Projective Geometry in Computer Vision

Dmitry Ryabokon, github.com/dryabokon

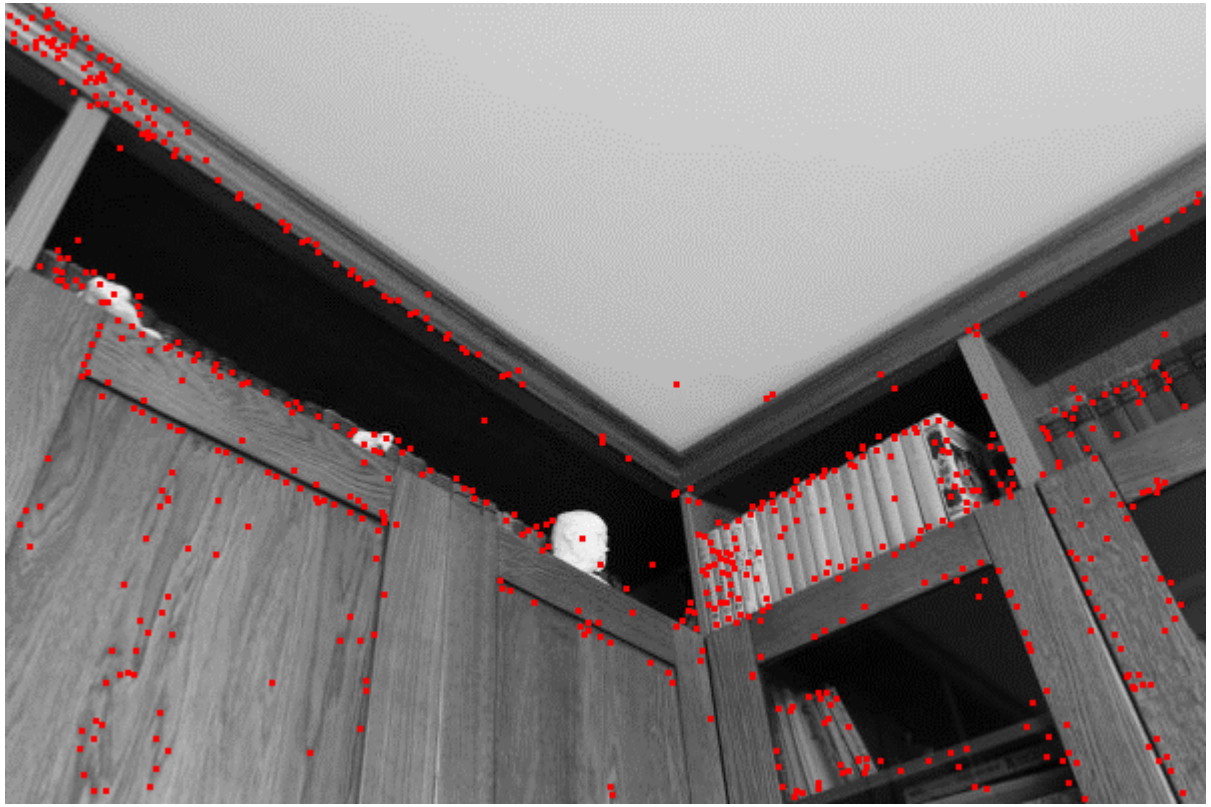
Content

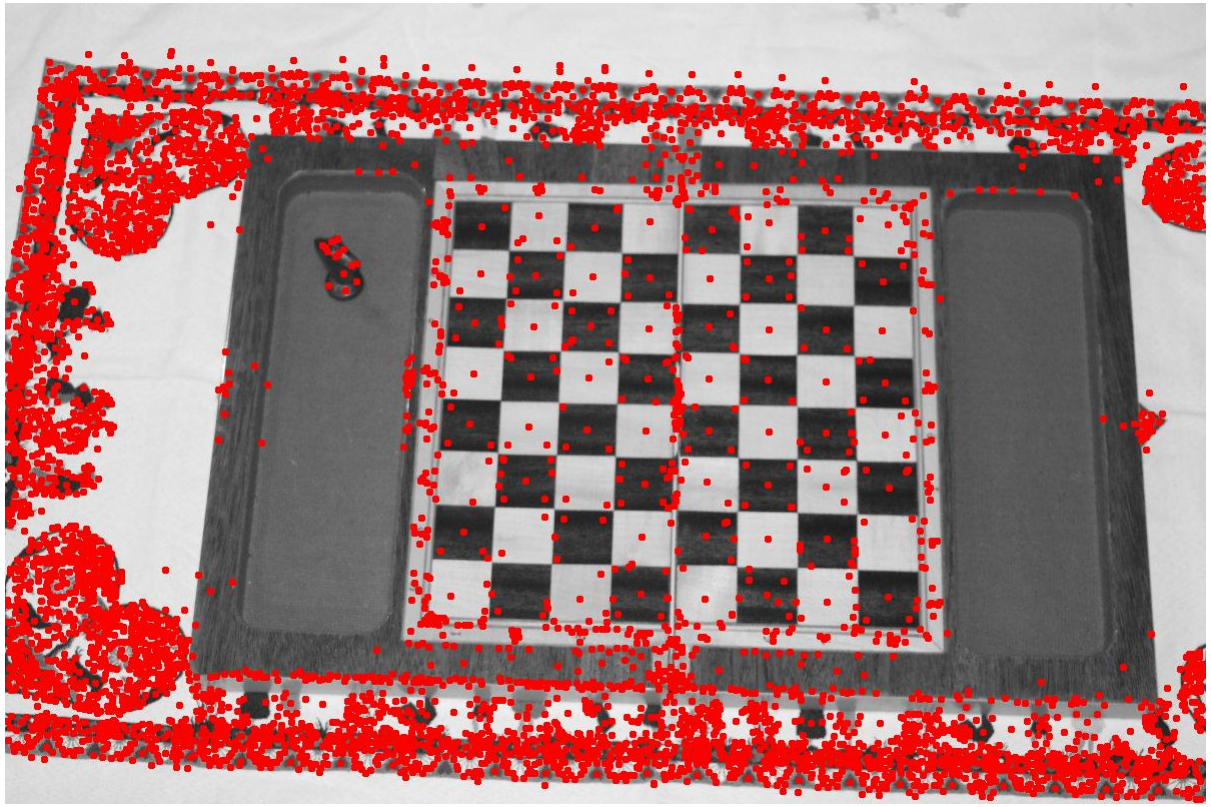
1. KeyPoints detection
2. Matching the KeyPoints
3. Homography
4. Blending
5. Panorama
6. Camera calibration
7. Stereopair rectification
8. Template matching
9. Stereo Vision
10. Pose estimation
11. Aruco
12. Augmented reality

