Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчет по НИР

Разработка алгоритмов для беспилотных транспортных средств

Выплолнил студент
гр. 3530904/60105 Стойкоски Н.С.

Руководитель И.А. Селин

Санкт-Петербург
2019 г.

Оглавление

Введение

Беспилотный автомобиль – транспортное средство, оборудованное системой автоматического управления, которое может передвигаться без участия человека. Программное обеспечение беспилотного автомобиля может включать машинное зрение и нейросети. Некоторые системы полагаются на инфраструктурные системы (например, встроенные в дорогу или около неё), но более продвинутые технологии позволяют имитировать присутствие человека на уровне принятия решений о изменении положения руля и скорости, благодаря набору камер, сенсоров, радаров и систем спутниковой навигации.

Компьютерное зрение — теория и технология создания машин, которые могут производить обнаружение, отслеживание и классификацию объектов. Как научная дисциплина, компьютерное зрение относится к теории и технологии создания искусственных систем, которые получают информацию из изображений. Видеоданные могут быть представлены множеством форм, таких как видеопоследовательность, изображения с различных камер или трехмерными данными, например с устройства Kinect или медицинского сканера. Как технологическая дисциплина, компьютерное зрение стремится применить теории и модели компьютерного зрения к созданию систем компьютерного зрения.

Семантическая сегментация изображений означает разделение изображения на отдельные группы пикселей, области, соответствующие одному объекту с одновременным определением типа объекта в каждой области. В автономное вождение семантическая сегментация используется для обнаружения дорожных разметок, других транспортных средств, людей, а также любых представляющих интерес объектов. Результат сегментации затем используется для принятия решений по правильному управлению транспортного средства. Недостатком такого применения технологии в автономном вождении является необходимость производить сегментацию в реальном времени. Решением может стать локальная интеграция GPU в транспортное средство. Кроме этого, чтобы повысить эффективность работы системы можно использовать более легкие нейронные сети (с меньшим количеством параметров).

Постановка задачи

Рассматривается задача разработки системы зрения для беспилотного автомобиля, позволяющая воспринять окружающего пространства из видеопотока. Данная система на входе будет принимать кадры из видеопотока, при этом предполагается что камера находится в передней части автомобиля и направлена вперед, глядя на окружающую среду перед автомобилем.

Эта система должна извлечь полезную информацию из видеопотока для восприятия окружающего пространства, на основе которой строится траектория движения (по середине дороги) которая покадрово отдается в качестве выхода системы для дальнейшего принятия решения об управляющих действий автомобиля.

Вопросы, подлежащие проработке:

1. Методы распознавания дороги/трассы
2. Перспективное преобразование и построение траектория движения
3. Программная реализация алгоритмов

Исходные данные

Имеется набор видео файлов с высоком расширением, снятые на автогоночную трассу. Также известные (открытые) датасеты Kitti, CamVid, CityScapes для семантическая сегментация изображения дорожного транспорта.

