

Санкт-Петербургский политехнический университет Петра Великого
Институт прикладной математики и механики
Высшая школа прикладной математики и вычислительной физики

ОТЧЁТ
по преддипломной практике

**Подготовка выпускной квалификационной работы бакалавра
на тему: *Поиска пересечений линий на изображении***

по направлению 01.03.02 «Прикладная математика и информатика»
(профиль Системное программирование)

Студент гр. 360102/60201 Туников Д.А.
Оценка научного руководителя ВКР Шубников В.Г.

Санкт-Петербург
2020

Содержание

1. Введение	3
1.1. Актуальность задачи	3
1.2. Постановка задачи	3
2. Основная часть	4
2.1. Метод, основанный на преобразовании Хафа	4
2.1.1. Преобразование Хафа	4
2.1.2. Алгоритм метода	5
2.1.3. Поиск границ на изображении	5
2.1.4. Разбиение изображения на горизонтальные блоки .	8
2.1.5. Поиск рельсов	9
2.1.6. Поиск стрелок	9
2.1.7. Описание данных для тестирования	17
2.1.8. Результаты	18
Список литературы	23

1. Введение

1.1. Актуальность задачи

Задача поиска пересечений линий на изображении является очень распространенной и её решение может быть использовано в таких задачах компьютерного зрения как:

- 1) Автоматическое определение полей во время спортивных трансляций
- 2) Детектирование ориентиров при автоматическом управлении роботом
- 3) Детектирование железнодорожных стрелок на изображении, сделанном с локомотива поезда

В данной статье будут рассмотрены подходы к решению задачи применительно к детектированию ж/д стрелок.

Для автоматизации процесса управления поездом необходимо создать систему, которая позволяла бы стать надежным автоматическим ассистентом/помощником для машиниста локомотива (стать неким driving assistance), но ни в коем случае не заменить действия человека полностью. Одной из задач такой системы является обнаружение железнодорожных стрелок для обеспечения движения поезда по правильному пути. Решение этой задачи позволит обнаруживать разветвления пути заранее (визуально) и анализировать возможные дальнейшие пути движения поезда, выдавая полезную информацию для машиниста локомотива, такую как: "на правой ветке обнаружены вагоны - путь занят!". Это решение значительно повысит безопасность движения на железных дорогах.

1.2. Постановка задачи

На вход подается изображение — фотография сделанная с головы поезда. Необходимо найти на входном изображении места пересечений железнодорожных путей. Пересечением будем называть точку, в которой пересекаются рельсы одной ж/д колеи с другой. На выходе алгоритма должен быть набор точек, в которых было обнаружено пересечение.



Рис. 1. Пример входного изображения

Замечания:

- 1) Точность поиска стрелок - окрестность 30 на 30 пикселей
- 2) Необходимо правильно находить стрелки в пределах 20-30 метрах перед поездом. Нет задачи детектировать стрелки, которые находятся слишком далеко от поезда(40 метров и далее).

2. Основная часть

2.1. Метод, основанный на преобразовании Хафа

2.1.1. Преобразование Хафа

Преобразование Хафа [1] - алгоритм для поиска геометрических объектов на изображении(линии, круги и т.д). В нашем случае понадобится алгоритм Хафа для поиска прямых линий на изображении.

На вход алгоритму подается набор точек, на выходе имеется набор прямых в координатах (ρ, θ) , где ρ - расстояние от начала координат до прямой, θ - угол между перпендикуляром к прямой, проведенным из начала координат, и осью абсцисс.

Алгоритм:

Имеется массив аккумулятора $A[M, N]$, где M - квантованные значения параметра ρ , N - квантованные значения параметра θ . Изначально в каждой ячейке A находится значения ноль.

Через каждую входную точку проводятся прямые с различными параметрами $\theta \in [0, \pi]$. 2

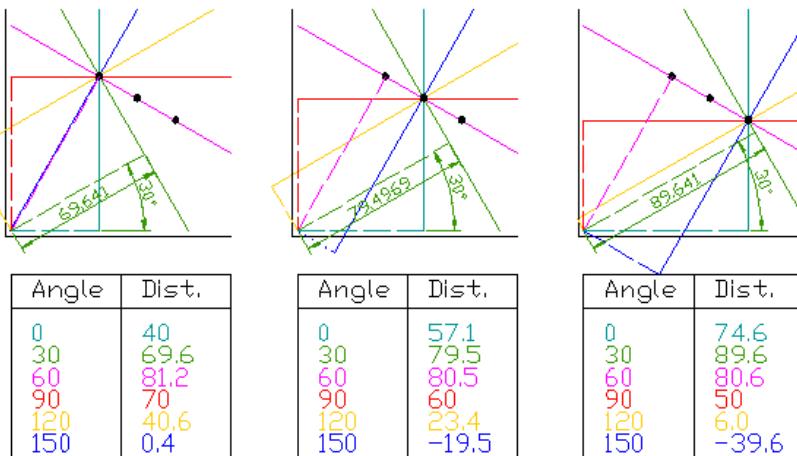


Рис. 2.

Каждая прямая голосует за свои параметры (ρ, θ) , то есть значение значения аккумулятора $A[\rho, \theta]$ увеличивается на единицу. После голосования максимумы в массиве аккумуляторе соответствуют параметрам итоговых прямых.

