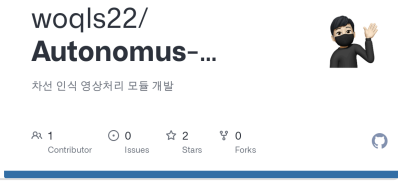
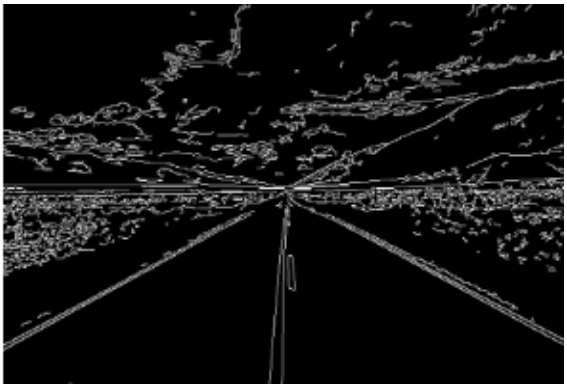
자율주행 영상처리 모듈 개발 코드 리뷰

GitHub - woqls22/Autonomus-Vehicle: 차선 인식 영상처리 모듈 개발
차선 인식 영상처리 모듈 개발. Contribute to woqls22/Autonomus-Vehicle development by creating an account on GitHub.
https://github.com/woqls22/Autonomus-Vehicle/tree/master

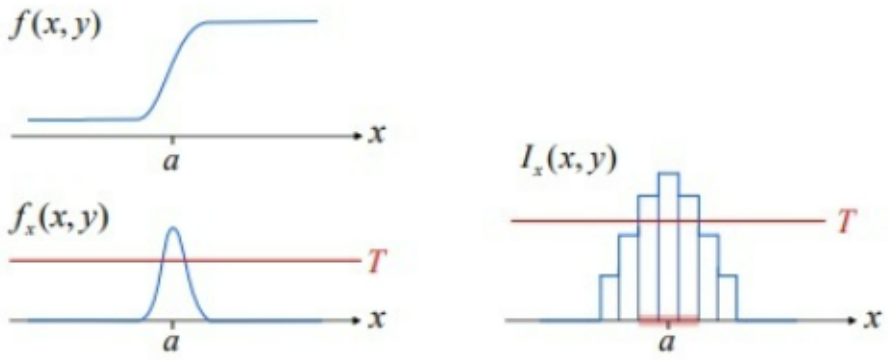


- 차선 인식 (LaneRecognition.py)
  - **캐니 에지 디텍션**

```
def Canny(frame):
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray, (5,5), 0)
    canny = cv2.Canny(blur, 50, 200)
    return canny
```



    - 개념 : 픽셀 값이 급격하게 변하는 부분을 이용해 테두리 인식
      - 미분 : 픽셀 값의 변화율 측정



- 미분값이 특정 값 T보다 크다면 edge로 판단

- 흑백 변환 : 케니 에지 검출 시 입력 영상으로 컬러 영상을 입력하면 RGB 각각의 픽셀 값에 대해 전부 미분하기 때문에 계산량을 줄이기 위해 흑백으로 바꿔 입력하는 것이 좋음

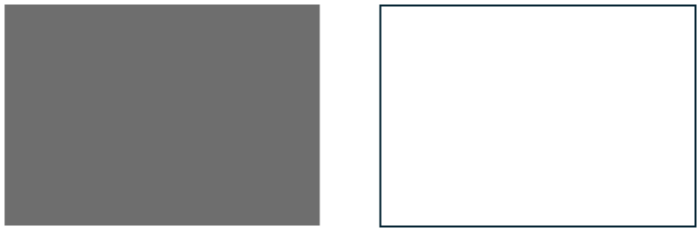
- **관심 영역 추출 (차선 검출)**

```
def InterestRegion(frame, width, height):
    # Width 1280기준
    frame = np.array(frame, dtype=np.uint8)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    lower_white = np.array([110,110,110])
    upper_white = np.array([255,255,255])

    mask_white = cv2.inRange(rgb, lower_white, upper_white)
    res = cv2.bitwise_and(frame, frame, mask = mask_white)

    area = np.array([[(width*0.5,(height*0.4)),(0,(height*0.65)),(0,height), (wi
    mask = np.zeros_like(frame)
    cv2.fillPoly(mask, area, (0,150,255))
    interestarea = cv2.bitwise_and(res, mask)
    #cv2.imshow('mask',interestarea)
    return interestarea
```