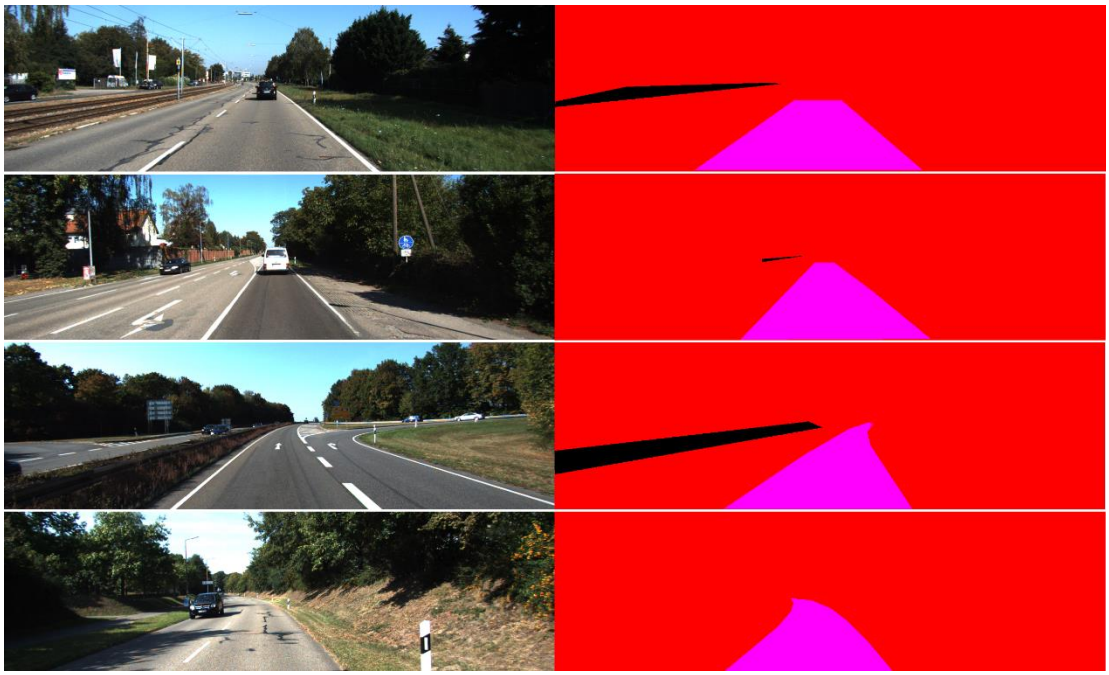

Рис.1 Образцы из датасет KITTI

Рис.2 Кадры из исходных видео файлов

Методы распознавания дороги

1. Алгоритм на основе обнаружения краев

Для обнаружения дороги был разработан алгоритм в начале которого выполняетсяобнаружение краев в исходном кадре. Для эту задучу была выбрана полностью сверточная нейронная сеть – HED, и был использован предварительно обученный модель на датасет BSDS500.

Обнаружение краев (Edge detection) включает в себя множество математических методов, которые направлены на определение точек на цифровом изображении, в которых яркость изображения резко изменяется или, более формально, имеет разрывы. В идеальном случае результатом выделения границ является набор связанных кривых, обозначающих границы объектов, граней и оттисков на поверхности, а также кривые, которые отображают изменения положения поверхностей.

Holistically-Nested Edge Detection (HED) – алгоритм выделение границ в изображения основанный на полностью сверточных нейронных сетях для классификации типа изображение-изображение - система принимает изображение на вход и непосредственно производит на выход карта краев (изображение, которое указывает, где находятся края входного изображения); Обучение основанное на глубоко контролируемые сети (deeply-supervised nets). которые выполняют глубокий контроль, чтобы «направлять» ранние результаты классификации.

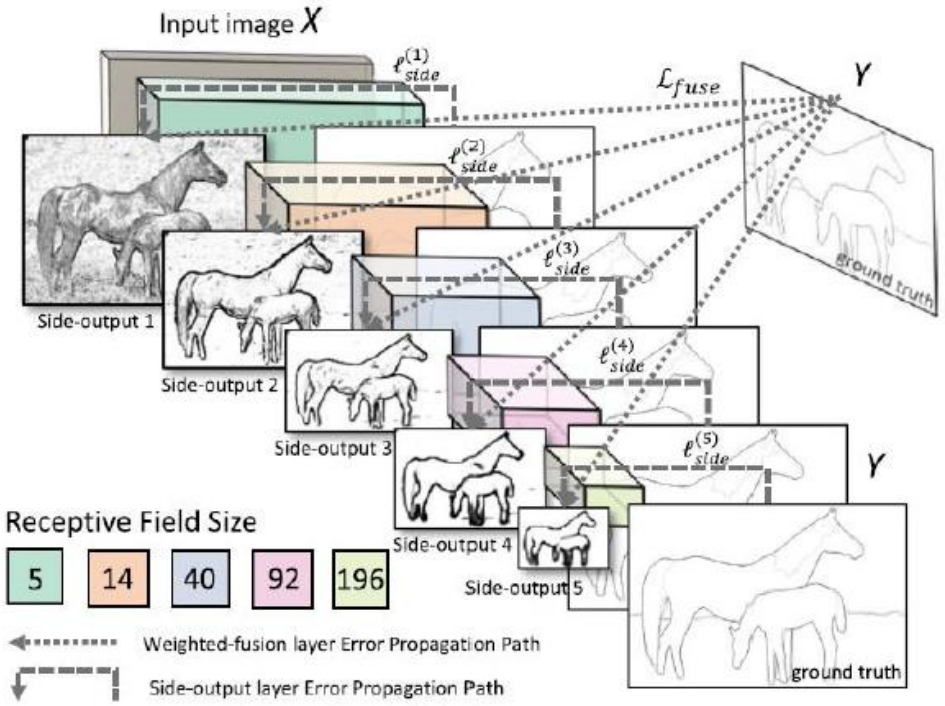

Рис.3 Иллюстрация сетевой архитектуры для обнаружения краев

На первом этапе алгоритма при использованиея нейросети HED выделяются края в исходном изображений. Это реализовано в среде python с использованием библиотеки pytorch.

Предполагается, что автомобиль уже находится на дороге по которой он движется и пиксель (начальный), находящиеся в середине (по x) нижней 1/5 изображения (по y) является частью дороги.