1. KeyPoints detection

KeyPoints detection

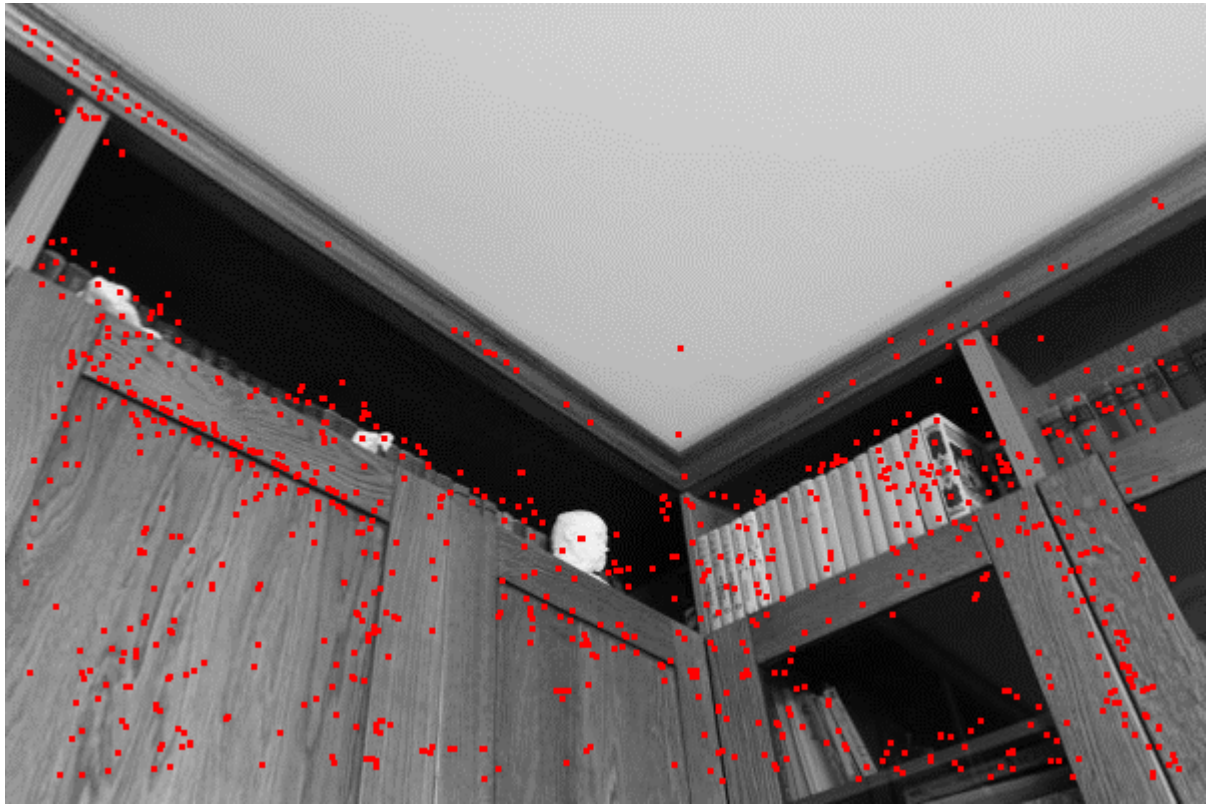
SIFT: Scale-Invariant Feature Transform

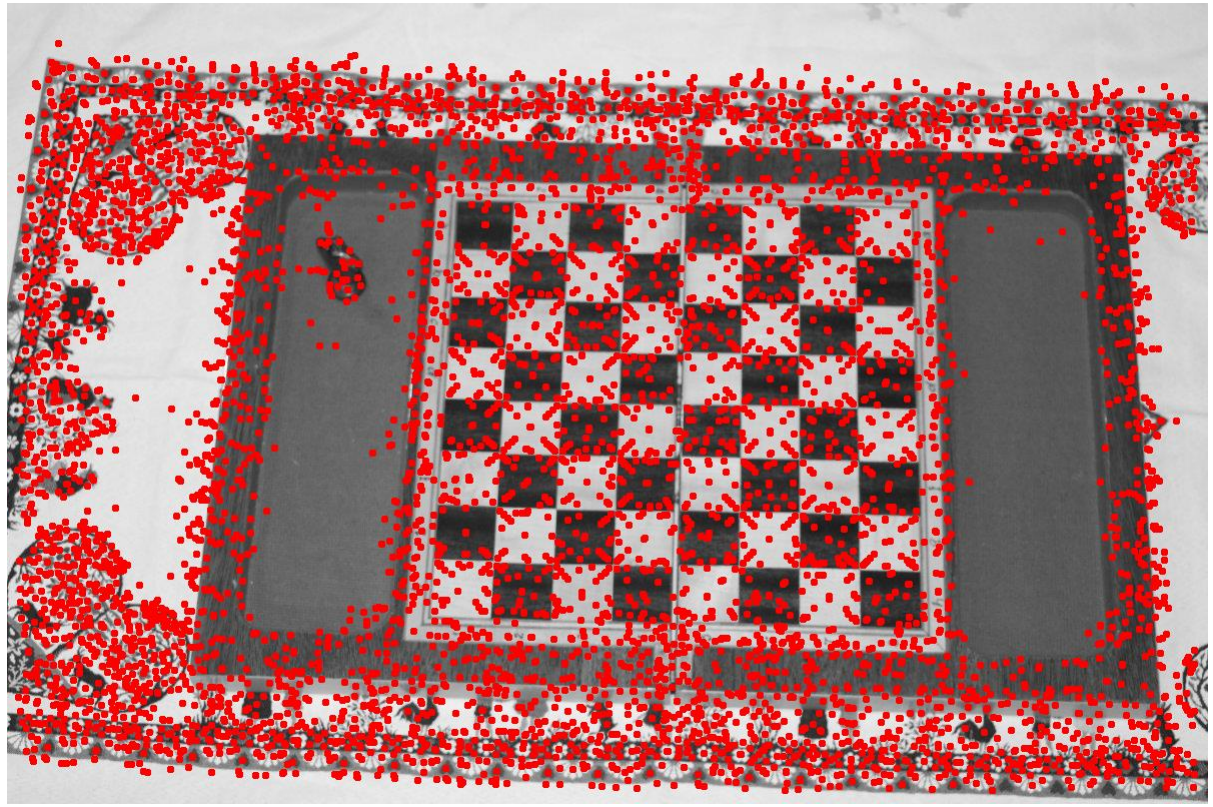




KeyPoints detection

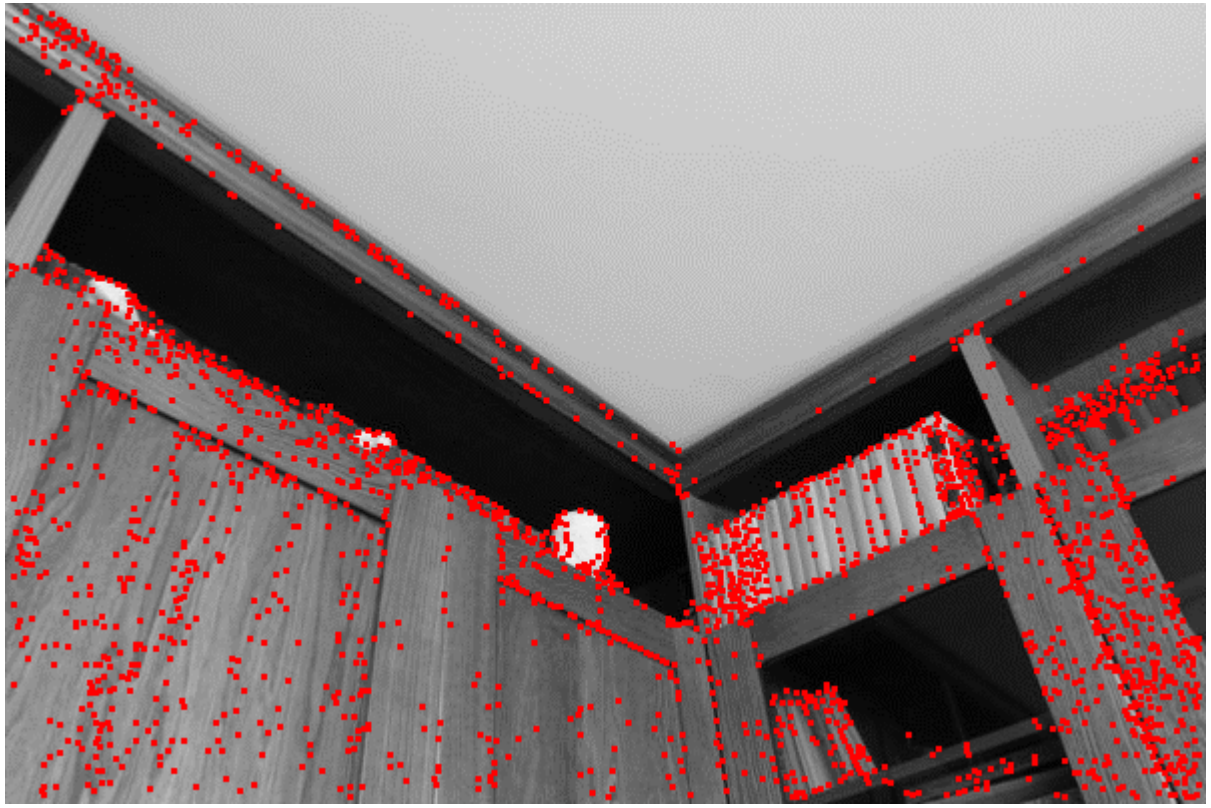
SURF: Speeded-Up Robust Features

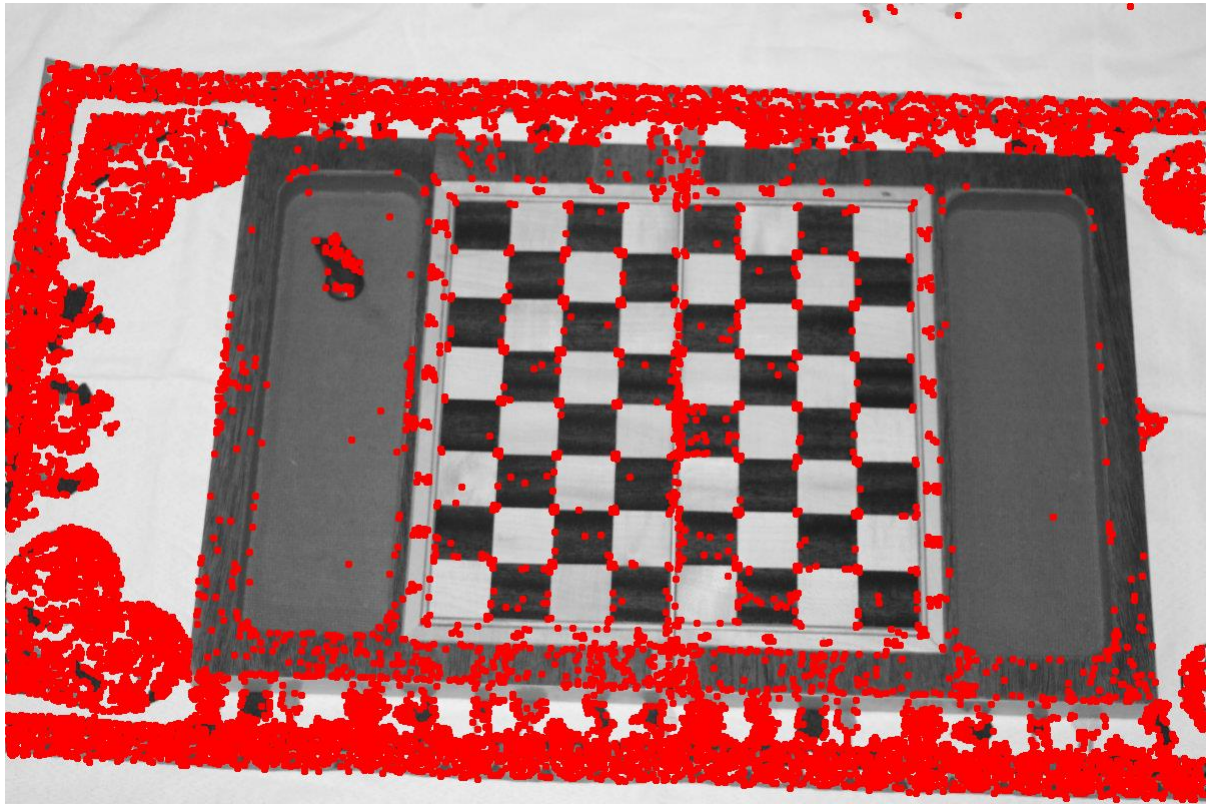




KeyPoints detection

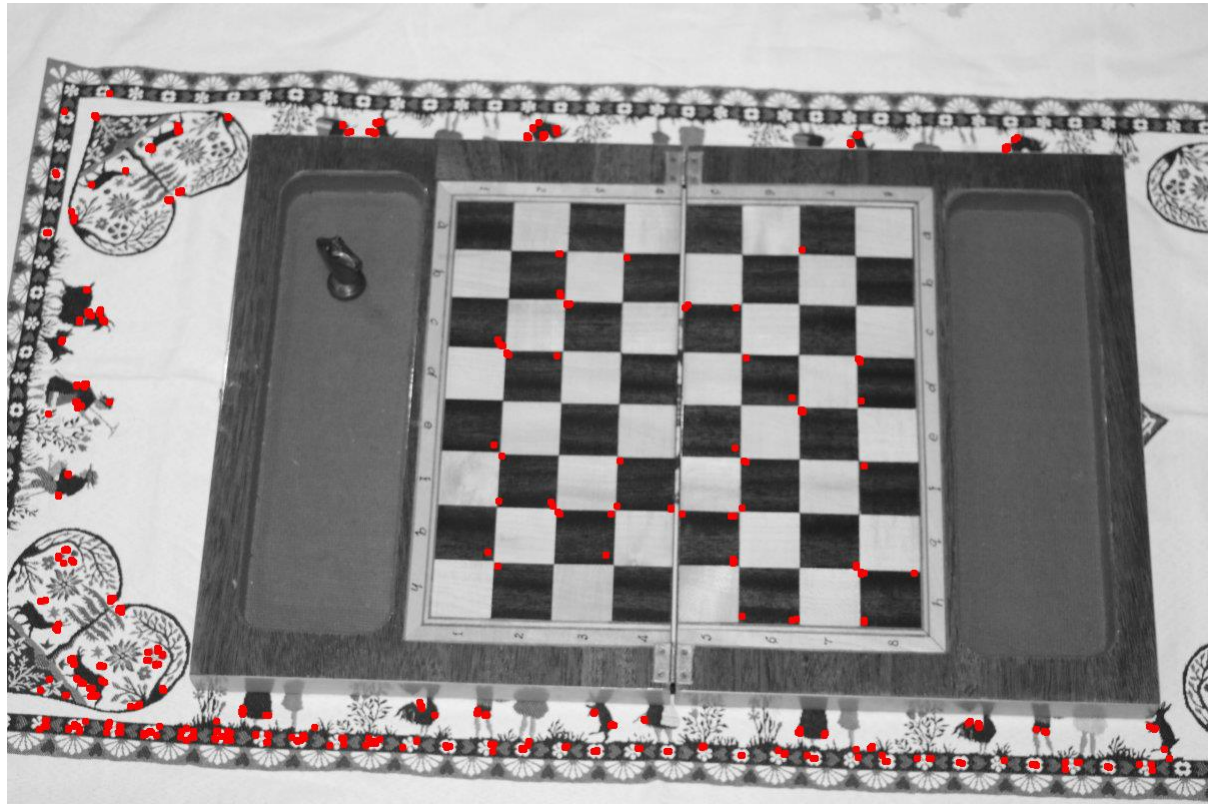
FAST: Features from Accelerated Segment Test





KeyPoints detection

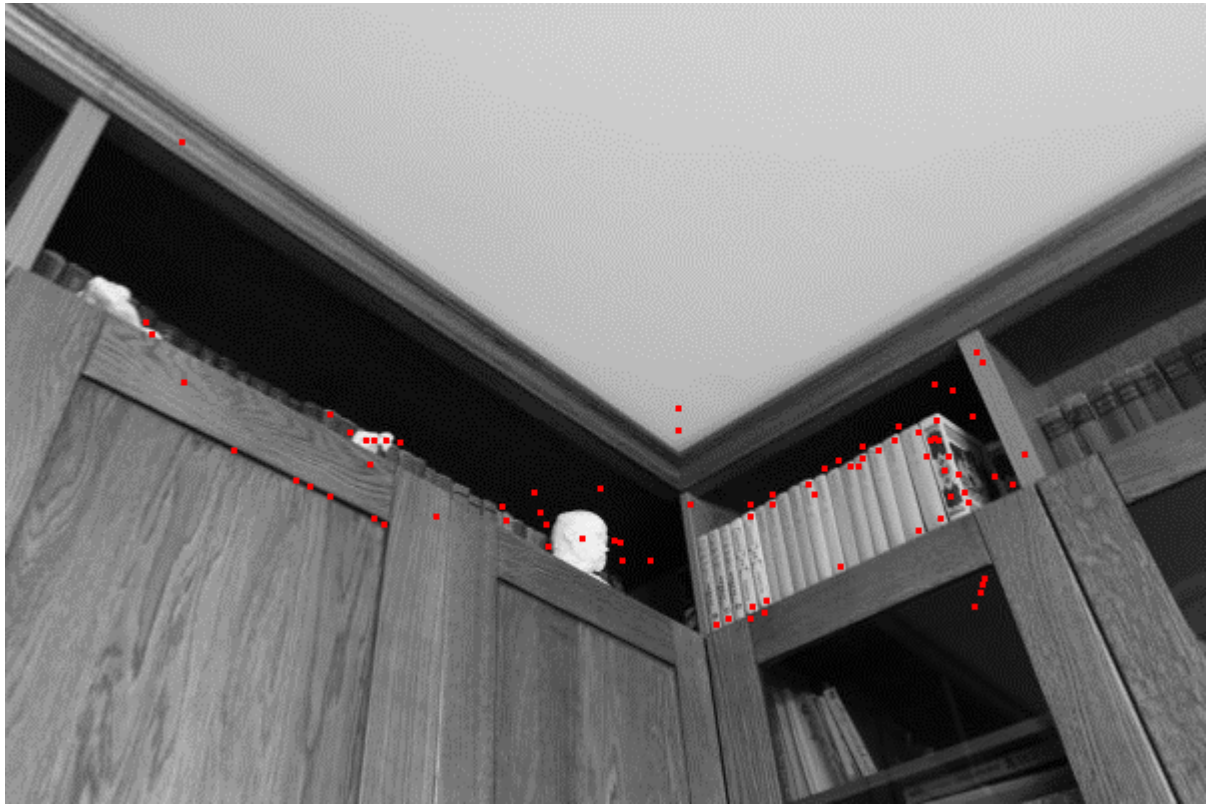
ORB: Oriented FAST and Rotated BRIEF

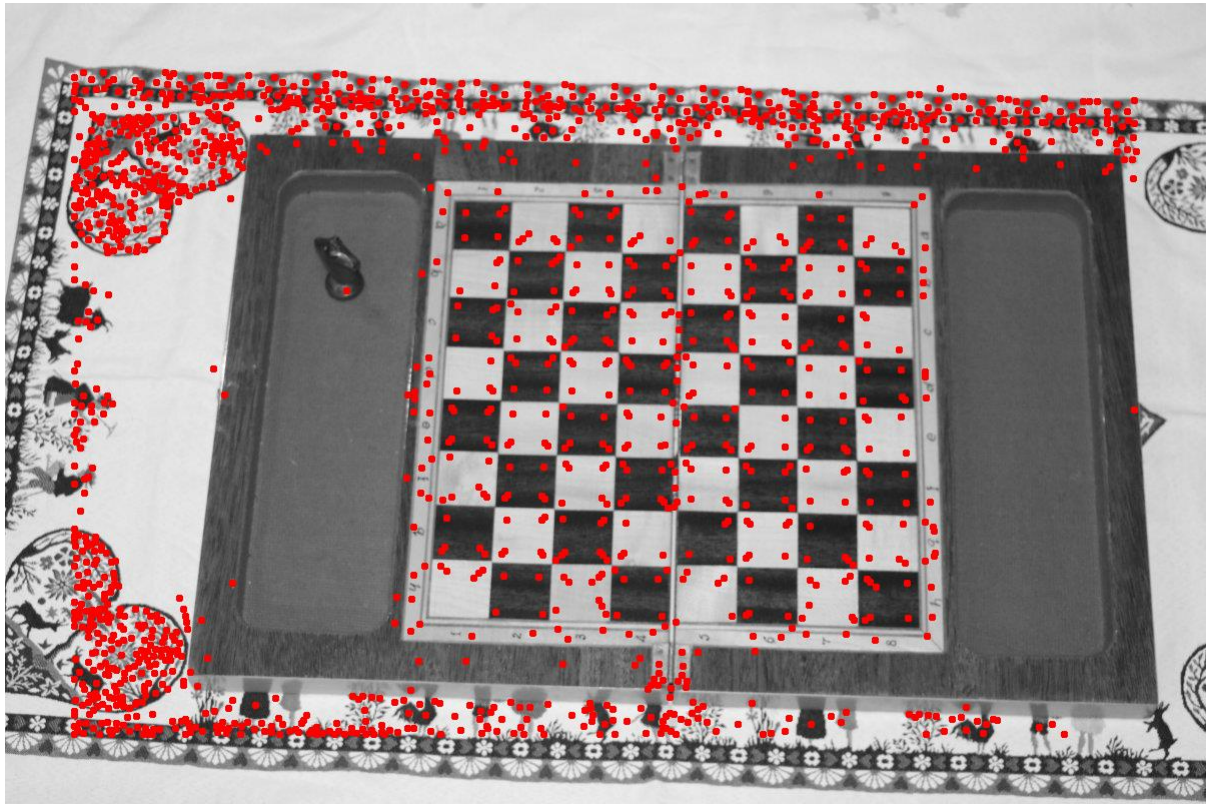


KeyPoints detection

STAR: scale-invariant **cen**ter-**surr**ound detector(CENSURE)

