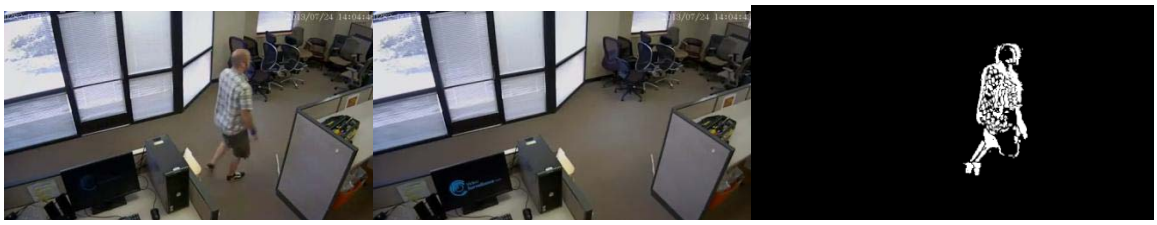

Рисунок 1. Результаты работы алгоритма вычисления переднего плана (слева на право: исходный кадр, задний план, передний план).

Низкоуровневая обработка является основополагающим шагом во всей системе, т.к. результаты её работы являются входными данными для обработки высокого уровня. Поэтому от первой стадии обработки видеоизображений будет зависеть, насколько корректно будет проходить этап распознавания аномальных событий.

Авторы системы планируют продолжать исследования, чтобы в дальнейшем разработать алгоритм, выполняющий высокоуровневую обработку изображения. Некоторые исследователи, учреждения и коммерческие организации проявляют интерес к будущей системе – это будет являться мотивацией для разработчиков.

Студенткой Национального авиационного университета Украины была описана система промышленного телевидения (СПТ) на Атомной электростанции, предназначенная для наблюдения в технологических зонах с повышенным уровнем радиации, а также контроля за техническим

состоянием оборудования для своевременного предотвращения аварийных ситуаций [**2**].

Система выполняет следующие основные функции:

- ✓ своевременное обнаружение неисправностей и нарушения режимов работы основного оборудования, находящегося в зоне контроля системы;
- ✓ анализ зарегистрированных изображений повреждений и планирования мер по их ликвидации;
- ✓ ускорение производства аварийно-восстановительных и плановых ремонтных работ;

в том числе функции, обеспечивающие безопасность персонала:

- ✓ определение возможности доступа оперативного персонала в контролируемые помещения после проектной аварии;
- ✓ уменьшение доз облучения персонала;
- ✓ видеонаблюдение в зонах с повышенным уровнем радиации.

СПТ автоматически формирует тревожный сигнал для оповещения персонала при возникновении аварийных и нештатных ситуаций. Такая система представляет собой многокамерную телевизионную систему, которая позволяет следить за событиями, происходящими одновременно в нескольких местах.

Сотрудниками Португальского Института ядерной инженерии была предложена система видеонаблюдения, которая оценивает дозу облучения, получаемую персоналом во время выполнения работы на АЭС [**3**]. Предполагается, что такая система будет проводить оценку дозы на основе построенной траектории движущегося объекта и нормы дозы облучения, принятой на реальном исследовательском ядерном реакторе Института ядерной инженерии – Argonauta. Целью системы является повышение уровня безопасности работы персонала. Подразумевается, что с помощью этой системы можно будет оптимизировать планы выполнения работ, тем самым обеспечивая снижение уровня облучения персонала.

На данном этапе реализации метода были проведены исследования только начальной стадии низкоуровневой обработки – сегментирование изображения или, по-другому, вычитание фона. Во время съёмок видеоклипа в помещении исследовательского реактора было обеспечено практически стабильное освещение (поскольку резкие перемены в освещение могут повлиять на корректную работу алгоритма вычисления переднего плана) и присутствие только уполномоченных сотрудников. Результаты работы алгоритма вычисления переднего плана можно увидеть на рисунке 2.

Рисунок 2. Результаты работы алгоритма вычисления переднего плана.

Конкретные методики отслеживания объектов авторы статьи планируют реализовать в будущем, что станет основополагающим шагом к реализации системы.

Резюмируя представленные в данном обзоре статьи, можно прийти к выводу, что системы интеллектуального видеонаблюдения, направленные на повышение уровня безопасности на АЭС, находятся на начальной стадии развития. Большинство из них в настоящее время осуществляют лишь низкоуровневую обработку видеоизображений. Однако исследование и разработка методов интеллектуального видеонаблюдения продолжаются, и в

дальнейшем, по мере их успешной реализации, они будут внедряться в областях с высокой степенью опасности.

2 Описание метода интеллектуального видеонаблюдения

В данной работе представлен метод интеллектуального видеонаблюдения за сотрудниками АЭС, реализующий обе ступени обработки (как высокий, так и низкий уровень). Распознавание аномального поведения будет рассмотрено на примере конкретных ситуаций, смоделированных на основе правил техники безопасности.

Метод интеллектуального видеонаблюдения для выявления аномального поведения людей, рассматриваемый в данной дипломной работе, основан на объектно-ориентированном логическом программировании. Главная идея метода заключается в описании искомого сценария аномального поведения людей с последующим анализом видеоизображения посредством логического программирования (используя логику предикатов первого порядка). Отличительными особенностями этого метода являются использование объектно-ориентированного логического языка Акторный Пролог и трансляция логических программ интеллектуального видеонаблюдения в язык Java. В этом случае, объектно-ориентированные средства логического языка позволяют разделить программу на взаимодействующие параллельные процессы, которые реализуют различные этапы обработки видеоизображения и анализа видеосцены, в то время как трансляция в язык Java гарантирует надёжность, переносимость и открытость программного обеспечения интеллектуального видеонаблюдения, включая возможность использования современных библиотек обработки изображений низкого уровня.

Как правило, различают низкий уровень обработки видеоизображений (вычитание фона, распознавание людей и машин, выстраивание траекторий движения объектов, распознавание позы и некоторых частей человеческого тела, оценка скорости, обнаружение резких движений и т.п.) и высокий

уровень обработки видеоизображений (драка, разбойное нападение, кража, оставленные без присмотра предметы и другое). Высокоуровневая обработка реализуется с помощью использования логических средств; при этом входные данные получаются в результате низкоуровневого распознавания, реализованного с использованием стандартных методов анализа видеоизображений и языков программирования более низкого уровня.

В Акторном Прологе текст логической программы состоит из отдельных классов, а параллельные процессы представляют собой экземпляры классов. Например, программа может создать два параллельных процесса. Первый процесс считывает последовательность изображений JPEG, которые имитируют ввод видео в режиме реального времени, и осуществляет анализ низкого уровня, а именно, вычитает фон, распознаёт объекты, отслеживает объекты и определяет точки их взаимодействия. Второй процесс анализирует информацию, подготовленную в первом процессе, и отображает результаты видеонаблюдения на экране. Последние две операции также могут быть реализованы в форме отдельных процессов.

2.1 Низкий уровень обработки видеоизображения

В настоящее время в Акторном Прологе низкий уровень обработки видеоизображения делится на 5 стадий [**4**]:

1. Первая стадия видеообработки заключается в вычитании заднего плана (фона). Программа получает последовательность видеокадров и вычисляет математическое ожидание и разность между кадром и вычисленным математическим ожиданием для каждого отдельного пикселя на изображении. Пиксели, яркость которых отличаются от математического ожидания на значение большее, чем заданный порог (порог задаётся в виде количества среднеквадратических отклонений), рассматриваются как элементы переднего плана. Дисперсия и математическое ожидание значений яркости пикселей может постоянно уточняться во время выполнения программы или на основе заданного количества первых

кадров. Перед вычислением математического ожидания и разницы значений пикселей изображение может быть обработано двумерным гауссовым фильтром для уменьшения цифрового шума, а также переведено в формат «оттенки серого».

2. Вторая стадия низкоуровневой обработки видеоизображений – это распознавание блобов (объектов переднего плана). Для детектирования блобов был придуман простой алгоритм, создающий прямоугольники вокруг объектов переднего плана. Идея алгоритма состоит в том, что пиксели на переднем плане заключаются в прямоугольные рамки; пересекающиеся (соприкасающиеся) рамки, в свою очередь, также заключаются в рамки, процесс продолжается, пока все пиксели объектов на переднем плане не будут обведены в непересекающиеся рамки. Вычисленные таким образом прямоугольники и являются блобами.
3. Третья стадия обработки – это создание треков – траекторий блобов. Предполагается, что последовательные (соседние) кадры видеоизображений содержат те же самые блобы, если прямоугольники блобов пересекаются в смежных кадрах. Если есть несколько пересекающихся блобов, то блоб с наибольшей площадью пересечения берётся в качестве продолжения трека, остальные блобы рассматриваются как продолжения других треков, пересекающих данный трек. Трек, для которого не найдены соответствующие блобы в следующем кадре, будет ещё существовать некоторое определённое количество времени. Если продолжение трека не будет найдено в течение этого периода, то трек будет считаться законченным. Законченные треки сохранятся ещё некоторое время и затем стираются. Кроме того, треки, длина которых меньше определённого значения, рассматриваются как ложные и забраковываются сразу же. Одновременно с построением треков сохраняются точки их пересечений. Это необходимо для построения графа связей между блобами, а также для точной оценки скорости

отдельных блобов; чтобы вычислить скорость блобов, выбираются области треков, в которых нет пересечений с другими треками.

4. Четвёртая стадия обработки – это определение скорости блобов. Для вычисления скорости был придуман алгоритм, который вычисляет нижнюю границу скорости блоба [**5**]. Для этой цели на основе обратной матрицы проективных преобразований вычисляются физические координаты четырёх углов прямоугольного блоба. Числовое дифференцирование координат даёт грубую оценку скорости в каждом углу блоба на основе предположения, что точка рассматривается на уровне пола. Предполагается, что это так, по крайней мере, для одного угла блоба, и что для всех других углов блоба оценки скорости могут быть только завышены, так как камера находится выше уровня пола, и верхняя часть человека визуально соответствует более далёким точкам пола. Основываясь на этих предположениях, берётся наименьшее из четырёх вычисленных значений скорости углов блоба. Для увеличения стабильности оценок скорости на различных стадиях вычислений осуществляется медианная фильтрация, а именно, медианной фильтрации подвергается последовательность значений координат углов блоба (отдельно по осям X и Y), затем последовательность значений оценок скорости блоба отдельно по осям X и Y (вычисленных как минимумы по четырём абсолютным значениям производных от координат четырёх углов блоба), а также последовательность значений скорости блоба, вычисленных как квадратный корень из суммы квадратов скоростей блоба по осям X и Y [**6**].
5. Пятая стадия низкоуровневой обработки – это построение связных графов движения блобов. Каждый граф включает все треки, которые имеют точки пересечения в какие-либо моменты времени. При необходимости алгоритм позволяет исключать треки неподвижных и медленно движущихся объектов.

В Акторном Прологе все перечисленные выше стадии низкоуровневого анализа осуществляет предопределённый класс «*ImageSubtractor*».

2.2 Эксперименты с обработкой видеоизображений

Для изучения алгоритмов обработки видеоизображений низкого уровня было снято несколько видеоклипов. Видеоклипы снимались в разных условиях: на улице, в помещении. При дальнейшем анализе этих видеоклипов в среде программирования Акторный Пролог было выявлено несколько проблем, из-за которых используемый метод давал сбои.

Видео снималось на камеру модели SONY HANDYCAM HDR-XR 350 7.1 MPx, с частотой кадров равной 25, разрешение видео составляло 1280x720 px. Первоначально эксперименты проводились на компьютере модели ASUS с процессором Intel Core i3-2350M CPU 2.3 GHz. В дальнейшем мы были вынуждены перейти на более мощный компьютер модели iMac с процессором Intel Xeon CPU E5645 @2.40GHz 2.53GHz (2 процессора), 64-разрядная ОС, установленная память 18.0 ГБ. На обоих компьютерах установлена операционная система Windows 10. Рассмотрим полученные результаты.

