Using Computer Vision To Make Decisions

MRE/EME 5983 Robot Operating Systems

Overview

- Line following
- Advanced image processing
- Lane following
- Summary

Overview

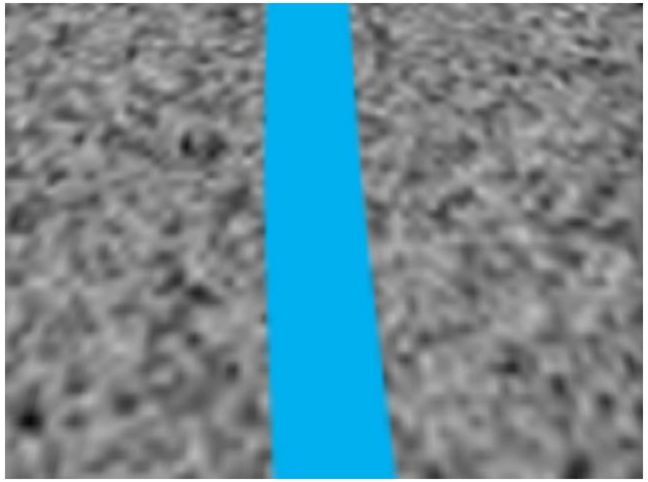
- Line following
- Advanced image processing
- Lane following
- Summary

Line Following

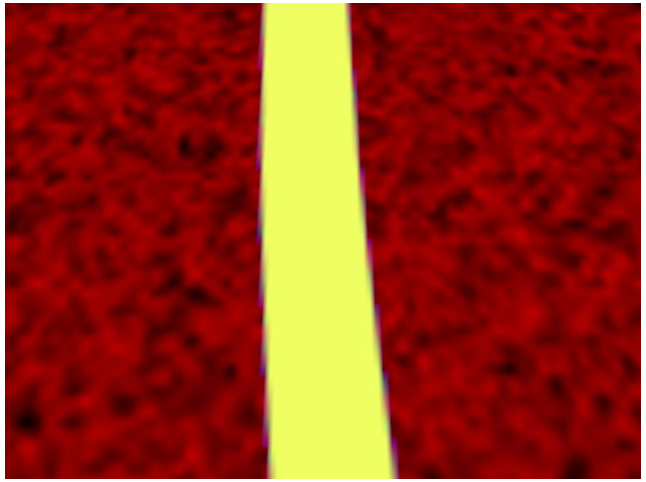
- Line following with a robot is a simple exercise for integrating sensing and controls in a practical application
- Sensing
  - We will use a camera sensor with image progressing in our application
- Controls (several options)
  - Zig-zag method
  - Binning method
  - P, PI and PID methods
- For simplicity, we will using a differential steer robot in this lecture

Line Following – Sensing

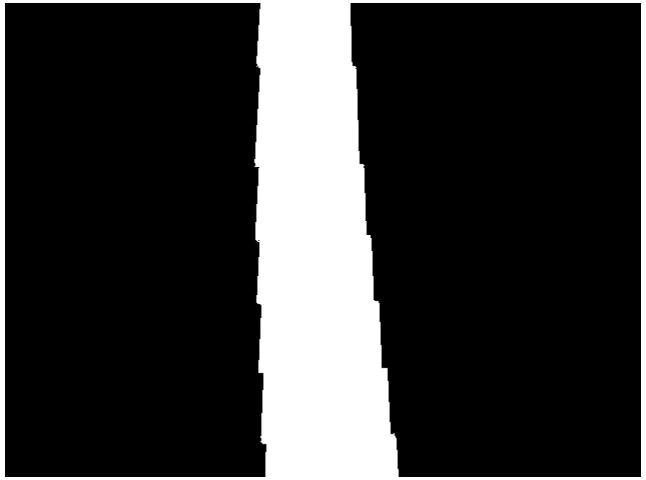
- We will develop our line following robots to process binary images (black and white)
- We can use OpenCV to process the image from the camera



Source



HSV

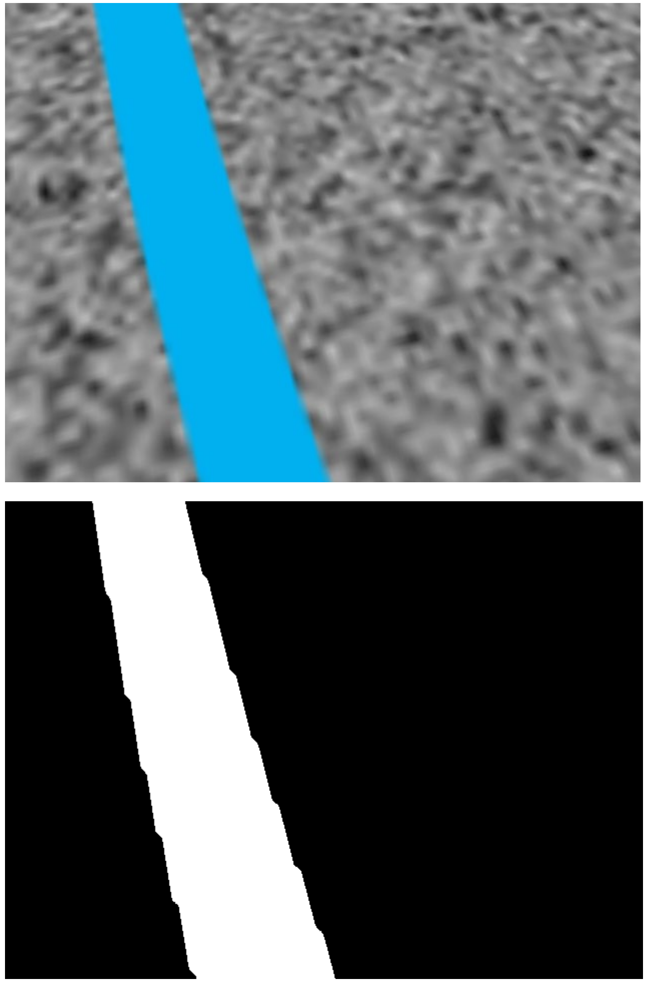


Black & White

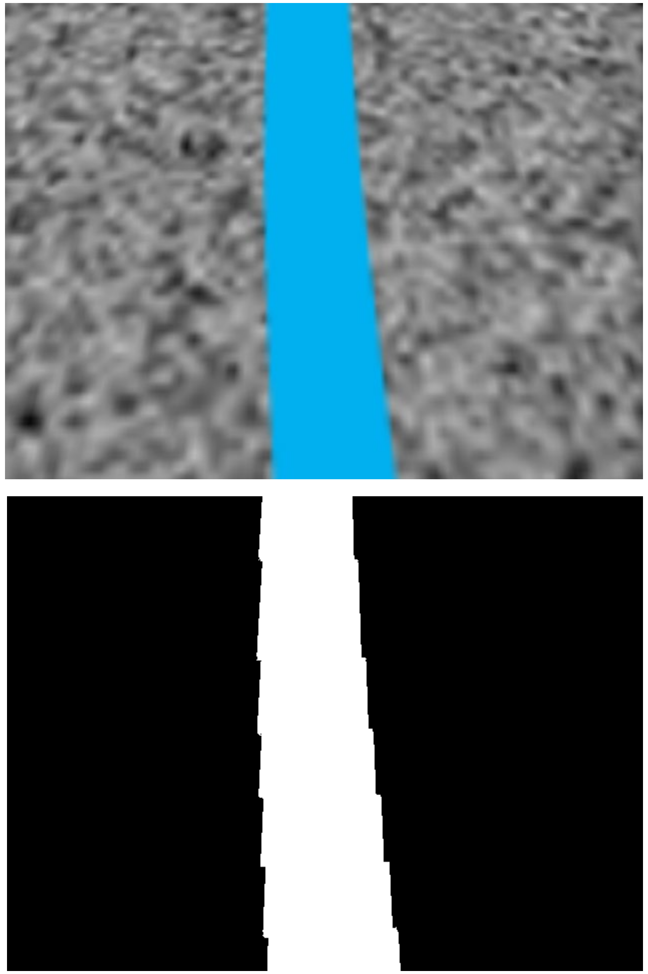
Line Following – Control

- Which way do we direct the robot steer? And, how much?

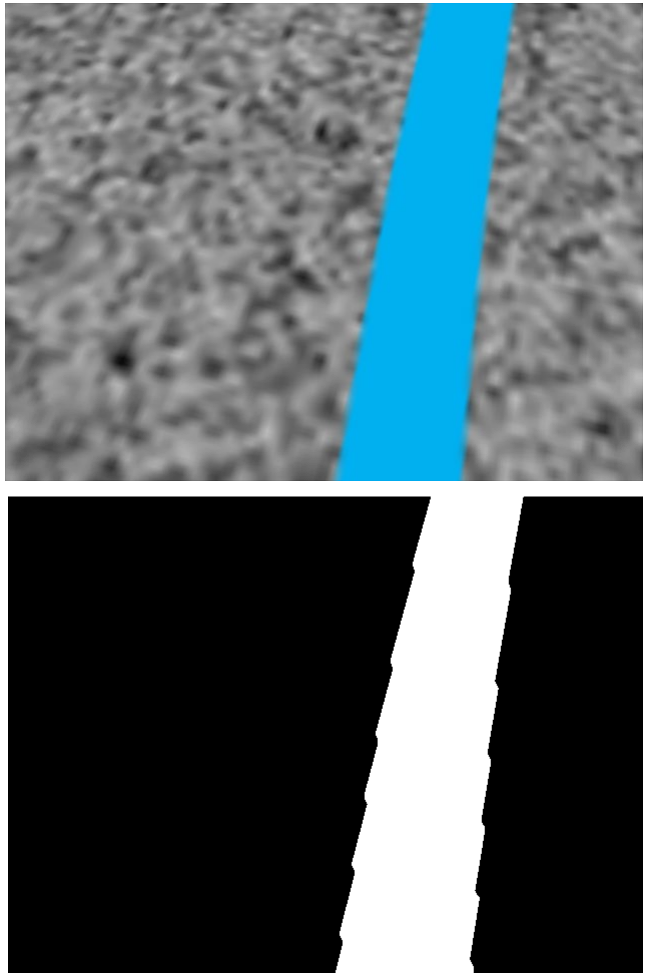
Yaw Left



Go Straight?

