Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 21.Б10-мм

Оптимизация разметки ground truth data пешеходов

Житнухина Мария Андреевна

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры системного программирования, к.т.н., Ю. В. Литвинов

Консультант:
инженер-программист, АО «Кама», М. С. Осечкина

Санкт-Петербург
2023

Оглавление

Введение

Появление автопилотов и систем помощи водителю автомобильных средств делают вождение более безопасным, комфортным.

Усовершенствованная система помощи водителю (ADAS) (англ. Advanced driver-assistance systems) помогает водителю управлять машиной, парковать ее, предупреждает о возможных столкновениях и прочее. ADAS руководствуется данными, полученными с датчиков, и через интерфейс сообщает рекомендации. Одна из причин высокой смертности пешеходов на дороге — слишком позднее их обнаружение водителем. В том числе такие системы помощи водителю могут, основываясь на данных с камеры, отслеживать пешеходов и предупреждать человека у руля о необходимости замедлиться.

Для корректной работы ADAS необходима точная модель мира. Для этого необходимо уметь максимально точно распознавать и отслеживать в пространстве окружающие объекты в режиме реального времени. Как правило, беспилотные системы ориентируются в пространстве с помощью радара, gps, лидара или камеры. Преимущество использования камеры заключается в более низкой стоимости по сравнению с лидаром. Эталонные данные (англ. ground truth data) в данном контексте — верная разметка данных с камеры: полос дороги, передвижения пешеходов, светофоров и прочего. Ручная разметка занимает много времени, но эти данные необходимы для тестирования ADAS, и поэтому нужны инструменты для автоматизации процесса.

При создании эталонных данных не ставятся ограничения на скорость работы, но важна точность, поэтому используемый алгоритм не обязан быть применим для трекинга в реальном времени.

Для автоматизации разметки пешеходов с целью эталонных данных подойдут и алгоритмы трекинга, в том числе и те, которые не могут работать в реальном времени. Алгоритмы слежения используют в качестве входных данных распознанный объект. При исчезновении объекта (пешехода) из поля зрения или при естественных помехах (таких как снег, дождь) существующие алгоритмы не всегда выдают приемлемый

результат в рамках поставленной перед ними задачи, а так же многие алгоритмы предназначены для статичной камеры, тогда как автомобиль движется с большой скоростью.

Не существует алгоритма, работающего одинаково хорошо при всевозможных входных данных. Существующие алгоритмы, описанные в статьях [3, 5, 6, 10] лучше приспособлены для других задач, отличных от трекинга пешеходов по данным с камеры, закрепленной на автомобиле. У автоматизации разметки пешеходов по данным с видеокамеры есть своя специфика (скорость движения камеры, отслеживаемые объекты меняют свое положение, могут быть плохие погодные условия), поэтому для этой задачи стоит использовать специализированный трекер.

1. Постановка задачи

Целью работы является оптимизация отслеживания пешеходов по данным с камеры для разметки эталонных данных (англ. ground truth data). Трекинг для разметки в данной работе необходимо автоматизировать, и он должен быть как можно более точным.

Для выполнения работы были поставлены следующие задачи:

- Проанализировать текущие подходы к трекингу пешеходов.
- Выбрать и реализовать алгоритм трекинга.
- Улучшить работу алгоритма при временном исчезновении пешеходов из поля зрения камеры и затрудняющих трекинг осадках.
- Создать инструмент разметки на основе созданного алгоритма трекинга.
- Провести апробацию инструмента.

2. Обзор

2.1. Существующие реализации алгоритмов трекинга

2.1.1. A pedestrian tracking algorithm based on background unrelated head detection

Алгоритм, изложенный в статье «A pedestrian tracking algorithm based on background unrelated head detection» [10] состоит из нескольких частей. Сперва по первым n кадрам выделяется фон. Затем сегментируется фон, пикселям, соответствующим фону, присваивается значение 255 (белый цвет). С помощью признаков Хаара (Haar-like feature) и алгоритма Adaboost происходит детекция головы. Признак Хаара представляет собой разность сумм пикселей внутри двух смежных прямоугольников. Прямоугольники могут быть любого размера и в любом месте изображения. К примеру, общим признаком для большинства лиц является то, что область, содержащая глаза, темнее области под глазами, как показано на рис. 1.