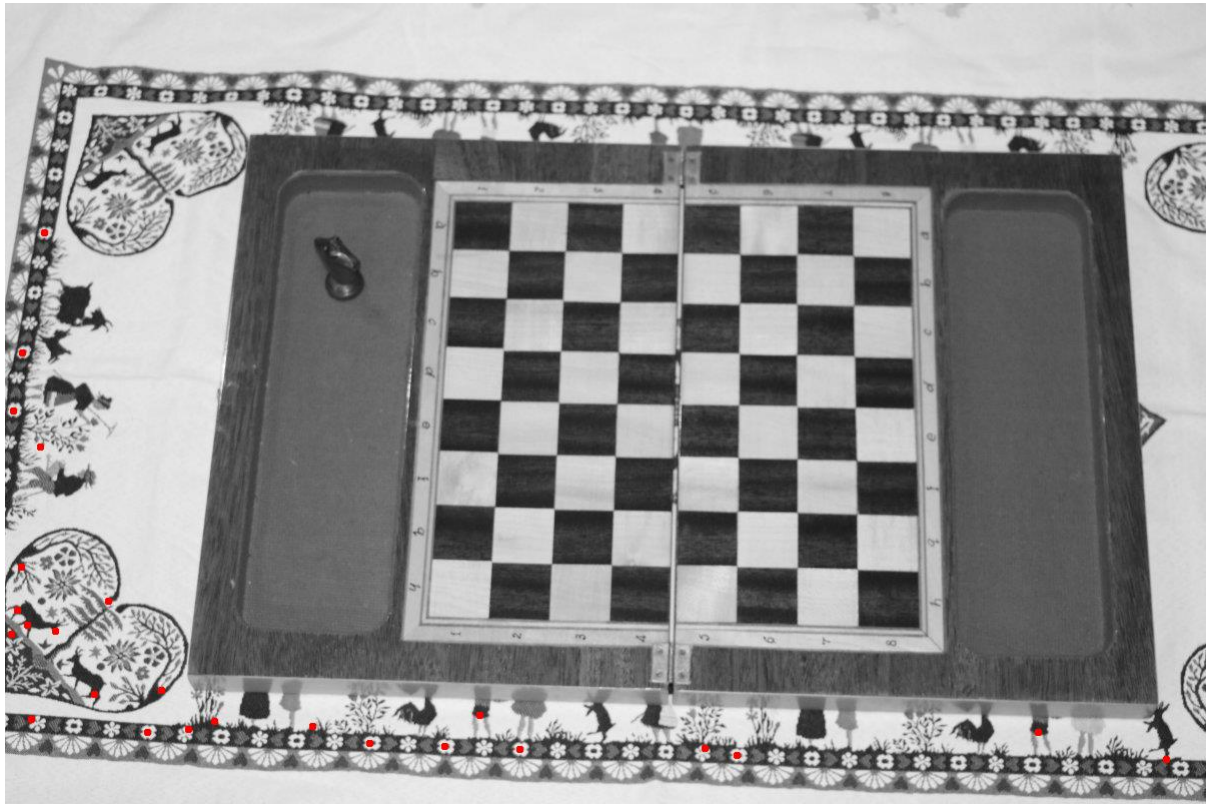
called STAR detector in OpenCV





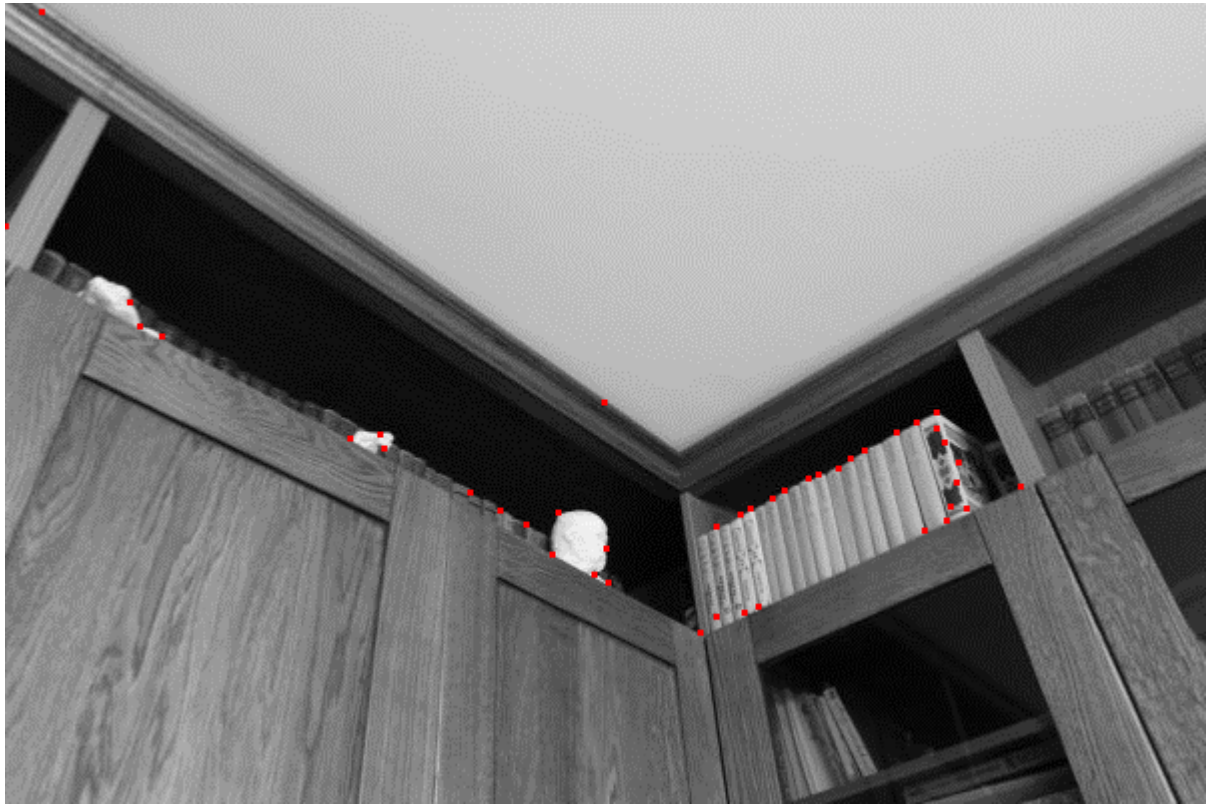
KeyPoints detection

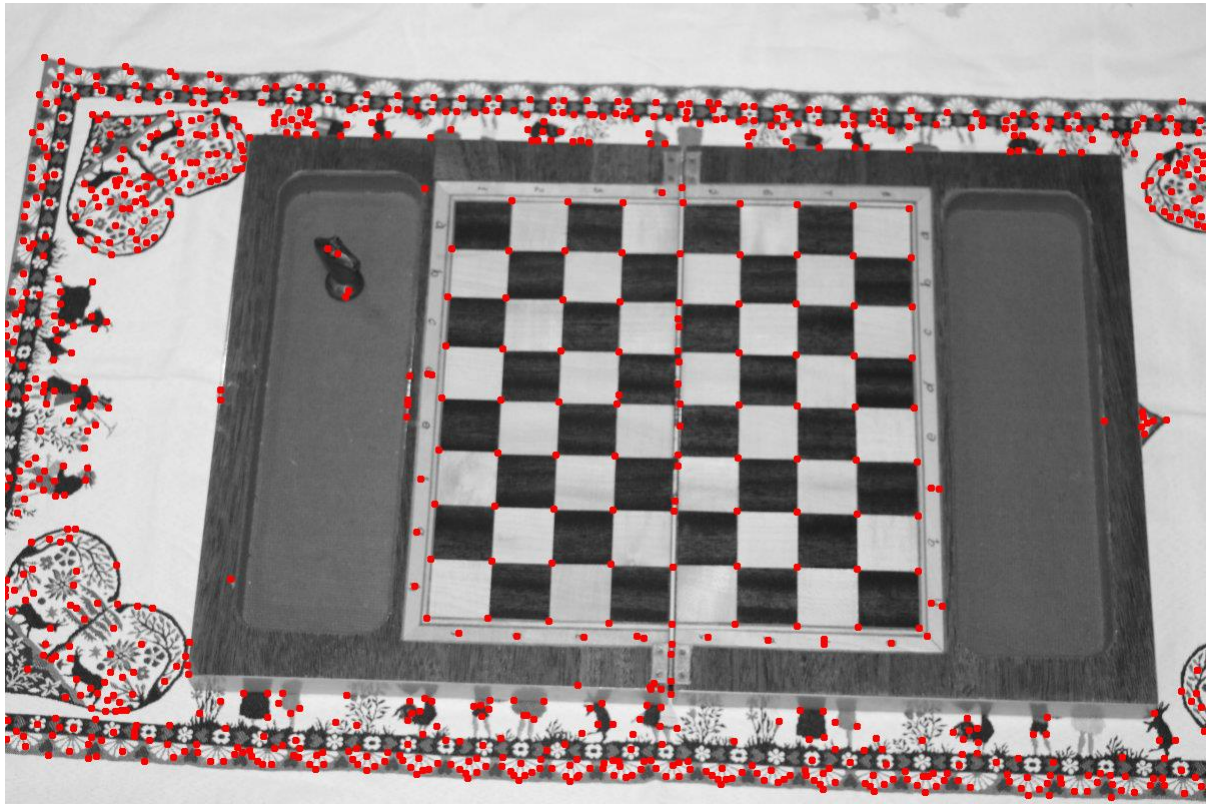
Shi-Tomasi



KeyPoints detection

Harris

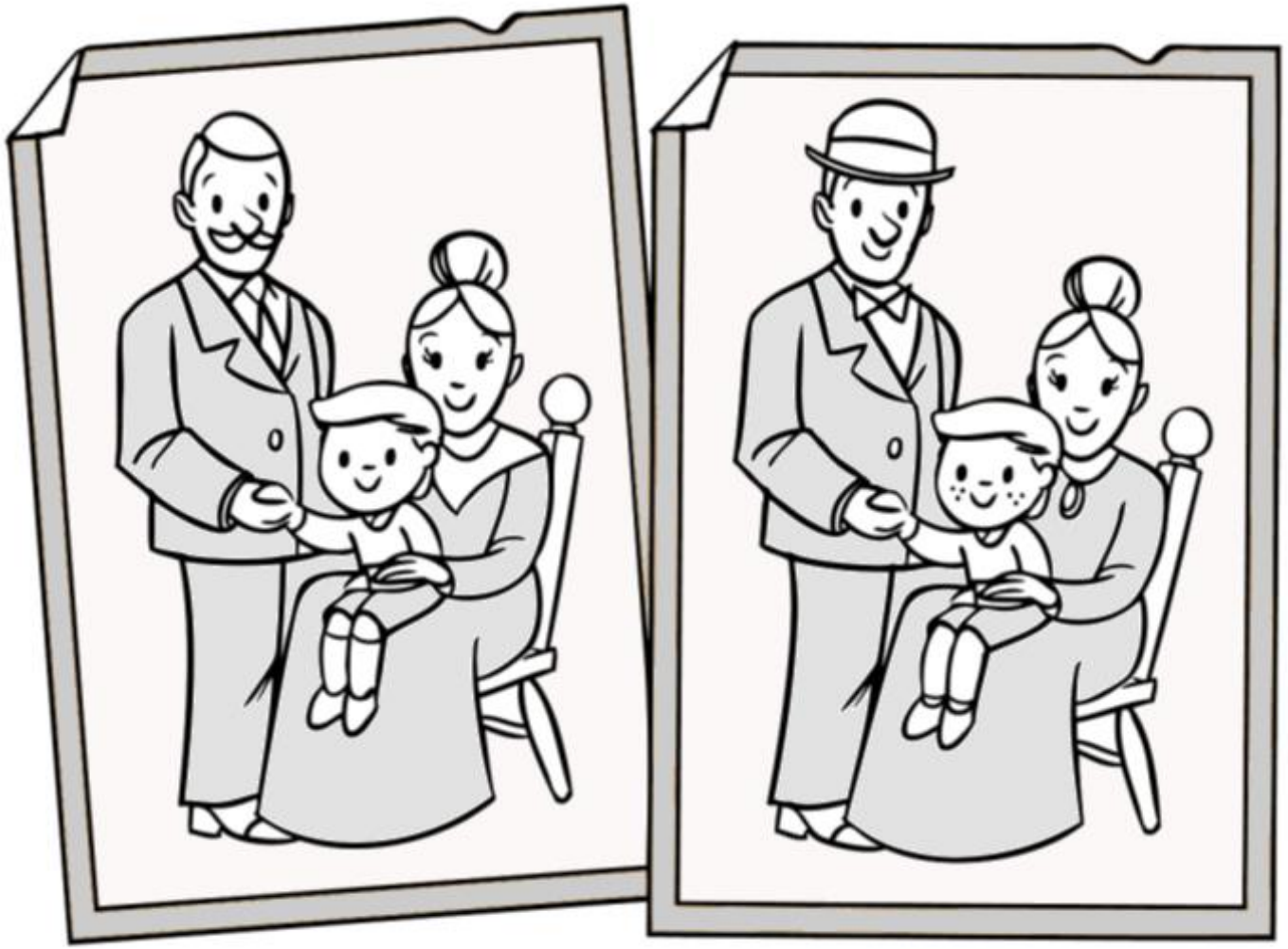




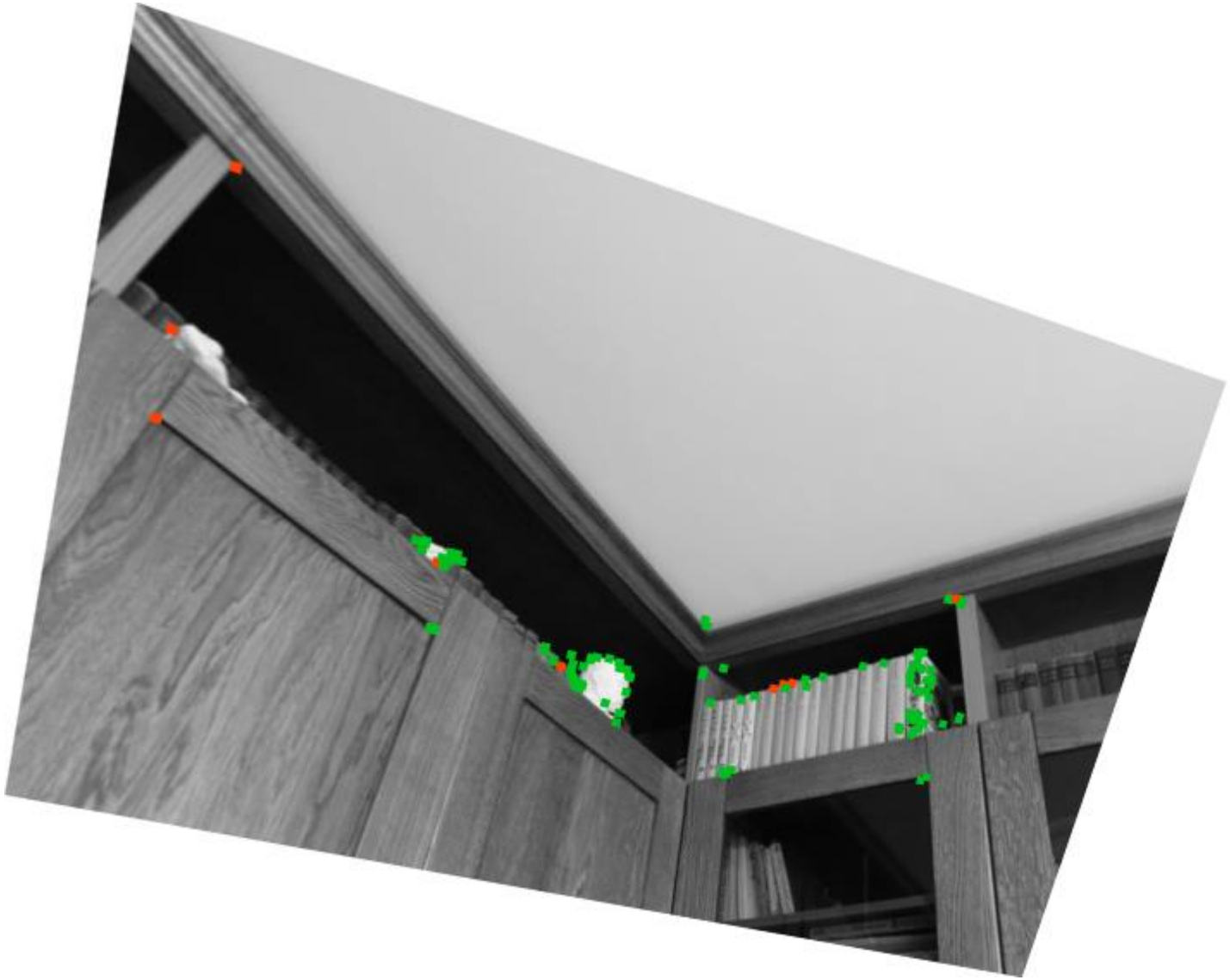
KeyPoints detection

```
def get_keypoints_desc(image1, detector='SIFT'):
    if len(image1.shape) == 2:
        im1_gray = image1.copy()
    else:
        im1_gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

    if detector == 'SIFT':
        detector = cv2.xfeatures2d.SIFT_create()
    elif detector == 'SURF':
        detector = cv2.xfeatures2d.SURF_create()
    else:  # detector == 'ORB'
        detector = cv2.ORB_create()

    kp, desc = detector.detectAndCompute(im1_gray, None)

    points = []
    if (len(kp) > 0):
        points = numpy.array([kp[int(idx)].pt for idx in range(0, len(kp))]).astype(int)

    return points, desc
```