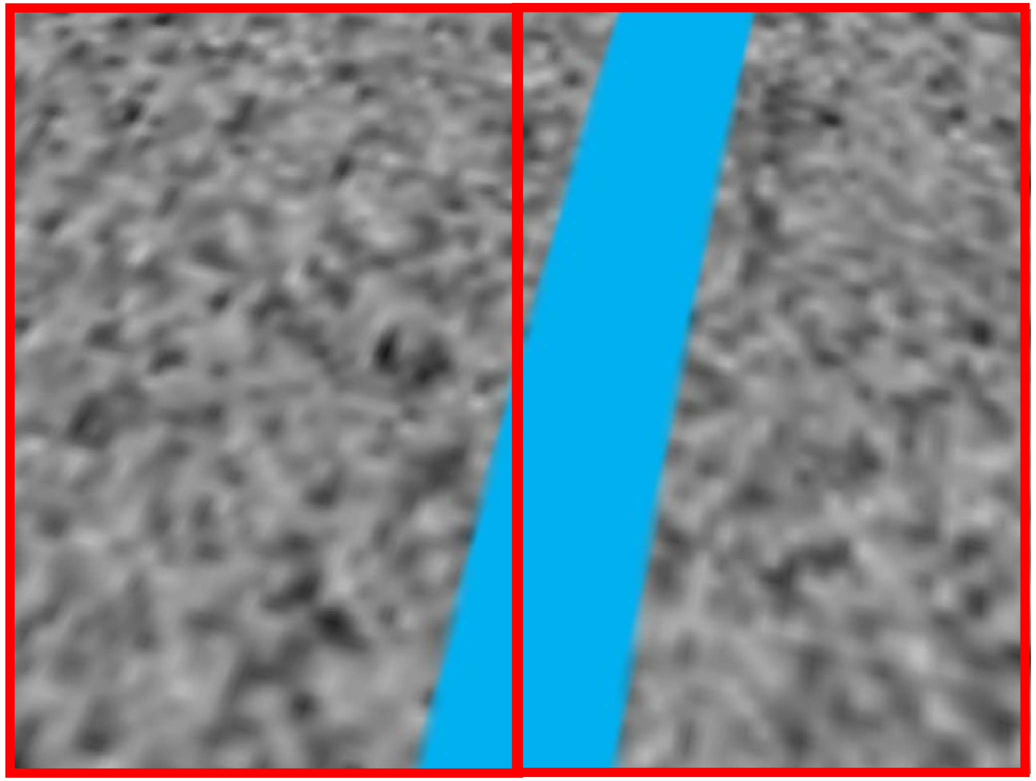


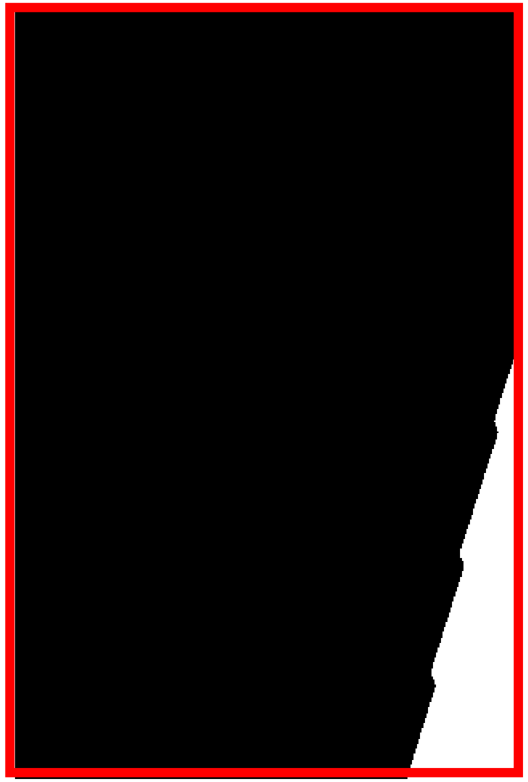
Yaw Right



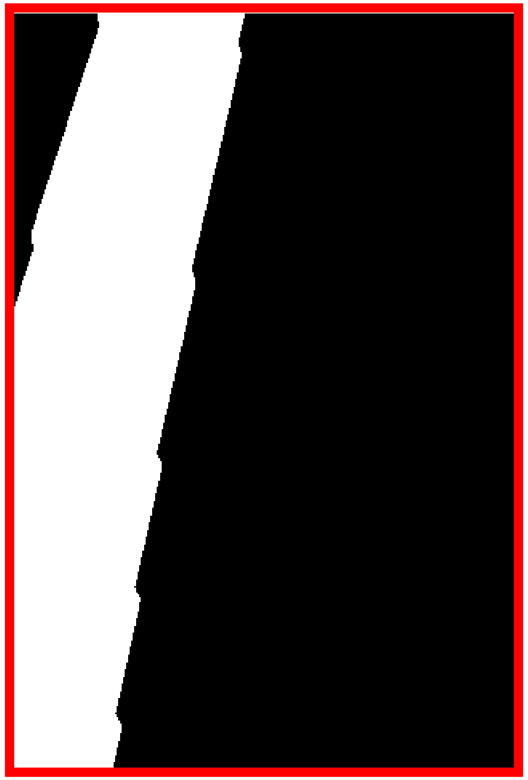
Zig-Zag

- Simplest, but least stable of the options
- Split the camera view in half vertically and steer towards side with more white pixels





NNZ = 10,603

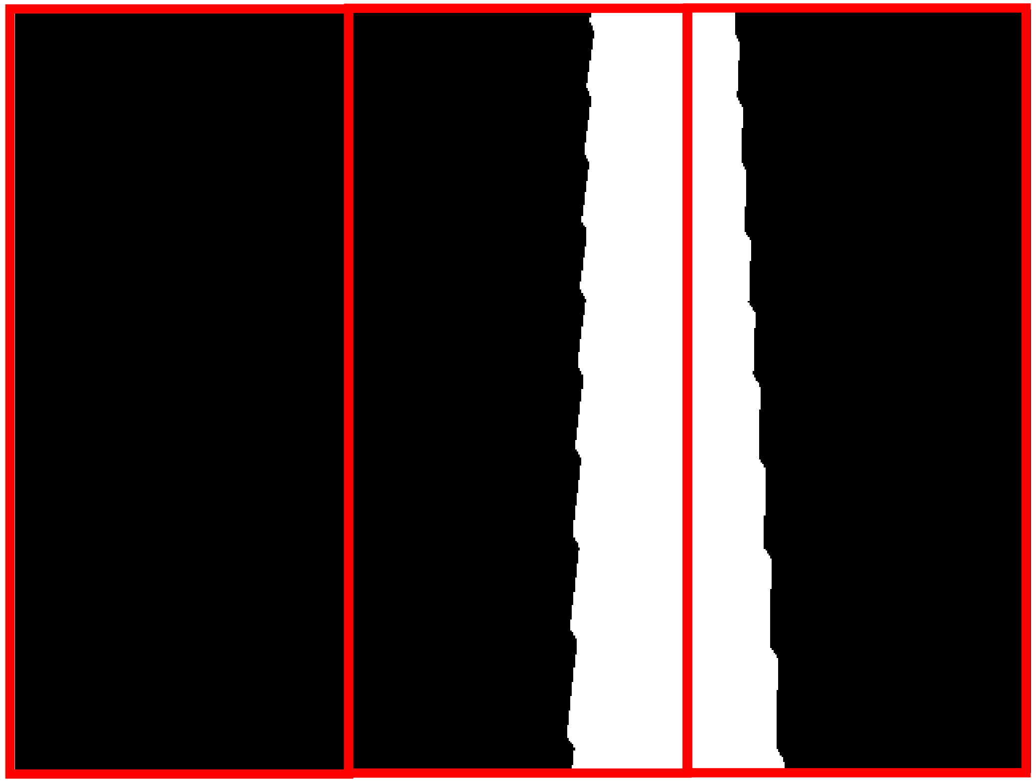


NNZ = 43,924

YAW RIGHT

Binning

- Similar to zig-zag, but provides "N" regions of analysis
- Typically chose N odd, for example N = 3



Left NNZ = 0
Center NNZ = 33,457
Right NNZ = 19,768

ZERO YAW RATE (Go Straight)

Proportional Control

- Proportional control uses the error in line following to compute the direction and magnitude of the yaw rate