Adaboost — алгоритм машинного обучения, который строит улучшенный классификатор по нескольким другим классификаторам. Он используется, чтобы подобрать наилучшие признаки Хаара для детекции лиц. Для этого был использован датасет лиц 16×16 с отсутствующим фоном. В статье предлагается описание трекинга нескольких пешеходов одновременно с помощью построения цепочек маршрутов. Каждому маршруту присваивается степень достоверности равная 1, которая в ходе алгоритма может увеличиваться и уменьшаться в соответствии с функцией оценки, и пешеход перестает отслеживаться, если степень достоверности оказывается равна 0. Алгоритм отличается высокой устойчивостью к перекрыванию пешеходами друг друга, а так же при выходе их из зоны видимости, но не предполагает движения камеры, что не позволяет его использовать для создания ground truth data. Человек в белой шапке (совпадающей по яркости со значением, выставленным в области фона) может быть плохо распознан. Также

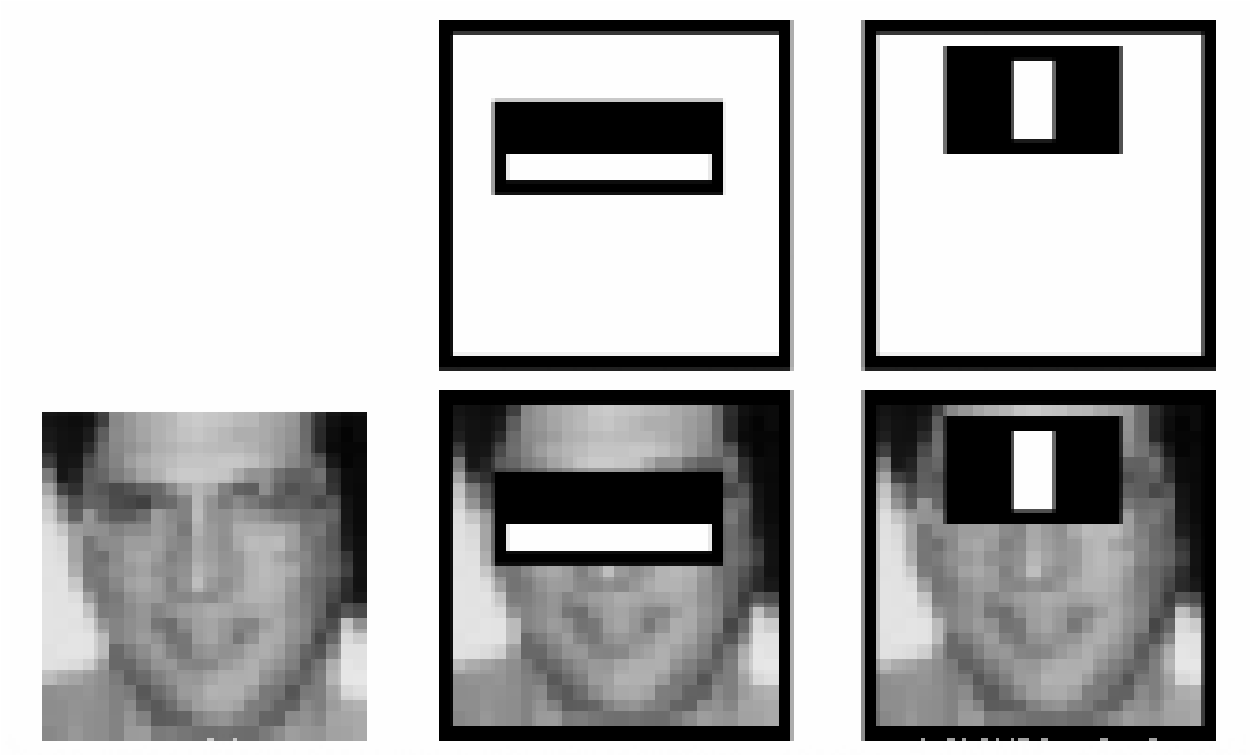

Рис. 1: Пример признаков Хаара

невозможно отслеживать человека, у которого не видно лицо.

В качестве метрики для оценки результата была использована точность, вычисляемая как отношение количества верных указываний местоположения пешехода с точностью до допустимой ошибки к общему количеству кадров. Ее значение в помещении было равно 80.31, в коридоре – 89.38, на улице – 82.15.

