

Redirected Walking

230728 19101188 고은수

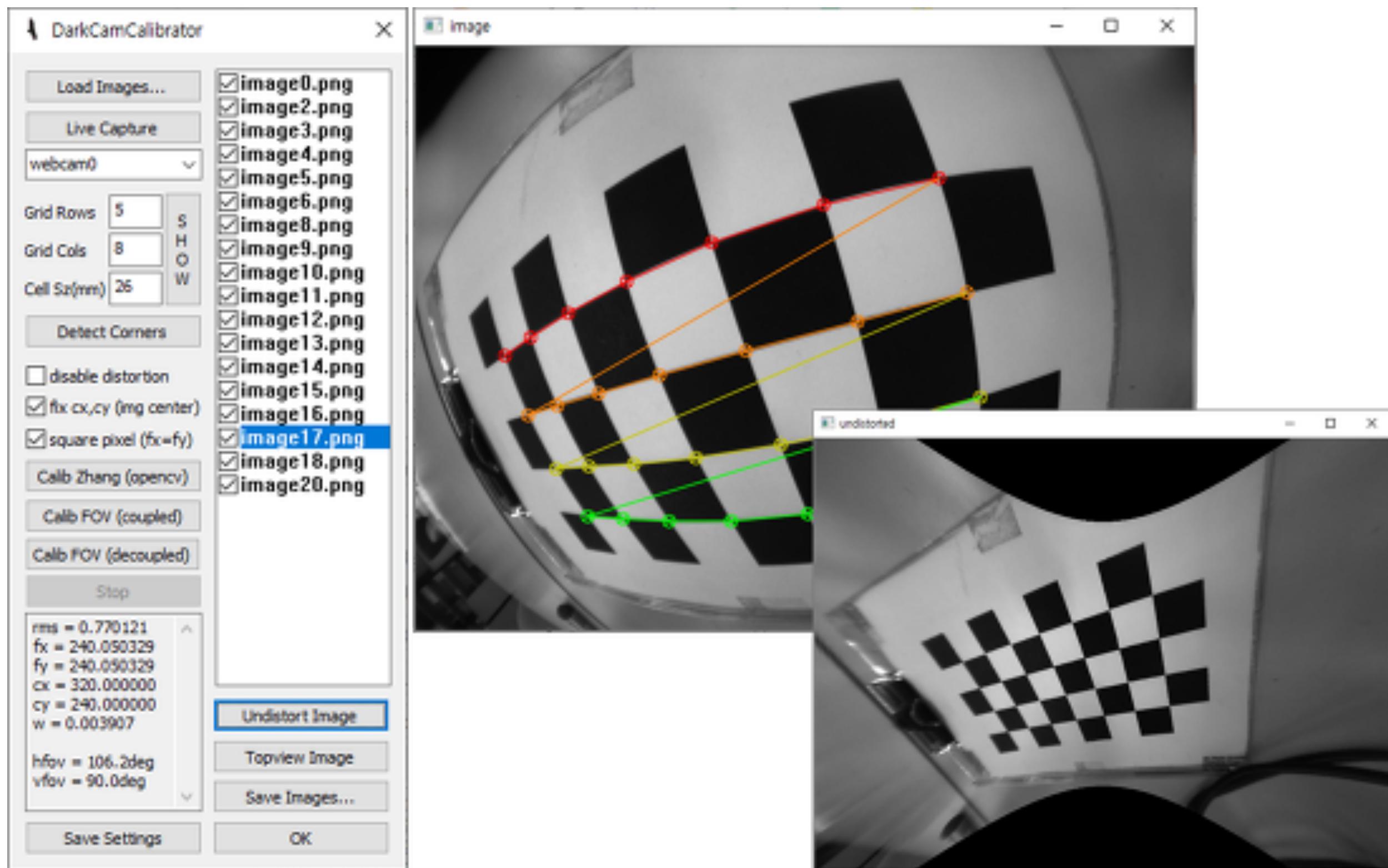
Todo

- 두 영상의 왜곡 보정 후 스티칭
- 발 좌표로 위치 찾기
- 영상을 언리얼엔진으로 시뮬레이션

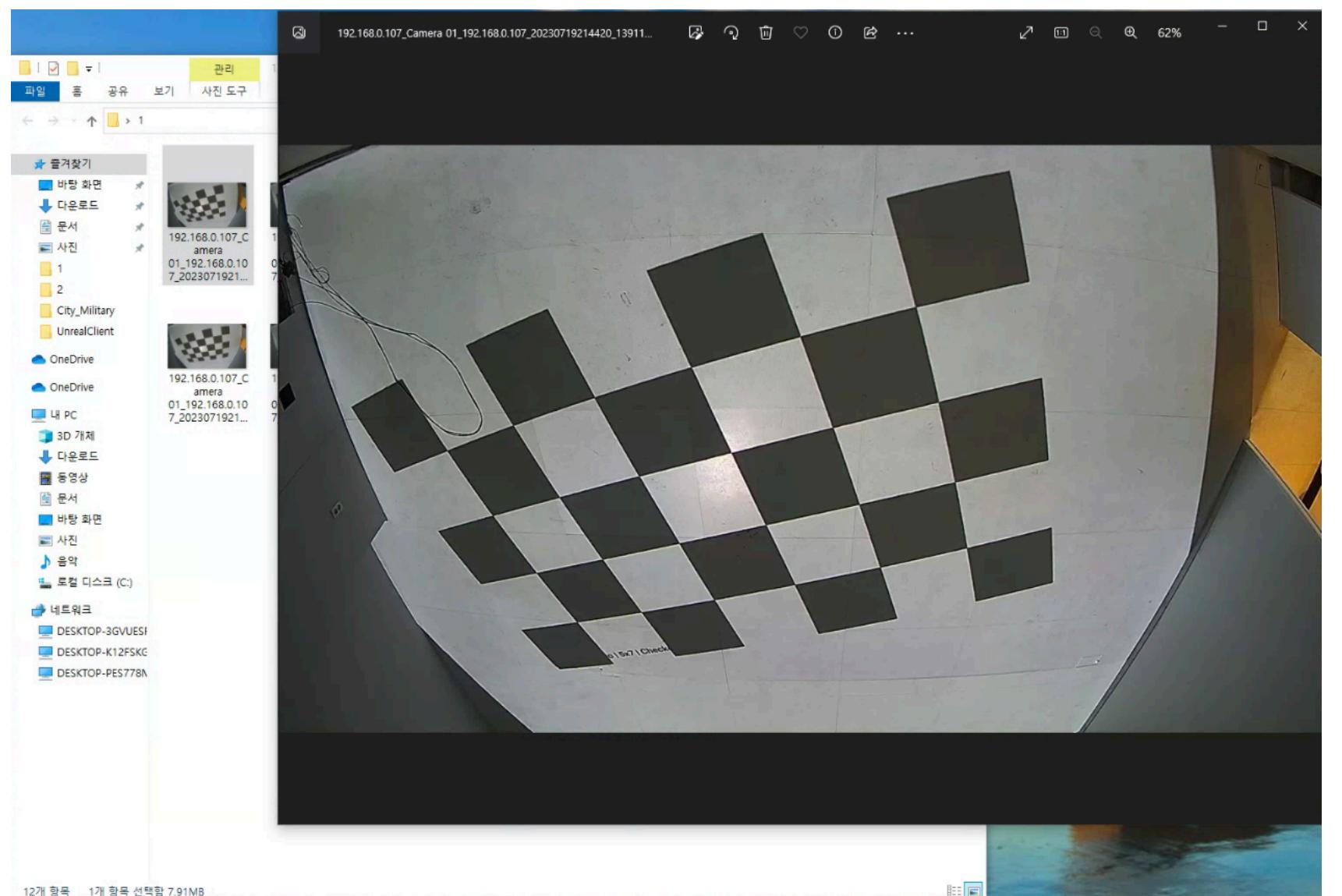
두 영상의 왜곡 보정 후 스티칭

1. 카메라 캘리브레이션 및 왜곡 보정
2. 수직으로 내려다보는 시점으로 와핑
3. 영상 스티칭

카메라 캘리브레이션 및 왜곡 보정



카메라 캘리브레이션 프로그램 이용하여 parameter들을 구합니다



CAVE에서 얻은 image

카메라 캘리브레이션 및 왜곡 보정

```
5     def undistort_image(img, fx, fy, cx, cy, k1, k2, p1, p2):
6         h, w = img.shape[:2]
7
8             # 카메라 매트릭스 (intrinsic matrix)
9             mtx = np.array([[fx, 0, cx],
10                             [0, fy, cy],