2.2.1 Анализ изображения перемещающегося на открытом пространстве человека

Первая версия видеоклипа была снята на открытом пространстве (на улице) в солнечную погоду. На заднем плане видео находилось одноэтажное кирпичное здание. На видео присутствует пробегающая девушка.

Для того чтобы проанализировать видеоклип в логической программе, видео следует представить в виде последовательности кадров. Конвертирование видеоклипа осуществлялось с помощью программы Free Video to GPG Converter, из видео формата MP4 были получены кадры формата JPEG. Логическая программа, реализующая метод, загружает

последовательность кадров и воспроизводит видео в режиме реального времени.

При дальнейшем анализе видеоизображений было обнаружено, что из-за солнечной погоды тени деревьев на здании находились в движении. Также, поскольку видеосъёмка проводилась на улице, камера со штативом была подвержена влиянию ветра, что мы и обнаружили в результате анализа – камера в некоторый момент времени дёрнулась. В связи с выявленными проблемами, стало необходимым проверить качество работы алгоритма вычитания фона (по-другому – вычисления переднего плана, соответственно, изображения бегущего человека).

При анализе данного видеоклипа с помощью программы, реализованной на логическом языке программирования Акторный Пролог, было проведено несколько экспериментов по проверке метода вычитания фона. Рассмотрим суть этих экспериментов и их результаты.

1. Для начального изучения был использован предикат *get_background_image*, который возвращает результаты усреднения кадров. В результате на экран выводится неподвижная (чёрно-белая или цветная) картинка (см. рисунке 3). В связи с тем, что в некоторый момент времени камера дёрнулась, алгоритм вычитания фона стал работать неправильно, поэтому картинка получилась смазанной, как видно на рисунке 3.

Рисунок 3. Результаты усреднения кадров (изображение получено с помощью предиката get_background_image).

2. Для рассмотрения переднего плана был использован предикат *get_foreground_image*. Согласно описанию предиката, статические

объекты он должен опознавать как задний план, т.е., если изображение полностью неподвижно, мы должны получить картинку однотонного цвета (в нашем случае, серого). Однако на рисунке 4 видно, что некоторые элементы здания и тени деревьев предикат распознаёт в качестве переднего плана, что является ошибкой.

Рисунок 4. Изображение переднего плана (получено с помощью предиката get_foreground_image).

3. Ради эксперимента был использован предикат *get_blobs*, который возвращает найденные программой блобы. Результаты работы алгоритма выявления блобов приведены на рисунке 5. Алгоритм выделил на видеоизображении большое количество ложных блобов.

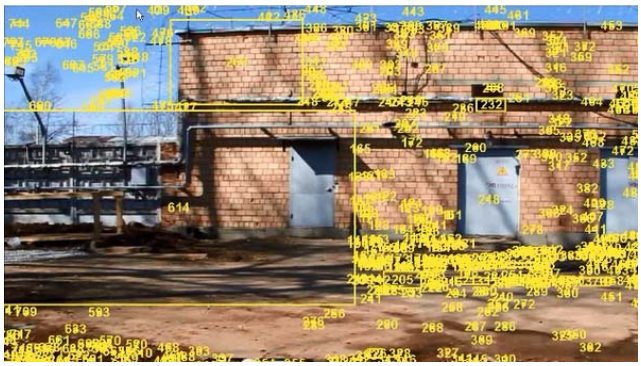

Рисунок 5. Результаты распознавания блобов.

Анализируя всё вышеизложенное, можно прийти к выводу, что на данном этапе реализации метода выявления аномального поведения людей пока не предусмотрена работа с движущимися тенями (алгоритм перестаёт правильно определять блобы, их получается большое количество). Также можно заметить, что видеокамера обязательно должна быть статична, иначе алгоритм вычитания фона выполняет работу некачественно. Программный код, реализующий алгоритм вычитания фона и обнаружения блобов,

прилагается к настоящей работе на электронном носите (CD-ROM), пример под названием «Blob».

2.2.2 Анализ изображения перемещающегося в закрытом пространстве человека

Затем была снята вторая серия видеоклипов уже в закрытом пространстве, однако эта серия не пошла в работу по причине того, что движущиеся объекты находились слишком близко к камере.

Во избежание проблем с обработкой, выявленных при анализе первой и второй серий видеоклипов, была снята третья серия видеоклипов, в которой в роли закрытого пространства выступал спортзал. Камера была установлена на балконе, который исполнял роль второго этажа.

Было снято два видеоклипа: девушка, проходящая спокойным шагом, и девушка, пробегающая с высокой скоростью.

Для данных видеоклипов также проверялся алгоритм вычитания фона, к тому же была изучена ещё одна стадия обработки низкого уровня – вычисление скорости движущегося объекта.

Для того чтобы вычислить скорость, требуется вычислить матрицу проективных преобразований. Матрица проективных преобразований позволяет преобразовать 3D картинку (реальный мир) в 2D изображение (кадр видеоклипа). С помощью этой матрицы мы можем сопоставить расстояние, которое пробежал человек на самом деле (в спортзале) в метрах, с расстоянием, которое человек преодолел в кадре в пикселях (px). Для вычисления матрицы проективных преобразований требуется отметить набор опорных точек на полу снимаемой сцены и измерить их физические координаты в метрах. Затем вычислить координаты этих самых точек в пикселях. Минимальное количество точек – 4, однако для получения более точной матрицы на видео отмечено 10 (на рисунке 7 точки отмечены с помощью жёлтых квадратиков).

Матрица была вычислена с помощью программы, написанной на языке Matlab. Вычисления проводились с помощью готового алгоритма (*method_two.m*). Данный метод был взят из папки «*inverse matrix*», расположенной в директории «*A_Prolog/EXAMPLES/Demo programs/Vision/*» дистрибутива Акторного Пролога. Вычисления и сама матрица представлены на рисунке 6.

```
% Это в спортзале (версия 3):
%
input_points= [0, 791; 0, 527; 0, 293; 288, 781.5; 288, 437; 288, 127];
base_points= [251,264; 512,256: 742,248; 166,463: 599,445; 986,431];
%
```

```
Inverse matrix of projective transformation:
   0.1057   -1.6064    0.0001
   3.0386    0.9820    0.0023
-830.8996 1416.8348    1.0000
This matrix is to be used in the demos.
```

Рисунок 6. Вычисление матрицы с помощью программы на языке Matlab.

Для того чтобы убедиться, что точки были отмечены корректно, а соответственно, и матрица посчитана верно, было проверено, какой с помощью матрицы проективных преобразований получается вид сверху на видеосцену. Если всё было посчитано правильно, параллельные линии на полу должны выглядеть действительно параллельными на изображении. В правильности матрицы можно убедиться на рисунке 7. Слева на рисунке 7 показан вид спереди, справа на рисунке, соответственно, вид сверху, полученный с помощью обратных проективных преобразований. Заметим, что из-за сжатия картинки круг на изображении получился в виде эллипса.

Рисунок 7. Преобразование видеоклипа в «вид сверху».

Для преобразования видеоклипа в «вид сверху» также использовался готовый скрипт (*reconstruct_01.m*). Он находится в той же папке, что и скрипт для вычисления матрицы (*method_two.m*).

Теперь, поскольку все предварительные вычисления были выполнены верно, можно приступить к изучению низкоуровневого анализа видеоизображений в Акторном Прологе.

1. Для начала было проверено, как на данном видеоклипе работает предикат, который выделяет передний план – *get_foreground_image*. Алгоритм вычитания фона в этом случае сработал корректно, на экране присутствовало минимальное количество цифровых шумов. Одной из причин возникновения пятен на переднем плане является присутствие цифровых шумов в камере во время съёмок, т.е. пиксели на изображении могут случайным образом изменять свою яркость, именно поэтому для устранения пятен выполняется усреднение по N-му количеству кадров.

   Проходящую и пробегающую девушку программа распознала, как и следовало, в качестве переднего плана. Однако выявилась следующая проблема: на видеоизображении присутствовало отражение девушки в полу (из-за лакированного пола и хорошего освещения). Результаты можно увидеть на рисунке 8.

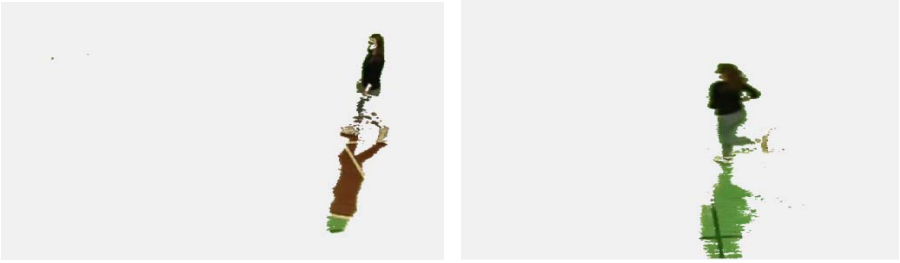

Рисунок 8. Результаты работы предиката get_foreground_image.

В связи с этим, проверяя, как работает предикат *get_blobs* (предикат, возвращающий блобы) можно убедиться в том, что блоб получается примерно в два раза больше, чем следовало, т.е. он захватывает помимо самого человека ещё и его отражение (см. рисунок 9).

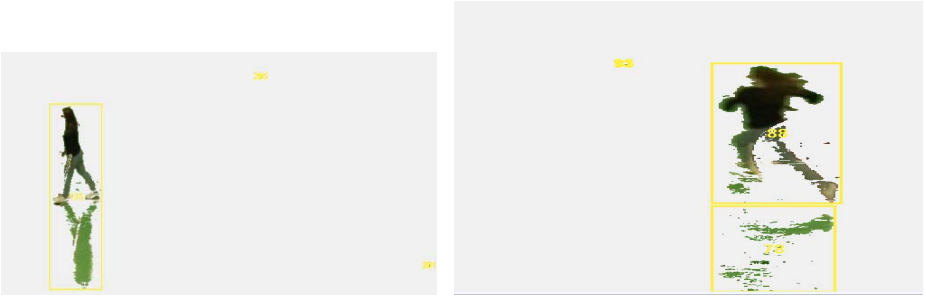

Рисунок 9. Результаты работы предиката get_blobs.

Логичным будет предположить, что, сталкиваясь с такой проблемой, программа, которая строит треки и вычисляет скорость человека, тоже будет ошибаться. Убедимся в этом.

2. Программа, с помощью которой будет реализован данный эксперимент, выводит на экран легенду с цветами скоростей и рисует трек движущегося человека цветом, соответствующим легенде. При запуске программы было обнаружено, что процесс тормозится (кадры поступают с большим интервалом, в связи с этим человек движется в кадре рывками). По этой причине обрабатывать данные на медленном компьютере было нецелесообразно. При уменьшении разрешения изображения до 320x240 px проблема, к сожалению, не решилась. В связи с этим было принято решение проводить анализ видеоизображения на более мощном компьютере. В качестве такового был выбран стационарный компьютер модели iMac с процессором Intel Xeon CPU E5645 @2.40GHz 2.53GHz (2 процессора), 64-разрядная ОС, установленная память 18.0 ГБ.

   Для наглядности примера также был использован предикат *get_foreground_image* (программа рисовала передний план). Результаты построения трека и вычисления скорости представлены на рисунке 10.

