Санкт-Петербургский государственный университет

БАХМЕТЬЕВ Владимир Андреевич

Выпускная квалификационная работа

Определение полос движения на заснеженной дороге по видео

Направление: 02.03.02

«Фундаментальная информатика и информационные технологии»

Основная образовательная программа СВ.5003.2017

«Программирование и информационные технологии»

Профиль «Автоматизация научных исследований»

Научный руководитель:
доц. каф. СП, к.т.н. Литвинов Ю.В.

Рецензент:
Пименов А.А., ген. директор
ООО «Системы компьютерного зрения»

Технический консультант:
М.С. Осечкина, инженер-программист
ООО «Тиквижн»

Санкт-Петербург
2021

Оглавление

Список условных обозначений

ADAS (англ. Advanced driver-assistance systems) – Усовершенствованная система помощи водителю.

Оператор Собеля – дискретный дифференциальный оператор, вычисляющий приближённое значение градиента яркости изображения.

Оператор Кэнни (детектор границ Кэнни, алгоритм Кэнни) – алгоритм обнаружения границ на изображении.

Метод Оцу – алгоритм вычисления порога бинаризации для полутонового изображения.

Преобразование Хафа – вычислительный алгоритм, применяемый для параметрической идентификации геометрических элементов растрового изображения

LLVM (Low Level Virtual Machine) – проект программной инфраструктуры для создания компиляторов и сопутствующих им утилит.

HSL (от англ. hue, saturation, lightness (intensity)) – цветовая модель, в которой цветовыми координатами являются тон, насыщенность и яркость.

IoT (Интернет вещей, англ. internet of things) – концепция сети передачи данных между физическими объектами («вещами»), оснащёнными встроенными средствами и технологиями для взаимодействия друг с другом или с внешней средой.

DBSCAN (от англ. Density-based spatial clustering of applications with noise) – алгоритм кластеризации, основанный на плотности.

Введение

В настоящее время информационные технологии активно внедряются в транспортные средства. Они помогают водителю управлять транспортным средством в течении всего пути, начиная от старта и заканчивая процессом парковки. Наиболее известными примерами являются антиблокировочная система и система курсовой устойчивости. Все они в разной степени помогают водителю, начиная от проблем, возникающих с механической частью автомобиля и заканчивая автономным движением без участия водителя в процессе.

Основная идея всех этих технологий заключается в автоматизации контроля систем автомобиля на разных уровнях. Одной из систем, помогающих вождению автомобиля, является система ADAS.

Системы ADAS используют датчики, подключенные к IoT, и на основе получаемых данных решают в режиме реального времени, стоит ли предупреждать водителя об опасной ситуации и брать ли функции вождения в критических ситуациях на себя. Благодаря подобным системам значительно снижается риск ДТП.

Одной из актуальных задач систем ADAS является помощь водителю во время управления автомобилем на заснеженной дороге. Основной сложностью решения данной задачи является определение границ дороги, на основе которых система ADAS должна принимать решения. Существующие алгоритмы и системы способны распознавать границы дороги только при помощи дополнительных датчиков или при идеальных условиях. Под идеальными условиями подразумевается солнечный дневной свет и отсутствие осадков в виде снега на дороге. Алгоритмов, распознающих границы заснеженной дороги по видео в режиме реального времени и дающих качественный результат, сейчас не существует.

Постановка задачи

В рамках данной работы основной целью является проектирование и реализация алгоритма, способного распознавать границы дороги в режиме реального времени. Для достижения данной цели были поставлены следующие задачи:

1. Исследовать видео и определить основные признаки, по которым можно выделить область дороги с границами.
2. Провести анализ алгоритмов с целью выделить те, которые смогут определить признаки дороги с максимальной точностью за минимальное время.
3. Основываясь на полученных результатах анализа спроектировать алгоритм распознавания границ.
4. Реализовать алгоритм.
5. Провести профилирование алгоритма с целью получить характеристики точности и производительности алгоритма.
6. Оптимизировать алгоритм до уровня работы в режиме реального времени.

1. Исследование записи видеорегистратора

Для начала необходимо определить, с помощью каких признаков можно идентифицировать участок дороги. Также нужно учесть все условия, которые могут позитивно и негативно сказаться на процессе распознавания.

Пример рассматриваемого кадра из видео изображен на рис.1.

Рис.1 – Кадр, снятый на видеорегистратор.

1.1. Определение условий распознавания

В результате анализа были выявлены следующие условия, от которых может зависеть качество распознавания.

- Время суток, когда производилась запись (день/ночь).
- Количество источников освещения, которые зависят от того, на каком участке дороги была произведена запись (город/загород).
- Количество выпавших осадков (обильные/средние/отсутствие).

Наличие осадков в виде снега напрямую связано с текущим сезоном (в основном с зимой), что в свою очередь связано с длиной светового дня. Зимой световой день значительно короче, из-за чего видимость дороги

обуславливается в основном искусственными источниками освещения, такими как фары автомобиля или уличные фонари. Основным временем, когда трафик повышен, являются периоды с 8.00 до 12.00 и с 17.00 до 22.00, когда большинство людей находятся в пути либо на работу, либо с работы домой. В эти периоды интенсивность солнца довольна слаба, из чего можно сделать вывод, что в большинстве случаев работа будет происходить с записями, сделанными при искусственном освещении.

От количества источников освещения может зависеть зашумленность изображения. То есть при достаточной освещенности количество шумов будет довольно мало, что может быть только в рамках городских дорог, где количество фонарей довольно велико. При недостаточной освещенности количество шумов будет велико, что может негативно сказаться при распознавании. Также при большой чувствительности видеорегистратора к освещению может произойти следующая проблема: при нахождении источника искусственного освещения довольно близко к центру, видеорегистратор может перефокусироваться на сам источник освещения, из-за чего сама дорога в этот момент может стать неразличимой.

Так как запись производится на видеорегистратор, чаще всего расположенный за лобовым стеклом, то от количества осадков может зависеть количество снега и воды, которые будут находиться на лобовом стекле и перекрывать рассматриваемый кадр.

В результате анализа были выделены следующие негативные факторы, которые нужно учесть при проектировании алгоритма:

- Шум, возникающий при слабой освещенности.
- Перепады баланса белого из-за динамических источников света.
- Осадки в виде сухого и мокрого снега.

1.2. Определение признаков распознавания

В результате анализа записей при различных вариантах освещенности, времени суток и количества осадков были сделаны следующие выводы:

1) Распознавание границ дороги сильно зависит от количества выпавших осадков. Если опираться на распознавание границ дороги согласно ПДД, то в таком случае при большом количестве осадков, когда дорожная разметка не видна, а сами осадки (в данном случае снег) снижают уровень сцепления с дорогой, то возникает необходимость в новом определении.

2) В случае, когда количество осадков велико, под границами можно подразумевать некоторый участок заснеженной дороги, на котором вероятность снижения управляемости автомобиля максимально низка.

3) В случае, когда осадков мало, под границами дороги можно понимать определение согласно ПДД.

4) Участок дороги, на котором вероятность потерять управление наиболее низка, можно охарактеризовать определенным набором признаков.

При исследовании записей были выделены следующие ключевые факторы, при помощи которых можно распознать признаки дороги и составить новое определение границ дороги согласно задаче:

1) След автомобилей. Следы в большинстве случаев сонаправлены с дорогой и имеют различную степень заметности. В городах дороги обрабатывают специальными веществами, из-за чего колея имеет характерный цвет, отличный от цвета чистого снега. Также в отличие цвета следов вносит свой вклад земля и глина. В регионах, где данная обработка слаба или отсутствует и трафик значительно ниже, следы представляют из себя малозаметное углубление в дороге, которое мы можем отследить благодаря следующему признаку: цвет чистого снега на кадре напрямую зависит от количества света, попавшего на него. В углубления попадает значительно

меньше света, из-за чего оно выделяется на фоне ровного слоя снега более тусклым оттенком.

2) Перепады яркости пикселей дороги значительно отличаются от перепадов яркости окружающих дорогу объектов. За перепады отвечает освещенность и грязь, которая распространяется другими автомобилями. На окружающих объектах грязи нет, или её намного меньше, что можно использовать в данной задаче.
3) Камера всегда находится в неподвижном положении относительно ориентации автомобиля, что можно использовать для упрощения обработки и калибровки.

1.3. Заключение

В результате анализа были выявлены основные признаки дороги и характеристики, по которым их можно выявить. Тогда за определение признака дороги можно принять следующую формулировку: признак дороги – некоторый объект на кадре, вытянутый преимущественно вдоль оси Oy, состоящий из набора соседних пикселей, размер которого должен быть больше определенного значения, который можно выделить по отличию яркости относительно соседних пикселей в меньшую сторону.

Запись представляет из себя последовательность изображений в формате RGB. Согласно выделенному критерию отличия яркости имеет смысл перейти в другое цветовое пространство, одной из координат которого была бы яркость. Наиболее подходящим цветовым пространством является пространство HSL [12].

HSL представляет из себя цветовое пространство, основные координаты которого являются тон, насыщенность и яркость. Переход в данное цветовое пространство из пространства RGB можно провести по формулам 1.1, 1.2, 1.3.

$$H = \begin{cases} undefinded \ \ if\ MAX = MIN \\ 60^0 * \frac{G-B}{MAX+MIN} + 0^0,\ if\ MAX = R\ and\ G \geq B \\ 60^0 * \frac{G-B}{MAX+MIN} + 360^0,\ if\ MAX = R\ and\ G < B \\ 60^0 * \frac{G-B}{MAX+MIN} + 120^0,\ if\ MAX = G \\ 60^0 * \frac{G-B}{MAX+MIN} + 240^0,\ if\ MAX = B \end{cases}, \quad (1.1)$$

$$S = \begin{cases} 0,\ if = 0\ or\ MAX = MIN \\ \frac{MAX-MIN}{MAX+MIN},\ if\ 0 < L \leq \frac{1}{2} \\ \frac{MAX-MIN}{2-(MAX+MIN)},\ if\ \frac{1}{2} < L < 1 \end{cases}, \quad (1.2)$$

$$L = \frac{1}{2}(MAX + MIN), \quad (1.3)$$

где

- R, G, B — значения цвета в цветовой модели RGB, значения в диапазоне $[0;\ 1$ красный, G — зелёный, B — синий).
- MAX — максимум из трёх значений (R, G, B).
- MIN — минимум из трёх значений (R, G, B).
- H — тон $[0;\ 360]$.
- S — насыщенность $[0;\ 1]$.
- L — яркость $[0;\ 1]$.

Также было выявлено, что рассматриваемый кадр можно разделить на области, каждая из который может как представлять, так и не представлять определенный интерес в рамках данной задачи. Пример разделения изображен на рис.2.

Список выделенных областей:

1. Область капота (красная область).

2. Область дороги (зеленая область).
3. Область, включающая линию горизонта и все, что находится выше неё (синяя область).
4. Части дороги, включающие соседние полосы и ограждения (белая область).

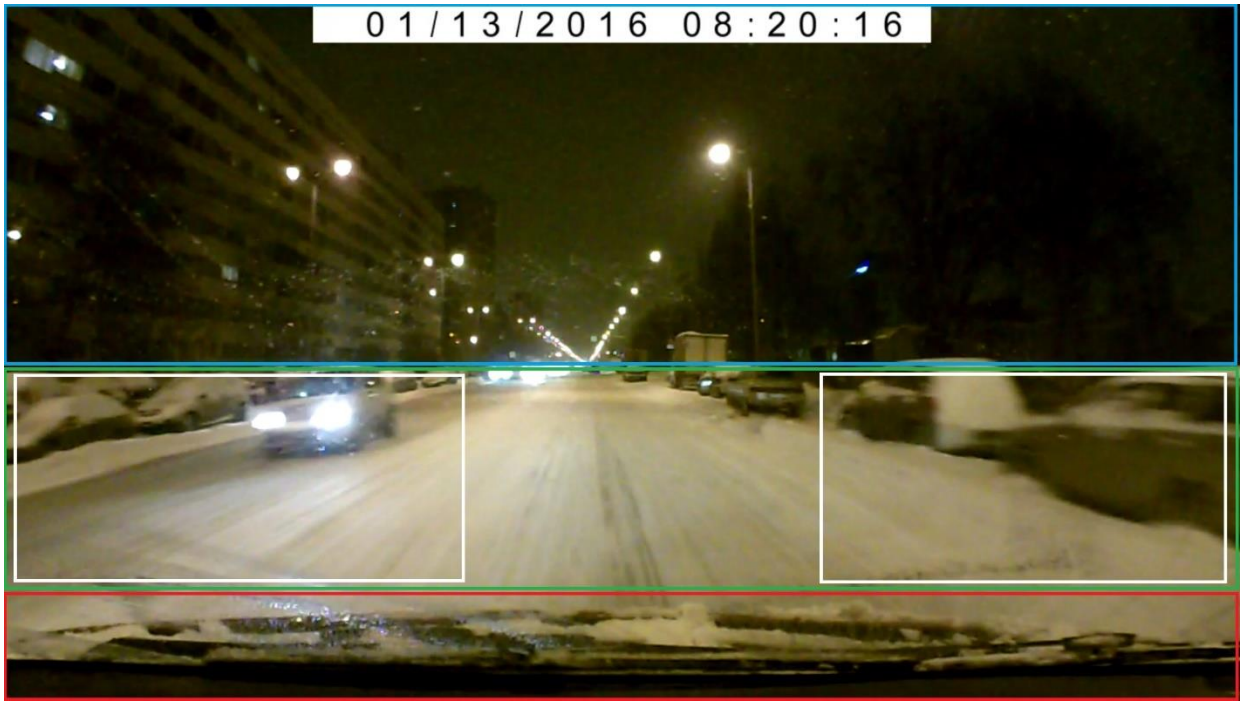

Рис.2 – разбиение кадра на области

Первая и третья область не представляют интереса в поставленной задаче, так как никакой информации о дороге они не несут.

