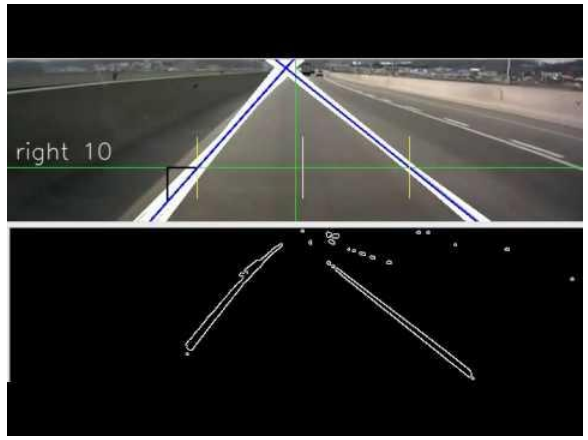
Использование ROS, Gazebo и OpenCV для распознавания дорожной разметки

Марков Алексей, ВолгГТУ

Введение


83.72,82.91
204.65,202.17
1
0.922


0
100
200
300
400
500
0
200
400
600
800


right 10

Идея решения

1. Бинаризуем изображение на основе цвета, чтобы выделить линии
2. Применим преобразования Хаффа, чтобы найти линии

И всё это сделаем не в Jupyter Notebook’е со статической картинкой из датасета, а с виртуальной камерой в Gazebo :)

Шаблон проекта

Заготовка проекта уже сделана:

roboschool2018/car_hackathon/scripts/line_mover.py

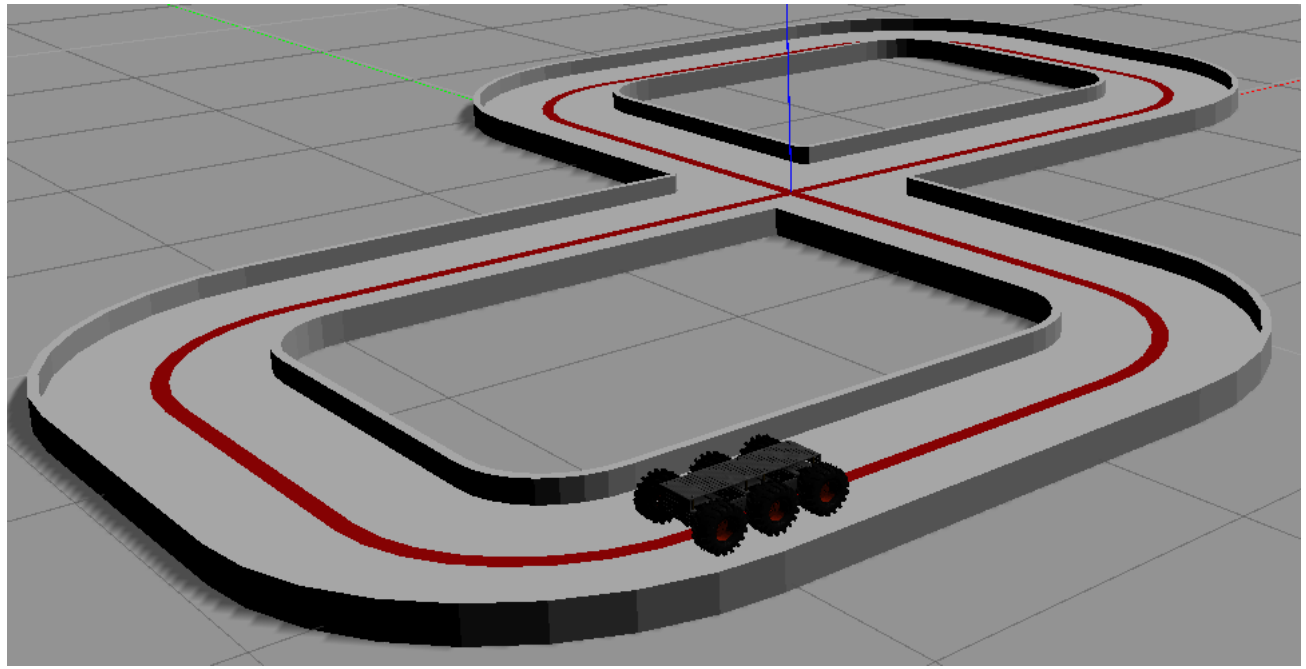
Запуск:

