

# Dynamic Object Tracking



**AVG**enius



**AVG**enius

## ❖ 동적 객체의 움직임 추적(Dynamic Object Tracking)

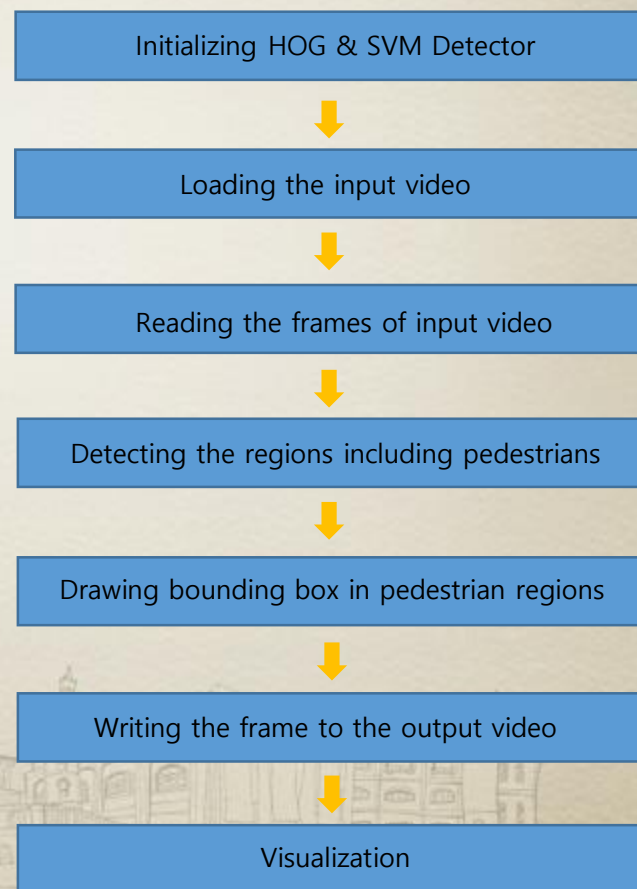
### ➤ 보행자 추적(Pedestrian Tracking)

- HOG(Histogram of Oriented Gradients)를 활용한 보행자 감지
- pedestrian\_tracking.py

```

pedestrian_tracking.py x
1  import cv2
2
3  # Initializing the HOG detector for detecting pedestrians
4  hog = cv2.HOGDescriptor()
5  hog.setSVMDetector(cv2.HOGDescriptor.getDefaultPeopleDetector())
6
7  # Loading the input video
8  cap = cv2.VideoCapture('/home/jinhakim/anaconda3/envs/opencv_lec/Final_Term_Project/pedestrians/pedestrians.mp4')
9
10 # Reading the frames of video
11 width = 600
12 ret, frame = cap.read()
13 height = int(frame.shape[0] * (width / frame.shape[1])) # height = image.shape[0], width = image.shape[1]
14 fps = cap.get(cv2.CAP_PROP_FPS)
15
16 # Setting the output video
17 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
18 output = cv2.VideoWriter('args: pedestrian_tracking.mp4', fourcc, fps, (width, height))
19
20 # Reading the frames of video while the input file is opened
21 while cap.isOpened():
22     ret, image = cap.read()
23     if ret:
24         image_resized = cv2.resize(image, dsize=(width, height))
25
26         # Detecting the regions including pedestrians
27         (region, _) = hog.detectMultiScale(image_resized, winStride=(4, 4), padding=(4, 4), scale=1.05)
28
29         # Drawing bounding boxes in the regions including pedestrians
30         for (x, y, w, h) in region:
31             cv2.rectangle(image_resized, pt1=(x, y), pt2=(x + w, y + h), color=(0, 255, 0), thickness=2)
32
33         # Writing the frame to the output video
34         output.write(image_resized)
35
36         # Visualization of result
37         cv2.imshow('winname: Original Video', cv2.resize(image, dsize=(width, height)))
38         cv2.imshow('winname: Pedestrians Tracking', image_resized)
39         if cv2.waitKey(1) == 27:
40             break
41

