ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยขอนแก่น

จัดทำโดย นายภูบดินทร์ มีหอม 663040126-6

Mini Project : Lane Line Detection

วัตถุประสงค์:

เพื่อศึกษาและประยุกต์ใช้เทคนิคการประมวลผลภาพดิจิทัล ในการแก้ปัญหาจริง เช่น การแปลงภาพสีเป็นภาพ Grayscale, การลดสัญญาณรบกวน (Gaussian Blur), และการตรวจจับขอบ (Canny Edge Detection) และการนำ Hough Transform มาใช้ในการตรวจจับและระบุตำแหน่งของเส้นตรง จากข้อมูลขอบของวัตถุในภาพ

Introduction

1. Grayscale Conversion

กระบวนการแปลงข้อมูลภาพสีซึ่งอยู่ในปริภูมิสี (Color Space) แบบ RGB ที่ประกอบด้วย 3 ช่องสัญญาณสี (Red, Green, Blue) ให้เป็นภาพเกรย์สเกล (Grayscale) ซึ่งมีเพียง 1 ช่องสัญญาณที่แสดงระดับความสว่าง (Intensity) ของแต่ละพิกเซล

วัตถุประสงค์ ในการทดลองนี้คือเพื่อลดมิติของข้อมูล (Dimensionality Reduction) ทำให้การประมวลผลในขั้นตอนต่อไป เช่น การตรวจจับขอบ ทำได้อย่างมีประสิทธิภาพและรวดเร็วยิ่งขึ้น โดยการคำนวณค่าความสว่างจะใช้สมการเฉลี่ยถ่วงน้ำหนักดังนี้:

$$Intensity = (0.299 \times R) + (0.587 \times G) + (0.114 \times B)$$

2. Gaussian Blur

เป็นเทคนิคการกรองในระนาบพื้นที่ (Spatial Filtering) ที่ใช้เพื่อลดทอนสัญญาณรบกวนความถี่สูง (High-frequency Noise) ในภาพ หลักการทำงานคือการประมวลผลแบบคอนโวลูชัน (Convolution) ระหว่างภาพต้นฉบับกับเคอร์เนลแบบเกาส์ (Gaussian Kernel) ซึ่งเป็นการเฉลี่ยค่าความสว่างของแต่ละพิกเซลกับพิกเซลข้างเคียงโดยให้น้ำหนักตามการแจกแจงแบบปกติ

วัตถุประสงค์ คือเพื่อปรับสภาพของภาพให้มีความนุ่มนวล (Smooth) และกำจัดสัญญาณรบกวนที่อาจส่งผลให้เกิดการตรวจจับขอบที่ผิดพลาด (False Edge Detection)

3. Canny Edge Detection

อัลกอริทึมการตรวจจับขอบ (Edge Detection Algorithm) ที่มีประสิทธิภาพสูงและเป็นที่นิยม จัดเป็นส่วนหนึ่งของกระบวนการแบ่งส่วนภาพโดยใช้ขอบ (Edge-Based Segmentation) อัลกอริทึมนี้ประกอบด้วยขั้นตอนย่อย ได้แก่ การลดสัญญาณรบกวน, การคำนวณขนาดและทิศทางของเกรเดียนต์, การกำจัดขอบที่ไม่ใช่ค่าสูงสุดเฉพาะที่ (Non-maximum Suppression), และการตัดสินขอบด้วยค่าขีดแบ่งสองระดับ (Hysteresis Thresholding)

วัตถุประสงค์ คือเพื่อสกัดหาพิกเซลที่เป็นขอบของเส้นแบ่งเลน ซึ่งเป็นบริเวณที่มีการเปลี่ยนแปลงของค่าความสว่างอย่างรวดเร็ว ผลลัพธ์ที่ได้คือภาพไบนารี (Binary Image) ที่แสดงเฉพาะเส้นขอบที่บางและมีความต่อเนื่อง

