

Общество с ограниченной ответственностью «Цифровые решения»

УДК: 621.397

Регистрационный №АААА-А19-119042390070-0

Инв. №02-2019

УТВЕРЖДАЮ
Генеральный директор
_____ Матвеев Д.В.
" ____ " _____ 20__ г.
М.П.

ОТЧЕТ

о выполнении НИОКР по теме:

"Доработка алгоритмов трекинга и детектирования на основе сверточной нейронной сети по результатам испытаний. Разработка прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий. Натурные испытания прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий. Доработка по результатам испытаний прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий"

(договор 2962ГС1/45326 от 01.04.2019)

(заключительный)

Научный руководитель работ

_____ Хрящев В.В.
подпись, дата

Ярославль, 2020

1. Реферат

Отчет 91 стр., 22 илл., 2 табл., 2 приложения, 40 источников

детектирование объектов, сопровождение объектов, машинное обучение, компьютерное зрение, нейронные сети

Цель работы на третьем этапе выполнения НИОКР – доработка алгоритмов детектирования и отслеживания объектов на глубокого машинного обучения, а также разработка прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий. К полученным результатам относится:

- По результатам испытаний был доработан алгоритм трекинга и детектирования на основе сверточной нейронной сети.
- Разработан программно-аппаратный комплекс для видеоанализа спортивных мероприятий.
- Проведены испытания алгоритма детектирования и трекинга, а также разработанного прототипа программно-аппаратного комплекса.
- По результатам испытания был доработан прототип программно-аппаратного комплекса для видеоанализа спортивных мероприятий.

Областями применения разработанных алгоритмов являются комплексы спортивной видеоаналитики, которые будут использованы спортивными клубами, многофункциональными спортивными комплексами.

Работы по данному этапу НИОКР были выполнены в полном объеме в соответствии с техническим заданием и календарным планом. Полученные результаты будут использованы на следующих этапах НИОКР и в коммерческой деятельности ООО «Цифровые решения».

СПИСОК ИСПОЛНИТЕЛЕЙ

Должность	Подпись	ФИО, номер раздела
Генеральный директор, ООО «Цифровые решения», к.т.н.		Матвеев Д.В. (подразделы 3.1 – 3.4)
Научный руководитель работ, к.т.н.		Хрящев В.В. (подразделы 3.1 – 3.4)
Программист ООО «Цифровые решения»		Казина Е.М. (подразделы 3.1 – 3.2)
Консультант ООО «Цифровые решения»		Федькина А.А. (подразделы 3.3 – 3.4)

СОДЕРЖАНИЕ

1. Реферат	2
2. Введение	5
3. Основная часть	8
3.1. Доработка алгоритмов трекинга и детектирования на основе сверточной нейронной сети по результатам испытаний	8
3.2. Разработка прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий	26
3.3. Натурные испытания прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий	29
3.4. Доработка по результатам испытаний прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий	32
4. Заключение	36
5. Список использованных источников	38
Приложения	42
Приложение А. Фрагменты кода программы для спортивных аналитических систем на основе технологий машинного обучения	42
Приложение Б. Описание программы	88

2. ВВЕДЕНИЕ

На сегодняшний день, в индустрии спорта важнейшую роль играет статистика об игроках, собранная во время проведения соревнований. Статистическая информация полезна как для тренеров, для оценки работоспособности игроков и формирования тренировочного процесса, так и для скаутов, целью которых является привлечение необходимых спортсменов в команду [1]. И хотя физические возможности каждого игрока могут быть предварительно получены с помощью специально проведенных исследований, эффективность действий конкретного спортсмена, а также всей команды в целом можно оценить лишь во время матча.

В области компьютерного зрения одним из ключевых направлений исследований является создание алгоритмов обнаружения и отслеживания объектов на видеоданных. Под отслеживанием объекта здесь понимается вычисление местоположения объекта на определенном кадре видеопоследовательности [2]. Такая информация позволяет получить данные о суммарном пробеге игрока, количестве взаимодействий с другими спортсменами, а также построить тепловую карту перемещений, по сути отражающую зоны игровой площадки, в которых чаще всего находился спортсмен. Это является фундаментом для оценки действий конкретных игроков и для проведения дальнейшего анализа более высокого уровня, например, для вычисления коэффициента xG-Plot, построения карты перепасовок игроков или определения ролей спортсменов в команде [3, 4]. Помимо этого, собранная информация позволит тренерам скорректировать процесс тренировок игроков и разработать подходящие наступательные/оборонительные стратегии для достижения в перспективе наилучших спортивных результатов.

В то же время, обнаружение и отслеживание игрока – довольно сложная задача компьютерного зрения и искусственного интеллекта. Основными трудностями здесь являются:

- Динамичность спортивного мероприятия, а именно высокая скорость передвижения игроков по площадке.
- Взаимодействие игрока с другими спортсменами на площадке. Возникают ситуации, когда игроки могут находиться очень близко друг к другу.
- Наличие оптических препятствий на кадрах видеопоследовательности (засветы, тени от игроков, искажения кадров, полученных с камеры видеонаблюдения и проч.).

Основная причина состоит в том, что баскетбол – очень динамичный вид спорта, в котором игроки постоянно меняют свое положение и позы.

В данной НИОКР предлагается новый подход к обнаружению игроков, определению их позиций на игровой площадке, а также подсчету статистики по спортсменам (суммарный пробег, количество взаимодействий с другими игроками, тепловая карта перемещений) для видеоданных со спортивных мероприятий, таких как баскетбол, волейбол и мини-футбол, полученных при съемке с купольных камер. Разработанный программно-аппаратный комплекс позволяет не только довольно точно собрать статистическую информацию о спортсменах, но и наглядно представить ее и отослать на электронную почту. Предлагаемая система обнаружения и отслеживания игроков основана на использовании цифровой купольной камеры, установленной на стене спортивного зала Ярославского государственного университета им. П.Г. Демидова. Такое местоположение камеры позволяет получить изображение сразу всей спортивной площадки. Для обнаружения и отслеживания спортсменов на видеоданных со спортивных мероприятий был использован и модернизирован новейший алгоритм глубокого обучения JDE (Joint Detection and Embedding), разработанный в 2019 г. [5].

На данном этапе НИОКР решаются следующие поставленные задачи:

1. Доработка алгоритмов трекинга и детектирования на основе сверточной нейронной сети по результатам испытаний.

2. Разработка прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий.
3. Натурные испытания прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий.
4. Доработка по результатам испытаний прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий.

Полученные результаты будут использованы на следующих этапах НИОКР и в коммерческой деятельности ООО «Цифровые решения».

3. ОСНОВНАЯ ЧАСТЬ

3.1. Доработка алгоритмов трекинга и детектирования на основе сверточной нейронной сети по результатам испытаний.

На сегодняшний день, задача обнаружения и отслеживания движущихся объектов на видеоданных остается одной из самых актуальных в области разработки искусственного интеллекта. Человек без особого труда способен замечать и отслеживать спортсменов во время трансляций матчей. Однако для компьютера детектирование и трекинг являются трудновыполнимой задачей. Её сложность определяется:

- разнообразием игроков,
- быстрым перемещениям спортсменов по игровой площадке,
- наличием оптических препятствий.

Ежегодно среди аналитиков данных и специалистов по data science проводятся различные конкурсы, целью которых является продвижение исследований в соответствующей области машинного обучения и получение совершенно новых алгоритмов детектирования и трекинга, выполняющих поставленные перед ними задачи в режиме реального времени. Вот лишь небольшой список этих конкурсов:

1. MOT Challenge – Multiple Object Tracking Challenge [6].

Ежегодное соревнование, проводимое с 2015 г., по обнаружению и отслеживанию пешеходов на видеоданных, снятых на улицах и площадях городов с диагонального ракурса камеры высокого разрешения. Целью этого конкурса также является создание единого инструмента по оценке качества работы алгоритмов детектирования и трекинга нескольких объектов. Примеры кадров из видеоданных для конкурса MOT Challenge приведены на рисунке 1.



Рисунок 1 – Кадры из видеопоследовательностей для конкурса MOT Challenge

2. UA-DETRAC Challenge [7]

Соревнование, проводимое в 2018 г. для обнаружения и отслеживания всех целевых объектов, присутствующих на кадрах видеопоследовательностей, в данном случае транспортных средств. Набор данных состоит из 10 часов видео, снятых на камеру Canon EOS 550D в 24 различных местах Пекина и Тяньцзина в Китае при различных погодных условиях (облачно, солнечно, дождливо и при ночном наблюдении). Видеопоследовательности записываются с кадровой частотой 25 FPS и с разрешением каждого кадра 960×540 пикселей. В наборе данных UA-DETRAC содержится более 140 тыс. кадров и 8250 транспортных средств, которые были проаннотированы экспертами вручную. В общей сложности разметка видеоданных составила 1,21 млн выделенных ограничивающих рамок вокруг целевых объектов. Примеры кадров из видеоданных для конкурса UA-DETRAC Challenge приведены на рисунке 2.



Рисунок 2 – Кадры из видеопоследовательностей для конкурса UA-DETRAC Challenge

3. CVPR19 Tracking and Detection Challenge [8]

Конкурс, который прошел в 2019 г., во время конференции разработчиков искусственного интеллекта IEEE Conference on Computer Vision and Pattern Recognition в Калифорнии, США, с целью создания высокоточных моделей обнаружения и отслеживания пешеходов на улицах, а также в местах массового скопления людей. Набор данных состоит из 8 видеофайлов, на трех из которых были засняты многолюдные площади американских городов. На кадрах видеоданных могло одновременно находиться до 246 пешеходов. Все видеопоследовательности были засняты как в помещении, так и на открытой местности в дневное и ночное время суток. Примеры кадров из видеоданных для конкурса CVPR19 Tracking and Detection Challenge приведены на рисунке 3.



Рисунок 3 – Кадры из видеопоследовательностей для конкурса
CVPR19 Tracking and Detection Challenge

В системах обнаружения и отслеживания объектов на видеопоследовательностях, в общем случае реализуются следующие этапы работы:

1. Извлечение кадра из видеоданных
2. Предварительная обработка кадра (при необходимости):
снижение помех, удаление засветов, фильтрация, повышение четкости и проч.
3. Выделение рамок, в которых может содержаться целевой объект
4. Точное определение рамки объекта
5. Вывод и сохранение полученных результатов

Реализованный на втором этапе НИОКР алгоритм отслеживания спортсменов на видеопоследовательностях, снятых с вертикального ракурса купольной камеры показал приемлемый результат: точность обнаружения игроков на спортивной площадке составила порядка 82.2%. Однако для видеоданных, снятых с диагонального ракурса камеры, алгоритм детектирования и трекинга спортсменов с помощью двойного байесовского вывода, разработанный на первом этапе выполнения НИОКР, уже не демонстрировал такое высокое качество работы. Более того, этот алгоритм не позволял обнаруживать и отслеживать целевые объекты на видеоданных в

режиме реального времени. Таким образом, возникла необходимость модернизации данного алгоритма.

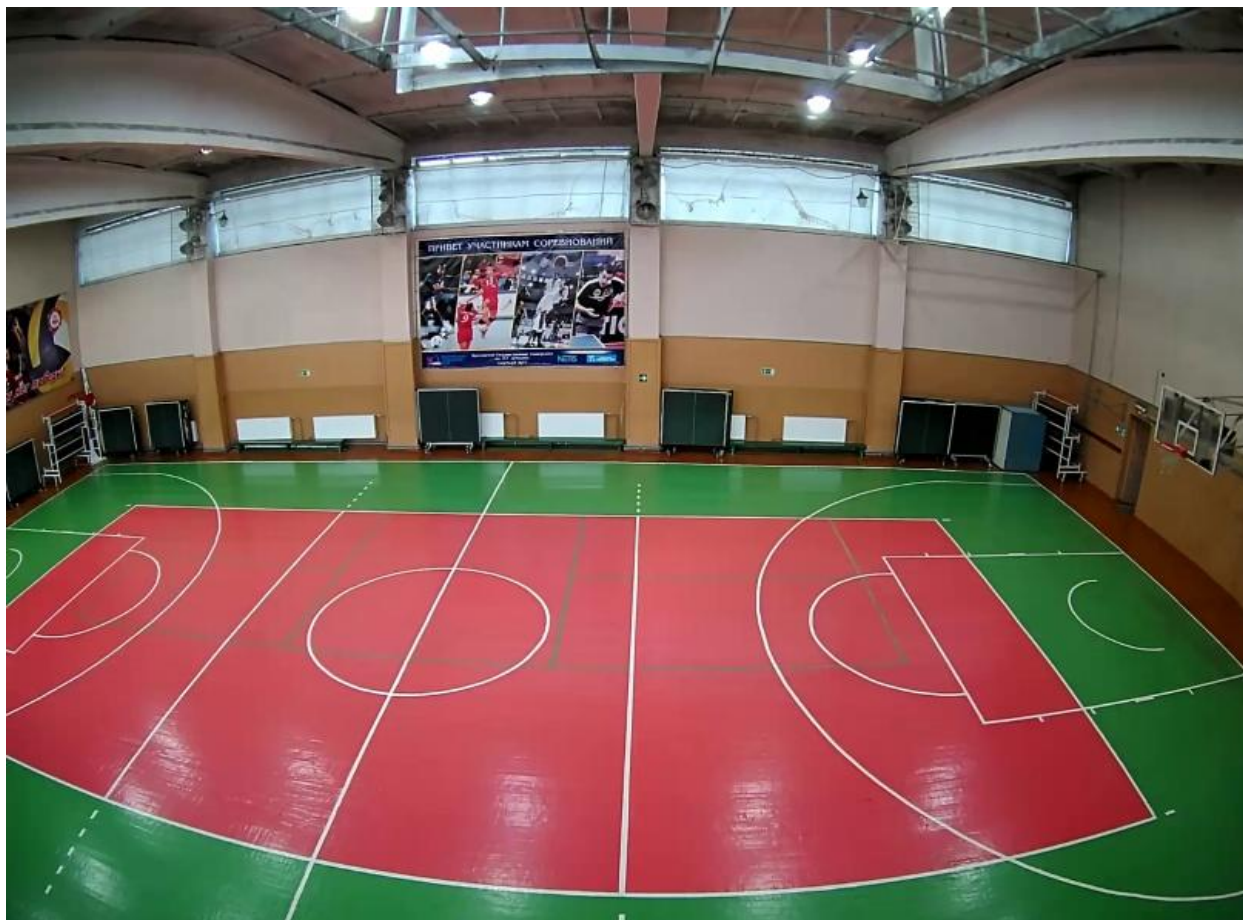


Рисунок 4 – Пример изображения на выходе купольной камеры

Для доработки алгоритма обнаружения и отслеживания спортсменов на игровой площадке, как и при реализации предыдущих двух этапов НИОКР, использовалась купольная камера GV-EVD3100, применяемая для съемок подобного рода мероприятий без угрозы выведения ее из строя в результате внешних физических воздействий, таких как попадание мячом. Размер светочувствительной CMOS-матрицы 1/2.8” позволял получать видеоданные в формате avi с кадровой частотой 30 FPS. Каждый кадр отснятой видеопоследовательности имеет высокое разрешение, порядка 2048×1536 пикселей. Регуляция положения объектива купольной камеры давало возможность разместить данный аппарат как на потолке спортивного зала, так и на стене. Пример изображения, полученного на выходе, с купольной камеры, показан на рисунке 4. Расположение купольной камеры

(диагональный ракурс) на стене спортивного зала Ярославского государственного университета им. П.Г. Демидова показано на рисунке 5.



Рисунок 5 – Расположение купольной камеры на стене спортивного зала ЯрГУ

Целью моделей обнаружения объектов определенного класса является поиск их точного расположения на изображении или кадре видеопоследовательности. В результате работы алгоритмов детектирования, целевые объекты обычно выделяются с помощью окантовочных прямоугольных рамок. Поскольку чаще всего перед разработчиками искусственного интеллекта ставится задача об обнаружении всех объектов одного или нескольких целевых классов, каждой выделенной прямоугольной рамке ставится в соответствие класс или порядковый номер выделенного объекта. В частности, в задачи обнаружения и отслеживания спортсменов на

игровой площадке, каждой выделенной прямоугольной рамке ставился в соответствие уникальный идентификатор – номер отмеченного спортсмена.

Для алгоритмов детектирования критерием обнаружения объекта служит величина IoU (intersection-over-union), а точнее говоря ее сравнение с некоторой заданной заранее пороговой величиной (threshold), которая, как и коэффициент IoU, принимает значение из отрезка $[0, 1]$. Значение IoU вычисляется, как отношение площади пересечения размеченной (detection box) и предсказанной (prediction box) окантовочных рамок вокруг целевого объекта к площади их объединения [9]. Наглядное представление данной формулы представлено на рисунке 6.