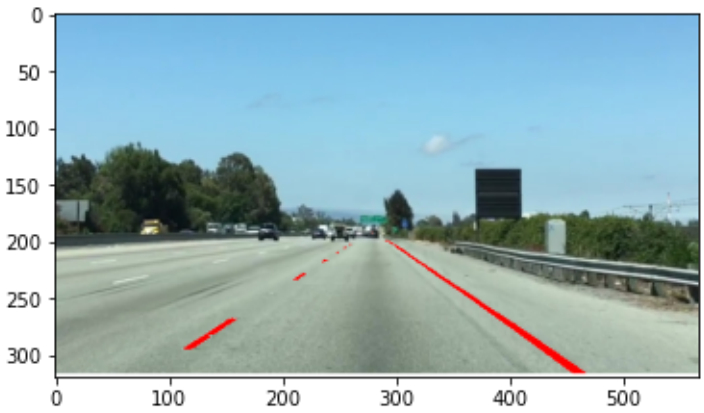
- mask_white 생성

- BGR 이미지를 RGB 이미지로 변환

- [110, 110, 110] ~ [255, 255, 255] 사이의 색을 가지는 부분 찾기 (InRange 함수 사용)



- 흰색 차선 추출
  - 찾은 흰색 영역을 원본 프레임에 적용 → 해당 영역 추출
  - cv2.bitwise_and() 이용 → 원본 이미지와 마스크 적용하여 흰색 영역만 추출
- 관심 영역 마스크 생성
  - 관심 영역 정의 위해 다각형 모양 마스크 생성
  - cv2.fillPoly() 이용 → 관심 영역 채우기
  - cv2.bitwise_and() 이용 → 관심 영역 마스크를 원본 이미지에 적용
- 추출된 관심 영역 반환 → 차선 강조, 다른 영역 무시

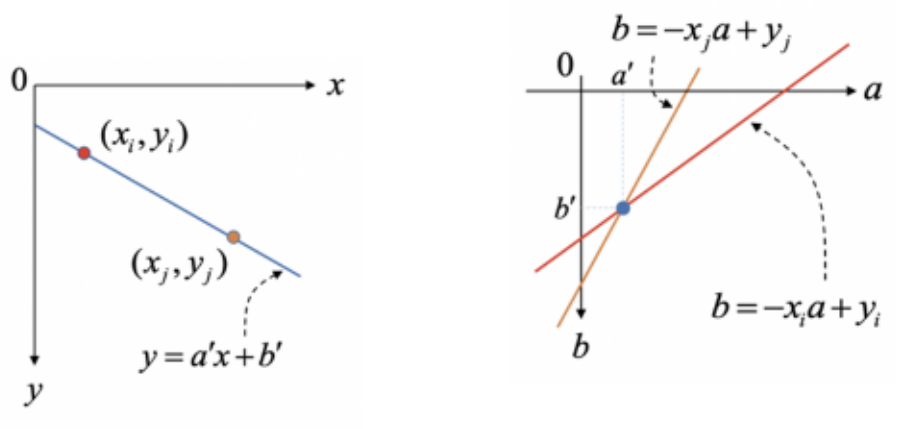


- 허프 변환

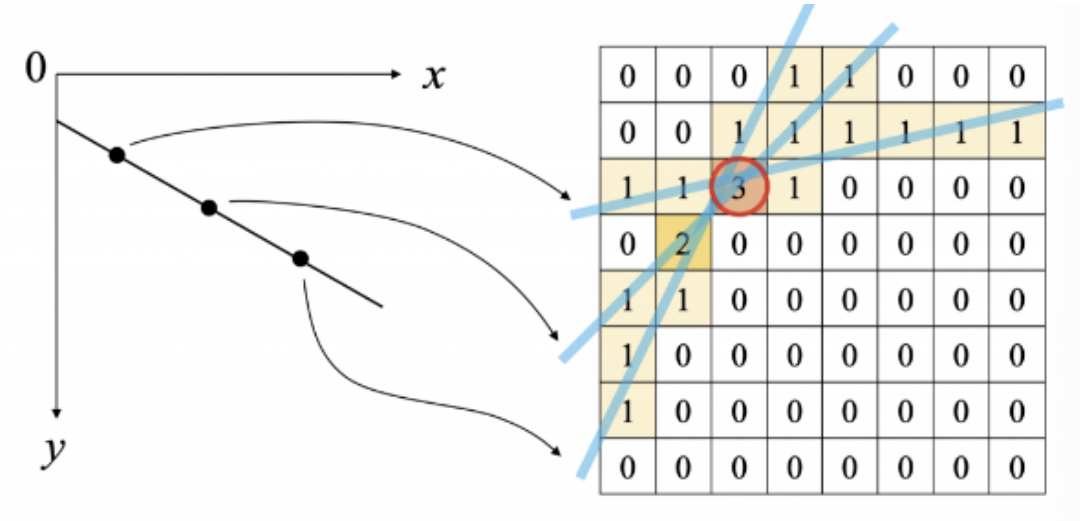
```
def Hough_lines(interestregion, rho, theta, threshold, min_line_len, max_line_
    lines = cv2.HoughLinesP(interestregion,rho,theta,threshold,np.array([]),mi
    return lines
```