```

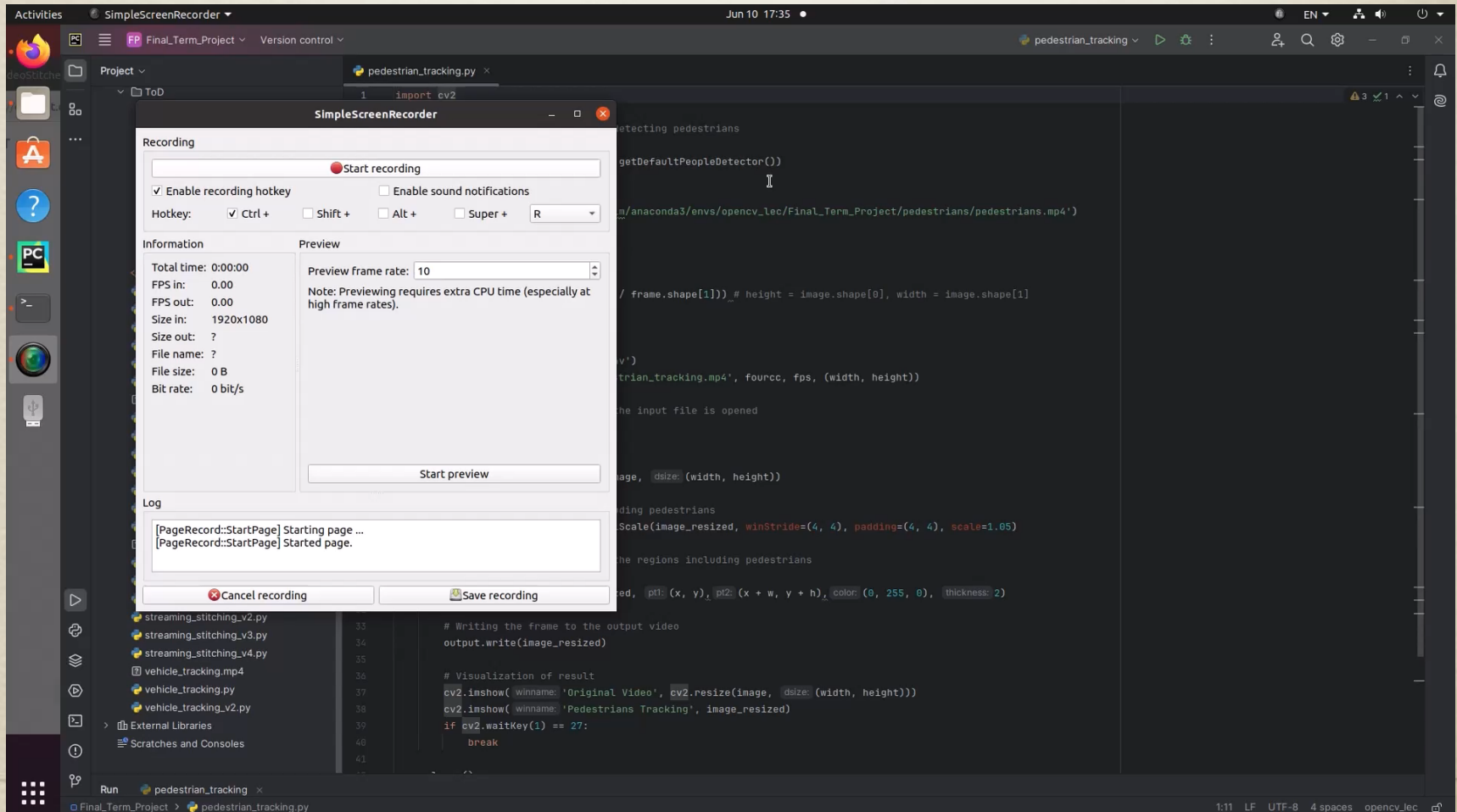


## ❖ 동적 객체의 움직임 추적(Dynamic Object Tracking)

### ➤ 보행자 추적(Pedestrian Tracking)

#### ■ 결과

Reference : <https://github.com/G-Karishni/OpenCV-Python>



## ❖ 동적 객체의 움직임 추적(Dynamic Object Tracking)

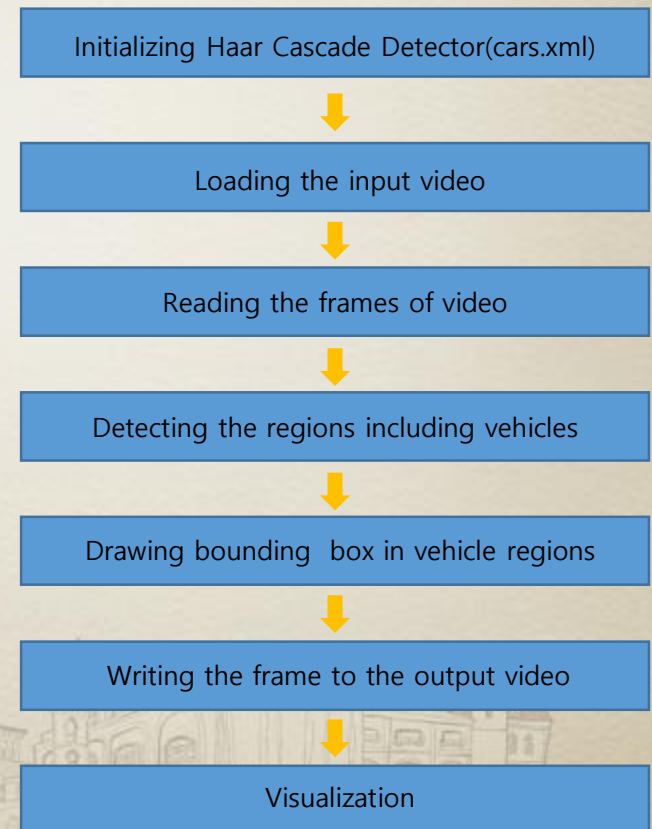
### ➤ 차량 추적(Vehicle Tracking)

- Haar-like feature를 활용한 차량 감지
- vehicle\_tracking.py

```

vehicle_tracking.py x
1 import cv2
2
3 # Initializing the Haar Cascade detector(pre-trained model) for detecting vehicles
4 haar_vehicle_cascade = cv2.CascadeClassifier('cars.xml')
5
6 # cars.xml : https://github.com/andrewsobral/vehicle_detection_haarcascades/blob/master/cars.xml
7
8 # Loading the input video
9 cap = cv2.VideoCapture('/home/jinhakim/anaconda3/envs/opencv lec/Final_Term_Project/dynamic_object/test_07.mp4')
10 width = 800
11 ret, frame = cap.read()
12 height = int(frame.shape[0] * (width / frame.shape[1])) # height = image.shape[0], width = image.shape[1]
13 fps = cap.get(cv2.CAP_PROP_FPS)
14
15 # Setting the output video
16 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
17 output = cv2.VideoWriter('args: vehicle_tracking.mp4', fourcc, fps, (width, height))
18
19 # Reading the frames of video while the input file is opened
20 while cap.isOpened():
21     ret, image = cap.read()
22     if ret:
23         image_resized = cv2.resize(image, dsize=(width, height))
24
25         # Detecting the regions including vehicles
26         gray_image = cv2.cvtColor(image_resized, cv2.COLOR_BGR2GRAY)
27         vehicle_regions = haar_vehicle_cascade.detectMultiScale(gray_image, scaleFactor=1.2, minNeighbors=5)
28
29         # Drawing bounding boxes in the regions including vehicles
30         for (x, y, w, h) in vehicle_regions:
31             cv2.rectangle(image_resized, pt1=(x, y), pt2=(x + w, y + h), color=(0, 0, 255), thickness=2)
32
33         # Writing the frame to the output video
34         output.write(image_resized)
35
36         # Visualization of result
37         cv2.imshow( winname: 'Original Video', cv2.resize(image, dsize=(width, height)))
38         cv2.imshow( winname: 'Dynamic Objects Tracking', image_resized)
39         if cv2.waitKey(1) == 27:
40             break
41

