Лицей «Физико-техническая школа» им. Ж. И. Алфёрова
Санкт-Петербургского Академического университета

Курсовая работа (отчет о практике)

Поиск прерывистой разметки в кадре

Работу выполнили:
ученики 11в класса Солостовский Василий,
Макогон Артём.

Научные руководители:
Кринкин Кирилл Владимирович
Чайка Константин Владимирович

Место прохождения практики:
JetBrains Research

Санкт-Петербург
2020

Аннотация

В ходе работы был написан алгоритм, который с хорошей точностью находит прерывистую дорожную разметку в городе проекта Duckietown.

Оглавление

Аннотация.....2
Введение.....4
Постановка задачи.....5
Данные.....5
Методика.....5
OpenCV.....5
Готовые функции, которые мы использовали.....7
Фильтрация данных.....8
Менее эффективные подходы:.....9
Результаты.....11
Выводы.....12
Благодарности.....12
Список литературы.....12

Введение

Автономное движение автомобилей - серьезный вызов для человечества. Люди проводят большое количество времени за рулем, которое могло бы освободиться, если бы их автомобили могли бы самостоятельно их доставлять до места назначения. Однако, научить машину безопасно для себя и окружающих передвигаться по дороге - очень сложная задача. Один из проектов с целью разработки технологий управления автомобилем является Duckietown. Мы решили принять в нем участие. Duckietown — это небольшой город, в котором есть сеть дорог, которая размечена наподобие классических автомобильных, перекрестки, дорожные знаки, и участники дорожного движения - машинки (duckiebot). Суть проекта в том, чтобы при небольшом бюджете привлечь в отрасль автономного управления автомобилями как можно больше студентов для ее стремительного развития.

Постановка задачи

Дан набор данных (несколько тысяч изображений и десятки видео из игрушечного города). Требовалось разработать метод, с помощью которого можно эффективно по времени (со скоростью более 30 кадров в секунду) и как можно точнее определять местоположение элементов прерывистой разметки.

Данные

Даны видео (примеры – кадры из них)

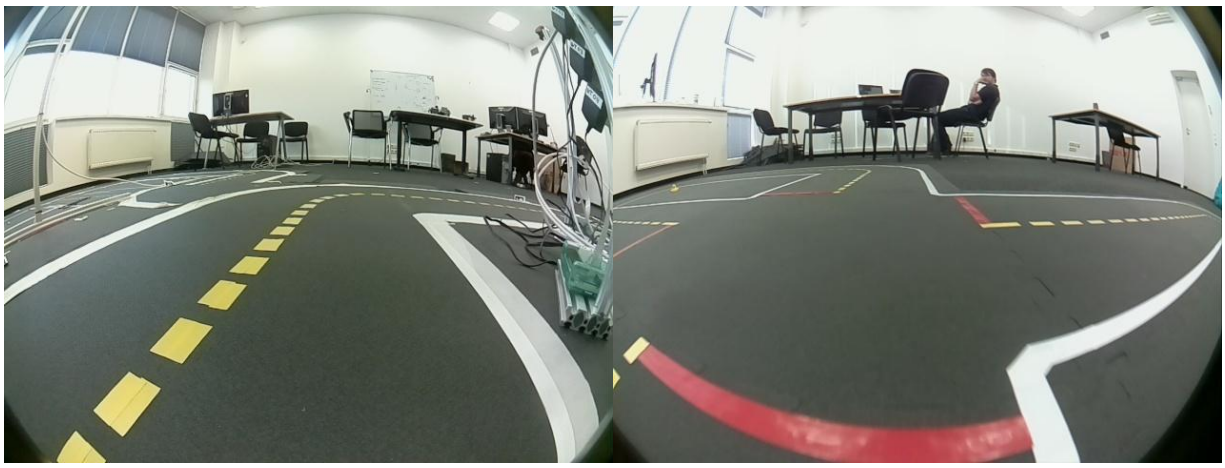


Рис. 1, 2: примеры входных данных

Методика

OpenCV

Библиотека OpenCV – библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для других языков (в том числе Python, который был использован)

Библиотека содержит более 2500 оптимизированных алгоритмов, которые включают в себя полный набор как классических, так и современных алгоритмов компьютерного зрения и машинного обучения. Некоторые из них мы использовали в своей работе.

Обработка данного изображения

1) Выпрямление:

   Камера, установленная на роботе, снимает через широкоугольную линзу. При такой съемке возникает эффект “рыбьего глаза”, из-за чего прямоугольная разметка искажена. Чтобы избавиться от этой проблемы, изображение нужно выпрямить. Для этого нужно сделать снимок шахматной доски на камеру, методами OpenCV вычисляется матрица - коэффициенты искажения для данной камеры. Нам эта матрица преобразования уже была дана вместе с базой видео (для каждой камеры она своя), и мы ее использовали для выпрямления каждого кадра функцией undistort() библиотеки OpenCV.