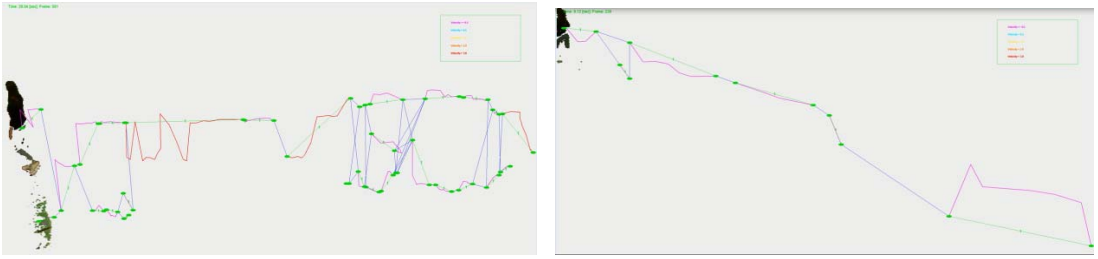

Рисунок 10. Построение трека и вычисление скорости.

Из представленных рисунков видно, что форма блоба постоянно меняется, потому что алгоритм выделения блобов то включает отражение человека в состав блоба, то нет. В некоторых случаях алгоритм даже ошибочно включает в состав блоба отражение человека, но не его самого. В результате трек строится как для человека, так и для его отражения, поэтому на некоторых участках можно наблюдать трек в виде «пилы». По этой причине вычисление скорости происходит некорректно. На рисунке, где изображена ходьба, данная ошибка представлена более явно, на различных участках – различная скорость, далёкая от реальной. На изображении, где девушка бежит, скорость вычисляется более правдоподобно по причине малого отражения.

Чтобы изучить алгоритм вычисления скорости и избежать при этом проблем с отражением, было снято ещё несколько видеороликов, в которых пол был застелен зелёными (версия 4) и чёрными (версия 5) матами. Помимо этого, теперь девушка шла и бежала двумя разными способами: параллельно камере и под углом (по диагонали). Также были сняты варианты видео, где девушка доходила до середины пути спокойным шагом, а затем разгонялась и убегала. Во время съёмок была вычислена реальная скорость девушки, которая для спокойного шага составляет 1,1 м/с, а для бега – 3,8 м/с. Все эти эксперименты были выполнены с той целью, чтобы понять, в каком случае алгоритм вычисляет скорость более точно, зависит ли точность оценки скорости от угла траектории и расположения человека относительно камеры.

Качество видео, как уже говорилось выше, соответствовало 1280x720 px. Координаты отмеченных точек в пикселях, а также обратные матрицы (для расчёта скорости) обоих версий видео представлены в Приложении А.

Во время тестирования алгоритма программа частично теряла видеозахват, т.е. строила трек рывками, что, соответственно, влекло за собой неверно посчитанную скорость. Чтобы попытаться справиться с данной проблемой, были изменены некоторые параметры:

- Была отключена медианная фильтрация (apply_median_filtering_to_velocity= 'no'). Медианный фильтр – один из видов цифровых фильтров, используемых для уменьшения уровня шума. Предполагалось, что медианная фильтрация значений скорости может сильно тормозить процесс выполнения программы.
- Был задан минимальный размер блобов (minimal_blob_size= 10000) по причине того, что на видеоизображении имели место шумы, которые программа выделяла как отдельные объекты – блобы.
- Было задано максимальное время усреднения фона (minimal_training_interval= 100, maximal_training_interval= 100), поскольку в то время как движущийся объект где-то притормаживал, программа начинала воспринимать его как задний план (элемент фона).
- Был увеличен порог выделения переднего плана (background_standard_deviation_factor= 1.7), чтобы предотвратить ошибочное распознавания шума в качестве блобов.

Благодаря настройке указанных выше параметров, удалось обеспечить нормальный видеозахват людей. Результаты работы алгоритма можно увидеть на рисунке 11.

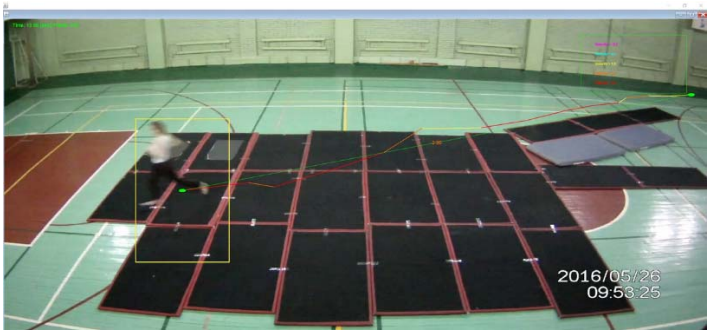

Рисунок 11. Построение трека и вычисление скорости объекта на видеоизображении с разрешением 1280x720 px.

Ознакомиться со значениями скоростей, полученных с помощью алгоритма вычисления скорости, можно в таблице 2.1.

Таблица 2.1 – Значения скоростей для видео высокого разрешения (1280x720 px).

Бег	5,5	4,53	3,4	3,5	1,4	1,4	2,2	1,9
Шаг	1,4	1,5	0,6	0,6	0,55	0,6	0,8	0,9

Как видно из таблицы, диапазон скоростей получился достаточно большим – для шага от 0,55 до 1,5 м/с, для бега от 1,4 до 5,5 м/с. В связи с таким разбросом значений скорости было принято решение протестировать алгоритм при более низком разрешении изображения – 640x480 px. Это позволило включить медианную фильтрацию (apply_median_filtering_to_velocity= 'yes', velocity_median_filter_halfwidth= 3), что, соответственно, повлекло за собой изменение некоторых других параметров: minimal_blob_size= 1000, при низком разрешении размер блобов уменьшился; фон усреднялся теперь в течение всего времени осуществления действий на сцене, иначе выявленные на первых кадрах шумы продолжали существовать как ошибочные блобы. Видеозахват в таком случае осуществлялся корректно, т.е. не терялся.

Также стоит заметить, что при уменьшении разрешения видеоизображения значения отмеченных точек в пикселях изменились, а соответственно изменились и их обратные матрицы. Проследить данное изменение для обоих видео (версия 4 и версия 5) можно в Приложении А.

Полученные значения скорости при низком разрешении можно сравнить со значениями при высоком разрешении видеоизображений в таблице 2.2. Результаты представлены в относительных единицах – отношение вычисленной скорости к реальной.

Таблица 2.2 – Сравнительная таблица скоростей для высокого и низкого разрешения видеоизображений. Результаты представлены в относительных единицах «отношение вычисленной скорости к реальной».

	Бег		**Шаг**	
	Высокое разрешение	**Низкое разрешение**	**Высокое разрешение**	**Низкое разрешение**
	1,45	0,89	1,27	0,55
	1,18	0,97	1,36	0,95
	0,89	0,76	0,55	0,18
	0,92	0,68	0,55	0,55
	0,37	0,27	0,45	0,18
	0,37	0,34	0,55	0,45
	0,58	0,24	0,73	0,18
	0,5	0,29	0,82	0,36
Ср. знач.	0,7825	0,555	0,785	0,425

Стоить заметить, что, если оценивать единичный результат, т.е. взять полученную скорость, максимально приближенную к реальному значению скорости (значения близкие к единице), получается, что он характерен для низкого разрешения (данные значения выделены зелёным цветом). Однако, если оценивать полученные результаты по среднему значению, то выходит наоборот. При высоком разрешении алгоритм оценивает скорость лучше, как для бега, так и для спокойного шага. Отсюда видно, что результаты носят противоречивый характер. Поэтому, для того, чтобы получить более конкретные выводы, анализ следует провести более подробно.

Чтобы наглядно представить экспериментально полученные данные и изучить работу алгоритма в зависимости от изменения определённых условий, вычисленные значения скоростей сравнивались по параметрам: в зависимости от угла направления траектории, в зависимости от наличия медианной фильтрации и в зависимости от разрешения изображения, которое

было описано выше. Все результаты также приведены в относительных единицах.

В таблице 2.3 рассмотрены результаты, зависимость точности оценки скорости движения от угла, под которым шла или пробегала девушка.

Таблица 2.3 – Сравнительная таблица значений скорости, в зависимости от угла направления траектории движения. Результаты представлены в относительных единицах «отношение вычисленной скорости к реальной».

	Бег		**Шаг**	
	Параллельно	**По диагонали**	**Параллельно**	**По диагонали**
	0,37	1,45	1,27	0,73
	0,37	0,89	0,55	0,82
	0,27	0,58	0,45	0,18
	0,34	1,18	1,36	0,36
		0,92	0,55	
		0,5	0,55	
		0,89	0,55	
		0,97	0,95	
		0,76	0,18	
		0,68	0,18	
		0,24	0,45	
		0,29	0,55	
Ср.знач.	0,3375	0,77916667	0,6325	0,5225

Из таблицы видно, что влияние данного параметра на работу алгоритма неоднозначно, т.к. для пробегающей девушки максимально приближенная к реальной скорости оценка получается при диагональном направлении, а для спокойного шага при параллельном, как при оценке единичных значений скорости, так и среднего значения. Возможно, разброс данных величин носит случайный характер, и требует более точного анализа.

В следующей таблице (таблица 2.4) показаны результаты оценки скорости, в зависимости от использования медианной фильтрации (включена она или нет).

Таблица 2.4 – Сравнительная таблица значений скорости, в зависимости от использования медианной фильтрации. Результаты представлены в относительных единицах «отношение вычисленной скорости к реальной».

	Бег		**Шаг**	
	МФ	**Без МФ**	**МФ**	**Без МФ**
	1,45	1,18	1,27	1,36
	0,89	0,92	0,55	0,55
	0,58	0,5	0,45	0,55
	0,37	0,37	0,73	0,82
	0,89	0,97	0,55	0,95
	0,76	0,68	0,18	0,55
	0,24	0,29	0,18	0,45
	0,27	0,34	0,18	0,36
Ср. знач.	0,68125	0,65625	0,51125	0,69875

Значения, максимально приближенные к единице, как для бега, так и для спокойного шага, соответствуют скорости, вычисленной без медианного фильтра (значения выделены зелёным цветом). Однако, если оценивать полученные данные по среднему значению, видно, что результаты друг от друга практически не отличаются. То, что алгоритм максимально приближенную к реальной скорость вычислил без использования медианного фильтра, можно попытаться объяснить следующим образом: медианная фильтрация используется для уменьшения уровня шумов за счёт удаления выбросов (ошибочных значений большой величины), наряду с этим она удаляет также и экстремумы в последовательности значений скорости, возникающие из-за неравномерного движения человека. Таким образом, отсутствие этих величин может привести к заниженным оценкам средней

скорости. Однако, несмотря на это, для более точного объяснения влияния данного параметра на алгоритм требуется более глубокий анализ.

Общая таблица, которая содержит все значения скоростей, вычисленные с помощью алгоритма, приведена в Приложении Б (Таблица Б.1).

Для всех полученных значений скорости было посчитано среднеквадратическое отклонение (б). Среднеквадратическое отклонение характеризует меру рассеяния данных. В настоящей работе этот показатель используется для определения погрешности серии последовательных измерений скорости. Для скорости, соответствующей бегу, сигма составила 1.37, для шага – 0.39. Для того чтобы воспользоваться правилом трёх сигм, были вычислены средние скорости: средняя оценка скорости бега соответствует 2.55 м/с, шага 0.66 м/с. Из полученных значений можно сделать вывод, что средние значения скорости отличаются от реальных не больше, чем на одну сигму, что, по правилу трёх сигм, говорит о нормально распределённой случайной величине. В нашем случае подтверждение данного правила доказывает корректность работы алгоритма.

Благодаря поставленным экспериментам, были выявлены проблемы в алгоритмах низкоуровневой обработки видеоизображений, использованных для реализации метода логического программирования интеллектуального видеонаблюдения аномального поведения людей в Акторном Прологе. Решение данных проблем позволит методу быть независимым от каких-либо внешних воздействий (влияние ветра, освещения и т.д.), а также от некоторых параметров наблюдаемой среды (покрытие пола).