2. Matching the KeyPoints



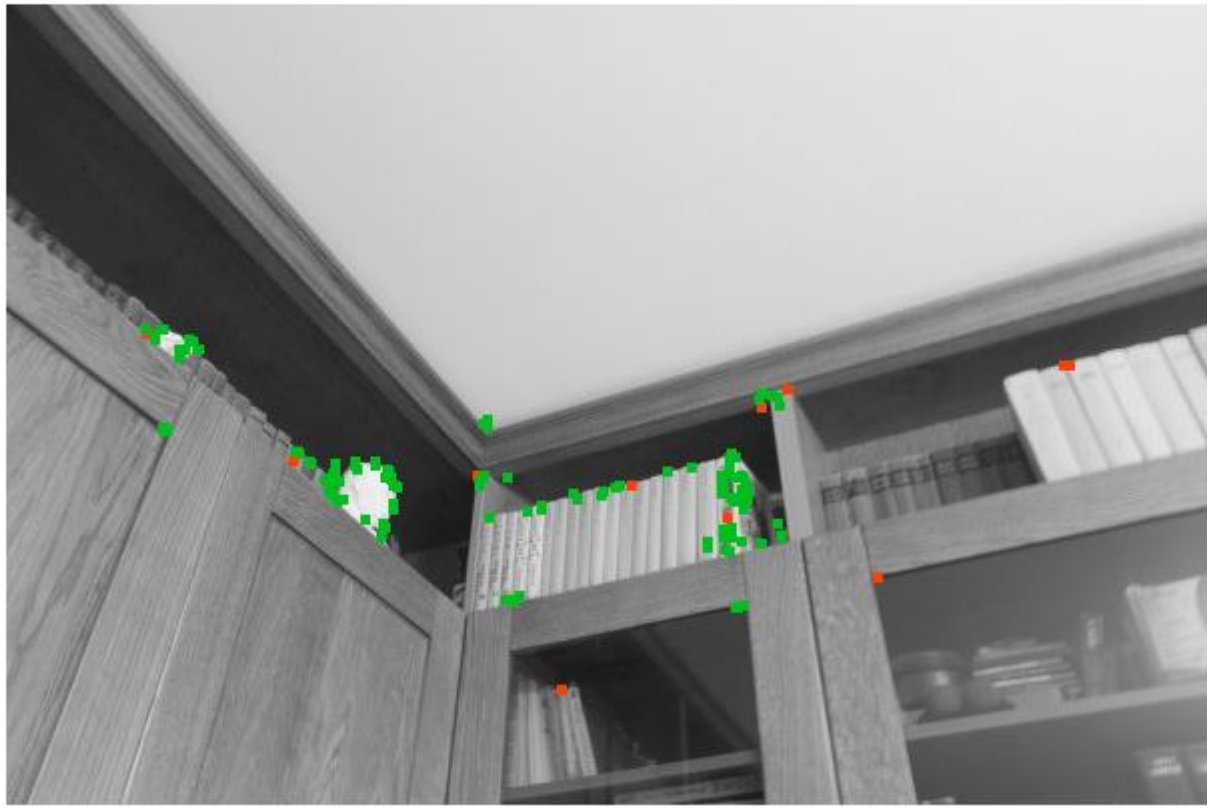
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

True Positive
False Positive

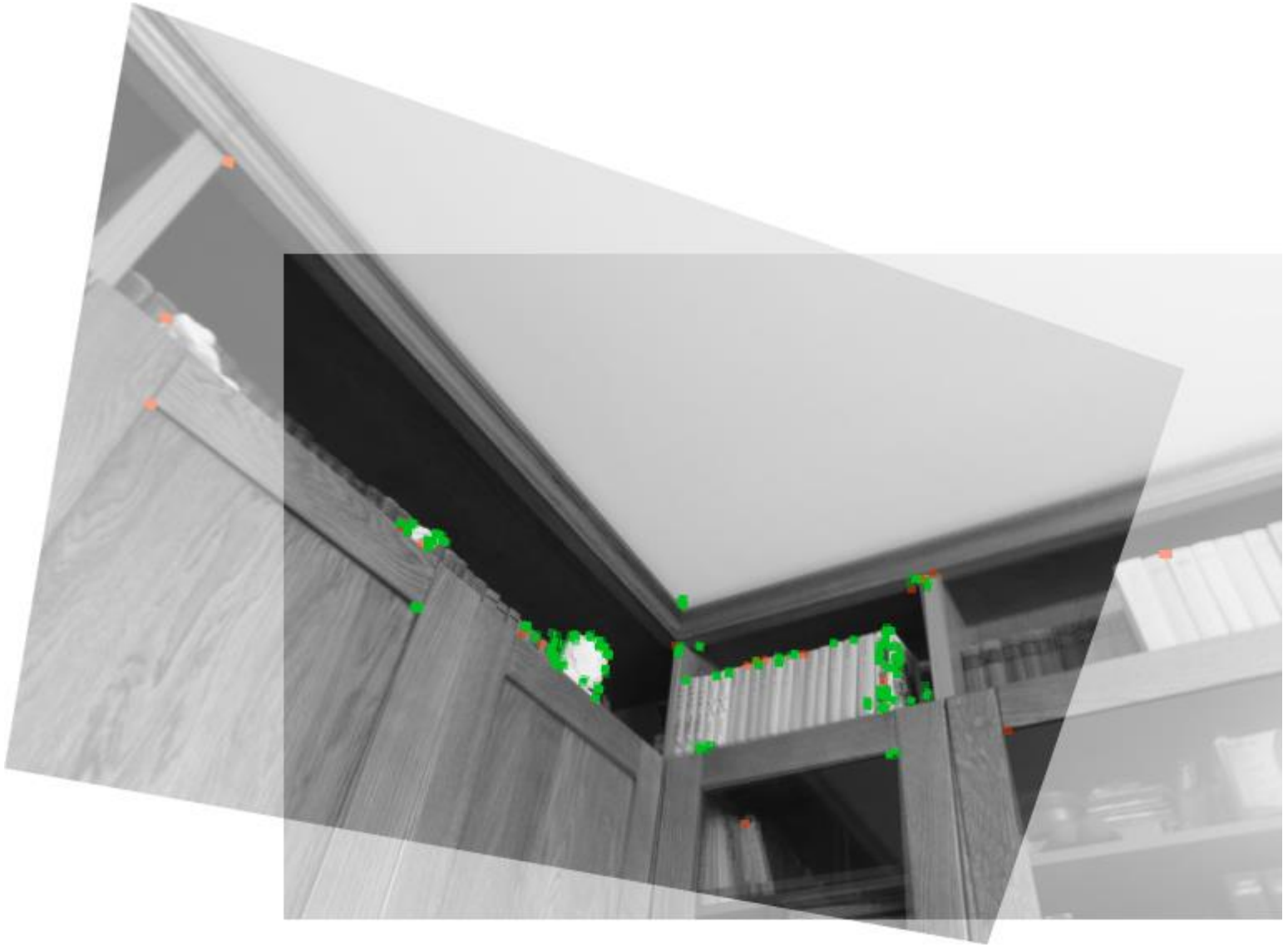
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

True Positive
False Positive

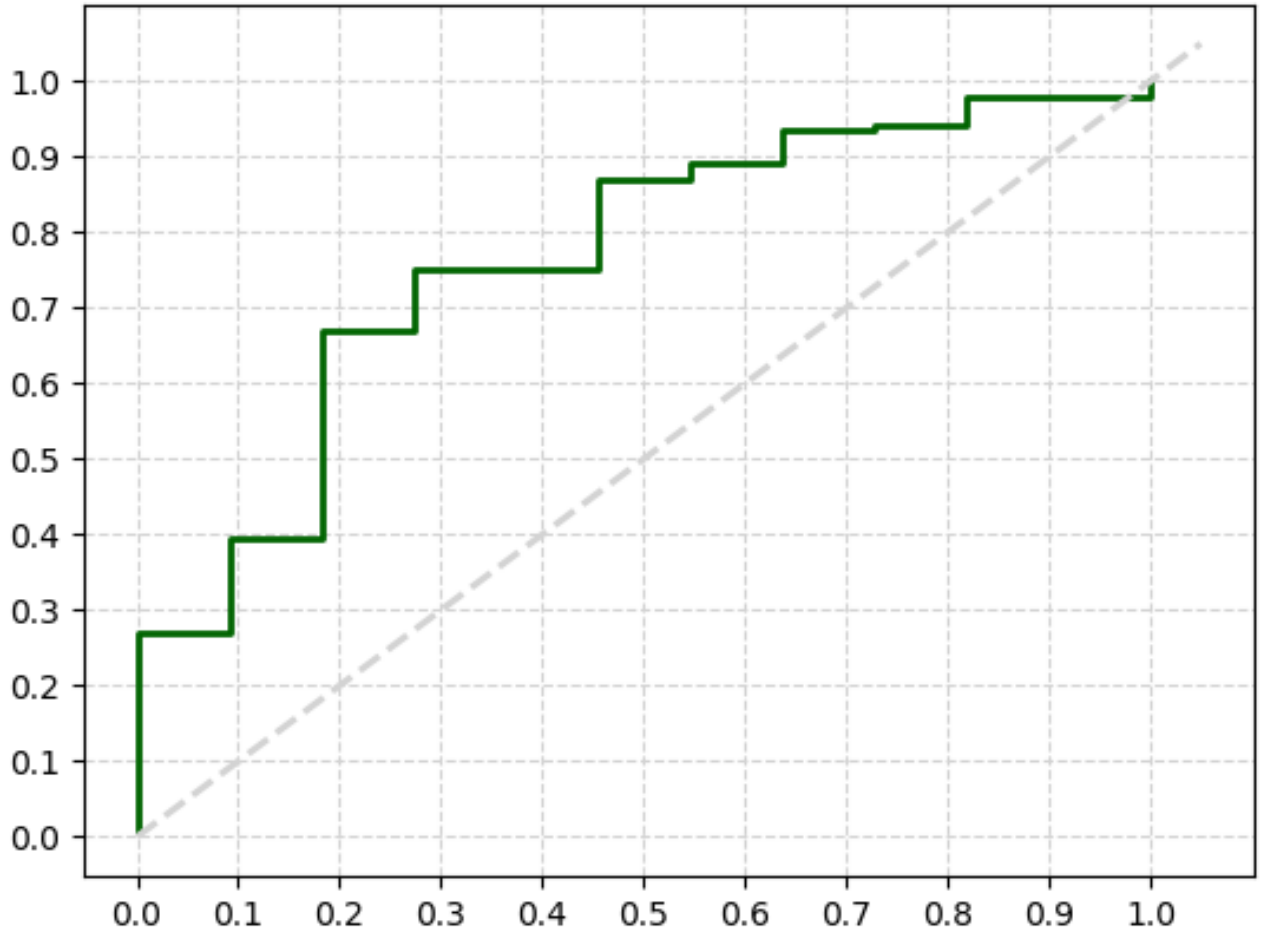
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