Наибольший интерес будет представлять область дороги (рис.3). На ней и будет происходить поиск дороги вместе с границами. Центральная часть дороги будет сигнализировать о том, насколько автомобиль будет стабильно держаться относительно основной траектории его движения. Границы дороги (4) будут рассматриваться в случае перестроения автомобиля или приближения его к краю дороги.

Рис.3 – Область дороги.

2. Декомпозиция задачи и обзор алгоритмов

2.1. Определение основных этапов решения задачи

Чтобы корректно спроектировать алгоритм, решающий поставленную задачу, нужно изучить решения задачи схожей предметной области, а именно распознавание границ дороги при идеальных условиях по разметке [18]. Также изучить методы обработки негативных внешних условий [10]. После этого следует декомпозировать задачу на последовательные этапы. Далее для каждого из этапов нужно определить задачу, которую он будет решать, рассмотреть различные алгоритмы для данного этапа и выбрать наиболее подходящий алгоритм согласно критериям качества и времени выполнения.

Поскольку каждый из признаков дороги значительно отличается от слоя снега по яркости, то для поиска признаков дороги следует использовать метод расчета значений градиента в точках. Далее, следует отфильтровать пиксели согласно значению нормы градиента и угла градиента.

После этого согласно значениям градиента необходимо объединить полученные пиксели в признаки дороги, на основе которых будет происходить построение области дороги и её границ.

На финальном этапе нужно объединить признаки в область дороги и ограничить её, тем самым получив желаемый результат.

В итоге были выделены следующие этапы, решающие поставленную задачу:

1. Поиск пикселей, относящихся к признакам дороги.
2. Объединение пикселей в признаки.
3. Построение на основе признаков области дороги и ограничение её границами.

2.2. Методы поиска граничных пикселей

Одним из самых популярных алгоритмов выявления граничных пикселей является Оператор Кэнни [7]. Данный алгоритм состоит из пяти основных этапов:

1) Предварительное размытие – предварительный этап, который применяется в большинстве алгоритмов компьютерного зрения. Основная цель – устранение шумов, возникающих при плохих условиях освещенности. Для этого этапа используется Гауссово распределение с параметром σ, который обычно берется равным 1,4.

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{x^2+y^2}{2\sigma^2}} \quad (2.1)$$

Формула двумерного гауссова распределения задается формулой (2.1). Используя формулу, строим матрицу свертки размера 5 на 5, после чего применяем операцию свертки к нашему изображению. В результате получаем размытое изображение без шумов.

2) Поиск градиентов – этап, который ищет направление градиента изменения изображения в каждом пикселе относительно всех соседних пикселей. Для данного этапа используется операция свертки и два оператора Собеля [9]: оператор для поиска вертикального градиента (2.2) и оператор для поиска горизонтального градиента (2.3).

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A, \quad (2.2)$$

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \quad (2.3)$$

Далее, используя полученные матрицы, мы можем посчитать абсолютное значение градиента (2.4) и угол наклона градиента (2.5) в каждом пикселе.

$$G = \sqrt{{G_x}^2 + {G_y}^2} \quad (2.4)$$

$$\Theta = \arctan\left(\frac{G_y}{G_x}\right) \quad (2.5)$$

3) Подавление не максимумов – этап, в котором происходит поиск локальных максимумов. Идея данного этапа заключается в следующем: пиксель, значение градиента которого является максимальным среди своих соседей, с наибольшей вероятностью является граничным пикселем.
4) Двойная пороговая фильтрация – на данном этапе происходит разбиение полученных пикселей на 3 основных класса на основании двух порогов. Если значение пикселя ниже, чем нижний порог, то тогда этот пиксель точно не является границей изображения. Если значение пикселя выше, чем верхний порог, то этот точно является границей изображения. Если пиксель попадает в интервал между верхней и нижней границей, то он относится к третьему классу – классу неоднозначности.
5) Устранение неоднозначности – на данном этапе происходит однозначное отношение пикселей из класса неоднозначности к одному из двух других классов по следующему правилу: если пиксель находится рядом с пикселем, который отнесен к граничному классу, то он тоже относится к этому классу, иначе ко второму классу.

Также алгоритмом, решающим схожую задачу, является алгоритм бинаризацией изображения.

Алгоритмы бинаризации изображения разбивают пиксели изображения на два класса: “искомые” и “фоновые”. В общем случае задаётся некоторое пороговое значение, при удовлетворении которого пиксель относится к классу “искомые”, иначе к классу “фоновые”.

При бинаризации изображения встаёт задача определения условия, по которому нужно разбить пиксели, и определение оптимального порогового значения. Одним из известных алгоритмов определения порогового значения является метод Оцу.

Метод Оцу [11] работает с полутоновыми изображениями. Под условием разбиения он подразумевает снижение внутриклассовой дисперсии искомого класса пикселей. Под внутриклассовой дисперсией подразумевается взвешенная сумма дисперсий двух классов (2.6).

$$\sigma_w^2 = w_1\sigma_1^2 + w_2\sigma_2^2, \tag{2.6}$$

где

- w_1 – вероятность пикселя принадлежать первому классу
- w_2 – вероятность пикселя принадлежать второму классу
- σ_1 – дисперсия первого класса
- σ_2 – дисперсия второго класса

Одним из свойств метода Оцу является то, что при снижении дисперсии в классе нужных пикселей, межклассовая дисперсия увеличивается. Упоминание метода Оцу является целесообразным в виду того, что по установленным особенностям класс “признак” будет отличаться от класса “фоновый снег/ окружение признака дороги”, а внутриклассовая дисперсия в этом случае будет схожа. Основная идея метода Оцу – перебор всех уровней яркости для нахождения максимальной межклассовой дисперсии.

1) Вычислить гистограмму монохромного изображения и частоту для каждого уровня интенсивности.
2) Вычислить начальные значения $\omega_1(0), \omega_2(0), \mu_1(0), \mu_2(0)$, где ω — это вероятность для каждого класса содержать пиксель интенсивности 0, а μ это среднее арифметическое соответствующего класса.
3) Для каждого $t = \overline{1..k}$, где k - то максимальный уровень интенсивности, нужно обновить ω_1, ω_2, μ_1, μ_2, рассчитать $\sigma_b^2 = \omega_1(t)\omega_2(t)[\mu_1(t)\mu_2(t)]^2$, и если новое значение больше чем последнее запомненное, то запомнить t.

В результате мы получим значение t, которое и будет пороговым значением при бинаризации изображения.

В данном контексте под полутоновым изображением можно рассматривать канал L из цветового пространства HSL.

Наиболее подходящим вариантом поиска граничных пикселей является алгоритм, основанный на операторе Собеля и установленных свойствах признаков дороги. Основная идея заключается в том, чтобы посчитать оператором норму и угол градиента при помощи формул 2.2 – 2.5 и затем, основываясь на установленных свойствах, разбить пиксели на классы согласно значениям угла градиента и выбрать наиболее подходящие классы.

Применять оператор будем на канале яркости (Lightness) из цветового пространства HSL.

У искомых признаков есть определенная общая черта: все эти объекты принадлежат прямым, которые сходятся в единственной точке, расположенной выше горизонта и либо сонаправленной с траекторией движения автомобиля либо с небольшим отклонением согласно свойствам перспективы.

После подсчета градиента в каждом из пикселей можно рассмотреть полученные характеристики: норму градиента и его угол. Так как все искомые

объекты принадлежат прямым и каждый из объектов вытянут вдоль оси Oy и находится в контрасте с окружающим его снегом, то большинство пикселей объекта будут обладать следующим свойством: если рассматривать прямую, которой принадлежит объект, угол градиента будет совпадать или находится в окрестности нормали, проведенной к этой прямой. В зависимости от направления градиента можно определить, какой переход наблюдается в данном пикселе. Если знак положительный, то это переход объект – снег, если отрицательный, то снег – объект. Будем разбивать все пиксели на четыре класса со следующими значением углов соответственно: $\left[-\frac{\pi}{4}, \frac{\pi}{4}\right]$, $\left[\frac{\pi}{4}, \frac{3\pi}{4}\right]$, $\left[\frac{3\pi}{4}, \frac{5\pi}{4}\right]$, $\left[\frac{5\pi}{4}, \frac{7\pi}{4}\right]$ и далее будем рассматривать углы из первого и третьего класса, так как эти интервалы совпадают с большинством прямых, сонаправленных с движением автомобиля.

2.3. Методы кластеризации

После получения граничных пикселей нужно объединить их в кластеры. Это можно использовать для удаления объектов, являющихся осадками на лобовом стекле, так как количество соседних пикселей в них довольно мало.

Одной из немаловажных операций в кластеризации является определение метрики [4].

Существуют следующие варианты метрик:

1. Евклидово расстояние: $\rho(x, \acute{x}) = \sqrt{\sum_i^n (x_i - \acute{x}_i)^2}$ (2.7)
2. Квадрат Евклидова расстояния: $\rho(x, \acute{x}) = \sum_i^n (x_i - \acute{x}_i)^2$ (2.8)
3. Манхэттенское расстояние $\rho(x, \acute{x}) = \sum_i^n |x_i - \acute{x}_i|$ (2.9)
4. Расстояние Чебышева: $\rho(x, \acute{x}) = max(|x_i - \acute{x}_i|)$ (2.10)
5. Степенное расстояние: $\rho(x, \acute{x}) = \sqrt[r]{\sum_i^n (x_i - \acute{x}_i)^p}$ (2.11)

Метрики 2.9 и 2.10 не подходят для данного этапа из-за определенной особенности искомых объектов: они вытянуты вдоль оси Oy и в данном случае идет потеря информации, связанной с этим свойством. Метрика 2.11 является универсальной и при подстановке параметров r и p можно получить метрику 2.7 (при $r = p = 2$) и метрику 2.8 (при $r = 1,\ p = 2$). Разница в метриках 2.7 и 2.8 заключается в том, что во второй метрике отдаленным объектам дается больший вес, чем в первой метрике. В поставленной задаче метрика 2.8 подходит больше, так как благодаря такому способу подсчета расстояния намного легче выделить кластеры меньшего размера, которые в дальнейшем можно будет игнорировать.

Алгоритмы кластеризации можно разделить по следующим двум критериям:

1. Структура кластера

2. Однозначность кластера

Суть первого критерия – наличие некоторой иерархии при разбиении множества на более мелкие и общее количество уровней вложенности кластеров. В случае плоских алгоритмов разбиение производится всего одно. В случае иерархического разбиения количество уровней составляет $n - 1$ при n элементах.

Суть второго критерия – принадлежность каждого из пикселей либо к одному, либо нескольким кластерам одновременно.

Поскольку основная цель сейчас это объединение пикселей в признаки дороги, то наиболее подходящим будет четкий и плоский алгоритм.

Наиболее известными такими алгоритмами являются методы K-means и алгоритм минимального покрывающего дерева.

После объединения пикселей в кластеры появляется возможность подсчитать количество элементов в каждом из кластеров и затем отфильтровать кластеры по количеству входящих в них элементов.

Метод k-means работает следующим образом: случайно в пространстве генерируется k случайных точек, которые являются изначальными центрами k кластеров. После этого элементы распределяются между этими центрами согласно полученной при помощи метрики близости. Далее рассчитываются новые центры кластеров, как центр тяжести с единичным весом, после чего процесс распределения и поиска нового центра начинается заново. Количество итераций задается пользователем самостоятельно, как и параметр k. Алгоритм также демонстрирует сильную зависимость от изначального распределения точек.

Алгоритм минимального покрывающего дерева основан на теории графов. Суть данного алгоритма в том, чтобы представить точки как вершины полного взвешенного графа, веса в котором будет определять метрика. Далее убираются ребра, длина которых больше порогового значения. Таким образом задав определенное пороговое значение можно сразу получить набор связных

подграфов, каждый из которых будет представлять из себя признак дороги. Основная проблема данного алгоритма заключается во времени расчета матрицы смежности и большого размера самой матрицы.