- One method for determining the error is by computing the centroid of the line image
  - If the centroid is on the vertical centerline of the image, zero yaw rate is required
  - If the centroid is off the vertical centerline, request a yaw rate that is proportional to the amount the centroid is off the vertical center line

- OpenCV provides the cv::moment function to compute the weighted average of the image pixel intensities

$$\mathtt{m}_{ji} = \sum_{x,y} \left(\mathtt{array}(x, y) \cdot x^j \cdot y^i \right)$$

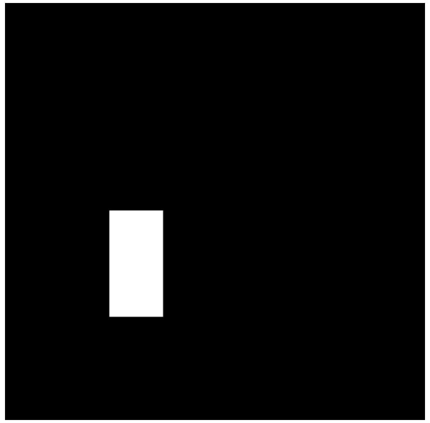
OpenCV Moments

opencv_moments.py

```python
#!/usr/bin/env python3

# Import OpenCV and NumPy
import cv2 as cv
import numpy as np

# Define a blank, grayscale image
img = np.zeros( shape=(400,400,1), dtype='uint8')
print('Image shape = ', img.shape)

# Draw a rectangle
pt1 = (100,200)
pt2 = (150,300)
color = 255
thickness = -1
cv.rectangle(img, pt1, pt2, color, thickness)

# Compute moments
M = cv.moments(img)
print('Rectangle pt1: ', pt1, 'to pt2 ', pt2)
print('m00  ', M['m00'])
print('m01  ', M['m01'])
print('m10  ', M['m10'])

print('m10/m00  ', M['m10']/M['m00'])
print('m01/m00  ', M['m01']/M['m00'])

cv.imshow('Test', img)
cv.waitKey(0)
```

Output

```
Image shape =  (400, 400, 1)
Rectangle pt1:  (100, 200) to pt2  (150, 300)
m00   1313505.0
m01   328376250.0
m10   164188125.0
m10/m00   125.0
m01/m00   250.0
```



- m00 = (51*101) * 255
- x_CG = m10/m00
- y_CG = m01/m00

$$\mathrm{m}_{ji} = \sum_{x,y} \left(\mathrm{array}(x,y) \cdot x^j \cdot y^i \right)$$

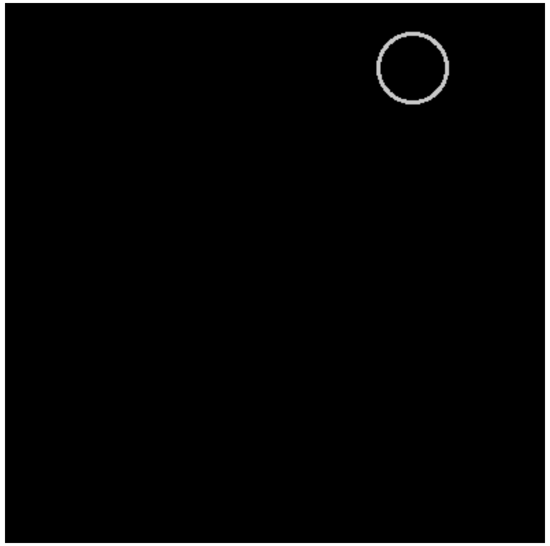
OpenCV Moments

opencv_moments.py

```python
# Draw a circle
img[:] = 0
pt_center = (300, 50)
radius = 25
color = 200
thickness = 2
cv.circle(img, pt_center, radius, color, thickness)

# Compute moments
M = cv.moments(img)
print(' ')
print(' ')
print('m10/m00   ', M['m10']/M['m00'])
print('m01/m00   ', M['m01']/M['m00'])

cv.imshow('Test', img)
cv.waitKey(0)
```

Output

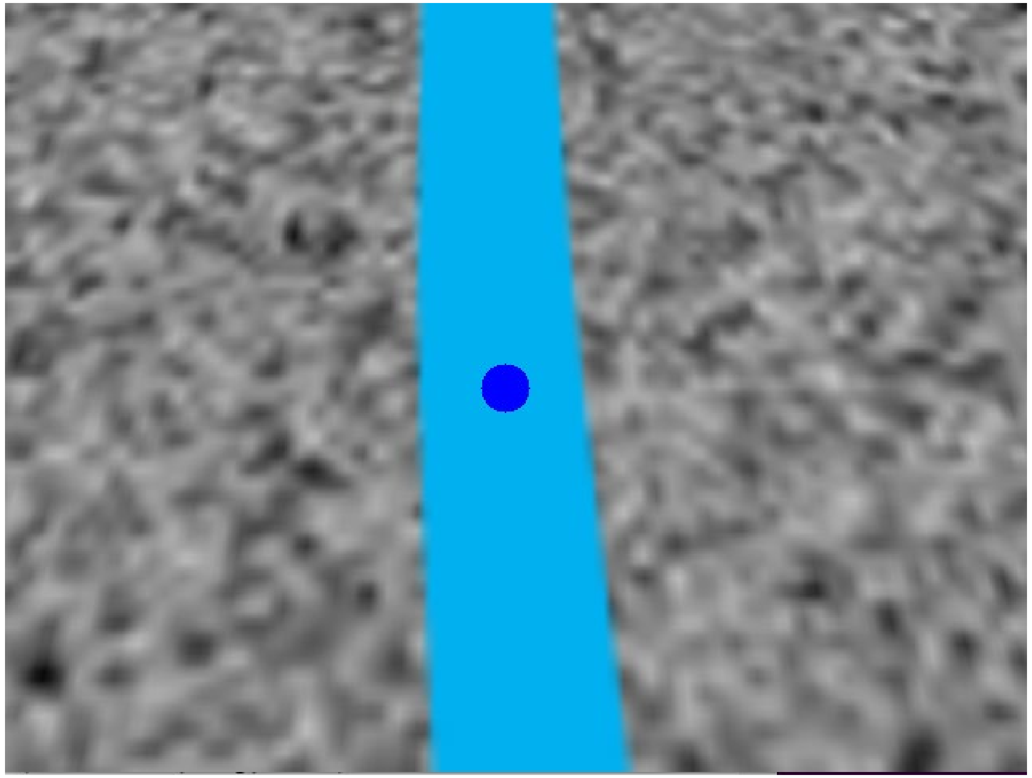
```
m10/m00    300.0655737704918
m01/m00    50.0655737704918
```



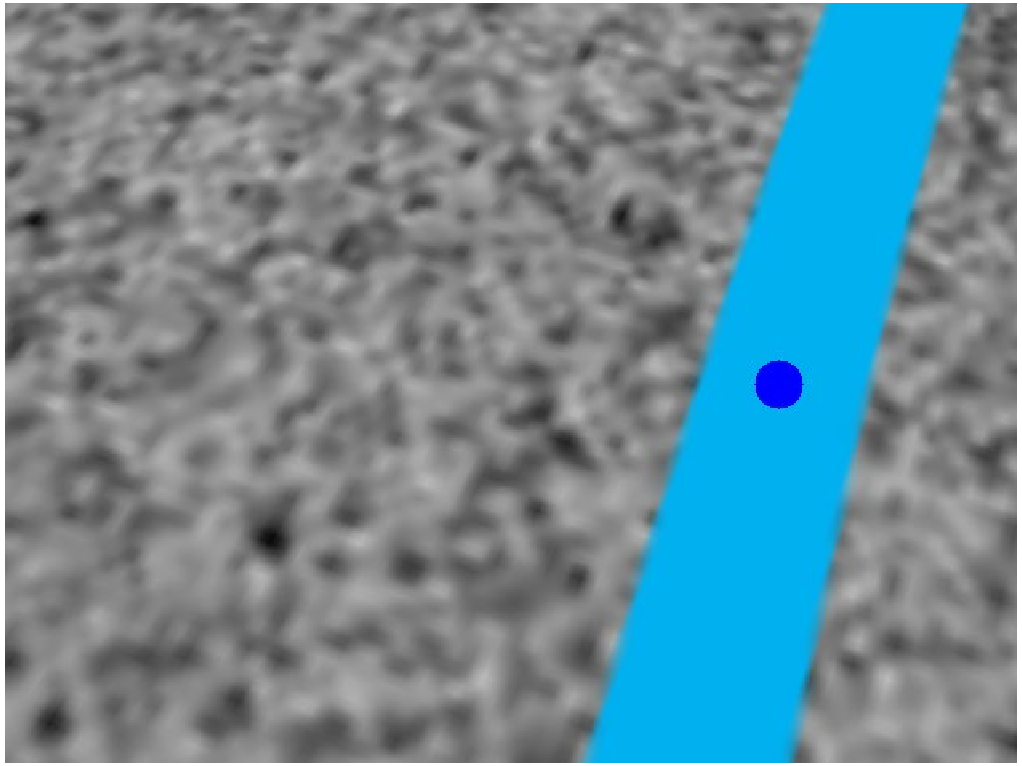
$$\mathrm{m}_{ji} = \sum_{x,y} \left(\mathrm{array}(x,y) \cdot x^j \cdot y^i \right)$$