2.1.2. Single Camera Pedestrian Tracking

Алгоритм, описанный в работе «Single Camera Pedestrian Tracking» [3] компенсирует движение камеры, находя билинейное преобразование изображения с наименьшей ошибкой. После этого видео выглядит, словно камера неподвижна. По набору кадров без объекта можно определить фоновое изображение, затем удалить фон. Получившееся изображение можно представить как набор компонент связности (объектов). Необходимо научиться сопоставлять компонентам связности с одного кадра компоненты связности со следующего (построить двудольный граф на компонентах связности с наименьшей

функцией ошибки). При этом компонента связности от кадра к кадру может либо объединиться с другой, либо разделиться на несколько, но не одновременно. Так мы сможем отслеживать любую компоненту связности, которой соответствует объект.

Недостатком данного алгоритма является то, что он не может отслеживать неподвижный объект, а так же то, что билинейное преобразование кадра имеет свою погрешность, которая будет накапливаться. При быстром движении камеры фоны на последовательных кадрах будут отличаться достаточно сильно, чтобы преобразование заметно ухудшало качество фото. В данном проекте это является серьезным недостатком, поскольку скорость машины бывает велика, а видео длится дольше, чем несколько кадров.

В качестве метрик для оценки результата были использованы полнота (recall) и точность (precision). Значение первого — 0.67, второго — 0.84. Пешеход считается верно распознанным на кадре, если прямоугольник, содержащий пешехода, пересекается с эталоном хотя бы наполовину по площади.

2.1.3. An improved motion pedestrian tracking algorithm based on CamShift

Алгоритм из статьи «An improved motion pedestrian tracking algorithm based on CamShift»[1] является модификацией известного алгоритма CamShift[2]. CamShift (Continuous Adaptive Mean Shift) — усовершенствованная версия MeanShift. На каждой итерации применяется MeanShift и корректируется окно поиска согласно сдвигу и вращению цели. В свою очередь, MeanShift получает на вход набор признаков изображения, которое ищется на следующем кадре, область поиска, в многомерном пространстве с некоторой метрикой ищет центр масс особых точек в области поиска и возвращает вектор сдвига к наиболее вероятному образцу (центру масс).

[1] https://dl.acm.org/doi/10.1145/3366194.3366265 (дата обращения: 23.11.2022)

[2] Более подробно о CamShift и Meanshift: https://docs.opencv.org/4.x/d7/d00/tutorial_meanshift.html (дата обращения: 23.11.2022)

Как правило, фильтр Калмана применяется для фильтрации зашумленных данных с датчиков. Он делится на два этапа: прогнозирования и коррекции. В данном случае значения фильтруемой функции являются векторами, у которых координатами являются:

- координата центра объекта по оси x;
- координата центра объекта по оси y;
- скорость по оси x;
- скорость по оси y.

Так же то, что мы получаем с камеры — лишь проекция на плоскость матрицы камеры. Чтобы получить координаты объекта в глобальном пространстве, придется перепроецировать объект с использованием матрицы наблюдения, что влечет за собой погрешность. Это мешает использовать данный алгоритм для создания эталонных данных.

Благодаря применению фильтра Калмана алгоритм стабильнее работает при перекрывании отслеживаемого объекта, поскольку тот умеет предсказывать наиболее вероятное положение цели и ее скорости на следующем кадре. Однако данный алгоритм чувствителен к цвету объекта и освещению, что может быть препятствием при трекинге пешехода на улице. Помимо этого, в статье не указаны метрики, по которым производилась оценка алгоритма, и не указаны тестовые данные.

2.1.4. Tracking Algorithm of Multiple Pedestrians Based on Particle Filters in Video Sequences

В алгоритме, описанном в статье «Tracking Algorithm of Multiple Pedestrians Based on Particle Filters in Video Sequences»[3], сперва делается предположение о возможных следующих положениях объекта (координаты центра прямоугольника, содержащего объект, его ширина и

[3] https://dl.acm.org/doi/10.1145/3366194.3366265 (дата обращения: 23.11.2022)

высота). Затем извлекаются признаки объекта: HSV (Hue, Saturation, Value) и LBP (Local Binary Patterns) гистограммы. Используются одновременно два вида признаков для борьбы с шумами (например, интерференцией), а так же чтобы меньше терять информацию в связи со сложным фоном и прочими факторами.

Объекты перемещаются независимо, и для каждого из них по кадру $t-1$ делается предсказание наблюдаемой величины для кадра t на основе Гауссова распределения. Затем вычисляется схожесть состояний предполагаемых объектов и их ожидаемых состояний по каждому из двух признаков, а затем по совокупности, а так же пересчитываются веса признаков. На основе этих данных вычисляется, какое положение объекта ближе к ожидаемому.