Третий вариант основан на алгоритме DBSCAN [3] и свойстве объектов поставленной задачи. Поскольку все признаки представляют из себя замкнутый объект, который обладает сильным контрастом по сравнению с фоном, то можно использовать следующий алгоритм:

1. Задаем матрицу размерности изображения, в которой каждому пикселю будет причисляться номер кластера, по умолчанию 0.
2. Задаем номер нового кластера равный 1.
3. Задаем пустой список, в котором по индексу кластера будет содержаться количество пикселей в нём.
4. По очереди рассматривается каждый пиксель изображения.
5. Если пиксель не относится к классам, то смотрим следующий.
6. Если пиксель относится к границам и не отмечен кластером, проверяем, есть ли у него соседи в радиусе 1, отмеченные кластером, если есть, то отмечаем этот пиксель этим же номером и увеличиваем количество пикселей в кластере на единицу, если нет, то инициализируем новый кластер: помечаем пиксель номером нового кластера, увеличиваем номер на единицу и устанавливаем количество установленного кластера равным единице и переходим к следующему пикселю.

Далее, после того как весь кадр обработан, можно отфильтровать пиксели, согласно кластерам, которым они принадлежат, и минимальному допустимому количеству элементов в каждом из кластеров.

2.4. Методы построения областей

После кластерной фильтрации необходимо объединить все кластеры в область дороги, которая будет ограничиваться прямыми.

Одним из алгоритмов для нахождения прямых, дающих максимальную точность, является преобразование Хафа [2]. При помощи него можно объединить объекты в прямые, которые их содержат. Далее можно объединить прямые в искомую область, где крайняя прямая будет являться границей дороги. Идея заключается в следующем: на плоскости можно описать прямую через параметры b и k при помощи формулы 2.11.

$$y = kx + b \tag{2.12}$$

Также можно описать прямую через другие два параметра r и θ, где r – длина вектора нормали, проведенная к прямой из центра координат, а θ это угол вектора нормали относительно оси ординат. Данное преобразование помогает легче работать с вертикальными прямыми имеющими параметр k, равный бесконечности. В таком случае можно описать любую прямую лишь одной точкой на плоскости $r\theta$.

Если есть некоторые точки, принадлежащие определенной прямой, существует возможность рассмотреть все прямые (с некоторым шагом), проходящие через каждую из этих точек. Далее идет голосование за комбинацию параметров r и θ представляющих определенную прямую. После перебора всех точек, искомые прямые, содержащие наибольшее количество точек, получат наибольшее кол-во голосов. Таким образом при помощи фильтрации по пороговому значению можно получить параметры r и θ, которые могут являться прямыми, принадлежащими области дороги.

Если точка имеет координаты x_0 и y_0, то уравнение всех прямых, проходящих через эту точку, можно описать при помощи формулы (2.13). Тогда при последовательной подстановке параметра θ из интервала $[0,\ \pi]$ с определенным шагом, можно получать различные прямые, проходящие через точку (x_0, y_0).

$$r(\theta) = x_0 \cos(\theta) + \ y_0 \sin(\theta) \tag{2.13}$$

Так как основной целью является выделение области дороги, то можно использовать другой алгоритм. Его суть заключается в следующем: искомая область дороги представляет из себя четырехугольник, который из-за перспективного искажения на кадре выглядит как трапеция с основанием снизу.

Для задания трапеции необходимы четыре точки на плоскости. На данном этапе можно вместо задания четырех точек задать трапецию при помощи двух прямых, с условием того, что высота трапеции равняется высоте кадра, а точка пересечения прямых находится выше, чем область кадра.

Для задания обеих прямых при помощи формулы 2.12 необходимо для каждой из них определить параметры k и b.

Согласно результатам анализа видео, боковая часть кадра занимает примерно по 33% изображения по горизонтали с каждой из сторон. Если провести вертикальную линию разделения областей и провести линию предполагаемой границы дороги, то существует некоторая точка пересечения. Если найти данную точку, то можно получить предполагаемую границу дороги повернув вертикальную прямую, взяв за опорную точку точку пересечения.

На данном этапе полученные пиксели представляют из себя признаки дороги и границы дороги, такие как ограждения, заборы и припаркованные

машины, которые не должны идти в итоговый результат. По результатам анализа видео пиксели, соседствующие с проведенной вертикальной линией, наполовину состоят из пикселей признаков и наполовину состоят из пикселей границ. Из этого можно сделать вывод, что наилучшей опорной точкой вертикальной прямой является центр тяжести этих пикселей.

Следующим шагом нужно определить, на какой угол нужно повернуть прямую, чтобы она была максимально приближена к искомой границе дороги. Так как при повороте прямой количество пикселей признаков увеличивается, а количество пикселей граничных объектов уменьшается, то среднее соотношение отсеченных пикселей к не отсечённым не должно сильно измениться. Тогда определим максимальное изменение доли до 0.2 снизу и 0.25 сверху.

Введем необходимые условия и формулы для работы алгоритма.

Расчет коэффициента k для обеих прямых будет производиться по формуле 2.14.

$$k_i = \arctan(\varphi_i) \tag{2.14}$$

Расчет коэффициента b для обеих прямых будет производиться по формуле 2.15.

$$b_i = x_{0i} \cdot k - y_{0i} \tag{2.15}$$

$$0.3 < \frac{S_{side,}}{S_{All}} < 0.33 \tag{2.16}$$

$$y_{0i} = \frac{\sum \text{index}}{S_{edge}} \tag{2.17}$$

$$0.2 < \frac{S_{side}}{S_{All}} < 0.25 \tag{2.18}$$

где $i = \overline{1,2}$,

- S_{side} – количество классовых пикселей слева (справа) от x_{0i},
- S_{All} – количество всех пикселей на кадре,

- S_{edge} – количество классовых пикселей в столбце,
- $index$ – номер строки классового пикселя из столбца.

Тогда получаем следующим алгоритм нахождения прямых с индексом i:

1. Найти такую вертикальную прямую x_0, чтобы выполнялось соотношение 2.16.
2. Определить центр масс классовых пикселей, стоящих справа (слева) от прямой по формуле 2.17.
3. На основании преобразования Хафа рассмотреть и выбрать такую прямую, чтобы выполнялось соотношение 2.18.

В результате работы алгоритма получаем параметры k_1, b_1 и k_2, b_2, каждая пара из которых задаёт прямую, приближенную к искомой границе дороги. Все пиксели, которые принадлежат трапеции, образованной этими прямыми, должны принадлежать области дороги.

3. Проектирование конечного алгоритма

3.1. Предварительная обработка изображения

Первым этапом работы алгоритма должна быть обработка входного изображения из видеопотока. На ней должна происходить обрезка кадра, снижение разрешения при помощи матрицы свертки и получения кадра в формате HSL. На большинстве изображений присутствует шум из-за низкой освещенности. Обычно используется Гауссово размытие, но в данном случае было принято решение вместо Гауссова размытия использовать снижение разрешения входного изображения, так как в них различается только сам механизм усреднения значения пикселей. Схема данного этапа представлена на рис.3.

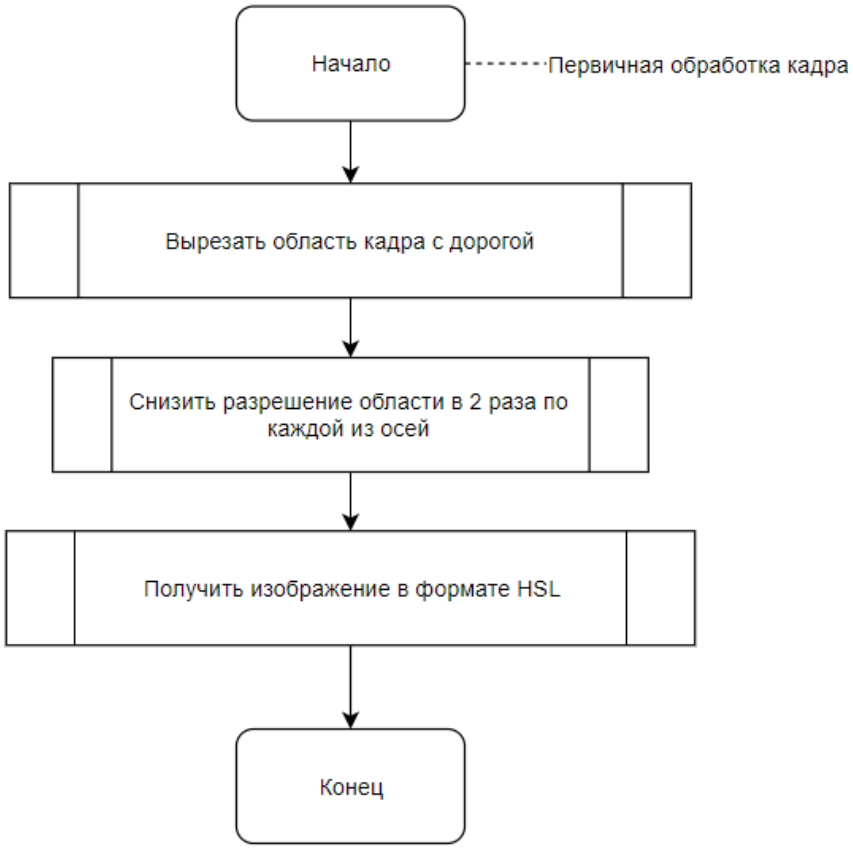

Рис.3 – Блок-схема предварительной обработки кадра.

3.2. Поиск граничных пикселей

Для второго этапа были рассмотрены следующие алгоритмы: Оператор Кэнни, метод Оцу и модифицированный оператор Собеля.

Метод Оцу не подошел для данной задачи из-за низкой точности полученного разбиения. Он мог бы использоваться как метод предобработки изображения для увеличения контрастности, но в дальнейшем все равно пришлось бы использовать оператор Собеля или оператор Кенни, что определяет его полезность как довольно низкую.

Оператор Кенни корректно находит все граничные пиксели на кадре, но из-за наличия многих дополнительных этапов проверки является алгоритмически сложным. Другим минусом оператора Кэнни является потеря информации, связанной с углом градиента в процессе обработки, которая могла использоваться для получения искомого результата. Также из-за сниженного разрешения многие границы, наблюдаемые человеческим глазом, игнорируются оператором Кэнни, из чего можно сделать вывод, что данный алгоритм не подходит для решения поставленной задачи.

Наилучшим алгоритмом был модифицированный оператор Собеля. Он берет основные операции по поиску градиента из оператора Кэнни и классифицирует пиксели согласно условиям задачи. Схема данного этапа изображена на рис.4.

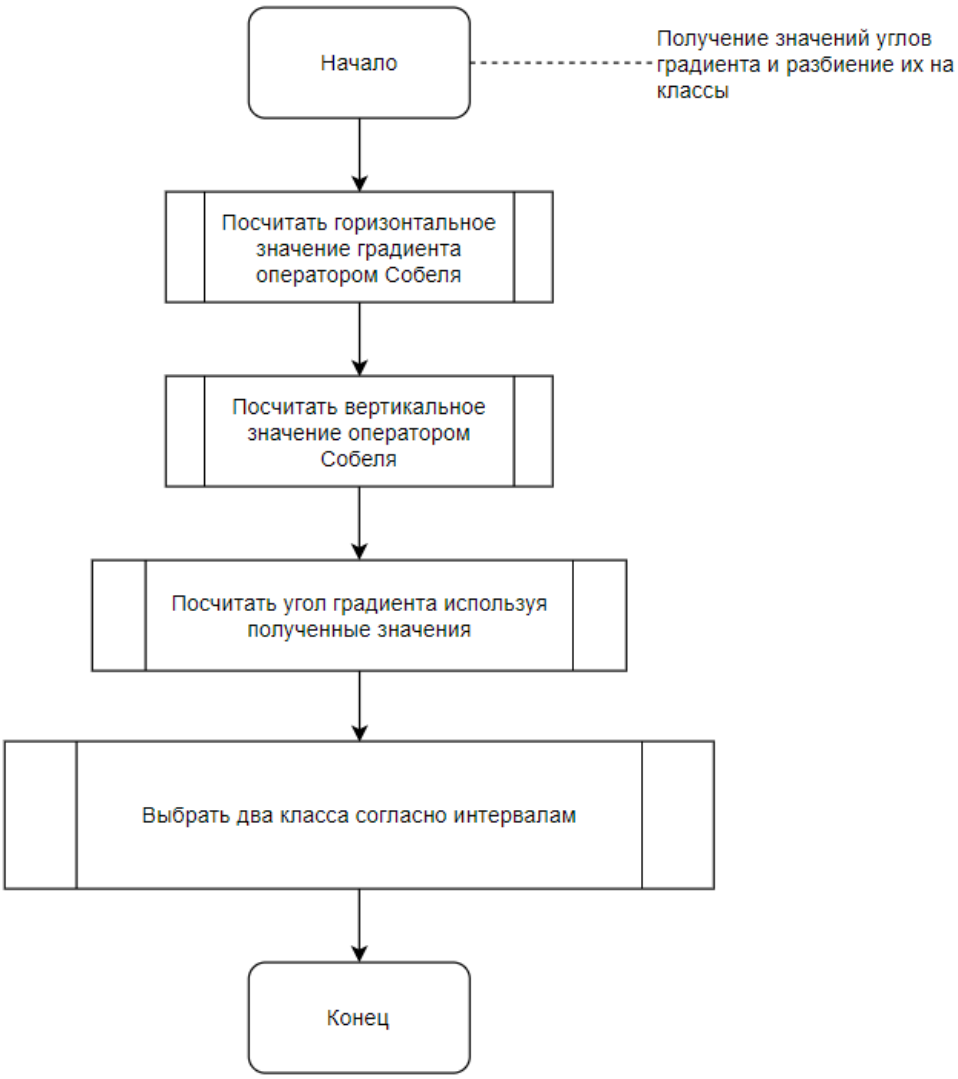

Рис.4 – Блок-схема расчета углов градиента и разбиения их на классы.