- 개념 : Canny edge를 통해 추출한 에지 정보를 사용해서 직선을 검출

- xy 공간에서의 직선 방정식 → ab 공간에서의 직선 방정식



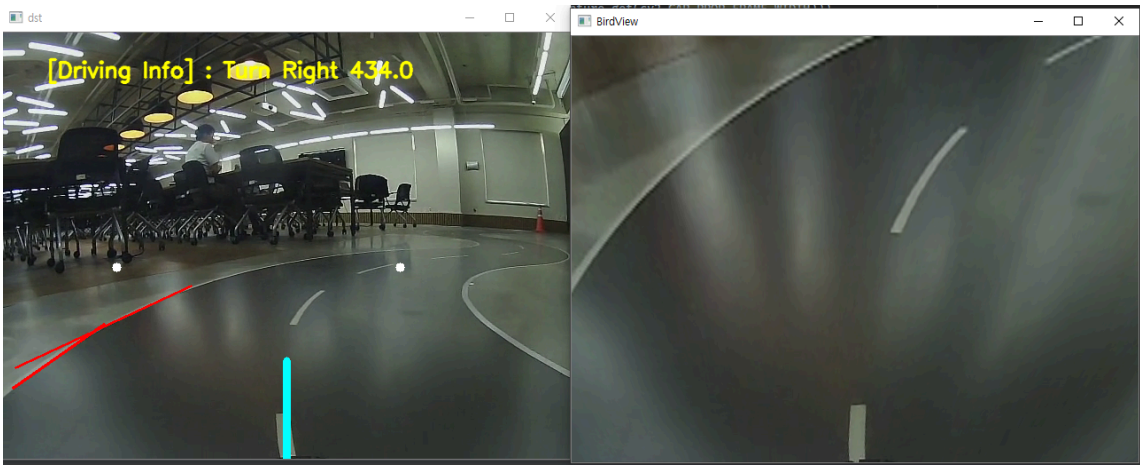
  - 두 직선의 방정식의 교점 좌표는 xy 공간에서 두 점 (xi, yi)와 (xj, yj)을 동시에 지나는 직성의 (기울기, y절편)을 의미
- xy 공간에서 영상의 에지 픽셀들을 허프 변환하여 표현된 ab 공간 상의 직선들이 많이 교차되는 지점으로 직선 검출 가능
- 축적 배열 : 0으로 초기화된 2차원 배열에서 직선이 지나가는 위치의 배열 원소 값을 1씩 증가시켜 생성



  - 축적 배열 값이 일정 임계값 이상인 점의 a와 b로 직선 검출

◦ bird view

```
def top_view(frame, width, height):
    #area = np.array([[(width*0.5,(height*0.4)),(0,(height*0.65)),(0,height), (w
    #좌상, 좌하, 우상, 우하
    '''
    cv2.circle(frame, (int(width*0.2), int(height*0.6)), 5, (255, 255, 255), -1)
    cv2.circle(frame, (int(width * 0.7), int(height * 0.6)), 5, (255, 255, 255), -1)
    '''
    left_bottom = [0,height]
    right_bottom = [width,height]
    left_top = [int(width*0.2),int(height*0.55)]
    right_top = [int(width*0.8), int(height*0.55)]
    pts1 = np.float32([[left_top,left_bottom,right_top,right_bottom]])
    # 좌표의 이동점
    pts2 = np.float32([[0, 0], [0, 480], [640, 0], [640, 480]])
    # pts1의 좌표에 표시. perspective 변환 후 이동 점 확인.
    M = cv2.getPerspectiveTransform(pts1, pts2)
    dst = cv2.warpPerspective(frame, M, (640, 480))
    return dst
```



- 개념 : 원근 변환 (perspective transformation)을 사용하여 입력 이미지를 원하는 방향에서 바라보도록 수정
- 관심 영역 좌표 정의
  - 이미지 내에서 관심 영역의 네 개의 꼭지점 좌표 정의