True Positive
False Positive

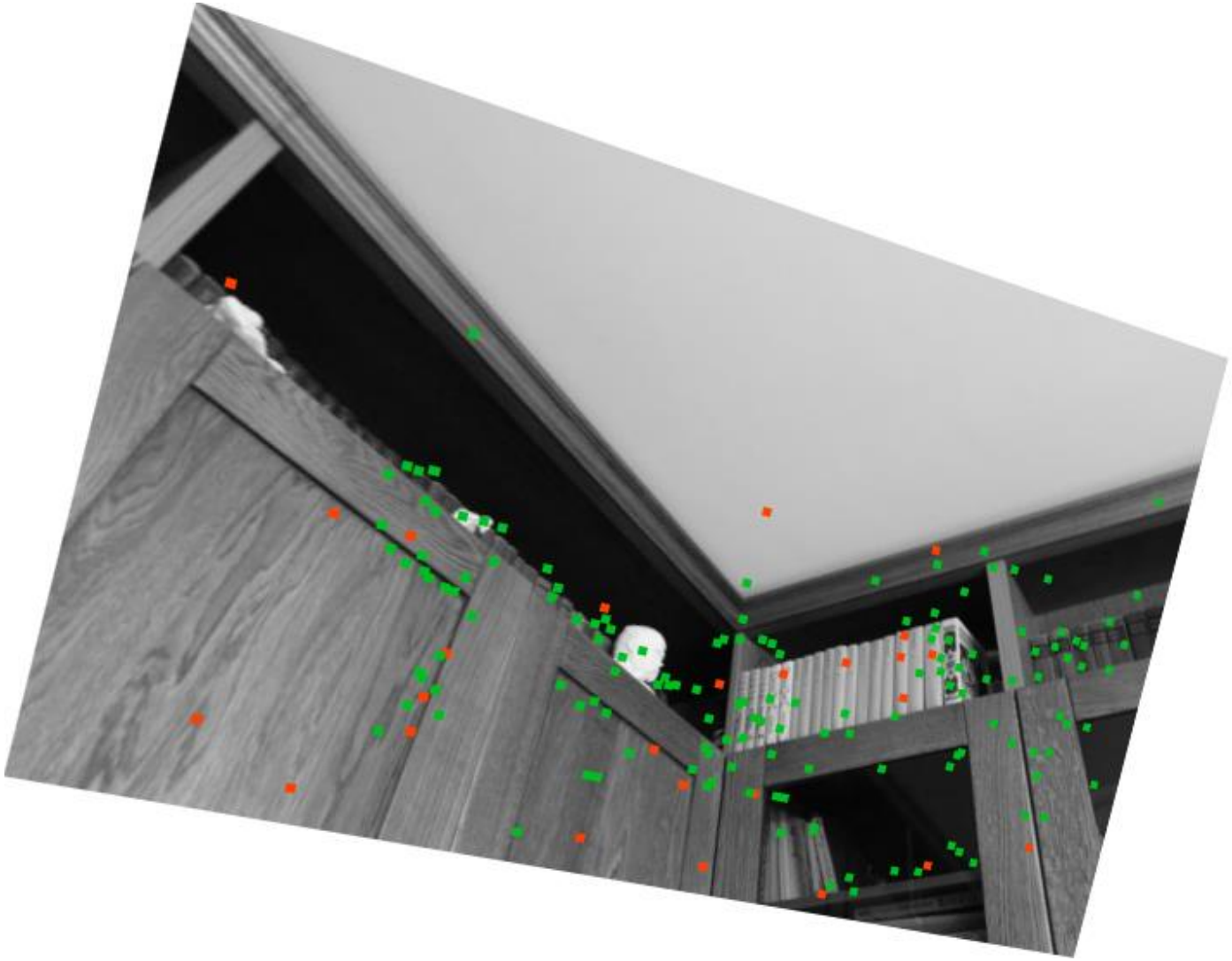
Matching the KeyPoints



Detector: ORB
Matcher: FLANN

AUC = 76%

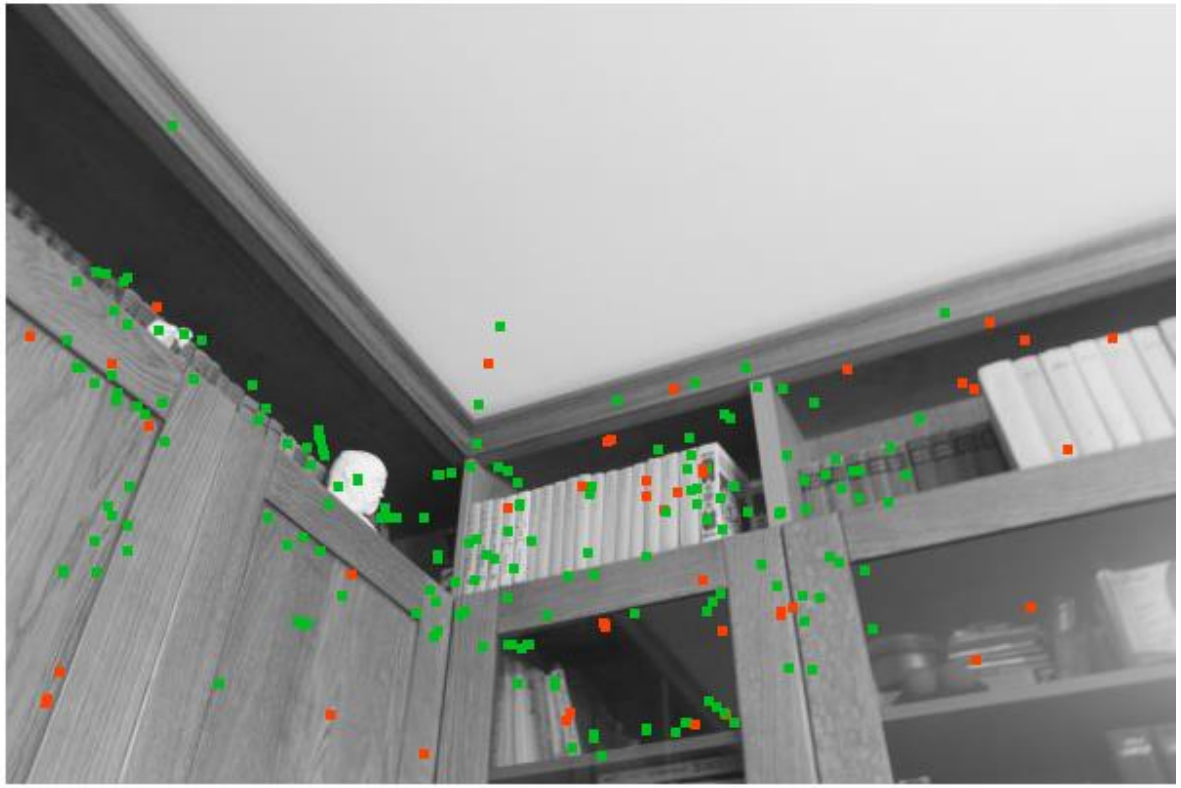
Matching the KeyPoints



Detector: SURF
Matcher: KNN

True Positive
False Positive

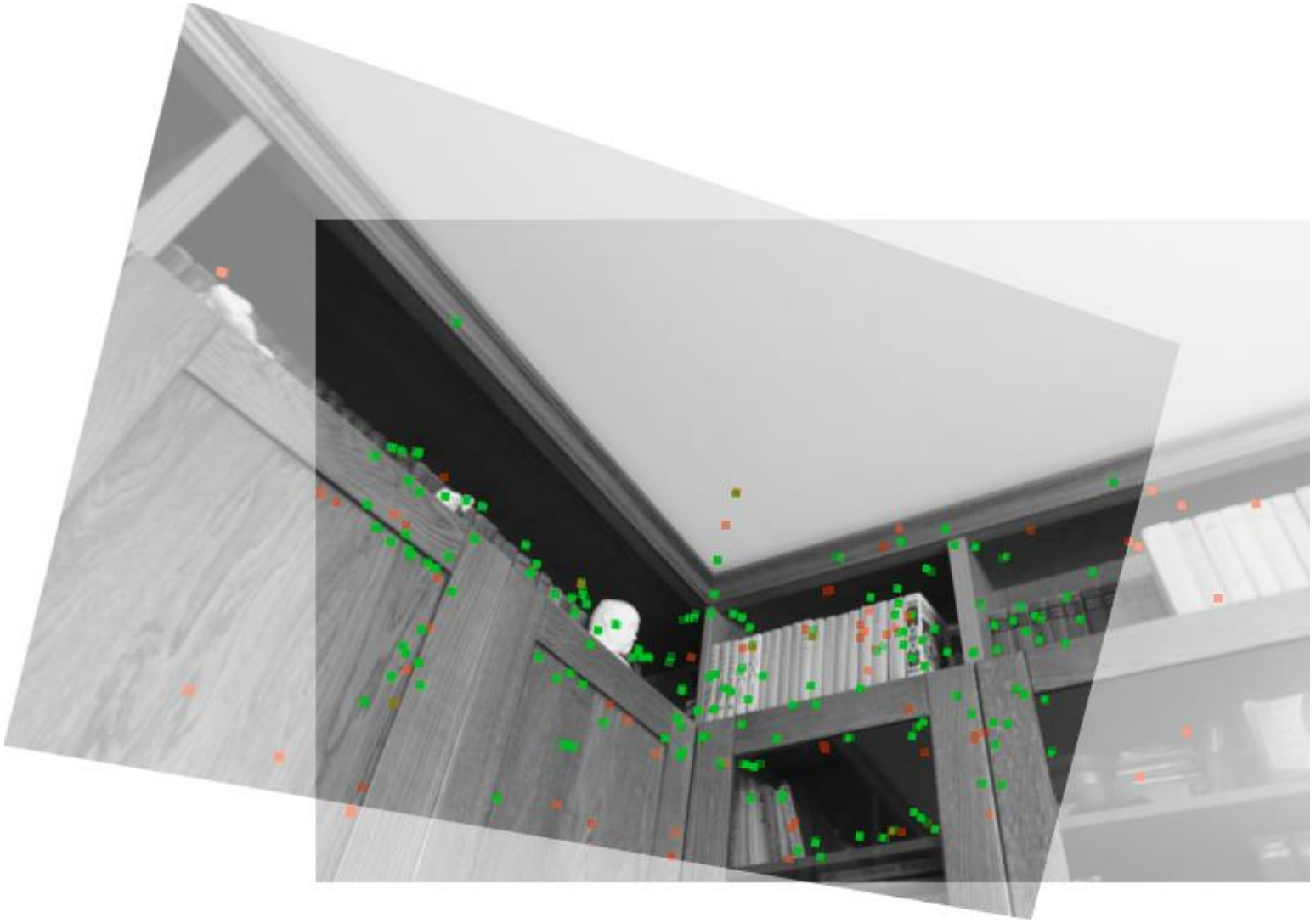
Matching the KeyPoints



Detector: SURF
Matcher: KNN

True Positive
False Positive

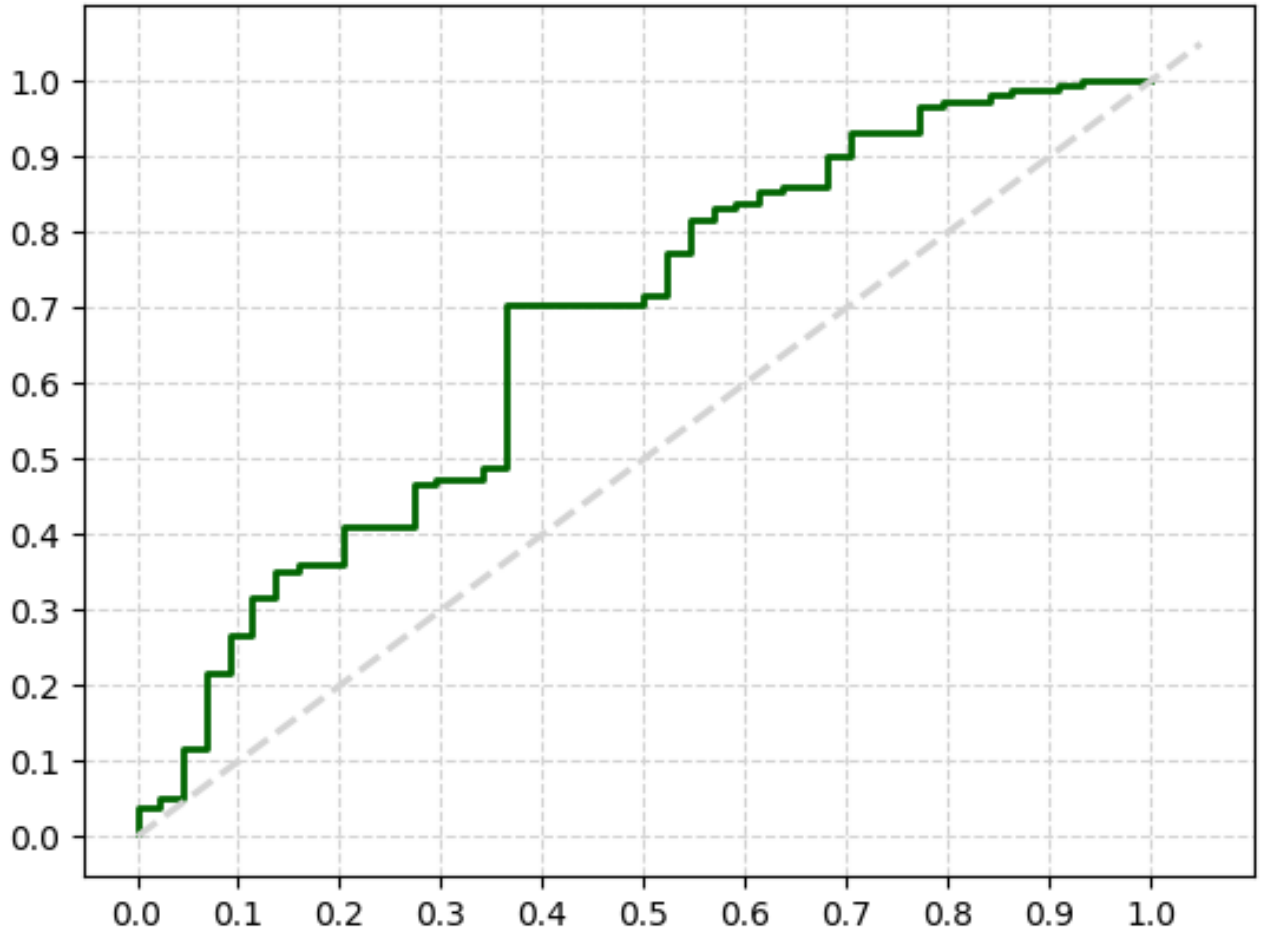
Matching the KeyPoints



Detector: SURF
Matcher: KNN

True Positive
False Positive

Matching the KeyPoints



Detector: SURF
Matcher: KNN

AUC = 67%

Matching the KeyPoints

	BFMatcher.**match**	BFMatcher.**knnMatch**	**Flann**BasedMatcher.knnMatch
ORB	91%	75%	76%
SIFT	63%	69%	70%
SURF	-	67%	69%

Matching the KeyPoints

```
def get_matches_on_desc_flann(des_source, des_destin):

    FLANN_INDEX_KDTREE = 1
    index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
    search_params = dict(checks=50)  # or pass empty dictionary
    flann = cv2.FlannBasedMatcher(index_params, search_params)
    matches = flann.knnMatch(des_destin.astype(numpy.float32),
des_source.astype(numpy.float32), k=2)

    verify_ratio = 0.8
    verified_matches = []
    for m1, m2 in matches:
        if m1.distance < 0.8 * m2.distance:
            verified_matches.append(m1)

    return verified_matches
```

Matching the KeyPoints

```
def get_matches_on_desc_knn(des_source, des_destin):

    bf = cv2.BFMatcher()
    matches = bf.knnMatch(des_destin, des_source, k=2)

    verify_ratio = 0.8
    verified_matches = []
    for m1, m2 in matches:
        if m1.distance < 0.8 * m2.distance:
            verified_matches.append(m1)

    return verified_matches
```

Matching the KeyPoints

```
def get_matches_on_desc(des_source, des_destin):

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des_destin, des_source)
    matches = sorted(matches, key=lambda x: x.distance)

    return matches
```

3. Homography

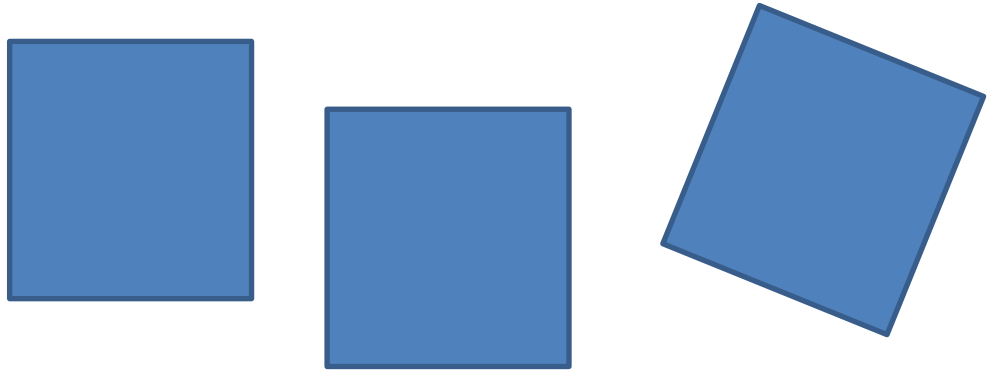
Homography: transformations

- *findTransformECC*
- *findHomography*

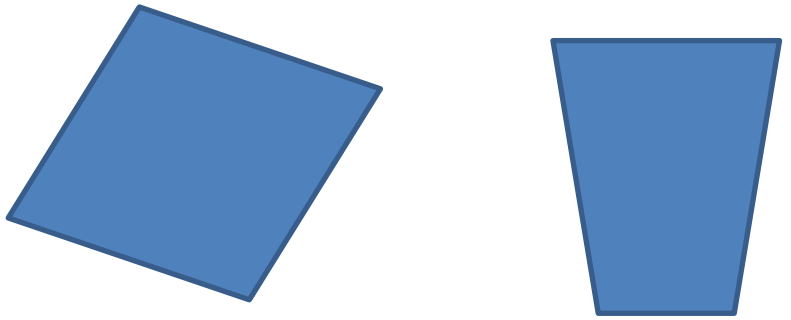
- *estimateAffine2D*,
- *estimateAffinePartial2D*