```

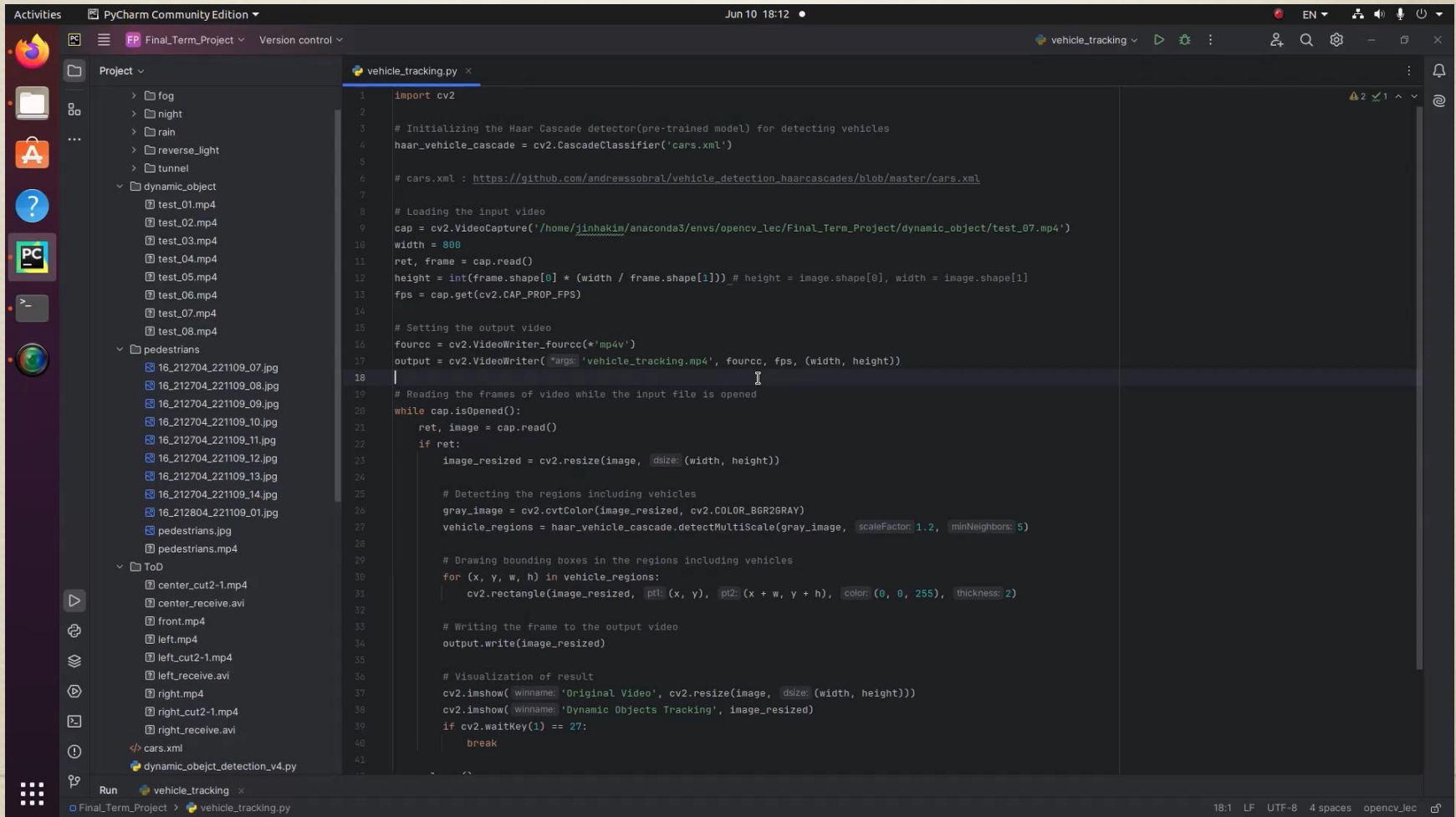




## ❖ 동적 객체의 움직임 추적(Dynamic Object Tracking)

### ➤ 차량 추적(Vehicle Tracking)

#### ■ 결과



- ❖ 동적 객체의 움직임 추적(Dynamic Object Tracking)
  - 차량 추적(Vehicle Tracking / Waymo Open Dataset)
    - 결과