3.3. Кластеризация граничных пикселей

Следующим идет этап кластеризации точек. В нем были рассмотрены методы k-means, метод покрывающего дерева и метод, основанный на идее метода DBSCAN. После более подробного изучения было выявлено, что метод k-means не подходит для данной задачи, так как кол-во предполагаемых признаков дороги заранее невозможно определить. Также метод покрывающего дерева не подошел из-за его чрезмерной сложности и большого объема памяти, который нужно задействовать.

В результате было принято решение использовать третий алгоритм, который берет основную идею из метода DBSCAN, а именно поиск соседей в радиусе определенного размера. Итоговая схема данного этапа для каждого из классов изображена на рис.5 и рис.6.

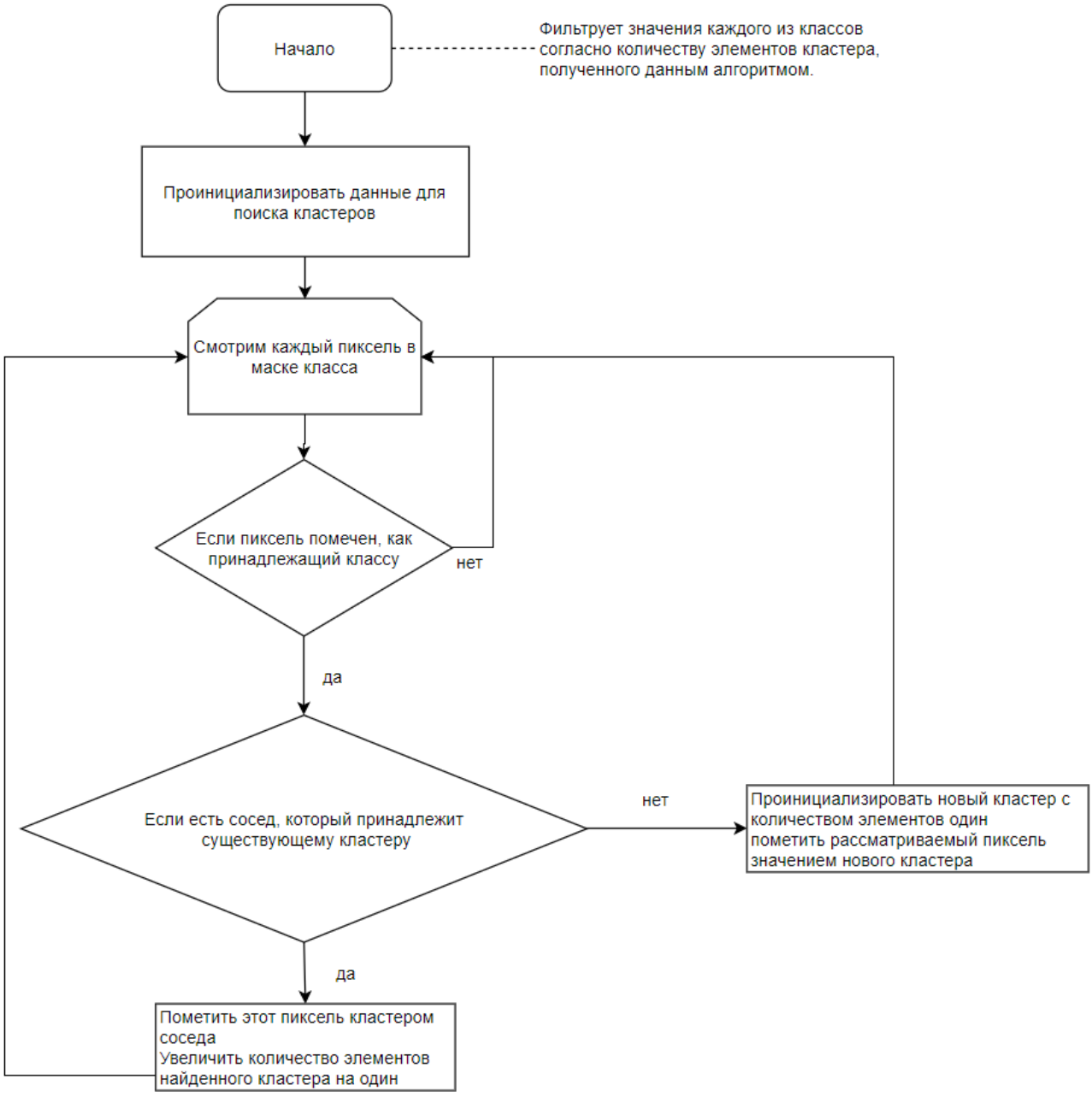

Рис.5 – Блок-схема алгоритма кластеризации, инициализация кластеров.

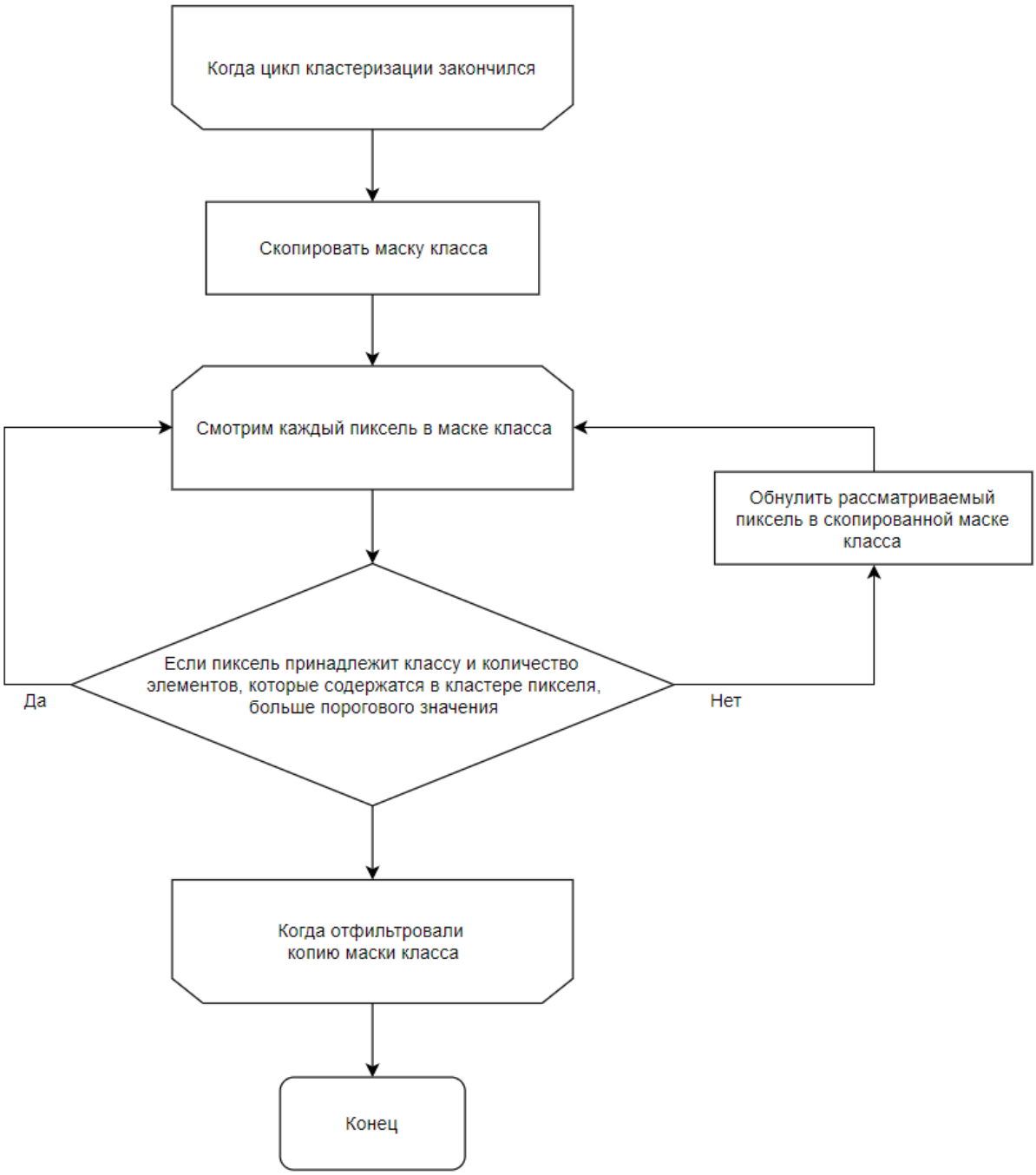

Рис.6 – Блок-схема алгоритма кластеризации, фильтрация кластеров.

3.4. Определение области дороги

На последнем этапе рассматривалось два алгоритма для объединения кластеров в области дороги.

Объединение на основе преобразование Хафа было отклонено, потому при слишком низком количестве пикселей из классов невозможно построить определенную область дороги, а при слишком большом алгоритм имеет слишком высокую алгоритмическую сложность, которая зависит линейно от количества пикселей в классах.

В результате было принято решение использовать алгоритм построения трапеции, основанный на соотношении найденных пикселей классов. Данный алгоритм имеет низкую алгоритмическую сложность и опирается на признаки, выявленные в поставленной задаче. Схема поиска прямых изображена на рис.7 и рис.8. Схема фильтрации маски изображена на рис.9.

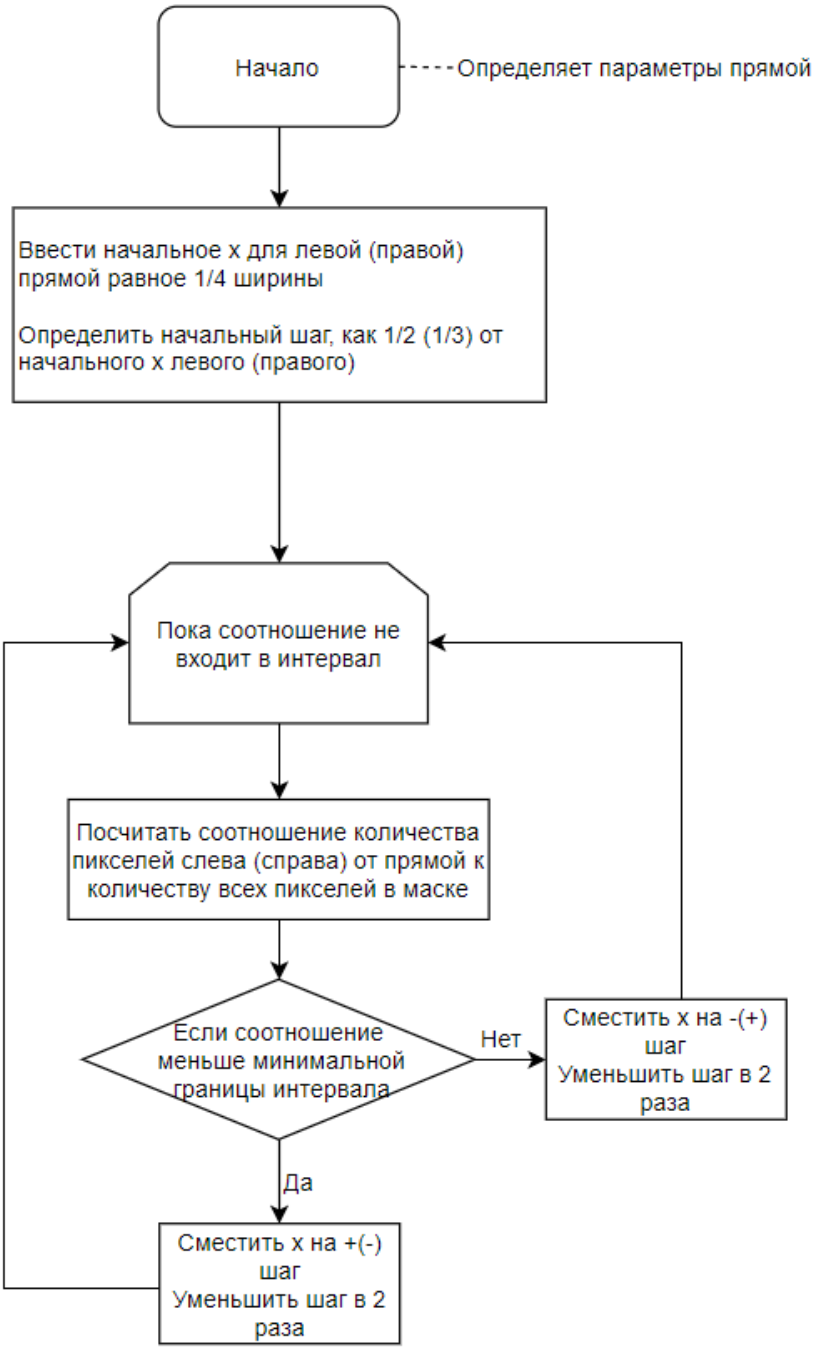

Рис.7 – Блок-схема поиска левой/правой прямой трапеции, поиск опорной точки.