- *transform*
- *perspectiveTransform*
- *getPerspectiveTransform*

- *warpAffine*
- *warpPerspective*



Original **Translation** **Euclidian**



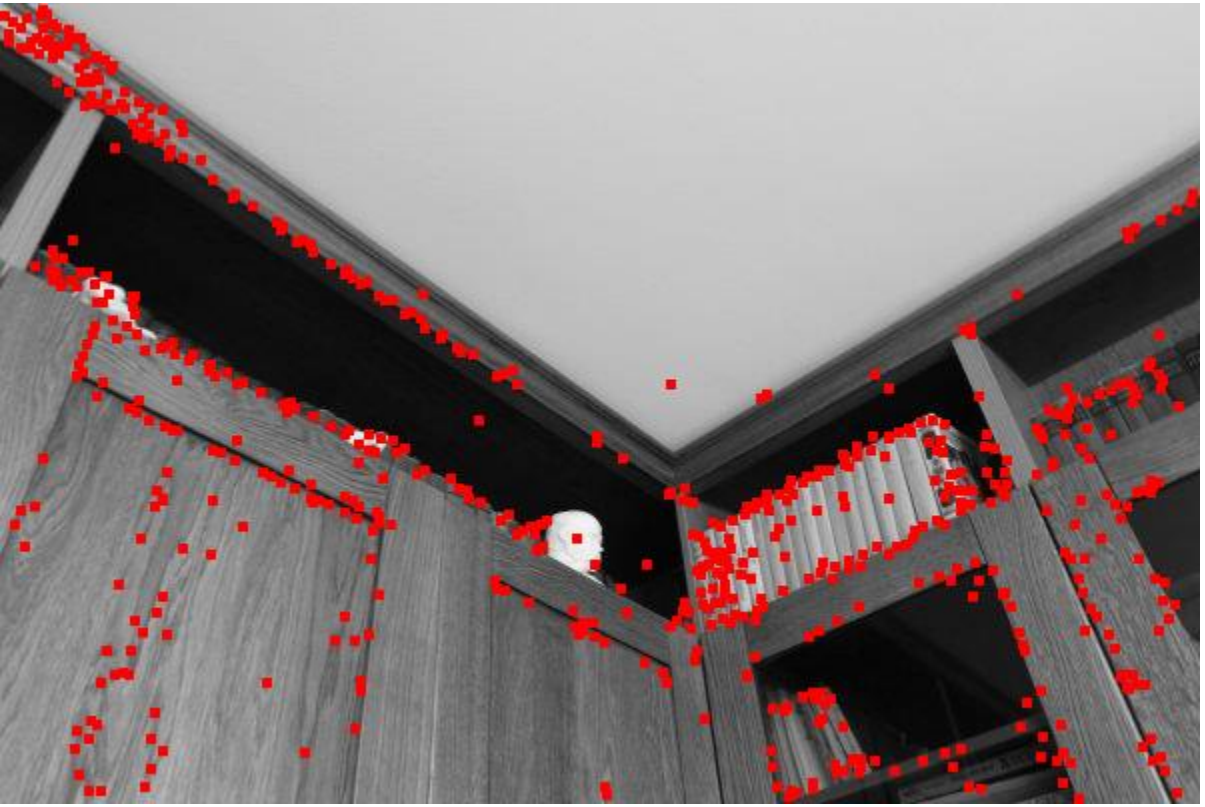
Affine **Homography**

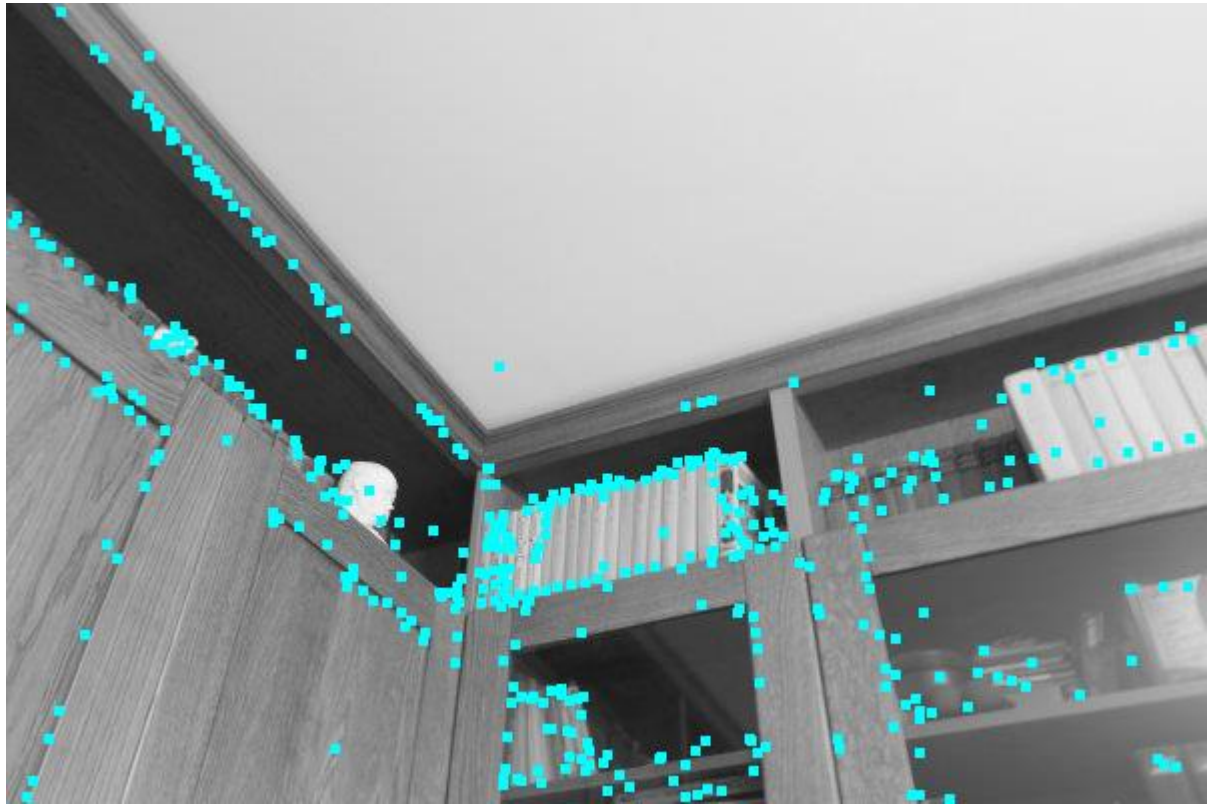
Homography: manual match

Homography: manual match

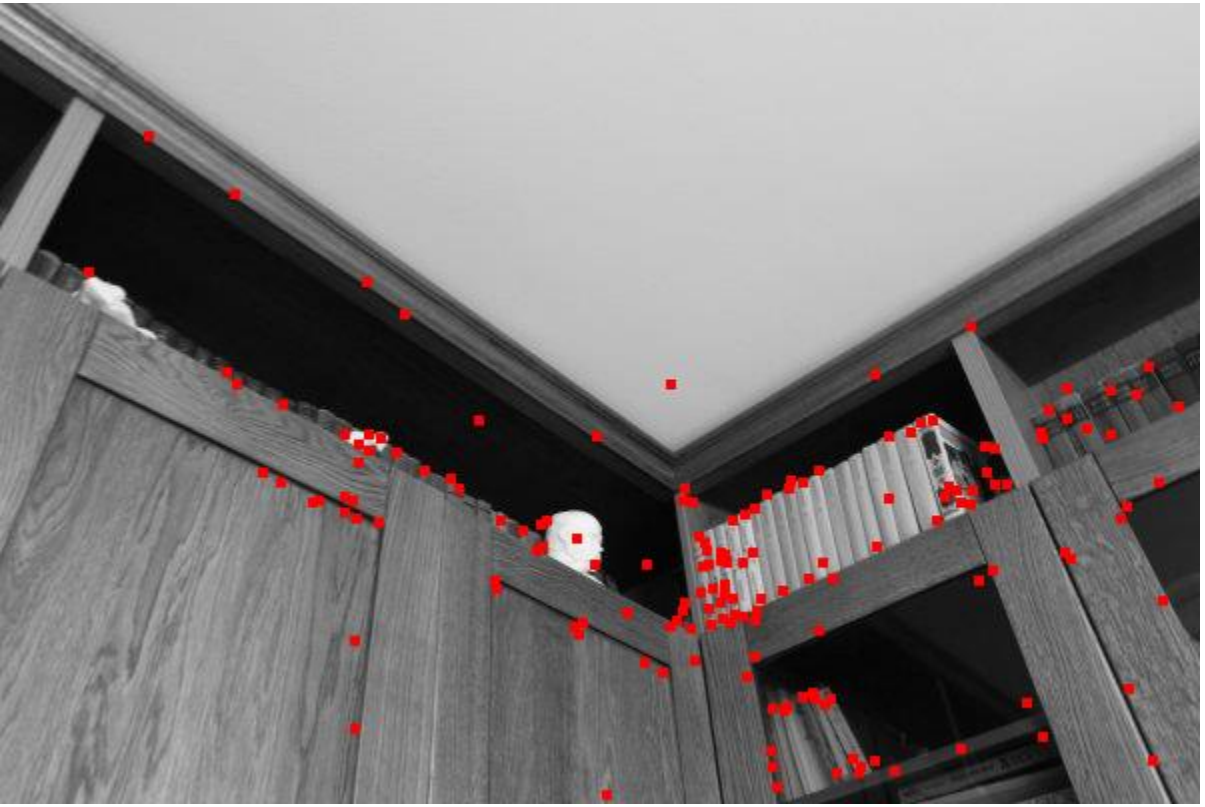
```
def example_03_find_homography_manual():

    folder_input            = '_images/ex03_manual/'
    im_scene                = cv2.imread(folder_input + 'background.jpg')
    im_rect                 = cv2.imread(folder_input + 'rect.jpg')
    folder_output           = '_images/output/'

    filename_output_rect    = folder_output + 'rect_out.jpg'
    filename_output_blend   = folder_output + 'bachground_out.jpg'

    if not os.path.exists(folder_output):
        os.makedirs(folder_output)
    else:
        tools_IO.remove_files(folder_output)

    pts_scene = numpy.array([[296, 95], [571, 178], [404, 614], [130, 531]])
    pts_rect  = numpy.array([[0  , 0]  , [300, 0]  , [300, 400], [0 , 400]])

    homography, status = cv2.findHomography(pts_rect,pts_scene)
    im_rect_out = cv2.warpPerspective(im_rect, homography, (im_scene.shape[1], im_scene.shape[0]))
    im_scene = tools_image.put_layer_on_image(im_scene, im_rect_out)

    cv2.imwrite(filename_output_rect, im_rect_out)
    cv2.imwrite(filename_output_blend, im_scene)

    return
```