2.1.2. Алгоритм метода

2.1.3. Поиск границ на изображении

Первым шагом метода является поиск границ на входном изображении. Это делается с помощью алгоритма Canny [2]. Алгоритм состоит из пяти важных шагов:

- 1) Сглаживание изображения для удаления шумов. Это делается фильтрацией изображения с использованием фильтра Гаусса:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} \exp \frac{-(x^2 + y^2)}{2\sigma^2}$$

- 2) Поиск градиентов. Границы выделяются там, где находятся максимумы градиентов. Градиенты ищутся с помощью оператора Собеля, который является аналогом производной в дискретном пространстве. Псевдокод поиска градиентов с помощью оператора Собеля:

$$MGx := \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \quad MGy := \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Реализация (сопоставляет каждой точке вектор градиента):

```
SobelOperator(Matrix) := | for iY ∈ 1..rows(Matrix) - 2
                           |   for iX ∈ 1..cols(Matrix) - 2
                           |     A ← submatrix(Matrix, iY - 1, iY + 1, iX - 1, iX + 1)
                           |     GX ← ∑_{y=0}^2 ∑_{x=0}^2 (A_{y,x} · MGx_{y,x})
                           |     GY ← ∑_{y=0}^2 ∑_{x=0}^2 (A_{y,x} · MGy_{y,x})
                           |     G ← √(GX² + GY²)
                           |     θ ← round(atan2(GX, GY) / (π/4)) · π/4 - π/2 if G ≠ 0
                           |     θ ← ErrCode otherwise
                           |     SobMtrix_{iY, iX} ← G
                           |     SobMtrix_{iY, iX+1+cols(Matrix)} ← θ
                           |
                           | return SobMtrix
```

Рис. 3.

- 3) Подавление не-максимумов. Среди всех максимумов оставляем только те, что являются локальными максимумами в окрестности.
- 4) Двойная пороговая фильтрация. Если значения градиента в точке максимума ниже порога - эта точка отсекается. Таким образом, чем выше значение порога, тем меньше границ будет найдено на изображении. Фильтрация в алгоритме Canny является двойной, лучше всего этот момент разъясняет иллюстрация 4, здесь только зелёные точки будут выбраны граничными. В нашем случае значение верхнего порога вычисляется с помощью бинаризации Отцу[3]. Значение нижнего порога берется как половина от значения верхнего порога. За счёт такого выбора порога достигается нормальное выявление границ на изображения с различным уровнем освещенности.

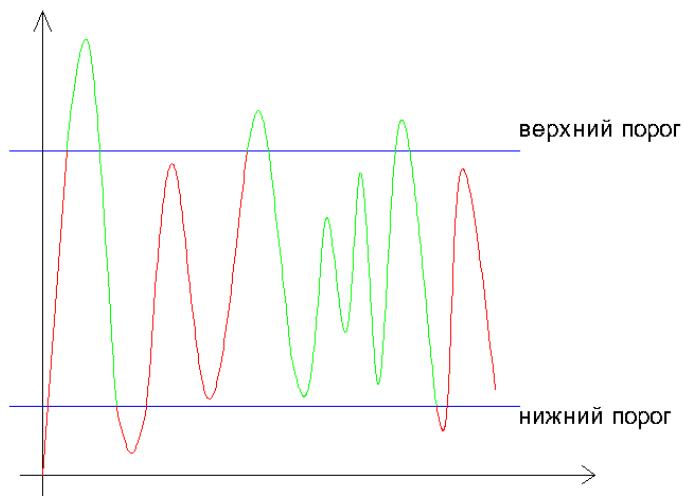


Рис. 4.

- 5) Трассировка области неоднозначности. Итоговые границы определяются путём удаления всех краёв, несвязанных с "сильными" границами. Проще говоря, пиксели, которые не относятся ни к какой группе - подавляются.

Пример применения алгоритма Canny к входному изображению:

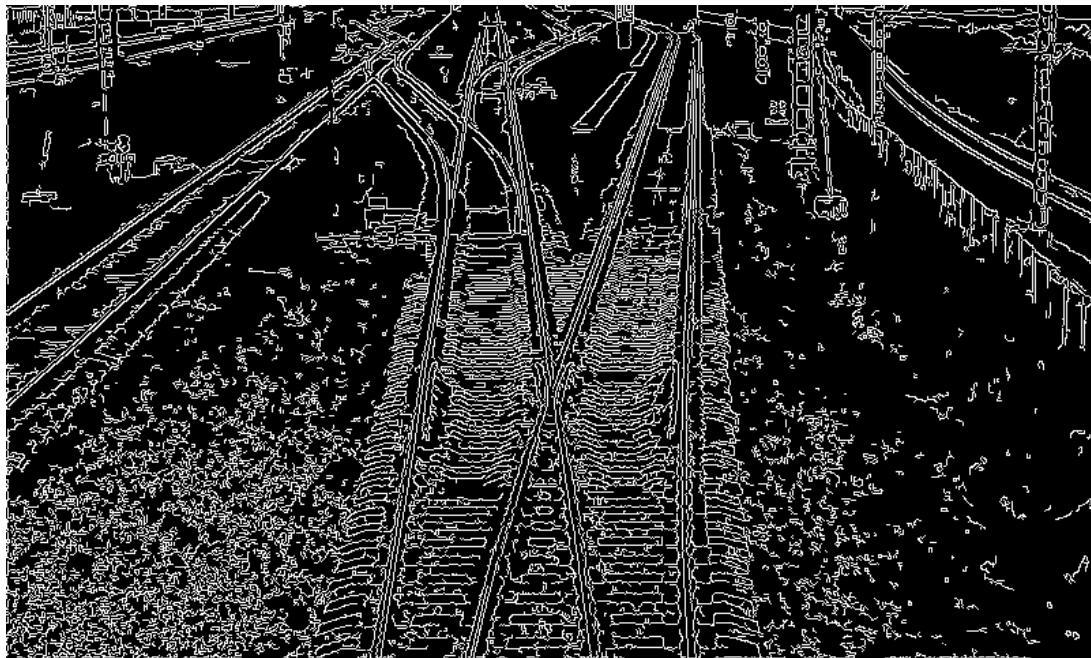


Рис. 5.

2.1.4. Разбиение изображения на горизонтальные блоки

Сначала разбиваем изображение на горизонтальные блоки б. Размер блоков уменьшается в зависимости от координаты Y на изображении, это сделано для учета перспективной проекции объектов из реального мира на изображении - чем дальше объект, тем меньше он на изображении.

Формула для расчёта размера блока в зависимости от координаты y:

$$blockSize = minBlockSize + \frac{maxBlockSize - minBlockSize}{imageHeight - maxBlockSize}y, \text{ где } minBlockSize = 7\text{pix}, maxBlockSize = 45\text{pix}$$

Данные значения были выбраны из соображений минимизации процента ошибочных предсказаний стрелок и максимизации процента предсказания реальных стрелок. В идеале размер горизонтального блока нужно рассчитывать с использованием правил перспективного проецирования[4], но для этого необходимо знать параметры камеры(угол наклона, расстояния до земли). В нашем случае эти параметры не известны, поэтому blockSize рассчитывается просто с помощью линейной интерполяции в зависимости от координаты Y.



Рис. 6.

2.1.5. Поиск рельсов

Теперь рассмотрим процесс поиска рельсов на изображении, полученном после применения алгоритма Canny5.

После разбиения б в каждом из горизонтальных блоков ищутся прямые линии с помощью алгоритма Хафа [1]. Можно заметить, что прямые близкие к горизонтальным можно отсекать, т.к. они не могут относиться к рельсам. Таким образом, будем искать прямые с помощью модифицированной алгоритма Хафа, в котором $\theta \in [0; \frac{\pi}{3}] \cup [\frac{2\pi}{3}; \pi]$. В качестве порогового значения аккумулятора будем использовать: $\frac{2 \cdot blockSize}{3}$, это нужно для отсечения маленьких прямых.

Результат применения алгоритма (красным выделены найденные прямые линии в каждом блоке):

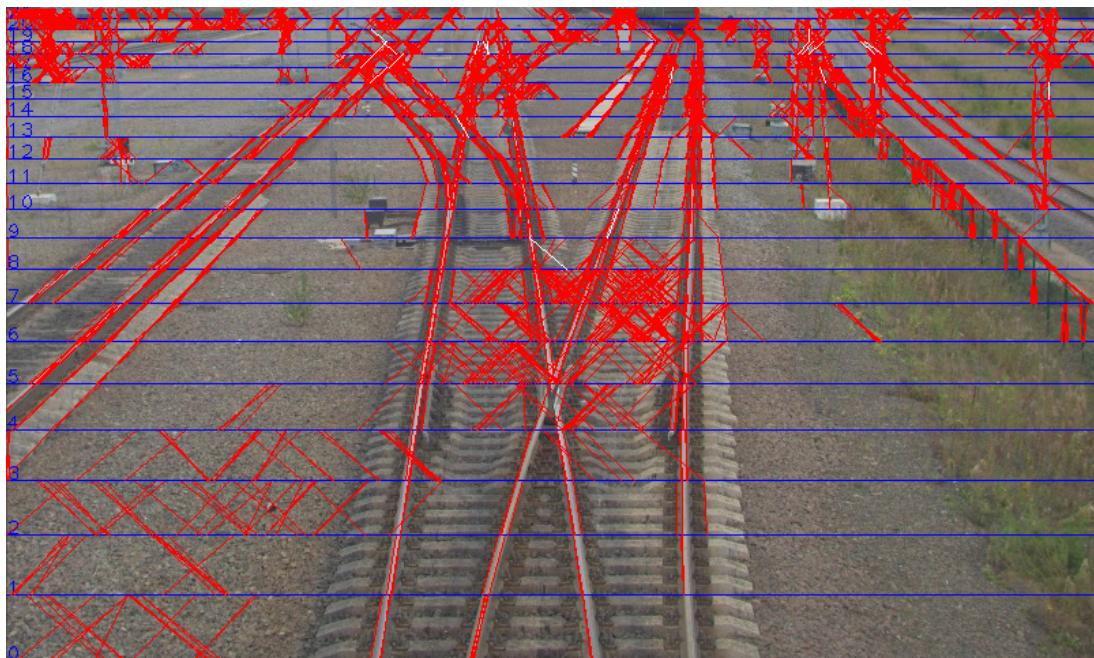


Рис. 7.

