Vision system

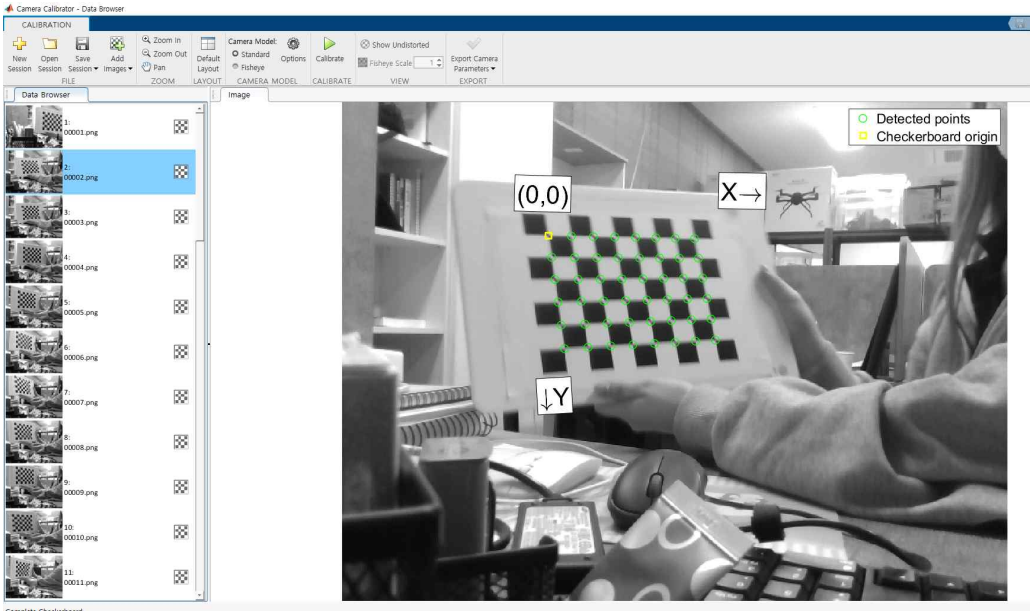
Term Project

제출일 : 2021년 06월 18일

제출자 : 21813645 이병화

1. 카메라 calibration

저는 Matlab의 Camera calibrator 앱을 사용하여 calibration 하였습니다.



사진이 흔들려 calibration에 방해가 되는 이미지 몇장을 뺀 약 40장의 이미지를 calibration을 하였습니다.