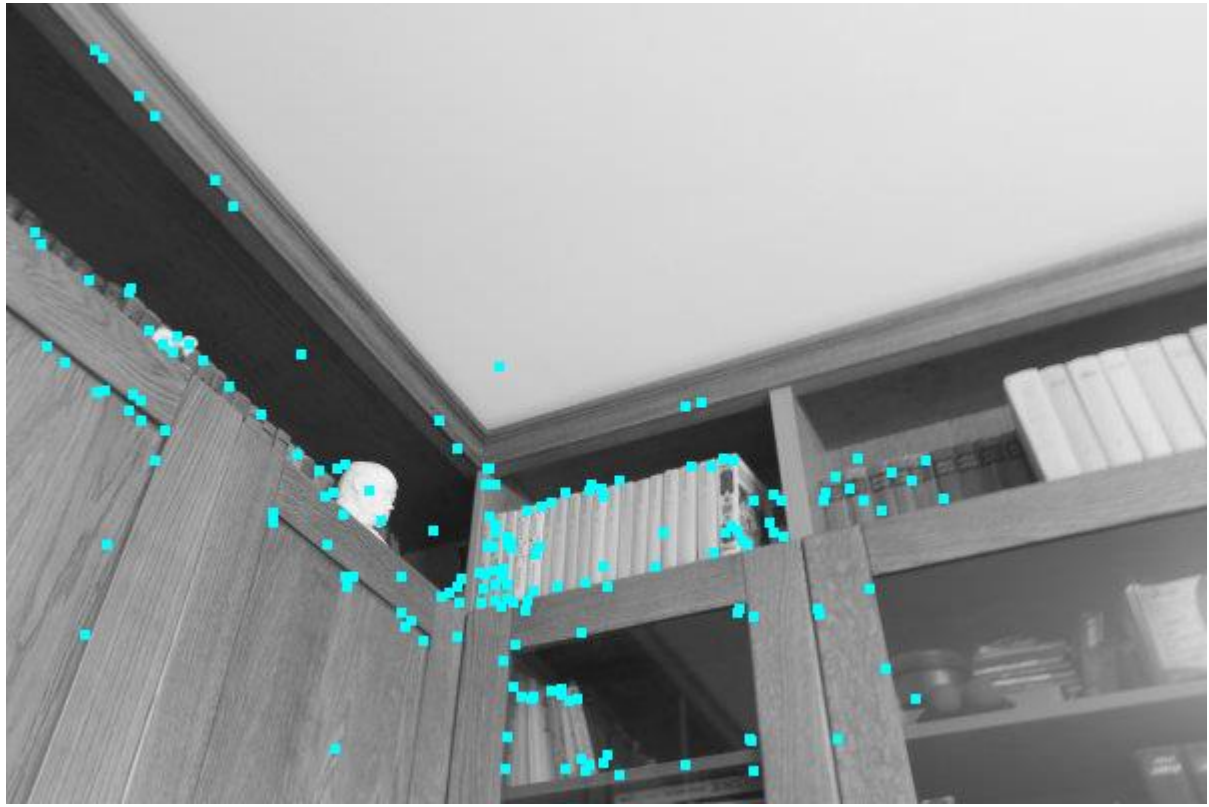
Homography: auto-match



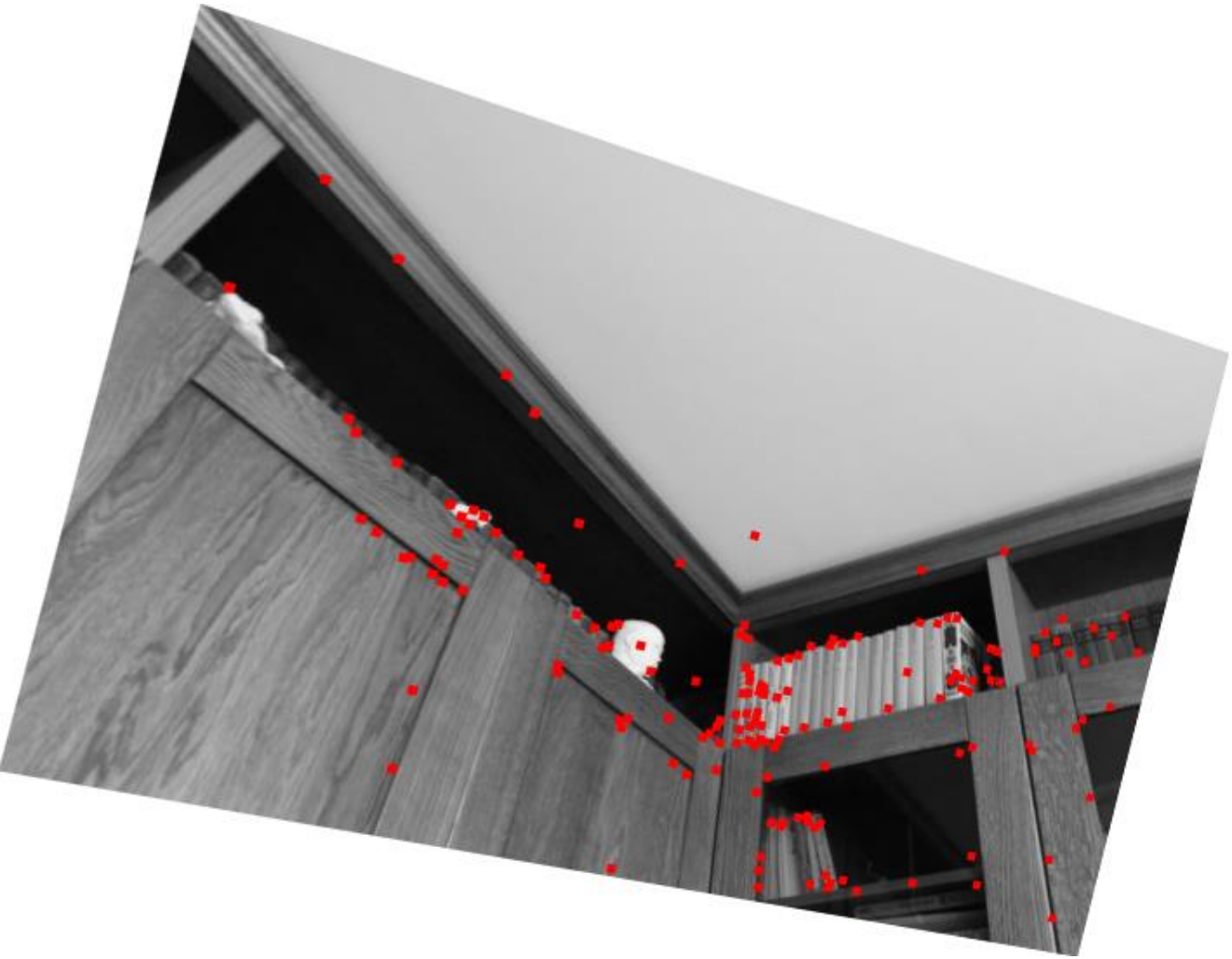


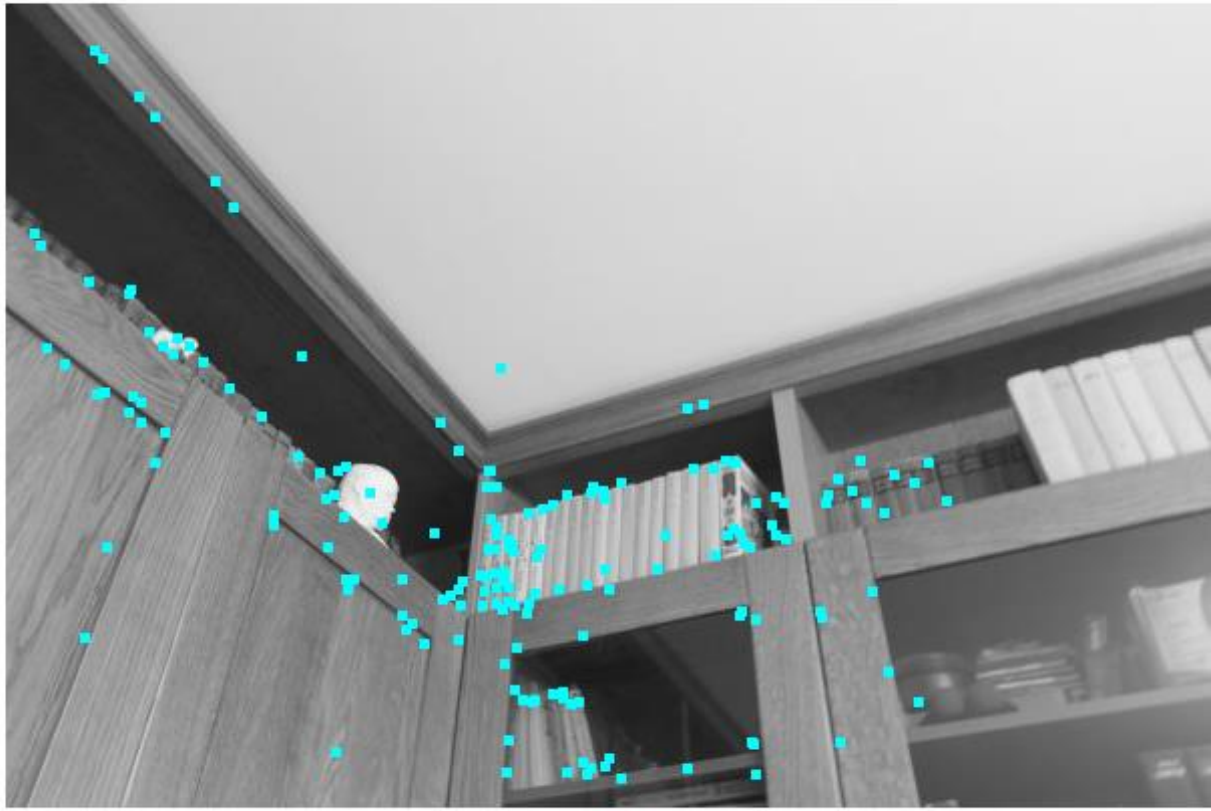
Homography: auto-match



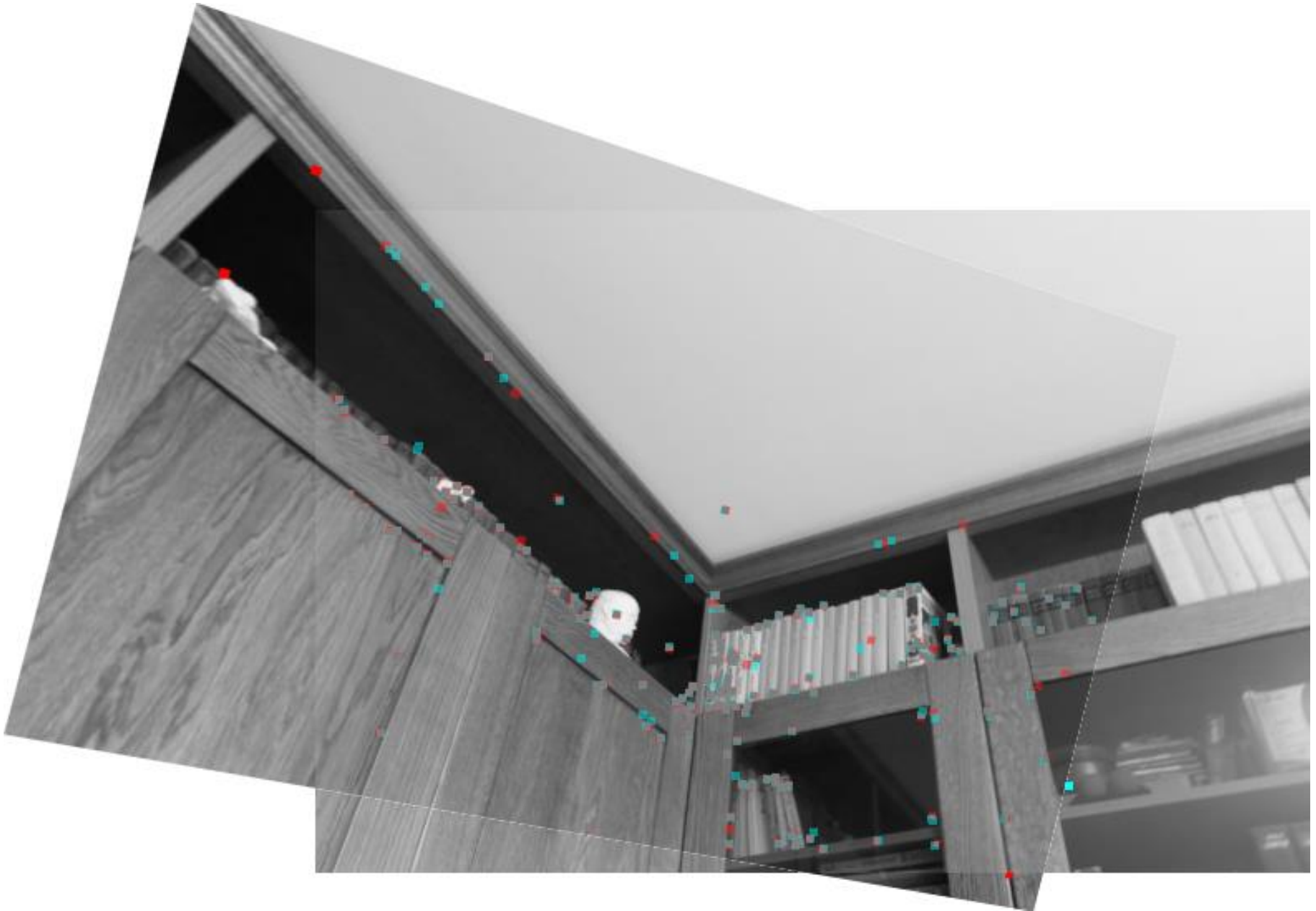


Homography: auto-match

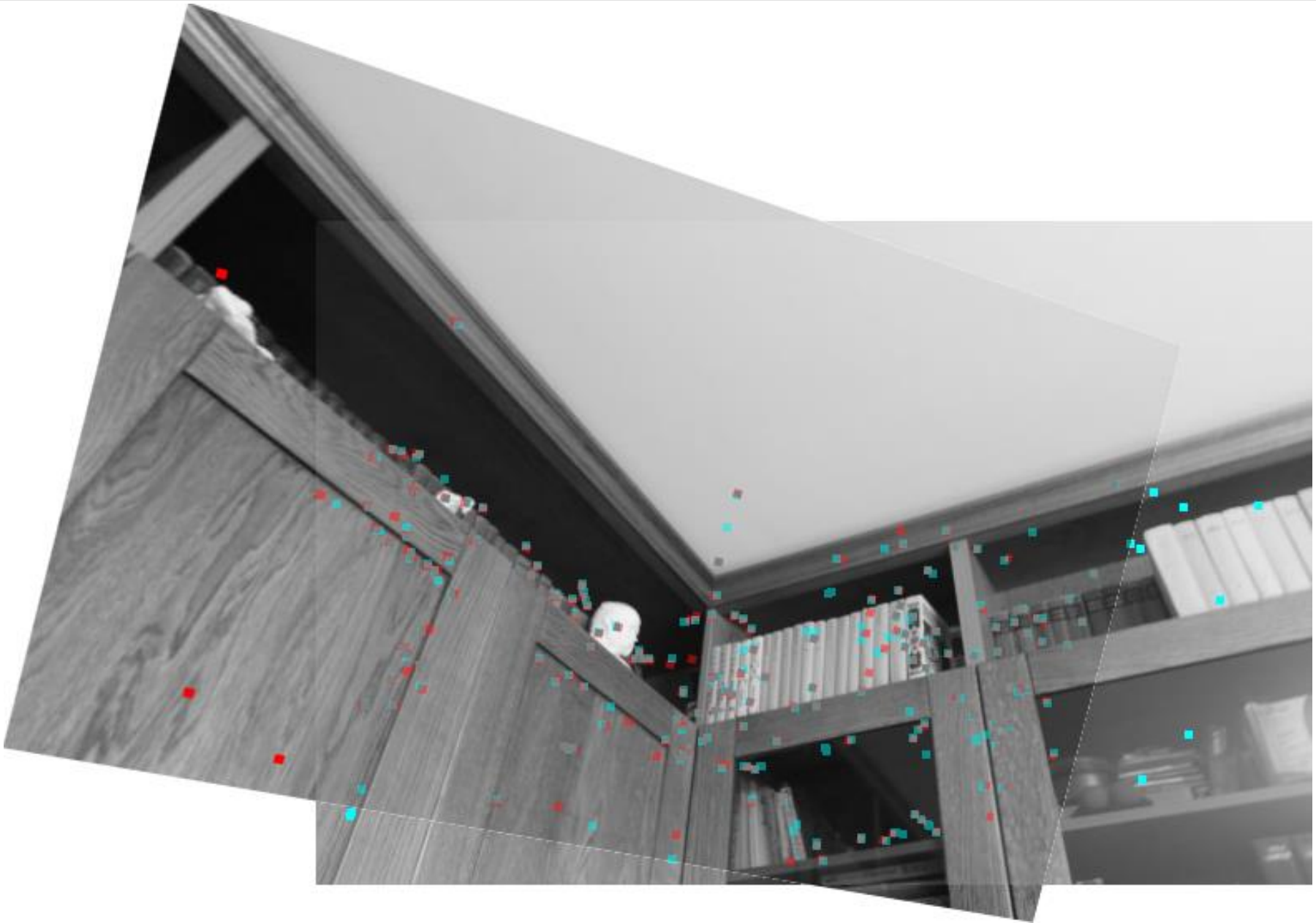




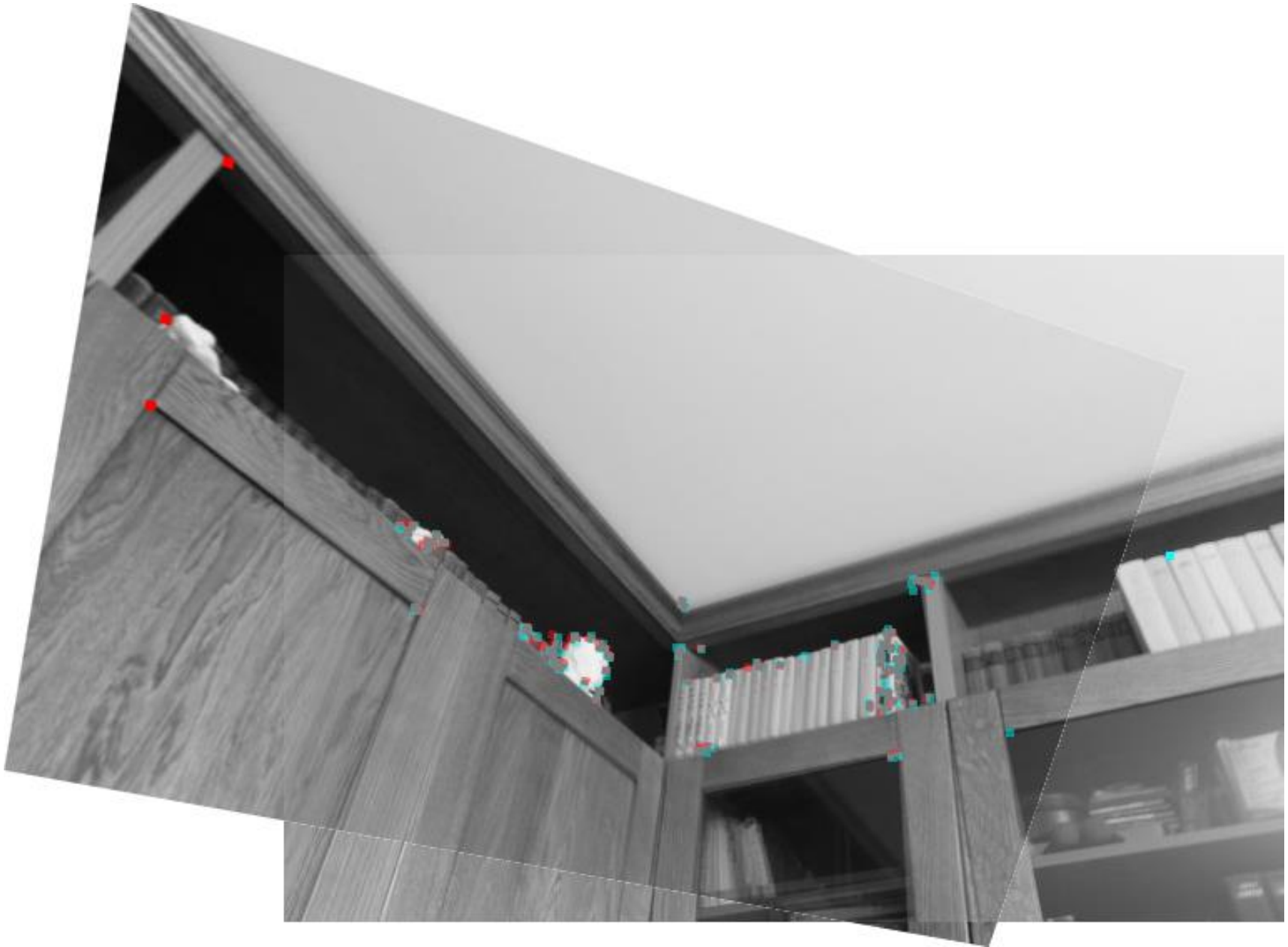
Homography: auto-match SIFT



Homography: auto-match SURF



Homography: auto-match ORB



Homography: auto-match

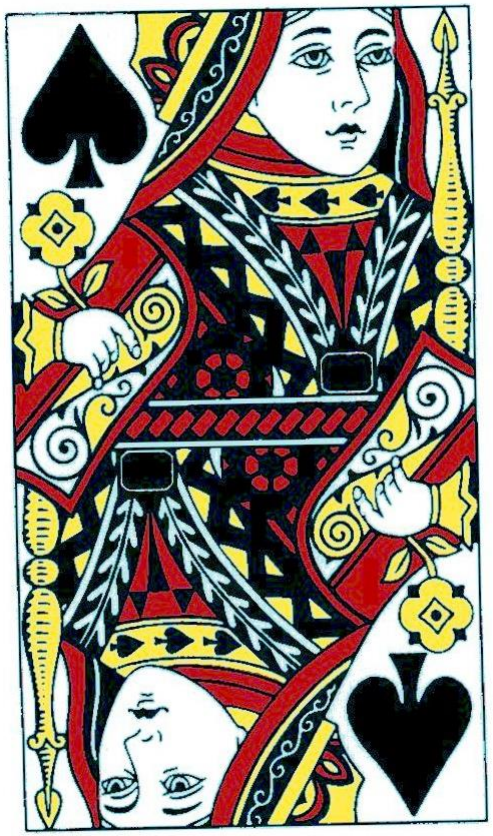
```
def get_matches_on_desc(des_source, des_destin):

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des_destin, des_source)
    matches = sorted(matches, key=lambda x: x.distance)

    return matches
```

Homography: auto-match

```
def get_keypoints_desc_SIFT(image1):
    if len(image1.shape) == 2:
        im1_gray = im1.copy()
    else:
        im1_gray = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

    detector = cv2.xfeatures2d.SIFT_create()
    kp, desc= detector.detectAndCompute(image1, None)

    points=[]
    if (len(kp) > 0):
        points = numpy.array([kp[int(idx)].pt for idx in range(0,
len(kp))]).astype(int)

    return points, desc


def get_homography_by_keypoints_desc(points_source,des_source, points_destin,des_destin,matchtype='knn'):

    if matchtype=='knn':
        matches = get_matches_on_desc_knn(des_source, des_destin)
    else:
        matches = get_matches_on_desc(des_source, des_destin)

    src = numpy.float32([points_source[m.trainIdx] for m in matches]).reshape(-1, 1, 2)
    dst = numpy.float32([points_destin[m.queryIdx] for m in matches]).reshape(-1, 1, 2)

    M = get_homography_by_keypoints(src, dst)