Proportional Control

- We can the OpenCV moment calculations to find the error

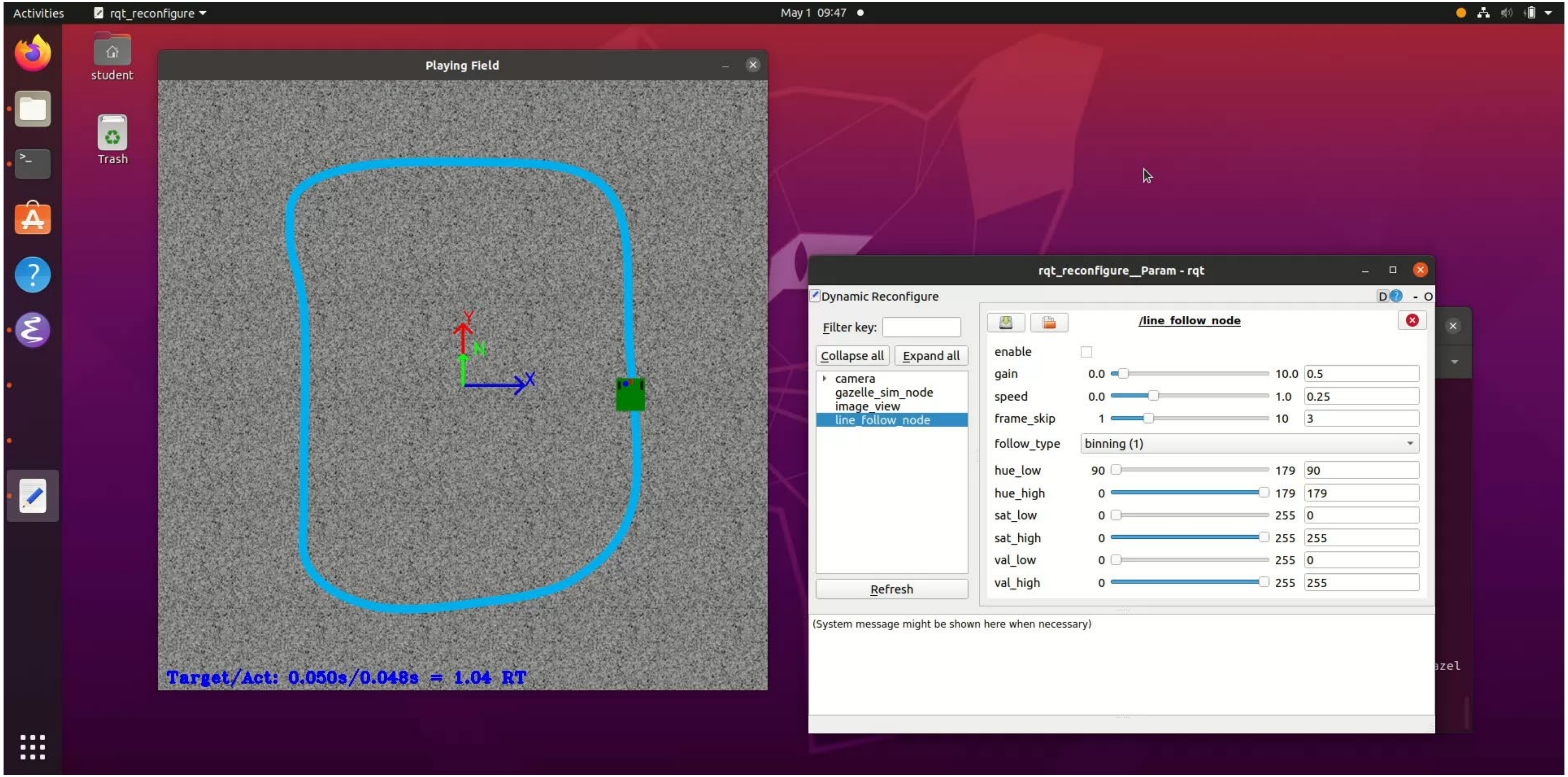


x_cg = 314
cols/2 = 320
Low error, low yaw (left) request



x_cg = 487
cols/2 = 320
High error, high yaw (right) request

Line Follow Example

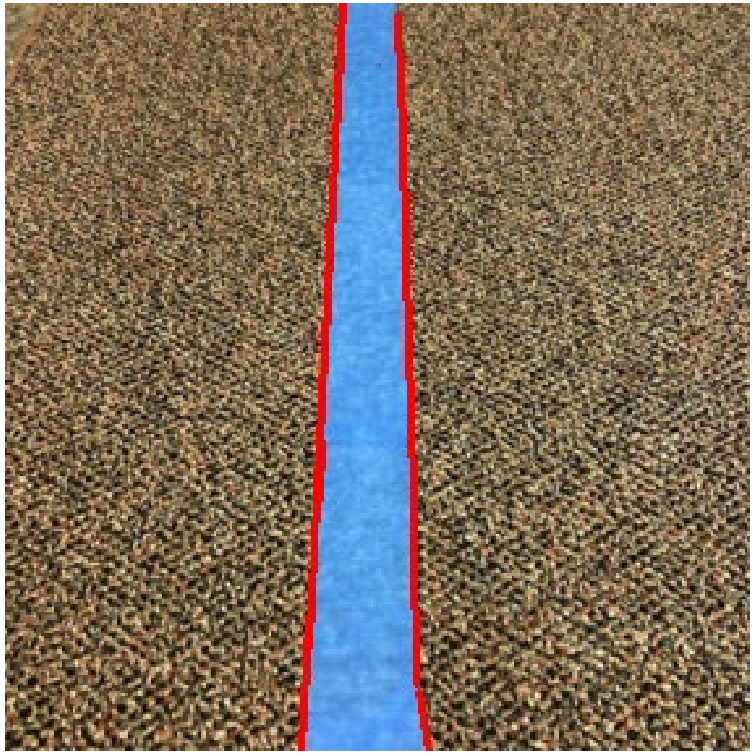


Overview

- Line following
- Advanced image processing
- Lane following
- Summary

Advanced Image Processing Concepts

- We have learned methods for using color spaces and thresholds to perform basic image processing
- Now we wish to use more advanced techniques
- Problems of interest
  - Edge detection
  - Shape detection
  - Object recognition
- Here, we will focus on edge detection
- Goal: Find the edges of the line



Edge Detection

- To detect edges of objects in an image using OpenCV, we will leverage two fundamental methods
  - Canny Edge Detection
  - Hough Transforms

- Canny edge detection operates on a single channel input image and returns an image consisting of edges

- Hough transforms can be used to post-process the Canny images to lines, circles or other simple forms from the edge data

Edge Detection – Example

- Example of the Canny Edge Detection Algorithm

Canny Edge Detection Algorithm

1. Noise reduction
   - Smooth with filter

2. Gradient calculation
   - Calculate the gradient of pixel intensities in the horizontal and vertical directions

3. Non-maximum suppression
   - Thin out the edges (goal is to have thin edges)

4. Double threshold
   - Find the strong, weak, and non-relevant pixel edges

5. Edge Tracking by Hysteresis
   - Transform weak pixels into strong ones or non-relevant

Source: https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

OpenCV Canny

◆ Canny() [1/2]

```
void cv::Canny ( InputArray   image,
                 OutputArray  edges,
                 double       threshold1,
                 double       threshold2,
                 int          apertureSize = 3 ,
                 bool         L2gradient = false
               )
```

Python:

```
cv.Canny( image, threshold1, threshold2[, edges[, apertureSize[, L2gradient]]] ) -> edges
cv.Canny( dx, dy, threshold1, threshold2[, edges[, L2gradient]]                ) -> edges
```

`#include <opencv2/imgproc.hpp>`

Finds edges in an image using the Canny algorithm **[40]** .

The function finds edges in the input image and marks them in the output map edges using the Canny algorithm. The smallest value between threshold1 and threshold2 is used for edge linking. The largest value is used to find initial segments of strong edges. See http://en.wikipedia.org/wiki/Canny_edge_detector

Parameters

image	8-bit input image.
edges	output edge map; single channels 8-bit image, which has the same size as image .
threshold1	first threshold for the hysteresis procedure.
threshold2	second threshold for the hysteresis procedure.
apertureSize	aperture size for the Sobel operator.
L2gradient	a flag, indicating whether a more accurate L_2 norm $=\sqrt{(dI/dx)^2+(dI/dy)^2}$ should be used to calculate the image gradient magnitude (L2gradient=true), or whether the default L_1 norm $=\lvert dI/dx\rvert+\lvert dI/dy\rvert$ is enough (L2gradient=false).

Examples:

samples/cpp/edge.cpp, **samples/cpp/lsd_lines.cpp**, **samples/cpp/squares.cpp**, **samples/cpp/tutorial_code/ImgTrans/houghlines.cpp**, and **samples/tapi/squares.cpp**.