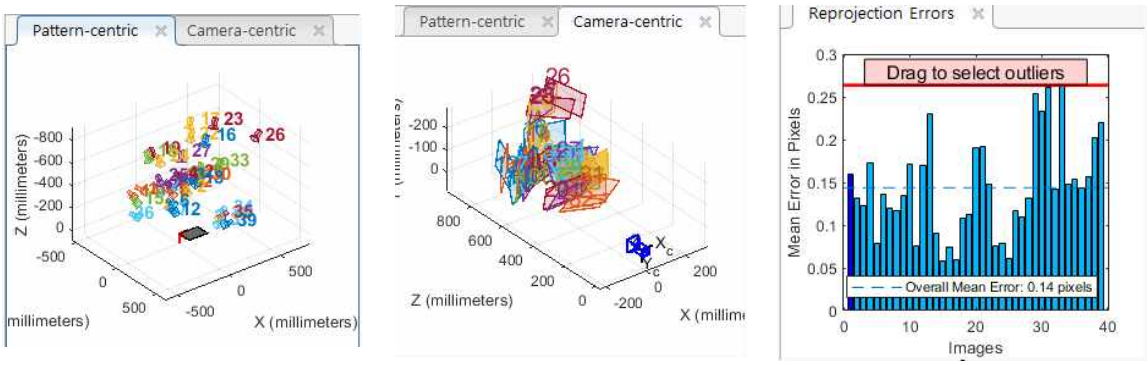


Figure 1. 캘리브레이션 결과

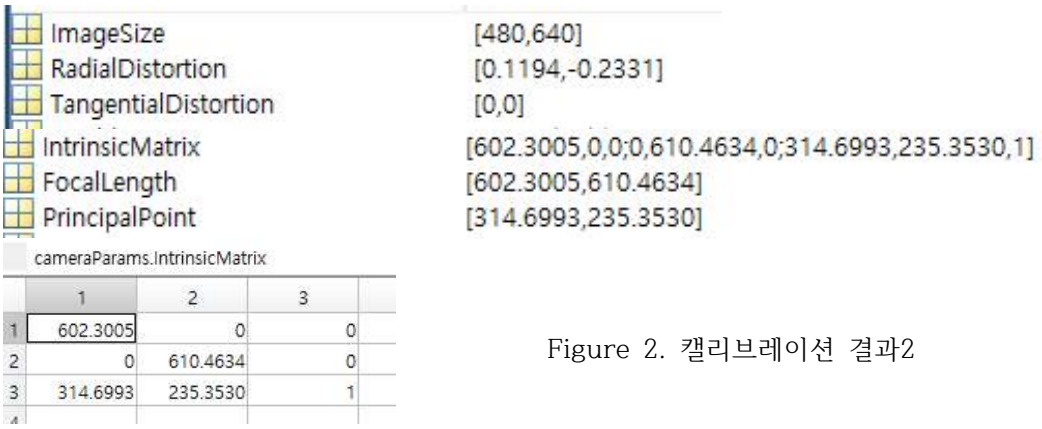
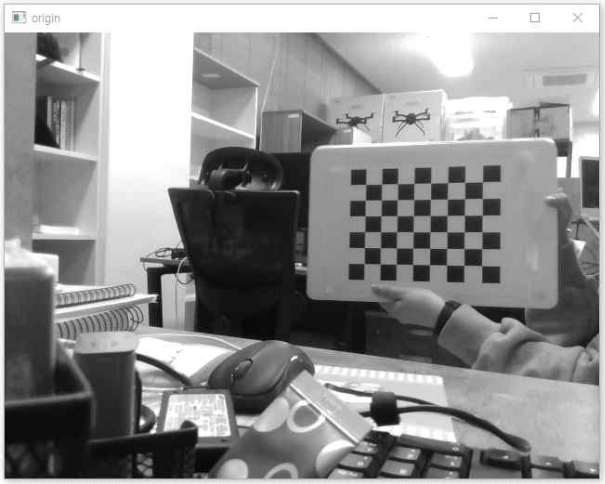


Figure 2. 캘리브레이션 결과2

위와 같이 Calibration을 파이썬에서 왜곡제거 처리와 거리를 측정할 때 필요한 FocalLength 값을 얻었습니다.

```python
#calibration
img = cv2.imread('./Inteld435i/00006_image.png')

mtx = np.array([[602.3005,0,314.6993],[0,610.4634,235.3530],[0,0,1]])
dist = np.array([[0.1194,-0.2331,0,0,0]])

h, w = img.shape[:2]

newcameramtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w,h), 1, (w,h))

cap1 = cv2.undistort(img, mtx, dist, None, newcameramtx)

#cv2.imshow('origin',img)
cv2.imshow('undistort',cap1)
```

Figure 3. Calibration code



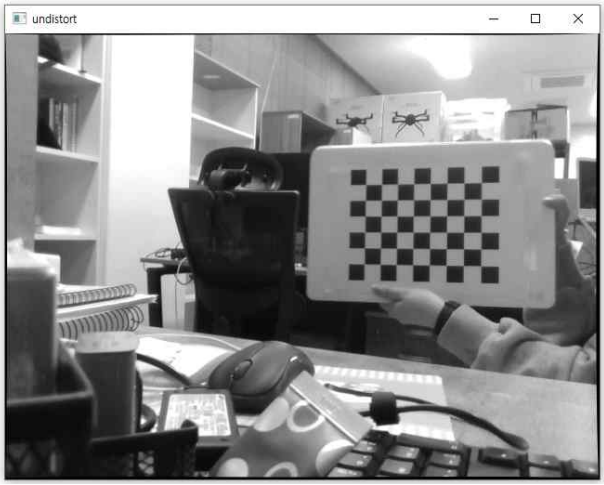


Figure 4. <왼쪽> 왜곡제거 전 <오른쪽> 왜곡제거 후

cv2.undistort 함수와 Matlab에서 얻은 카메라 파라미터와 왜곡계수를 이용하여 이미지의 왜곡을 제거하였습니다.