Программный код, с помощью которого был изучен алгоритм вычисления скорости, прилагается к настоящей работе на электронном носителе (CD-ROM), пример под названием «Velocity».

2.3 Высокий уровень обработки видеоизображения

Для высокоуровневой обработки видеоизображений, а именно, для описания и анализа поведения людей используются следующие принципы:

1. Язык Пролога, сам по себе, является декларативным языком с выразительными возможностями, которые позволяют, например, сформулировать задачу распознавания аномального поведения людей в терминах анализа графов движения блобов. Однако неизбежно приходится сталкиваться с необходимостью введения элементов нечёткого логического вывода. Это связано с неоднозначностью результатов низкоуровневого анализа и расплывчатостью самого сценария аномального поведения.
2. Простейшие элементы нечёткого логического вывода могут быть легко введены в язык Пролог с помощью стандартных арифметических предикатов языка. Например, предикат для распознавания бегущего человека может учитывать одновременно две характеристики блобов, а именно, среднюю скорость блоба и длину пути. Объединение этих двух характеристик выполняется с помощью очень простой нечёткой метрики, описанной с помощью арифметических функций. С точки зрения декларативной семантики языка, процедура распознавания бегущего человека будет формулой логики предикатов первого порядка.

Для реализации высокоуровневой обработки видеоизображений используется объектно-ориентированный логический язык Акторный Пролог. На низком уровне обработки логическая программа анализирует видеоизображение и представляет результаты работы в виде графов, которые в свою очередь представлены в виде термов Пролога – структур, списков и т.д. На высоком уровне обработки видеоизображений логическая программа анализирует данные, полученные на низком уровне, и на этой основе делает выводы – доказывает, что один блоб представляет собой бегущего человека, другой идущего и т.д., в зависимости от того, какие предикаты будут заданы в программе.

3 Изучение предметной области. Постановка задач для интеллектуального видеонаблюдения.

В привычном для нас понимании, под аномальным поведением подразумеваются такие ситуации как драка, разбой, теракты, люди, бегущие с высокой скоростью и т.д. Такое определение является несколько неподходящим для области, где все подчиняются строгим правилам, для Атомной электростанции. Так как данная отрасль является ключевым моментом настоящей работы, было важно изучить, какое поведение сотрудников АЭС недопустимо, и что для интеллектуального видеонаблюдения можно было бы считать аномальным поведением.

В связи с этим, Главным инженером Портновым Александром Алексеевичем, Начальником службы эксплуатации реактора Яковлевым Леонидом Ивановичем, Начальником службы обеспечения и контроля радиационной безопасности Савкиным Владимиром Алексеевичем, Инженером 1 категории службы СУЗ (система управления и защиты) Карцевым Константином Павловичем была проведена экскурсия на Реактор ИРТ Атомного центра МИФИ (ИРТ МИФИ), сооружённого по типовому проекту ТП-3304М, который относится к универсальным исследовательским ядерным реакторам средней мощности. Данный реактор был запущен 26 мая 1967 года и действовал до 2009 года. В настоящее время реактор находится в недействующем, но работоспособном состоянии.

Сотрудниками были рассказаны реальные истории, когда нарушались правила техники безопасности. Одной из них был пример того, как сотрудник оставил ящик с инструментами в помещении, в котором была проведена дезактивация (дезактивация – удаление или снижение радиоактивного загрязнения с какой-либо поверхности или из какой-либо среды). Был предоставлен ряд документов (инструкция по безопасности труда, радиационной технике безопасности в комплексе ИРТ МИФИ, должностные инструкции начальника смены и инженера по управлению

центральной службой), в которых содержатся правила, соблюдение которых обеспечивает безопасную и эффективную работу.

По результатам изучения полученной документации, были сделаны следующие выводы и смоделированы ситуации, эквивалентные аномальному поведению [**7**]:

1. «В зоне строгого режима не допускается работа персонала без индивидуальных дозиметров». Такой предмет (дозиметр) может выступать в роли элемента спец. одежды, отсутствие которого является нарушением. «Работа в помещениях с повышенной концентрацией радиоактивных газов (концентрация выше контрольных уровней) должна производиться в скафандрах, необходимый поддув в которых обеспечивается специальной системой вентиляции (П-2)». В данном правиле также делается упор на специализированную одежду сотрудников АЭС. «В здании, где располагается реактор, сотрудники должны носить белые халаты, за исключением зон строго режима» (в таких зонах к специализированной одежде предъявляются более жёсткие требования, например, описанные выше). Все правила, перечисленные в данном пункте, можно отнести к одной ситуации, которая может быть названа следующим образом: «Сотрудник АЭС не одет в специализированную одежду, соответствующую зоне, в которой он находится».
2. «Запрещено оставлять дозиметр вблизи источников ионизирующих излучений и в любом другом месте, кроме стенда обмена дозиметров». Дозиметр можно рассматривать как оставленный без присмотра и в запрещённом месте предмет.

   К такой ситуации также может быть отнесён случай, когда в стерильном помещении (помещение, где была проведена дезактивация) был оставлен ящик с инструментами. Этот поступок может привести к неблагоприятным последствиям, например, к радиоактивному заражению и, соответственно, необходимости повторной дезактивации.

3. На действующем реакторе пультовая комната всегда должна быть под присмотром, в ней должны присутствовать от 1 до 3 человек. Соответственно, выявление ситуации, когда пультовая остаётся пустой, может рассматриваться как ещё одна задача для видеонаблюдения.

   Если представить ситуацию, когда сотрудник АЭС во время смены засыпает или теряет сознание за пультом управления реактором, то можно прийти к аналогичной задаче, так как за показаниями приборов уже никто не следит. Анализируя вышесказанное, логично назвать распознаваемую ситуацию так: «пультовая комната осталась без присмотра».
4. Существуют определённые нормы, регламентирующие количество дозы облучения за смену, квартал, полугодие, год и т.д. Также, чтобы избежать превышения предельной нормы облучения, существуют ограничения по времени нахождения в зоне строго режима: «При работе с источниками излучений должны предусматриваться меры по ограничению длительности пребывания людей вблизи источников ионизирующих излучений». Такие правила должны строго соблюдаться, так как их нарушение влечёт за собой отстранение от работы. Следовательно, для видеонаблюдения является актуальной ещё одна ситуация, когда сотрудник АЭС находится вблизи источника излучения выше предельно допустимого времени.

Среди перечисленных ситуаций, по моему мнению, наиболее приоритетной является ситуация с пультовой комнатой. Именно в ней можно увидеть все процессы, происходящие на реакторе, и оценить уровень безопасности. Поэтому в настоящей работе метод интеллектуального видеонаблюдения будет описан на основе примера логической программы, с помощью которой можно отслеживать наличие сотрудников в пультовой, а также отслеживать уровень их активности – потерял сознание (уснул) человек или находится в работоспособном состоянии.

4 Реализация метода интеллектуального видеонаблюдения на примере наблюдения за персоналом в пультовой комнате АЭС

4.1 Описание метода

Рассмотрим применение метода логического программирования интеллектуального видеонаблюдения на примере программы, анализирующей видео в реальном времени с целью выявления отсутствия оператора АЭС на рабочем месте и анализа его состояния – уснул человек или потерял сознание. Программа состоит из трёх файлов: в файле с расширением a (.a) описывается вся логика программы; в файле с расширением def (.def) содержатся определения доменов и предикатов; в файле с расширением dlg (.dlg) указывается местонахождение файлов данных, которые прилагаются к программе.

Для создания программы используются предварительно определённые пакеты Акторного Пролога: *Java2D*, который используется для работы с двумерной графикой, и *Vision*, который реализует процесс обработки видео низкого уровня.

```
import .. from "morozov/Java2D";
import .. from "morozov/Vision".
```

Класс *«Main»* – главный класс, с которого начинается выполнение программы. Этот класс является наследником класса *«Timer»*. Класс *«Timer»* нужен для последовательного считывания видеоизображения – он вызывает определённый предикат (в нашем случае предикат *tick*) через заданные промежутки времени. Класс *«Main»* включает следующие слоты: *sampling_rate = 25*, который обозначает частоту поступления кадров; *circle_radius = 0.01*, который задаёт радиус круга на рисунках; *subtractor*, который содержит экземпляр встроенного класса *«ImageSubtractor»*, который, в свою очередь, отвечает за выполнение низкого уровня обработки (вычитание фона, обнаружение блобов, построение треков, вычисление скорости, построение графов). Слоты этого класса будут рассмотрены ниже.

Далее, класс *«Main»* содержит слот *graphic_window,* который содержит экземпляр встроенного класса *«Canvas2D»*. Данный слот задаёт атрибуты графического окна: *background_color = 'SystemControl'* – цвет, используемый операционной системой по умолчанию (серый), и высоту окна *height= 22*. Слот *prompt_window* содержит экземпляр встроенного класса *«Report»,* который отвечает за создание текстового окна (атрибуты задаются по аналогии с графическим окном). Кроме того, в классе *«Main»* определяются такие слоты как *text*, который содержит экземпляр встроенного класса «*Text*», выполняющий текстовые операции, слот *image*, который содержит экземпляр встроенного класса «*BufferedImage*», и слот *state*, который содержит экземпляр класса «*ProgramState*» – локальная база данных, в которой хранятся текущее время и номер кадра.

```
class 'Main' (specialized 'Timer'):
constant:
        sampling_rate           = 25;
        circle_radius           = 0.01;
internal:
        subtractor    = ('ImageSubtractor',
                                extract_blobs= 'yes',
                                track_blobs= 'yes',
                                use_grayscale_colors= 'yes',
                                background_standard_deviation_factor= 1.2,
                                minimal_blob_size= 10000);
        graphic_window  = ('Canvas2D',
                                background_color= 'SystemControl',
                                y= 0,
                                height= 22);
        prompt_window   = ('Report',
                                font_size= 24,
                                y= 22,
                                height= 3);
        text            = ('Text');
        image           = ('BufferedImage');
        state           = ('ProgramState');
```

4.1.1 Низкий уровень обработки видеоизображения

В конструкторе класса *«ImageSubtractor»* используются следующие значения аргументов: *extract_blobs* и *track_blobs*, которые включают

(приводят в действие) вычисление блобов и треков. Слот *use_grayscale_colors* запускает преобразование изображений в формат шкалы яркости. Слот *background_standard_deviation_factor* задаёт порог для вычитания фона. Слот *minimal_blob_size* задаёт минимальный размер обнаруженного блоба.