К полученной карты краев применяется порог и соответственно получается бинарное изображение над котором применяем алгоритм floodfill (библиотеки OpenCv) из начального пикселя. Соответственно получается бинарная маска дороги (для каждый пиксель знаем о том является ли он часть дороги или нет).

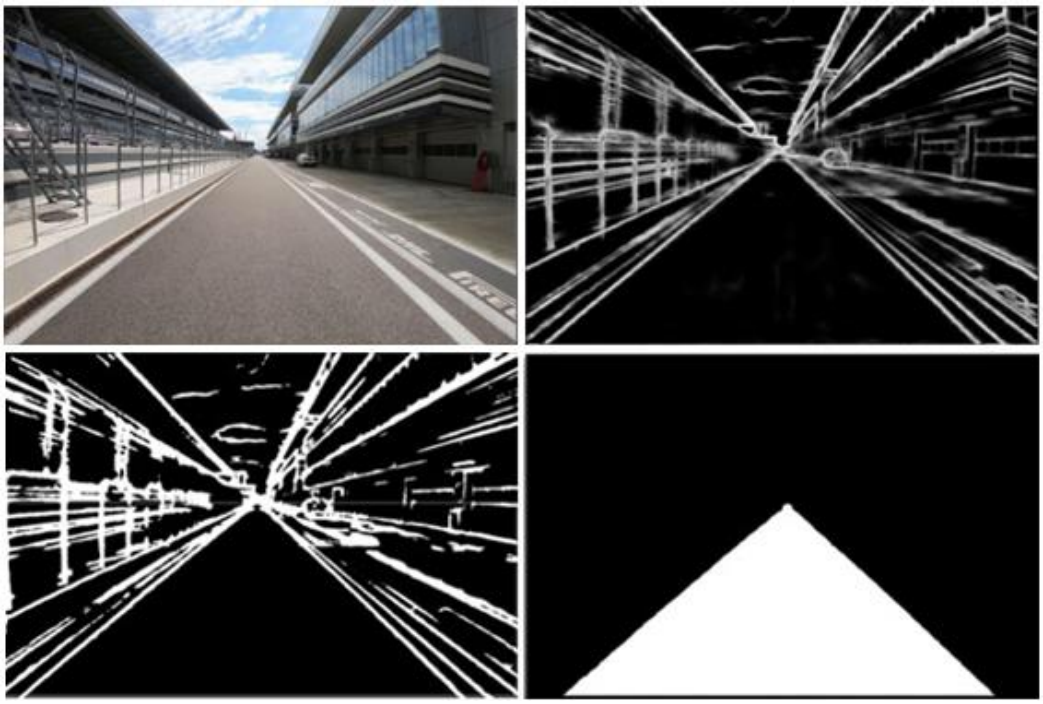

Рис.4 Этапы работы алгоритма распознавания дороги

Данный алгоритм имеет недостаток в том, что полученная карта краев не всегда польностью закрывает дорогу в рамку и следовательно floodfill "вытекает" из ее границы, в результате чего получается неверная маска дороги. Для борьбы с этой проблемой были использованы ограничения по x, y при выполнения алгоритма floodfill.

Когда изображение дороги четкое, этот метод может успешно обрабатывать около 95% кадров, могут быть некоторые проблемы при крутых поворотах, туннелях и т.д. Это одна из причин, по которой мы предпочитаем обработку изображений и непосредственное получение маски дороги в нейронных сетях для семантической сегментации.

2. Алгоритм на основе семантической сегментации

Семантическая сегментация изображений - это разделение изображения на отдельные группы пикселей, области, соответствующие одному объекту с одновременным определением типа объекта в каждой области. Задача семантической сегментации является высокоуровневой задачей обработки изображений, относящейся к группе задач т.н. слабого искусственного интеллекта. Она является даже более сложной, чем задача классификации изображений и поиска объектов, что обусловлено не только необходимостью определения классов объектов, но и выявления их структуры, правильного выделения частей объектов на изображении.

- Сбор и разметка данных для обучения

Был создан датасет состоящий из N=246 пары изображения-меток с расширением 960x640. Количество классов = 2 (дорога / не дорога). При этом были отобраны 27 образцов из kitti_lane, 89 образцы из kitti_road, 130 образцов полученные из исходных видео файлов после обработке.

Образцы из датасет KITTI были первоначально обработаны для достижения соответствия с выбранном расширением изображения. Для этого использовались стандартные операции для изменения размера и обрезка изображения.