2.1.6. Поиск стрелок

Построение графа прямых Построим граф, вершинами которого будут найденные прямые. Каждая вершина будет иметь структуру:

$(p1, p2, children, parents)$, где $p1$ - нижняя точка прямой, $p2$ - верхняя точка прямой, $children$ - дети вершины(прямые, которые выходят из прямой, соответствующей текущей вершине), $parents$ - родители вершины(прямые, которые входят в прямую, соответствующую текущей вершине).

Ребра в графе будут строиться по следующему принципу: начиная с нижнего горизонтального блока, для каждой прямой(*curLine*) ищем детей в следующем блоке. Прямая(*nextLine*) будет считать ребенком, если:

$$|nextLine.p1.x - curLine.p2.x| < blockEps, \text{ где}$$

$$blockEps = minNeighsEps + \frac{maxNeighsEps - minNeighsEps}{imageHeight - maxNeighsEps} y,$$

где $minNeighsEps = 3$, $maxNeighsEps = 10$. То есть $blockEps$ линейно увеличивается в зависимости от координаты Y.

Найденные вершины добавляются, как дети, в текущую вершину. А также текущая вершина добавляется, как родитель, в каждую из дочерних вершин.

Например, на рисунке 8:

- 1) прямые 1, 2 являются детьми для прямой 4. А прямая 4 является родителем для прямых 1 и 2.
- 2) Прямая 3 лежит дальше, чем $blockEps$ от прямой 4, поэтому между ними нет связи в графе.
- 3) прямая 4 является ребенком для прямых 5 и 6, но не является ребенком для прямой 7. А прямые 5 и 6 в свою очередь являются родителями для прямой 4.

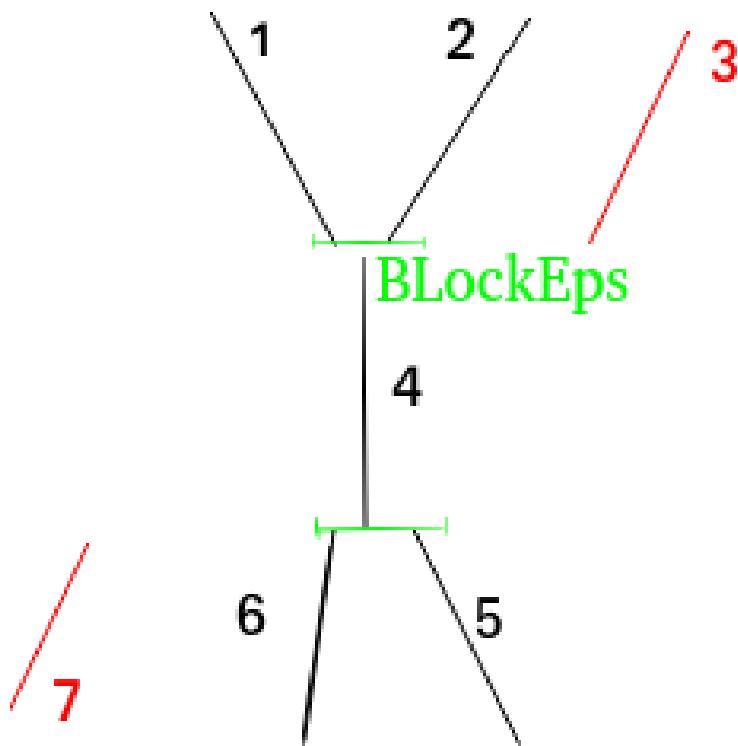


Рис. 8.

Поиск стрелок в графе В этом параграфе будет рассмотрен алгоритм поиска стрелок в построенном на предыдущем шаге графе.

Алгоритм поиска пересекающихся вершин в графе:

В цикле для каждой вершины графа производим следующие действия:

- 1) Если вершина не имеет параллельного родителя - переходим к следующей вершине
- 2) Для каждой пары детей текущей вершины проверяем, пересекаются ли они(2.1.6), если да - верхнюю точку текущей вершины добавляем в итоговый результат найденных стрелок. Таким образом, будут найдены Y и X пересечения.

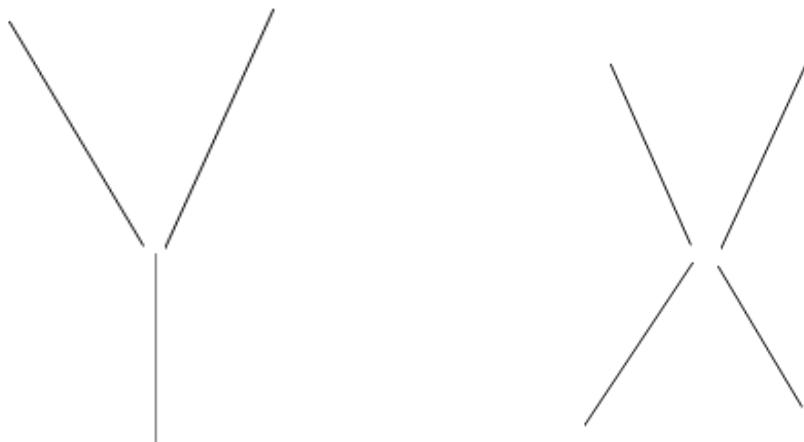


Рис. 10.

Рис. 9.

- 3) Пункт 2) повторяется для родителей текущей вершины. Таким образом, будут найдены Y - обратное и X образные пересечения.