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}}$$

The diagram illustrates the calculation of the Intersection over Union (IoU) metric. It consists of two parts. The top part shows two overlapping rectangles: a white 'Prediction Box' and a white 'Detection Box'. The intersection of these two boxes is shaded red. The bottom part shows the same two rectangles, but the 'Prediction Box' and 'Detection Box' are blue, while the intersection remains red. Labels include (X_{11}, Y_{11}) for the top-left corner of the prediction box, (X_{21}, Y_{21}) for the bottom-left corner of the prediction box, (X_{12}, Y_{12}) for the bottom-left corner of the detection box, and (X_{22}, Y_{22}) for the bottom-right corner of the detection box.

Рисунок 6 – Наглядное представление формулы для величины IoU

Если коэффициент IoU превышал значение пороговой величины, то считалось, что было осуществлено корректное обнаружение данного объекта. Чем больше было значение пороговой величины, тем более точно детектор определял местоположение и размер окантовочной прямоугольной рамки вокруг целевого объекта, однако в то же самое время большее число верных обнаружений могло быть пропущено алгоритмом [10]. Таким образом, выбор

порогового значения является задачей очень тонкой настройки параметра в зависимости от дальнейшего использования алгоритма.

Помимо коэффициента IoU, для оценки качества работы алгоритма обнаружения объектов часто используются величины TP (true positive – количество верных срабатываний), FP (false positive – ошибки первого рода, количество ложных срабатываний), FN (false negative – ошибки второго рода, количество ложных пропусков), P (precision – точность модели), R (recall – полнота модели).

Отслеживание (трекинг) объекта – процесс определения местоположения движущегося объекта на видеоданных. Целью алгоритмов отслеживания объектов на видеоданных и является поиск схожих объектов, местоположение которых было определено ранее, на предыдущих кадрах. На выходе алгоритм отслеживания выдает набор прямоугольных рамок, окружающих интересующие объект, на каждом кадре видеопоследовательности. Основным отличием алгоритма отслеживания от модели детектирования является то, что после первого кадра, на котором был обнаружен целевой объект, на последующих кадрах алгоритму ничего не требуется знать об объекте, кроме его местоположения, полученного ранее. На новом кадре видеоданных поиск прямоугольной рамки, окружающей объект, осуществляется в окрестности ее местоположения на предыдущем кадре [11].

В отличие от модели отслеживания одного объекта алгоритм трекинга нескольких объектов направлен на прогнозирование траекторий перемещения все целевых объектов на видеопоследовательности. Однако такой модели в течение длительного времени требуется обрабатывать гораздо больше данных о местоположениях объектов, полученных при обработке предыдущих кадров [12].

Основные трудности, возникающие при разработке алгоритма отслеживания нескольких объектов:

1. Скорость работы, которая не всегда позволяет алгоритму трекинга отслеживать все целевые объекты в режиме реального времени.
2. Изменение внешнего вида объекта с течением времени (в результате динамики движения, изменения освещения, точки обзора и проч.).
3. Взаимодействие объектов на видеоданных (перекрытия, визуальная схожесть объектов).

Критерием оценки качества работы алгоритма отслеживания нескольких объектов на видеоданных служит коэффициент MOTA (multiple object tracking accuracy), который вычисляется по следующей формуле:

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + ID_switches_t)}{\sum_t GT_t},$$

где *ID_switches* - количество ошибок отслеживания объекта, когда алгоритм отслеживания вместо одной, истинной траектории передвижения объекта определяет как минимум ещё одну ложную, а GT (ground truth) – все правильно детектированные объекты на t-ом кадре. Ключевой особенностью коэффициента MOTA является то, что он напрямую соотносится с тем, как человеческий глаз отслеживает целевые объекты на видеоданных. Исходя из формулы коэффициента, величина MOTA «наказывает» потенциально идеальный алгоритм трекинга за ложные срабатывания (FP), ложные пропуски (FN), а также ошибки *ID_switches* на каждом t-ом кадре [13]. На практике, чаще всего величина MOTA, значение которой согласно формуле лежит в отрезке [0, 1], переводится в проценты.

Одним из современных подходов при решении задачи обнаружения и отслеживания объектов является использование методов глубокого машинного обучения на основе сверточных нейронных сетей. Сверточные сети нашли свое применение в области компьютерного зрения и биоинформатики. Несмотря на то, что такой алгоритм машинного обучения требует более тонкой настройки параметров, его главными преимуществами

являются совместное использование параметров разных типов, а также тот факт, что значения признаков вычисляются в результате применения формирующихся по ходу обучения сверточных фильтров. Согласно последним исследованиям, сгенерированные таким образом признаки, как правило, позволяют получить наилучшие результаты во многих задачах машинного обучения, а прогресс в области создания высокопроизводительных компьютеров позволил разработчикам эффективно обучать и тестировать нейросетевые алгоритмы, которые имеют миллионы параметров [14, 15].

Сверточные слои являются самыми важными строительными блоками глубоких нейронных сетей. При работе этого слоя набор признаков I , поступивших на вход с предыдущего слоя с помощью матричного умножения на различные окна свертки K фиксированного размера, преобразуется в другой, преобразованный набор признаков $I \times K$. Другими словами, происходит взвешенное суммирование значений интенсивности признаков по мере прохождения окна по всем данным. Тем самым, сверточные слои на самом деле применяют взаимную корреляцию, похожую на математическую операцию свертки [16]. Схема работы сверточного слоя показана на рисунке 7.

Современные системы отслеживания нескольких объектов обычно следуют парадигме tracking-by-detection («отслеживание по обнаружению») [17], включающей в себя последовательное выполнение двух этапов:

1. Этап детектирования объектов, на котором выделяются области местоположения объектов.
2. Этап ассоциации данных, на котором обнаруженным объектам ставятся в соответствие траектории, ранее построенные при детектировании и трекинге целевых объектов.

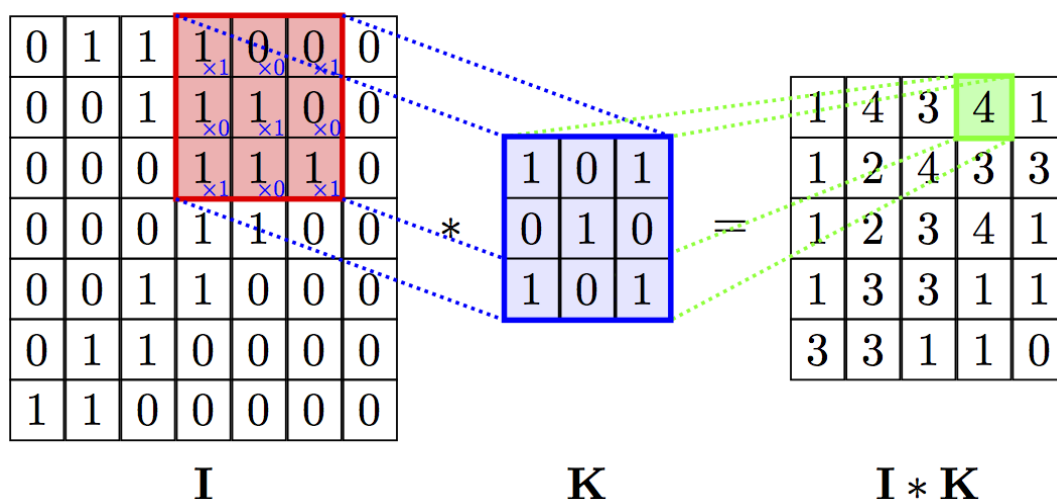


Рисунок 7 – Схема работы сверточного слоя

Следование парадигме tracking-by-detection означает, что система отслеживания нескольких объектов требует наличие двух отдельных вычислительных компонент: алгоритма обнаружения и модели встраивания (или повторной идентификации) объектов. Последовательное выполнение двух этих моделей может привести к проблемам с эффективностью и запуском алгоритма трекинга нескольких объектов в режиме реального времени, ведь общее время работы системы будет просто составлять сумму времен выполнения каждого этапа, поскольку на втором этапе алгоритм внедрения обнаруженных областей локализации не получает необходимой информации об ассоциации объектов.

Новые парадигмы алгоритмов трекинга предполагают, в частности, включать в этап детектирования этап ассоциации данных путем прогнозирования пространственного смещения каждого отслеживаемого объекта, как, например, в случае с алгоритмом Tracktor из [18]. Системы, следующие этой парадигме, привлекательны благодаря своей простоте, но при этом их точность отслеживания объектов не удовлетворительна без использования дополнительной модели внедрения, что фактически сводит работу системы отслеживания к выполнению тех же двух этапов, что и для алгоритмов, следующих парадигме tracking-by-detection.

Другим способом решения задачи отслеживания спортсменов на видеоданных является построение системы поиска людей, направленной на

определение местоположения и распознавание человека из большого набора фреймов (заготовок), как в [19]. Основное различие алгоритмами трекинга и системами поиска людей состоит в том, что первые предъявляют более строгие требования к времени выполнения (работа в режиме реального времени), и поэтому подходы к поиску людей из большого набора фреймов не могут быть заимствованы напрямую для решения задачи обнаружения и отслеживания спортсменов на игровой площадке.

Для экономии времени вычислений целесообразной является идея интеграции алгоритма детектирования и модели внедрения в единую сеть. Таким образом, две модели могут использовать один и тот же набор функций низкого уровня, а это значит, что повторное вычисление карт признаков исключается.

Одним из вариантов интеграции алгоритма детектирования и модели встраивания является использование структуры двуступенчатых детекторов, типа Faster R-CNN [20]. В частности, на первом этапе работы системы в качестве алгоритма выделения областей локализации объектов RPN (Region Proposal Network) используется классическая сверточная нейронная сеть Faster R-CNN, тогда как на втором этапе в качестве модели внедрения используется преобразованная сверточная нейронная сеть Fast R-CNN [21]. Несмотря на то, что в такой системе отслеживания модель внедрения использует некоторые вычисления, которые были выполнены на этапе детектирования объектов, алгоритм остается ограниченным по скорости своей работы из-за своей двухэтапной структуры организации. Он позволяет работать с видеоданными с кадровой частотой не более 10 FPS, что далеко от современных требований индустрии компьютерного зрения.

Алгоритм JDE (Joint Detection and Embedding – совместная модель по обнаружению и отслеживанию объектов) объединяет воедино оба шага этой парадигмы в единую, общую модель. В ней алгоритм внедрения обнаруженных областей локализации для ассоциации данных сопряжен с алгоритмом детектирования целевого объекта SSD (Single Shot Detector) [22]

таким образом, что система отслеживания может одновременно и выполнять детектирование и ассоциировать обнаруженные области расположения целевых объектов. Схема работы алгоритма JDE изображена на рисунке 8.

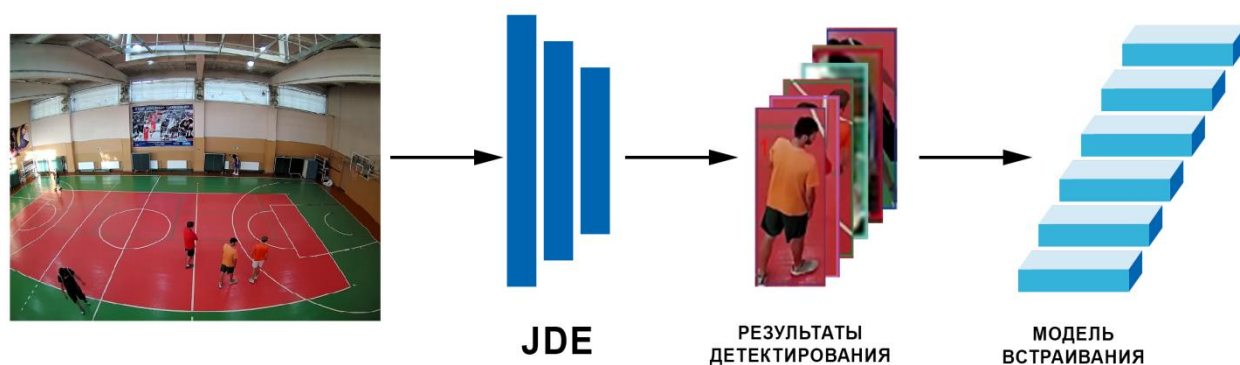


Рисунок 8 – Схема работы алгоритма JDE

Алгоритм JDE позволяет обнаруживать и отслеживать несколько объектов в режиме реального времени для видеоданных с кадровой частотой от 22 до 38 FPS, в зависимости от разрешения кадров видеопоследовательности, тогда как качество работы алгоритма сопоставимо с лучшими моделями обнаружения и отслеживания нескольких объектов, следующих парадигме tracking-by-detection. Значения коэффициента MOTA некоторых современных алгоритмов трекинга отражены в таблице 1.

Таблица 1 – Сравнение результатов систем отслеживания нескольких объектов, обученных на базе данных MOT16

Алгоритмы	Метрики качества	
	MOTA	FPS
Tracktor	54,4	8 – 22
Faster R-CNN+ Fast R-CNN	50,2	< 10
JDE	60,6	22 – 38

В качестве базовой архитектуры для алгоритма JDE используется одна из разновидностей полносвязных нейронных сетей (FCN – Fully Convolutional Network) [23] – пирамидальная сеть FPN (Feature Pyramid Network), структура которой приведена в статье [24]. Подобного рода сети используют пирамидальную архитектуру для выделения более сложных признаков и учета тех признаков, которые могли быть пропущены

сверточной нейронной сетью, при прохождении по ней информации. FPN делает прогнозы по нескольким шкалам, выделяя карту комплексных признаков и тем самым улучшая обнаружение необходимых объектов в том случае, когда их масштаб на кадрах видеопоследовательности сильно разнится. На рисунке 9 схематически показана структура пирамидальной сети, используемой в алгоритме JDE.

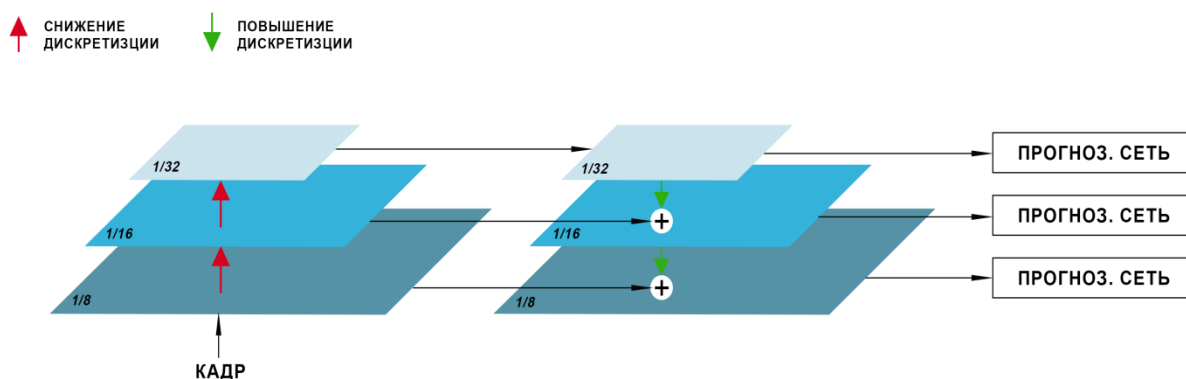


Рисунок 9 – Архитектура пирамидальной сети алгоритма JDE

Входной изображение (кадр видеопоследовательности) в начале подвергается прямому проходу через глубокую сеть для получения карт признаков в трех масштабах: с частотой дискретизации 1/32, 1/16, 1/8 соответственно. Далее карта признаков с наименьшим размером (также с наиболее выраженными объектами) подвергается дискретизации и объединяется с картой признаков из второй нейронной сети путем пропуска соединения (то же самое относится и к другим масштабам). Наконец, к объединенным картам признаков добавляются прогнозирующие сети (prediction head) во всех трех масштабах.

Прогнозирующая сеть состоит из нескольких уложенных друг на друга сверточных слоев и выводит плотную карту предсказания размером $(6A + D) \times H \times W$, где A – количество шаблонов якорей назначенных соответствующему масштабу ($A = 4$), D – показатель встраивания, а H и W – размеры изображения, поступившего на вход алгоритму JDE. Далее, осуществляется этап прогнозирования, при выполнении которого можно получить три набора данных:

1. Результаты классификации объектов (box classification results), находящихся в окантовочных прямоугольных рамках, размера $2A \times H \times W$. Они необходимы для определения типа выделенного объекта.
2. Коэффициенты регрессии (box regression coefficients) окантовочных прямоугольных рамок размера $4A \times H \times W$. Они необходимы для определения на изображении местоположения прямоугольных рамок.
3. Плотная карта вложения (dense embedding map) размера $D \times H \times W$.