11                             [0, 0, 1]])
12
13             # 왜곡 계수 (distortion coefficients)
14             dist = np.array([k1, k2, p1, p2, 0])
15
16             # 보정된 이미지 생성
17             newCameraMtx, roi = cv2.getOptimalNewCameraMatrix(mtx, dist, (w, h), 1, (w, h))
18             undistorted_img = cv2.undistort(img, mtx, dist, None, newCameraMtx)
19
20         return undistorted_img
```

cam1(문방향)

rms = 0.747896
fx = 1138.992081
fy = 1138.992081
cx = 960.000000
cy = 540.000000
k1 = -0.354276
k2 = 0.141542
p1 = 0.016332
p2 = 0.001623

hfov = 80.3
vfov = 50.7

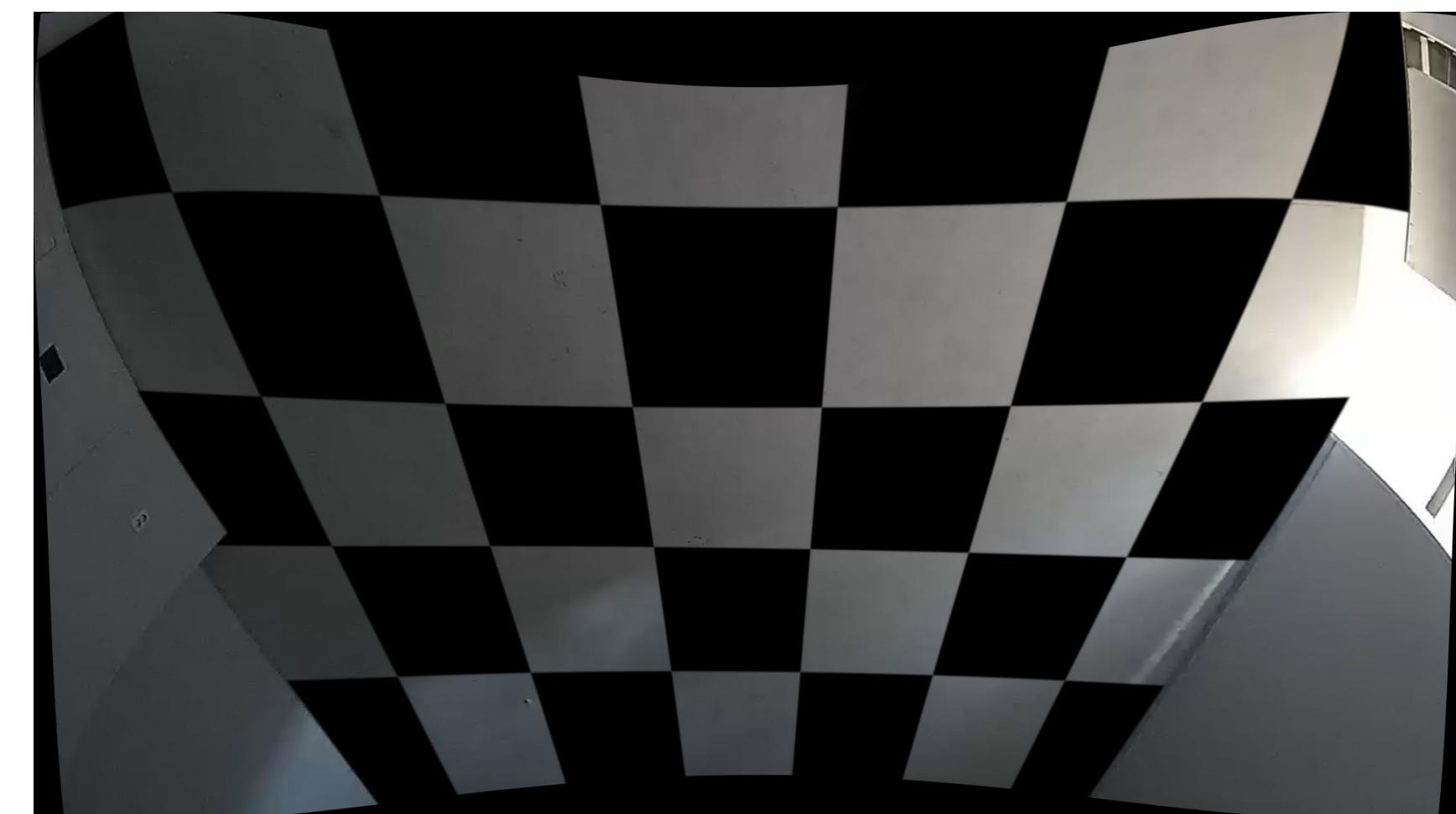
cam2(안쪽)