The screenshot displays the PyCharm Community Edition interface. The main editor window shows a Python script named `vehicle_tracking.py` with the following code:

```

1 import cv2
2
3 # Initializing the Haar Cascade detector(pre-trained model) for detecting vehicles
4 haar_vehicle_cascade = cv2.CascadeClassifier('cars.xml')
5
6 # cars.xml : https://github.com/andrewssohral/vehicle_detection_haarcascades/blob/master/cars.xml
7
8 # Loading the input video
9 cap = cv2.VideoCapture('/home/jinhakin/anaconda3/envs/opencv lec/Final_Term_Project/dynamic_object/videos_train_00005.mp4')
10 width = 800
11 ret, frame = cap.read()
12 height = int(frame.shape[0] * (width / frame.shape[1])) # height = image.shape[0], width = image.shape[1]
13 fps = cap.get(cv2.CAP_PROP_FPS)
14
15 # Setting the output video
16 fourcc = cv2.VideoWriter_fourcc(*'mp4v')
17 output = cv2.VideoWriter('args:vehicle_tracking.mp4', fourcc, fps, (width, height))
18
19 # Reading the frames of video while the input file is opened
20 while cap.isOpened():
21     ret, image = cap.read()
22     if ret:
23         image_resized = cv2.resize(image, dszie: (width, height))
24
25         # Detecting the regions including vehicles
26         gray_image = cv2.cvtColor(image_resized, cv2.COLOR_BGR2GRAY)
27         vehicle_regions = haar_vehicle_cascade.detectMultiScale(gray_image, scaleFactor: 1.3, (minNeighbors: 3))
28
29         # Drawing bounding boxes in the regions including vehicles
30         for (x, y, w, h) in vehicle_regions:

```

The bottom panel shows the Run output window with the following text:

```

/home/jinhakin/anaconda3/envs/opencv lec/bin/python /home/jinhakin/anaconda3/envs/opencv lec/Final_Term_Project/vehicle_tracking.py
Traceback (most recent call last):
  File "/home/jinhakin/anaconda3/envs/opencv lec/Final_Term_Project/vehicle_tracking.py", line 21, in <module>
    ret, image = cap.read()
KeyboardInterrupt
Process finished with exit code 1

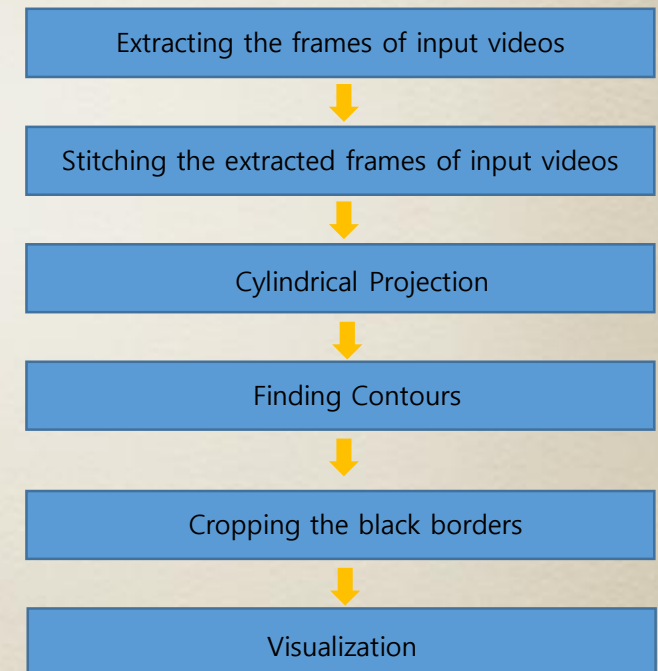
```