2) Бинаризация:

   Можно легко заметить, что дорожная разметка светлая на фоне темной дороги, поэтому мы приводим картинку к бинарному виду (каждый пиксель либо черный, либо белый). Сначала мы делаем изображение серым при помощи функции cvtColor() OpenCV, то есть каждый пиксель принимает значение от 0 до 255. Затем к изображению применяется пороговая функция threshold(), в результате которой если значение пикселя больше порогового, он становится белым, иначе черным. Пороговое значение мы подобрали исходя из того, чтобы разметка была белой, дорога черной, и чтобы на дороге было как можно меньше бликов. Стоит отметить, что это пороговое значение не является универсальным и лучше всего подходит именно под те условия, на которых мы тестировали нашу программу.

Рис 3, 4, 5: постепенная обработка изображения

Теперь, когда кадр приведен к бинарному виду, найти разметку на нем уже гораздо легче.[1]

[1] Идею бинаризации мы увидели в статье Введение в OpenCV применительно к распознаванию линий дорожной разметки

Готовые функции, которые мы использовали

Для начала заметим, что элементы разметки довольно сильно выделяются на фоне черной дороги, поэтому мы использовали функцию findCountours(), которая по находит контуры в изображении – кривые, вдоль которых происходит сильное изменение цвета (в нашем случае с черного на белое).

Т.к. исследуемые элементы в проекции очень похожи на прямоугольники, то мы использовали функцию approxPolyDP(), которая основана на Алгоритме Рамера - Дугласа - Пекера. Это алгоритм, позволяющий уменьшить число точек кривой, аппроксимированной большей серией точек для того, чтобы с этой кривой было удобнее работать.[2]

Мы использовали уже готовую реализацию этого алгоритма вместо того, чтобы реализовывать самостоятельно, потому что это серьезно экономит время в написании программы при том же результате. В нашем случае этот алгоритм выявляет, каким многоугольником можно приблизить кривую, которая получается из анализа контуров предыдущей функцией.

[2] подробнее про суть работы этого алгоритма можно прочитать в статье https://en.wikipedia.org/wiki/Ramer–Douglas–Peucker_algorithm

Фильтрация данных

Используя вышеописанные методы, мы на вход получали большое количество многоугольников, которые находятся на кадре (возможно, среди них есть ошибочно определенные многоугольники). Теперь задача состоит в том, чтобы из этого множества отобрать те, которые являются разметкой. Для этого мы решили использовать систему фильтров, которые наше множество многоугольников постепенно проходит. И мы считаем, что если элемент прошел их все, то он является элементом разметки. Расскажем о фильтрах поподробнее.

Перевод многоугольников в эллипсы

Для удобства еще немного упростим наши элементы разметки. С помощью функции нахождения многоугольников мы нашли все четырехугольники, треугольники и отрезки в кадре. Рассмотрим их все и для каждого определим самую верхнюю, нижнюю, левую и правую точки. По этим четырем точкам построим прямоугольник, стороны которого параллельны сторонам кадра. Наш элемент полностью содержится в построенном прямоугольнике. Проведя несколько экспериментов, мы поняли, что эффективнее всего представлять элементы как эллипсы, вписанные в эти многоугольники. Теперь заметим, что нам довольно просто что-то понимать про взаимное расположение этих элементов на кадре, поскольку мы знаем все координаты точно.

Фильтр размеров

Так как наш кадр – кадр робота с дороги, мы можем считать, что у него есть горизонт (линия, выше которой никогда не появятся элементы разметки). Просмотрев достаточно много кадров из базы данных, мы пришли к выводу, что разумная оценка сверху для данной линии – 0,7 высоты всего кадра. Дальше заметим, что чем дальше элемент разметки от камеры, тем он меньше. Конечно, это зависит от расположения камеры на роботе, но в среднем оно не сильно отличается от того, с которым мы имели дело. Будем считать, что чем выше находится многоугольник, тем он дальше (понятно, что в общем случае это неверно, но это неплохое приближение). Введем параметр, зависящий от высоты центра эллипса на кадре — максимальный размер полуоси, который он может иметь. Пусть этот параметр убывает линейно. На самом деле зависимость экспоненциальная в силу перспективы кадра, но, чтобы корректно обрабатывать повороты, где предположение “выше — значит дальше” неверно, мы приняли линейную зависимость. Все элементы, у которых размер полуоси больше, чем параметр - точно не подходят как кандидаты в элементы разметки, их можно отбросить. Так же заведем минимальный размер полуоси эллипса, при котором мы можем считать этот

элемент разметкой (для отбрасывания случайных мелких прямоугольников из-за шумов на фотографии).