4. Region of Interest (ROI)

การกำหนดขอบเขตพื้นที่ที่สนใจภายในภาพ เพื่อจำกัดการประมวลผลให้อยู่ในบริเวณที่เกี่ยวข้องเท่านั้น ในการทดลองนี้ ROI ถูกสร้างขึ้นโดยการกำหนดรูปหลายเหลี่ยม (Polygon) ให้ครอบคลุมเฉพาะบริเวณพื้นผิวถนน จากนั้นจึงสร้างหน้ากาก (Mask) และใช้การดำเนินการทางตรรกะแบบบิตไวส์ (Bitwise AND Operation) เพื่อกรองเอาเฉพาะข้อมูลขอบที่อยู่ภายใน ROI

วัตถุประสงค์ คือเพื่อกำจัดข้อมูลที่ไม่จำเป็น (เช่น ขอบของต้นไม้, ท้องฟ้า) ออกจากการประมวลผล ซึ่งช่วยเพิ่มความแม่นยำและลดภาระการคำนวณในการตรวจจับเส้นในขั้นตอนสุดท้าย

5. Hough Transform

เทคนิคการสกัดคุณลักษณะเด่น (Feature Extraction Technique) ที่ใช้สำหรับตรวจจับรูปร่างทางเรขาคณิตที่สามารถแทนด้วยพารามิเตอร์ได้ เช่น เส้นตรงหรือวงกลม หลักการสำคัญคือการแปลงพิกัดของจุดจากระนาบภาพ (Image Space) ไปยังระนาบพารามิเตอร์ (Parameter Space) หรือ Hough Space จากนั้นจึงทำการ "โหวต" ในเซลล์ของ Accumulator Array เพื่อหาค่าพารามิเตอร์ของรูปร่างที่ปรากฏเด่นชัดที่สุด

วัตถุประสงค์ ในการทดลองนี้คือเพื่อตรวจจับเส้นตรงจากกลุ่มพิกเซลขอบที่ได้จากขั้นตอนก่อนหน้า โดยสมการเส้นตรงในพิกัดเชิงขั้วที่ใช้คือ:

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

โดย Hough Transform จะทำการหาค่าพารามิเตอร์ ที่มีคะแนนโหวตสูงสุด ซึ่งก็คือตัวแทนของเส้นแบ่งเลนบนถนนนั่นเอง

ขั้นตอนการทดลอง

1. Download ไฟล์การทดลอง โดยเข้า CMD แล้วพิมพ์คำสั่ง

```
$ git clone  https://github.com/phubadinee/Lane-Detection.git
```

2. หลังจาก Clone เสร็จ จะได้โฟลเดอร์ชื่อ Lane-Detection
3. เลือกไฟล์ lane_detectionV2.ipynb ในการทำการทดลอง ซึ่งเป็นไฟล์ Jupyter Notebook สามารถ execute ในแต่ละ cell ได้
4. ทำการ Import Library ที่สำคัญ

```
import os
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
import cv2
import math
%matplotlib inline
```

ทดสอบการอ่านไฟล์รูปภาพ

```
image = mpimg.imread('test_images/solidWhiteRight.jpg')
print('This image is:', type(image), 'with dimensions:', image.shape)
plt.imshow(image)
```

```
This image is: <class 'numpy.ndarray'> with dimensions: (540, 960, 3)
```

Color Selection

ทำการเลือกสี (Color Selection) โดยเลือกพิกเซลที่มีค่าสีอยู่ในช่วงที่กำหนด เพื่อแยกเส้นเลนสีขาวเหลืองออกจากพื้นถนนสีเข้ม

อ่านภาพเข้ามาในรูปแบบ Array

```
image = mpimg.imread('test_images/solidWhiteRight.jpg')
ysize = image.shape[0]
xsize = image.shape[1]
color_select = np.copy(image)
```

กำหนดค่าเกณฑ์สี (Threshold) ถ้าพิกเซลใดมีค่าสี (R, G, หรือ B) น้อยกว่าค่านี้จะถือว่าไม่ใช่เส้นเลน

```
red_threshold = 200
green_threshold = 200
blue_threshold = 200
rgb_threshold = [red_threshold, green_threshold, blue_threshold]
```