Если объект исчез и не был обнаружен на протяжении трех кадров, то он перестает отслеживаться. Если объект появился на протяжении трех последовательных кадров, то он считается новым объектом. Данный алгоритм может справиться с частичным перекрыванием пешеходов, в более сложных ситуациях объект может начать считаться исчезнувшим.

В качестве метрики была указана среднеквадратическая ошибка положения (отклонения координат найденного объекта от действительных). Ее значение лежало в промежутке от 3.18 до 10.33 пикселей для различных пешеходов.

2.2. Возможные направления исследований

Optical flow — алгоритм, сравнивающий последовательные кадры и строящий для каждого пикселя вектор перемещения, то есть строящий для каждого кадра матрицу двумерных векторов. Перед построением делаются следующие допущения:

1. Яркость пикселей объекта не меняется.

2. Вектора перемещений соседних векторов достаточно близки.

$I(x, y, t)$ — яркость пикселя (x, y) в момент t.

По предположению 1 $I(x, y, t) = I(x + dx, y + dy, t + dt)$. Разложив по Тейлору и поделив на dt, получим приближенно $I'_x \frac{dx}{dt} + I'_y \frac{dy}{dt} + I'_t = 0$ — уравнение Optical flow, где $(\frac{dx}{dt}, \frac{dy}{dt})$ — искомый вектор. Уравнение одно, а неизвестных две, но существуют методы найти этот вектор. Один из них — метод Люкаса-Канаде[4], предполагающий, что в квадрате 3×3 вектора перемещений одинаковы. Недостатком этого алгоритма является основополагающее предположение про неизменность яркости. В условиях улицы освещение изменяется, к примеру, из-за мерцаний и света фар. Также алгоритм будет плохо отслеживать при больших скоростях камеры, поскольку использует разложение до линейных слагаемых. Но алгоритм может помочь оценить скорость автомобиля, что может помочь предсказать положение пешехода.

2.3. Существующие инструменты разметки

Одним из самых часто применяемых среди существующих инструментов разметки является Ground Truth Labeler[5]. Он позволяет автоматизировать разметку, дает выбрать один из нескольких алгоритмов трекинга. Из них лишь один специализируется на отслеживании пешеходов — ACF People detection. Остальные основываются на данных с лидара, специализируются на трекинге автомобиля или работают на основе интерполяции. При трекинге с использованием интерполяции раз в несколько кадров нужно указывать положение объекта, чтобы можно было проинтерполировать по его координатам и получить траекторию с хорошей точностью. Данный инструмент достаточно часто требует вмешательства человека при быстрой езде и зашумленном видео.

Еще один из примеров инструментов разметки — Label Studio[6]. Для разметки в нем необходимо частое вмешательство человека, так как инструмент предлагает разметку с помощью интерполяции.

[4]Более подробно: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html (дата обращения: 23.11.2022)

[5]Официальный сайт: https://www.mathworks.com/help/driving/ug/get-started-with-the-ground-truth-labeler.html (дата обращения: 28.11.2022)

[6]Официальный сайт: https://labelstud.io/ (дата обращения: 07.12.2022)

2.4. Используемые алгоритмы трекинга

Для текущей реализации используется ряд трекеров на основе трекеров из стандартной библиотеки OpenCV: CSRT, KCF, TLD, Optical Flow. Не были использованы остальные по причине того, что на используемом датасете выдавали менее точные результаты. [11].

2.4.1. CSRT

CSRT (Channel and Spatial Reliability Tracker) [2] — трекер на основе корреляционного фильтра [8]. В качестве признаков выделяются гистограммы ориентированных градиентов и цветов (color-named). Сперва алгоритм находит несколько вариантов, где может находиться объект на следующем кадре, находя прямоугольники с близкими признаками. После этого можно применить корреляционный фильтр к изображению, чтобы проверить гипотезы и выбрать самых похожий на шаблон фрагмент изображения.

2.4.2. KCF

В алгоритме KCF (Kernelized Correlation Filter) [7] сперва изображение переводится в оттенки серого. В качестве признаков выделяются гистограммы ориентированных градиентов. Делается несколько предположений о возможном положении объекта на следующем кадре, а затем корреляционный фильтр выделяет среди них лучшего кандидата. Одним из преимуществ этого трекера является то, что он может сообщить о потере объекта. Недостатком является то, что после потери объекта он не может его обнаружить и останавливается.