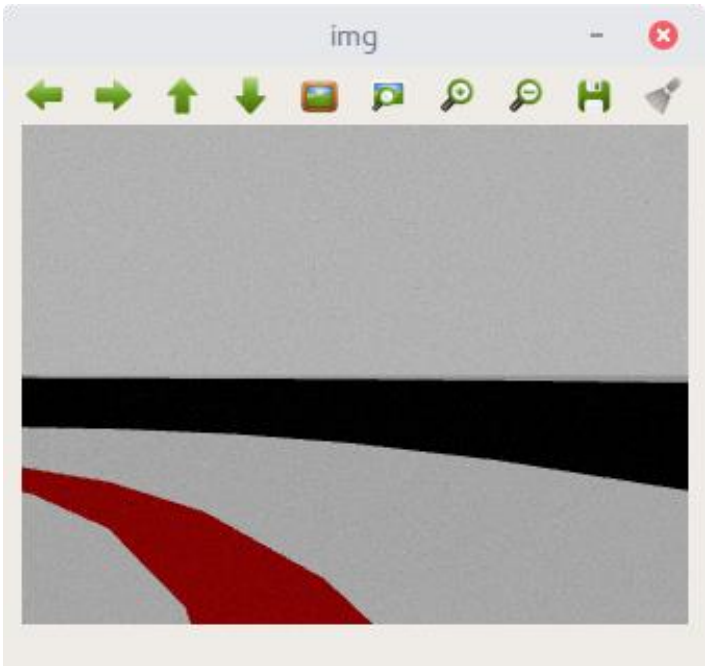
```
user@ros: ../car_hackathon/scripts$ roslaunch car_gazebo track.launch
```

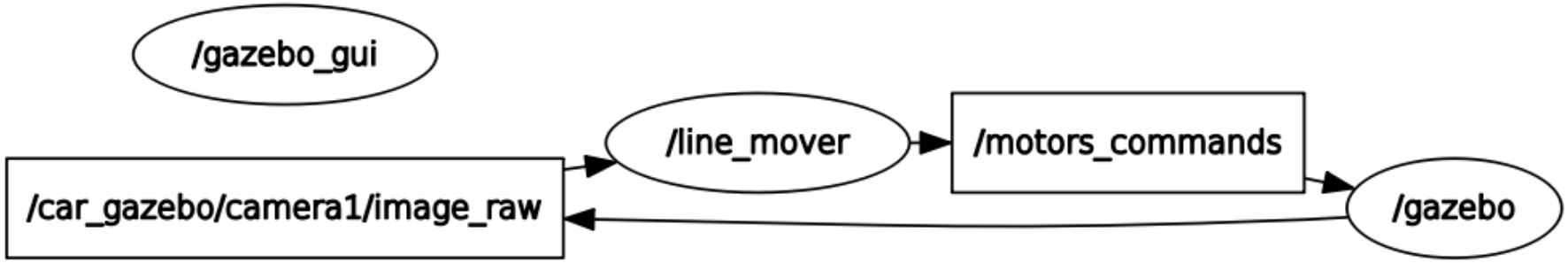
```
user@ros: ../car_hackathon/scripts$ chmod +x line_mover.py
user@ros: ../car_hackathon/scripts$ rosrun car_hackathon line_mover.py
```

```
user@ros: ../car_hackathon/scripts$ rqt_graph
```