สร้าง Boolean Mask (อาเรย์ของ True/False) เพื่อระบุพิกเซลทั้งหมดที่ *ไม่ผ่าน* เกณฑ์สีใดสีหนึ่ง (ใช้ \| คือ OR)

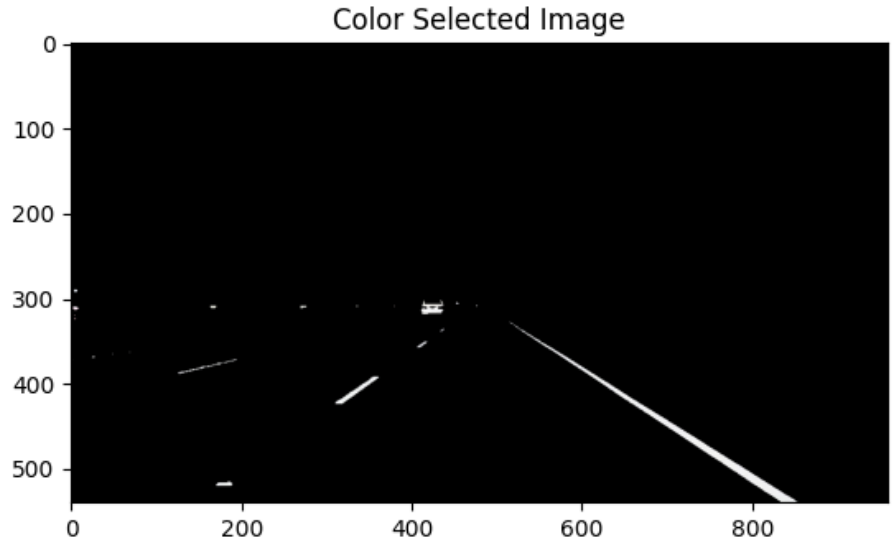
```
color_thresholds = (image[:,:,0] < rgb_threshold[0]) \|
            (image[:,:,1] < rgb_threshold[1]) \|
            (image[:,:,2] < rgb_threshold[2])
```

นำ Mask ไปใช้ โดยเปลี่ยนสีของพิกเซลที่ไม่ใช่เส้นเลน (ที่ไม่ผ่านเกณฑ์) ให้เป็น สีดำ ([0, 0, 0])

```
color_select[thresholds] = [0,0,0]
```

แสดงผลรูปภาพที่ได้

```
plt.imshow(color_select)
plt.title("Color Selected Image")
plt.show()
```



Region Masking with Line Equations

กำหนดพื้นที่ที่สนใจ (Region of Interest - ROI) โดยการวาดรูปสามเหลี่ยม (หรือสี่เหลี่ยมคางหมู) เพื่อให้ระบบประมวลผลเฉพาะส่วนที่เป็นถนนด้านหน้าของรถเท่านั้น

กำหนดจุดยอด ของรูปสามเหลี่ยมในภาพ

```
left_bottom = [100, 539]
right_bottom = [950, 539]
apex = [480, 290]
```

ใช้ฟังก์ชัน np.polyfit เพื่อหา สมการเส้นตรง $y = Ax + B$ ที่ลากผ่านจุดสองจุด (เช่น left_bottom และ apex) ซึ่งเป็นขอบของ ROI

```
fit_left = np.polyfit((left_bottom[0], apex[0]),
                      (left_bottom[1], apex[1]), 1)
fit_right = np.polyfit((right_bottom[0], apex[0]),
                       (right_bottom[1], apex[1]), 1)
fit_bottom = np.polyfit((left_bottom[0], right_bottom[0]),
                        (left_bottom[1], right_bottom[1]), 1)
```

สร้างตารางพิกัด และ ของภาพทั้งหมด

```
XX, YY = np.meshgrid(np.arange(0, xsize), np.arange(0, ysize))
```

ใช้สมการเส้นตรงที่ได้มาตรวจสอบพิกัด ของทุกพิกเซล ว่าอยู่ ภายใน ขอบเขตสามเหลี่ยมที่กำหนด หรือไม่ (ใช้ & คือ AND)