Рассмотрим выполнение класса «*ImageSubtractor*», отвечающего за вход видеоизображения и всю низкоуровневую обработку:

```
goal:-!,
        graphic_window ? show,
        prompt_window ? write(
                "Видео (c) 2016 ",
                "Анна Булычева и Юлия Тукмачёва, "
                "каф.82 МИФИ"),
        Time0== ?milliseconds(),
        state ? set_beginning_time(Time0),
        set_period((1/sampling_rate/2),0),
        activate.
```

С помощью предиката *show* открывается графическое окно. В текстовом окне при помощи предиката *write* выводится сообщение. Встроенная функция *milliseconds* возвращает текущее значение времени в миллисекундах, затем, с помощью предиката *set_beginning_time* текущее время записывается в локальную базу данных (БД) *state*. После чего задаётся период вызова предиката *tick* (встроенный предикат *set_period* с аргументами 1/*sampling_rate* – период в секундах, и 0, задержка по времени, после которой начинается вызов предиката *tick*). Затем активируется счётчик (предикат *activate*), вызывающий предикат *tick.*

```
tick:-
        T2== ?milliseconds(),
        state ? get_beginning_time(T1),
        Delta== (T2 - T1) / 1000.0 * sampling_rate,
        N== ?convert_to_integer(?round(Delta)),!,
        load_figure(N,T2).
```

С каждым вызовом предиката *tick* рассчитывается время (в миллисекундах), прошедшее с начала ввода видеоизображения (ищется

разница между текущим значением времени, полученным с помощью встроенной функции *milliseconds,* и временем, занесённым до этого в БД). Затем полученное в миллисекундах время пересчитывается в номер кадра (*Delta*), который должен быть показан в данный момент времени, и передаётся в предикат *load_figure* в качестве аргумента.

```
load_figure (N2,_):-
        state ? get_current_frame (N1),
                N1 == N2,!.
        load_figure (N,_):-
                state ? set_current_frame (N),
                ImageToBeLoaded==
                        "E:/рабочий стол/учеба/ДИПЛОМ/” +
                        “видео для ВКР/версия 4/кадры/two people/" +
                        "two people " + text?format("%04d",N) + ".jpg",
                image ? does_exist (ImageToBeLoaded),!,
                image ? load (ImageToBeLoaded),
                subtractor ? subtract (N,image),
                draw_scene.
        load_figure (_,T2):-
                state ? set_beginning_time (T2),
                subtractor ? reset_results.
```

Предикат *load_figure* (первое правило) сравнивает номер кадра, вычисленный с помощью предиката *tick* (т.е. тот кадр, который должен быть показан на экране), с номером текущего кадра (текущий кадр достаётся из БД с помощью предиката *get_current_frame*), и если они не совпадают, то загружает кадр в память из файла с соответствующим именем (переходит к выполнению второго правила). Имена файлов создаются автоматически и записываются в переменную *ImageToBeLoaded*. Загруженные изображения физически хранятся в экземпляре класса «*BufferedImage*» (слот *image*). Далее, в экземпляре класса «*ImageSubtractor*» вызывается встроенный предикат *subtractor*, первым аргументом которого является номер текущего кадра, а вторым – загруженное из файла изображение. Класс «*ImageSubtractor*» реализует низкоуровневую обработку изображения, все результаты

обработки остаются во внутренних массивах класса и могут быть извлечены, когда это потребуется.

В то время, когда видеоклип заканчивается, т.е. исчерпана последовательность файлов, предикат *does_exist* во втором правиле *load_figure* терпит неудачу, и вместо второго правила срабатывает третье. В результате этого устанавливается новое начальное время ввода видеоизображения, и все результаты, накопленные в экземпляре класса «*ImageSubtractor*», аннулируются (при помощи встроенного предиката *reset_results*). Обработка видеоклипа начинается заново.

4.1.2 Высокий уровень обработки видеоизображения

Обработке высокого уровня подвергаются данные, вычисленные на этапе низкоуровневой обработки. С помощью процедуры *draw_scene()* выполняется логический анализ сцены и проводится визуализация результатов интеллектуального видеонаблюдения:

```
draw_scene:-
        subtractor ? commit,
        subtractor ? get_recent_frame_number(F),
        graphic_window ? suspend_redrawing,
        graphic_window ? clear,
        graphic_window ? set_font({size:18}),
        graphic_window ? set_text_alignment('CENTER','CENTER'),
        subtractor ? get_recent_image(image),
        graphic_window ? draw_image(image,0,0,1,1),
        subtractor ? get_connected_graphs(Graphs),
        image ? get_size_in_pixels(IW,IH),
        draw_graphs(IW,IH,Graphs,F),
        check_graphs(Graphs),
        graphic_window ? draw_now.
```

Встроенный предикат *commit* класса «*ImageSubtractor*» вычисляет графы движения объектов на момент последнего кадра, обработанного классом. Встроенный предикат *get_recent_frame_number* возвращает номер последнего обработанного кадра к моменту вызова предиката *commit*. Встроенный предикат *get_recent_image* возвращает соответствующее

изображение (данные, записанные в экземпляре класса «*BufferedImage*», который хранится в слоте *image*). Встроенный предикат *get_connected_graphs* возвращает список связных графов движения объектов. Используются следующие предикаты встроенного класса «*Canvas2D*»: *suspend_redrawing* – приостанавливает перерисовку изображения в окне, *clear* – чистит окно, *set_font* – задаёт размер шрифта, *set_text_alignment* – выравнивает текст, *draw_image* – рисует изображение (первый аргумент является экземпляром класса «*BufferedImage*», второй и третий являются координатами верхнего левого угла изображения, четвертый и пятый являются шириной и высотой изображения по шкале от 0 до 1), и *draw_now* – возобновляет перерисовку изображения в окне. Предикат *draw_graphs* анализирует графы движения блобов и демонстрирует на экране дополнительную информацию в виде линий, прямоугольников и надписей. Его аргументами являются ширина и высота изображения (полученные с помощью предиката *get_size_in_pixels* встроенного класса «*BufferedImage*»), список графов и номер текущего кадра. Предикат *check_graphs* осуществляет проверку графов (будет описано ниже).

```
draw_graphs(IW,IH,[Graph|Rest],F):-!,
        draw_graph(IW,IH,Graph,Graph,F),
        draw_graphs(IW,IH,Rest,F).
draw_graphs(_,_,_,_).
```

Предикат *draw_graphs* разматывает список графов, анализирует их, рисует траектории блобов на экране, используя предикат *draw_graph,* и выводит результаты логического анализа.

Вычисленные графы поступают в виде списка, который имеет голову и хвост. В представленном выше предикате от списка графов отделяется голова – первый вычисленный программой граф, который затем обрабатывается предикатом *draw_graph.* После успешной обработки графа выполняется рекурсивный вызов, т.е. предикат *draw_graphs* вызывается вновь и обрабатывает уже следующий граф, содержащийся в списке. Операция выполняется до тех пор, пока не будут обработаны все графы из списка.

```
draw_graph(IW,IH,[Edge|Rest],Graph,F):-!,
        draw_edge(IW,IH,Edge,Graph,F),
        draw_graph(IW,IH,Rest,Graph,F).
draw_graph(_,_,_,_,_).
```

Отделённый от списка в предикате *draw_graphs* граф поступает в предикат *draw_graph* в виде списка дуг, имеющихся в графе. Этот список разматывается (аналогично списку, обрабатываемому предикатом *draw_graphs*) и идёт в обработку предиката *draw_edge.*

```
draw_edge( IW,IH,
                {       x1:X1a,y1:Y1a,x2:X2a,y2:Y2a,
                        inputs: Origins,
                        coordinates: TrackOfBlobs|_},
                Graph,F):-
        X1r== X1a / IW,
        Y1r== Y1a / IH,
        X2r== X2a / IW,
        Y2r== Y2a / IH,
        graphic_window ? set_brush('Blue'),
        draw_origins(X1a,Y1a,IW,IH,Origins,Graph),
        graphic_window ? set_pen({color:'Magenta',lineWidth:2}),
        draw_track_of_blobs(IW,IH,TrackOfBlobs),
        draw_rectangle(TrackOfBlobs,IW,IH,F),
        graphic_window ? set_brush('on'),
        graphic_window ? set_pen({color:'Green',lineWidth:1}),
        graphic_window ? draw_line(X1r,Y1r,X2r,Y2r),
        draw_circle(X1r,Y1r),
        draw_circle(X2r,Y2r),
        fail.
draw_edge(_,_,_,_,_).
```

Предикат *draw_edge* рисует на экране дуги графа. Он имеет следующие аргументы: *IW*, *IH* – ширина и высота изображения, недоопределённое множество (имеющее следующие поименованные пары: *x1:X1a, y1:Y1a, x2:X2a, y2:Y2a* – координаты начала и конца дуги, *inputs: Origins* – список номеров предшествующих дуг, coordinates: *TrackOfBlobs* – координаты блобов, принадлежащих данной дуге), *Graph* – граф, *F* – номер соответствующего кадра. Координаты дуги переводятся в пиксели. Затем

координаты начала дуги и список номеров предшествующих дуг передаются в качестве аргументов предикату *draw_origins.*

```
draw_origins(Sx,Sy,IW,IH,[Origin|Rest],Graph):-!,
        draw_origin(Sx,Sy,IW,IH,Origin,Graph),
        draw_origins(Sx,Sy,IW,IH,Rest,Graph).
draw_origins(_,_,_,_,_,_).
```

Предикат *draw_origins* соединяет на рисунке несоединённые дуги, чтобы можно было наглядно увидеть, как меняется позиция человека. Из списка номеров дуг извлекается голова (т.е. номер какой-либо дуги) и передаётся в обработку предиката *draw_origin.*

```
draw_origin(SxA,SyA,IW,IH,N,Graph):-
        get_edge(N,Graph,Edge),
        Edge == {x2:Nx2a,y2:Ny2a|_},!,
        draw_origin_arrow(SxA,SyA,Nx2a,Ny2a,IW,IH).
draw_origin(_,_,_,_,_,_).
```

Предикат *get_edge* извлекает из графа дугу, которая соответствует номеру *N*. После задаются координаты её конца, и с помощью предиката *draw_origin_arrow* рисуется линия, соединяющая конец предшествующей вычисленной дуги и начало следующей дуги (дуга, для которой вычислялись предшественники).

```
get_edge(1,[Edge|_],Edge):-!.
get_edge(N,[_|Rest],Edge):-
        N > 0,
        get_edge(N-1,Rest,Edge).
```

Суть предиката *get_edge*: предикат *get_edge* разматывает граф до тех пор, пока *N* не будет равно единице, к тому времени на месте головы списка окажется дуга, соответствующая номеру *N*, которая будет возвращена через третий аргумент предиката *get_edge* для дальнейшей обработки.

```
draw_origin_arrow(X,Y,X,Y,_,_):-!.
draw_origin_arrow(SxA,SyA,Nx2a,Ny2a,IW,IH):-
        SxR== SxA / IW,
        SyR== SyA / IH,
        Nx2r== Nx2a / IW,
        Ny2r== Ny2a / IH,
```

```
graphic_window ? draw_line(SxR,SyR,Nx2r,Ny2r).
```

Первое правило *draw_origin_arrow* проверяет, совпадает ли конец предшествующей дуги с началом следующей дуги (дуга, для которой вычислялись предшественники). Если совпадение есть, то предикат просто заканчивается успехом. Если первое правило претерпевает неудачу, то начинает выполняться второе правило, которое переводит координаты в пиксели и соединяет их тонкой синей линией (*set_brush('Blue')*).

Затем вызывается предикат *draw_track_of_blobs,* который рисует треки движущихся блобов пурпурным цветом с шириной линии, равной 2 px (использован предикат *set_pen*).

```
draw_track_of_blobs(IW,IH,[Blob|Rest]):-
        Blob== {x:Xa,y:Ya|_},!,
        Xr== Xa / IW,
        Yr== Ya / IH,
        draw_track_of_blobs(IW,IH,Xr,Yr,Rest).
draw_track_of_blobs(_,_,_).
draw_track_of_blobs(IW,IH,X1,Y1,[Blob|Rest]):-
        Blob== {x:Xa,y:Ya|_},!,
        X2== Xa / IW,
        Y2== Ya / IH,
        graphic_window ? draw_line(X1,Y1,X2,Y2),
        draw_track_of_blobs(IW,IH,X2,Y2,Rest).
draw_track_of_blobs(_,_,_,_,_).
```

Предикат *draw_track_of_blobs* получает аргумент в виде списка координат блоба (*TrackOfBlobs*). Данный предикат извлекает из списка координаты первого блоба и переводит их в пиксели. Аналогично вычисляются оставшиеся координаты блобов, и с помощью предиката *draw_line* рисуются линии, соединяющие вычисленные пиксели, тем самым получается трек блоба.