Метки изображений в датасет KITTI содержат больше классов чем интересующих нашие (дорога / не дорога) и соответственно была использована реализованная функция для выделения искомых меток.

Так же был создан набор из 130 образцов, извлеченные из исходных видео файлов sochi.mp4, sochi2.mp4 которые были приведены в расширение 960x640 и отмечены в полуавтоматическом режим с использованием реализованной функции на основе opencv.

Средние значения пикселей всех изображений в датасете, для дальнейшая нормализация: $[B, G, R]_{mean} = [109.851, 110.867, 108.510]$

Полный датасет можно найти по следующей ссылке:
https://drive.google.com/open?id=1JzXpOrm3I0BFlZlqJwiB7TSCe6Duvw6T

- Выбор модели

За основе решения данной задачи был выбран модель полностью свёрточной нейронной сети FCN (Fully Convolutional Network for Semantic Segmentation) основанный на предобученном VGG16.

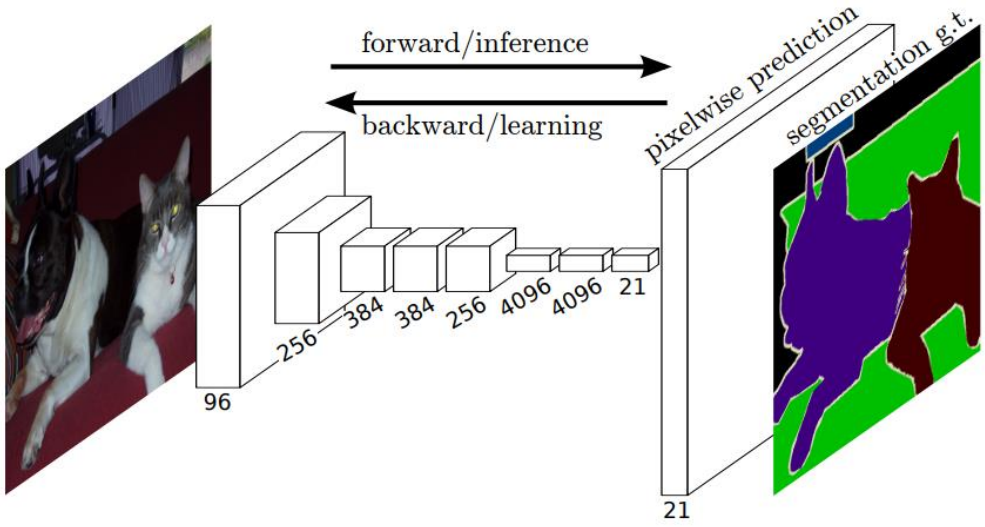

Рис. 5 Полностью сверточные сети могут эффективно научиться делать плотные прогнозы для задач на пиксель, таких как семантическая сегментация.

FCN - одна из самых простых и популярных архитектур, используемых для семантической сегментации. В статье “FCN для семантической сегментации” авторы используют FCN для первоначального преобразования входного изображения до меньшего размера (одновременно с этим увеличивая количество каналов) через серию сверток. Такой набор сверточных операций обычно называется кодировщик. Затем выход декодируется или через билинейную интерполяцию, или через серию транспонированных сверток, который носит название декодер.

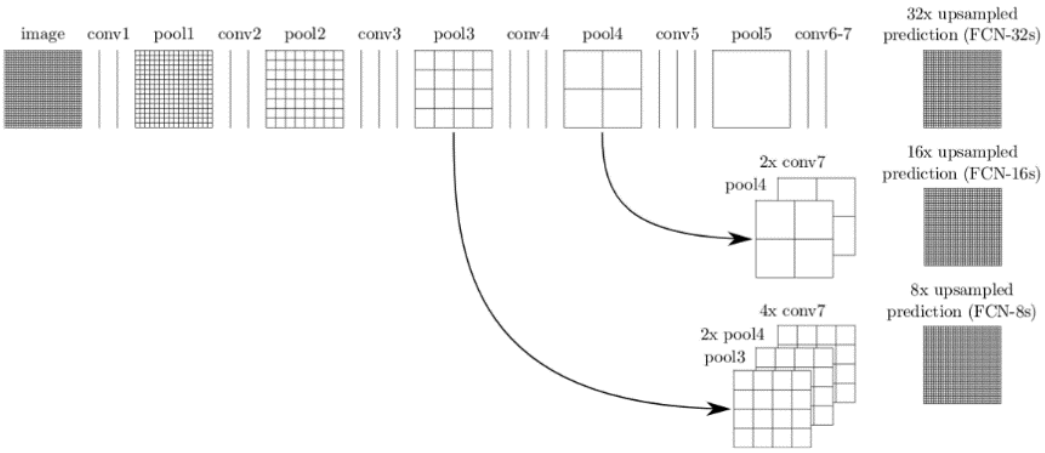

Рис.6 Архитектура полностью сверточной нейронной сети FCN

- Обучение модели

Наличие GPU ускорителя является критическим фактором для скорости обучения deep learning моделей. Без GPU обучение нейросети займет многие часы/дни и не позволит полноценно экспериментировать со структурой сети.