rms = 0.892450
fx = 1159.350155
fy = 1159.350155
cx = 960.000000
cy = 540.000000
k1 = -0.365702
k2 = 0.159884
p1 = -0.013554
p2 = 0.005254

hfov = 79.3
vfov = 50.0



원본 image



Undistorted image

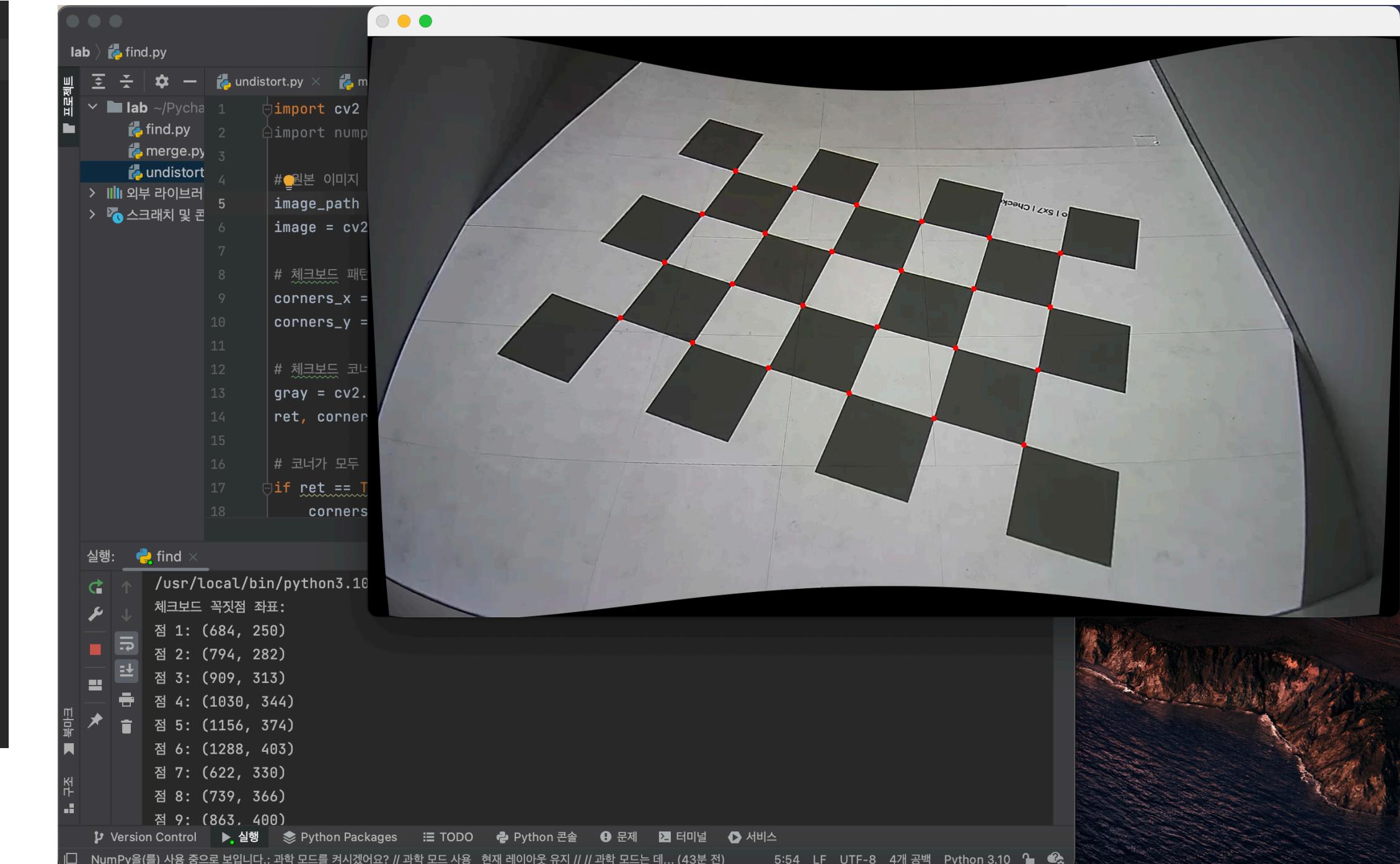
수직으로 내려다보는 시점으로 와핑

```
23     def warp_image(input_image, src_points, dst_points):
24         # 원근 변환 행렬 계산
25         warp_matrix = cv2.getPerspectiveTransform(src_points, dst_points)
26
27         # 원근 변환 적용
28         output_image = cv2.warpPerspective(input_image, warp_matrix, (input_image.shape[1], input_image.shape[0]))
29
30     return output_image
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56     # 원근 변환에 사용할 좌표 설정 (원본 이미지의 4개의 좌표)
57     src_points = np.array([[412, 328], [554, 301], [562, 396], [406, 429]], dtype=np.float32)
58
59     # 원하는 변형을 지정하여 dst_points를 설정하세요.
60     dst_points = np.array([[402, 531], [524, 491], [563, 615], [439, 655]], dtype=np.float32)
61
62     # 이미지 와핑
63     warped_img = warp_image(undistorted_img, src_points, dst_points)
```