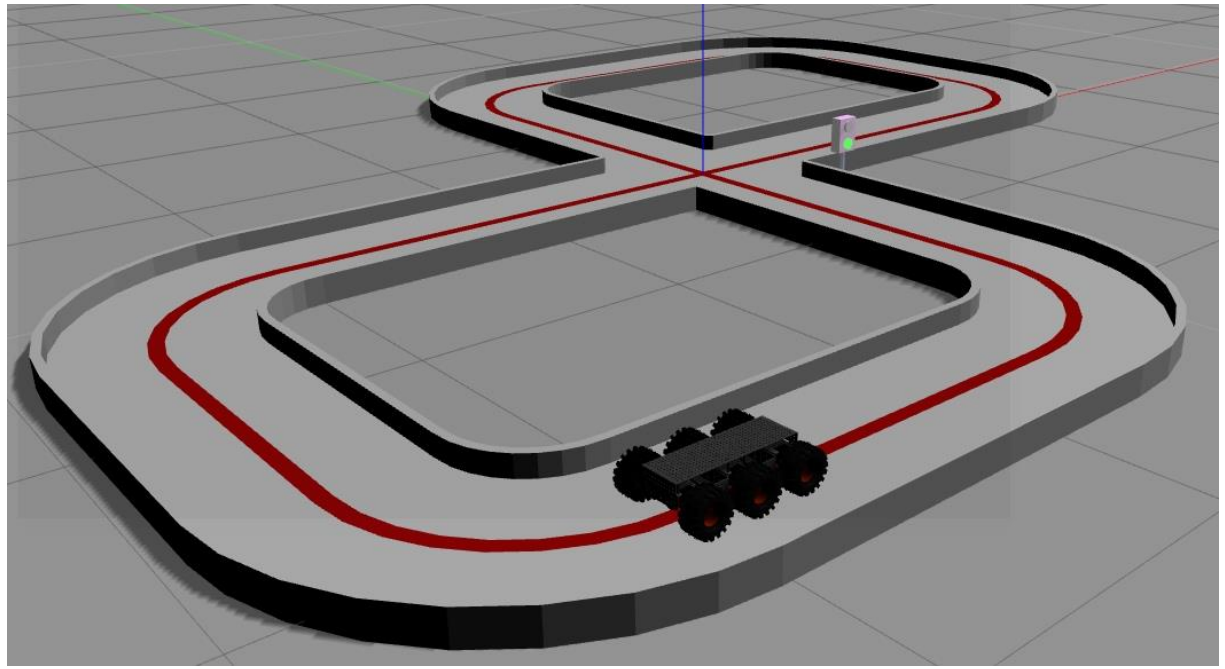
Шаблон проекта







Шаблон проекта



Подсказка: используйте **car_gazebo controller**, чтобы выключить моторы и управлять светофором

```
user@ros: ~ rosrun car_gazebo controller.py
Easy contoller for gazebo car simulation
Controls:
s - stop the car
r - turn on red signal
g - turn on green signal
```