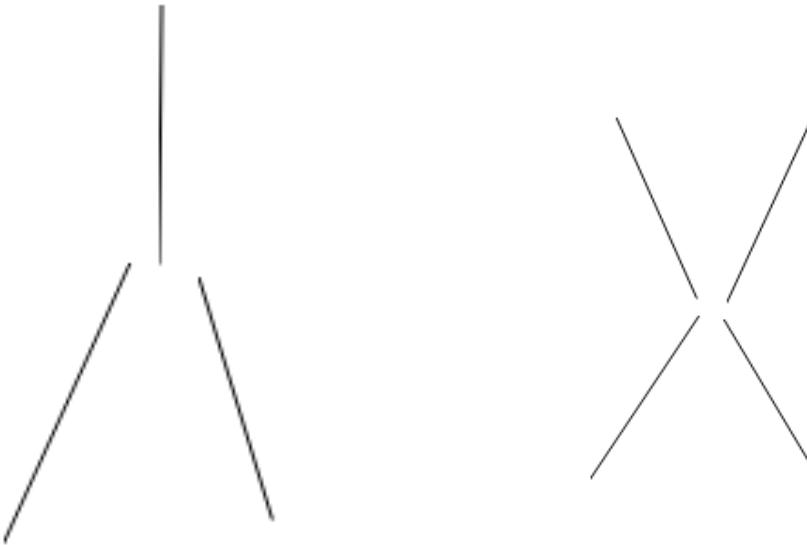


Рис. 12.

Рис. 11.

Алгоритм проверки параллельности прямых Рассмотрим функцию проверки параллельности прямых:

- 1) Вычисляем косинусы между прямыми и осью абсцисс.

$$Ox = (1, 0)$$

$$\cos1 = \cos(\text{line1}, Ox)$$

$$\cos2 = \cos(\text{line2}, Ox)$$

- 2) Вычисляем разность косинусов

$$\cos_diff = \text{abs}(\cos1, \cos2)$$

- 3) Вычисляем насколько близко должны быть прямые, чтобы считать их параллельными. Тут в очередной раз применяется линейная интерполяция. Так как чем ближе рельсы к камере, тем меньше угол должен быть между ними, чтобы посчитать их параллельными. Параметры `max_parallel_cos` и `min_parallel_cos` были получены экспериментально, рассматривая результаты работы алгоритма для крайних случаев реальных параллельных прямых на разных уровнях высоты.

$$\text{max_parallel_cos} = 0.4$$

$$\text{min_parallel_cos} = 0.05$$

$$\text{parallel_cos_eps} = \text{max_parallel_cos} - (\text{max_parallel_cos} - \text{min_parallel_cos}) / \text{image_height} * \text{current_y}$$

- 4) Если разность косинусов, вычисленная ранее, меньше parallel_cos_eps, прямые считаются параллельными.

```
return cos_diff < parallel_cos_eps
```

Алгоритм проверки пересечения вершин На вход поступают v1, v2 -вершины, для которых нужно определить пересекаются ли они. А также параметры: глубина проверки пересечений(intersection_depth) и глубина проверки соседей(check_neighs_depth).

Алгоритм:

- 1) Если глубина проверки пересечений достигла 0 - возвращаем True
- вершины пересекаются.

```
if intersection_depth == 0:  
    return True
```

- 2) Если прямые, соответствующие вершинам НЕ параллельны - выбираем у каждой прямой параллельного ребенка. Если параллельные дети существуют, рекурсивно вызываем функцию is_intersection с уменьшенной на единицу глубиной поиска пересечений.

```
if not is_parallel(v1, v2):  
    parallel_1 = v1.getParallelNeigh()  
    parallel_2 = v2.getParallelNeigh()  
    if parallel_1 and parallel_2:  
        return is_intersection(parallel_1, parallel_2,  
                               intersection_depth - 1, check_neighs_depth)
```

- 3) Если глубина проверки соседей ещё не достигла нуля И у текущих вершин v1, v2 существуют параллельные им дети, то рекурсивно вызываем is_intersection для найденных параллельных детей с уменьшенной на единицу глубиной проверки соседей.

```
if check_neighs_depth > 0:  
    parallel_1 = v1.getParallelNeigh()  
    parallel_2 = v2.getParallelNeigh()  
    if parallel_1 and parallel_2:  
        return is_intersection(parallel_1, parallel_2,  
                               intersection_depth, check_neighs_depth - 1)
```

- 4) Если не было выполнено ни одно из первых трех условий - вершины не являются пересекающимися - вернем False.

Описанный выше алгоритм применяется как для проверки пересечения между детьми, так и между родителями. В случае проверки пересечения между родителями слово "дети" в алгоритме везде нужно заменить на "родители".

Рассмотрим алгоритм на примере (синими линиями изображены грани-
цы горизонтальных блоков):

В данном примере рассматривается прямая 1.

На вход функции `is_intersection` поступают соседние прямые 2 и 3, и па-
раметры `intersection_depth = 2`, `check_neighs_depth = 1`.

Если прямые 2 и 3 НЕ параллельны, то запускаем `is_intersection` для
прямых 4 и 5 и `intersection_depth = 1`, `check_neighs_depth = 1`.

Если прямые 4 и 5 НЕ параллельны, то запускаем `is_intersection` для
прямых 6 и 7 и `intersection_depth = 0`, `check_neighs_depth = 1`. Теперь
функция возвращает True и верхняя точка прямой 1 добавляется, как
стрелка в результат.