2. LineDetection and Following

2-1) 바닥선의 라인 검출

```python
#ROI
cap2 = cap1[320:480, 0:640].copy()
cv2.imshow('ROI',cap2)

#Bird eye view
pts1 = np.float32([[200, 0],[470, 0],[60, 160], [640, 130]])
pts2 = np.float32([[60,0],[580,0],[60,160],[580,160]])

M = cv2.getPerspectiveTransform(pts1,pts2)

cap2 = cv2.warpPerspective(cap2,M,(640,160))

cv2.imshow("birdeye",cap2)
```

Figure 5. Roi and bird eyeview code

Figure 6.
Roi 이미지

차선을 인식하기위해 필요없는 차선밖의 부분을 제거하여 ROI 처리하였습니다.

Figure 7. Bird eye view 이미지

차선의 각도로 방향을 검출하기 위해 Bird eye view 로 변환하였습니다.

```
18  #hsv 이미지처리
19  def hsv(img):
20      caphsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
21
22      h,s,v= cv2.split(caphsv)
23
24      lower_yel = (15,110,85)
25      upper_yel = (30,255,255)
26
27      yelimg = cv2.inRange(caphsv,lower_yel,upper_yel)
28
29      lower_red = (160,100,85)
30      upper_red = (190,255,255)
31
32      redimg = cv2.inRange(caphsv,lower_red,upper_red)
33      mask = yelimg + redimg
34
35      hsvimg = cv2.bitwise_and(img,img,mask = mask)
36      return hsvimg
 80  #HSV 이미지 처리
 81
 82  hsvi = hsv(cap2)
 83
 84  edges = cv2.Canny(hsvi,50,200)
 85
 86
 87  #cv2.imshow('y',yelimg)
 88  #cv2.imshow('r',redimg)
 89  cv2.imshow('edges',edges)
```

Figure 8. HSV

이미지를 HSV이미지에서 특정색 노란색 영역과 빨간색 영역을 검출하여 두 개의 이미지를 합쳐 빨간색과 노란색만을 검출하였습니다.

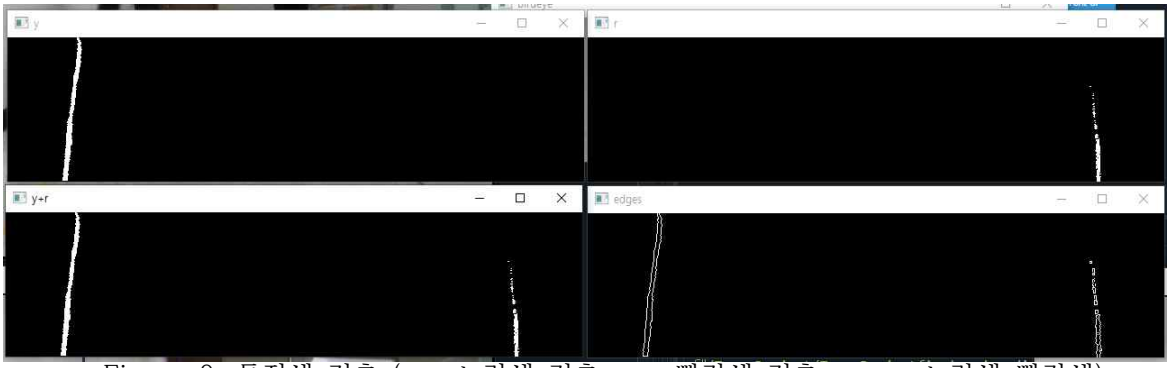


Figure 9. 특정색 검출 (y = 노란색 검출, r = 빨간색 검출, y+r = 노란색+빨간색)
edges = y+r의 엣지 검출

굳이 Edge를 검출하지 않아도 Hough transform에서 라인을 검출했지만, edge를 찾아서 검출하는게 더 잘됐습니다.