Для обучения данной модели исользовался сервис Google Colaboratory, предоставляющий доступ к Jupyter ноутбукам, с возможностью бесплатно использовать GPU Tesla K80 с 13Гб видеопамяти на борту.

Для обучения была использована функция потери - `torch.nn.BCEWithLogitsLoss`. Эта функция потери объединяет сигмоидальный слой и BCELoss в одном классе. Эта версия является более численно устойчивой, чем использование простого сигмоида, за которым следует BCELoss, поскольку, объединяя операции в один слой, мы используем трюк log-sum-exp для численной устойчивости.

$$\ell(x,y)=L=\{l_1,\dots,l_N\},\ l_n=-w_n[y_n\cdot\log\sigma(x_n)+(1-y_n)\cdot\log(1-\sigma(x_n))]$$

При обучении сообщаются метрики из общей семантической сегментации и анализа разбора сцены, которые являются вариациями по точности пикселей и пересечению области над объединением (IU):

Точность пикселей: $\sum_i n_{ii} \ / \ \sum_i t_i$

Средняя точность: $(1/n_{cl}) \sum_i n_{ii} \ / \ t_i$

Среднее IU: $(1/n_{cl}) \sum_i n_{ii} \ / \ (t_i + \sum_j n_{ji} - n_{ii})$

Стандартным показателем, который используется для оценки производительности алгоритмов семантической сегментации, является Mean IoU (пересечение по объединению), где IoU определяется как:

$$\text{IOU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{A_{pred} \ \cap \ A_{true}}{A_{pred} \ \cup \ A_{true}}$$

Такая метрика гарантирует, что мы фиксируем каждый объект (делая предсказанные метки накладывающимися на основную правду), но также делаем это точно (делая объединение как можно ближе к перекрытию).

Для обучения модели были использованы следующие параметры:

```
n_class    = 2           momentum   = 0
batch_size = 6           w_decay    = 1e-5
epochs     = 25          step_size  = 50
lr         = 1e-4        gamma      = 0.5
```

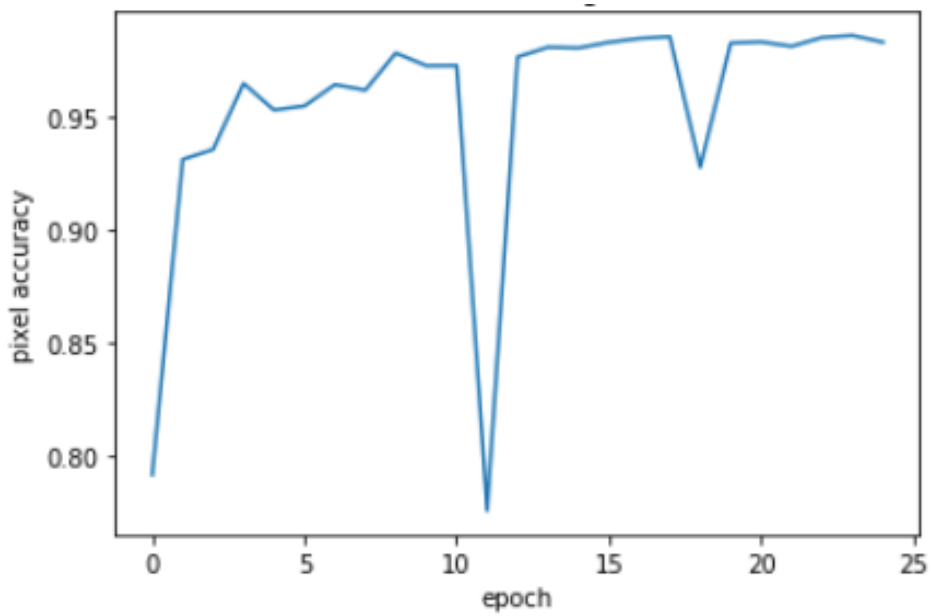

Рис.7 График точности обученя

Рис.8 Примеры результатов сегментации

Обратное перспективное преобразование

Из одного изображения, обращенного вперед, трудно определить расстояния до объектов впереди транспортного средства с любой степенью уверенности. Существует нелинейная связь между высота объекта на переднем изображении и его расстояние от камеры. Для решений это проблемы можно использовать метод - Inverse Perspective Mapping (IPM).