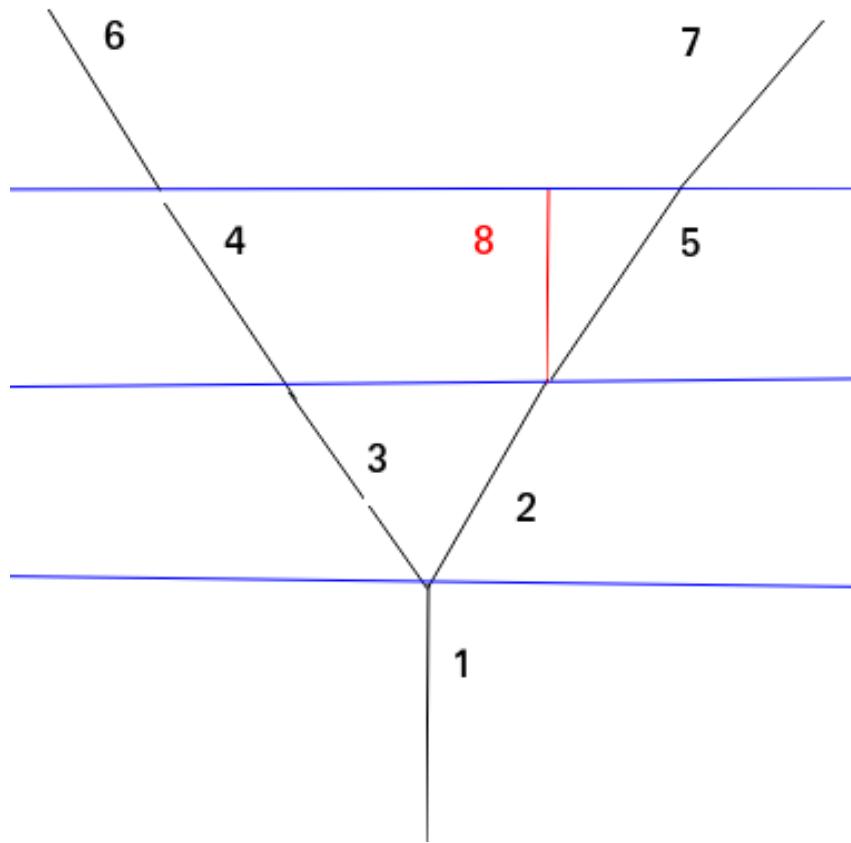


Рис. 13.

В случае пересечений между родителями пример будет зеркально от-
ражен по оси Y.

Параметр `check_neighs_depth` нужен в случае, когда сами соседи являются параллельными(прямые 2 и 3), но их продолжения(прямые 4 и 5) уже НЕ являются параллельными. Такая ситуация характерна для Y пересечений, которые находятся близко к камере.

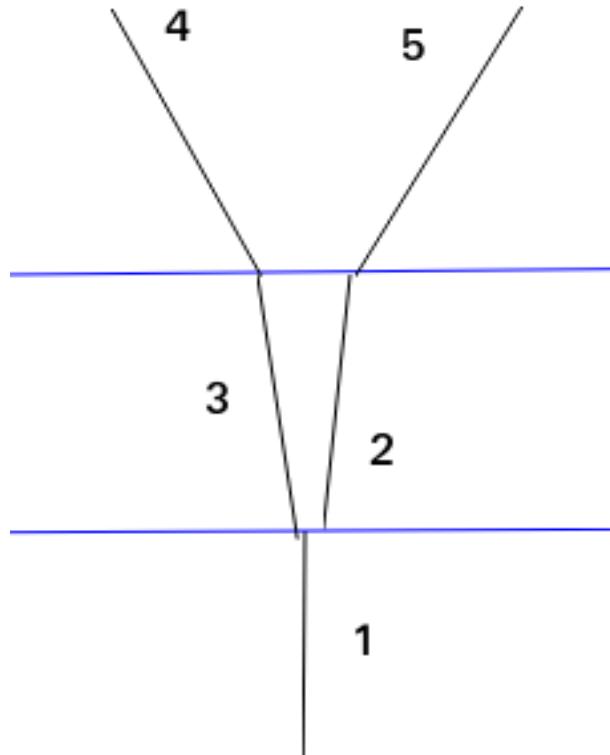


Рис. 14.

Аппроксимация близких вершин На последнем этапе близкие по расстоянию стрелки аппроксимируются их центрами масс. Эта процедура повторяется рекурсивно, пока в результирующем массиве все стрелки не станут полностью отделимыми друг от друга.

2.1.7. Описание данных для тестирования

Для тестирования алгоритма было размечено 204 фотографии ж/д путей, сделанных с локомотива. Часть фотографий была сделана при дневном свете 15, а часть при искусственном 17. Также имеются изображения, сделанные в летний 15 и в зимний период 18.

Параметры датасета

- 1) Всего стрелок размечено: 1308
- 2) Среднее количество стрелок на одном изображении: 6.3
- 3) Около 70% изображений сделаны в летний период. Около 30% в зимний.

2.1.8. Результаты

Примеры работы алгоритма Рассмотрим результаты работы алгоритма на различных изображениях.

Результаты работы на изображении сделанном при дневном:



Рис. 15.