## ❖ 영상 스티칭 (Video Stitching)

## ➤ 파노라마 영상 생성(Panorama Video Creation)

## ▪ panorama\_stitching\_final.py

```
panorama_stitching_final.py x
1 import cv2
2 import numpy as np
3
4 # Cylindrical Projection
5 usage
6 def cylindrical_projection(image, f):
7     h, w = image.shape[:2]
8     K = np.array([[f, 0, w / 2], [0, f, h / 2], [0, 0, 1]]) # Camera intrinsic parameter matrix
9
10    # Creating cylindrical coordinates
11    cylinder = np.zeros_like(image)
12    cyl_x, cyl_y = np.meshgrid(*xi: np.arange(w), np.arange(h))
13    cyl_x = cyl_x - w / 2
14    cyl_y = cyl_y - h / 2
15    theta = np.arctan(cyl_x / f)
16    h_ = np.sqrt(cyl_x ** 2 + f ** 2)
17    y_ = cyl_y * f / h_
18    x_ = f * np.tan(cyl_x / f)
19
20    xmap = x_ + w / 2
21    ymap = y_ + h / 2
22
23    # Performing cylindrical projection using remapping
24    cylinder = cv2.remap(image, xmap.astype(np.float32), ymap.astype(np.float32), cv2.INTER_LINEAR)
25
26    return cylinder
27
28 # Cropping the contours(black borders) of panorama video
29 usage
30 def crop_black_borders(image):
31     # Converting to grayscale
32     gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
33
34     # Detecting non-black pixels
35     _, thresh = cv2.threshold(gray, thresh: 1, maxval: 255, cv2.THRESH_BINARY)
36
37     # Finding contours of the non-black regions
38     contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
39
40     # Finding the bounding box of the largest contour
41     x, y, w, h = cv2.boundingRect(contours[0])
```