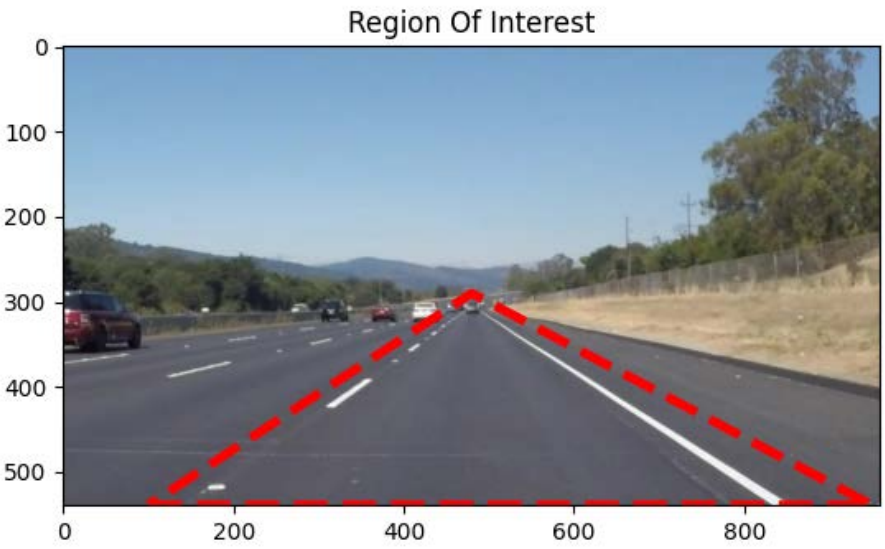
```
region_thresholds = (YY > (XX*fit_left[0] + fit_left[1])) & \
                    (YY > (XX*fit_right[0] + fit_right[1])) & \
                    (YY < (XX*fit_bottom[0] + fit_bottom[1]))
```

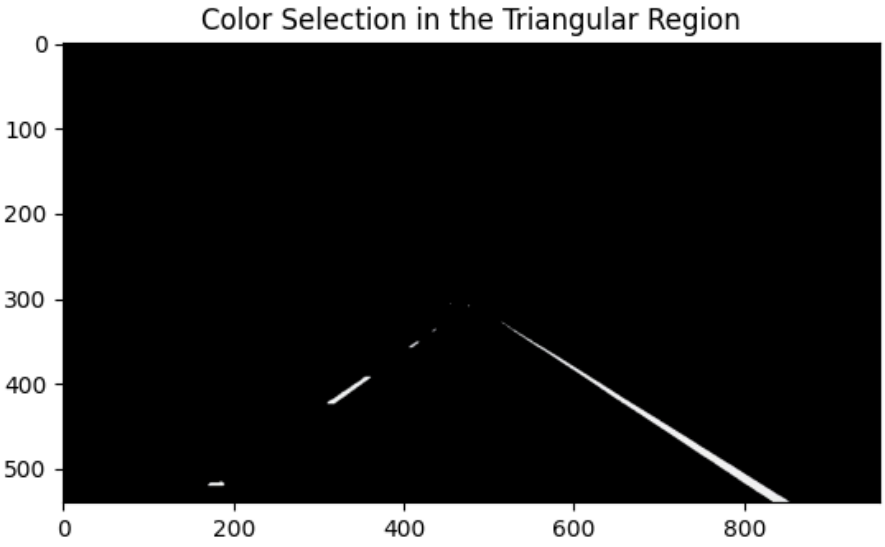
รวม Mask การเลือกสี กับ Mask พื้นที่ที่สนใจ (โดยใช้เครื่องหมาย ~ คือ NOT เพื่อกลับค่า ROI) เพื่อให้เหลือเฉพาะพิกเซลสีที่อยู่ใน ROI เท่านั้น

```
color_select[color_thresholds | ~region_thresholds] = [0, 0, 0]
```