    - 변환 행렬 계산 : cv2.getPerspectiveTransform() 함수 이용 → 입력, 출력 좌표 사이 변환 행렬 계산
    - 원근 변환 적용 : cv2.warpPerspective() 함수 이용 → 목표 크기에 맞게 이및 조정
- 장애물 인식 (YOLO3)
    - 각종 장애물들에 대해서 학습

- 하드웨어 제어, 시리얼 통신 (ros_drive.py)

```
global pub
    rospy.init_node('my_driver')
    pub = rospy.Publisher('xycar_motor_msg', Int32MultiArray, queue_size=1)
    rate = rospy.Rate(30)
    Speed = 20
    Angle = -50

    FileName = "/home/chanju/catkin_ws/src/xycar_simul/src/track-s.mkv"
    capture = cv2.VideoCapture(FileName)

    YOLO_FLAG = False
    if (not capture.isOpened()):
        print("Error : Opening Video")

    if (YOLO_FLAG):
        net = cv2.dnn.readNet("./yolov3-tiny.weights", "./yolov3-tiny.cfg")
        classes = []
```

```
        with open("./coco.names", "r") as f:
            classes = [line.strip() for line in f.readlines()]
        layer_names = net.getLayerNames()
        output_layers = [layer_names[i[0] - 1] for i in net.getUnconnectedOutLayers
        colors = (0, 0, 255)
        # colors = np.random.uniform(0, 255, size=(len(classes), 3))

    height, width = (int(capture.get(cv2.CAP_PROP_FRAME_HEIGHT)), int(capture
    retval, frame = capture.read()
    i = 0
```

```
retval, frame = capture.read()
        cv2.imshow('Original Video', frame)
        if (YOLO_FLAG):
            YOLO = frame.copy()
            heights, widths, channel = YOLO.shape
            blob = cv2.dnn.blobFromImage(YOLO, 0.00392, (416, 416), (0, 0, 0), True
            net.setInput(blob)
            outs = net.forward(output_layers)
            class_ids = []
            confidences = []
            boxes = []
            for out in outs:
                for detection in out:
                    scores = detection[5:]
                    class_id = np.argmax(scores)
                    confidence = scores[class_id]
                    if confidence > 0.7:
                        # Object detected
                        center_x = int(detection[0] * widths)
                        center_y = int(detection[1] * heights)
                        w = int(detection[2] * widths)
                        h = int(detection[3] * heights)
                        # Rectangle coordinates
                        x = int(center_x - w / 2)
                        y = int(center_y - h / 2)
```