2.4.3. TLD

TLD (Tracking Learning Detection) [9] — трекер, состоящий из трех частей: трекера на основе трекера MedianFlow, обучения и детекции. Детектор реализован путем каскадирования трех бинарных классификаторов: классификатора дисперсии, классификатора ансамбль и клас-

сификатора ближайшего соседа. Классификатор дисперсии отклоняет изображение с уровнем серого, сильно отличающимся от модели объекта. Модель объекта — структура, представляющая объект и окружение. Ансамбль генерирует попиксельное сравнение, и в зависимости от того, насколько отличаются изображения, выдает результат, а последний вычисляет схожесть изображения с моделью, и если полагает изображения схожими, то выдает положительный результат. Трекер отслеживает объект, а детектор при необходимости корректирует трекер. Обучающая часть работает согласно стратегии P-N learning, при нарушении непрерывности траектории объекта корректируются трекер и детектор.

3. Реализация

3.1. OpenCV

Для работы с изображениями была выбрана библиотека OpenCV как наиболее обширная, поддерживаемая и стабильная. Является бесплатной, активно поддерживается разработчиками, имеет большое сообщество. Аналогом среди библиотек для работы с изображениями является библиотека SimpleCV, построенная на основе OpenCV, но она не обновлялась с 2014 года. Microsoft Computer Vision API, Amazon Rekognition, Google Cloud Vision API — платные библиотеки, ориентированные на распознание объектов, что не является целью данной учебной практики. Была выбрана последняя версия библиотеки OpenCV — 4.7.0.

3.2. C++

Использование OpenCV возможно на ряде языков. Среди них Python, Java, C#, Matlab, C, C++. Был выбран язык C++, так как на нем ведется разработка в АО «Кама», а так же его производительность достаточно высока.

3.3. Qt

Для создания промежуточного пользовательского интерфейса для упрощения разработки был выбран фреймворк Qt, поскольку позволяет быстро создавать кроссплатформенный функциональный интерфейс. Среди аналогов можно выделить FLTX, wxDev-C++. Среди преимуществ Qt большее количество пользователей, чем у прочих библиотек для создания интерфейса. Это ускоряет его создание, поскольку упрощает поиск ответов и на возникающие в ходе разработки вопросы.

3.4. CVAT

CVAT (Computer Vision Annotation Tool) — инструмент для разметки и аннотации видео и фото. Это крупный проект с открытым исходным кодом, который активно разрабатывается. Также у этого инструмента есть обширная документация[7] и возможность встраивать трекер с вычислением на сервере на основе serverless функций. Был выбран именно этот инструмент для интеграции, поскольку только CVAT среди инструментов разметки обладает приведенными выше характеристиками.

3.5. Nuclio

В CVAT используется платформа Nuclio[8] для разворачивания serverless функций в nuclio-контейнере, которые используются для запуска ресурсоемкого кода на сервере.

3.6. cpprestsdk

Cpprestsdk — C++-библиотека для написания серверов, клиентов, взаимодействующих через http-запросы. Среди аналогов можно выделить библиотеки Qt, POCO, libcurl. Была выбрана именно cpprestsdk для реализации серверной части трекера по причине простоты и лаконичности синтаксиса.

3.7. Алгоритм трекинга

Для текущей реализации используется ряд трекеров на основе трекеров из стандартной библиотеки OpenCV: CSRT, KCF, TLD, Optical Flow Tracker с использованием особых точек. Они были выбраны как наиболее стабильные среди библиотечных.

[7]Документация CVAT: https://opencv.github.io/cvat//docs/ (дата обращения: 08.05.2023)
[8]Официальный сайт Nuclio: https://nuclio.io/ (дата обращения: 08.05.2023)

3.7.1. Использование приведенных выше алгоритмов

Для нового трекера используются перечисленные выше алгоритмы трекинга. Каждый из них был обернут в класс, унаследованный от класса Tracker. Этот класс хранит в себе последний отслеживаемый кадр и координаты прямоугольника, содержащего пешехода.