Для всех якорей сети, необходимых для обнаружения объектов на снимке и их классификации, устанавливалось соотношение сторон потенциальных окантовочных прямоугольных рамок 1:3. Каждому якорю на изображении ставилось в соответствие 12 шаблонов, а ширина рамки якоря варьировалась в диапазоне от $11 \approx 8 \times 2^{1/2}$ до $512 = 8 \times 2^{12/2}$. Обучение алгоритма JDE сводилось к обнаружению нескольких объектов с помощью «якорной» классификаций, формированию вокруг каждого целевого объекта окантовочной прямоугольной рамки и внедрению обнаруженных областей для ассоциации с объектами, выделенными на предыдущих кадрах. Для оценки качества работы алгоритма JDE использовался коэффициент MOTA [13].

Реализация алгоритма JDE осуществлялась с помощью фреймворка PyTorch. PyTorch – библиотека машинного обучения, разработанная отделом искусственного интеллекта компании Facebook для проектов, написанных на языке программирования Python. Этот фреймворк используется для решения таких задач компьютерного зрения и обработки естественного языка. Фреймворк PyTorch имеет также большое количество встроенных модулей для предварительной обработки данных, поддержки сторонних библиотек, а также специальный пакет для поддержки методологии объектно-ориентированного программирования [25]. В библиотеке также есть

возможность запуска стандартных алгебраических и статистических операций над тензорами, единицами данных в PyTorch, как на центральном процессоре компьютера (CPU – Central Processing Unit), так и на графическом ускорителе видеокарты (GPU – Graphical Processing Unit).

Обучение и тестирование алгоритма JDE осуществлялось на наборе видеофайлов с конкурсов MOT-16 Challenge и MOT-17 Challenge [6]. Набор видеоданных с конкурса MOT-16 Challenge содержал обучающую и тестовую выборки, в каждой из которых было по шесть видеопоследовательностей. Каждый видеофайл имел кадровую частоту 14, 25 или 30 FPS, с разрешением кадров от 640×480 до 1920×1080 пикселей. Разметка каждого видеофайла была сохранена в одноименном текстовом файле. Суммарно объем обучающих и тестовых данных с конкурса MOT-16 Challenge составил порядка 1.9 Гб. Набор видеоданных с конкурса MOT-17 Challenge содержал обучающую и тестовую выборки, в каждой из которых было по 21 видеопоследовательности. Каждый видеофайл имел кадровую частоту 14, 25 или 30 FPS, с разрешением кадров от 640×480 до 1920×1080 пикселей. Разметка каждого видеофайла была сохранена в одноименном текстовом файле. Суммарно объем обучающих и тестовых данных с конкурса MOT-16 Challenge составил порядка 5.5 Гб.

Запуск процессов обучения и тестирования алгоритмов JDE осуществлялся на графическом процессоре суперкомпьютера NVIDIA DGX-1 с использованием технологии параллельных вычислений NVIDIA CUDA. Доступ к суперкомпьютеру предоставил индустриальный партнер ООО «Цифровые решения» – Ярославский государственный университет им. П.Г. Демидова. CUDA позволяет решить проблему высокой ресурсоемкости алгоритмов с помощью распараллеливания вычислений на графическом процессоре. Эта технология является кроссплатформенной и поддерживается всеми современными видеокартами NVIDIA [26, 27].

В результате обучения и тестирования на наборах данных с конкурсов MOT-16 Challenge и MOT-17 Challenge алгоритм JDE показал хорошие

результаты, сопоставимые с качеством работы современных алгоритмов обнаружения и отслеживания объектов: значение метрики MOTA составило порядка 60,6. Результаты детектирования и трекинга пешеходов на данных с конкурса MOT Challenge и игроков на видеофайлах со спортивных мероприятий, снятых с диагонального ракурса купольной камеры показаны на рисунках 10 и 11.



Рисунок 10 – Результаты обнаружения и отслеживания пешеходов
на данных с конкурса MOT Challenge

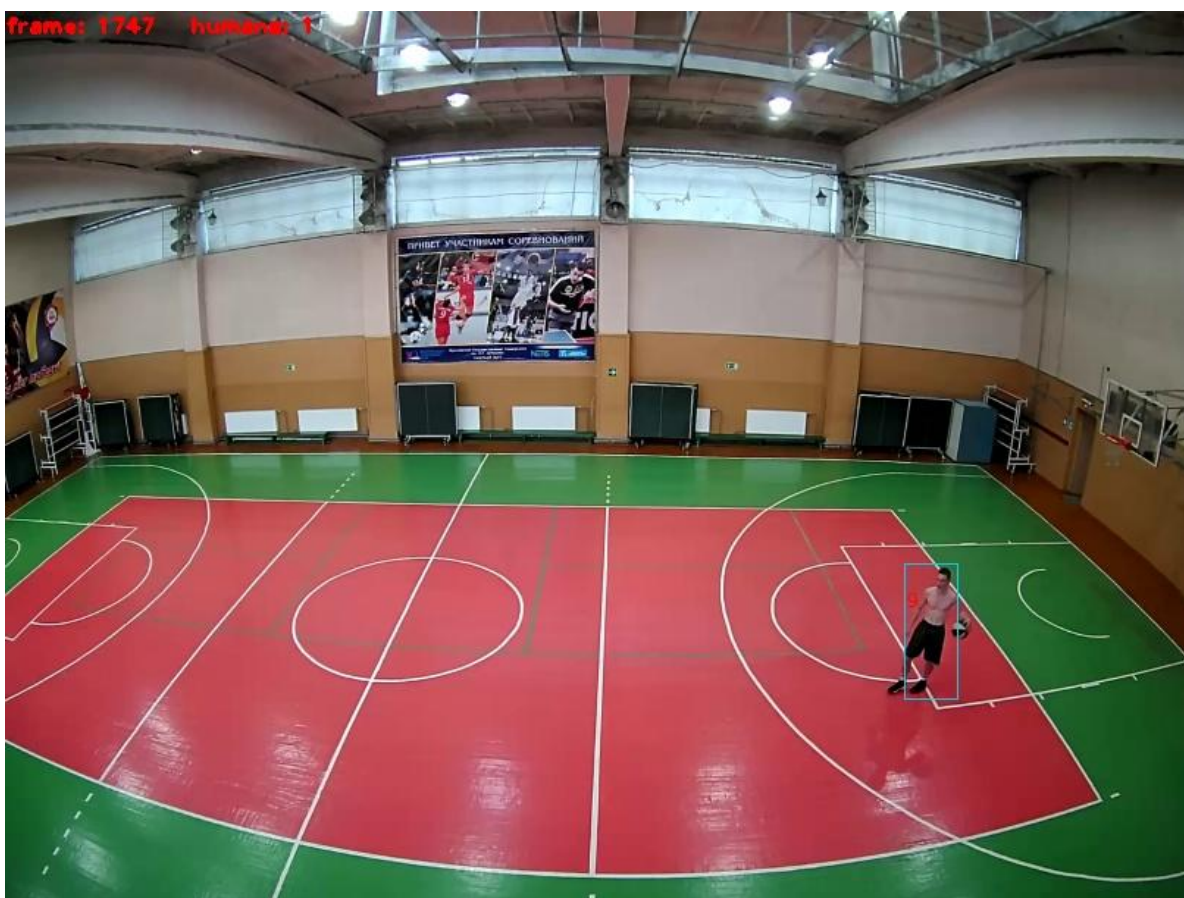
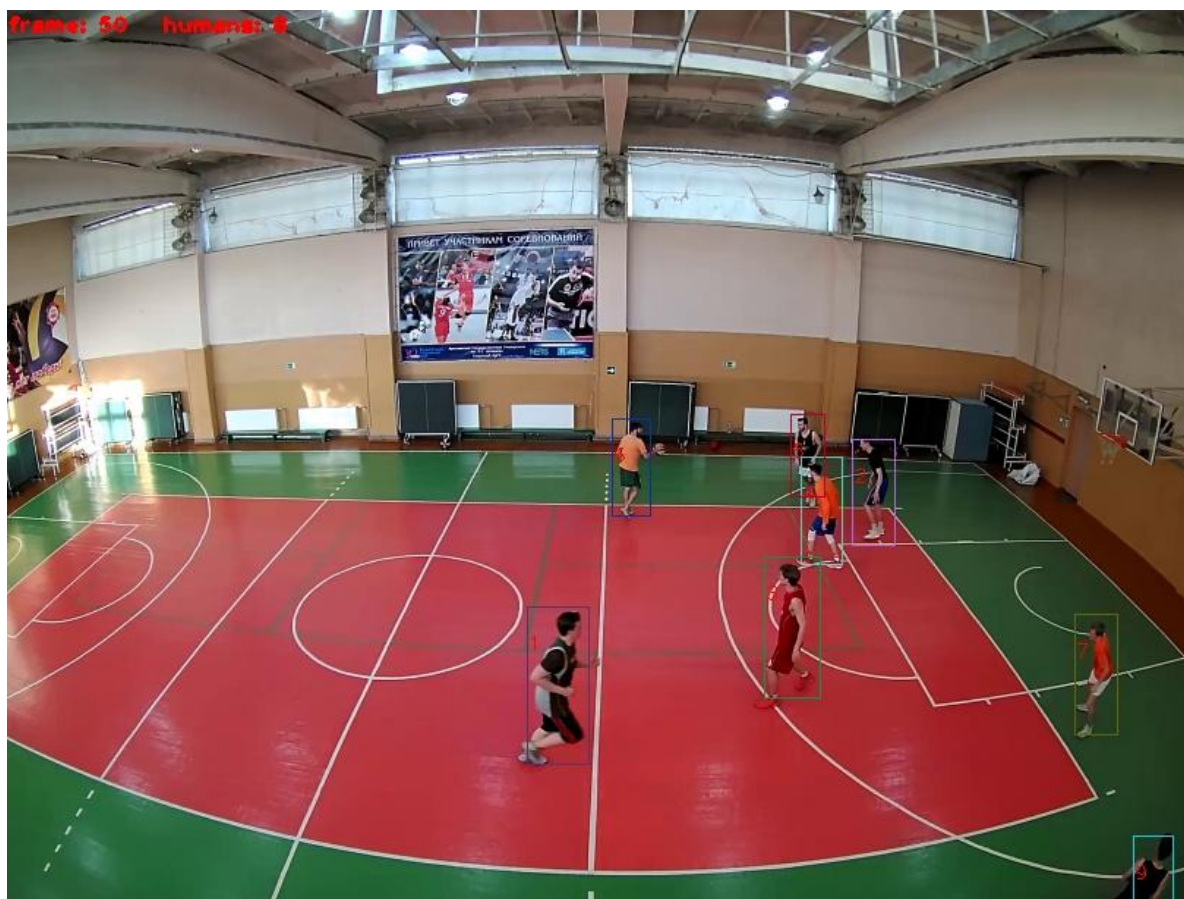


Рисунок 11 – Результаты обнаружения и отслеживания игроков на видеофайлах со спортивных мероприятий, снятых с диагонального ракурса купольной камеры

3.2. Разработка прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий.

Помимо доработки алгоритмов трекинга и детектирования на основе сверточной нейронной сети на данном этапе выполнения НИОКР требовалось также разработать прототип программно-аппаратного комплекса для видеоанализа спортивных мероприятий. Другими словами, необходимо было:

1. Создать приложение с удобным для пользователя графическим интерфейсом, которое осуществляет обнаружение и отслеживание спортсменов на игровой площадке.
2. Внедрить в созданный программный продукт разработанные алгоритмы искусственного интеллекта.
3. Осуществить испытания прототипа программного комплекса и при необходимости доработать экспериментальный вариант приложения.

Созданный прототип программно-аппаратного комплекса был разработан на языке Python с использованием библиотек PyQt и OpenCV.

Python – это сценарный высокоуровневый интерпретируемый язык программирования общего назначения, применимый для большого спектра задач. Python нередко используется как универсальная среда разработки в качестве замены многим коммерческим продуктам. Этот язык поддерживает парадигмы структурного и объектно-ориентированного программирования. К полезным свойствам Python также относятся минималистичный синтаксис языка, возможность автоматического управления памятью и динамическая типизация данных [28, 29].

Для создания интерфейса программного комплекса (GUI – Graphical User Interface) была выбрана библиотека PyQt. PyQt – это набор расширений графического фреймворка Qt, разработанный компанией Riverbank Computing, для Python, выполненный в виде расширения языка программирования. PyQt практически полностью реализует возможности Qt, а

это более 600 классов, более 6000 функций и методов, включая существующий набор виджетов графического интерфейса, их стили, а также поддержку воспроизведения видео и аудио [30]. Qt – это кроссплатформенный фреймворк для разработки программного обеспечения. Эта библиотека включает в себя стандартные элементы пользовательского интерфейса, а также основные классы, которые могут потребоваться при разработке GUI-приложений. Со времени своего появления этот фреймворк лег в основу многих проектов. Qt является полностью объектно-ориентированным и легко расширяемым [31].

OpenCV (Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) – свободно распространяемая библиотека алгоритмов компьютерного зрения с открытым исходным кодом. Она позволяет производить различные операции над изображениями, а также запускать численные алгоритмы общего назначения. В общей сложности фреймворк OpenCV содержит около 2500 уже реализованных алгоритмов для изменения размеров изображения, устранения оптических искажений определения формы объекта на снимке и т.д. Большая часть алгоритмов, реализованных в фреймворке OpenCV, является ресурсоемкой, что может требовать большого объема вычислительных ресурсов, поэтому возможно использование как использование вычислительных ядер центрального процессора компьютера, так и графических ускорителей видеокарты. Библиотека изначально предназначалась для внедрения в проекты, написанные на языке C/C++, однако на сегодняшний день OpenCV разрабатывается также и для Python [32].

Возможности языка Python, а также фреймворков PyQt OpenCV позволили создать экспериментальный образец требуемого программно-аппаратного комплекса. Скриншот главного экрана приложения показан на рисунке 12. Фрагменты кода экспериментального образца программного

комплекса приведены в приложении А. Описание программы и ее основных возможностей указаны в приложении Б данного научного отчета.

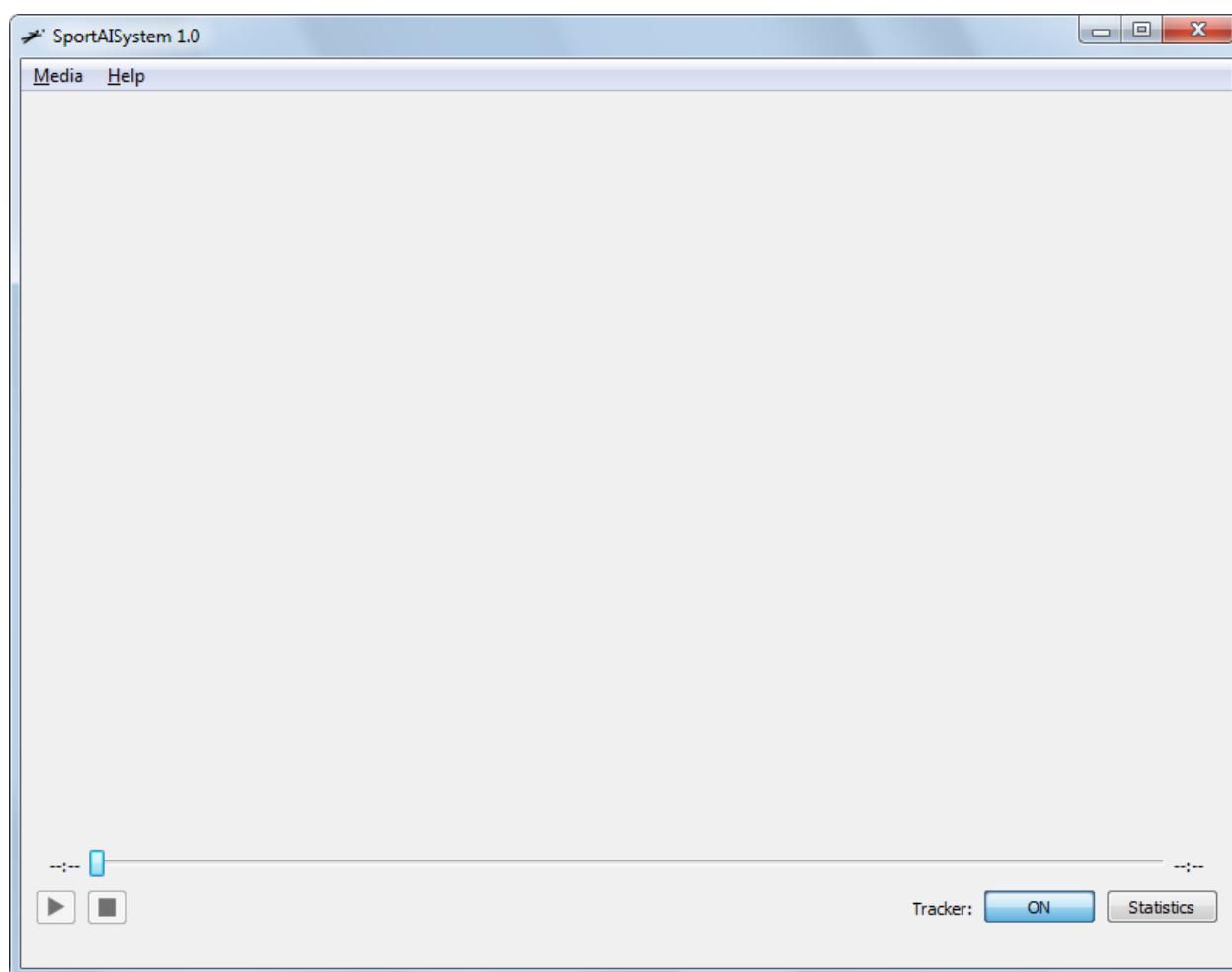


Рисунок 12 – Главный экран программного комплекса

3.3. Натурные испытания прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий.