Далее с помощью предиката *draw_rectangle* рисуется прямоугольник, окружающий блоб.

```
draw_rectangle([Blob],IW,IH,F):-
        Blob == {frame:F,x:X0,y:Y0,width:W1,height:H1|_},!,
        graphic_window ? set_brush('off'),
```

```
        graphic_window ? set_pen({color:'Yellow',lineWidth:3}),
        X2== (X0 - W1 / 2) / IW,
        Y2== (Y0 - H1 / 2) / IH,
        W2== W1 / IW,
        H2== H1 / IH,
        graphic_window ? draw_rectangle(X2,Y2,W2,H2).
draw_rectangle([_|R],IW,IH,N):-!,
        draw_rectangle(R,IW,IH,N).
draw_rectangle(_,_,_,_).
```

Предикат *draw_rectangle* имеет следующие аргументы: координаты первого блоба, ширину и высоту изображения и *F* – номер кадра. Координаты блоба представляются в виде недоопределённого множества, имеющего следующие поименованные пары: *frame:F* – номер кадра, *x:X0, y:Y0* – координаты блоба, *width:W1, height:H1* – ширина и высота блоба. Затем вычисляется нижний левый угол блоба в пикселях, а также его ширина и высота, после чего жёлтой линией шириной 3 пикселя рисуется прямоугольник с шириной *W2* и высотой *H2*. Путём рекурсивного разматывания списка координат блоба (работает второе правило) находится последняя позиция блоба по времени, соответствующая дуге, и для неё строится жёлтый прямоугольник (работает первое правило). Третье правило срабатывает в том случае, если список пуст.

Далее в графическом окне с помощью предиката *draw_line* зелёной линией шириной 1 пиксель соединяются начало и конец дуги.

```
draw_circle(X0,Y0):-
        X1== X0 - circle_radius,
        Y1== Y0 - circle_radius,
        Width== circle_radius * 2,
        Height== circle_radius * 2,
        graphic_window ? draw_ellipse(X1,Y1,Width,Height).
```

Предикат *draw_circle* вычисляет координаты эллипса, его ширину и высоту, и, используя предикат *draw_ellipse*, рисует эллипсы по вычисленным параметрам в начале и конце дуги (эллипс конца дуги находится в центре прямоугольника – блоба).

Предикат *fail* означает неудачу при выполнении программы, он вызывает поиск следующего решения. Программа откатывается до места, где программа могла бы пойти по другому пути исполнения (и, возможно, вычислить другое решение). В примере представленной логической программы откат происходит до предиката *get_edge*, после чего программа переходит к выполнению второго правила, которое заканчивается успехом. Это влечёт за собой рекурсивный вызов предиката *draw_graphs,* который начинает обрабатывать следующий граф.

С помощью предиката *check_graphs* программа проверяет отсутствие людей на видеоизображении и, в случае успешного выполнения данного предиката, выводит на экран сообщение: «Сотрудник АЭС покинул пультовую комнату!». Графы и их дуги соответствуют траекториям движущихся объектов; исходя из этого, логично предположить, что, если список графов пуст, значит, движущихся объектов в кадре нет, из чего следует вывод о том, что пультовая комната осталась без присмотра.

```
check_graphs([]):-!,
        report_target_event.
check_graphs(_).
report_target_event:-
        graphic_window ? set_text_alignment('CENTER','CENTER'),
        graphic_window ? set_font({size:48}),
        graphic_window ? set_pen({color:'Red'}),
        graphic_window ? draw_text(
                0.5,0.2,
                " Сотрудник АЭС покинул пультовую комнату!").
```

Если первое правило *check_graphs* претерпевает неудачу, т.е. список графов не пуст, и, соответственно, в комнате есть движущиеся объекты, то программа переходит к выполнению второго правила, которое заканчивается успехом, и сообщение о нарушении правила техники безопасности не выводится.

Полный программный код представлен в Приложении В, а также на электронном носителе (CD-ROM) – пример под названием «Empty_room».

На рисунке 12 приведён результат работы программы, которая выявила отсутствие оператора АЭС на рабочем месте.

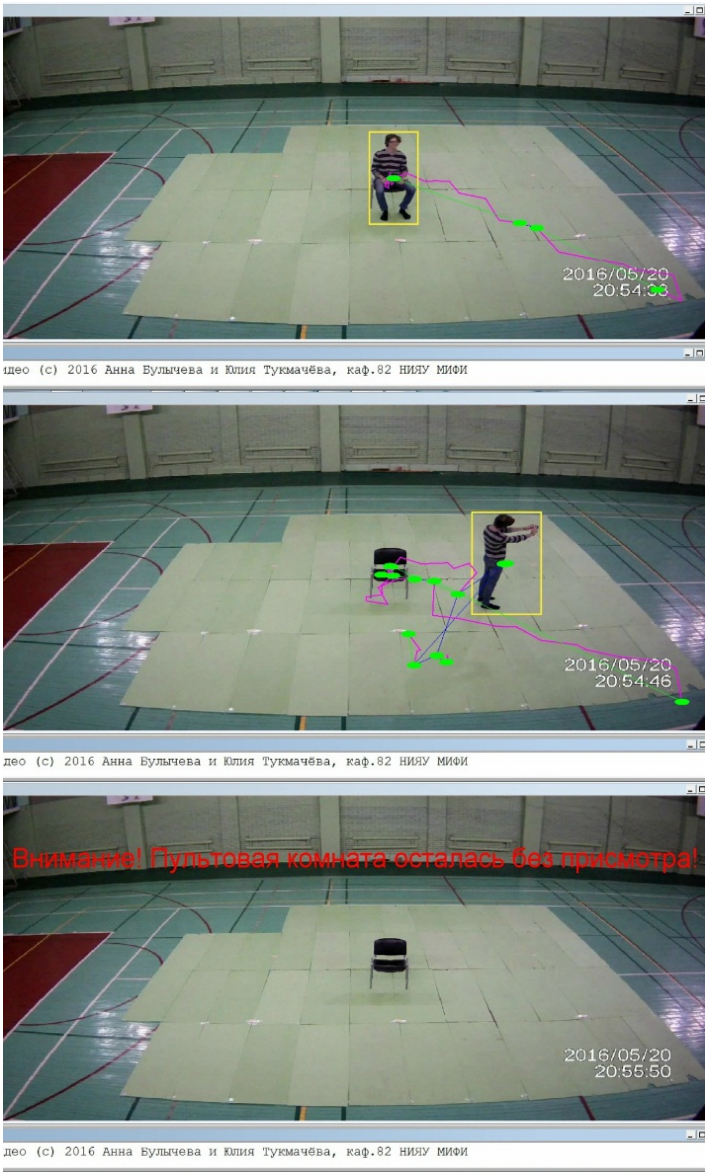

Рисунок 12. Результат работы программы, которая выявляет отсутствие оператора АЭС на рабочем месте.

Данный метод полностью подходит для проверки отсутствия движущихся людей в кадре.

Предикат, который выявляет уснувшего или потерявшего сознание диспетчера во время смены, представлен следующим образом.

```
check_graphs([Graph|Rest],CurrentFrame):-!,
        check_graph(Graph,CurrentFrame),
        check_graphs(Rest,CurrentFrame).
check_graphs(_,_).
check_graph([Edge|_],CurrentFrame):-
```

```
        Edge == {coordinates:TrackOfBlob,mean_velocity:V|_},
        there_is_dreaming_person(
                TrackOfBlob,'no',0,
                FirstFrame,
                LastFrame,V),
        LastFrame == CurrentFrame,!,
        Duration== (LastFrame - FirstFrame) / sampling_rate,
        report_target_event(Duration).
check_graph([_|Rest],CurrentFrame):-!,
        check_graph(Rest,CurrentFrame).
check_graph(_,_).
there_is_dreaming_person([Blob],'no',_,LF,LF,V):-
        this_is_a_dreaming_person(Blob,LF,V),!.
there_is_dreaming_person([Blob],'yes',FF,FF,LF,V):-
        this_is_a_dreaming_person(Blob,LF,V),!.
there_is_dreaming_person([Blob|Rest],'no',_,FF2,LF,V):-
        this_is_a_dreaming_person(Blob,FF1,V),!,
        there_is_dreaming_person(Rest,'yes',FF1,FF2,LF,V).
there_is_dreaming_person([Blob|Rest],'yes',FF1,FF2,LF,V):-
        this_is_a_dreaming_person(Blob,_,V),!,
        there_is_dreaming_person(Rest,'yes',FF1,FF2,LF,V).
there_is_dreaming_person([_|Rest],IsLying,FF1,FF2,LF,V):-
        there_is_dreaming_person(Rest,IsLying,FF1,FF2,LF,V).
this_is_a_dreaming_person(Blob,CurrentFrame,V):-
        Blob == {       frame:CurrentFrame,
                        width:W,
                        height:H,
                        characteristic_length:CL|_},
        RectangleArea== W * H,
        StandardizedArea== RectangleArea / (CL*CL),
        M1== ?fuzzy_metrics(StandardizedArea,0.5,0.2),
        M2== 1 - ?fuzzy_metrics(V,0.225,0.09),
        M1 * M2 >= 0.25,!.
fuzzy_metrics(X,T,H) = 1.0 :-
        X >= T + H,!.
fuzzy_metrics(X,T,H) = 0.0 :-
        X <= T - H,!.
fuzzy_metrics(X,T,H) = V :-
        V== (X-T+H) * (1 / (2*H)).
report_target_event(Duration):-
        Duration > 2.0,!,
        graphic_window ? set_text_alignment('CENTER','CENTER'),
        graphic_window ? set_font({size:48,weight:'WEIGHT_BOLD'}),
        graphic_window ? set_pen({color:'Red'}),
```

```
        graphic_window ? draw_text(
                0.5,0.25,
                "Сотрудник АЭС уснул или потерял сознание!").
    report_target_event(_).
```

Аналогично тому, как работает предикат *draw_origins*, предикат *check_graphs* разматывает список графов, которые затем посылает в виде аргумента в предикат *check_graph*. Предикат *check_graph*, в свою очередь, разматывает граф, представленный в виде дуг графа. Последовательность блобов дуги посылается в обработку предиката *there_is_dreaming_person*. Предикат *there_is_dreaming_person* регистрирует факт неподвижности сотрудника за счёт оценки стандартизированной площади и скорости блоба при помощи функции *fuzzy_metrics*. В программе представлено пять правил *there_is_dreaming_person*, которые содержат следующие аргументы: список координат блобов *TrackOfBlob*, принадлежащих дуге; переменная *IsLying*, имеющая одно из двух значений (*'yes'* или *'no'*); номер первого кадра, на котором человек уже лежит, в качестве входного аргумента; номер первого кадра, на котором человек уже лежит, в качестве выходного аргумента; номер последнего кадра, на котором человек лежит. Так как вначале человек движется, программа исполнит пятое правило *there_is_dreaming_person* (поскольку это единственное правило, которое успешно пройдёт унификацию аргументов в заголовке). Это правило будет выполняться до тех пор, пока не подтвердится факт неподвижности человека. В этом случае программа выполнит третье правило, в котором логическая переменная *IsLying* примет значение *'yes'*, что позволит перейти к выполнению четвёртого правила, в котором список координат блобов будет рекурсивно обрабатываться, и будет подтверждаться факт неподвижности человека до тех пор, пока в списке не останется один элемент. Тем самым, предикат *there_is_dreaming_person* завершится успехом только при выполнении первого или второго правила. Затем будет вычислено время, в течение которого сотрудник был неподвижен (*Duration*), и программа перейдёт к

выполнению предиката *report_target_event*, который выведет на экран сообщение «Сотрудник АЭС уснул или потерял сознание!», если переменная *Duration* будет больше 2.0 (т.е. человек был неподвижен более двух секунд). Результат работы программы проиллюстрирован на рисунке 13.