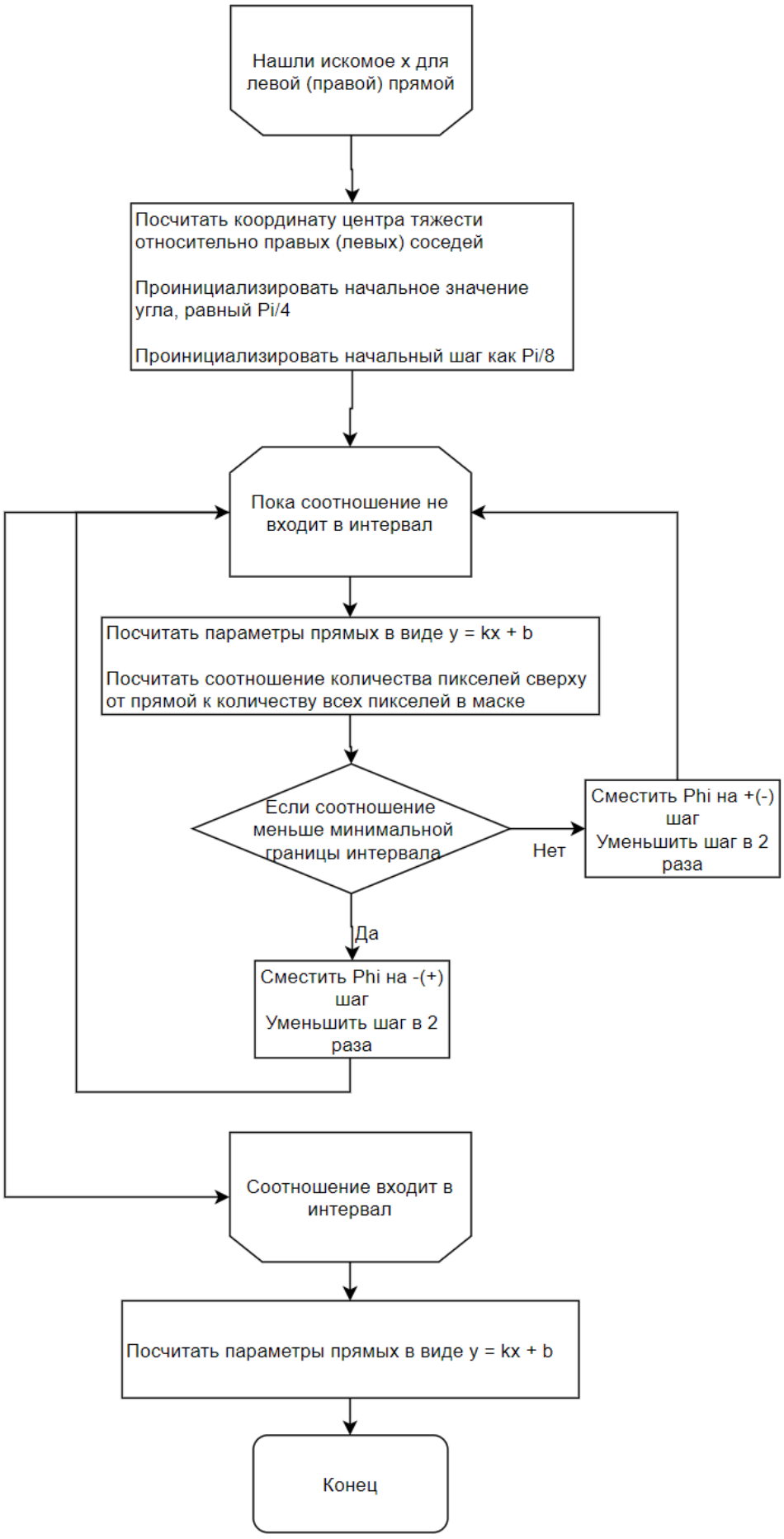

Рис.8 – Блок-схема поиска правой/левой прямой, определение конечный параметров b и k.

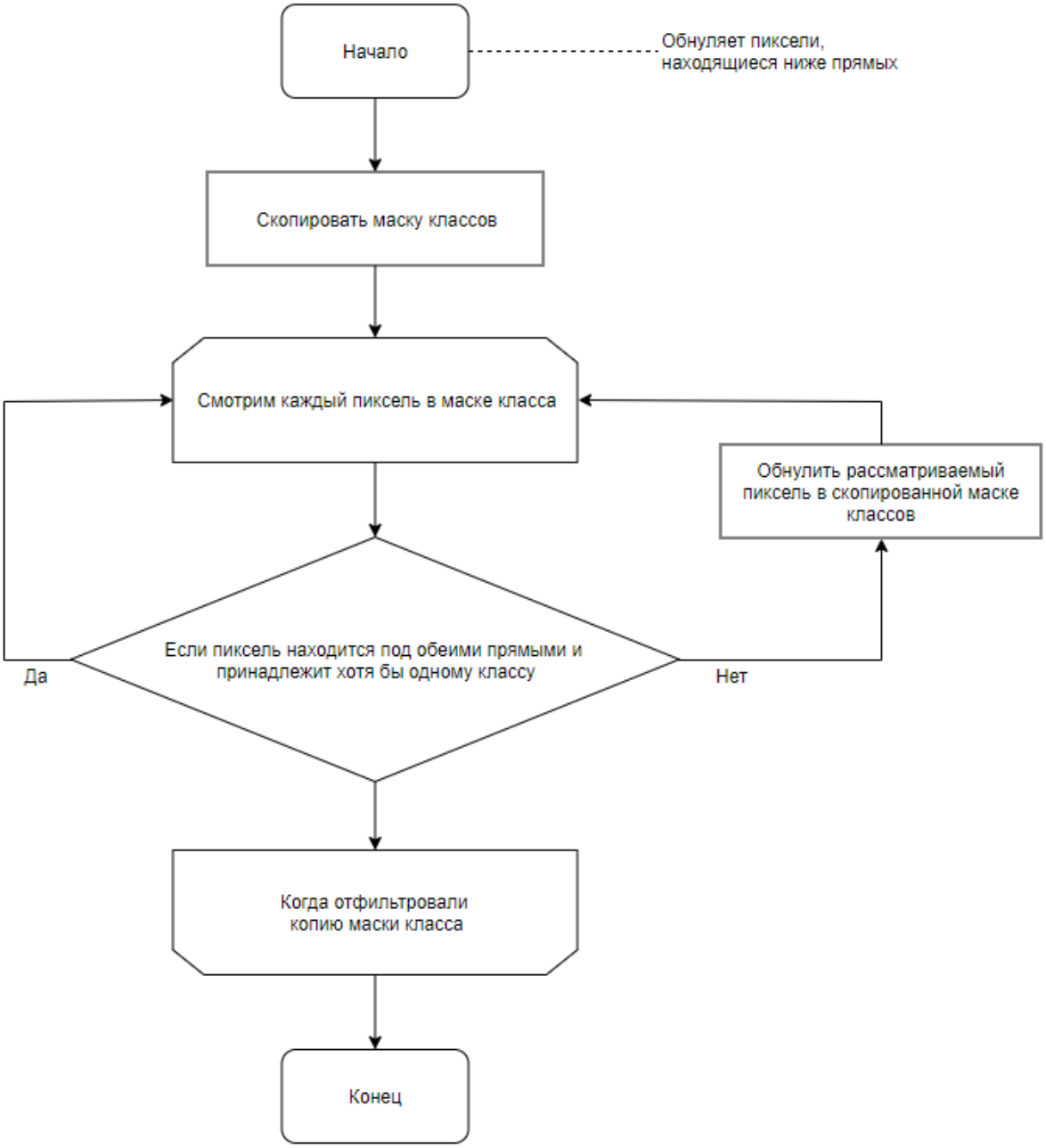

Рис.9 – Блок-схема применения правой/левой прямой, фильтрация маски по условию.

3.5. Финальная структура алгоритма

Если использовать последовательность этапов только для одного разбиения классов, то тогда получится определить признаки дороги, которые преимущественно сонаправлены траектории движения. Для определения соседних полос и границ дороги необходимо добавить обработку смещенных классов, после чего полученные результаты соединить и применить поиск границ дороги. Итоговая схема полученного алгоритма изображена на рис.10.

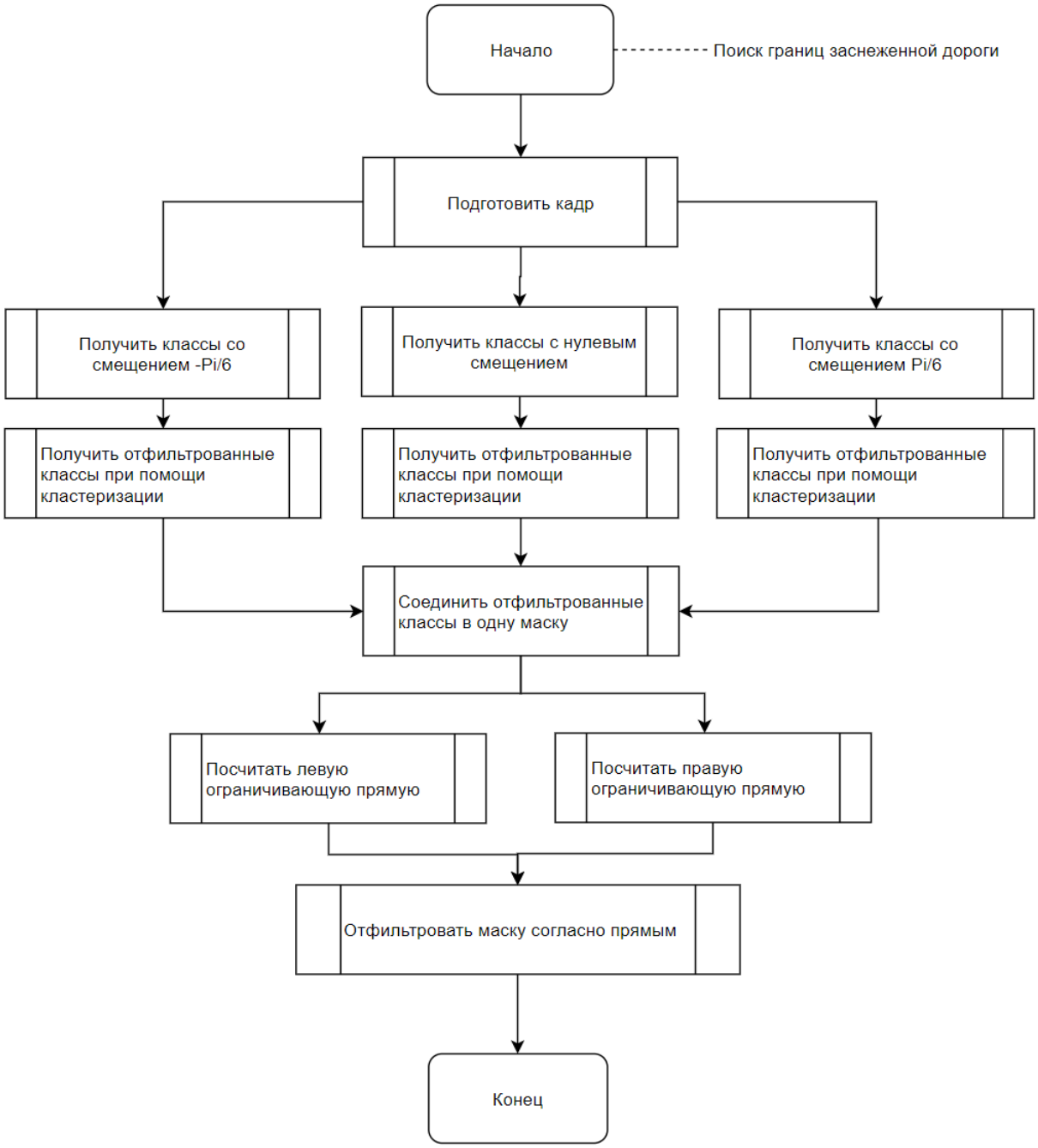

Рис.10 – Блок-схема итогового алгоритма.