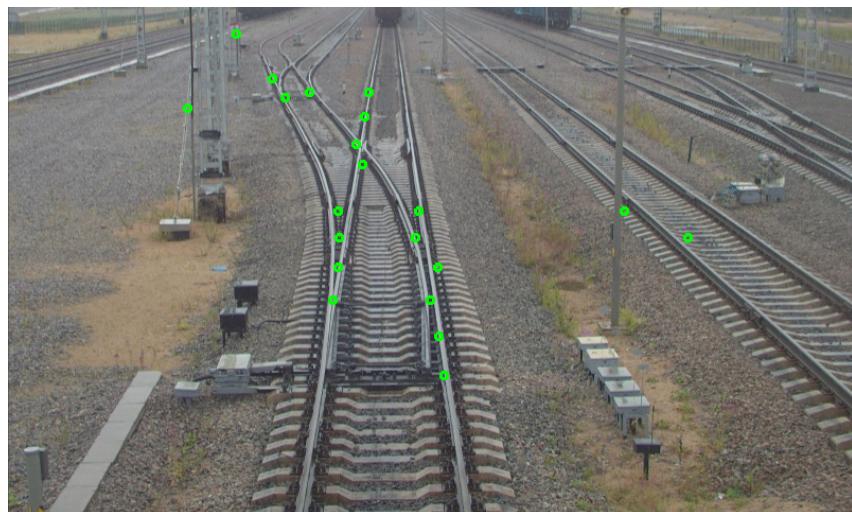


Рис. 16.

Истинное положение стрелок (маркеры) на рисунках 15, 16, 17, 18 - не обозначены, так как их положения очевидны

Видно, что есть большие стрелки - те, которые находятся достаточно близко к камере, алгоритм распознает очень хорошо. С маленькими стрелками есть небольшие проблемы, например на изображении 16 самая дальняя маленькая стрелка слева не распознавалась.

Также видно, что достаточно много false positive стрелок получившихся от столбов и заборов. Один из способов их подавления - использовать алгоритм Region growing up([5]) для сегментации изображения снизу вверх.

Теперь рассмотрим результат работы алгоритма при других погодных условиях и искусственном свете. Видно, что алгоритм хорошо справляется и с такими изображениями.

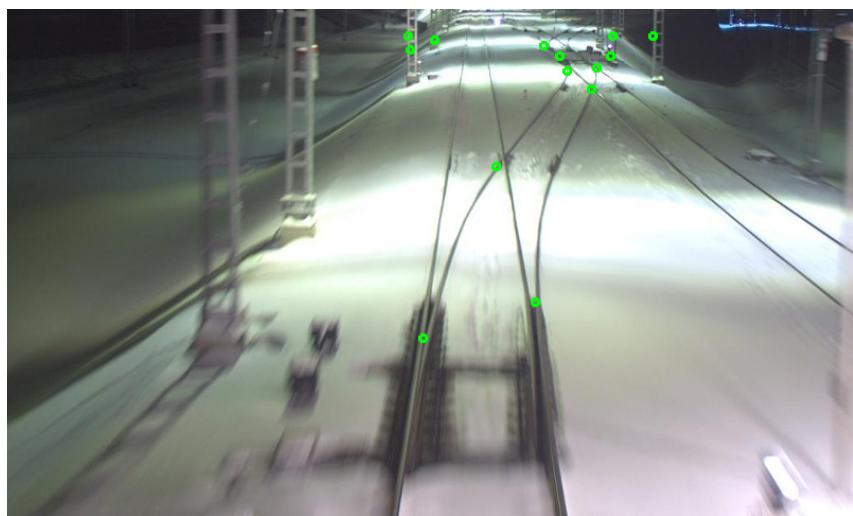


Рис. 17.

Теперь посмотрим на пример изображения, на котором алгоритм выдал очень плохой результат:

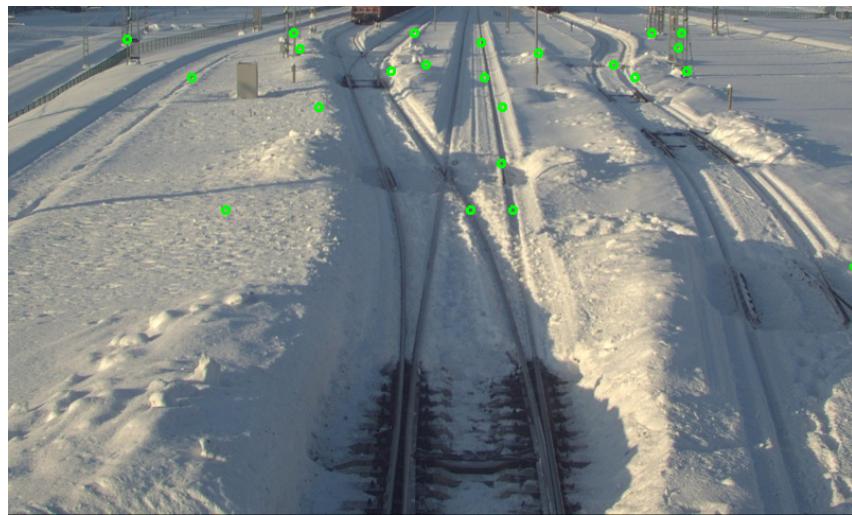


Рис. 18.