Бинаризация

`cv2.cvtColor(src, code[, dst[, dstCn]]) → dst`

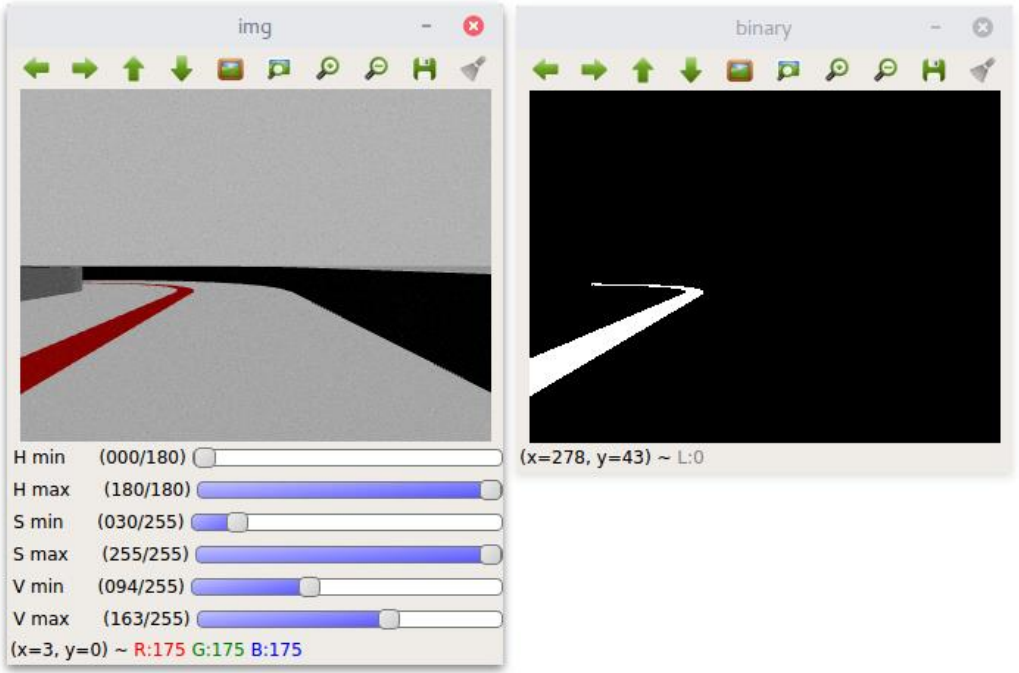
- src – входное изображение
- dst - выходное изображение (тот же размер и кол-во каналов)
- code – целевое цветовое пространство
- dstCn – кол-во каналов в целевом изображении

`cv2.inRange(src, lowerb, upperb[, dst]) → dst`

- src – входное изображение
- lowerb – нижняя граница, массив или скаляр (включается)
- upperb – верхняя граница, массив или скаляр (включается)
- dst – выходное изображение (тот же размер, тип CV_8U)

Бинаризация

```
image = bridge.imgmsg_to_cv2(msg, 'bgr8')
create_trackbars(image)*

h_min = cv2.getTrackbarPos('H min', 'img')
h_max = cv2.getTrackbarPos('H max', 'img')
s_min = cv2.getTrackbarPos('S min', 'img')
s_max = cv2.getTrackbarPos('S max', 'img')
v_min = cv2.getTrackbarPos('V min', 'img')
v_max = cv2.getTrackbarPos('V max', 'img')

hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
binary = cv2.inRange(hsv, (h_min, s_min, v_min),
(h_max, s_max, v_max))

cv2.imshow('img', image)
cv2.imshow('binary', binary)
```



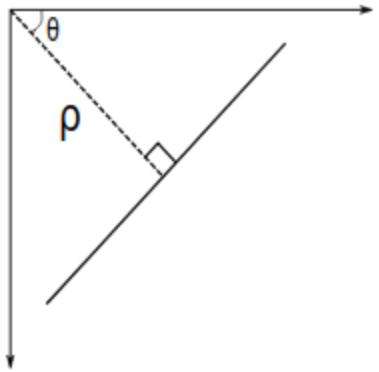
* cv2.namedWindow() вешает коллбеки ROS, поэтому трекбар надо создавать после того, как первый раз был вызван cv2.imshow()

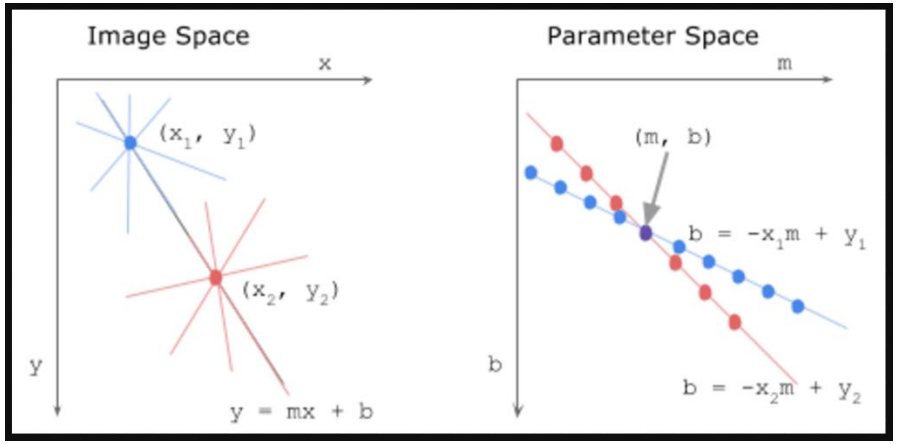
Преобразования Хаффа

Уравнение прямой:

$y = kx + b$

$\rho = x\cos\theta + y\sin\theta$





Learn more:

- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html
- https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html
- https://youtu.be/ebfi7qOFLuo

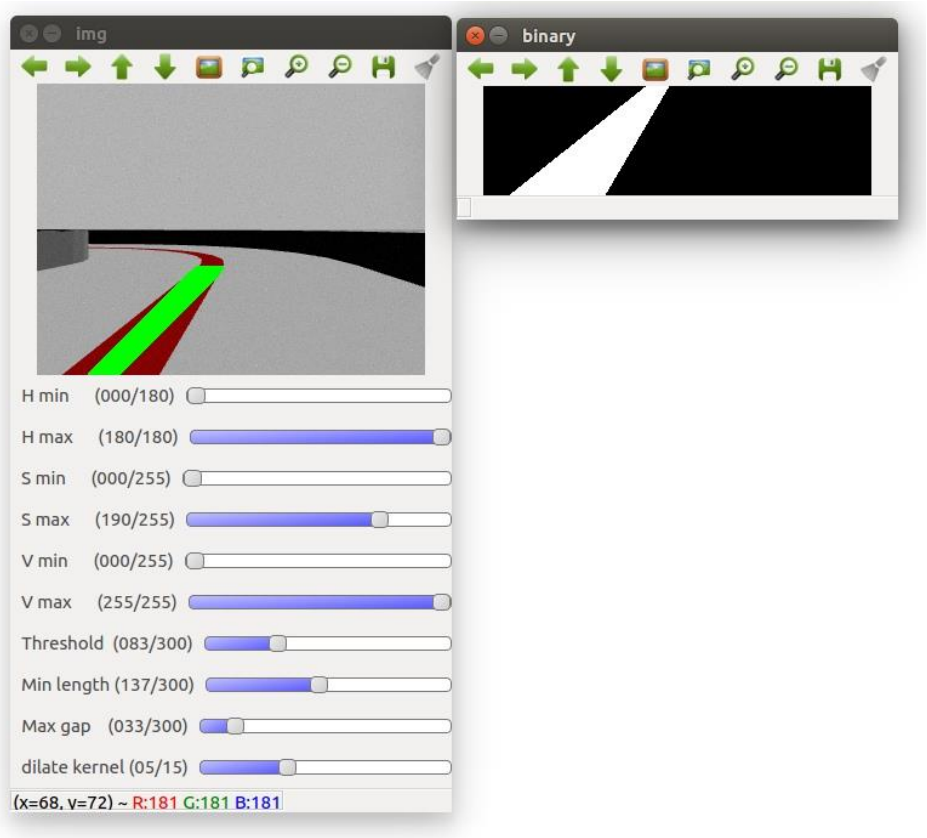
Преобразования Хаффа

`cv2.HoughLinesP(src, rho, theta, threshold, minLineLength, maxLineGap) → lines`

- rho – разрешение параметра ρ в пикселях
- theta – разрешение параметра θ в радианах
- threshold – минимальное кол-во пересечения для детектирования линии
- minLineLength – минимальное число точек, которые могут формировать линию (если точек меньше – линия отбрасывается)
- maxLineGap – максимальный зазор между двумя точками, чтобы они считались в одной линии

Преобразования Хаффа

```
lines = cv2.HoughLinesP(binary, 1, np.pi/180.0, min_length, max_gap)
if lines is not None:
    for line in lines:
        for x1, y1, x2, y2 in line:
            cv2.line(image, (x1, y1), (x2, y2), (0, 255, 0), 2)
```