แสดงผลรูปภาพที่ได้

```
plt.imshow(image)
x = [left_bottom[0], right_bottom[0], apex[0], left_bottom[0]]
y = [left_bottom[1], right_bottom[1], apex[1], left_bottom[1]]
plt.plot(x, y, 'r--', lw=4)
plt.title("Region Of Interest")
plt.show()
plt.imshow(color_select)
plt.title("Color Selection in the Triangular Region")
plt.show()
plt.imshow(line_image)
plt.title("Region Masked Image [Lane Lines in Green]")
plt.show()
```


Region Of Interest
0
100
200
300
400
500
0
200
400
600
800


Color Selection in the Triangular Region
0
100
200
300
400
500
0
200
400
600
800


Region Masked Image [Lane Lines in Green]
0
100
200
300
400
500
0
200
400
600
800

การตรวจจับขอบด้วย Canny Edge Detection

การแปลงภาพให้เหลือแค่ขอบ (Edge)

แปลงภาพสี (RGB) เป็น ภาพขาวดำ (Grayscale) เพราะ Canny ทำงานได้ดีที่สุดบนภาพช่องสีเดียว

```
color_select[color_thresholds | ~region_thresholds] = [0, 0, 0]
```

ใช้ Gaussian Blur เพื่อลดสัญญาณรบกวน (Noise) ก่อนการตรวจจับขอบ

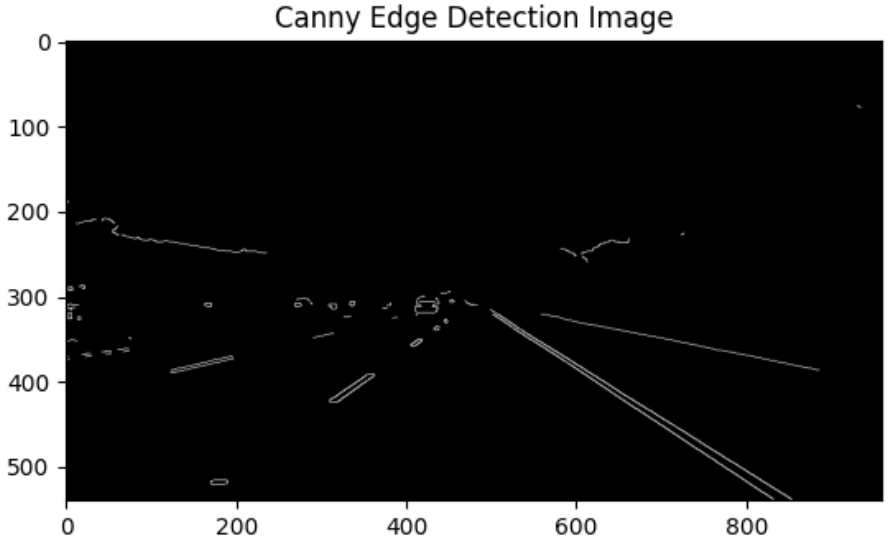
```
kernel_size = 5
blur_gray = cv2.GaussianBlur(gray,(kernel_size, kernel_size),0)
```

ใช้ฟังก์ชัน Canny Edge Detector เพื่อหาขอบ (Edges) โดยจะเก็บเฉพาะพิกเซลที่มีการเปลี่ยนแปลงความเข้มของสีอย่างรวดเร็ว (Edges)

```
low_threshold = 180
high_threshold = 240
edges = cv2.Canny(blur_gray, low_threshold, high_threshold)
```

แสดงผลรูปภาพที่ได้

```
plt.imshow(edges, cmap='Greys_r')
plt.title("Canny Edge Detection Image")
plt.show()
```



Hough Transform

นำขอบที่ได้มาค้นหาเส้นตรงจริง ๆ บนภาพด้วยเทคนิค Hough Transform

สร้างอาเรย์ว่างเปล่า mask ที่มีขนาดเท่ากับภาพขอบ Canny (edges) และกำหนดให้ทุกพิกเซลมีค่าเป็น 0

```
mask = np.zeros_like(edges)
```