원본 이미지와 카메라 캘리브레이션 프로그램으로 얻은 topview 시점의 이미지간의 좌표 차이를 이용하여 와핑합니다

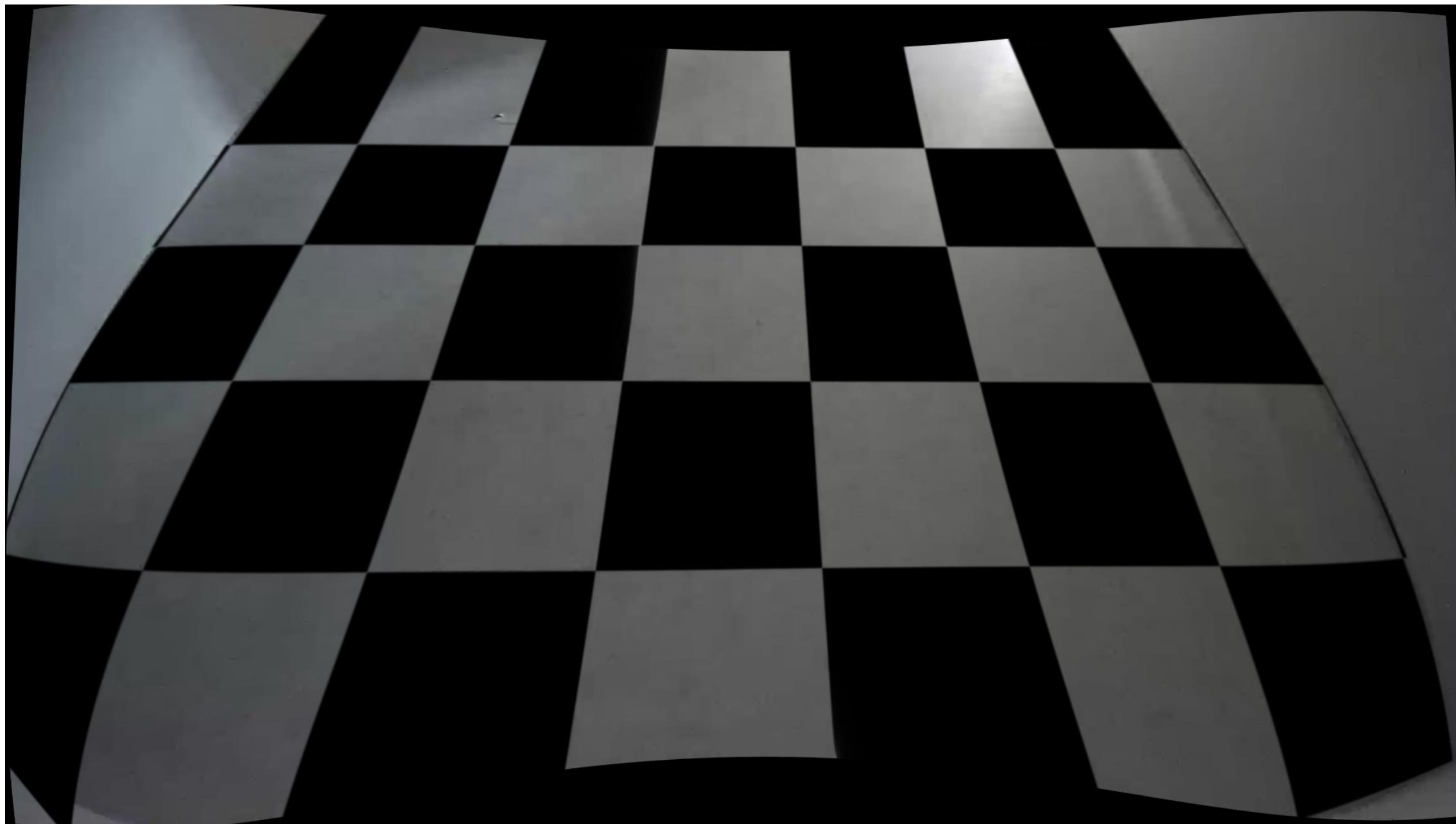
수직으로 내려다보는 시점으로 와핑

```
12 # 체크보드 코너 검출
13 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
14 ret, corners = cv2.findChessboardCorners(gray, (corners_x, corners_y), None)
15
16 # 코너가 모두 검출되면 좌표 출력 및 점 표시
17 if ret == True:
18     corners = corners.reshape(-1, 2) # (48, 1, 2) -> (48, 2)
19
20     print("체크보드 꼭짓점 좌표:")
21     for i, corner in enumerate(corners):
22         x, y = int(corner[0]), int(corner[1])
23         print(f"점 {i+1}: ({x}, {y})")
24         cv2.circle(image, (x, y), 5, (0, 0, 255), -1) # 빨간색 원으로 표시
25
26 # 이미지에 점들을 표시한 결과를 새 창에 출력
27 cv2.imshow("체크보드 점 표시 이미지", image)
28 cv2.waitKey(0)
29 cv2.destroyAllWindows()
```