Детектор границ Канни

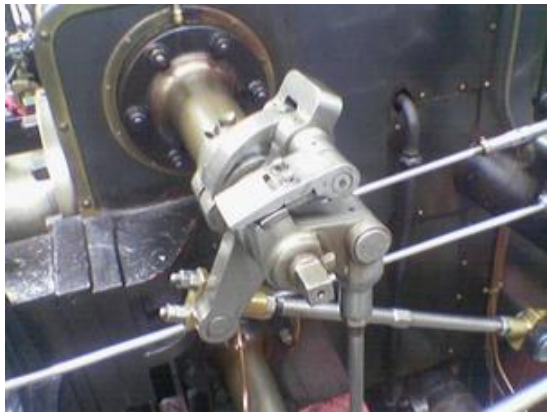
1. Устранение шумов (размытие)
2. Поиск градиента

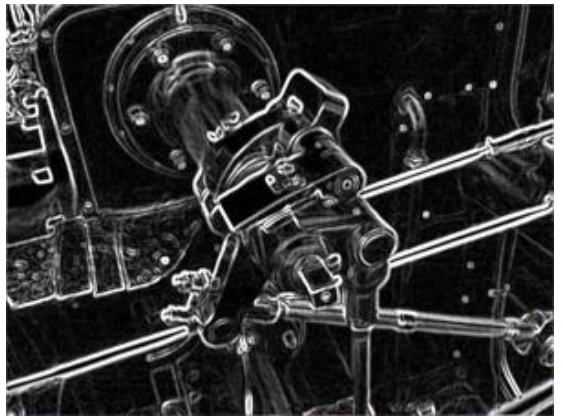
   Горизонтальные и вертикальные производные с помощью оператора Соболя

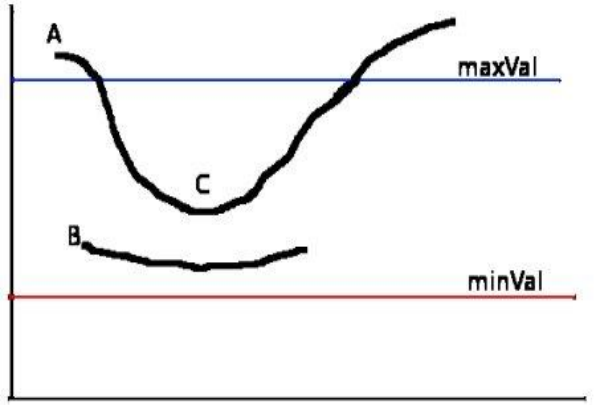
   $$G = \sqrt{G_x^2 + G_y^2}$$

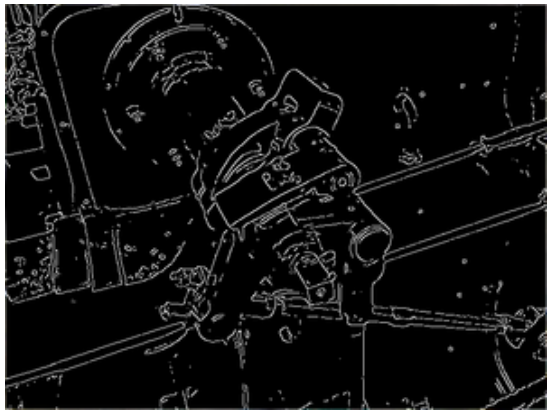
   $$\theta = \tan^{-1}\frac{G_y}{G_x}$$

3. Поиск локальных максимумов и направлений градиента в них
4. Порог с гистерезисом