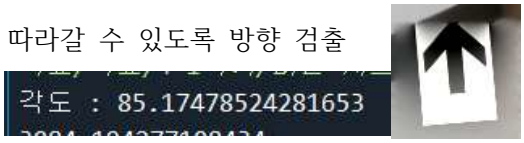
```
92  #hough transform
93
94  a = 0
95
96  lines = cv2.HoughLinesP(edges, 1, 1*np.pi/180, 40, np.array([]), 5, 20)
97  if lines is not None:
98      for line in lines:
99          for x1,y1,x2,y2 in line:
100             cv2.line(cap2, (x1, y1), (x2, y2), color=[0,0,255], thickness=2)
101             b = math.atan2((y2-y1),(x2-x1))*(180/3.14)
102             a=a+abs(b)
103     c = a/len(lines)
104     print("각도 :",c)
105
106 cv2.imshow('lines',cap2)
```

Figure 10. Hough transform code

Figure 11. Hough Transform image

edge 이미지에서 cv2.HoughLinesP()함수를 이용하여 확률적 라인검출을 하여 라인을 찾았습니다.

2-2) 차량이 따라갈 수 있도록 방향 검출



허프 변환을 하면 라인의 x, y 좌표를 알려주는데 이렇게 알아낸 x, y 좌표와 atan2를 이용하여 라인의 각도를 검출했습니다. 이렇게 검출한 각도를 평균하여 차량이 따라갈 방향을 지시해 줍니다.

3. Object Detection and Localization

3-1) 사람검출

```python
#objectdetection

cap3 = cap1.copy()

net = cv2.dnn.readNet("yolov3.weights", "yolov3.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 = np.random.uniform(0, 255, size=(len(classes), 3))

cap4 = cv2.resize(cap3, None, fx=0.4, fy=0.4)
height, width, channels = cap3.shape

blob = cv2.dnn.blobFromImage(cap4, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
net.setInput(blob)
outs = net.forward(output_layers)
```

Figure 12. Yolov3 code

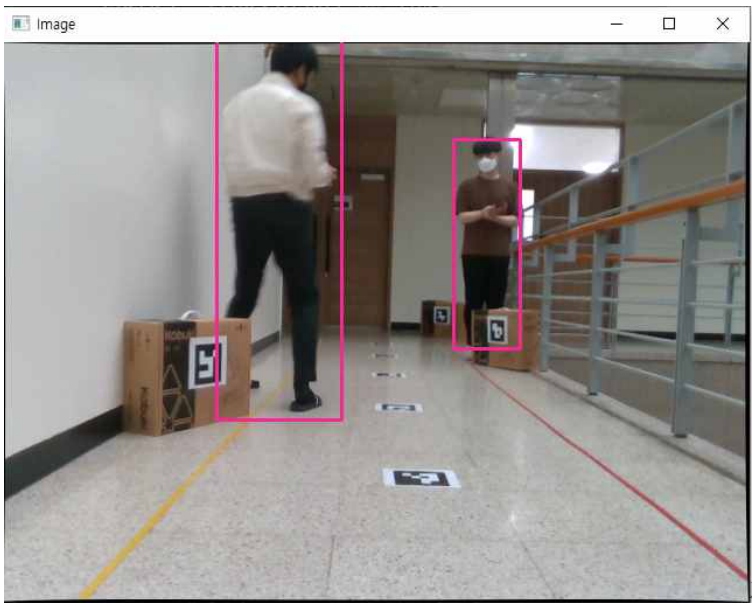
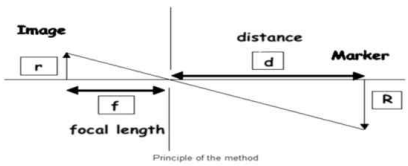


Figure 13. Object detection

HOGDescriptor 보다 간단하지는 않고 다른 weight 파일과 cfg 파일, name파일이 필요했지만 HOGDescriptor 보다 강력한 객체 인식을 할 수 있었다.