Canny Edge Detection Example

opencv_canny.py

```
#!/usr/bin/env python3
import cv2 as cv
import sys

# Parameters
thres1 = 0
thres2 = 0
aperture = 0
aperture_list = [3,5,7]

# Canny trackbar callback
def callback(x):
    global thres1, thres2, aperture
    thres1 = cv.getTrackbarPos('thres1','controls')
    thres2 = cv.getTrackbarPos('thres2','controls')
    aperture = cv.getTrackbarPos('aperture_3_5_7','controls')
    return

# Create the a controls window
cv.namedWindow('controls',2)

# Create trackbars for canny thresholds
cv.createTrackbar('thres1','controls',    0, 255, callback)
cv.createTrackbar('thres2','controls',    0, 255, callback)
cv.createTrackbar('aperture_3_5_7','controls',  0, 2, callback)

# Read an image
img = cv.imread(sys.argv[1])
```

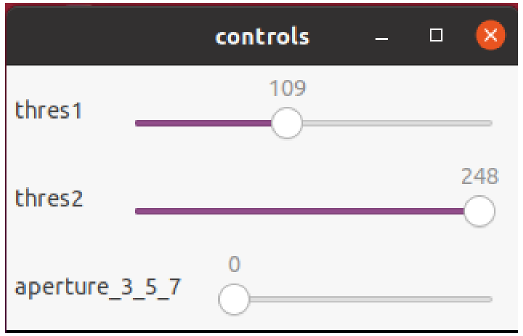
opencv_canny.py

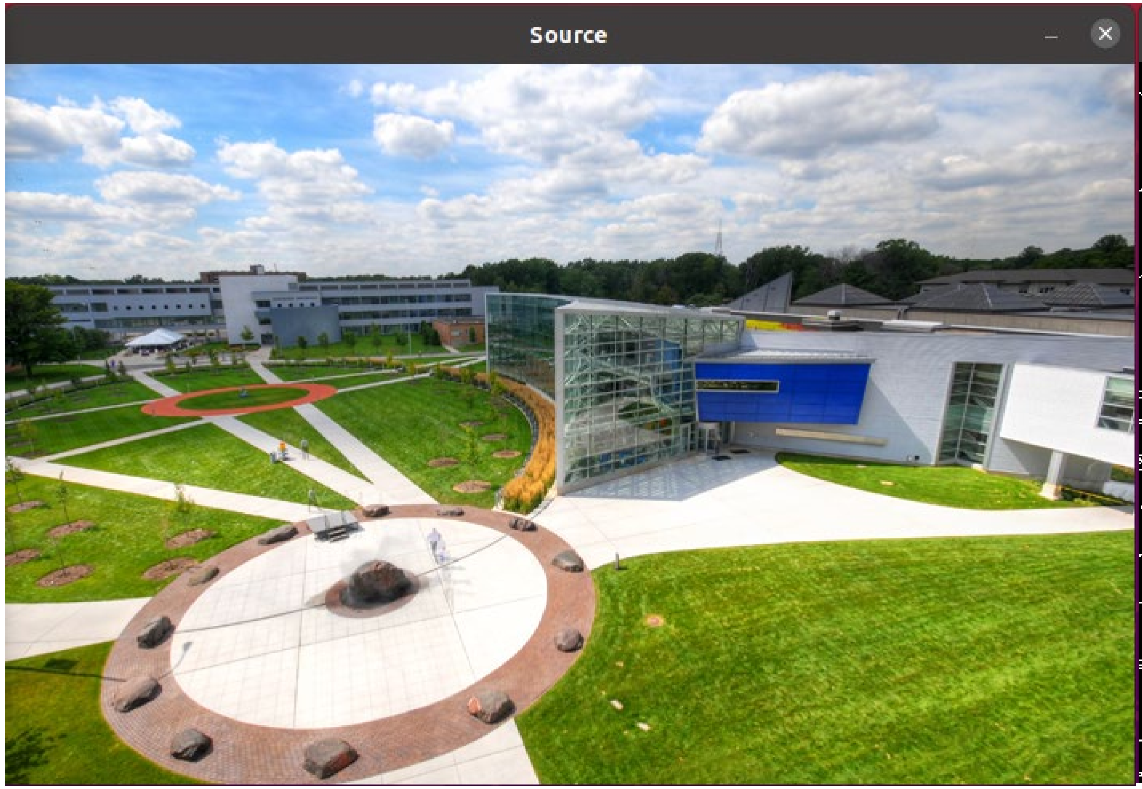
```
# Loop for edits
while(1):

    # Show the image
    cv.imshow('Source', img)

    # Convert to grayscale
    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

    # Apply Canny edge detector
    img_canny = cv.Canny(img_gray, thres1, thres2,
                         apertureSize=aperture_list[aperture])

    # Show the Canny image
    cv.imshow('Canny', img_canny)

    # Exit on key enter
    k = cv.waitKey(1) & 0xFF
    if k == 27:
        break

# Close all windows
cv.destroyAllWindows()
```

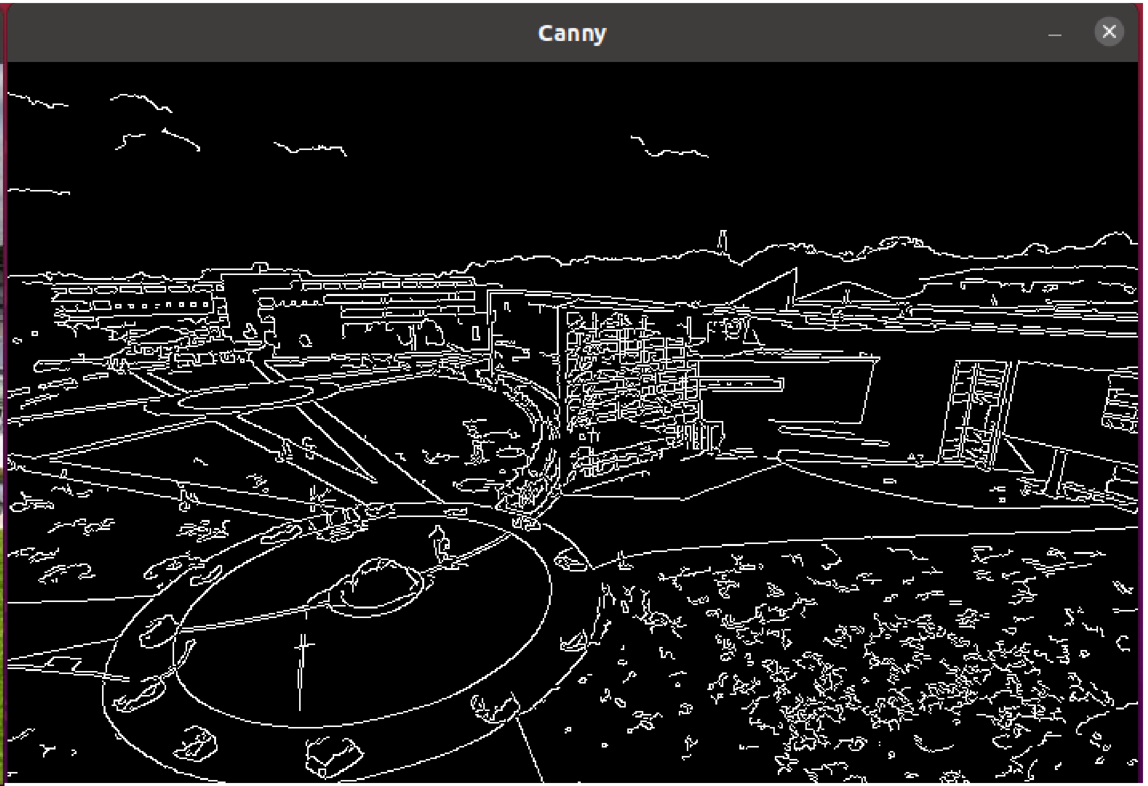
Canny Edge Detection Example

Execute from ~/catkin_ws/src/course_tutorials/scripts

$ python3 opencv_canny.py ../img/ltu_campus.jpg





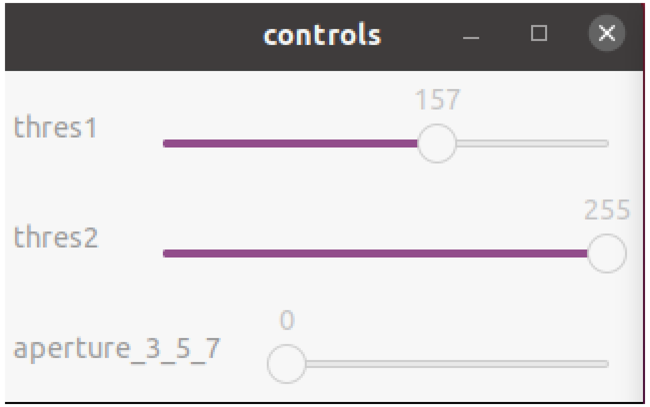


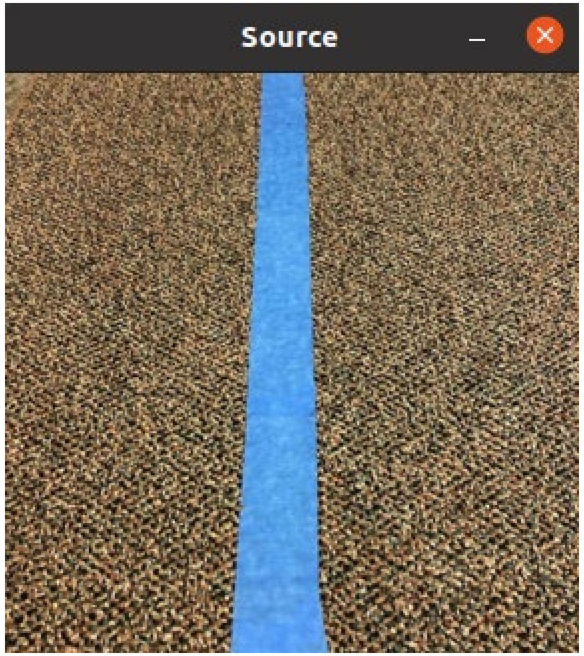
Canny Edge Detection Example

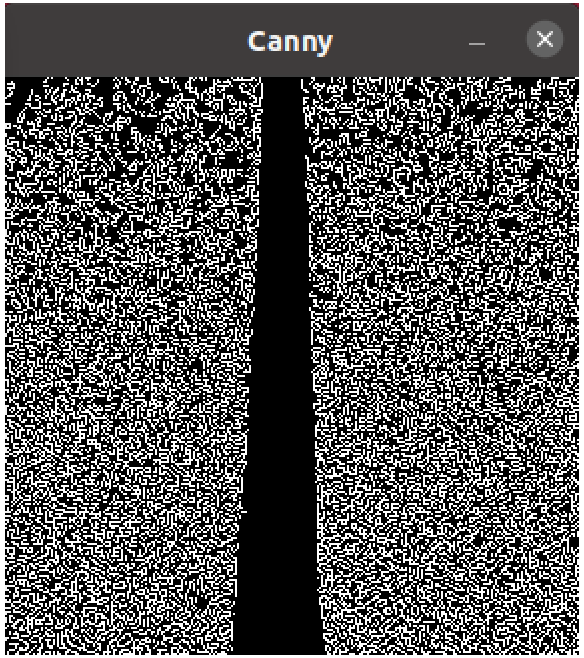
Let’s revisit our line following and color spaces discussion…