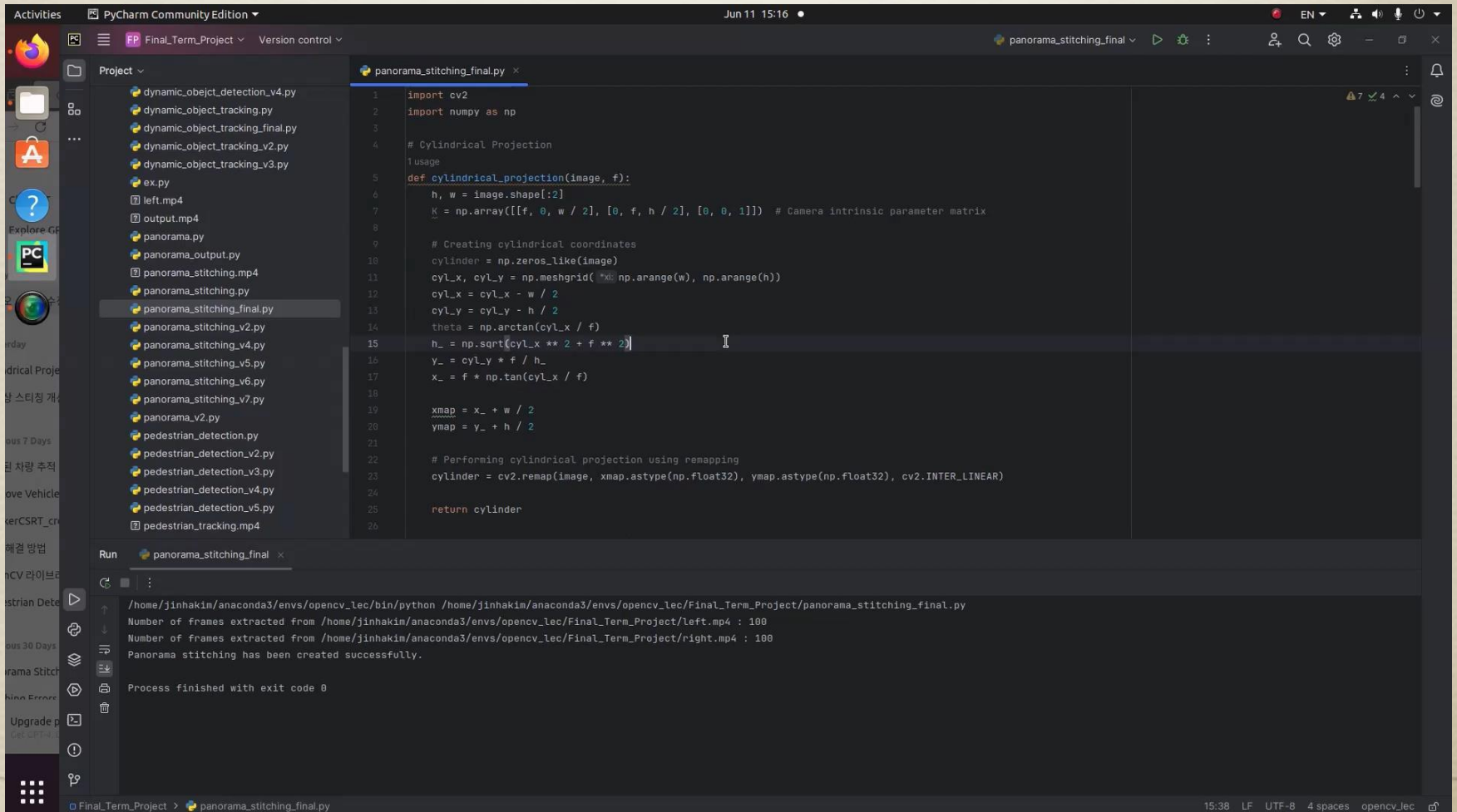


## ❖ 영상 스티칭 (Video Stitching)

## ➤ 파노라마 영상 생성(Panorama Video Creation)

## ■ 결과

Reference : <https://github.com/SuTanTank/VideoStitchingViaShakinessRemoving>



The screenshot displays the PyCharm Community Edition interface. The left sidebar shows the project structure with files like `dynamic_object_tracking.py`, `ex.py`, and `panorama_stitching.py`. The main editor window shows the code for `panorama_stitching_final.py`, which includes imports for `cv2` and `numpy`, and a function `cylindrical_projection` that takes an image and a focal length `f` as input. The function calculates cylindrical coordinates and performs a remapping of the image. The bottom panel shows the output of the script, indicating that the panorama stitching was successful.

```
1 import cv2
2 import numpy as np
3
4 # Cylindrical Projection
5 def cylindrical_projection(image, f):
6     h, w = image.shape[:2]
7     K = np.array([[f, 0, w / 2], [0, f, h / 2], [0, 0, 1]]) # Camera intrinsic parameter matrix
8
9     # Creating cylindrical coordinates
10    cylinder = np.zeros_like(image)
11    cyl_x, cyl_y = np.meshgrid(np.arange(w), np.arange(h))
12    cyl_x = cyl_x - w / 2
13    cyl_y = cyl_y - h / 2
14    theta = np.arctan(cyl_x / f)
15    h_ = np.sqrt(cyl_x ** 2 + f ** 2)
16    y_ = cyl_y * f / h_
17    x_ = f * np.tan(cyl_x / f)
18
19    xmap = x_ + w / 2
20    ymap = y_ + h / 2
21
22    # Performing cylindrical projection using remapping
23    cylinder = cv2.remap(image, xmap.astype(np.float32), ymap.astype(np.float32), cv2.INTER_LINEAR)
24
25    return cylinder
```

Run panorama\_stitching\_final x

```
/home/jinhakim/anaconda3/envs/opencv lec/bin/python /home/jinhakim/anaconda3/envs/opencv lec/Final_Term_Project/panorama_stitching_final.py
Number of Frames extracted from /home/jinhakim/anaconda3/envs/opencv lec/Final_Term_Project/left.mp4 : 100
Number of frames extracted from /home/jinhakim/anaconda3/envs/opencv lec/Final_Term_Project/right.mp4 : 100
Panorama stitching has been created successfully.

Process finished with exit code 0
```



## ❖ 프로젝트 결과 고찰

### ■ 동적 객체의 움직임 추적(Dynamic Object Tracking)

- 보행자, 차량의 동적 객체에 대한 tracking은 적절히 수행되는 것을 확인
- Future work : 보행자, 차량이 혼합된 환경에서 동적 객체 동시 추적 방법 연구

### ■ 영상 스티칭(Video Stitching)

- 정합한 파노라마 영상의 검은색 테두리(black border)를 없애기 위해 cylindrical projection, remapping, cropping의 방법을 사용하였지만 방법 적용 전과 큰 차이가 없었음.
- Camera intrinsic parameter, focal length 등 최적 파라미터 조합 탐색
- Future work : 차량의 left, front, right 3가지 영상에 대한 streaming 스티칭 및 파노라마 구현 연구



# THANK YOU



**AVG**enius



**AVG**enius