3-2) 사람 위치 인식(카메라로부터의 거리)



Principle of the method

Using the principle of Similar Triangles, we can obtain the formulas as follows:

$$\frac{f}{d} = \frac{r}{R} \quad (1)$$

$$f = d \times \frac{r}{R} \text{ pixels} \quad (2)$$

$$d = f \times \frac{R}{r} \text{ cm} \quad (3)$$

Figure 14. 픽셀과 거리 관계

카메라가 평행, 사람의 키가 1.7M라는 가정을 하고 Figure 14 이미지의 관계를 이용하여 사람과의 거리를 구하였습니다.

```
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]
        color = colors[10]
        cv2.rectangle(cap3, (x, y), ((x + w), (y + h)), color, 2)
        print((602.325*1700)/h,"mm")
```

```
3084.194277108434 mm
5720.405027932961 mm
```

Figure 15. 사람거리

L1:Y1 = L2:Y2 L1 = Focallength, Y1 = Pixel(박스의 높이), L2 = 카메라와 거리, Y2 = 사람의키

사람을 검출하였을 때 그리는 박스의 높이와 calibration에서 구했던 Focallength 사람의 키 크기 Y2 = L1*Y2/Y1 식에 대입하여 카메라와의 거리를 구할 수 있다.

이렇게 검출한 사람과의 거리는 3m와 5.7m로 검출되었다.

4. Marker Detection and Localization

4-1)Aruco 마커 검출

```
for (arucoName, arucoDict) in ARUCO_DICT.items():
    # load the ArUCo dictionary, grab the ArUCo parameters, and
    # attempt to detect the markers for the current dictionary
    arucoDict = cv2.aruco.Dictionary_get(arucoDict)
    arucoParams = cv2.aruco.DetectorParameters_create()
    (corners, ids, rejected) = cv2.aruco.detectMarkers(
        image, arucoDict, parameters=arucoParams)
    # if at least one ArUco marker was detected display the ArUco
    # name to our terminal

    if len(corners) > 0:
        print("[INFO] detected {} markers for '{}'".format(
            len(corners), arucoName))
```

Figure 16. 마커 종류 검출

트랙에서 어떤 마커가 검출되었는지 모르기 때문에 마커 종류 검사 루프를 이용하여 이미지에 사용된 마커를 검출하였습니다.

```
[INFO] loading image...
[INFO] detected 2 markers for 'DICT_4X4_50'
[INFO] detected 2 markers for 'DICT_4X4_100'
[INFO] detected 2 markers for 'DICT_4X4_250'
[INFO] detected 2 markers for 'DICT_4X4_1000'
```

Figure 17. 마커종류

이미지에 사용된 마커는 4X4의 마커가 검출되었습니다.

```
arucoDict = cv2.aruco.Dictionary_get(cv2.aruco.DICT_4X4_50)
arucoParams = cv2.aruco.DetectorParameters_create()
(corners, ids, rejected) = cv2.aruco.detectMarkers(cap6, arucoDict,
    parameters=arucoParams)
rvec, tvec, markerPoints = cv2.aruco.estimatePoseSingleMarkers(corners, 0.02, mtx, dist)
arc = cv2.aruco.drawDetectedMarkers(cap6, corners, ids, (0,255,0))
if rvec is not None:
    for i in range(len(rvec)):
        arc =cv2.aruco.drawAxis(cap6,mtx, dist, rvec[i], tvec[i], 0.01)
```

Figure 17. 마커 및 ID 검출

cv2.aruco.DetectMarkers()를 이용하여 마커의 코너와 id를 검출합니다. 검출된 마커의 코너의 위치를 이용하여 cv2.aruco.drawDetectedMarkers()함수로 마커에 사각형과 id를 출력하여줍니다.

4-2) Aruco 마커 위치 (마커 자세) 인식

cv2.aruco.estimatePoseSingleMarkers()함수를 이용하여 회전 및 변환 백터를 구합니다. 이렇게 구한 값으로 cv2.aruco.drawAxis()함수를 이용해 마커에 좌표를 그렸습니다.