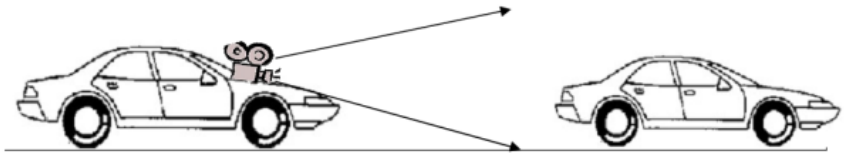

Рис. 9 Передняя камера обнаруживает объекты впереди автомобиля

Используя IPM, мы можем преобразовать изображение, обращенное вперед, в нисходящий вид «с высоты птичьего полета», в что есть линейная зависимость между расстояниями в изображении и в реальном мире.

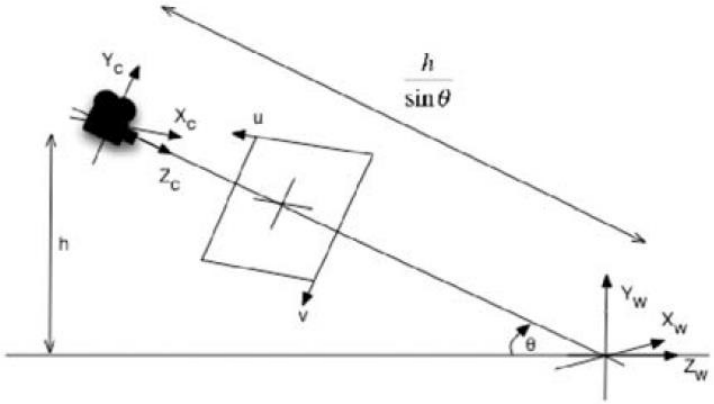

Рис.10 Система координат изображения относительно системе координат реального мира

Для того, чтобы создать вид сверху вниз, сначала нужно найти отображение точки на плоскости дороги (X_w, Y_w, Z_w) к ее проекции на плоскости изображения (u, v). Из рис.10 видно, что это требует выполнить вращение под углом θ, трансляция вдоль оптической оси камеры и масштабирование матрицы параметров камеры.

Это отображение может быть выраженное как:

$$(u, v, 1)^T = \boldsymbol{KTR}(x, y, z, 1)^T$$

Где **R** – матрица вращения:

$$\boldsymbol{R} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

T – матрица трансляции:

$$\boldsymbol{T} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -\dfrac{h}{\sin\theta} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

К – матрица параметров камеры:

$$\boldsymbol{K} = \begin{pmatrix} f \times ku & s & u_o & 0 \\ 0 & f \times kv & v_o & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Сдесь, f – фокусное расстояние камеры, s – параметр перекоса пикселей, и $(ku \times kv)$ – соотношение сторон пикселей.

При этом следует выбрать параметры (высота камеры, focal length, угол поворота) в соответствие с имеющиеся инструменты (спецификации камеры, ее местоположение в отношением с автомобиля).

Была создана функция которая вычисляет матрица преобразования по параметры камеры и вследствие выполняет преобразование входного изображения с использованием функции библиотеки OpenCV.

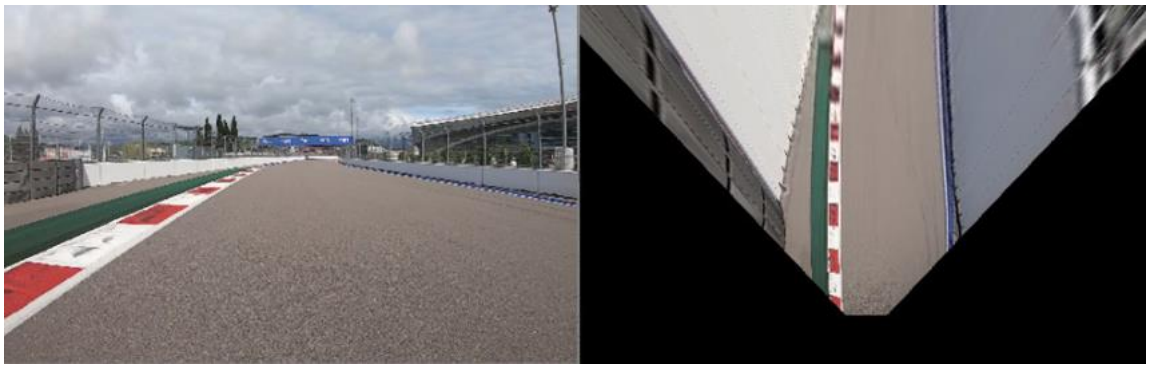

Рис.11 Результат работы алгоритма обратного перспективного преобразования

Построение траектория движения

В результате этапы сегментация дороги получаем бинарная маска для дорогу т.е. изображение в котором белый пиксель – дорога, черный – нет. Нужно построить по данному кадру траектория движения по середине дороги.