4. Анализ сложности алгоритма

Необходимо проанализировать спроектированный алгоритм, чтобы понять его асимптотическую сложность. Для этого проанализируем каждый этап по отдельности [1].

Основным параметром, по которому можно будет оценить сложность, это разрешение обрезанного изображения, так как все расчеты будут выполняться на основе фрагмента с дорогой. Обозначим ширину входного изображения параметром n, а высоту входного изображения параметром m.

4.1. Анализ предварительной обработки изображения

На этапе предобработки происходят следующие шаги:

1. Обрезка изображения.
2. Снижение разрешения изображения.
3. Переход в цветовое пространство HSL и выделение канала L.

Обрезка изображения представляет из себя копирование подматрицы размера m на n. Если взять операцию копирования значения матрицы за единицу, то полученная сложность будет mn.

Операция снижения разрешения представляет из себя усреднение соседних пикселей при помощи умножения каждой подматрицы изображения на матрицу свертки. Тогда основная сложность в данном шаге будет заключаться в количестве умножений матриц. Сложность умножения матриц составляет n^3. Количество умножений зависит от размера самой матрицы свертки. При максимальном размере матрицы свертки будет m, тогда минимальное количество умножений составляет $m-(n-1)$. Если размер матрицы свертки будет меньше высоты изображения, тогда количество умножений будет $(m-k-1)(n-k-1)$, где k-размер матрицы свертки. Тогда итоговая сложность будет $(m-(k-1))(n-(k-1))k^3$.

Согласно формуле 1.3 можно получить значение канал L при помощи двух стандартных операций. Тогда нужно получить данное значение у каждого пикселя. В таком случае сложность составит $2mn$.

Тогда итоговая сложность всего этапа будет $mn + \big(m-(k-1)\big)\big(n-(k-1)\big)k^3 + \frac{2mn}{k^2}$, или асимптотически округляя получаем сложность mnk^3.

4.2. Анализ поиска граничных пикселей

На этапе поиска градиента происходят следующие шаги:

1. Определение значений при помощи оператора Собеля.
2. Расчет угла градиента.
3. Разбиение на 2 класса точек.

Определение значений градиента представляет из себя умножение изображения на матрицу свертки оператора Собеля. Умножение происходит дважды, так что сложность данного шага равна $2(m-(k-1))(n-(k-1))k^3$, где k это размер ядра оператора.

Расчет угла градиента представляет из себя операцию $\arctan\left(\frac{y}{x}\right)$, которую нужно применить к каждому элементу обеих матриц размера mn. Тогда сложность данного шага будет $2mn$.

Разбиение на 2 класса точек представляет из себя обход матрицы с копированием элементов, удовлетворяющих условию принадлежности к классам. Тогда необходимо будет провести $2mn$ сравнений.

Тогда итоговая сложность всего этапа будет $mn + 2\big(m-(k-1)\big)\big(n-(k-1)\big)k^3 + 2mn$, или асимптотически округляя получаем сложность mnk^3.

4.3. Анализ кластеризации граничных пикселей

На этапе кластеризации происходят следующие шаги:

1. Поиск соседних кластеров относительно пикселя.
2. Фильтрация кластеров.

Поиск соседних кластеров заключается в рассмотрении уже пройденных пикселей в радиусе один. Так как всего соседей у пикселя восемь, количество пройденных пикселей одинаково и равно четырем, то есть нужно для каждого пикселя провести по четыре сравнения. Итоговая сложность $4mn$ для каждого из классов.

Далее необходимо проверить, содержат ли кластеры достаточное количество пикселей для их дальнейшего рассмотрения. Для этого нужно рассмотреть каждый пиксель еще раз и посмотреть размер кластера, к которому он принадлежит. Итоговая сложность mn для каждого из классов.

Тогда итоговая сложность всего этапа будет $30mn$, или асимптотически округляя получаем сложность mn.

4.4. Анализ объединения кластеров различных классов

При объединении классов происходит обход всех трех матриц размера m на n. В результирующую маску указывается значение, если оно есть хотя бы в одной из трех матриц. Результирующая сложность $3mn$, или асимптотически округляя mn.

4.5. Анализ проецирования результата на кадр записи

На данном этапе происходят следующие шаги:

1. Расчет параметров левой ограничивающей прямой.
2. Расчет параметров правой ограничивающей прямой.
3. Фильтрация итоговой маски согласно ограничивающим прямым.

Сложности поиска первой и второй прямой будут совпадать.

Для начала необходимо подсчитать, сколько всего пикселей классов содержится в маске размера mn, сложность mn. Поиск значения x_0 для прямой начинается с точки $\frac{n}{4}$ и продолжается итеративно с последовательным уменьшением шага в 2 раза. Начальный размер шага равен $\frac{n}{8}$. Минимальный размер шага равен 1. В итоге данный под-этап остановится за $log_2 \frac{n}{8}$ итераций. Для проверки каждой итерации понадобится пересчитывать новое количество пикселей, расположенный слева от прямой. Сложность данный процедуры составляет в среднем $\frac{mn}{4}$.

Следующим шагом будет поиск подходящего угла. Так как размер угла аналогично уменьшается в 2 раза с каждой итерацией и является вещественным числом, то для защиты от бесконечного деления было принято решение установить максимум пять итераций алгоритма. За пять итераций максимальный угол по модулю может составить $\frac{63\pi}{64}$, что является более чем достаточным значением. Тогда единственное, от чего может зависеть сложность при поиске угла – просмотр и подсчет значений массива, которые находятся выше прямой и имеют значение. Среднее количество ячеек, которое будет просматриваться при каждом сдвиге равно $\frac{mn}{3}$. Тогда сложность поиска угла составит $\frac{5mn}{3}$. Итоговая сложность поиска одной прямой следующая: $mn + log_2\left(\frac{n}{8}\right)\frac{mn}{4} + \frac{5mn}{3}$.

Сложность фильтрации маски относительно полученных прямых будет равна $2mn$.

Тогда итоговая сложность поиска двух прямых и фильтрации маски классов будет следующая: $2\left(mn + log_2\left(\frac{n}{8}\right)\frac{mn}{4} + \frac{5mn}{3}\right) + 2mn + mn$, или асимптотически округляя: $log_2(n)mn$.

4.6. Финальная сложность спроектированного алгоритма

По результатам исследования итоговая сложность алгоритма получилась следующая: $mnk^3 + log_2(n)mn$. Алгоритм напрямую зависит от разрешения изображения. Также можно наблюдать кубическую зависимость от переменной k, но эта переменная отвечает за снижение разрешения входного изображения и так как ожидаемое снижение будет фиксированным, то данную переменную можно будет игнорировать. Тогда итоговая асимптотическая сложность алгоритма будет $log_2(n)mn$.

5. Реализация алгоритма

5.1. Обзор используемых библиотек

Для реализации алгоритма был выбран язык программирования Python. Основным аргументом при выборе языка была скорость реализации программного обеспечения. Также для данного языка существует множество библиотек, которые можно использовать при решении разного рода задач.

Основными библиотеками, с помощью которых будет решаться данная задача, являются библиотеки python-OpenCV и NumPy.

Python-OpenCV представляет из себя программную оболочку скомпилированной библиотеки opencv, написанной на языке C++, что даёт возможность использовать полная функциональность данной библиотеки со скоростью разработки языка Python и скоростью выполнения языка C++.

NumPy представляет из себя удобный интерфейс операций над многомерными массивами данных, который имеет достоинства, аналогичные Python-OpenCV.

5.2. Оптимизация алгоритма

Узким местом алгоритма на уровне операционной системы является интерпретатор языка Python. Также другим узким местом является Global Interpreter Lock (GIL) [6] [13].

GIL является простым способом избежать конфликтов при обращении нескольких потоков к одному и тому же участку памяти единовременно. Он используется в интерпретаторе языка Python и отвечает за синхронизацию потоков. Из-за этого достичь настоящей параллельности в языке Python нельзя.

Для ускорения работы алгоритма было проведено исследование и были найдены следующие библиотеки, ускоряющие язык Python: Cython [14], PyPy [15], Pyston [16], Nuitka [17], Numba [5].

Cython является надстройкой над языком Python и является особой его версией, которая может компилироваться в язык Си. Так как данная надстройка предполагает усложнение уже написанного кода, то было принято решение данный вариант не рассматривать.

PyPy использует JIT-компиляцию аналогично языку JavaScript, что подразумевает компиляцию во время выполнения программы, что может также негативно сказаться на производительности.

Pyston основан на проекте LLVM [8] и использует JIT-компиляцию. Данная библиотека больше подходит для поставленной задачи, но её основной минус заключается в слабой поддержке новых версий языка Python.

Nuitka представляет из себя компилятор языка Python в высокопроизводительный код языка C++, что может позитивно сказаться на производительности. Сложность использования Nuitka заключается в настройке окружения для компиляции.

Numba представляет из себя JIT-компилятор языка Python, который в свою очередь основан проекте LLVM. Она анализирует исходный код, оптимизирует его и компилирует в высокопроизводительный машинный код. Также основное преимущество данного компилятора в том, что итоговый код выполняется на более низком уровне абстракции, что позволяет обойти GIL и достигнуть настоящей параллельной работы автоматически.

В результате было принято решение использовать библиотеку Numba, так как она требует минимального времени на внедрение в проект, довольно высокую производительность и возможность скомпилировать код заранее.

Производительность на программном уровне зависит от разрешения входного изображения и степени сжатия входного изображения.

6. Метрики качества

6.1. Тестовая запись

Тестирование будет производится на записи с видеорегистратора, установленного на машине, с целью приблизить работу алгоритма к реальным условиям.

- Условия освещения: искусственное, состоящее из уличного освещения и автомобильных фар.
- Место, где была произведена запись: город.
- Наличие осадков в виде снега на дороге: обильное.
- Наличие осадков на лобовом стекле: незначительное, но заметное.
- Разрешение видео: 1280 на 720.
- Количество кадров в секунду: 30.
- Формат файла: mp4.
- Продолжительность видео: 2 минуты 51 секунда.
- Количество кадров: 5130.
- Лимит по времени обработки одного кадра: 33 миллисекунды.

6.2. Профилирование реализованного алгоритма

Для профилирования алгоритма будет использоваться стандартная библиотека языка Python time. Само время будет замеряться следующим образом: будет выделятся участок каждого этапа в программе, через библиотеку time будет взято системное время перед выполнением блока кода и далее будет взято системное время после блока кода. Полученная разница, называемая временем выполнения, будет суммироваться и в дальнейшем делиться на количество обработанных кадров для усреднения результата.

Также будет замеряться время выполнения суммарного блока кода с целью измерить промежуточные накладные расходы.

Алгоритм будет тестироваться на процессоре *Intel core i7 3770k*

Подробные характеристики процессора:

- Тактовая частота процессора 3.5 ГГц.
- Максимальная частота с учетом технологии Turbo Boost: 4.2 ГГц.
- Количество физических ядер: 4.
- Количество логических ядер: 8.
- Микроархитектура Sandy bridge.

6.3. Качественные метрики алгоритма

Алгоритм находит область дороги, в которой содержатся признаки дороги. Для оценки качества полученных результатов необходимо вручную разметить границы дороги на тестируемом видео и сравнить найденные человеком границы и найденные алгоритмом. Результат сравнения можно использовать для оценки.

Объектом классификации будет выступать пиксель, который либо будет относится к классу дороги, либо нет. Сама оценка качества будет происходить при помощи метрик Precision, Recall, False Positive Rate и F-меры.

Для данной оценки необходимо ввести следующие термины и формулы:

- TP – истинно положительное решение
- TN – истинно отрицательное решение
- FP – ложно положительное решение
- FN – ложно отрицательное решение
- $Precision = \frac{TP}{TP + FP}$ – точность
- $Recall = \frac{TP}{TP + FN}$ – полнота
- $FPR = \frac{FP}{FP + TN}$ – доля неверного предсказания
- $F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$ – F – мера

$F-$мера позволяет агрегировать результаты $Precision$ (точность относительно всех положительно предсказанных пикселей) и $Recall$ (точность, относительно всех пикселей искомых объектов) при помощи среднего гармонического с параметром β. При помощи этой метрики можно посчитать точность классификации искомых объектов.

7. Результаты тестирования

7.1. Результаты оптимизации алгоритма

Профилирование алгоритма проводилось указанным выше способом. Так как узким местом в производительности являлся интерпретатор языка Python, то проводилось тестирование как без использования компилятора Numba, так и с ним.

Результаты профилирования приведены в таблице 1 и таблице 2.