Видно, что на изображении есть ярко освещенные солнцем участки, и темная тень от поезда, в связи с этим алгоритм Отцу [3] выдал достаточно высокое значение пороговых фильтров и рельсы, находящиеся в тени не были восприняты как рёбра алгоритмом Canny[2]. В следствии чего соответствующие рельсам линии не были найдены алгоритмом Хафа [1].

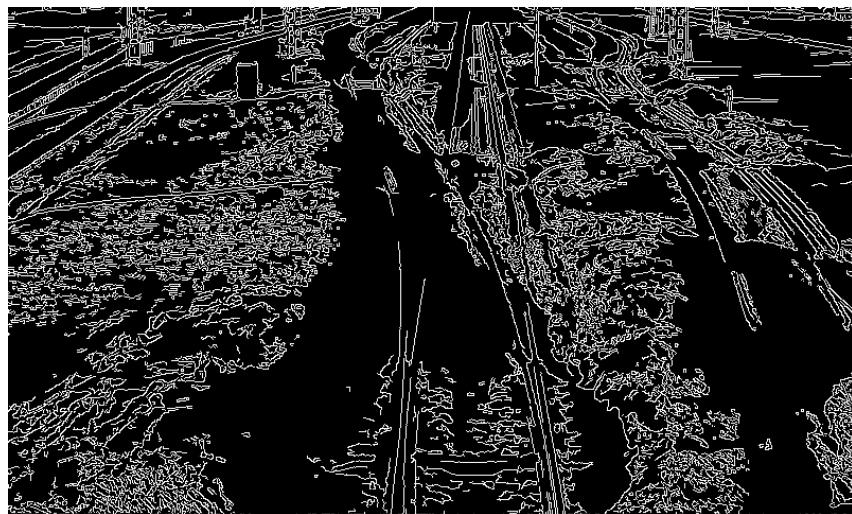


Рис. 19.

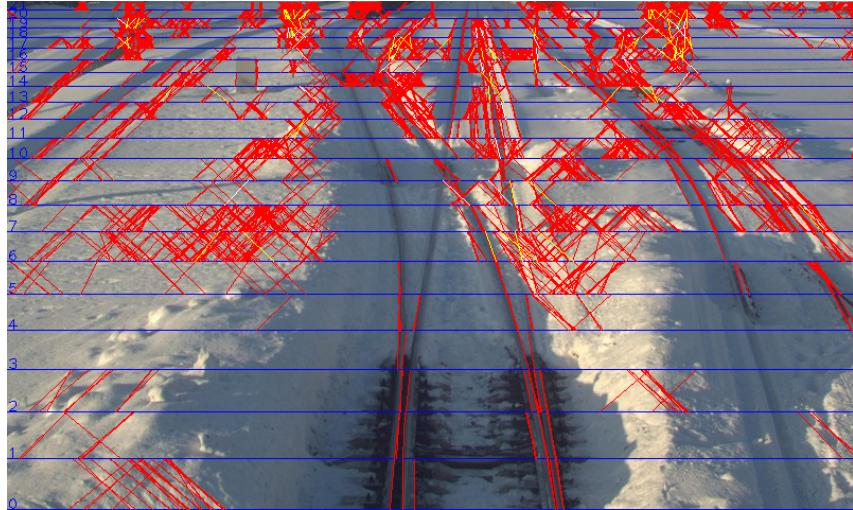


Рис. 20.

Статистические результаты Вспомним о том, что мы не знаем параметры камеры и поэтому ширину горизонтальных блоков выбираем с помощью линейной интерполяции 2.1.6. Интерполяция происходит между значениями `maxBlockSize` и `minBlockSize`. Рассмотрим зависимость результатов обнаружения от выбора этих параметров:

<code>minBlockEps</code>	<code>maxBlockEps</code>	<code>time(s)</code>	<code>Precision</code>	<code>Recall</code>
7	45	0.8	0.27	0.92
7	30	5.5	0.22	0.97
20	45	0.08	0.4	0.44

$Precision = \frac{TruePositive}{TruePositive + FalsePositive}$ - процент корректных предсказаний.

$Recall = \frac{TruePositive}{TruePositive + FalseNegative}$ - мера, определяющая как хорошо алгоритм находит позитивные примеры(стрелки)

Видно, что при уменьшении максимального размера блока растёт время работы алгоритма. Уменьшается процент корректных предсказаний. И увеличивается количество найденных положительных примеров.

Также и в обратную сторону, если увеличивать минимальный размер блока, то время работы уменьшается, НО количество найденных положительных примеров и процент корректных предсказаний уменьшаются.

При большой нижней границе minBlockEps далекие стрелки вообще не находятся, зато хорошо ищутся близкие стрелки, вот пример для $\text{minBlockEps} = 20\text{pix}$:



Рис. 21.

Список литературы

- [1] **Преобразование Хафа.** Википедия. [Электронный ресурс] URL - https://en.wikipedia.org/wiki/Hough_transform
- [2] **Алгоритм Canny.** [Электронный ресурс] URL - <https://habr.com/ru/post/114589/>
- [3] **Алгоритм Отцу.** [Электронный ресурс] URL - <https://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html>
- [4] **Перспективная проекция.** [Электронный ресурс] URL - <http://stratum.ac.ru/education/textbooks/kgrafic/lection04.html>
- [5] **Efficient railway tracks detection and turnouts recognition method using HOG features.** [Электронный ресурс] URL - <https://link.springer.com/article/10.1007/s00521-012-0846-0>