Согласно проведенным испытаниям алгоритм JDE имеет большее количество переключений на другой объект при отслеживании (ID_switches) по сравнению с некоторыми существующими методами отслеживания нескольких объектов. Основная причина такого эффекта заключается в недостаточно точном обнаружении спортсменов на игровой площадке, когда один из игроков имеет достаточно большую площадь перекрытия другим объектом. Как следствие, после этого, алгоритм JDE мог совершить ошибку детектирования спортсмена на игровой площадке. На рисунках 13, 14 продемонстрированы случаи, когда алгоритм JDE совершал ошибки детектирования в случае перекрытия одного из объектов и последующего переключения на другой объект при отслеживании на данных с конкурса MOT Challenge и на видеофайлах со спортивных мероприятий, снятых с диагонального ракурса купольной камеры.



Рисунок 13 – Ошибки в случае перекрытия одного из объектов и последующего переключения на другой при отслеживании на данных с конкурса MOT Challenge

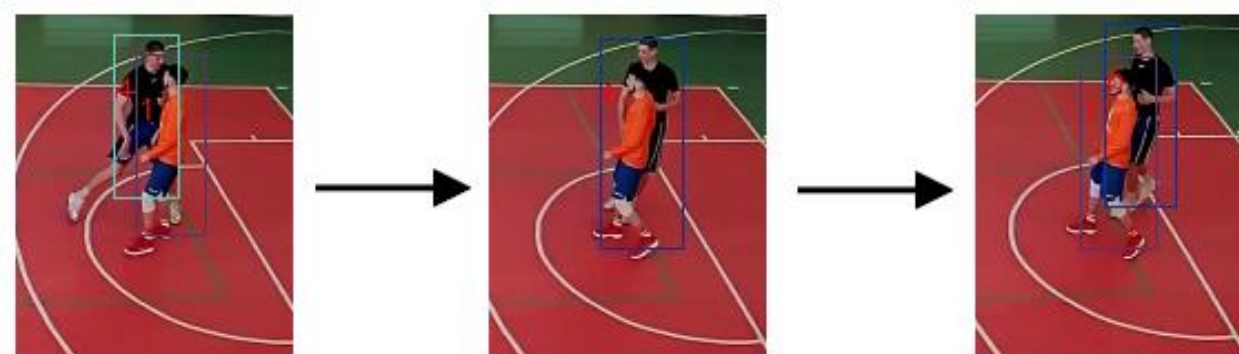


Рисунок 14 – Ошибки в случае перекрытия одного из объектов и последующего переключения на другой на видеофайлах со спортивных мероприятий

Немного улучшить значение метрики MOTA позволила корректировка значений пороговых величин, необходимых для выделения фона и объектов на переднем плане. Если коэффициент IoU превышал верхнюю пороговую величину 0.5, то найденный объект выводился на передний план. Это согласуется с общими настройками алгоритмов детектирования при обнаружении объектов на снимках. В том случае, если коэффициент IoU принимал значение меньше нижней пороговой величины, равной 0.4, то считалось что выделенный объект – фоновый. Для большинства алгоритмов обнаружения и для первоначального запуска алгоритма JDE это значение считалось равным 0.3.

Таблица 2 – Сравнение значений метрик качества алгоритма JDE для различных значений пороговых величин

	MOTA
Стандартные значения пороговых величин	60,6
Обновленные значения пороговых величин	64,4

В таблице 2 представлено сравнение работы алгоритма JDE при начальном, стандартном наборе значений пороговых величин и на обновленном, где нижняя граница считалась равной 0.4. Как видно из результатов, доработка метода позволила увеличить значение метрики MOTA: рост величины оказался равным около 4%.

Результаты обнаружения и отслеживания спортсменов на игровой площадке для алгоритма JDE с обновленными значениями параметров показаны на рисунке 15.



Рисунок 15 – Результаты обнаружения и отслеживания спортсменов на игровой площадке для алгоритма JDE с обновленными значениями параметров

3.4. Доработка по результатам испытаний прототипа программно-аппаратного комплекса для видеоанализа спортивных мероприятий.

Первые осуществленные натурные испытания прототипа программно-аппаратного комплекса показали, что экспериментальный образец позволяет пользователю достаточно успешно работать с видеоданными со спортивных мероприятий. Однако по ходу тестирования программного продукта стало ясно, что информация о распознанных на видеофайлах окантовочных прямоугольных рамок вокруг спортсменов недостаточно информативна. Для потенциальных клиентов приложения, спортивных аналитиков и тренеров, такая информация поможет разве что быстрее обнаруживать игроков на кадрах видеопоследовательности. В связи с этим, возникла идея о создании и встраивании в проект дополнительного модуля, осуществляющего подсчет базовых статистических показателей для всех игроков на игровой площадке или для какого-то определенного спортсмена. Исходя из собранных модулем статистических показателей, формируются также:

1. Данные о дистанции, пройденной игроком.
2. Тепловая карта перемещений спортсменов.
3. Количество взаимодействий между спортсменами.

Это является фундаментом для оценки действий конкретных игроков и для проведения дальнейшего анализа более высокого уровня, например, для вычисления других коэффициентов эффективности игроков или определения ролей спортсменов в команде [3, 4]. Помимо этого, собранная информация позволит скорректировать процесс тренировок. Подсчет статистических показателей для всех игроков или какого-то конкретного спортсмена, а так же их наглядное представление в виде сформированных изображений осуществлялось на языке Python с использованием библиотек NumPy и Matplotlib.

NumPy – это кроссплатформенное расширение для языка Python, добавляющее поддержку дополнительных математических функций. Эта

библиотека широко используется для решения задач с данными матричного вида [33, 34]. Многие другие разработанные для языка Python модули используют NumPy как основной элемент своей инфраструктуры. Поскольку Python – это интерпретируемый язык, математические алгоритмы, реализованные на нем, часто работают гораздо медленнее своих аналогов, написанных на компилируемых языках программирования. Библиотека NumPy позволяет решить эту проблему для многих вычислительных алгоритмов, оптимизированных для работы с многомерными массивами.

Matplotlib – это кроссплатформенная библиотека для языка Python, построенная на принципах объектно-ориентированного программирования [35]. Расширение является гибким и легко конфигурируемым, которое вместе с NumPy позволяет решать задачу визуализации матричных данных [36, 37]. Matplotlib поддерживает многие виды двухмерных и трехмерных графиков. Ко всему прочему эта библиотека также позволяет визуализировать данные в виде вертикальных и горизонтальных столбчатых диаграмм,.

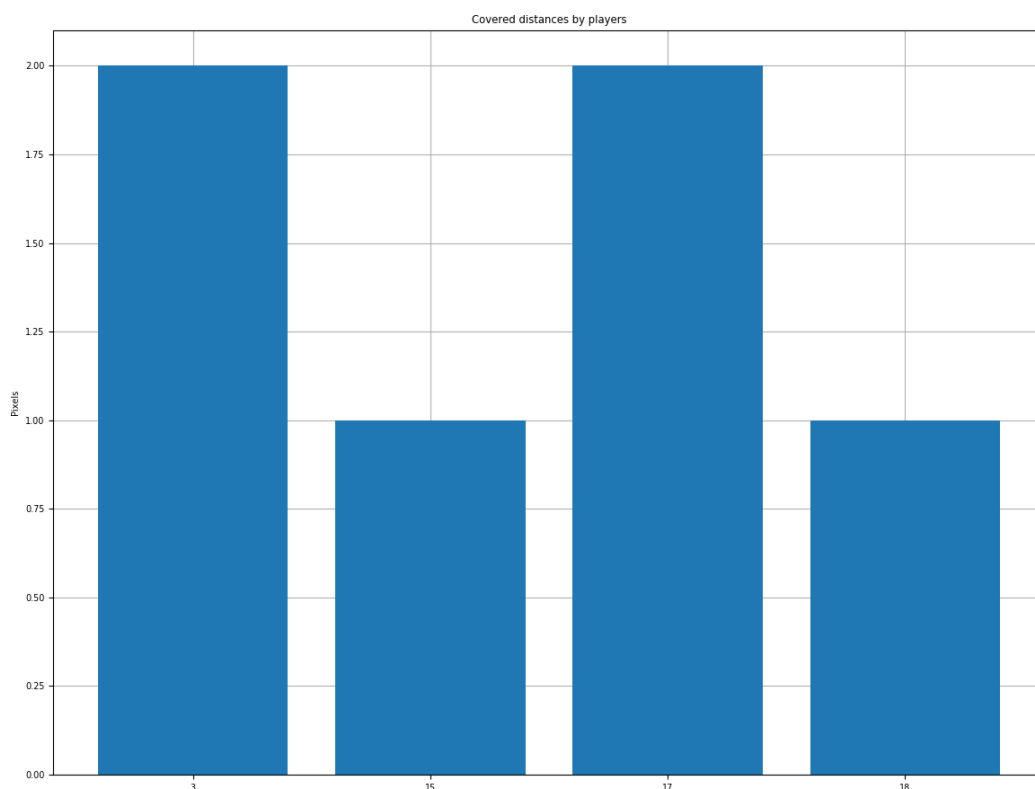


Рисунок 16 – Диаграмма, отражающая взаимодействие спортсмена с другими игроками

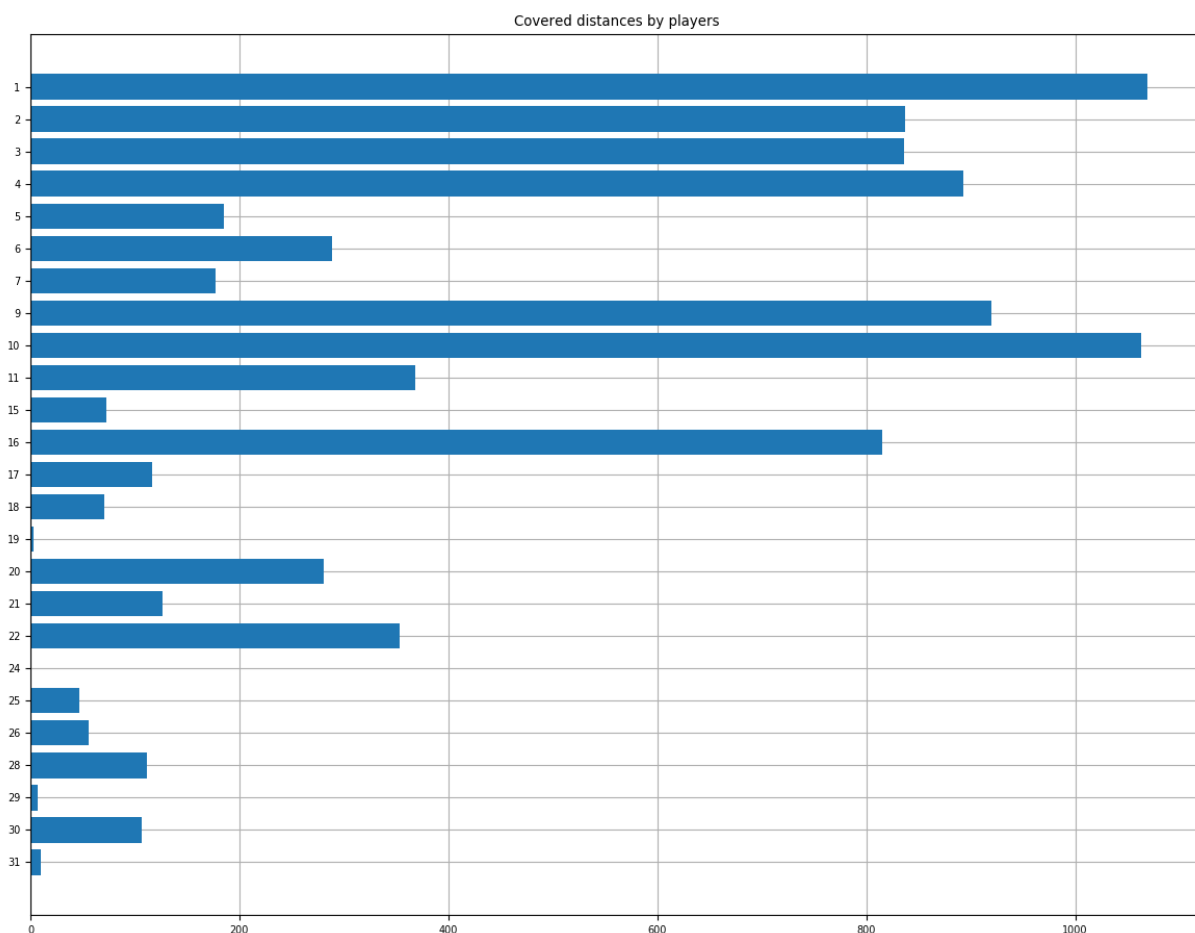


Рисунок 17 – Диаграмма, отражающая дистанции (в пикселях),
проденные спортсменами во время игры

Тепловая карта перемещений строится на основании всех окантовочных прямоугольных рамок, полученных с помощью алгоритма обнаружения и отслеживания объектов. Для каждой прямоугольной рамки вычисляются координаты так называемой «опорной точки» спортсмена: она располагается на середине нижней стороны окантовочной прямоугольной рамки. После этого все такие точки моделируются на изображение игровой площадки. И в конце формируется окончательное изображение тепловой карты перемещений: та зона игровой площадки, где спортсмен провел больше всего времени, будет выделена красным цветом, тогда как зоны, где игрок практически не находился, будут отмечены синим цветом или не выделены вовсе. На рисунке 18 показан рисунок сгенерированной тепловой карты перемещении одного из спортсменов.



Рисунок 18 – Тепловая карта перемещения спортсмена по игровой площадке

4. ЗАКЛЮЧЕНИЕ

К основным результатам, полученным на втором этапе НИОКР могут быть отнесены следующие:

- Был доработан алгоритм обнаружения и отслеживания спортсменов для видеоданных со спортивных мероприятий, снятых с диагонального ракурса купольной камеры. В его основу был взят современный метод глубокого машинного обучения, основанный на использовании сверточных нейронных сетей. Был использован алгоритм JDE, в которой модель внедрения обнаруженных областей локализации сопряжена с алгоритмом детектирования целевого объекта таким образом, что система отслеживания может одновременно и выполнять обнаружение и ассоциировать обнаруженные области расположения целевых объектов. Обучение и тестирование алгоритма JDE осуществлялось на наборе видеофайлов с конкурса MOT Challenge. Обучение и тестирование алгоритмов JDE осуществлялся на графическом процессоре суперкомпьютера NVIDIA DGX-1. Критерием оценки качества работы алгоритма отслеживания на видеоданных служила метрика MOTA. В результате осуществления процессов обучения и тестирования алгоритма JDE, ее значение было равным 60,6.
- Был разработан прототип программно-аппаратного комплекса для видеоанализа спортивных мероприятий на языке Python с использованием библиотек PyQt и OpenCV.
- Первые натурные испытания прототипа программно-аппаратного комплекса показали, что экспериментальный образец позволяет пользователю успешно работать с видеоданными со спортивных мероприятий. В то же время тестирование алгоритма JDE продемонстрировало, что временами, при отслеживании игрока

модель может переключаться на другой объект. Основная причина такого эффекта заключается в недостаточно точном обнаружении спортсменов на игровой площадке, когда один из целевых объектов имеет достаточно большую площадь перекрытия другим. Как следствие, после этого, возникала ошибка детектирования игрока. Немного улучшить значение метрики МОТА позволила корректировка значений пороговых величин, необходимых для выделения фона и объектов на переднем плане. Доработка модели позволила увеличить метрику качества алгоритма обнаружения и отслеживания примерно на 4%: значение метрики МОТА составило 64,4.

- По ходу тестирования программного продукта стало ясно, что информация о распознанных на видеофайлах окантовочных прямоугольных рамок вокруг спортсменов недостаточно информативна. В связи с этим, возникла идея о создании дополнительного модуля по подсчету базовых статистических показателей, таких как пройденная дистанция, количество взаимодействий с игроками и тепловая карта перемещений для всех спортсменов или для какого-то определенного человека. Это является фундаментом для проведения дальнейшего анализа более высокого уровня, например, для вычисления других коэффициентов эффективности игроков или определения ролей спортсменов в команде. Подсчет статистических показателей, а также их наглядное представление в виде сформированных изображений осуществлялось на языке Python с использованием библиотек NumPy и Matplotlib.