กำหนดค่าที่จะใช้ระบายลงบนพื้นที่ที่ต้องการ (ROI) ซึ่งในภาพขาวดำ/ขอบ (8-bit) จะใช้ค่า 255 (สีขาว)

```
ignore_mask_color = 255
```

สร้างอาเรย์ของพิกัด สี่เหลี่ยมคางหมู (4 จุดยอด) เพื่อครอบคลุมพื้นที่ถนนด้านหน้าของรถเท่านั้น

```
vertices = np.array([[(0,imshape[0]),(450, 290), (490, 290),
                      (imshape[1],imshape[0])]], dtype=np.int32)
```

ใช้ฟังก์ชัน cv2.fillPoly เพื่อระบายสีขาว (255) ลงบนอาเรย์ mask เฉพาะในพื้นที่ที่กำหนดโดยพิกัด vertices

```
cv2.fillPoly(mask, vertices, ignore_mask_color)
```

ใช้การดำเนินการ cv2.bitwise_and เพื่อรวมภาพขอบ (edges) กับภาพ mask ผลลัพธ์ที่ได้คือ masked_edges ซึ่งมีขอบเฉพาะส่วนที่ทับซ้อนกับสีขาวบน mask เท่านั้น (ขอบนอก ROI จะถูกตัดทิ้ง)

```
masked_edges = cv2.bitwise_and(edges, mask)
```

กำหนดความละเอียดของกริด Hough ในหน่วยพิกเซล (1 พิกเซล)

```
rho = 1
```

กำหนดความละเอียดเชิงมุมของกริด Hough ในหน่วยเรเดียน (เท่ากับ 1 องศา)

```
theta = np.pi/180
```

กำหนดจำนวนคะแนนโหวต (Intersections) ขั้นต่ำที่ต้องสะสมในเซลล์ของกริด Hough เพื่อให้ถูกพิจารณาว่าเป็นเส้นตรง

```
threshold = 2
```

กำหนดความยาวต่ำสุดของเส้น (ในหน่วยพิกเซล) ที่จะถูกยอมรับว่าเป็นเส้นตรง

```
min_line_length = 4
```

กำหนดช่องว่างสูงสุดระหว่างกลุ่มพิกเซลที่อยู่ในเส้นตรงเดียวกัน หากช่องว่างน้อยกว่าค่านี้ กลุ่มพิกเซลเหล่านั้นจะถูกเชื่อมต่อกันเป็นเส้นตรงเดียว

```
max_line_gap = 5
```

สร้างภาพว่าง (สีดำทั้งหมด) ที่มีขนาดและจำนวนช่องสีเท่ากับภาพต้นฉบับ เพื่อใช้เป็นผ้าใบสำหรับวาดเส้นเลนที่ตรวจพบ

```
line_image = np.copy(image)*0
```

ใช้ฟังก์ชัน Probabilistic Hough Line Transform เพื่อค้นหาเส้นตรงจากภาพ masked_edges ผลลัพธ์ที่ได้คืออาเรย์ lines ซึ่งประกอบด้วยพิกัดจุดเริ่มต้น (x_1, y_1) และจุดสิ้นสุด (x_2, y_2) ของแต่ละเส้นตรงที่ตรวจพบ

```
lines = cv2.HoughLinesP(masked_edges, rho, theta, threshold, np.array([]),
                        min_line_length, max_line_gap)
```

การวาดเส้นและการรวมภาพ

```
for line in lines:
    for x1,y1,x2,y2 in line:
        cv2.line(line_image,(x1,y1),(x2,y2),(255,0,0),10)

color_edges = np.dstack((edges, edges, edges))

lines_edges = cv2.addWeighted(color_edges, 0.8, line_image, 1, 0)
lines_edges = cv2.polylines(lines_edges,vertices, True, (0,0,255), 10)
```

แสดงผลรูปภาพที่ได้

```
plt.imshow(lines_edges)
plt.title("Colored Lane line [In RED] and Region of Interest [In Blue]")
plt.show()
```