Execute from ~/catkin_ws/src/course_tutorials/scripts

$ python3 opencv_canny.py ../img/blue_line_floor_1.png







This is somewhat helpful, but not really helping us find the line edge…

Canny Edge Detection Example

opencv_hsv_canny.py

```python
import cv2 as cv
import numpy as np
import sys

# Color ranges
H_low = 0
H_high = 179
S_low= 0
S_high = 255
V_low= 0
V_high = 255
thres1 = 0
thres2 = 0
aperture = 0
aperture_list = [3,5,7]

# Trackbar callback fucntion to update HSV value
def callback(x):
    global H_low, H_high, S_low, S_high, V_low, V_high
    global thres1, thres2, aperture
    H_low = cv.getTrackbarPos('low H','controls')
    H_high = cv.getTrackbarPos('high H','controls')
    S_low = cv.getTrackbarPos('low S','controls')
    S_high = cv.getTrackbarPos('high S','controls')
    V_low = cv.getTrackbarPos('low V','controls')
    V_high = cv.getTrackbarPos('high V','controls')
    thres1 = cv.getTrackbarPos('thres1','controls')
    thres2 = cv.getTrackbarPos('thres2','controls')
    aperture = cv.getTrackbarPos('aperture_3_5_7','controls')

    return
```

opencv_hsv_canny.py

```python
# Create trackbars for Low and High B, G, R
cv.createTrackbar('low H','controls',    0, 179, callback)
cv.createTrackbar('high H','controls', 179, 179, callback)
cv.createTrackbar('low S','controls',    0, 255, callback)
cv.createTrackbar('high S','controls', 255, 255, callback)
cv.createTrackbar('low V','controls',    0, 255, callback)
cv.createTrackbar('high V','controls', 255, 255, callback)
cv.createTrackbar('thres1','controls',    0, 255, callback)
cv.createTrackbar('thres2','controls',    0, 255, callback)
cv.createTrackbar('aperture_3_5_7','controls',  0, 2, callback)

# Read the image
img_orig = cv.imread(sys.argv[1])
```

Canny Edge Detection Example

opencv_hsv_canny.py

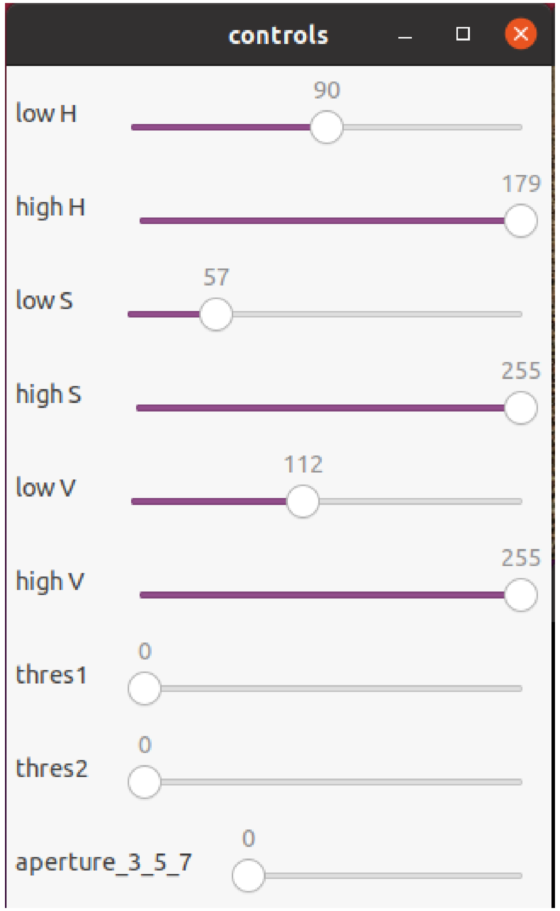
```
while(1):

    # Conver to HSV
    img_hsv = cv.cvtColor(img_orig, cv.COLOR_BGR2HSV)

    # Set up bounds
    hsv_low = np.array([H_low, S_low, V_low], np.uint8)
    hsv_high = np.array([H_high, S_high, V_high], np.uint8)

    # Get filter image
    mask = cv.inRange(img_hsv, hsv_low, hsv_high)
    img_hsv_mask = cv.bitwise_and(img_orig, img_orig, mask=mask)

    # Apply Canny edge detector
    img_canny = cv.Canny(mask, thres1, thres2,
                         apertureSize=aperture_list[aperture])

    # Show images
    cv.imshow('Source',img_orig)
    cv.imshow('Mask',mask)
    cv.imshow('HSV Mask', img_hsv_mask)
    cv.imshow('Canny', img_canny)

    # Exit on key enter
    k = cv.waitKey(1) & 0xFF
    if k == 27:
        break

# Close all windows
cv.destroyAllWindows()
```

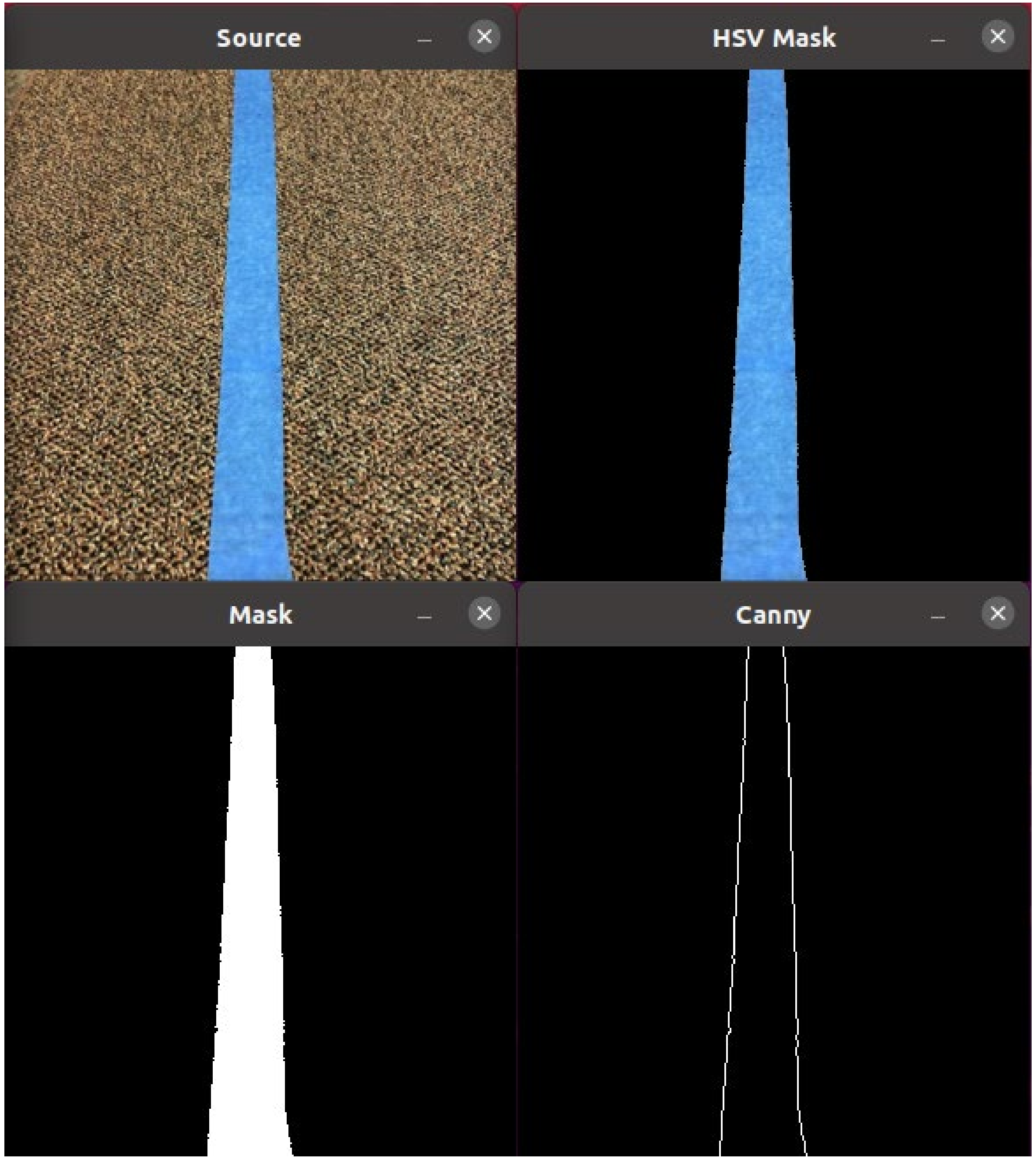
Canny Edge Detection Example

Let’s revisit our line following and color spaces discussion…

Execute from ~/catkin_ws/src/course_tutorials/scripts

$ python3 opencv_**hsv**_canny.py ../img/blue_line_floor_1.png





Hough Transforms

- Hough transforms provides methods to converting the Canny edges image into a database of lines, circles or other parametric curves