5) 기능통합

```
# 기능 통합

cap5 = cap1.copy()

hsv1 = hsv(cap5)

roi_h = cap5.shape[0]
roi_w = cap5.shape[1]

region = np.array([
        [[60, roi_h], [285, 240],[330, 230], [roi_w, 430]]
    ], dtype = np.int32)

mask = np.zeros_like(cap5)

cv2.fillPoly(mask, region, 255)

region = np.array([
        [[140, roi_h], [290, 230],[350, 230], [600, 480]]
    ], dtype = np.int32)
cv2.fillPoly(mask,region, 0)

cap5 = cv2.bitwise_and(hsv1, mask)

edgesa = cv2.Canny(cap5,100,200)

cap7 = cap1.copy()

lin = hough(edgesa, 1, 1*np.pi/180, 20,10,20,cap7)

edgesa = hough(edgesa, 1, 1*np.pi/180, 20,10,20,cap3)
```

```
arrow = cv2.imread('arrow.png')
arrow = cv2.resize(arrow, dsize =(0,0), fx=0.1,fy=0.1,interpolation=cv2.INTER_LINEAR)

ih, iw = arrow.shape[:2]
#rotation
if lines is not None:
    M = cv2.getRotationMatrix2D((iw/2,ih/2),(180-c),1)
    arrot = cv2.warpAffine(arrow,M,(iw,iw))

    mask = np.full_like(arrot,255)

    al = cv2.seamlessClone(arrot,al,mask,(70,380),cv2.NORMAL_CLONE)
```

Figure 18. 기능통합 코드

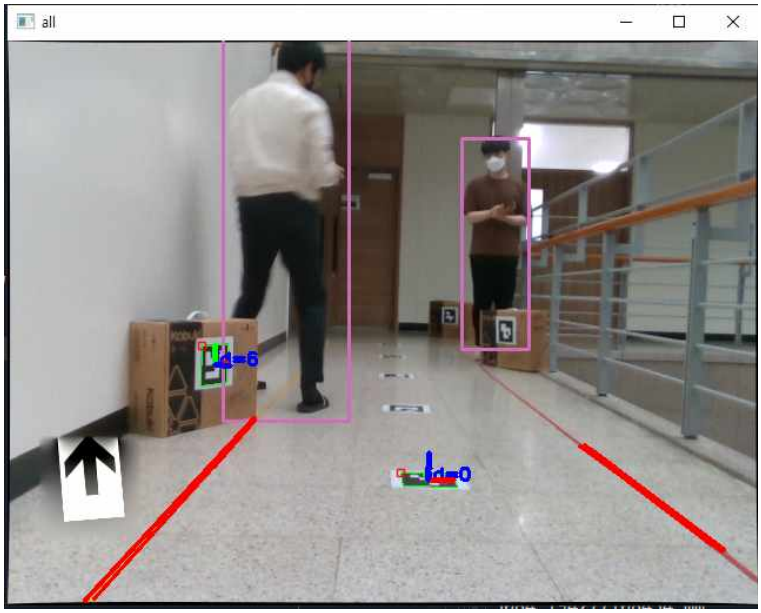


Figure 19.
기능통합 이미지

기능통합을 하기위해 원래 이미지에서 새로 ROI처리 후 Line을 그렸습니다.
라인을 그리고 라인을 그리고 앞서 했던 Objectdetection과 Markerdetection을 추가하였습니다.

6. 독자기능 추가

화살표를 표시하여 로봇의 방향을 지시했습니다.
화살표의 방향은 앞서 검출한 각도를 이용하여 cv2.RotationMatrix2D()함수와 cv2.warpAffine()함수를 이용하여 각도 값만큼 화살표를 회전시켜 나타냈습니다.
그리고 cv2.seamlessclone을 이용하여 이미지에 붙여넣기 하였습니다.