Детектор границ Канни

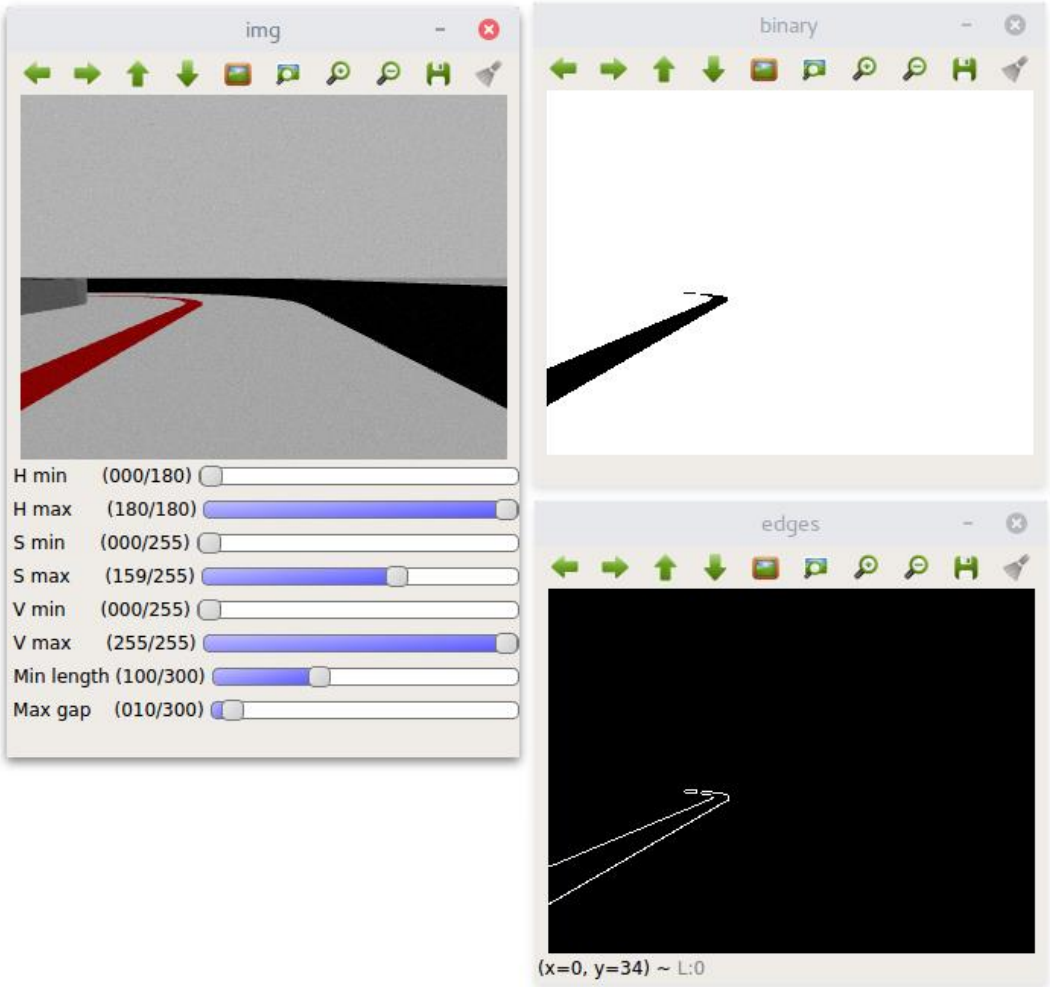
cv2.Canny(image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]]) → edges

- image – одноканальное 8-бит изображение
- edges – выходное изображение с границами
- threshold1 – первый порог
- threshold2 – второй порог
- apertureSize – размер ядра для оператора Собеля
- L2gradient – флаг включения более точной L2 нормы