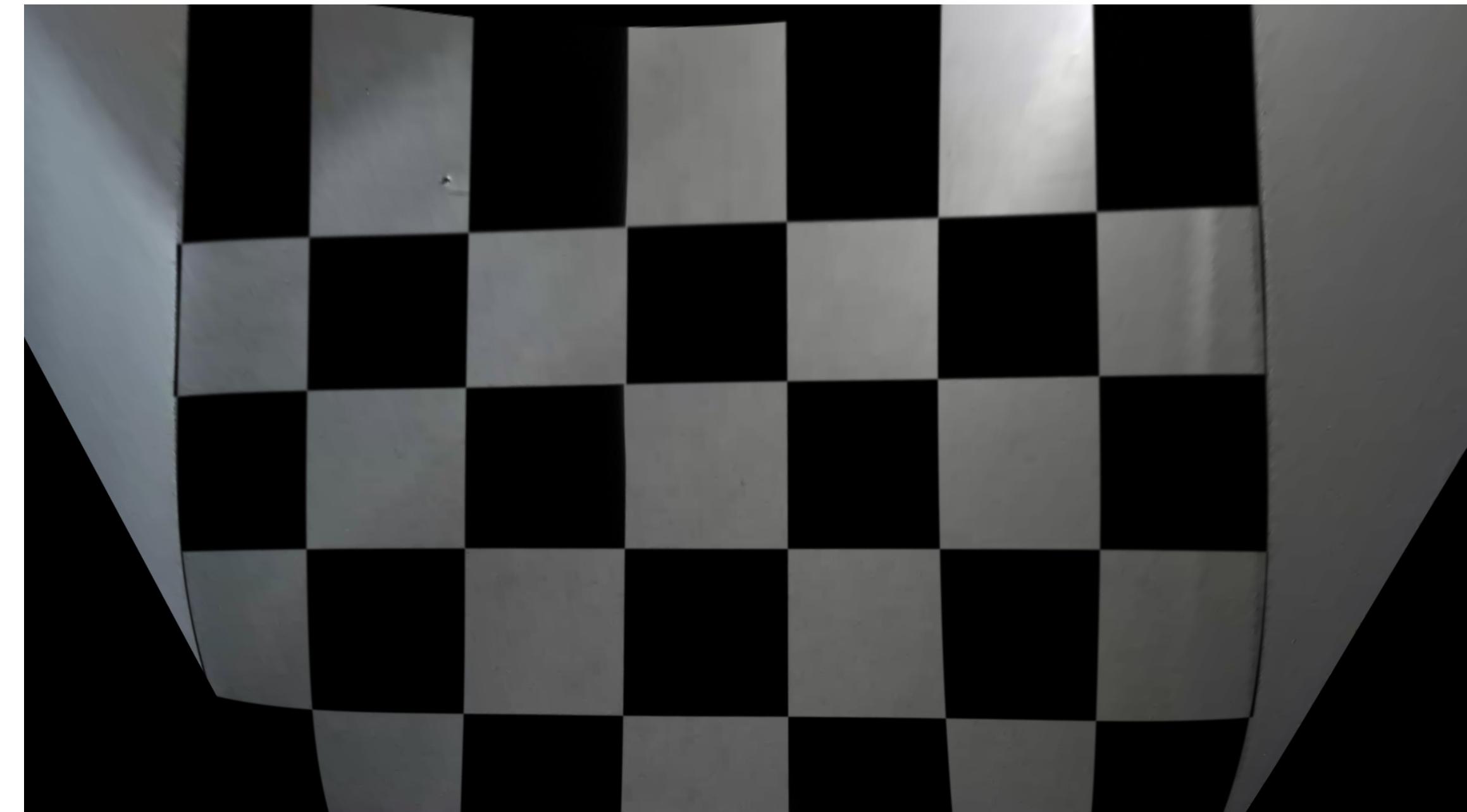


체크보드의 꼭짓점 좌표를 이용하여 원근 변환에 사용되는 점들을 구합니다

수직으로 내려다보는 시점으로 와핑



Undistorted image



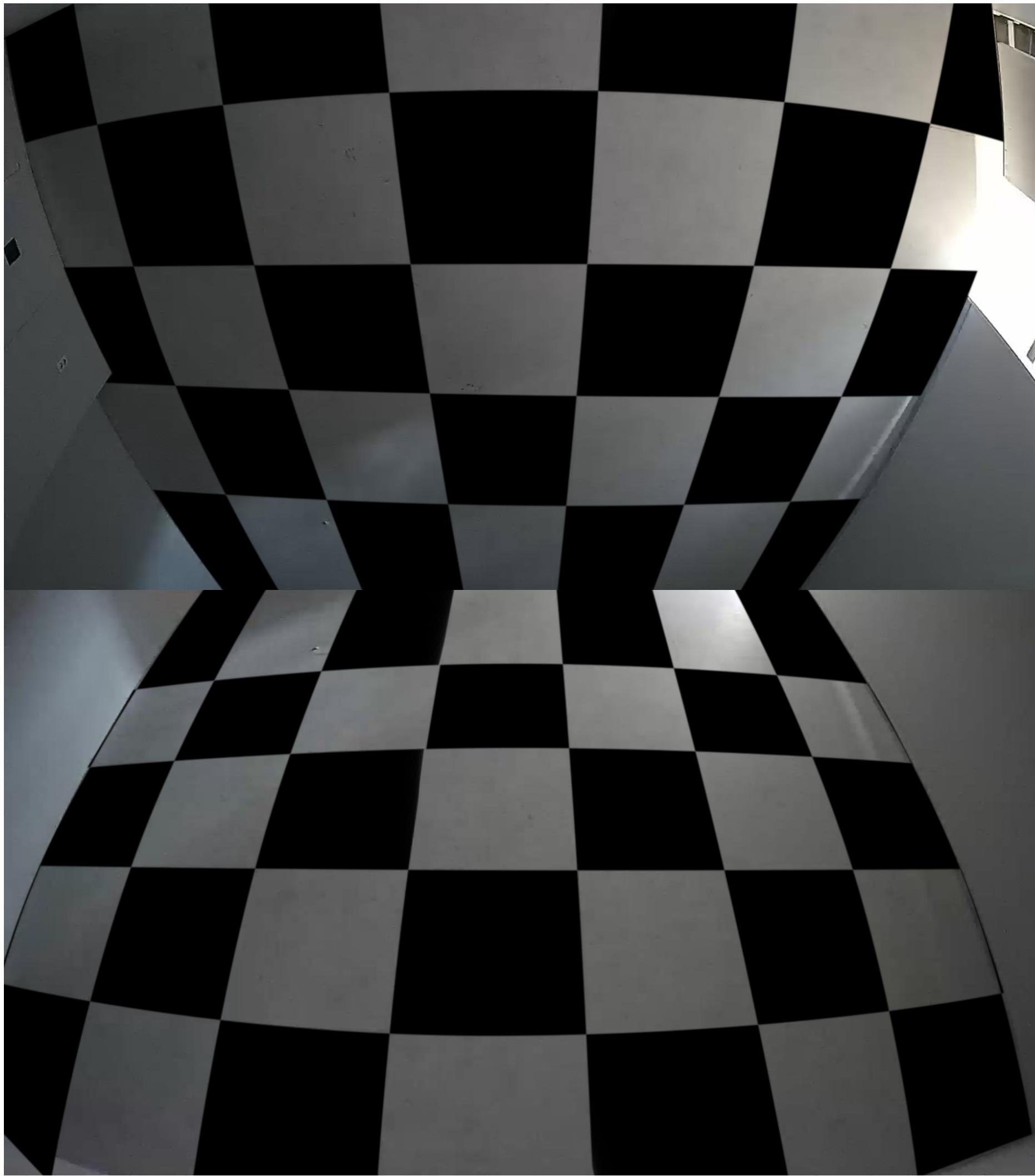
Topview

영상 스티칭

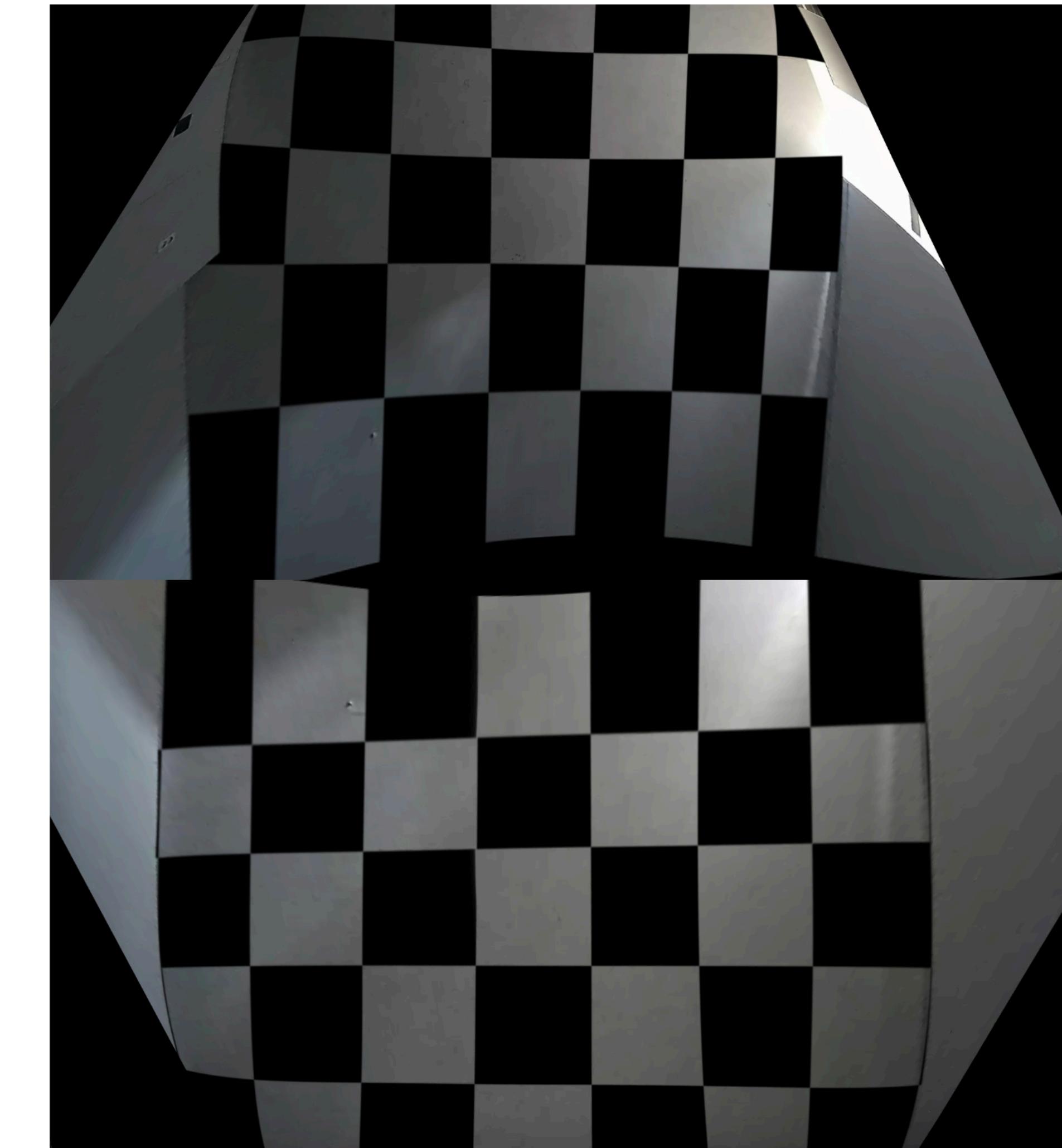
```
11  # 영상 매칭을 위해 특징점 검출과 매칭을 수행합니다.
12  orb = cv2.ORB_create()
13  kp1, des1 = orb.detectAndCompute(image1, None)
14  kp2, des2 = orb.detectAndCompute(image2, None)
15
16  bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
17  matches = bf.match(des1, des2)
18  matches = sorted(matches, key=lambda x: x.distance)
19
20  # 좋은 매칭점들만 선택합니다.
21  good_matches = matches[:50]
22
23  # 좋은 매칭점들을 기준으로 변환 행렬을 찾습니다.
24  src_pts = np.float32([kp1[m.queryIdx].pt for m in good_matches]).reshape(-1, 1, 2)
25  dst_pts = np.float32([kp2[m.trainIdx].pt for m in good_matches]).reshape(-1, 1, 2)
26
27  M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
28
29  # 이미지 합치기
30  h1, w1 = image1.shape[:2]
31  h2, w2 = image2.shape[:2]
32  result = cv2.warpPerspective(image1, M, (max(w1, w2), h1 + h2))
33  result[0:h1, 0:w1] = image1
34  result[h1:h1+h2, 0:w2] = image2
```