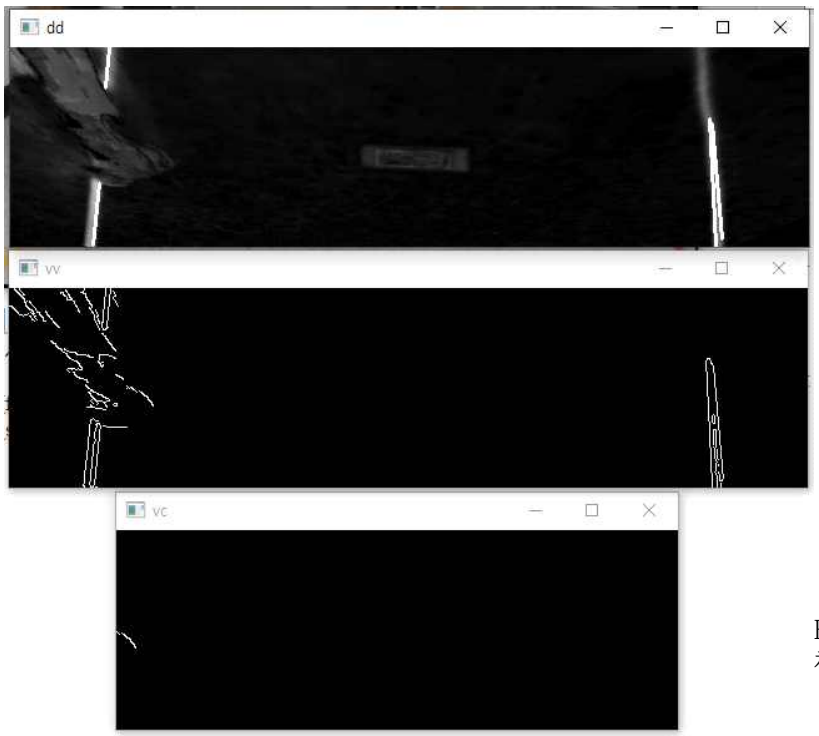


Figure 20.
차선내사람 감지

```
331    vv = cv2.Canny(s,100,200)
332
333    cv2.imshow('vv',vv)
334
335    vc = vv[0:160, 100:550].copy()
336
337    cv2.imshow('vc',vc)
338
339    vc = np.sum(vc,0)
340
341    if sum(vc)>0:
342        print('멈춰!')
```

```
각도 : 85.30367581516539
2553.4975062344142 mm
5657.196132596686 mm
[INFO] loading image...
[INFO] detected 2 markers for 'DICT_4X4_50'
[INFO] detected 2 markers for 'DICT_4X4_100'
[INFO] detected 2 markers for 'DICT_4X4_250'
[INFO] detected 2 markers for 'DICT_4X4_1000'
멈춰!
```

HSV 이미지의 S 이미지를 이용하여 마커 부분을 제외한 차선과 사람의 엣지를 검출합니다. 차선 내의 영역을 ROI 처리하여 영역 내에 사람이 검출되었다면 차량이 정지합니다.

문제점

1차 발표 때 camera calibration을 핸드폰으로 촬영한 이미지를 사용하고, 나머지 과제들은 inteld435i로 촬영한 공유 이미지를 사용하였습니다. 이로 인해 서로의 카메라 메트릭스와 왜곡 계수가 달랐었고 이러한 이유 때문에 Detection 과제에 사용한 이미지에 calibration을 적용 하지 못했습니다.
 - 공유한 자료 이미지로 Camera calibration을 수행하고 나머지 Detection과제에 사용한 이미지에도 Calibration 결과를 적용하였습니다.

처음에는 이미지를 HSV 처리 후 S 이미지를 사용하였습니다.
 - S이미지는 특정색을 가진 라인만을 검출하긴 어려웠고 특정색을 가진 라인만 검출하기 위하여 V이미지의 색깔 범위를 지정하여 특정색만 검출하여 사용하였습니다.

HOGDescriptor를 사용하여 사람을 검출하였습니다.
-HOGDescriptor는 사람의 크기와 형태의 영향이 컸고 좀 더 강력한 물체인식(사람인식)이 필요해 YOLOv3 러닝기반 descriptor를 사용하였습니다.

이미지를 복사할 때 얕은 복사를 하여 원본 이미지에 변화가 생겨서 다른 작업을 할 때 방해가 되었습니다
-깊은 복사를 사용하여 각각 별개의 작업을 수행하였습니다.

아쉬운 점

차량의 방향 검출과 사람의 위치인식을 Vanishing Point를 이용하여 구해보려고 시도를 했었으나 구하지 못하고 다른 방법을 사용하게 되었습니다.
Depth 이미지를 이용하여 거리를 측정하려 했으나 자료를 찾기 어려워서 하지 못했습니다.
차가 움직인 각도를 평면상의 좌표로 나타내고 싶었는데 어떻게 해야 할지 구상이 되지 않아 못했습니다.