Все элементы, прошедшие этот фильтр, отправляются дальше.

Фильтр кучности

Заметим, что элемент разметки почти всегда имеет рядом с собой элемент спереди и элемент сзади. И так мы придумали фильтр кучности. Давайте возьмем некоторый элемент разметки, возьмем некоторую область вокруг него и посчитаем количество других к данному моменту найденных элементов. Если таковых не нашлось, то скорее всего текущий элемент нам не подходит. Если рядом с ним слишком много найденных, то он нам тоже не подойдет. Также он нам не подойдет, если внутри него найден центр другого элемента (так как каждый элемент разметки – сплошной многоугольник). Экспериментальным путем мы выяснили, что максимально эффективно запускать этот фильтр два раза.

Вероятностный фильтр

Изображения поступают в нашу программу последовательно, и от кадра к кадру происходят небольшие изменения — это значит, что позиции элементов разметки меняются не сильно, это можно эффективно использовать. Будем считать, что элемент может “переехать” между соседними кадрами не больше, чем на какое-то выбранное расстояние (зададим его заведомо больше, чем может быть для корректной работы). Теперь найдем на нашем кадре разметку алгоритмом, который мы описали выше. Сравним полученные эллипсы с теми, которые были на предыдущем кадре. И если рядом с новым эллипсом есть какой-то из старых, то точно берем его. А если он уникален (и рядом не нашлось элементов разметки с прошлого кадра), то возьмем его с какой-то вероятностью p, которую подберем экспериментально (эффективнее всего из проверенных вариантов получалась работа при p = 1/20). Получается, что алгоритму необходимо некоторое «время на адаптацию» после резких поворотов и в самом начале пути.

Менее эффективные подходы:

Фильтр черных многоугольников

Мы посмотрели на разметку и поняли, что так как элементы разметки белые, а фон – черный, то рядом с каждым элементом разметки находится какое-то количество черных пикселей. Для каждого элемента разметки мы взяли прямоугольную картинку около него, предположили, что через эту картинку проходит белая линия и так посчитали примерное отношение количества черных пикселей к общему количеству пикселей в этом прямоугольнике.

Впоследствии оказалось, что этот фильтр является затратным по времени и не очень эффективным, поэтому данный фильтр мы убрали.

Трекинг разметки методами OpenCV

У нас была идея отслеживать разметку на видео методами OpenCV, чтобы делать предсказания, но эти методы работают очень долго относительно нашей программы, так как они созданы для более общих случаев и на кадре много элементов разметки, за которыми надо следить.

Результаты

Достоинства:

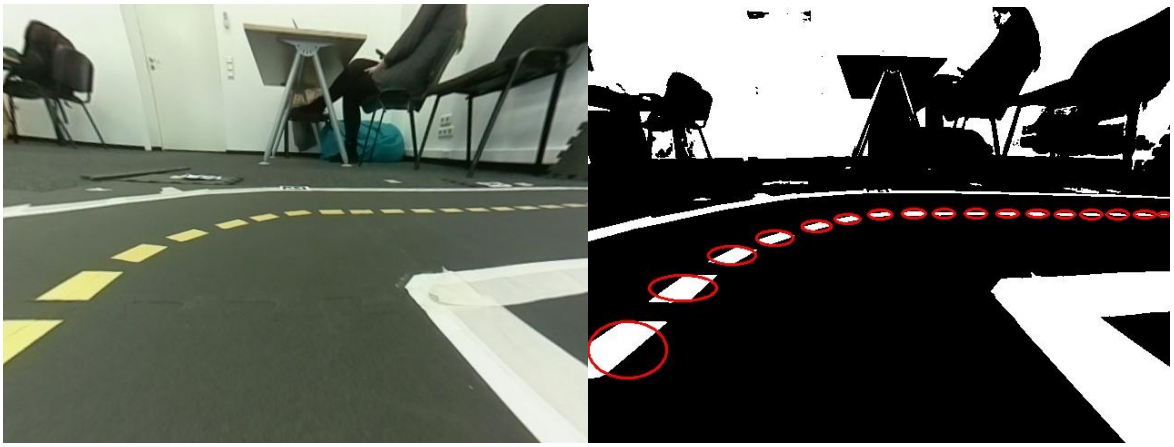
1) Скорость. Нам удалось добиться работы нашей программы с частотой примерно 35-40 кадров в секунду. В частности, этого получилось добиться, потому что мы использовали быстрые методы, реализованные в библиотеке OpenCV, которые эффективно работают.
2) Хорошая точность. На наш взгляд, мы добились очень неплохой точности в определении элементов разметки, тем самым решив поставленную задачу на данном наборе данных.