	Язык Python, значение в миллисекундах	**Numba, значение в миллесекундах**
Первичная обработка кадра	0.99 ± 0.059	0.99 ± 0.06
Получение значений градиента по осям Ox и Oy при помощи оператора Собеля	0.99 ± 0.06	0.99 ± 0.06
Расчет угла градиента	486.98 ± 15.58	1.99 ± 0.12
Разбиение пикселей на классы согласно углу градиента	255.99 ± 8.19	1.99 ± 0.12
Кластеризация каждого класса	267.99 ± 8.58	1.00 ± 0.06

Фильтрация маски при помощи ограничивающих прямых	431.39 ± 13.80	1.00 ± 0.06

Таблица 1- Среднее время работы простых этапов алгоритма.

	Язык Python, значение в миллисекундах	**Numba, значение в миллесекундах**
Суммарное время разбиения и фильтрации кластеризации классов по трем смещениям	1909.94 ± 65.15	10.99 ± 1.35
Совокупность времени выполнения всех операций, начиная с считывания кадра из видеопотока до получения финальной маски дороги	2586.92 ± 82.97	19.99 ± 2.85

Таблица 2 - Среднее время работы сложных этапов алгоритма.

По полученным результатам профилирования можно сказать, что Numba, как и ожидалось, не смогла увеличить производительность библиотек, написанных на языке C++. Но на всех остальных этапах она позволила значительно увеличить скорость обработки кадра. Можно сделать вывод, что простые этапы со сложной логикой дали больший прирост производительности чем сложные этапы и простые этапы с простой логикой. Минимальный прирост производительности составил 128 раз. Максимальный прирост производительности составил 431 раз. В среднем прирост производительности на простых этапах составил 267 раз, а на сложных этапах составил около 150 раз. Суммарное время обработки кадра было уменьшено в 133 раза, что можно считать позитивным результатом.

Среднеквадратичное отклонение составило менее трёх миллисекунд, что говорит нам о том, что среднее время обработки кадра в большинстве случаев будет лежать в интервале от 17 до 23 миллисекунд.

Основная цель оптимизации была достигнута, время обработки кадра стало меньше максимального доступного времени и было равно 20 миллисекундам.

7.2. Качественные результаты алгоритма

Для оценки качества работы алгоритма вручную была размечена тестовая запись. Так как границы дороги представляют из себя сложные объекты, то было сделано два варианта разметки:

- Разметка, в которой область дороги ограничена прямыми, внутри которых автомобиль может ехать без осложнений.
- Разметка, в которой область включает в себя припаркованные машины, бордюр и заборы, расставленные вдоль дороги.

Оценивать будем на основе маски граничных пикселей, после выделения области и маски до выделения области. Если пиксель есть в выделенной области, и он находится между линиями разметки, то относим его к классу $True\ Positive$. Если пиксель находился снаружи от линий разметки и в самой области, то он относится к классу $False\ Positive$. Если пиксель не находится в области, но находится между линиями разметки, то относим его к $False\ Negative$. Если он находится за границами области и за линиями разметки, то относим его к $True\ Negative$.

Далее после просмотра всего кадра подсчитываем характеристики $Precision,\ Recall,\ FPR$ и $F-$меру, после чего проводим аналогичные вычисления для всех других кадров тестируемого видео.

Результаты тестирования относительно первого варианта разметки изображены на рисунках 11-14.

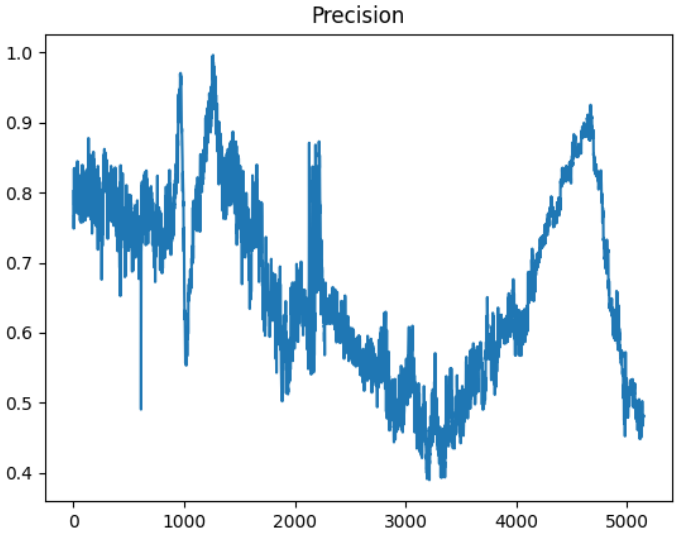

Рис. 11 – График зависимости Precision от номера кадра.

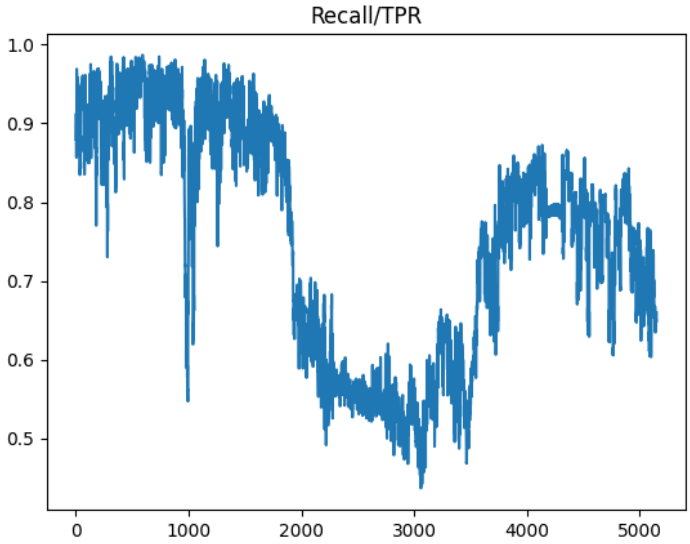

Рис. 12 – График зависимости Recall от номера кадра.

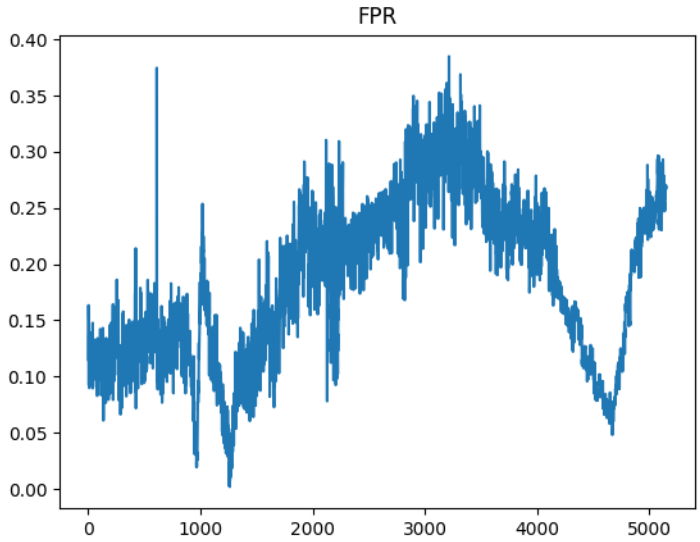

Рис. 13 – График зависимости FPR от номера кадра.

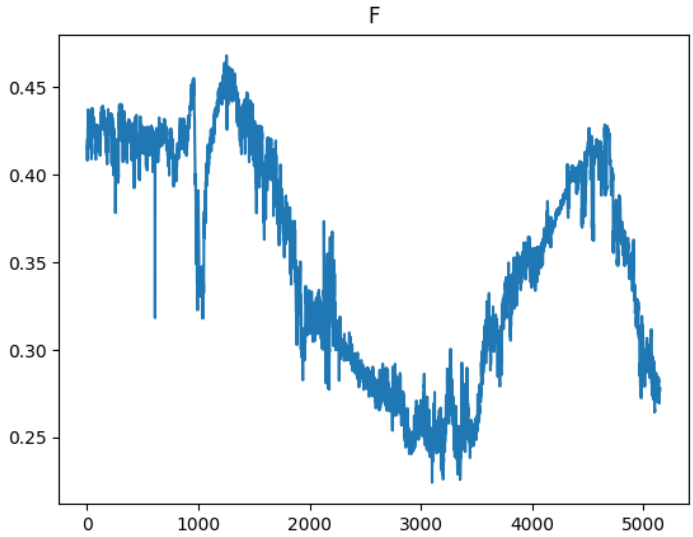

Рис. 14 – График зависимости F-меры от номера кадра.

Средние результаты метрик следующие:

- $Precision$: 0.669
- $Recall$: 0.753
- $F-$мера: 0.352
- FPR: 0.185

Согласно графикам, начиная с 2000 кадра до 4000 кадра идет значительный спад точности по всем метрикам. Это связано с тем, что на этих

кадрах в исходном видео за признаки дороги принимаются края припаркованных машин, которые находятся на крайней правой полосе движения. В разметке эталонной записи данные участки помечены как области за дорогой, из-за чего и произошло снижение качества распознавания.

Результаты точности относительно второго варианта разметки изображены на рисунках 15 – 19.

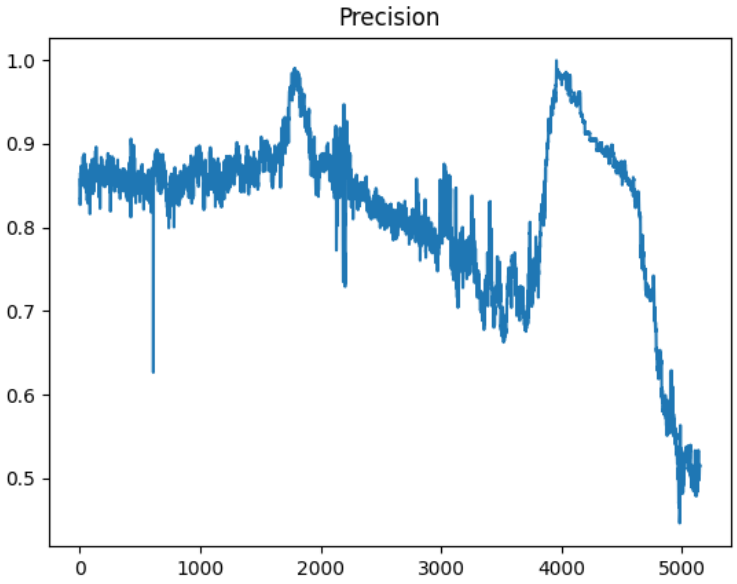

Рис. 15 – График зависимости Precision от номера кадра.

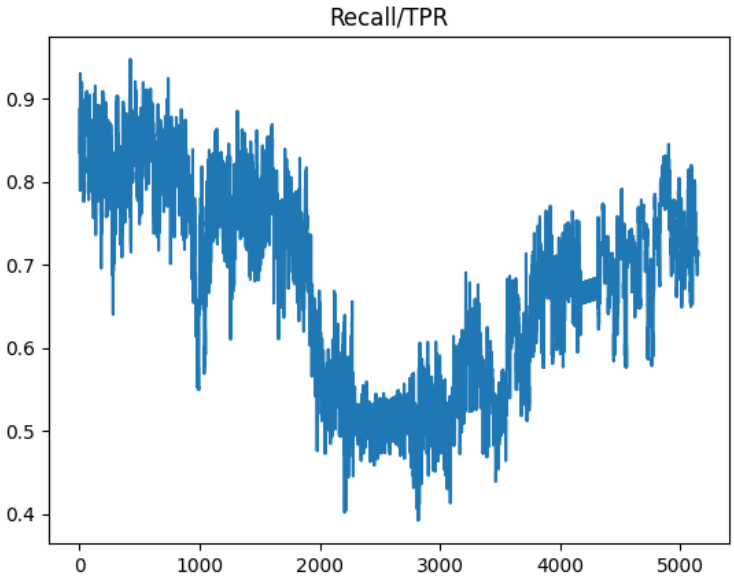

Рис. 16 – График зависимости Recall от номера кадра.

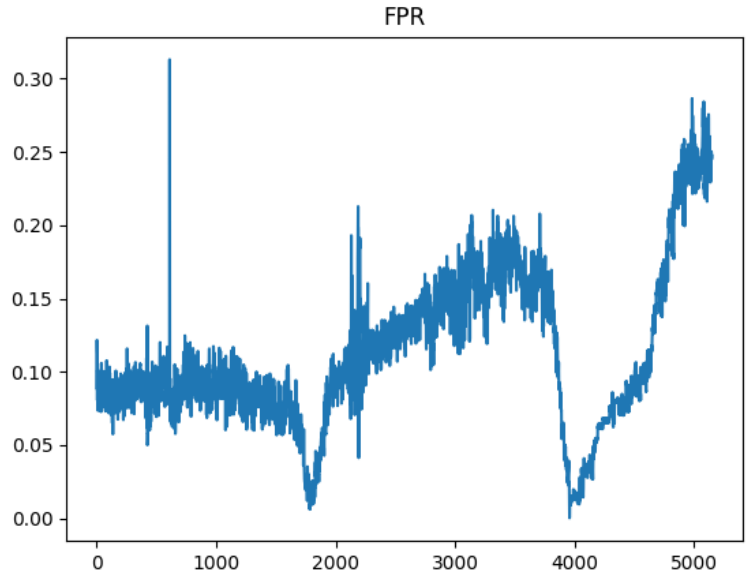

Рис. 17 – График зависимости FPR от номера кадра.