- We will focus on line extraction

◆ HoughLines()

```
void cv::HoughLines ( InputArray   image,
                      OutputArray  lines,
                      double       rho,
                      double       theta,
                      int          threshold,
                      double       srn = 0 ,
                      double       stn = 0 ,
                      double       min_theta = 0 ,
                      double       max_theta = CV_PI
                    )
```

Python:

cv.HoughLines(image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]]) -> lines

`#include <opencv2/imgproc.hpp>`

Finds lines in a binary image using the standard Hough transform.

The function implements the standard or standard multi-scale Hough transform algorithm for line detection. See http://homepages.inf.ed.ac.uk/rbf/HIPR2/hough.htm for a good explanation of Hough transform.

Parameters

image	8-bit, single-channel binary source image. The image may be modified by the function.
lines	Output vector of lines. Each line is represented by a 2 or 3 element vector (ρ, θ) or $(\rho, \theta, \textrm{votes})$. ρ is the distance from the coordinate origin $(0, 0)$ (top-left corner of the image). θ is the line rotation angle in radians ($0 \sim \textrm{vertical line}, \pi/2 \sim \textrm{horizontal line}$). $\textrm{votes}$ is the value of accumulator.
rho	Distance resolution of the accumulator in pixels.
theta	Angle resolution of the accumulator in radians.
threshold	Accumulator threshold parameter. Only those lines are returned that get enough votes ($>\texttt{threshold}$).
srn	For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The coarse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive.
stn	For the multi-scale Hough transform, it is a divisor for the distance resolution theta.
min_theta	For standard and multi-scale Hough transform, minimum angle to check for lines. Must fall between 0 and max_theta.
max_theta	For standard and multi-scale Hough transform, maximum angle to check for lines. Must fall between min_theta and CV_PI.

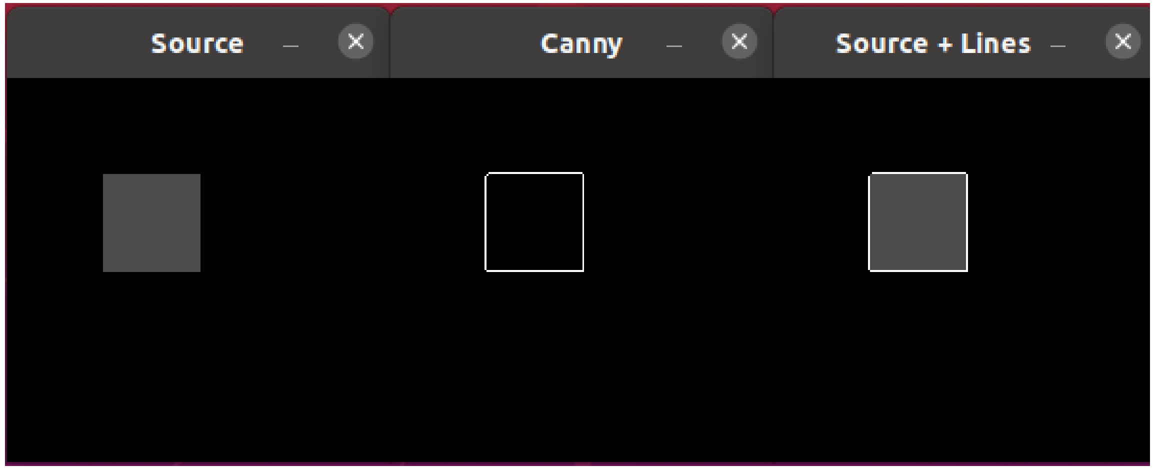
Hough Transforms Example

opencv_hough.py

```python
#!/usr/bin/env python3

# Import OpenCV and NumPy
import cv2 as cv
import numpy as np

# Define a blank, grayscale image
img = np.zeros( shape=(200,200,1), dtype='uint8')

# Draw a rectangle
pt1 = (50,50)
pt2 = (100,100)
color = 75
thickness = -1
cv.rectangle(img, pt1, pt2, color, thickness)

# Apply Canny edge detector
img_canny = cv.Canny(img, 0, 127)

# Apply the Hough Transform to find lines
rho = 1 # distance precision in pixel, i.e. 1 pixel
angle = np.pi / 180  # angular precision in radian, i.e. 1 degree
min_threshold = 10  # minimal of votes
line_segments = cv.HoughLinesP(img_canny, rho, angle,
                               min_threshold, np.array([]),
                               minLineLength=10, maxLineGap=10)

# Draw new image with lines
img_lines = img.copy()
if( line_segments is not None ):
    for line in line_segments:
        pts = line[0]
        pt1 = (pts[0],pts[1])
        pt2 = (pts[2],pts[3])
        color = 255
        thickness = 1
        cv.line(img_lines, pt1, pt2, color, thickness)
        print('Line (%d,%d) -> (%d,%d)' % (pts[0], pts[1], pts[2], pts[3]))

# Show the Canny image
cv.imshow('Source', img)
cv.imshow('Source + Lines', img_lines)
cv.imshow('Canny', img_canny)
cv.waitKey(0)
```

Output

```
Line (100,100) -> (100,50)
Line (49,99) -> (49,51)
Line (51,49) -> (99,49)
Line (50,100) -> (99,100)
```



Putting it all together

- We can use the following tools to achieve our goal of find the edges of a lane line

- Process
    1. Apply HSV filtering to get an image of only the lane line
    2. Apply Canny Edge Detection to find the edges of the line
    3. Use Hough Line Transform to get a list of lines (and associated points) defining the edges of the lane lines

Complete Solution

opencv_hsv_canny_hough.py

```python
import cv2 as cv
import numpy as np
import sys

# Color ranges
H_low = 0
H_high = 179
S_low= 0
S_high = 255
V_low= 0
V_high = 255
thres1 = 0
thres2 = 0
aperture = 0
aperture_list = [3,5,7]

# Trackbar callback fucntion to update HSV value
def callback(x):
    global H_low, H_high, S_low, S_high, V_low, V_high
    global thres1, thres2, aperture
    H_low = cv.getTrackbarPos('low H','controls')
    H_high = cv.getTrackbarPos('high H','controls')
    S_low = cv.getTrackbarPos('low S','controls')
    S_high = cv.getTrackbarPos('high S','controls')
    V_low = cv.getTrackbarPos('low V','controls')
    V_high = cv.getTrackbarPos('high V','controls')
    thres1 = cv.getTrackbarPos('thres1','controls')
    thres2 = cv.getTrackbarPos('thres2','controls')
    aperture = cv.getTrackbarPos('aperture_3_5_7','controls')

    return
# Create the a controls window
cv.namedWindow('controls',2)

# Create trackbars for Low and High B, G, R
cv.createTrackbar('low H','controls',    0, 179, callback)
cv.createTrackbar('high H','controls', 179, 179, callback)
cv.createTrackbar('low S','controls',    0, 255, callback)
cv.createTrackbar('high S','controls', 255, 255, callback)
cv.createTrackbar('low V','controls',    0, 255, callback)
cv.createTrackbar('high V','controls', 255, 255, callback)
cv.createTrackbar('thres1','controls',    0, 255, callback)
cv.createTrackbar('thres2','controls',    0, 255, callback)
cv.createTrackbar('aperture_3_5_7','controls',  0, 2, callback)
```