Это изображение преобразуем в вид птичьего полета с использованием обратного перспективного преобразования.

Была создана функция которая для заданная строка в изображением дает номер столбца который является середина дороги в данной строке изображения. Берём координаты (row, col) всех белых пикселев в бинарном изображений и используется метод найменьшых квадратов с полиномом третьей степени для аппроксимацию крывой. Это реализовано с помощю функии библиотеки numpy (numpy.polyfit).

Был использован весовой метод для аппроксимации, при этом к каждой точке (пиксель являющийся часть дороги) был добавлен вес зависящий от расстояния этого пискеля до траектория из предыдущего кадра. Это было сделано для повышения стабильности и гладкости при имеющиеся шумов и неправильности в предсказания.

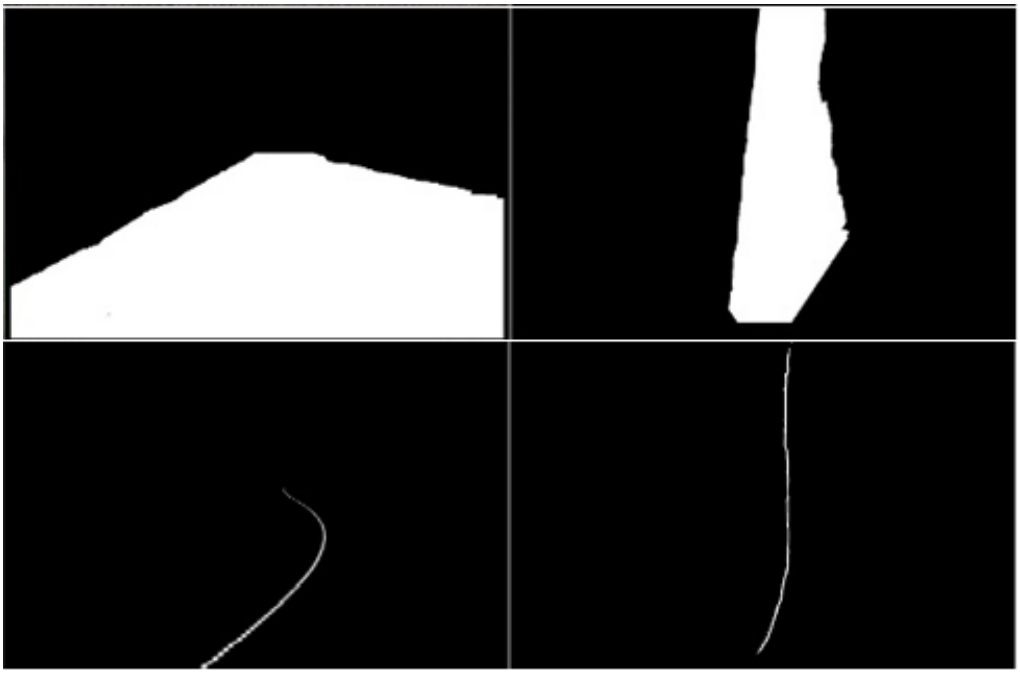

Рис.12 Работа алгоритма построения траекторию движения

Полученные результаты

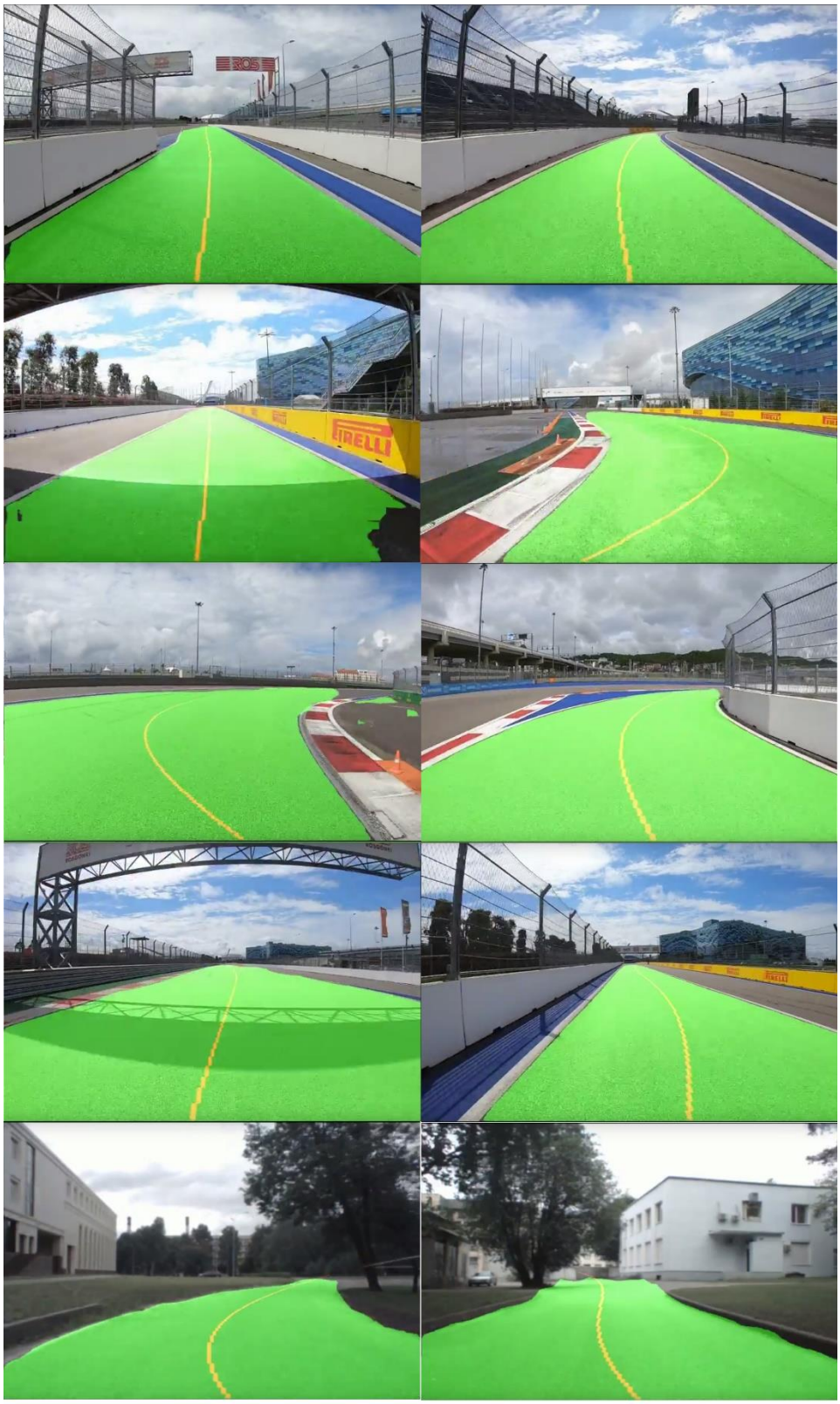

Рис.13 Результаты работы алгоритма на различных кадров исходных видео файлов

Заключение

В рамках этой работы были получены навыки работы с разных задач компьютерного зрения при разработки и применения алгоритмов для беспилотных транспортных средств. Я так же ознакомился с библиотеку pytorch при реализации в неё модели нейронных сетей для решения задач обнаружения краев и семантическая сегментация.