Реализован класс MyTracker, так же унаследованный от класса Tracker. При вызове метода для получения следующего положения пешехода каждый из трекеров возвращает координаты следующей позиции отслеживаемого объекта. Берется центр масс результатов, и он считается текущим положением пешехода. Таким образом, если лишь один из трекеров начнет выдавать некорректный результат, то остальные скомпенсируют его ошибку. Поскольку трекеры могут потерять объект или начать отслеживать фон, то раз в 10 кадров вызывается реинициализация трекеров. Для улучшения работы трекера CSRT удаляются шумы с изображения с помощью метода cv::fastNlMeansDenoisingColored. Этот метод удаляет гауссов шум, сравнивая похожие фрагменты изображения и усредняя.

Для уменьшения повторов в коде был создан класс TrackerPattern, от которого наследуются TrackerKCF, TrackerCSRT, TrackerTLD, который в свою очередь наследуется от класса Tracker и переопределяет его виртуальные методы.

На рис. 2 показана диаграмма классов реализации трекера.

3.8. Интеграция

Была произведена интеграция с проектом CVAT[9]. Были рассмотрены два пути встраивания стороннего трекера: вместо трекера TrackerMIL или с использованием Nuclio и serverless функций.

Процесс разработки был замедлен некорректной инструкцией по сборке версии CVAT для отладки, в которой несколько раз не были указаны необходимые версии библиотек. Так же была обнаружена ошибка

[9]Репозиторий: https://github.com/opencv/cvat (дата обращения: 08.05.2023)

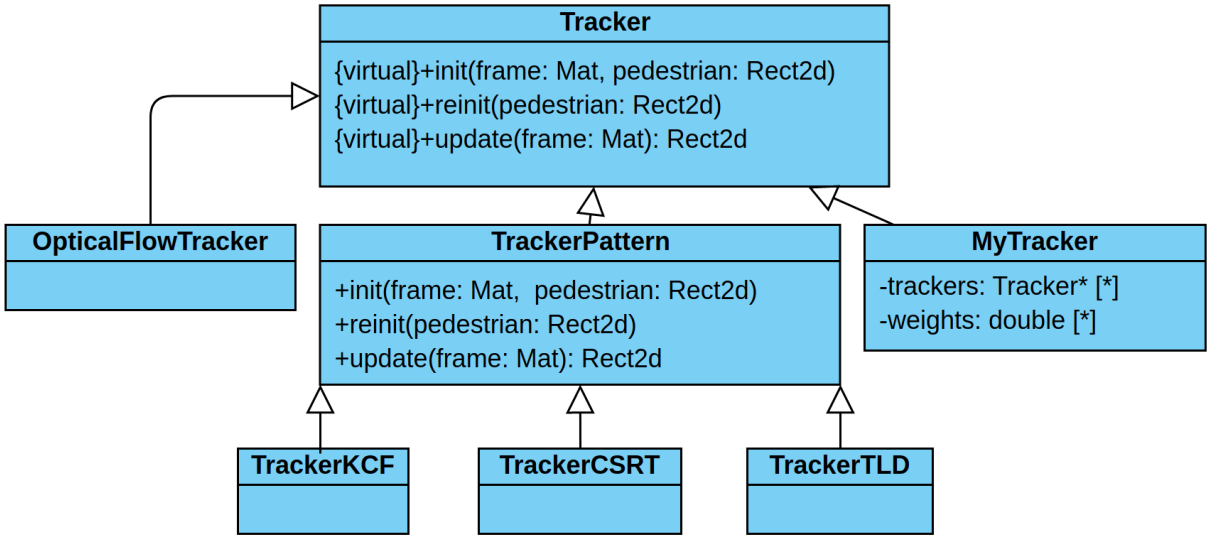

Рис. 2: UML-диаграмма классов текущей версии трекера

в работе CVAT, был отправлен вопрос[10] разработчикам, на который они ответили, а затем исправили ошибку в проекте.

Были исследованы следующие подходы к интеграции:

1. Заменить TrackerMIL на реализованный трекер. TrackerMIL — стандартный трекер из OpenCV. Код CVAT обращался к библиотеке OpenCV, собранной для использования в javascript при помощи Emscripten[11]. Интерфейс CVAT написан на языке typescript. Была изучена документа и написаны аддоны[12] для вызова кода, написанного на языке C++ из кода на javascript/typescript. Позднее выяснилось, что модуль Node.js не может быть подключен к коду, который исполняется в браузере. Было принято решение не исполнять ресурсоемкий код в браузере и не собирать библиотеку с трекером при помощи Emscripten, как было сделано с OpenCV для использования в исходном коде CVAT.