Недостатки:

1) У нас в коде большое количество констант. Эти константы подобраны так, чтобы программа хорошо работала на кадрах, на которых мы эту программу пробовали. А значит, если эту программу запускать на другой камере или другом роботе, то программа будет работать не так эффективно и константы нужно подбирать вручную.

Задачи для дальнейшей работы:

1) Внедрить нашу программу в операционную систему роботов (ROS)



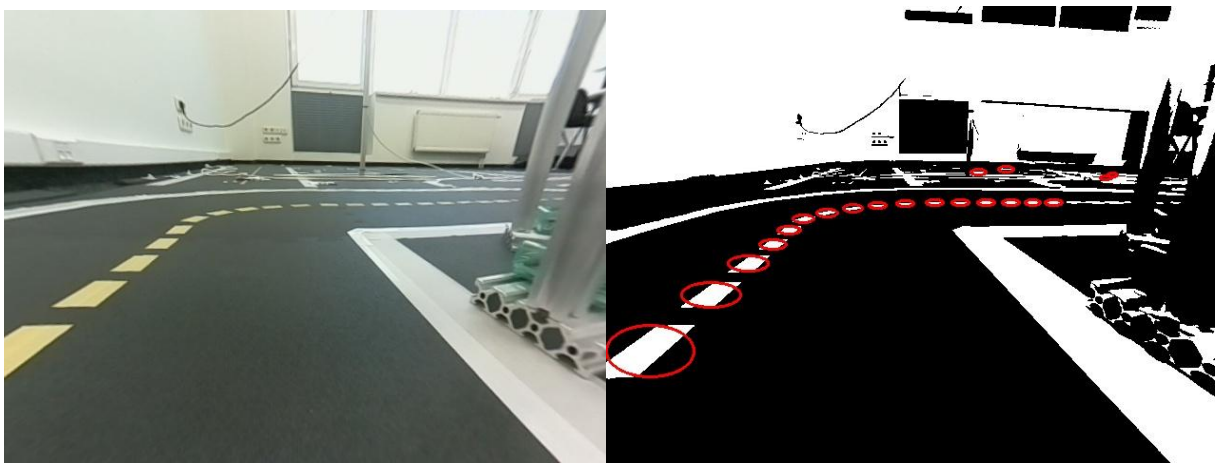


Рис 6 – 9: примеры данных и результатов (то, что обведено программа помечает как разметку)

Выводы

Нашу программу можно использовать в прямом назначении - на роботах в Duckietown, но также у нее есть еще несколько не менее интересных применений.

Сейчас все более популярным становится использование алгоритмов машинного обучения в подобных задачах. С помощью нашей программы можно создать очень большую базу обработанных кадров для обучения моделей и комбинируя наши методы с методами машинного обучения создать программу, которая будет решать эту задачу еще более эффективно.

Благодарности

Хотим поблагодарить Кринкина Кирилла Владимировича и Чайку Константина Владимировича за большой вклад в работу.

Список литературы

1) Ramer–Douglas–Peucker algorithm: https://en.wikipedia.org/wiki/Ramer–Douglas–Peucker_algorithm
2) Документация OpenCV: https://opencv.org/
3) “Введение в OpenCV применительно к распознаванию линий дорожной разметки”: https://habr.com/ru/company/newprolab/blog/328422/
4) Все коды, используемые в программе: https://gitlab.com/osll/Duckietown-Software/-/blob/Dash-line-cv-detection/Dash-line-cv-detection/current-version.py
5) Проект Duckietown: https://www.duckietown.org

Отзыв

Солостовский Василий и Макогон Артём проходили практику в период с 24.09.2019 по 26.01.2020 в лаборатории алгоритмов мобильных роботов JetBrains Research. За время прохождения практики Василием и Артёмом был разработан подход к детекции разметки по ее геометрической форме на изображениях камеры мобильного робота исследовательского проекта Duckietown. Оба обучающихся показали высокий уровень самостоятельной работы. Поддерживали на протяжении всей практики оперативную коммуникацию, четко описывали возникающие проблемы. В течение практики обучающиеся самостоятельно овладели навыками работы с новыми для них инструментами, в том числе, такими как библиотека компьютерного зрения и система контроля версий Git. Разработки Василия и Артёма в дальнейшем планируется интегрировать в основной проект.

Солостовский Василий и Макогон Артём оба заслуживают оценки “отлично”.

Код разработанного решения доступен в репозитории проекта лаборатории: https://gitlab.com/osll/Duckietown-Software/-/blob/Dash-line-cv-detection/Dash-line-cv-detection/current-version.py

Руководитель практики, программист-исследователь:
Чайка Константин Владимирович
От организации:
АНО ДПО “Научно-исследовательский и образовательный центр “ДжетБрейнс”

25.04.2020 _____/Чайка К.В./