Был разработан алгоритм обратного перспективного преобразования – получение вид птичьего полета. Были разработаны алгоритмы обнаружения дороги в видеопотоке которые базируются на нейронные сети для обнаружение и получения карта краев (HED) и для семантическая сегментация (FCN). Так Был разработан алгоритм для построения траектория движения где используется весовой метод найменьших квадратов при аппрокисимирование результатов от этапе распознавания дороги.

Дальнейшие этапы включают ознакомление с алгоритмы на основе Байесовского метода одновременной локализации и построения карт (SLAM, simultaneous localization and mapping). А так же ознакомление с ROS (Robot Operating System) — Операционная система для роботов — это фреймворк для программирования роботов, предоставляющий функциональность для распределённой работы

Список литературы

1. Tuohy, Shane & O'Cualain, D & Jones, Edward & Glavin, Martin. (2010). Distance determination for an automobile environment using Inverse Perspective Mapping in OpenCV. 2010. 100 - 105. 10.1049/cp.2010.0495.
2. Xie, Saining & Tu, Z. (2017). Holistically-Nested Edge Detection. International Journal of Computer Vision. 125. 1-16. 10.1007/s11263-017-1004-z.
3. Long, Jonathan, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
4. https://github.com/sniklaus/pytorch-hed
5. https://github.com/pochih/FCN-pytorch
6. https://docs.opencv.org/3.0-beta/modules/refman.html
7. https://docs.scipy.org/doc/scipy/reference/