특징점 검출 및 매칭 후 변환행렬을 찾아 이미지 스티칭

영상 스티칭



원본



왜곡 보정 후 topview 시점으로 합친 이미지

영상 스티칭

```
h1, w1 = image1.shape[:2]
top_cut_percentage = 0.3
cut_height1 = int(h1 * top_cut_percentage)
cropped_image1 = image1[:h1 - cut_height1, :]

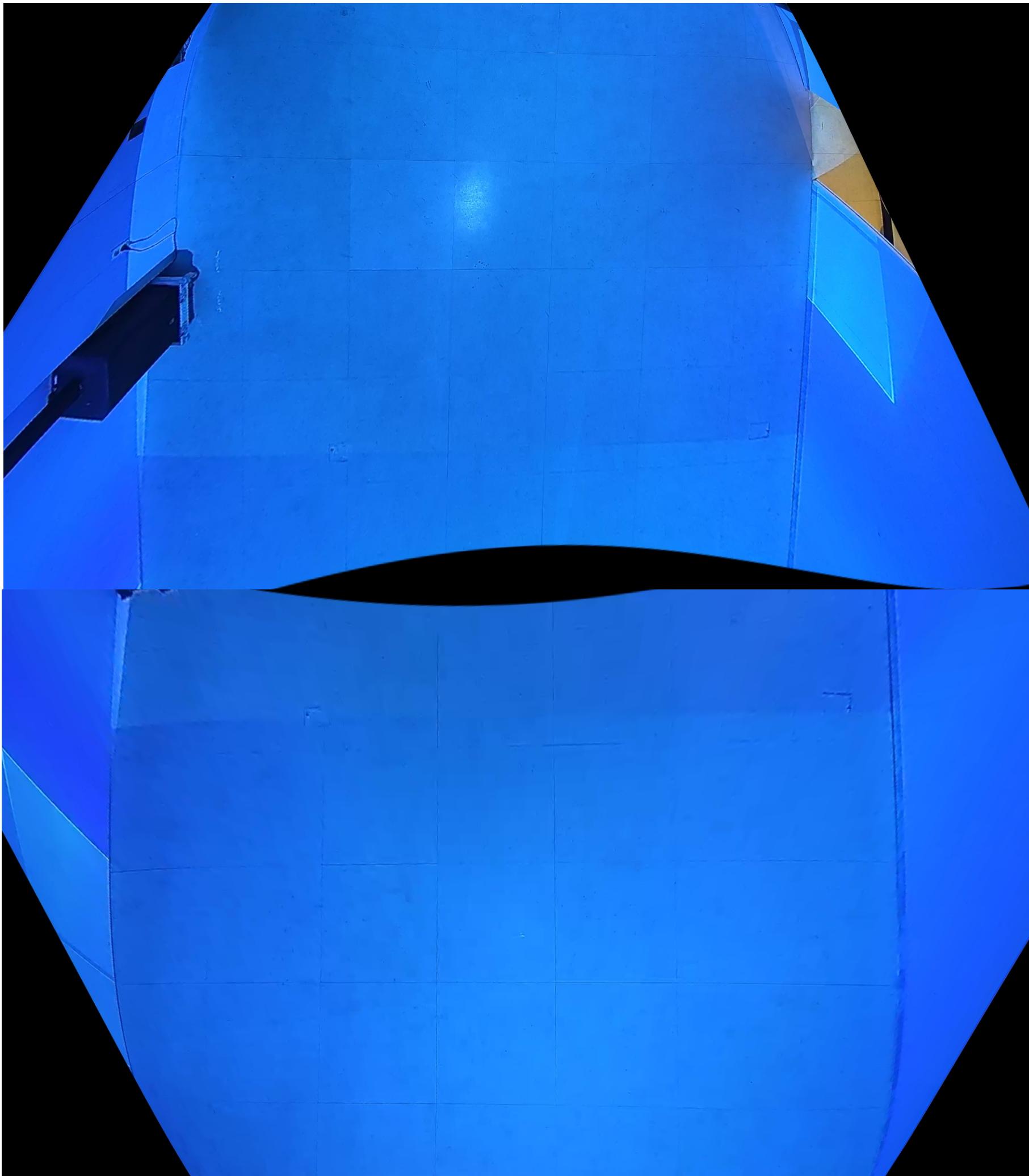
h2, w2 = image2.shape[:2]
bottom_cut_percentage = 0.1
cut_height2 = int(h2 * bottom_cut_percentage)
cropped_image2 = image2[cut_height2:, :]

scale_percent = 85
width = int(cropped_image2.shape[1] * scale_percent / 100)