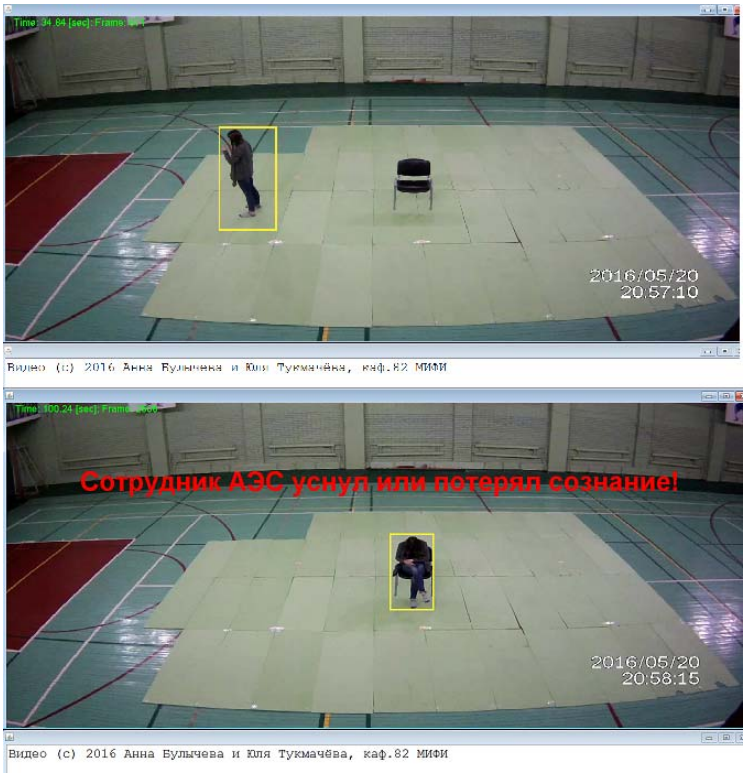

Рисунок 13. Результат работы программы, которая распознаёт уснувшего или потерявшего сознание сотрудника АЭС.

4.2 Тестирование разработанного метода

Проверка правильности работы алгоритма распознавания аномального поведения людей является достаточно сложной задачей, поскольку его работа зависит от нескольких взаимосвязанных процессов. В данной работе в качестве этих процессов можно выделить, как минимум, два:

1. Распознавание движущихся объектов.
2. Поиск подграфов, соответствующих искомым схемам аномального поведения, в графе траекторий движения объектов.

В более сложных алгоритмах этих процессов, естественно, гораздо больше.

Простое тестирование алгоритмов распознавания, а именно, подсчёт правильно и неправильно распознанных тестовых образцов является совершенно недостаточным, потому что не даёт представления о том, как небольшие изменения условий видеонаблюдения, ухудшающие количественные характеристики работы методов низкого уровня, могут повлиять на качество решения задач более высокого уровня. Поэтому для полноценного тестирования алгоритмов интеллектуального видеонаблюдения необходимо [**8**]:

1. Создание тестовых видеоклипов, отражающих реальные условия видеонаблюдения, включая возможные ракурсы съёмки, возможные изменения освещения и фона, состав наблюдаемых объектов и пр.
2. Всестороннее тестирование отдельных этапов обработки видеоданных с пониманием того, какие параметры одного этапа обработки существенно влияют на реализацию последующих этапов.
3. Совместное тестирование всех этапов анализа, включая проверку производительности системы при различных уровнях нагрузки.

Тестирование программы по указанным выше параметрам с созданием всех необходимых условий является трудоёмкой и занимающей много времени задачей. Поэтому в настоящей работе для начальной оценки правильности работы алгоритма был проведён подсчёт количества правильно и неправильно распознанных ситуаций. Причинами сбоев в работе программы, анализирующей видеоизображение в реальном времени, могут быть всевозможные факторы, влияющие на производительность компьютера, а именно, замедление работы компьютера из-за фоновых программ Windows, фрагментация памяти, неравномерная работа сборщика мусора Java и др. Программа, регистрирующая пультовую комнату, оставленную без присмотра, была запущена 30 раз. Во время выполнения программа ни разу не дала сбоя. Поэтому, на начальном этапе разработки, алгоритм можно считать корректным.

Заключение

Настоящая работа включает в себя несколько этапов исследования. Первый из них представляет собой исследование работы алгоритмов, выполняющих низкоуровневую обработку видео. Во время исследования, во-первых, проверялся алгоритм распознавания движущихся объектов (вычитание фона), в результате которого выяснилось, что алгоритм чувствителен к изменению освещения, а также к глобальному смещению кадра. Во-вторых, был изучен алгоритм, вычисляющий скорость движущегося объекта. Была исследована зависимость качества работы алгоритма от изменения некоторых параметров (направление движения относительно камеры, разрешение видеоизображения, наличие фильтра, устраняющего шумы во время работы программы). По итогам исследования были сделаны количественные оценки работы данного алгоритма, которые в рамках настоящей работы подтверждают его корректность. Однако полученные оценки носят неоднозначный характер, т.к. не удалось проследить систематическую зависимость алгоритма от параметров, для этого требуется провести более глубокий анализ.

Чтобы оценить новизну представленной работы, а также то, насколько интеллектуальное видеонаблюдение внедрено в атомную отрасль, был проведён анализ существующих в мире проектов. Оказалось, что подобные работы уже осуществляются, однако все они находятся лишь на начальной стадии, разработчики выполняют только лишь низкоуровневую обработку видеоизображений.

Для оценки актуальности разрабатываемого метода распознавания аномального поведения на АЭС были изучены правила техники безопасности в данной области. При изучении документации были смоделированы ситуации, иллюстрирующие аномальное поведение персонала АЭС в контексте интеллектуального видеонаблюдения.

После проведённых исследований был разработан метод интеллектуального видеонаблюдения на примере логической программы,

регистрирующей оставленную без присмотра пультовую комнату. Программа была протестирована путём подсчёта количества правильно и неправильно распознанных ситуаций. По результатам 30 тестовых запусков не было обнаружено ни одного ошибочного срабатывания программы.

По результатам настоящей выпускной квалификационной работы, все поставленные задачи выполнены успешно. Процесс контроля соблюдения персоналом правил техники безопасности автоматизирован с помощью метода интеллектуального видеонаблюдения аномального поведения персонала АЭС посредством логического языка Акторный Пролог. Как было подтверждено экспериментальным тестированием программы, алгоритм справляется с поставленной задачей успешно, т.е. распознаёт пустую комнату или регистрирует неподвижного человека, что говорит об успешном достижении целей настоящей работы.

Список литературы

1. Singh G.K., Shukla V., Patil S., Shah P. Automatic detection of abnormal event using smart video surveillance system in a nuclear power plant // 55th Annual Meeting of the Institute of Nuclear Materials Management – Atlanta, USA: Institute for Nuclear Materials and Management, 2014. – Vol. 1. – pp. 3139-3146.
2. Петянчина И.В. Система промышленного телевидения на атомной электростанции // Международный научно-исследовательский журнал. – 2014. – Т. 25, № 6-1. – с. 67-68.
3. Jorge C.A.F., Seixas J.M., Silva E.A.B., Mól A.C.A., Cota R.E., Ramos B.L. People detection in nuclear plants by video processing for safety purpose // International Nuclear Atlantic Conference (INAC 2011). – Belo Horizonte, MG, Brazil: 2011. – Vol. 42. – pp. 15-30.
4. Morozov A.A. Development of a Method for Intelligent Video Monitoring of Abnormal Behavior of People Based on Parallel Object-Oriented Logic Programming // Pattern Recognition and Image Analysis. – 2015. – Vol. 25, No. 3. – pp. 481-492.
5. 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 platform for the intelligent monitoring of anomalous human activities // Biomedical Engineering Systems and Technologies. 7th International Joint Conference, BIOSTEC 2014, Angers, France, March 3-6, 2014, Revised Selected Papers / Ed. by Guy Plantier, Tanja Schultz, Ana Fred, and Hugo Gamboa. – CCIS 511. – Springer International Publishing, 2015. – pp. 82-97.
6. Morozov A.A. A GitHub repository containing source codes of Actor Prolog built-in classes. – 2016. – https://github.com/Morozov2012/actor-prolog-java-library/ .

7. Инструкция по безопасности труда. Инструкция по радиационной технике безопасности в комплексе ИРТ МИФИ № 609Р.07.14-И-06 – Москва: НИЯУ МИФИ, 2014.
8. Морозов А.А., Сушкова О.С. Анализ видеоизображений в реальном времени средствами языка Акторный Пролог // Международная конференция и молодёжная школа «Информационные технологии и нанотехнологии» (ИТНТ-2016), 17-19 мая 2016 года, Самара, Россия. – Самара: СГАУ & ИСОИ, 2016. – с. 350-356.

Приложение А. Реперные точки и обратные матрицы проективных преобразований

Версия видео № 4 (пол застелен зелёными матами)

Разрешение видео 1280x720 px:

Координаты отмеченных точек в (м/с) и px:

[4.43, 0; 0, 0; 4.43, 2.385; 2.515, 2.41; 0, 2.38], (м/с)

[503, 356; 954, 353; 465, 501; 719, 499; 1039, 489], px

Обратная матрица проективных преобразований:

[[0.0328, 0.0103, 0.0003],

[0.0430, -0.0669, 0.0022],

[-10.0751, 5.9672, 1.0000]].

Разрешение видео 640x480 px:

Координаты отмеченных точек в (м/с) и px:

[4.43, 0; 0, 0; 4.43, 2.385; 2.515, 2.41; 0, 2.38], (м/с)

[252, 237; 478, 236; 233, 333; 359, 333; 520, 328], px

Обратная матрица проективных преобразований:

[[-0.0567, 0.0005, -0.0003],

[0.0257, 0.0921, 0.0083],

[21.0432, -21.9583, 1.0000]].

Версия видео № 5 (пол застелен чёрными матами)

Разрешение видео 1280x720 px:

Координаты отмеченных точек в (м/с) и px:

[0, 0; 2.99, 0; 6.36, 0; 0, 2.10; 2.99, 2.10; 6.36, 2.10; 0, 4.10; 2.99, 4.10; 6.36, 4.10], (м/с)

[904, 270; 628, 265; 320, 266; 956, 364; 627, 360; 260, 356; 1023, 496; 623, 502; 178, 485], px

Обратная матрица проективных преобразований:

[[-0.0231, -0.0005, -0.0000],

[0.0121, 0.0555, 0.0042],

[17.6322, -14.4389, 1.0000]].

Разрешение видео 640x480 px:

Координаты отмеченных точек в (м/с) и px:

[0, 0; 2.99, 0; 6.36, 0; 0, 2.10; 2.99, 2.10; 6.36, 2.10; 0, 4.10; 2.99, 4.10; 6.36, 4.10], (м/с)

[452, 180; 314, 177; 161, 177; 478, 243; 314, 240; 130, 238; 512, 331; 312, 335; 90, 332], px

Обратная матрица проективных преобразований:

[-0.0446, -0.0007, -0.0001],

[0.0177, 0.0780, 0.0060],

[16.9544, -13.5901, 1.0000].

Приложение Б. Результаты тестирования алгоритма оценки скорости движения

Таблица вычисленных логической программой значений скоростей для анализа видео, снятого в помещении. Значения в таблице представлены в «м/с».