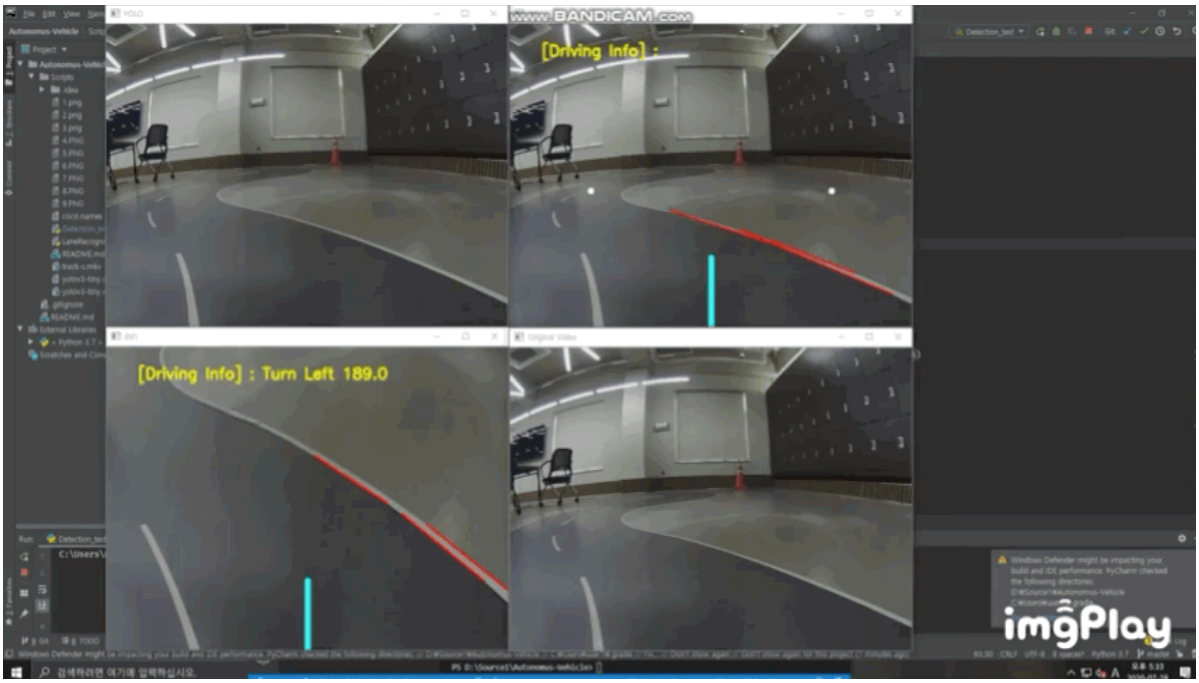
```
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)
        indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
        font = cv2.FONT_HERSHEY_PLAIN

        for i in range(len(boxes)):
            if i in indexes:
                x, y, w, h = boxes[i]
                label = str(classes[class_ids[i]])
                color = colors[i]
                cv2.rectangle(YOLO, (x, y), (x + w, y + h), (0, 255, 0), 2)
                cv2.putText(YOLO, label, (x, y + 30), font, 3, (0, 255, 255), 2)
```

```
InterestArea = LR.InterestRegion(frame, width, height)
canny = LR.Canny(InterestArea)
# bird View Point
dst = LR.top_view(frame, width, height)
src1 = LR.Canny(dst)

lines = cv2.HoughLinesP(src1, 0.8, np.pi / 180, 100, minLineLength=100, maxLine
cv2.line(dst, (int(width / 2), height), (int(width / 2), int(height / 1.3)), (255, 255, 0)
```

```
# bird View Point
lines = cv2.HoughLinesP(src1, 0.8, np.pi / 180, 100, minLineLength=100, maxLine
cv2.line(dst, (int(width / 2), height), (int(width / 2), int(height / 1.3)), (255, 255, 0)
direction = ""
if (lines is not None):
  direction = ""
  for i in lines:
    cv2.line(dst, (i[0][0], i[0][1]), (i[0][2], i[0][3]), (0, 0, 255), 2)
    if (i[0][2] < int(width / 3)):
      # cv2.circle(dst, (i[0][2], i[0][3]), 5, (255, 255, 255), -1)
      direction = "Turn Right " + str(abs(width / 2 + i[0][2]))
    elif (i[0][0] > int(width * 2 / 3)):
      # cv2.circle(dst, (i[0][0], i[0][1]), 5, (255, 255, 255), -1)
```

```
        direction = "Turn Left " + str(abs(width / 2 - i[0][0]))

dst = cv2.putText(dst, '[Driving Info] : ' + direction, (50, 50), cv2.FONT_HERSHE
                (0, 255, 255), 2, cv2.LINE_AA)
```

```
# Origin Frame-----------------------
        cv2.circle(frame, (int(width * 0.2), int(height * 0.55)), 5, (255, 255, 255), -1)
        cv2.circle(frame, (int(width * 0.8), int(height * 0.55)), 5, (255, 255, 255), -1)
        lines = cv2.HoughLinesP(canny, 1.2, np.pi / 180, 100, minLineLength=100, m
        cv2.line(frame, (int(width / 2), height), (int(width / 2), int(height / 1.3)), (255,
        if (lines is not None):
            direction = ""
            for i in lines:
                cv2.line(frame, (i[0][0], i[0][1]), (i[0][2], i[0][3]), (0, 0, 255), 2)
                if (i[0][2] < int(width / 4)):
                    # cv2.circle(frame, (i[0][2], i[0][3]), 5, (255, 255, 255), -1)
                    Angle = 50
                    pub_motor(Angle, Speed)
                    direction = "Turn Right " + str(abs(width / 2 + i[0][2]))
                elif (i[0][0] > int(width * 3 / 4)):
                    # cv2.circle(frame, (i[0][0], i[0][1]), 5, (255, 255, 255), -1)
                    Angle = -50
                    pub_motor(Angle, Speed)
                    direction = "Turn Left " + str(abs(width / 2 - i[0][0]))

        frame = cv2.putText(frame, '[Driving Info] : ' + direction, (50, 50), cv2.FONT
                        (0, 255, 255), 2, cv2.LINE_AA)
```



bird view를 이용해서 더 정확한 '차선' 검출 + 방향 판별

original view를 이용해서 차선 및 '장애물' 검출 → 종합적으로 판단하여 모터 angle 최종 조절