Работы по третьему, заключительному этапу НИОКР были выполнены в полном объеме в соответствии с техническим заданием и календарным планом. Полученные результаты будут использованы на следующих этапах НИОКР и в коммерческой деятельности ООО «Цифровые решения».

5. СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- [1] Lu W.L., Ting J.A., Murphy K.P., Little J.J.: Identifying Players in Broadcast Sports Videos Using Conditional Random Fields. // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2011, pp. 3249 – 3256.
- [2] Гонсалес Р., Вудс Р. Цифровая обработка изображений. – М.: Техносфера, 2005. – 621 с.
- [3] Что такое xG? Веб: <https://www.futbologika.com/articles/skauting-analitika/chto-takoe-xg.html>
- [4] Клетте Р. Компьютерное зрение. Теория и алгоритмы. – М.: ДМК Пресс, 2019, 506 с.
- [5] Wang Z., Zheng L., Liu Y., Wang S. Towards Real-Time Multi-Object Tracking. Веб: <https://arxiv.org/pdf/1909.12605.pdf>.
- [6] MOT Challenge – Multiple Object Tracking Benchmark. Веб: <https://motchallenge.net>.
- [7] UA-DETRAC Challenge. Веб: <http://detrac-db.rit.albany.edu>.
- [8] CVPR Tracking and Detection Challenge. Веб: <https://arxiv.org/pdf/1906.04567.pdf>.
- [9] Intersection over Union (IoU) for object detection. Веб: <https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>
- [10] Object Detection IOU - Intersection Over Union. Веб: <https://medium.com/@nagsan16/object-detection-iou-intersection-over-union-73070cb11f6e>.
- [11] Mazzeo L., Ramakrishnan S., Spagnolo P. Visual Object Tracking with Deep Neural Networks // DOI: 10.5772/intechopen.80142, 2019.
- [12] Ozer C., Gurkan F., Gunsel B. Target Aware Visual Object Tracking // Image Analysis and Recognition. ICIAR 2019. Lecture Notes in Computer Science, vol 11663, 2019, pp. 186 – 192.

- [13] Ristani E., Solera F., Zou R.S., Cucchiara R., Tomasi C. Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking // ECCV 2016, LNCS, vol. 9914, 2016, pp. 17 – 35.
- [14] Гудфеллоу Я., Бенджио И., Курвилль А. Глубокое обучение. – М.: ДМК Пресс, 2017. – 652 с.
- [15] Николенко С., Кадуринов А., Архангельская Е. Глубокое обучение. Погружение в мир нейронных сетей. – СПб.: Питер, 2018, 480 с.
- [16] Рашид Т. Создаем нейронную сеть. – М.: ООО «И.Д. Вильямс», 2018, 272 с.
- [17] Milan A., Leal-Taixe L., Reid I., Roth S., Schindler K. MOT16: A Benchmark for Multi-Object Tracking. Веб: <https://arxiv.org/pdf/1603.00831.pdf>.
- [18] Bergmann P., Meinhardt T., Leal-Taixe L. Tracking without bells and whistles. Веб: <https://arxiv.org/pdf/1903.05625.pdf>.
- [19] Xiao T., Li S., Wang B., Lin L., Wang X. Joint detection and identification feature learning for person search // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 3415 – 3424.
- [20] Ren S., He K., Girshick R., Sun J. Faster R-CNN: Towards real-time object detection with region proposal networks // NIPS, 2015, pp. 91 – 99.
- [21] Girshick, R. Fast R-CNN. Веб: <https://arxiv.org/pdf/1504.08083.pdf>.
- [22] Liu W., Anguelov D., Erhan d., Szegedy C., Reed S., Fu C.-Y., Berg A.C. SSD: Single Shot MultiBox Detector // Computer Vision – ECCV 2016, pp. 21 – 37.
- [23] Long J., Shelhamer E., Darrell T. Fully convolutional networks for semantic segmentation // IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3431 – 3440.
- [24] Lin, T.-Y.; Dollar, P.; Girshick, R.; He, K.; Hariharan, B.; and Belongie, S. 2017. Feature pyramid networks for object detection. In CVPR.
- [25] Макмахан Б. Знакомство с PyTorch: глубокое обучение при обработке естественного языка – СПб.: Питер, 2020 – 250 с.

- [26] Боресков А., Харламов А. Основы работы с технологией CUDA. – М.: ДМК Пресс, 2013 – 232 с.
- [27] Сандерс Дж., Кэндрот Э. Технология CUDA в примерах. Введение в программирование графических процессоров. – М.: ДМК Пресс, 2013 – 232 с.
- [28] Лутц М. Python. Карманный справочник, 5-е изд. – М.: ООО «И.Д. Вильямс», 2014 – 320 с.
- [29] Любанович Б. Простой Python. Современный стиль программирования – СПб.: Питер, 2019, 480 с.
- [30] Прохоренок Н., Дронов В. Python 3. Самое необходимое. – СПб.: БХВ-Петербург, 2016 – 464 с.
- [31] Шлее М. Qt 5.10. Профессиональное программирование на C++. – СПб.: БХВ-Петербург, 2018 – 1072 с.
- [32] Гарсия Г.Б., Суарес О.Д., Аранда Х.Л.Э., Терсеро Х.С., Грасиа И.С., Энано Н.В. Обработка изображений с помощью OpenCV. – М.: ДМК Пресс, 2016 – 210 с.
- [33] Бэйдер Д. Чистый Python. Тонкости программирования для профи. – СПб.: Питер, 2020 – 288 с.
- [34] Шлюссер Т., Рейтц К. Автостопом по Python. – СПб.: Питер, 2017 – 336 с.
- [35] Мэтиз Э. Изучаем Python. Программирование игр, визуализация данных, веб-приложения. – СПб.: Питер, 2020, 512 с.
- [36] Жерон О. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем. – М.: ООО «И.Д. Вильямс», 2018 – 688 с.
- [37] Рашка С., Мирджалили В. Python и машинное обучение. Машинное и глубокое обучение с использованием Python, scikit-learn и TensorFlow. – М.: ООО «И.Д. Вильямс», 2019 – 656 с.
- [38] Милл И., Сейерс Э.Х. Docker на практике. – М.: ДМК Пресс, 2020 – 516 с.

- [39] Флах П. Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных. Учебник. – М.: ДМК Пресс, 2015 – 400 с.
- [40] Хайнеман Дж., Поллис Г. Алгоритмы. Справочник с примерами на C, C++, Java и Python. – М.: ООО «И.Д. Вильямс», 2017 – 432 с.

ПРИЛОЖЕНИЯ

Приложение А. Фрагменты кода программы для спортивных аналитических систем на основе технологий машинного обучения

background_extraction.py:

```
import argparse
import sys
import cv2
import os
import numpy as np
from tqdm import tqdm

import constants

def init_argparse():
    """
    Initializes argparse
    """
    parser = argparse.ArgumentParser(description='Background extraction from the
video')
    parser.add_argument(
        '--video',
        nargs='?',
        help='Path to the video',
        required=True,
        type=str)
    parser.add_argument(
        '--strategy',
```

```

    nargs='?',
    help='Choose the strategy of extracting the background from the video
    {}'.format(constants.BACKGROUND_STRATEGIES),
    default='median',
    type=str)
parser.add_argument(
    '--frequency',
    nargs='?',
    help="""Number of frames which took part in the background extraction
    For median or mean strategy - amount of randomly uniform taken frames to
    get the 'MEDIAN' or 'MEAN' frame
    For cumulated strategy - 1/frequency is a value of accumulating the weights
    of frames
    """,
    default=100,
    type=int)
return parser

```

```

def get_cumulated_background(cap, total_frames, freq):

```

```

    """
    Get matrix of background extracted from the video using cumulated weights of
    frames
    :param cap: captured video
    :param total_frames: total number of frames in the video
    :param freq: frequency of cumulating the weights of previous frames
    :return: cumulated background matrix
    """
    cap.set(cv2.CAP_PROP_POS_FRAMES, 0) # set the position of video into its
    start

```

```

_, frame = cap.read()
cumulated_frame = np.float32(frame)
for fid in tqdm(range(1, int(total_frames))):
    cap.set(cv2.CAP_PROP_POS_FRAMES, fid)
    _, frame = cap.read()
    if frame is not None:
        cv2.accumulateWeighted(frame, cumulated_frame, freq)
        res = cv2.convertScaleAbs(cumulated_frame)
return res

```

```

def get_calculated_background(cap, total_frames, freq, strategy):

```

```

    """

```

Get matrix of background calculated by the use of sum of random frames in the video

```

:param cap: captured video
:param total_frames: total number of frames in the video
:param freq: count of frames choosen randomly
:param strategy: strategy of calculating the matrix of background
:return: background matrix
    """

```

```

    # Get indices of frames wich will be used for background extraction

```

```

    frame_indices = total_frames*np.random.uniform(size=freq) # random uniform
rule

```

```

    # Store the frames

```

```

    collected_frames = [] # storage of frames

```

```

    for _, fid in enumerate(tqdm(frame_indices)):

```

```

        cap.set(cv2.CAP_PROP_POS_FRAMES, int(fid))

```

```

        _, frame = cap.read()

```

```

        collected_frames.append(frame)

```

```

# Calculate the background image according to choosen strategy
if strategy == 'median':
    res = np.median(collected_frames, axis=0).astype(dtype=np.uint8)
else:
    res = np.mean(collected_frames, axis=0).astype(dtype=np.uint8)
return res

```

```

def main():
    parser = init_argparse()
    # Extract arguments of script
    args = parser.parse_args()
    # Extract video and count of frames
    cap = cv2.VideoCapture(args.video)
    total_frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
    print('Amount of frames:\t{ }'.format(total_frames))
    # Extract strategy
    strategy = args.strategy
    if strategy not in constants.BACKGROUND_STRATEGIES:
        print('Not available strategy! Please choose the right
one:\t{ }'.format(constants.BACKGROUND_STRATEGIES))
        sys.exit(1)
    print('Strategy of background extraction:\t{ }'.format(strategy))
    # Extract frequency
    if args.frequency <= 0:
        print('Not available value of frequency! Please choose the positive value...')
        sys.exit(1)
    if strategy == 'cumulated':
        freq = 1.0/args.frequency
    else:

```

```

    freq = min(args.frequency, total_frames)
    print('Frequency of frames:\t{ }'.format(freq))
    # Extract the background from the video
    if strategy == 'cumulated':
        bg = get_cumulated_background(cap, total_frames, freq)
    else:
        bg = get_calculated_background(cap, total_frames, freq, strategy)
    # Save extracted background
    img_name = os.path.splitext(os.path.basename(args.video))[0]
    cv2.imwrite(img_name+'.png', bg)

if __name__ == '__main__':
    main()

```

combatscounter.py:

```

import argparse
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import defaultdict

import operations

plt.rcParams.update({'font.size': 7})

```

```

class CombatsCounter():

    def __init__(self, markup_file, out_dir, human_number=None):
        self.__data = pd.read_csv(markup_file, names=['frame', 'id', 'bb_y', 'bb_x',
'bb_h', 'bb_w'])
        self.__ids = list(self.__data['id'].unique())
        self.__countframes = max(self.__data['frame'])
        self.__countsids = len(self.__ids)
        self.__outdirectory = out_dir
        self.__human = human_number
        self.__combatsmatrix = np.zeros((len(self.__ids), len(self.__ids)),
dtype=np.int)

    def __buildCombatsMatrix(self):
        existing_combats = defaultdict(set) # combats, which were registered on
previous frame
        for frame_id in range(1, self.__countframes+1):
            # Extract all detected bbox on current frame
            frame_bboxes = self.__data[self.__data['frame'] == frame_id][['id', 'bb_y',
'bb_x', 'bb_h', 'bb_w']]
            n_bboxes = len(frame_bboxes)
            for i in range(n_bboxes-1):
                for j in range(i+1, n_bboxes):
                    id_i, id_j = int(frame_bboxes.iloc[i]['id']),
int(frame_bboxes.iloc[j]['id'])
                    if id_i != id_j:

```



```

        if operations.is_bbox_intersected(frame_bboxes.iloc[i],
frame_bboxes.iloc[j]):
            if id_j not in existing_combats[id_i]: # combat was not registered
                # Add a combat to matrix on symmetric places
                idx_i, idx_j = self.__ids.index(id_i), self.__ids.index(id_j)
                self.__combatsmatrix[idx_i, idx_j] += 1
                self.__combatsmatrix[idx_j, idx_i] += 1
                # Register new combat
                existing_combats[id_i].add(id_j)
                existing_combats[id_j].add(id_i)
            else:
                # stop taking account completed combat
                existing_combats[id_i].discard(id_j)
                existing_combats[id_j].discard(id_i)

```

```

def __buildHumanCombatsDictionary(self):
    self.__combatsdict = dict()
    for i, comb in enumerate(list(self.__combatsmatrix[self.__ids.index(self.__human)])):
        if comb != 0:
            self.__combatsdict[self.__ids[i]] = self.__combatsmatrix[i,
self.__ids.index(self.__human)]

```

```

def __drawCombatsMatrix(self):
    fig, ax = plt.subplots(figsize=(12, 9))
    plt.subplots_adjust(left=0.03, bottom=0.03, right=0.97, top=0.97)
    im = ax.imshow(self.__combatsmatrix)
    ax.set_xticks(np.arange(self.__countsids))

```

```

ax.set_yticks(np.arange(self.__countsids))
ax.set_xticklabels(self.__ids)
ax.set_yticklabels(self.__ids)
ax.tick_params(top=True, bottom=True, left=True, right=True,
labeltop=True, labelright=True)
colorbar = ax.figure.colorbar(im, ax=ax)
colorbar.ax.set_ylabel('Number of combats', rotation=-90, va='bottom')
for i in range(len(self.__ids)):
    for j in range(len(self.__ids)):
        ax.text(j, i, self.__combatsmatrix[i, j], ha='center', va='center', color='w')
plt.savefig(os.path.join(self.__outdirectory, 'combats_matrix.png'))

```

```

def __drawBarChartHumanCombats(self):
    self.__buildHumanCombatsDictionary()
    d = self.__combatsdict
    if len(d) != 0:
        fig, ax = plt.subplots(figsize=(12, 9))
        plt.subplots_adjust(left=0.04, bottom=0.04, right=0.96, top=0.96)
        plt.grid()
        ax.bar(np.arange(len(d)), list(d.values()), zorder=2)
        ax.set_xticks(np.arange(len(d)))
        ax.set_xticklabels(list(map(str, d.keys())))
        ax.set_title('Covered distances by players')
        ax.set_ylabel('Pixels')
        fig.savefig(os.path.join(self.__outdirectory,
'combats__human_{ }.png'.format(self.__human)))

```

```

def calculateCombatsStatistics(self):

```

```

self.__buildCombatsMatrix()
if self.__human is None:
    self.__drawCombatsMatrix()
else:
    self.__buildHumanCombatsDictionary()
    self.__drawBarChartHumanCombats()

def init_argparse():
    """
    Initializes argparse
    """
    parser = argparse.ArgumentParser(description='Background extraction from the
video')
    parser.add_argument(
        '--markup',
        nargs='?',
        help='Markup file',
        required=True,
        type=str)
    parser.add_argument(
        '--out_dir',
        nargs='?',
        help='Output directory for saving files with calculated statistics',
        required=True,
        type=str)
    parser.add_argument(
        '--human',
        nargs='?',
        help='Number of sportsman',

```

```
        default=None,  
        type=int)  
    return parser
```

```
def main():  
    parser = init_argparse()  
    # Extract arguments of script  
    args = parser.parse_args()  
    # Calculate statistics about combats  
    comb_acc = CombatsCounter(args.markup, args.out_dir, args.human)  
    comb_acc.calculateCombatsStatistics()  
  
if __name__ == '__main__':  
    main()
```

emailsending.py:

```
import argparse  
import os  
import yagmail  
import constants
```

```
class EMailSending:  
  
    def __init__(self):
```

```

self.__fromuser = constants.SENT_FROM_USER
self.__frompassword = constants.SENT_FROM_PASSWORD

def sendEMail(self, to_users: list, attached_files_paths: list):
    smtp_connection = yagmail.SMTP(user=self.__fromuser,
password=self.__frompassword,
                                host='smtp.gmail.com') # connect to SMTP server
    email_subject = 'SportAISystem 1.0 Statistics'
    smtp_connection.send(to_users, email_subject, attached_files_paths) # send
the email

def init_argparse():
    """
    Initializes argparse
    """
    parser = argparse.ArgumentParser(description='Sending statistics by the use of
e-mail')
    parser.add_argument(
        '--to',
        nargs='?',
        help='E-mail of recipient',
        required=True,
        type=str)
    parser.add_argument(
        '--folder',
        nargs='?',
        help='Folder with files to send',
        required=True,
        type=str)