Таблица Б.1

Высокое разрешение								**Низкое разрешение**							
Бег				**Шаг**				**Бег**				**Шаг**			
Параллельно		**По диагонали**		**Параллельно**		**По диагонали**		**Параллельно**		**По диагонали**		**Параллельно**		**По диагонали**	
МФ	**Без МФ**	**МФ**	**Без МФ**	**МФ**	**Без МФ**	**МФ**	**Без МФ**	**МФ**	**Без МФ**	**МФ**	**Без МФ**	**МФ**	**Без МФ**	**МФ**	**Без МФ**
		5,5	4,5	1,4	1,5					3,4	3,7	0,6	1,05		
		3,4	3,5	0,6	0,6					2,9	2,6	0,2	0,6		
1,4	1,4	2,2	1,9	0,5	0,6	0,8	0,9	1,03	1,3	0,9	1,13	0,2	0,5	0,2	0,4

Приложение В. Исходные тексты логической программы

Код логической программы, выявляющей отсутствие оператора АЭС на рабочем месте (пультовую комнату, оставленную без присмотра).

Файл с расширением a (.a).

```
import .. from "morozov/Java2D";
import .. from "morozov/Vision";
class 'Main' (specialized 'Timer'):
constant:
     sampling_rate       = 25;
     circle_radius       = 0.01;
internal:
     subtractor = ('ImageSubtractor',
                   extract_blobs= 'yes',
                   track_blobs= 'yes',
                   use_grayscale_colors= 'yes',
                   background_standard_deviation_factor= 1.2,
                   minimal_blob_size= 10000);
     graphic_window = ('Canvas2D',
                   background_color= 'SystemControl',
                   y= 0,
                   height= 22);
     prompt_window = ('Report',
                   font_size= 24,
                   y= 22,
                   height= 3);
     text          = ('Text');
     image         = ('BufferedImage');
     state         = ('ProgramState');
[
goal:-!,
     graphic_window ? show,
     prompt_window ? write(
          "Видео (с) 2016 ",
```

```
            "Анна Булычева и Юлия Тукмачёва, "
            "каф.82 НИЯУ МИФИ"),
        Time0== ?milliseconds(),
        state ? set_beginning_time(Time0),
        set_period((1/sampling_rate/2),0),
        activate.
tick:-
        T2== ?milliseconds(),
        state ? get_beginning_time(T1),
        Delta== (T2 - T1) / 1000.0 * sampling_rate,
        N== ?convert_to_integer(?round(Delta)),!,
        load_figure(N,T2).
load_figure(N2,_):-
        state ? get_current_frame(N1),
        N1 == N2,!.
load_figure(N,_):-
        state ? set_current_frame(N),
        ImageToBeLoaded==
            "E:/видео для ВКР/версия 3/кадры/ Empty_room/” +
            " Empty_room " +
            text?format("%04d",N) + ".jpg",
        image ? does_exist(ImageToBeLoaded),!,
        image ? load(ImageToBeLoaded),
        subtractor ? subtract(N,image),
        draw_scene.
load_figure(_,T2):-
        state ? set_beginning_time(T2),
        subtractor ? reset_results.
draw_scene:-
        subtractor ? commit,
        subtractor ? get_recent_frame_number(F),
        graphic_window ? suspend_redrawing,
        graphic_window ? clear,
        graphic_window ? set_font({size:18}),
        graphic_window ? set_text_alignment('CENTER','CENTER'),
```

```
     subtractor ? get_recent_image(image),
     graphic_window ? draw_image(image,0,0,1,1),
     subtractor ? get_connected_graphs(Graphs),
     image ? get_size_in_pixels(IW,IH),
     draw_graphs(IW,IH,Graphs,F),
     check_graphs(Graphs),
     graphic_window ? draw_now.
draw_graphs(IW,IH,[Graph|Rest],F):-!,
     draw_graph(IW,IH,Graph,Graph,F),
     draw_graphs(IW,IH,Rest,F).
draw_graphs(_,_,_,_).
draw_graph(IW,IH,[Edge|Rest],Graph,F):-!,
     draw_edge(IW,IH,Edge,Graph,F),
     draw_graph(IW,IH,Rest,Graph,F).
draw_graph(_,_,_,_,_).
draw_edge(     IW,IH,
          {    x1:X1a,y1:Y1a,x2:X2a,y2:Y2a,
               inputs: Origins,
               coordinates: TrackOfBlobs|_},
          Graph,F):-
     X1r== X1a / IW,
     Y1r== Y1a / IH,
     X2r== X2a / IW,
     Y2r== Y2a / IH,
     graphic_window ? set_brush('Blue'),
     draw_origins(X1a,Y1a,IW,IH,Origins,Graph),
     graphic_window ? set_pen({color:'Magenta',lineWidth:2}),
     draw_track_of_blobs(IW,IH,TrackOfBlobs),
     draw_rectangle(TrackOfBlobs,IW,IH,F),
     graphic_window ? set_brush('on'),
     graphic_window ? set_pen({color:'Green',lineWidth:1}),
     graphic_window ? draw_line(X1r,Y1r,X2r,Y2r),
     draw_circle(X1r,Y1r),
     draw_circle(X2r,Y2r),
     fail.
```

```
draw_edge(_,_,_,_,_).
draw_origins(Sx,Sy,IW,IH,[Origin|Rest],Graph):-!,
     draw_origin(Sx,Sy,IW,IH,Origin,Graph),
     draw_origins(Sx,Sy,IW,IH,Rest,Graph).
draw_origins(_,_,_,_,_,_).
draw_origin(SxA,SyA,IW,IH,N,Graph):-
     get_edge(N,Graph,Edge),
     Edge == {x2:Nx2a,y2:Ny2a|_},!,
     draw_origin_arrow(SxA,SyA,Nx2a,Ny2a,IW,IH).
draw_origin(_,_,_,_,_,_).
draw_origin_arrow(X,Y,X,Y,_,_):-!.
draw_origin_arrow(SxA,SyA,Nx2a,Ny2a,IW,IH):-
     SxR== SxA / IW,
     SyR== SyA / IH,
     Nx2r== Nx2a / IW,
     Ny2r== Ny2a / IH,
     graphic_window ? draw_line(SxR,SyR,Nx2r,Ny2r).
get_edge(1,[Edge|_],Edge):-!.
get_edge(N,[_|Rest],Edge):-
     N > 0,
     get_edge(N-1,Rest,Edge).
draw_track_of_blobs(IW,IH,[Blob|Rest]):-
     Blob== {x:Xa,y:Ya|_},!,
     Xr== Xa / IW,
     Yr== Ya / IH,
     draw_track_of_blobs(IW,IH,Xr,Yr,Rest).
draw_track_of_blobs(_,_,_).
draw_track_of_blobs(IW,IH,X1,Y1,[Blob|Rest]):-
     Blob== {x:Xa,y:Ya|_},!,
     X2== Xa / IW,
     Y2== Ya / IH,
     graphic_window ? draw_line(X1,Y1,X2,Y2),
     draw_track_of_blobs(IW,IH,X2,Y2,Rest).
draw_track_of_blobs(_,_,_,_,_).
draw_circle(X0,Y0):-
```

```
     X1== X0 - circle_radius,
     Y1== Y0 - circle_radius,
     Width== circle_radius * 2,
     Height== circle_radius * 2,
     graphic_window ? draw_ellipse(X1,Y1,Width,Height).
draw_rectangle([Blob],IW,IH,F):-
     Blob == {frame:F,x:X0,y:Y0,width:W1,height:H1|_},!,
     graphic_window ? set_brush('off'),
     graphic_window ? set_pen({color:'Yellow',lineWidth:3}),
     X2== (X0 - W1 / 2) / IW,
     Y2== (Y0 - H1 / 2) / IH,
     W2== W1 / IW,
     H2== H1 / IH,
     graphic_window ? draw_rectangle(X2,Y2,W2,H2).
draw_rectangle([_|R],IW,IH,N):-!,
     draw_rectangle(R,IW,IH,N).
draw_rectangle(_,_,_,_).
------------------------------------------------------------------
-- Проверка отсутствия людей в пультовой комнате              --
------------------------------------------------------------------
check_graphs([]):-!,
     report_target_event.
check_graphs(_).
report_target_event:-
     graphic_window ? set_text_alignment('CENTER','CENTER'),
     graphic_window ? set_font({size:48}),
     graphic_window ? set_pen({color:'Red'}),
     graphic_window ? draw_text(
          0.5,0.2,
          "Пультовая комната без присмотра!").
]
------------------------------------------------------------------
class 'ProgramState' (specialized 'Database'):
[
CLAUSES:
```

```
get_beginning_time(T):-
     Item== ?match(beginning_time(_)),
     Item == beginning_time(T),!.
set_beginning_time(T):-
     retract_all(beginning_time(_)),
     append(beginning_time(T)).
get_current_frame(N):-
     Item== ?match(current_frame(_)),
     Item == current_frame(N),!.
set_current_frame(N):-
     retract_all(current_frame(_)),
     append(current_frame(N)).
]
```

Файл с расширением def (.def).

```
pragma: INTERMEDIATE_SOURCE_CODE = JDK;
pragma: USE_COMPILER_INSTEAD_OF_PROVER = ON;
import .. from "morozov/Java2D";
import .. from "morozov/Vision";
interface 'Main' (specialized 'Timer'):
constant:
     sampling_rate       : INTEGER;
     circle_radius       : REAL;
internal:
     subtractor          : 'ImageSubtractor';
     graphic_window      : 'Canvas2D';
     prompt_window       : 'Report';
     text                : 'Text';
     image               : 'BufferedImage';
     state               : 'ProgramState';
[
PREDICATES:
imperative:
load_figure(INTEGER,INTEGER)            - (i,i);
draw_scene;
draw_graphs(
```

```
     PointX,
     PointY,
     GraphList,
     FrameNumber)        - (i,i,i,i);
draw_graph(
     PointX,
     PointY,
     ConnectedGraph,
     ConnectedGraph,
     FrameNumber)        - (i,i,i,i,i);
draw_edge(
     PointX,
     PointY,
     ConnectedGraphEdge,
     ConnectedGraph,
     FrameNumber)        - (i,i,i,i,i);
draw_origins(
     PointX,
     PointY,
     PointX,
     PointY,
     EdgeNumbers,
     ConnectedGraph)     - (i,i,i,i,i,i);
draw_origin(
     PointX,
     PointY,
     PointX,
     PointY,
     EdgeNumber,
     ConnectedGraph)     - (i,i,i,i,i,i);
draw_origin_arrow(
     PointX,
     PointY,
     PointX,
     PointY,
```

```
     PointX,
     PointY)             - (i,i,i,i,i,i);
determ:
get_edge(
     INTEGER,
     ConnectedGraph,
     ConnectedGraphEdge) - (i,i,o);
imperative:
draw_track_of_blobs(
     PointX,
     PointY,
     TrackOfBlob)        - (i,i,i);
draw_track_of_blobs(
     PointX,
     PointY,
     PointX,
     PointY,
     TrackOfBlob)            - (i,i,i,i,i);
draw_circle(PointX,PointY)   - (i,i);
draw_rectangle(
     TrackOfBlob,
     PointX,
     PointY,
     FrameNumber)            - (i,i,i,i);
check_graphs(GraphList)      - (i);
imperative:
report_target_event;
PREDICATES:
imperative:
'+'(PointX,PointX) = PointX  - (i,i);
'+'(PointY,PointY) = PointY  - (i,i);
'-'(PointX,PointX) = PointX  - (i,i);
'-'(PointY,PointY) = PointY  - (i,i);
'/'(PointX,PointX) = PointX  - (i,i);
'/'(PointY,PointY) = PointY  - (i,i);
```

```
]
interface 'ProgramState' (specialized 'Database'):
[
DOMAINS:
Target         = beginning_time(INTEGER);
                 current_frame(INTEGER).
PREDICATES:
determ:
get_beginning_time(INTEGER)        - (o);
get_current_frame(INTEGER)         - (o);
imperative:
set_beginning_time(INTEGER)        - (i);
set_current_frame(INTEGER)         - (i);
]
```