height = int(cropped_image2.shape[0] * scale_percent / 100)
dim = (width, height)
resized_image2 = cv2.resize(cropped_image2, dim, interpolation=cv2.INTER_AREA)

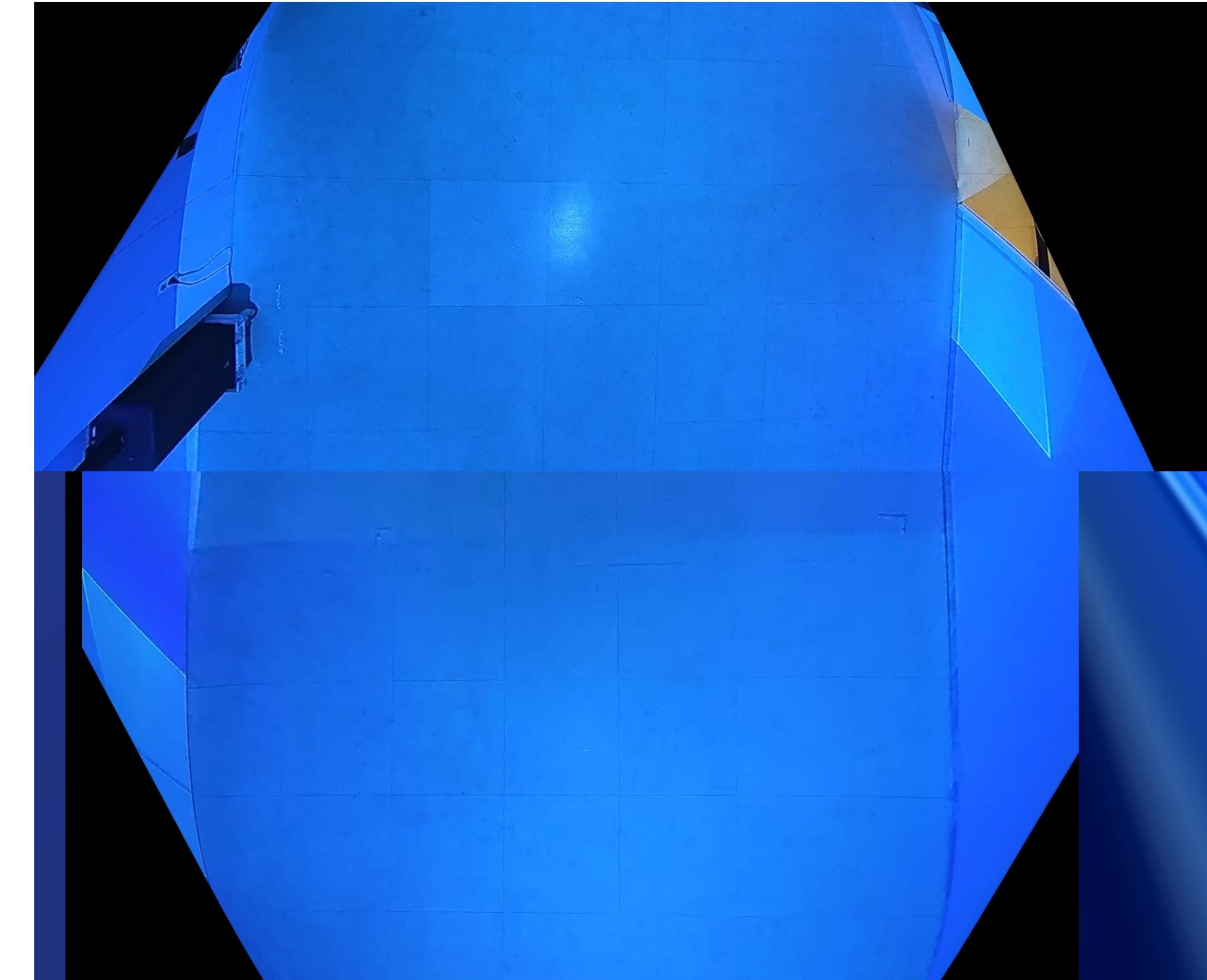
M_translation = np.float32([[1, 0, 27], [0, 1, 0]])
translated_image2 = cv2.warpAffine(resized_image2, M_translation, (width + 50, height))
```

겹치는 부분 잘라내고 두 이미지가 겹치도록 위치 조정

영상 스티칭



원본



크기 및 위치 조정

나쁘진 않지만 겹치는 부분 정확하게 파악 후 조정 필요

Bot-Sort에 적용

발 위치로 좌표 찾기

시뮬레이션

감사합니다