```

```
return parser
```

```
def main():
```

```
    parser = init_argparse()
```

```
    args = parser.parse_args() # extract arguments of script
```

```
    sending = EMailSending()
```

```
    # Collect sending files
```

```
    content = [os.path.join(os.path.abspath(args.folder), file) for file in  
os.listdir(args.folder)]
```

```
    sending.sendEMail([args.to], content)
```

```
if __name__ == '__main__':
```

```
    main()
```

heatmapper.py:

```
import numpy as np
```

```
from matplotlib.colors import LinearSegmentedColormap
```

```
from PIL import Image
```

```
import constants
```

```
class Heatmapper:
```

```
    def __init__(self, point_diameter=30, point_strength=0.2, opacity=0.5):
```

```

self.point_diameter = point_diameter
self.point_strength = point_strength
self.opacity = opacity
self.cmap = =
self.__setColorMapFromImage(constants.COLORMAP_IMAGE)

def __imageToOpacity(self, img, opacity):
    img = img.copy()
    alpha = img.split()[3]
    img.putalpha(alpha.point(lambda p: int(p * opacity)))
    return img

def buildHeatmapOnImage(self, points, background_img):
    width, height = background_img.size
    heatmap = self.__makeHeatmap(width, height, points)
    heatmap = self.__imageToOpacity(heatmap, self.opacity)
    if background_img is not None:
        return Image.alpha_composite(background_img.convert('RGBA'),
heatmap)
    else:
        return heatmap

def __setColorMapFromImage(self, colormap_img):
    # load colormap image
    img = Image.open(colormap_img)
    img = img.resize((256, img.height))
    # extract colors from colormap image
    colours = [img.getpixel((x, 0)) for x in range(256)]
    colours = [(r/255, g/255, b/255, a/255) for r, g, b, a in colours]
    return LinearSegmentedColormap.from_list('from_image', colours)

```



```

def __makeHeatmap(self, width, height, points):
    heatmap = Image.new('L', (width, height), color=255) # empty heatmap
    spot =
Image.open(constants.SPOT_IMAGE).copy().resize((self.point_diameter,
self.point_diameter),
resample=Image.ANTIALIAS)
    spot = self.__imageToOpacity(spot, self.point_strength)
    for x, y in points: # locate spots on the heatmap
        x, y = int(x - self.point_diameter/2), int(y - self.point_diameter/2)
        heatmap.paste(spot, (x, y), spot)
    res = self.cmap(np.array(heatmap), bytes=True) # paint over using heatmap
    return Image.fromarray(res)

```

motionheatmap.py:

```

import argparse
import os
import pandas as pd
from PIL import Image

import constants
import operations
from heatmapper import Heatmapper
from traceplace import Traceplace

```

```

class MotionHeatmap():

    def __init__(self, markup_file, out_dir, human_number=None,
marker_pos='lower_center'):

        self.__data = pd.read_csv(markup_file, names=['frame', 'id', 'bb_y', 'bb_x',
'bb_h', 'bb_w'])

        self.__outdirectory = out_dir

        self.__background = Image.open(constants.BACKGROUND_READY_IMAGE)

        self.__human = human_number

        self.__markerpos = Traceplace[str(marker_pos).upper()]

    def __loadPoints(self):

        if self.__human is not None:

            self.__points = [operations.get_point(row, self.__markerpos)
for _, row in self.__data[self.__data['id'] ==
self.__human].iterrows()]

        else:

            self.__points = [operations.get_point(row, self.__markerpos) for _, row in
self.__data.iterrows()]

    def buildHeatmap(self):

        self.__loadPoints()

        heatmapper = Heatmapper()

        heatmap_img = heatmapper.buildHeatmapOnImage(self.__points,
self.__background)

        if self.__human is not None:

```

```

        heatmap_img.save(os.path.join(self.__outdirectory,
'heatmap__human_{ }.png'.format(self.__human)))
    else:
        heatmap_img.save(os.path.join(self.__outdirectory, 'heatmap.png'))

def init_argparse():
    """
    Initializes argparse
    """
    parser = argparse.ArgumentParser(description='Background extraction from the
video')
    parser.add_argument(
        '--markup',
        nargs='?',
        help='Markup file',
        required=True,
        type=str)
    parser.add_argument(
        '--outfile',
        nargs='?',
        help='Output file to save heatmap',
        required=True,
        type=str)
    parser.add_argument(
        '--human',
        nargs='?',
        help='Number of sportsman',
        default=None,
        type=int)

```

```

parser.add_argument(
    '--traceplace',
    nargs='?',
    help='Place of marker on the bbox where the trace is drawing',
    default='lower_center',
    type=str)
return parser

```

```

def main():
    parser = init_argparse()
    # Extract arguments of script
    args = parser.parse_args()
    # Building heatmap of motions
    mh = MotionHeatmap(markup_file=args.markup, out_file=args.out_dir,
                       human_number=args.human, marker_pos=args.traceplace)
    mh.buildHeatmap()

if __name__ == '__main__':
    main()

```

motiontrajectories.py:

```

import argparse
import cv2
import json
import os

```

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from collections import defaultdict
from scipy.spatial import distance

import constants
import operations
from traceplace import Traceplace

TRAJECTORY_IMAGE = 'trajectories.png'

class MotionTrajectories():

    def __init__(self, markup_file, out_dir, human_number=None,
marker_pos='lower_center'):
        self.__data = pd.read_csv(markup_file, names=['frame', 'id', 'bb_y', 'bb_x',
'bb_h', 'bb_w'])
        self.__human = human_number
        if self.__human is not None:
            self.__data = self.__data[self.__data['id'] == self.__human]
        self.__outdirectory = out_dir
        self.__background = np.asarray(Image.open(constants.BACKGROUND_READY_IMAGE))
        self.__markerpos = Traceplace[str(marker_pos).upper()]
        self.__distances = defaultdict(int)

```

```

def calculateTraceStatistics(self):
    self.__drawTrajectories()
    self.__saveResults()

def __drawTrajectories(self):
    prev_point = dict()
    for _, row in self.__data.iterrows():
        human_id = int(row['id'])
        color = operations.get_color(human_id)
        point = operations.get_point(row, self.__markerpos)
        if int(human_id) not in prev_point.keys():
            self.__background = cv2.circle(self.__background, point, radius=3,
color=color, thickness=5)
            prev_point[human_id] = point
        else:
            self.__background = cv2.line(self.__background, point,
prev_point[human_id], color=color, thickness=2)
            self.__distances[human_id] += distance.euclidean(point,
prev_point[human_id])
            prev_point[human_id] = point

def __drawBarChartDistances(self):
    fig, ax = plt.subplots(figsize=(12, 9))
    plt.rcParams.update({'font.size': 8})
    plt.subplots_adjust(left=0.03, bottom=0.03, right=0.97, top=0.97)
    plt.grid()
    d = self.__distances

```

```

ax.barh(np.arange(len(d)), list(d.values()), zorder=2)
ax.set_yticks(np.arange(len(d)))
ax.set_yticklabels(list(map(str, d.keys())))
ax.invert_yaxis() # labels read top-to-bottom
ax.set_title('Covered distances by players')
ax.set_xlabel('Pixels')
fig.savefig(os.path.join(self.__outdirectory, 'covered_distances.png'))

def __saveResults(self):
    Image.fromarray(self.__background).save(os.path.join(self.__outdirectory,
TRAJECTORY_IMAGE))
    if self.__human is not None:
        json_filename =
'covered_distance__human_{}.json'.format(self.__human)
        with open(os.path.join(self.__outdirectory, json_filename), 'w') as fp:
            json.dump(self.__distances, fp)
    else:
        self.__drawBarChartDistances()

def init_argparse():
    """
    Initializes argparse
    """
    parser = argparse.ArgumentParser(description='Background extraction from the
video')
    parser.add_argument(
        '--markup',
        nargs='?',
        help='Markup file',

```



```

        required=True,
        type=str)
parser.add_argument(
    '--out_dir',
    nargs='?',
    help='Output directory for saving files with calculated statistics',
    required=True,
    type=str)
parser.add_argument(
    '--human',
    nargs='?',
    help='Number of sportsman',
    type=int)
parser.add_argument(
    '--traceplace',
    nargs='?',
    help='Place of marker on the bbox where the trace is drawing',
    default='lower_center',
    type=str)
return parser

```

```

def main():
    parser = init_argparse()
    # Extract arguments of script
    args = parser.parse_args()
    # Calculate statistics about trajectories
    mt = MotionTrajectories(markup_file=args.markup, out_dir=args.out_dir,
                           human_number=args.human, marker_pos=args.traceplace)
    mt.calculateTraceStatistics()

```

```
if __name__ == '__main__':
    main()
```

operations.py:

```
from traceplace import Traceplace
```

```
def time_string(ms):
    seconds = int((ms/1000) % 60)
    minutes = int((ms/(1000*60)) % 60)
    return '{:d}:{:02d}'.format(minutes, seconds)
```

```
def get_color(human_number):
    color = ((111*human_number)%255, (51*human_number)%255,
(87*human_number)%255)
    return color
```

```
def get_point(row_dataframe, marker_pos):
    if marker_pos == Traceplace.LOWER_LEFT:
        return (int(row_dataframe['bb_y']),
                int(row_dataframe['bb_x'] + row_dataframe['bb_w']))
    if marker_pos == Traceplace.LOWER_CENTER:
        return (int(row_dataframe['bb_y'] + row_dataframe['bb_h'] / 2.0),
```

```

        int(row_dataframe['bb_x'] + row_dataframe['bb_w']))
if marker_pos == Traceplace.LOWER_RIGHT:
    return (int(row_dataframe['bb_y'] + row_dataframe['bb_h']),
            int(row_dataframe['bb_x'] + row_dataframe['bb_w']))
if marker_pos == Traceplace.UPPER_LEFT:
    return (int(row_dataframe['bb_y']),
            int(row_dataframe['bb_x']))
if marker_pos == Traceplace.UPPER_CENTER:
    return (int(row_dataframe['bb_y'] + row_dataframe['bb_h'] / 2.0),
            int(row_dataframe['bb_x']))
if marker_pos == Traceplace.UPPER_RIGHT:
    return (int(row_dataframe['bb_y'] + row_dataframe['bb_h']),
            int(row_dataframe['bb_x']))
if marker_pos == Traceplace.CENTER:
    return (int((row_dataframe['bb_y'] + row_dataframe['bb_h']) / 2.0),
            int((row_dataframe['bb_x'] + row_dataframe['bb_w']) / 2.0))

```

```

def is_bbox_intersected(cur_bbox, other_bbox):

```

```

    bbox_y1 = cur_bbox['bb_y']
    bbox_y2 = cur_bbox['bb_y'] + cur_bbox['bb_h']
    bbox_x1 = cur_bbox['bb_x']
    bbox_x2 = cur_bbox['bb_x'] + cur_bbox['bb_w']
    y1 = other_bbox['bb_y']
    y2 = other_bbox['bb_y'] + other_bbox['bb_h']
    x1 = other_bbox['bb_x']
    x2 = other_bbox['bb_x'] + other_bbox['bb_w']
    if (x1 > bbox_x1 and x1 < bbox_x2 and y1 > bbox_y1 and y1 < bbox_y2) or \
        (x2 > bbox_x1 and x2 < bbox_x2 and y2 > bbox_y1 and y2 < bbox_y2) or \
        (x1 > bbox_x1 and x1 < bbox_x2 and y2 > bbox_y1 and y2 < bbox_y2) or \

```

```
(x2 > bbox_x1 and x2 < bbox_x2 and y1 > bbox_y1 and y1 < bbox_y2):  
    return True  
return False
```

traceplace.py:

```
from enum import Enum
```

```
class Traceplace(Enum):
```

```
    LOWER_LEFT = 0,  
    LOWER_CENTER = 1,  
    LOWER_RIGHT = 2,  
    UPPER_LEFT = 3,  
    UPPER_CENTER = 4,  
    UPPER_RIGHT = 5,  
    CENTER = 6
```

```
def __repr__(self):
```

```
    if self.value == Traceplace.LOWER_LEFT:  
        return 'lower_left'  
    if self.value == Traceplace.LOWER_CENTER:  
        return 'lower_center'  
    if self.value == Traceplace.LOWER_RIGHT:  
        return 'lower_right'  
    if self.value == Traceplace.UPPER_LEFT:  
        return 'upper_left'
```

```

if self.value == Traceplace.UPPER_CENTER:
    return 'upper_center'
if self.value == Traceplace.UPPER_RIGHT:
    return 'upper_right'
if self.value == Traceplace.CENTER:
    return 'center'

```

statdialog.py:

```

from PyQt5.QtWidgets import QDialog, QCheckBox, QVBoxLayout,
QHBoxLayout, \
    QRadioButton, QGroupBox, QSpinBox, QLineEdit, QLabel, QPushButton,
QSplitter, QFileDialog
from PyQt5.QtCore import Qt
import os

```

```

class StatDialog(QDialog):

```

```

    def __init__(self, parent=None, width=1024, height=768):
        super(StatDialog, self).__init__(parent)

```

```

        self.calculatePushButton = QPushButton('Calculate')

```

```

        self.heatmapCheckBox = QCheckBox('Heatmap', self)

```

```

        self.heatmapCheckBox.setToolTip('Motion heatmap on the field')

```

```

        self.heatmapCheckBox.setChecked(True)

```

```

self.combatsCheckBox = QCheckBox('Combats', self)
self.combatsCheckBox.setToolTip('Number of combats between rivals')
self.pathsCheckBox = QCheckBox('Motion paths', self)
self.pathsCheckBox.setToolTip('Motion trajectories and their lenght')


self.allRadioButton = QRadioButton('For all people')
self.allRadioButton.setChecked(True)
self.personRadioButton = QRadioButton('For specific person')
self.humanSpinBox = QSpinBox()
self.humanSpinBox.setMinimum(1)
self.allRadioButton.toggled.connect(self.allRadioButtonToggled)
self.personRadioButton.toggled.connect(self.personRadioButtonToggled)
self.allRadioButtonToggled()


self.localRadioButton = QRadioButton('Local saving')
self.localRadioButton.setChecked(True)
self.savepathLabel = QLabel('Please choose directory')
self.savepathLabel.setToolTip("")
self.savepathButton = QPushButton('...')
self.savepathButton.clicked.connect(self.chooseDirectory)
self.savepathButton.setFixedWidth(25)
saveLayout = QHBoxLayout()
saveLayout.addWidget(self.savepathLabel)
saveLayout.addWidget(self.savepathButton)
self.remoteRadioButton = QRadioButton('E-mail')
self.emailLineEdit = QLineEdit()


statGroupBox = QGroupBox('Sport statistics')
statLayout = QVBoxLayout()
statLayout.addWidget(self.heatmapCheckBox)

```

```
statLayout.addWidget(self.combatsCheckBox)
statLayout.addWidget(self.pathsCheckBox)
statLayout.addStretch(1)
statGroupBox.setLayout(statLayout)
```

```
optionsGroupBox = QGroupBox('Options')
optionsLayout = QVBoxLayout()
optionsLayout.addWidget(self.allRadioButton)
optionsLayout.addWidget(self.personRadioButton)
optionsLayout.addWidget(self.humanSpinBox)
optionsLayout.addStretch(1)
optionsGroupBox.setLayout(optionsLayout)
```

```
sendGroupBox = QGroupBox('Sending', alignment=Qt.AlignHCenter)
sendLayout = QVBoxLayout()
sendLayout.addWidget(self.localRadioButton)
sendLayout.addLayout(saveLayout)
sendLayout.addWidget(self.remoteRadioButton)
sendLayout.addWidget(self.emailLineEdit)
sendGroupBox.setLayout(sendLayout)
```

```
vertLayout = QHBoxLayout()
vertLayout.addWidget(statGroupBox)
vertLayout.addWidget(optionsGroupBox)
```

```
mainLayout = QVBoxLayout()
mainLayout.addLayout(vertLayout)
mainLayout.addWidget(QSplitter(Qt.Horizontal))
mainLayout.addWidget(sendGroupBox)
mainLayout.addWidget(QSplitter(Qt.Horizontal))
```



```

mainLayout.addWidget(self.calculatePushButton)

self.setLayout(mainLayout)
self.setMinimumWidth(300)
self.setWindowTitle('Videoanalysis')

def allRadioButtonToggled(self):
    self.humanSpinBox.setDisabled(True)
    self.humanSpinBox.setToolTip("")

def personRadioButtonToggled(self):
    self.humanSpinBox.setDisabled(False)
    self.humanSpinBox.setToolTip('Choose the number of human')

def chooseDirectory(self):
    dirname = QFileDialog.getExistingDirectory(self, 'Choose directory',
os.getcwd(), QFileDialog.ShowDirsOnly)
    if dirname:
        reduced_dirname = '...{}'.format(str(dirname)[-41:]) if len(dirname) > 44
    else str(dirname)
        self.savepathLabel.setText(reduced_dirname)
        self.savepathLabel.setToolTip(str(dirname))

```

videoplayer.py:

```

from PyQt5.QtCore import Qt, QUrl
from PyQt5.QtMultimedia import QMediaContent, QMediaPlayer

```

```
from PyQt5.QtMultimediaWidgets import QVideoWidget
from PyQt5.QtWidgets import QFileDialog, QHBoxLayout, QLabel, QSizePolicy,
QSlider, QStyle, QVBoxLayout, QMessageBox
from PyQt5.QtWidgets import QMainWindow, QWidget, QPushButton, QAction
from PyQt5.QtGui import QIcon
from PyQt5 import QtWidgets, QtCore
from PyQt5.QtWidgets import QApplication
import sys
import os
```

```
import constants
import operations
from statdialog import StatDialog
from mottracker import MOTTracker
```

```
class VideoPlayer(QMainWindow):
```

```
    def __init__(self, parent=None, width=1024, height=768):
```

```
        super(VideoPlayer, self).__init__(parent)
```

```
        self.setWindowTitle('SportAISystem 1.0')
```

```
        self.__filename = "
```

```
        self.mediaPlayer = QMediaPlayer(None, QMediaPlayer.VideoSurface)
```

```
        videoWidget = QVideoWidget() # widget for playing video
```

```
        # Connect mediaplayer with slots
```

```
        self.mediaPlayer.setVideoOutput(videoWidget)
```

```

self.mediaPlayer.stateChanged.connect(self.mediaStateChanged)
self.mediaPlayer.positionChanged.connect(self.positionChanged)
self.mediaPlayer.durationChanged.connect(self.durationChanged)
self.mediaPlayer.error.connect(self.handleError)

# Play button and its behaviour
self.playButton = QPushButton()
self.playButton.setEnabled(False)
self.playButton.setIcon(self.style().standardIcon(QStyle.SP_MediaPlay))
self.playButton.clicked.connect(self.playVideo)

# Stop button and its behaviour
self.stopButton = QPushButton()
self.stopButton.setEnabled(False)
self.stopButton.setIcon(self.style().standardIcon(QStyle.SP_MediaStop))
self.stopButton.clicked.connect(self.stopVideo)

# Slider to visualize the duration of video
self.positionSlider = QSlider(Qt.Horizontal)
self.positionSlider.setRange(0, 0)
self.positionSlider.sliderMoved.connect(self.setPosition)

# Create labels
self.durationLabel = QLabel()
self.durationLabel.setText('--:--')
self.durationLabel.setFixedSize(30, 20)
self.durationLabel.setAlignment(QtCore.Qt.AlignRight
QtCore.Qt.AlignVCenter)

self.totalLabel = QLabel()
self.totalLabel.setText('--:--')

```

```

self.totalLabel.setFixedSize(30, 20)
self.errorLabel = QLabel("")
self.errorLabel.setSizePolicy(QSizePolicy.Preferred, QSizePolicy.Maximum)

# Statistics button and its behaviour
self.statButton = QPushButton('Statistics')
self.statButton.setToolTip('Calculate statistics about sportsmen')
self.statButton.clicked.connect(self.pressStatistics)

# Tracker button and its behaviour
self.trackButton = QPushButton('ON')
self.trackButton.setCheckable(True)
self.trackButton.setChecked(True)
self.trackButton.setToolTip('JDE Tracker turned on')
self.trackButton.clicked.connect(self.trackHumans)

# Create open action
openAction = QAction(QIcon('Icons/play-button.png'), '&Open video', self)
openAction.setShortcut('Ctrl+O')
openAction.setStatusTip('Open video')
openAction.triggered.connect(self.openFile)

# Create live action
liveAction = QAction(QIcon('Icons/tv.png'), '&Live stream', self)
liveAction.setShortcut('Ctrl+L')
liveAction.setStatusTip('Live stream')

# Create exit action
exitAction = QAction(QIcon('Icons/log-out.png'), '&Quit', self)
exitAction.setShortcut('Ctrl+Q')

```

```

exitAction.setStatusTip('Exit application')
exitAction.triggered.connect(self.exitCall)

# Create about action
aboutAction = QAction(QIcon('Icons/info.png'), '&About program', self)
aboutAction.setShortcut('Ctrl+A')
aboutAction.setStatusTip('About program')
aboutAction.triggered.connect(self.showAboutDialog)

# Create menu bar with actions
menuBar = self.menuBar()
mediaMenu = menuBar.addMenu('&Media')
mediaMenu.addAction(openAction)
mediaMenu.addSeparator()
mediaMenu.addAction(liveAction)
mediaMenu.addSeparator()
mediaMenu.addAction(exitAction)
aboutMenu = menuBar.addMenu('&Help')
aboutMenu.addAction(aboutAction)

central_widget = QWidget(self) # widget for window contents
self.setCentralWidget(central_widget)

# Collect widgets
sliderLayout = QHBoxLayout()
sliderLayout.setContentsMargins(0, 0, 0, 0)
sliderLayout.addWidget(self.durationLabel)
sliderLayout.addWidget(self.positionSlider)
sliderLayout.addWidget(self.totalLabel)
controlLayout = QHBoxLayout()

```

```

        controlLayout.setContentsMargins(0, 0, 0, 0)
        controlLayout.addWidget(self.playButton)
        controlLayout.addWidget(self.stopButton)
        controlLayout.addItem(QtWidgets.QSpacerItem(100, 10,
QtWidgets.QSizePolicy.Expanding))
        controlLayout.addWidget(QLabel("Tracker:"))
        controlLayout.addWidget(self.trackButton)
        controlLayout.addWidget(self.statButton)
        mainLayout = QVBoxLayout()
        mainLayout.addWidget(videoWidget)
        mainLayout.addLayout(sliderLayout)
        mainLayout.addLayout(controlLayout)
        mainLayout.addWidget(self.errorLabel)

# Set widget to contain window contents
central_widget.setLayout(mainLayout)
self.resize(width, height)

def openFile(self):
    self.__filename, _ = QFileDialog.getOpenFileName(self, 'Open videos',
os.getcwd(),
'Videos (*.avi *.wmv)')
    if self.__filename:
        if self.trackButton.isChecked():
            QApplication.setOverrideCursor(Qt.WaitCursor)
            self.errorLabel.setText('Waiting for tracking the video')
            jde = MOTTracker(self.__filename, constants.GPU_NUMBER)
            jde.trackVideo()
            QApplication.restoreOverrideCursor()

```

```

        self.errorLabel.setText("")

        tracked_video      =      os.path.join(constants.RESULTS_FOLDER,
self.__filename)

self.mediaPlayer.setMedia(QMediaContent(QUrl.fromLocalFile(self.__filename)))
    else:
        self.errorLabel.setText("")

self.mediaPlayer.setMedia(QMediaContent(QUrl.fromLocalFile(self.__filename)))
    self.playButton.setEnabled(True)
    self.stopButton.setEnabled(True)
    self.mediaPlayer.pause()


def exitCall(self):
    sys.exit(0)


def playVideo(self):
    if self.mediaPlayer.state() == QMediaPlayer.PlayingState:
        self.mediaPlayer.pause()
    else:
        self.mediaPlayer.play()


def stopVideo(self):
    self.mediaPlayer.stop()


def pressStatistics(self):

```

```

markup = os.path.join(constants.RESULTS_FOLDER,
os.path.splitext(os.path.basename(self.__filename))[0]+'.txt')
if os.path.exists(markup):
    st = StatDialog(markup=markup)
    st.exec()
else:
    self.errorLabel.setText('Cannot find markup file! Please load marked video
or mark it using Tracker before')

```

```

def trackHumans(self):
    if self.trackButton.isChecked():
        self.trackButton.setText('ON')
    else:
        self.trackButton.setText('OFF')

```

```

def showAboutDialog(self):
    msg = QMessageBox()
    msg.setIcon(QMessageBox.Information)
    msg.setText('SportAISystem 1.0')
    msg.setInformativeText('Contacts: sportaisystemyar@gmail.com')
    msg.setWindowTitle('About program')
    msg.setStandardButtons(QMessageBox.Ok)
    msg.exec()

```

```

def mediaStateChanged(self, state):
    if self.mediaPlayer.state() == QMediaPlayer.PlayingState:
        self.playButton.setIcon(self.style().standardIcon(QStyle.SP_MediaPause))

```



```

else:
    self.playButton.setIcon(self.style().standardIcon(QStyle.SP_MediaPlay))

def positionChanged(self, position):
    self.positionSlider.setValue(position)
    if position >= 0:
        self.durationLabel.setText(operations.time_string(position))

def durationChanged(self, duration):
    self.positionSlider.setRange(0, duration)
    if duration >= 0:
        self.totalLabel.setText(operations.time_string(duration))

def setPosition(self, position):
    self.mediaPlayer.setPosition(position)

def handleError(self):
    self.playButton.setEnabled(False)
    self.errorLabel.setText('Error: ' + self.mediaPlayer.errorString())

```

videodataloader.py:

```

import numpy as np
import cv2

```

```
import constants
```

```
class VideoDataLoader:
```

```
    def __init__(self, path, img_size=(1088, 608)):
        self.cap = cv2.VideoCapture(path)
        self.frame_rate = int(round(self.cap.get(cv2.CAP_PROP_FPS)))
        self.vw = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        self.vh = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        self.width = img_size[0]
        self.height = img_size[1]
        self.count = 0
        self.w, self.h = self.__getFrameSize(self.vw, self.vh, self.width, self.height)
```

```
    def __iter__(self):
        self.count = -1
        return self
```

```
    def __next__(self):
        self.count += 1
        if self.count == len(self):
            raise StopIteration
        res, img0 = self.cap.read() # BGR
        while img0 is None:
            print('Failed to load frame {:d}'.format(self.count))
            self.count += 1
        if self.count >= len(self):
```

```

        raise StopIteration

    self.cap.set(cv2.CAP_PROP_POS_FRAMES, self.count)

    res, img0 = self.cap.read() # BGR
    img0 = cv2.resize(img0, (self.w, self.h), interpolation=cv2.INTER_AREA)
    # Padded resize
    img = self.__getPaddedRectangularFrame(img0)
    # Normalize RGB
    img = img[:, :, ::-1].transpose(2, 0, 1)
    img = np.ascontiguousarray(img, dtype=np.float32)
    img /= 255.0
    return self.count, img, img0

def __len__(self):
    n_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT)) # number
of files
    return n_frames

def __getFrameSize(self, vw, vh, dw, dh):
    wa, ha = float(dw)/vw, float(dh)/vh
    a = min(wa, ha)
    return int(vw*a), int(vh*a)

def __getPaddedRectangularFrame(self, img):
    shape = img.shape[:2] # shape = [height, width]
    width, height = constants.OUTPUT_FRAME_SIZE
    ratio = min(float(height)/shape[0], float(width)/shape[1])

```

```

    new_shape = (round(shape[1]*ratio), round(shape[0]*ratio)) # new_shape =
[width, height]
    dw = (width - new_shape[0]) / 2 # width padding
    dh = (height - new_shape[1]) / 2 # height padding
    top, bottom = round(dh - 0.1), round(dh + 0.1)
    left, right = round(dw - 0.1), round(dw + 0.1)
    img = cv2.resize(img, new_shape, interpolation=cv2.INTER_AREA) #
resized image, no border
    # Padded rectangular image
    img = cv2.copyMakeBorder(img, top, bottom, left, right,
cv2.BORDER_CONSTANT, value=(127.5, 127.5, 127.5))
    return img

```

mottracker.py:

```

import os
import logging
import argparse
import cv2
import torch
import numpy as np

import constants
import operations
from videodataloader import VideoDataLoader

from utils.log import logger
from utils.timer import Timer

```

```
from tracker.multitracker import JDETracker
```

```
class MOTTracker:
```

```
    def __init__(self, input_video, gpu_number):
        os.makedirs(constants.RESULTS_FOLDER, exist_ok=True)
        self.__video = input_video
        self.__gpu = gpu_number
        basename = os.path.splitext(os.path.basename(self.__video))[0]
        self.__markupfile = os.path.join(constants.RESULTS_FOLDER,
str(basename)+' .txt')
        self.__markedvideo = os.path.join(constants.RESULTS_FOLDER,
str(basename)+' .avi')
        self.__framedir = os.path.join(constants.RESULTS_FOLDER, str(basename))
        os.makedirs(self.__framedir, exist_ok=True)

    def __adjustTracker(self):
        logger.setLevel(logging.INFO)
        logger.info('Adjust tracker...')
        self.__dataloader = VideoDataLoader(self.__video,
constants.OUTPUT_FRAME_SIZE)
        logger.info('Video was loaded!')
        self.__framerate = self.__dataloader.frame_rate
        logger.info('FPS: \t {}'.format(self.__framerate))
        logger.info('Frames: \t {}'.format(len(self.__dataloader)))
        pass
```

```

def __makeMarkedVideo(self):
    logger.info('Making video...')
    cmd_str = 'ffmpeg -f image2 -i {}/%05d.jpg -c:v copy
{}'.format(self.__framedir, self.__markedvideo)
    os.system(cmd_str)
    logger.info('Video finished!')

```

```

def __trackObjects(self):
    logger.info('Start tracking...')
    try:
        self.__evalSeq()
    except Exception as e:
        logger.info(e)
    logger.info('Tracking finished!')

```

```

def trackVideo(self):
    self.__adjustTracker()
    self.__trackObjects()
    self.__makeMarkedVideo()

```

```

def __writeTrackingResults(self, results):
    with open(self.__markupfile, 'w') as f:
        for frame_id, tlwhs, track_ids in results:
            for tlwh, track_id in zip(tlwhs, track_ids):
                if track_id < 0:
                    continue
                x1, y1, w, h = tlwh

```

```

        x2, y2 = x1 + w, y1 + h
        line = '{frame},{id},{x1},{y1},{w},{h}\n'.format(frame=frame_id,
id=track_id,
                                x1=x1, y1=y1, x2=x2, y2=y2, w=w,
h=h)
        f.write(line)
        logger.info('Save results to {}'.format(self.__markupfile))

```

```

def __collectArgumentParserParams(self):
    parser.add_argument('--cfg',                                type=str,
default=constants.TRACKER_CONFIG)
    parser.add_argument('--weights',                            type=str,
default=constants.TRACKER_WEIGHTS)
    parser.add_argument('--img-size',                           type=int,
default=constants.OUTPUT_FRAME_SIZE)
    parser.add_argument('--iou-thres',                          type=float,
default=constants.IOU_THRESHOLD)
    parser.add_argument('--conf-thres',                         type=float,
default=constants.CONFIDENCE_THRESHOLD)
    parser.add_argument('--nms-thres',                          type=float,
default=constants.SUPPRESSION_THRESHOLD)
    parser.add_argument('--track-buffer',                       type=int,
default=constants.TRACKING_BUFFER)
    res = parser.parse_args()
    return res

```

```

def __evalSeq(self):
    os.environ['CUDA_VISIBLE_DEVICES'] = str(self.__gpu)

```

```

logger.info('GPU id: \t {}'.format(self.__gpu))
argument_parser = self.__collectArgumentParserParams()
tracker = JDETracker(argument_parser, frame_rate=self.__framerate)
timer = Timer()
results = []
frame_id = 0
for path, img, img0 in self.__dataloader:
    if frame_id % 10 == 0:
        logger.info('Processing frame {} ({:.2f} fps)'.format(frame_id,
1./max(1e-5, timer.average_time)))

        # run tracking
        timer.tic()
        blob = torch.from_numpy(img).cuda().unsqueeze(0)
        online_targets = tracker.update(blob, img0)
        online_tlwhs, online_ids = [], []
        for t in online_targets:
            tlwh, tid = t.tlwh, t.track_id
            if (tlwh[2]*tlwh[3] > constants.MIN_BOX_AREA) and (tlwh[2] /
tlwh[3] <= 1.6):
                online_tlwhs.append(tlwh)
                online_ids.append(tid)
        timer.toc()
        # save results
        results.append((frame_id, online_tlwhs, online_ids))
        online_img = self.__plotTracking(img0, online_tlwhs, online_ids,
frame_id=frame_id)
        cv2.imwrite(os.path.join(self.__framedir, '{:05d}.jpg'.format(frame_id)),
online_img)
        frame_id += 1
self.__writeTrackingResults(results)

```