2. Отправлять http-запрос к трекеру из кода, где вызывается TrackerMIL. Была реализована серверная часть для трекера при помощи библиотеки cpprestsdk. Но возникли сложности с отправ-

[10]Issue: https://github.com/opencv/cvat/issues/5996(дата обращения: 10.05.2023)
[11]Emscripten: https://emscripten.org/(дата обращения: 08.05.2023)
[12]Аддоны, документация: https://nodejs.org/api/addons.html (дата обращения: 08.05.2023)

лением запроса из-за архитектуры CVAT, при которой запросы не доходили до адресата.

3. После этого был выбран путь создавать serverless функцию, как это было сделано с трекером TransT, который уже был проинтегрирован с CVAT. Но возникли сложности с отправлением запроса из контейнера — он также не доходил до адресата. Для обращения к serverless функции используется запрос со стороны CVAT, но из nuclio-контейнера обратиться сложнее, чем обратиться к нему.

   Serverless функция используется, поскольку она легко подключается, ее может вопринимать CVAT как трекер и отправлять запросы к ней.

   Было реализовано промежуточное решение - общение трекера и nuclio-контейнера посредством создания файлов в общей смонтированной папке для контейнера и python скрипта, который мониторит ее содержимое, а затем отправляет запрос трекеру, получает ответ, и записывает его в папку.

   Со стороны CVAT каждый шаг трекинга создавались следующие файлы:

   - текущее изображение, в названии которого содержится индекс шага;
   - текстовый файл, в названии которого содержится индекс шага и тип запроса (инициализация трекера или обновление).

   Скрипт ждет появления изображения с нужным индексом, а затем ждет текстовый файл, обрабатывает его, посылает http-запрос серверу с трекером. Получив ответ, создает текстовый файл с индексом шага в наименовании с ответом трекера внутри. CVAT дожидается создания этого файла, и затем реагирует на него передачей информации интерфейсу.

   Созданная архитектура позволяет интегрировать трекер не только с CVAT, если из инструмента можно создать http-запрос, а так-

же позволяет не менять код трекера в случае обнаружения способа отправить запрос из CVAT. У этого подхода есть существенный недостаток в виде медленной работы и возможных ошибок при чтении/записи в файлы.

Для упрощения работы пользователей с созданным трекером была добавлена возможность запуска сервера с трекером в докер-контейнере. Это позволяет избавить пользователя от необходимости устанавливать новые библиотеки, а так же позволяет ему не менять операционную систему. Был написан bash-скрипт, создающий образ контейнера с сервером и запускающий его, запускающий скрипт, являющийся посредником между CVAT и сервером, а также запускающий nuclio-контейнер с serverless фукцией.

После этого была реализована интеграция посредством http-запросов при помощи python-библиотеки requests. Чтобы из nuclio-контейнера отправить запрос, необходимо знать ip-адрес устройства внутри локальной сети. Доступ к нему не получить из контейнера напрямую, поэтому он читается из файла из монтированной папки, куда предварительно записывается с помощью python-скрипта, который вызывается из bash-скрипта.

4. Апробация

4.1. Метрики

Инструмент можно оценивать по шкале удобства использования по методике SUS[13]. Эта система позволяет дать численную оценку инструменту на основе ответов пользователя по пятибалльной шкале на список вопросов. Вопросы позволяют понять, удобно ли пользоваться инструментом, легко ли им научиться пользоваться, выглядит ли он переусложненным и так далее.

4.2. Методика эксперимента

В качестве испытуемых были отобраны люди, способные непредвзято оценить инструмент, и знакомые с аналогами. Испытуемыми были консультант и научный руководитель. Им было предложено воспользоваться инструментом на нескольких видео и оценить его по шкале удобства использования.

4.3. Результаты

4.3.1. Апробация

Оценка результатов созданного инструмента для оптимизации разметки была проведена на основе оценок пользователей. Была получена оценка консультанта: “Интеграция в CVAT — очень удачное решение, так как интерфейс для трекинга простой, интуитивно понятный, в то же время есть все желаемые функции.” Им были поставлены следующие оценки по шкале удобства использования:
1) 5,
2) 1,
3) 5,

[13]System usability scale:
https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html (дата обращения: 24.11.2022)

4) 3,
5) 4,
6) 1,
7) 4,
8) 1,
9) 5,
10) 2.