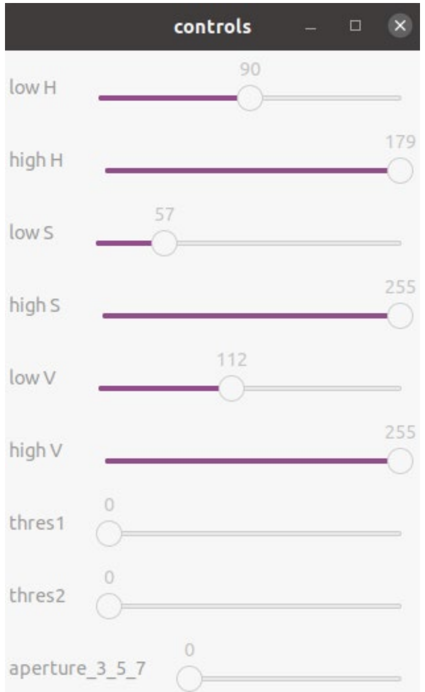
opencv_hsv_canny_hough.py

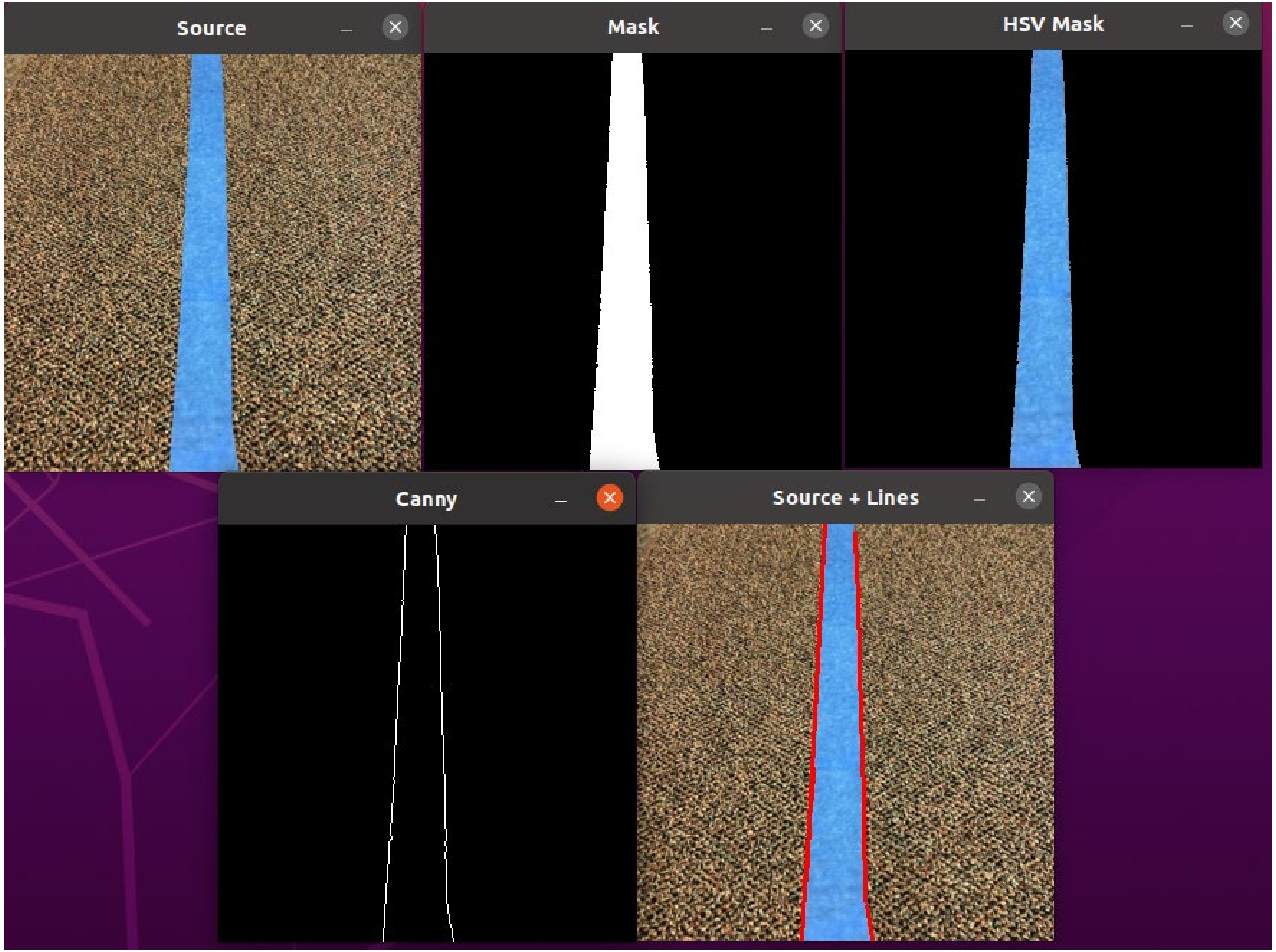
```python
while(1):

    # Conver to HSV
    img_hsv = cv.cvtColor(img_orig, cv.COLOR_BGR2HSV)

    # Set up bounds
    hsv_low = np.array([H_low, S_low, V_low], np.uint8)
    hsv_high = np.array([H_high, S_high, V_high], np.uint8)

    # Get filter image
    mask = cv.inRange(img_hsv, hsv_low, hsv_high)
    img_hsv_mask = cv.bitwise_and(img_orig, img_orig, mask=mask)

    # Apply Canny edge detector
    img_canny = cv.Canny(mask, thres1, thres2,
                         apertureSize=aperture_list[aperture])

    # Apply the Hough Transform to find lines
    rho = 1
    angle = np.pi / 180
    min_threshold = 10
    line_segments = cv.HoughLinesP(img_canny, rho, angle,
                                   min_threshold, np.array([]),
                                   minLineLength=10, maxLineGap=10)

    # Draw new image with lines
    img_lines = img_orig.copy()
    if( line_segments is not None  ):
        for line in line_segments:
            pts = line[0]
            pt1 = (pts[0],pts[1])
            pt2 = (pts[2],pts[3])
            color = (0,0,255)
            thickness = 2
            end_pt_rad = 5
            cv.line(img_lines, pt1, pt2, color, thickness)
```

Complete Solution

opencv_hsv_canny_hough.py

```
    # Show images
    cv.imshow('Source',img_orig)
    cv.imshow('Mask',mask)
    cv.imshow('HSV Mask', img_hsv_mask)
    cv.imshow('Canny', img_canny)
    cv.imshow('Source + Lines', img_lines)

    # Exit on key enter
    k = cv.waitKey(1) & 0xFF
    if k == 27:
        break

# Close all windows
cv.destroyAllWindows()
```





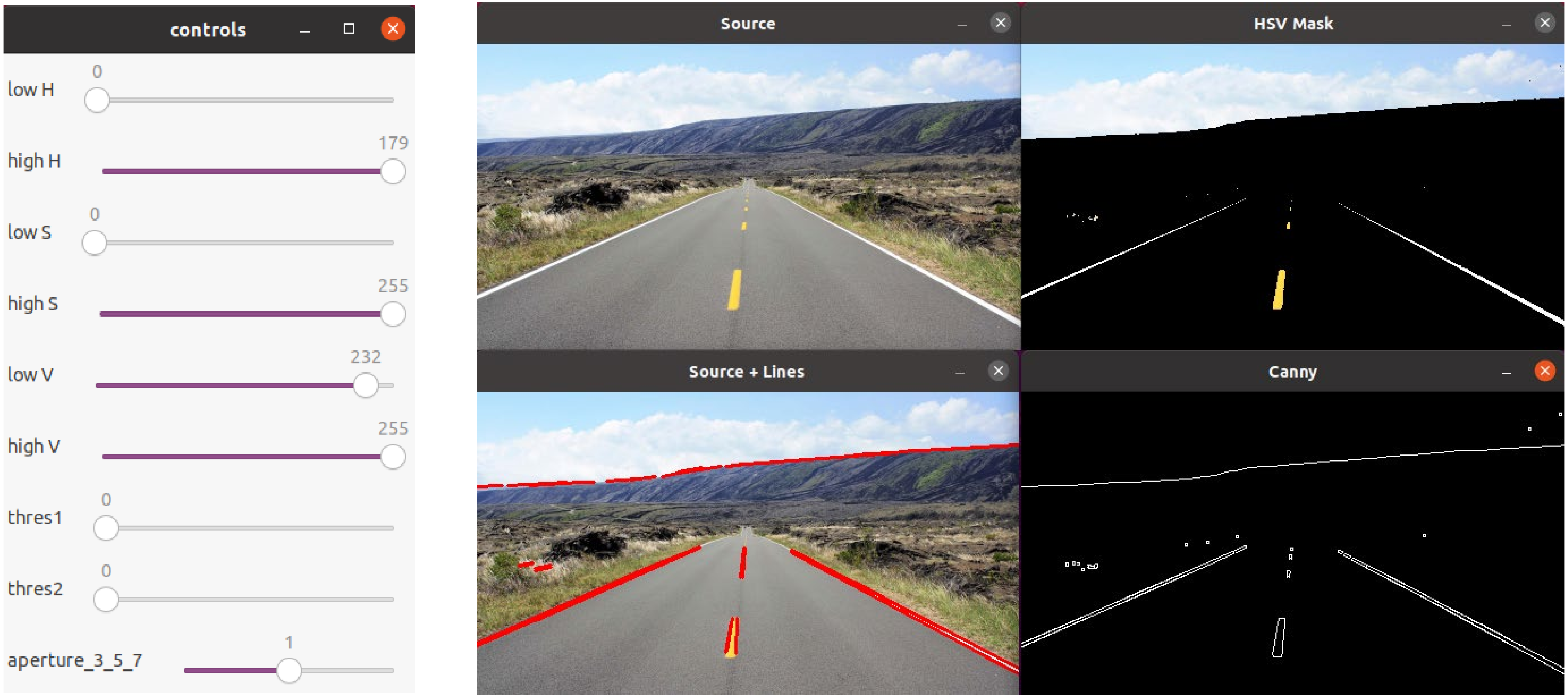
python3 opencv_hsv_canny_hough.py ../img/blue_line_floor_1.png

Overview

- Line following
- Advanced image processing
- Lane following
- Summary

Lane Following

- For real work scenarios, the problem becomes more challenging



Overview

- Line following
- Advanced image processing
- Lane following
- Summary

Summary

- We learned about different techniques to implement line following
- We also learned how to apply various image processing methods to find object edges
- This methods can be combined to developed more advanced line following and lane following/centering algorithms