```

return frame_id, timer.average_time, timer.calls

def __plotTracking(self, img, tlwhs, obj_ids, frame_id=0, ids2=None):
    img = np.ascontiguousarray(np.copy(img))
    text_scale = max(1, img.shape[1]/1500.0)
    text_thickness = 1 if text_scale > 1.1 else 1
    line_thickness = max(1, int(img.shape[1]/500.0))
    cv2.putText(img, 'frame: {} humans: {}'.format(frame_id, len(tlwhs)), (0,
int(15*text_scale)),
                cv2.FONT_HERSHEY_PLAIN, text_scale, color=(0, 0, 255),
thickness=2)

    for i, tlwh in enumerate(tlwhs):
        x1, y1, w, h = tlwh
        bbox = tuple(map(int, (x1, y1, x1+w, y1+h)))
        obj_id = int(obj_ids[i])
        id_text = '{}'.format(int(obj_id))
        if ids2 is not None:
            id_text = id_text + ', {}'.format(int(ids2[i]))
        cv2.rectangle(img, bbox[0:2], bbox[2:4],
color=operations.get_color(abs(obj_id)), thickness=line_thickness)
        cv2.putText(img, id_text, (bbox[0], bbox[1]+30),
cv2.FONT_HERSHEY_PLAIN, text_scale,
                    color=(0, 0, 255), thickness=text_thickness)

    return img

def init_argparse():
    """
    Initializes argparse

```

```

'''
    parser = argparse.ArgumentParser(description='Background extraction from the
video')
    parser.add_argument(
        '--input_video',
        nargs='?',
        help='Video to track objects',
        required=True,
        type=str)
    parser.add_argument(
        '--gpu',
        nargs='?',
        help='Number of GPU to implement tracking',
        default=constants.GPU_NUMBER,
        type=int)
    return parser

if __name__ == '__main__':
    parser = init_argparse()
    # Extract arguments of script
    args = parser.parse_args()
    # Launch tracker
    jde = MOTTracker(args.input_video, args.gpu)
    jde.trackVideo()

```

main.py:

```
from PyQt5.QtWidgets import QApplication
import sys
from PyQt5.QtGui import QIcon
from videoplayer import VideoPlayer
```

```
if __name__ == '__main__':
    app = QApplication(sys.argv)
    app.setWindowIcon(QIcon('Icons/volleyball-player.png'))
    player = VideoPlayer(width=800, height=600)
    player.show()
    sys.exit(app.exec_())
```

Приложение Б. Описание программы

Программа была написана на языке Python с динамической типизацией данных. Алгоритм обнаружения и отслеживания объектов был реализован с помощью библиотеки глубокого машинного обучения PyTorch. При разработке графического интерфейса программного комплекса были использованы инструменты фреймворка PyQt. Помимо этого, в ходе разработки программного комплекса использовались инструменты библиотек NumPy, SciPy, OpenCV и Matplotlib. Контроль над рабочими версиями проекта осуществлялся с помощью системы Git.

Для работы программы требуется операционная система Linux. Для запуска приложения в операционной среде Windows, требуется установка программного обеспечения Docker для создания среды с поддержкой контейнеризации и сборки соответствующего контейнера с помощью Dockerfile.

Как видно из рисунка 11, на главном экране программного комплекса расположены следующие элементы: панель меню с вкладками Media и Help, область для воспроизведения видеофайлов, а также вспомогательные кнопки для управления видеоплеером, подсчета статистики и включения/выключения системы отслеживания спортсменов на игровой площадке. Панель меню, помимо вкладки с информацией о программе и контактных данных разработчиков, позволяет загрузить необходимый видеофайл в формате avi, mp4 или wmv в область воспроизведения видеоданных. При включенном режиме отслеживания спортсменов на игровой площадке в область воспроизведения видеофайлов загружается видеопоследовательность, которая перед этим была обработана алгоритмом трекинга JDE. Режим работы приложения с выключенным JDE трекером дает возможность воспроизвести любую видеопоследовательность, будь то видеофайл, неразмеченный JDE трекером, или же уже предобработанный им ранее, при предыдущих запусках программного комплекса с включенным режимом обнаружения и отслеживания спортсменов на игровой площадке.

Окно выбора видеопоследовательности для ее загрузки в область воспроизведения видеофайла показано на рисунке 19.

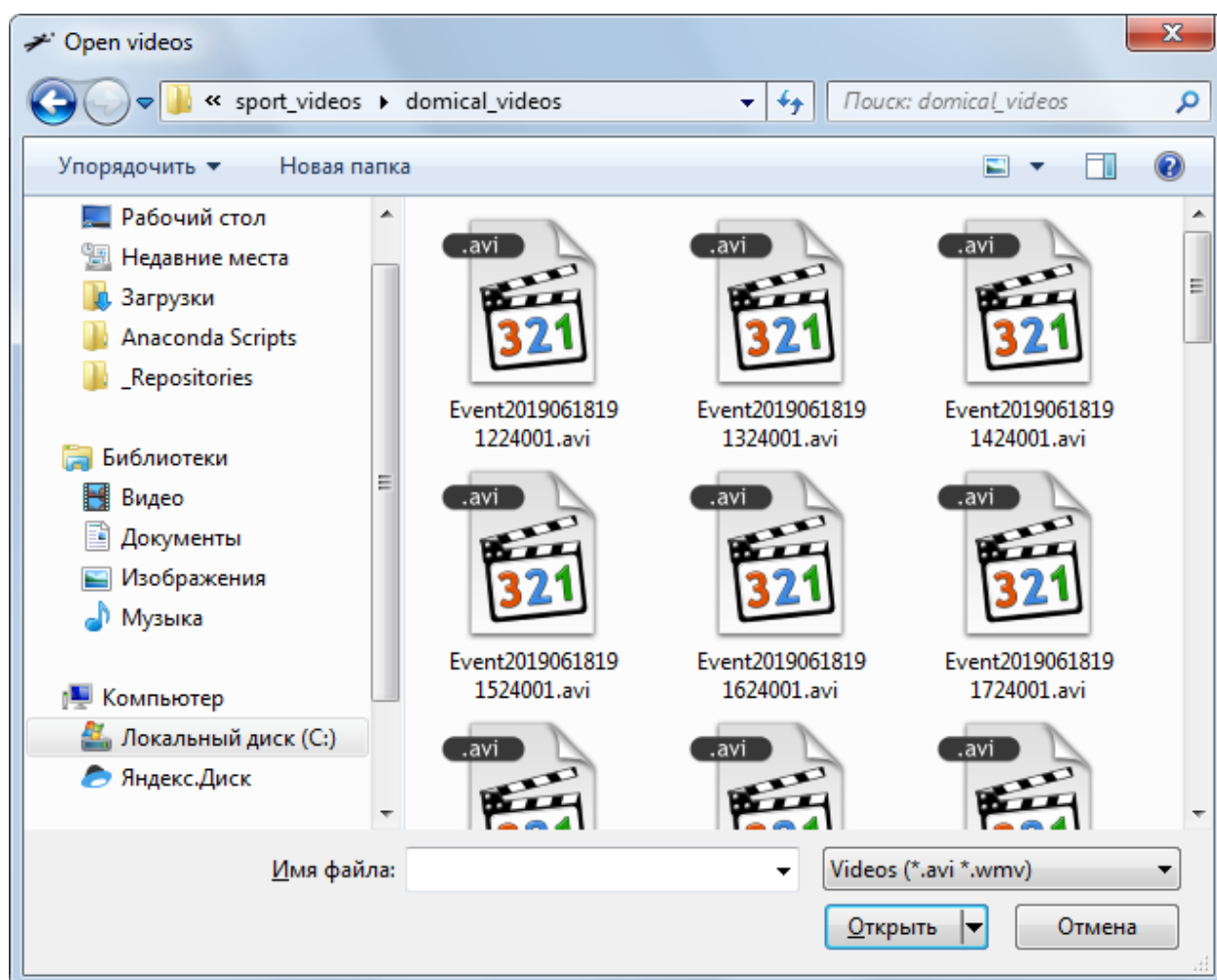


Рисунок 19 – Окно для выбора видеофайла

При воспроизведении файла, размеченного JDE-трекером, как показано на рисунке 20, в левом верхнем углу области отображения видеоданных показывается информация о порядковом номере обработанного JDE трекером кадра и количестве отслеживаемых в данный момент спортсменов на игровой площадке. В том случае, если система отслеживания спортсменов на игровой площадке отключена, какая-либо дополнительная информация в левом верхнем углу экрана приложения отсутствует, как это показано на рисунке 21.

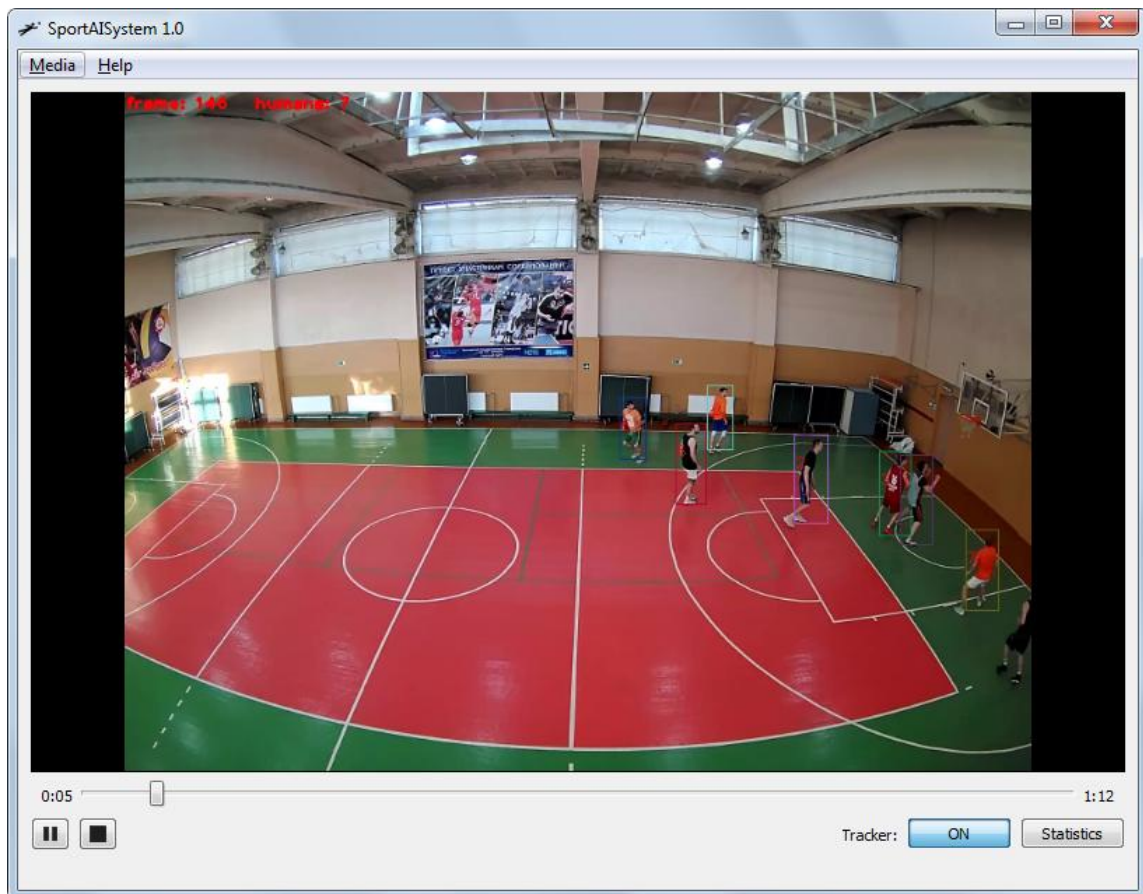


Рисунок 20 – Воспроизведение размеченного с помощью JDE трекера

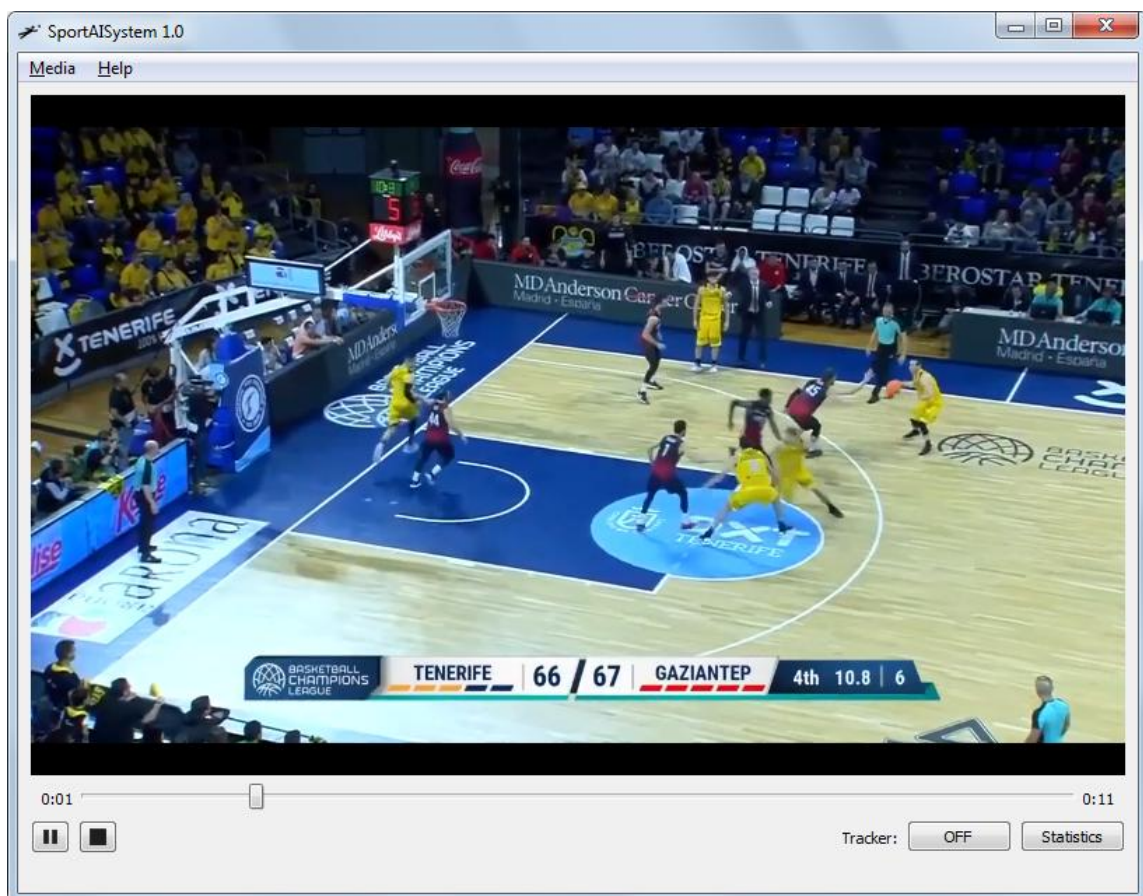


Рисунок 21 – Воспроизведение размеченного с помощью JDE трекера

Программный комплекс позволяет также подсчитать статистические показатели, такие как количество взаимодействий с другими спортсменами, построение траекторий передвижения по игровой площадке, расчет пройденного расстояния и построение тепловой карты перемещений. Статистические показатели, которые нужно подсчитать для выбранной ранее видеопоследовательности, можно выбрать в специальном окне после нажатия кнопки Statistics на главном экране приложения.

Окно с выбором статистических показателей содержит три области (Sport Statistics, Options и Sending), а также кнопку для выполнения расчетов. Выбранные пользователем статистические показатели можно подсчитать как для всех спортсменов, так и для конкретного игрока в частности. Результаты расчетов сохраняются в виде изображений со статистическими показателями, визуализированными в виде тепловой карты, а также горизонтальной и вертикальной столбчатой диаграммы. Сгенерированные изображения с посчитанной статистикой могут быть как сохранены на компьютере в выбранной локальной директории, так и отправлены на адрес электронной почты, вбитой пользователем в соответствующей строке ввода данных. Скриншот окна для выбора подсчитываемых статистических показателей изображен на рисунке 22.

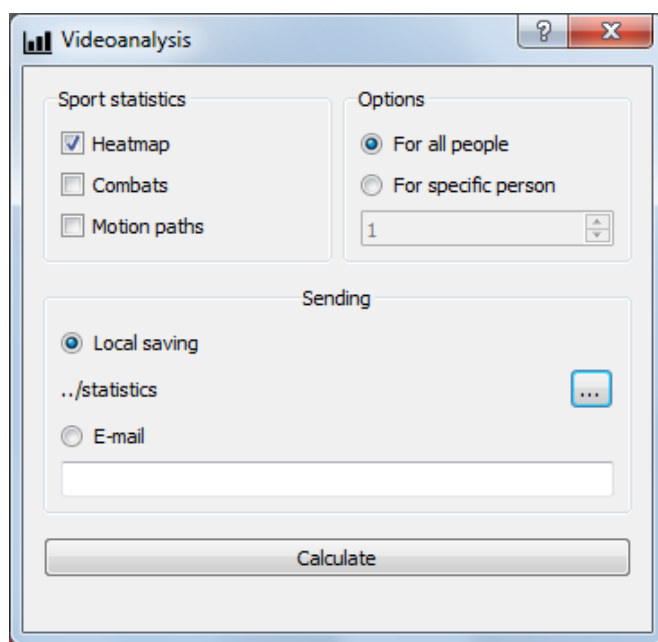


Рисунок 22 – Окно для выбора подсчитываемых статистических показателей