Детектор границ Канни

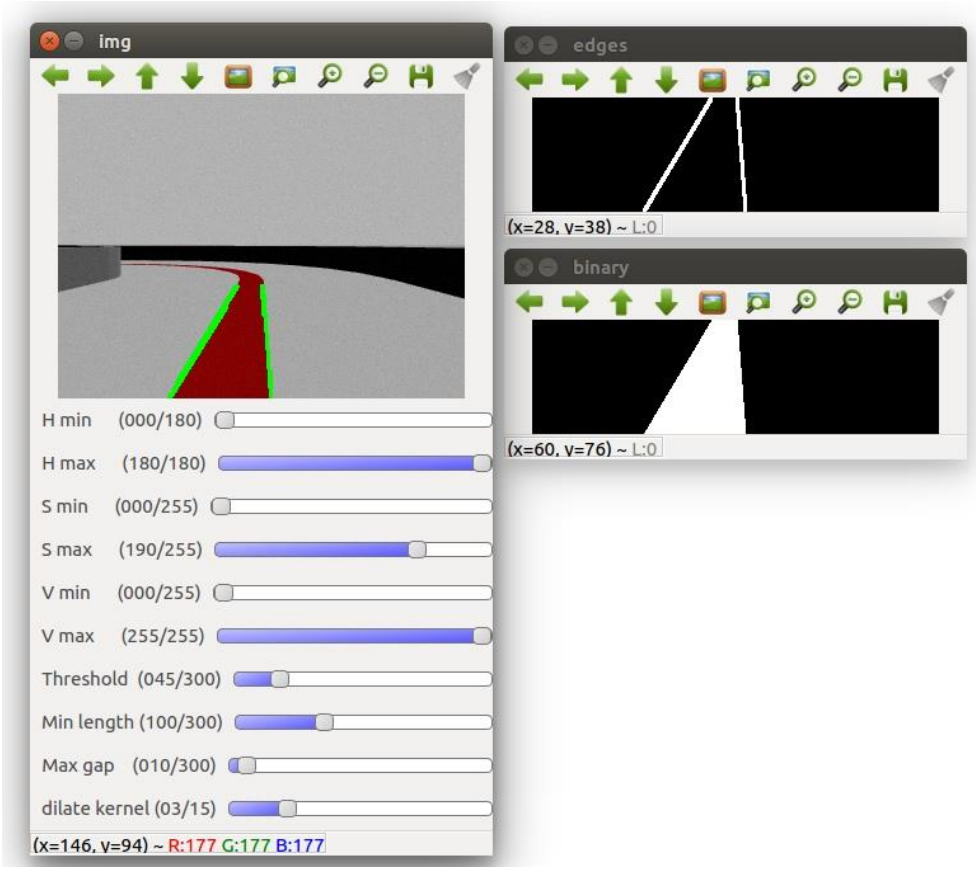
```
edges = cv2.Canny(binary, 100, 200)
cv2.imshow('edges', edges)
```

Для бинаризованного изображения параметры порога (почти?) не влияют. Теоретически, можно просто оператор Собеля применить, там либо градиент 0, либо сразу максимально возможный...



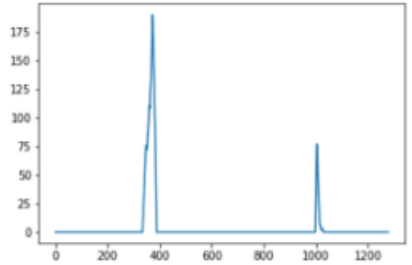
Canny + dilate + Hough transform

```
# Обрезка (только нижняя часть изображения)
image_cropped = image[150:]

# Преобразование в HSV и пороговая бинаризация
hsv = cv2.cvtColor(image_cropped, cv2.COLOR_BGR2HSV)
binary = cv2.inRange(hsv, (h_min, s_min, v_min),
    (h_max, s_max, v_max))
binary = 255 - binary

# Нахождение границ
edges = cv2.Canny(binary, 100, 200)

# Делаем границы толще (иначе Хафф плохо работает)
kernel = np.ones((kernel_size, kernel_size),np.uint8)
dilation = cv2.dilate(edges, kernel, iterations = 1)

# Находим линии с помощью преобразований Хаффа
lines = cv2.HoughLinesP(dilation, 0.5, np.pi/360.0,
    threshold, min_length, max_gap)

if lines is not None:
for line in lines:
x1, y1, x2, y2 = line[0]

cv2.line(image_cropped, (x1, y1), (x2, y2), (0, 255, 0), 2)
cv2.imshow('img', image)
cv2.imshow('binary', binary)
cv2.imshow('edges', dilation)
```



Другой вариант

1. Бинаризация
2. Перевод в bird-view проекцию
3. С помощью гистограммы находим начала линий
4. Отслеживаем линию вдаль с помощью сколзящего окна
5. Регрессия



И еще вариант…

Не обязательно находить линию, чтобы следовать за ней…