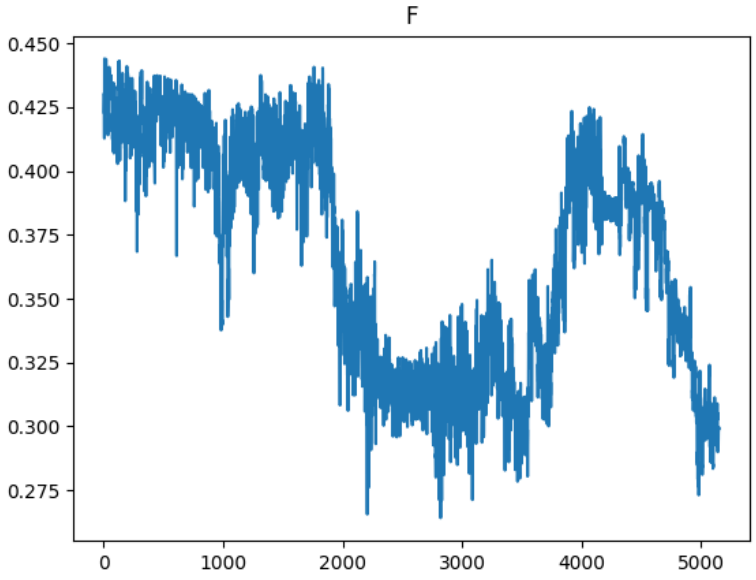

Рис. 18 – График зависимости F-меры от номера кадра.

Средние результаты метрик, следующие:

- $Precision$: 0.821
- $Recall$: 0.678
- $F-$мера: 0.367

- FPR: 0.113

При втором варианте эталона падение точности, которое наблюдалось в первом варианте, значительно снизилось. Повысилась точность распознавания самих пикселей и снизилась доля неверно классифицированных пикселей. Также снизилась степень полноты распознавания.

В итоге можно сделать вывод, что алгоритм корректно распознает в среднем более 70 % области дороги. Процент ошибочно классифицированных пикселей составляет от 10% до 20%, что также является следствием различных интерпретаций границ дороги.

Сложность интерпретации результатов заключается в вопросе включения объектов-границ в область дороги. Так как алгоритм получает область дороги, заданную при помощи прямых, то для степени уверенности можно изменять ширину полученной области: при максимальной ширине в область попадают припаркованные автомобили и ограждения, а при минимальной ширине в область будет только полоса движения, по которой едет сам автомобиль.

7.3. Поэтапная демонстрация алгоритма

Поэтапные преобразования кадра изображены на рисунках 19-25.

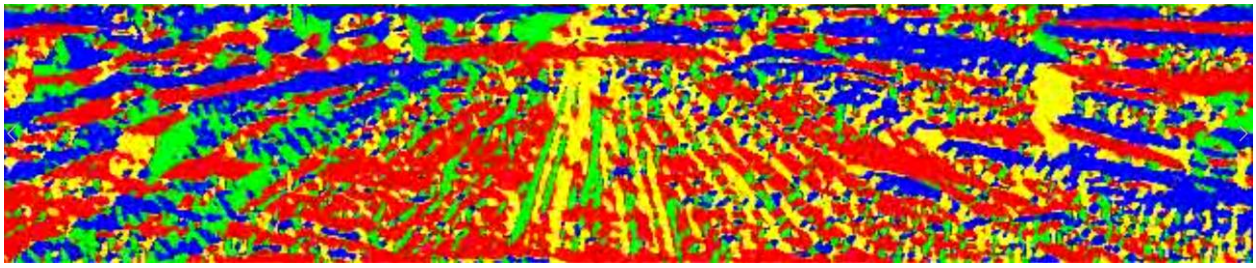

Рис.19 – Вид кадра после разбиения на четыре класса согласно углу градиента, где биссектриса класса сонаправлена с осью Oy.

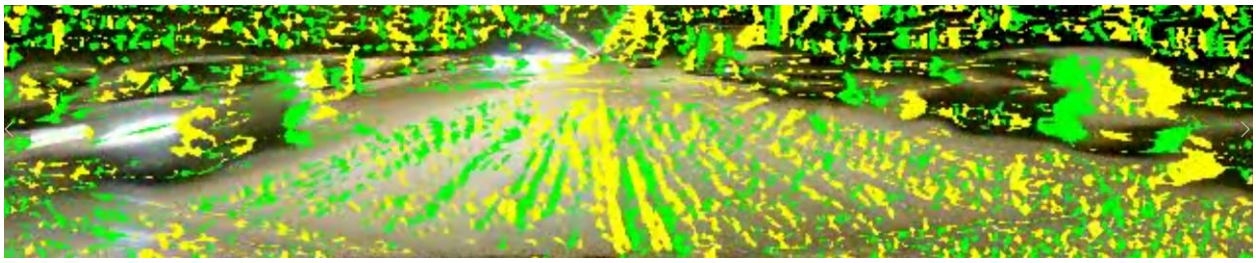

Рис.20 – Вид кадра после проецирования двух классов с нулевым смещением на исходный кадр без фильтрации.

Рис.21 - Вид кадра с классами после фильтрации с нулевым смещением.

Рис.22 - Вид кадра с классами после фильтрации со смещением $\frac{\pi}{3}$.

Рис.23 - Вид кадра с классами после фильтрации со смещением $-\frac{\pi}{3}$.

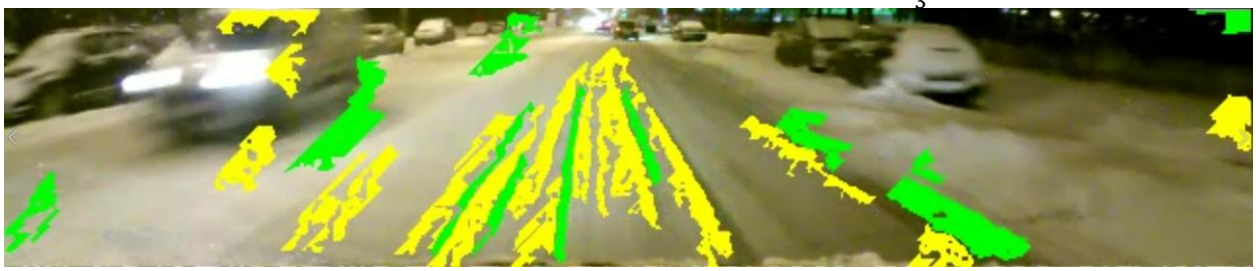

Рис. 24 - Вид кадра после проецирования отфильтрованных классов со всеми видами смещений.

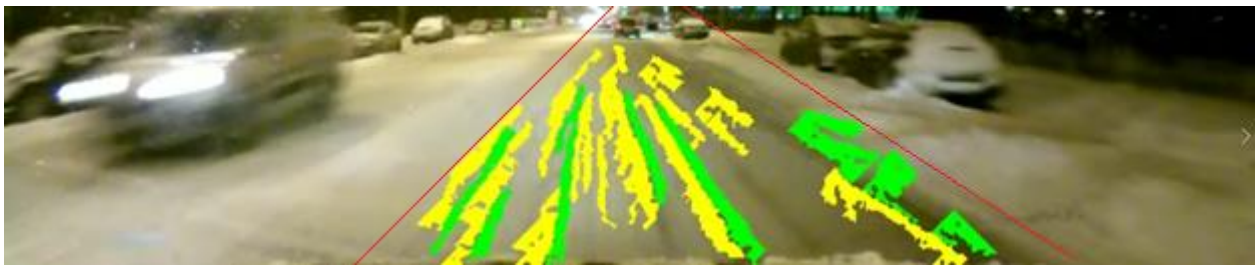

Рис. 25 – Вид кадра после фильтрации методом ограничивающих прямых с отрисованными прямыми.

На рисунке 25 изображена найденная алгоритмом область. За границы дороги можно принять полученные в процессе ограничивающие прямые. Также можно заметить, что правая прямая выходит за наблюдаемую область дороги, что связано с наличием большого количества объектов границы. С левой прямой ситуация обратная.

После проведенных экспериментов можно сказать, что алгоритм решает поставленную задачу с достаточной точностью и удовлетворяет ограничениям времени выполнения. Но также он требует внесения определенных модификаций перед внедрением в системы ADAS, таких как:

1. Первоначальное определение области кадра, зависящее от местоположения камеры, которое должно определяться один раз при её установке.
2. Определение степени включения объектов-границ в саму границу.

3. Логика принятия решений систем ADAS на основе данных, получаемых данным алгоритмом.

Заключение

В ходе данной работы были достигнуты следующие результаты:

- изучены методы поиска границ на изображении и методы оптимизации алгоритмов;
- произведена декомпозиция поставленной задачи;
- согласно результатам декомпозиции выбраны алгоритмы, подходящие по критериям алгоритмической сложности и качества;
- спроектирован, реализован и оптимизирован алгоритм распознавания границ заснеженной дороги;
- решение протестировано. Время обработки кадра составило 20 миллисекунд, точность составила 75%;

Список литературы

1. Алгоритмы, построение и анализ / Кормен Т. Лейзерсон Ч. Ривест Р. Штайн К.; перевод с англ. И.В. Красикова; под ред. канд. техн. наук И.В. Красиковой – Москва: Диалектика, 2019. – 1328 с.
2. Hough transform. [Электронный ресурс] // planetmath. – URL: https://planetmath.org/houghtransform (дата обращения: 13.05.2021).
3. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. [Электронный ресурс] // www.khoury.northeastern.edu. – URL: https://www.khoury.northeastern.edu/home/vip/teach/DMcourse/2_cluster_EM_mixt/notes_slides/revisitofrevisitDBSCAN.pdf (дата обращения: 13.05.2021).
4. Андрей Часовский. Обзор алгоритмов кластеризации данных. [Электронный ресурс] // Хабр. – URL: https://habr.com/ru/post/101338/ (дата обращения:22.03.2021).
5. Numba, documentation. [Электронный ресурс] // pydata.org. – URL: https://numba.pydata.org/ (дата обращения: 25.03.2021).
6. Initialization, Finalization, and threads. [Электронный ресурс] // docs.pythong.org – URL: https://docs.python.org/3/c-api/init.html (дата обращения: 28.03.2021)
7. Canny Edge Detection. [Электронный ресурс] // web.archive.org – URL: https://web.archive.org/web/20150421090938/http://www.cse.iitd.ernet.in/~pkalra/csl783/canny.pdf (дата обращения: 21.05.2021)
8. LLVM overview. [Электронный ресурс] // llvm.org – URL: https://llvm.org/ (дата обращения: 13.05.2021)
9. Sobel Edge detector. [Электронный ресурс] // homepages.inf.ed.ac.uk – URL: https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm (дата обращения: 13.05.2021)
10. Road Edge Detection in All Weather and illumination via Driving Video Mining. [Электронный ресурс] // researchgate.net – URL:

https://www.researchgate.net/publication/329980954_Road_Edge_Detection_in_All_Weather_and_Illumination_via_Driving_Video_Mining (дата обращения 10.05.2021)

11. N. Otsu. A threshold selection method from gray-level histograms // IEEE Trans. Sys., Man., Cyber. : journal. — 1979. — Vol. 9. — P. 62—66.
12. HSL and HSV. [Электронный ресурс] // Википедия. Свободная энциклопедия. – URL: https://en.wikipedia.org/wiki/HSL_and_HSV (дата обращения 10.05.2021)
13. Python documentation, Glossary. [Электронный ресурс] // docs.pythong.org – URL: https://docs.python.org/3/glossary.html#term-gil (дата обращения: 29.03.2021)
14. Cython documentation. [Электронный ресурс] //cython.org – URL: https://cython.org/ (дата обращения: 25.03.2021)
15. PyPy documentation. [Электронный ресурс] //pypy.org – URL: https://www.pypy.org/ (дата обращения: 25.03.2021)
16. Pyston documentation. [Электронный ресурс] //pyston.org – URL: https://www.pyston.org/ (дата обращения: 25.03.2021)
17. Nuitka documentation. [Электронный ресурс] //nuitka.net– URL: https://nuitka.net/pages/overview.html (дата обращения: 25.03.2021)
18. Alex Staravoitau. Detecting road features. [Электронный ресурс] // Alex Staravoitau Blog 2017. URL: https://navoshta.com/detecting-road-features/ (дата обращения:12.02.2021).