Сумма всех значений после необходимых преобразований равна 33. Сумму следует умножить на 2.5, и получится 87,5. Это значение больше 68, а следовательно является результатом выше среднего согласно исследованиям [4].

Была так же получена оценка от научного руководителя:
1) 3,
2) 2,
3) 5,
4) 1,
5) 5,
6) 2,
7) 4,
8) 1,
9) 3,
10) 5. Сумма всех значений после необходимых преобразований равна 29. Сумму следует умножить на 2.5, и получится 72,5.

4.3.2. Тестирование

Тестирование трекера проводилось по двум метрикам: точности и среднеквадратической ошибке отклонения. Точность трекера варьировалась от 85% до 96% на наборе видео [1], а среднеквадратическая ошибка отклонения по 4 координатам прямоугольника варьировалась от 15 до 45.

Заключение

Были выполнены следующие задачи.

- Проанализировать подходы к трекингу пешеходов.
- Выбрать несколько алгоритмов трекинга, которые будут использованы для создания трекера.
- Реализовать трекер.
- Интегрироваться с CVAT.
- Протеститровать созданный трекер.
- Провести апробацию созданного инструмента.

Реализация трекера находится в GitHub-репозитории по ссылке: https://github.com/osechkina-masha/adas_spbu/tree/pedestrian_tracking_4_sem, интеграция CVAT с ним произведена GitHub-репозитории по ссылке: https://github.com/zhitm/cvat_with_mytracker, разработка ведется под аккаунтом zhitm.

Список литературы

[1] URL: https://drive.google.com/drive/u/0/folders/1gfvI4_cbJpLdxonhMrFKk6NcRdkQu_Ij (online; accessed: 2023-05-26).

[2] Alan Lukežič Tomáš Vojíř Luka Čehovin Jiří Matas Matej Kristan. Discriminative Correlation Filter with Channel and Spatial Reliability. — 2016. — URL: https://arxiv.org/abs/1611.08461 (online; accessed: 2022-11-18).

[3] Alex Fandrianto Aman Neelappa Rohan Maheshwari. Single Camera Pedestrian Tracking. — URL: https://cvgl.stanford.edu/teaching/cs231a_winter1415/prev/projects/CS231aProjectFinalReport.pdf (online; accessed: 2022-11-18).

[4] Brooke John. SUS: A Quick and Dirty Usability Scale. — 2015. — URL: https://www.researchgate.net/publication/228593520_SUS_A_quick_and_dirty_usability_scale (online; accessed: 2023-05-14).

[5] Chao Zou GuoPing Yang. An improved motion pedestrian tracking algorithm based on CamShift. — 2019. — URL: https://dl.acm.org/doi/10.1145/3366194.3366265 (online; accessed: 2022-11-18).

[6] Hui Li Yun Liu Chuanxu Wang Shujun Zhang Xuehong Cui. Tracking Algorithm of Multiple Pedestrians Based on Particle Filters in Video Sequences. — 2016. — URL: https://pubmed.ncbi.nlm.nih.gov/27847514/ (online; accessed: 2022-11-18).

[7] Srishti Yadav Shahram Payandeh. Understanding Tracking Methodology of Kernelized Correlation Filter. — 2018. — URL: https://ieeexplore.ieee.org/document/8614990 (online; accessed: 2022-11-18).

[8] Tomasi Carlo. Image Correlation, Convolution and Filtering. — 2015. — URL: https://courses.cs.duke.edu/fall15/cps274/notes/convolution-filtering.pdf (online; accessed: 2022-11-18).

[9] Xinxin Zhen Shumin Fei Yinmin Wang Wei Du. A Visual Object Tracking Algorithm Based on Improved TLD. — 2020. — URL: https://www.researchgate.net/publication/338366942_A_Visual_Object_Tracking_Algorithm_Based_on_Improved_TLD (online; accessed: 2022-11-18).

[10] Yibing Zhang Tao Fan. A pedestrian tracking algorithm based on background unrelated head detection. — 2018. — URL: https://ieeexplore.ieee.org/document/8275014 (online; accessed: 2022-11-18).

[11] Смирнов Карим. Видео Карима Смирнова. — URL: https://www.youtube.com/watch?v=VJKElFaXjt8&ab_channel=%D0%9A%D0%B0%D1%80%D0%B8%D0%BC%D0%A1%D0%BC%D0%B8%D1%80%D0%BD%D0%BE%D0%B2 (online; accessed: 2022-12-06).