    return M
```

Homography: bitmap transformation

first



second

Homography: bitmap transformation

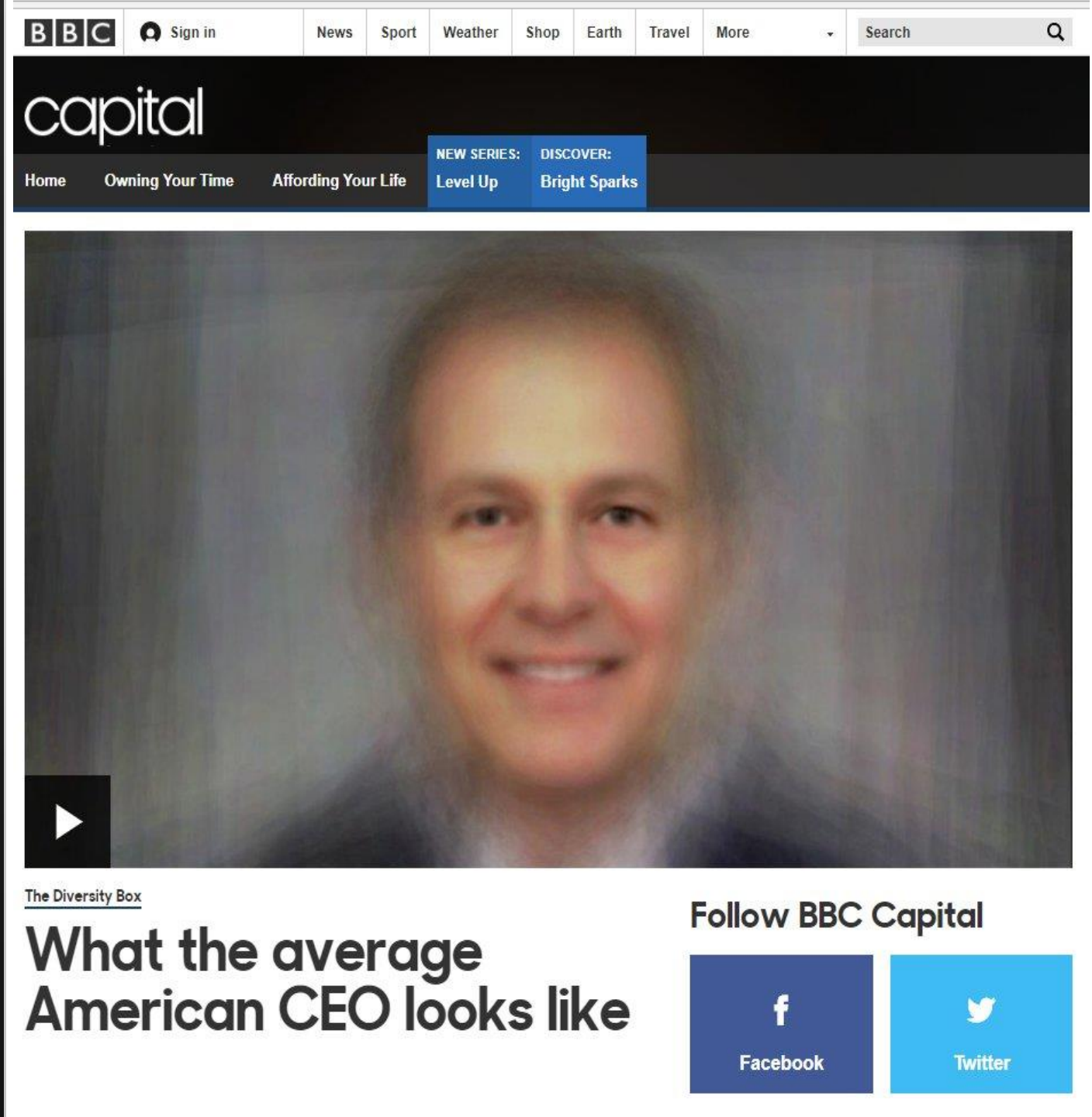
*transformed **second***

*transformed **first***

Homography: bitmap transformation

```
criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 5000, 1e-10)

if warp_mode!= cv2.MOTION_HOMOGRAPHY:
    (cc, transform1) = cv2.findTransformECC(im1_gray, im2_gray, numpy.eye(2, 3, dtype=numpy.float32), warp_mode,criteria)
    (cc, transform2) = cv2.findTransformECC(im2_gray, im1_gray, numpy.eye(2, 3, dtype=numpy.float32), warp_mode,criteria)
    result_image11, result_image12 = tools_calibrate.get_stitched_images_using_translation(im2, im1, transform1)
    result_image22, result_image21 = tools_calibrate.get_stitched_images_using_translation(im1, im2, transform2)
else:
    (cc, transform1) = cv2.findTransformECC(im1_gray, im2_gray, numpy.eye(3, 3, dtype=numpy.float32), warp_mode,criteria)
    (cc, transform2) = cv2.findTransformECC(im2_gray, im1_gray, numpy.eye(3, 3, dtype=numpy.float32), warp_mode,criteria)
    result_image11, result_image12 = tools_calibrate.get_stitched_images_using_homography(im2, im1, transform1)
    result_image22, result_image21 = tools_calibrate.get_stitched_images_using_homography(im1, im2, transform2)
```

4. Blending


BBC
Sign in
News
Sport
Weather
Shop
Earth
Travel
More
Search
capital
Home
Owning Your Time
Affording Your Life
NEW SERIES:
Level Up
DISCOVER:
Bright Sparks
The Diversity Box
What the average American CEO looks like
Follow BBC Capital
Facebook
Twitter

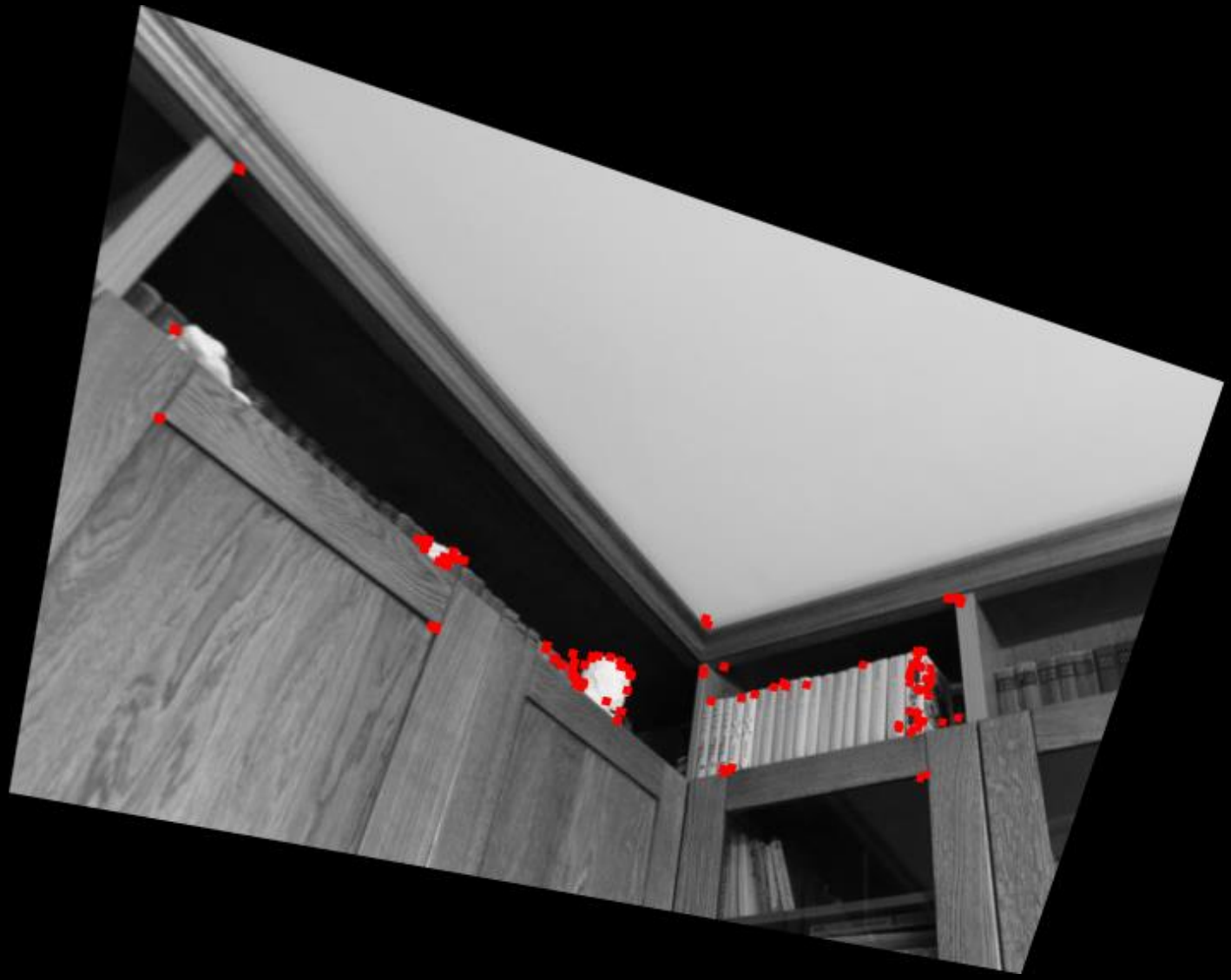
Blending: average

Blending: average

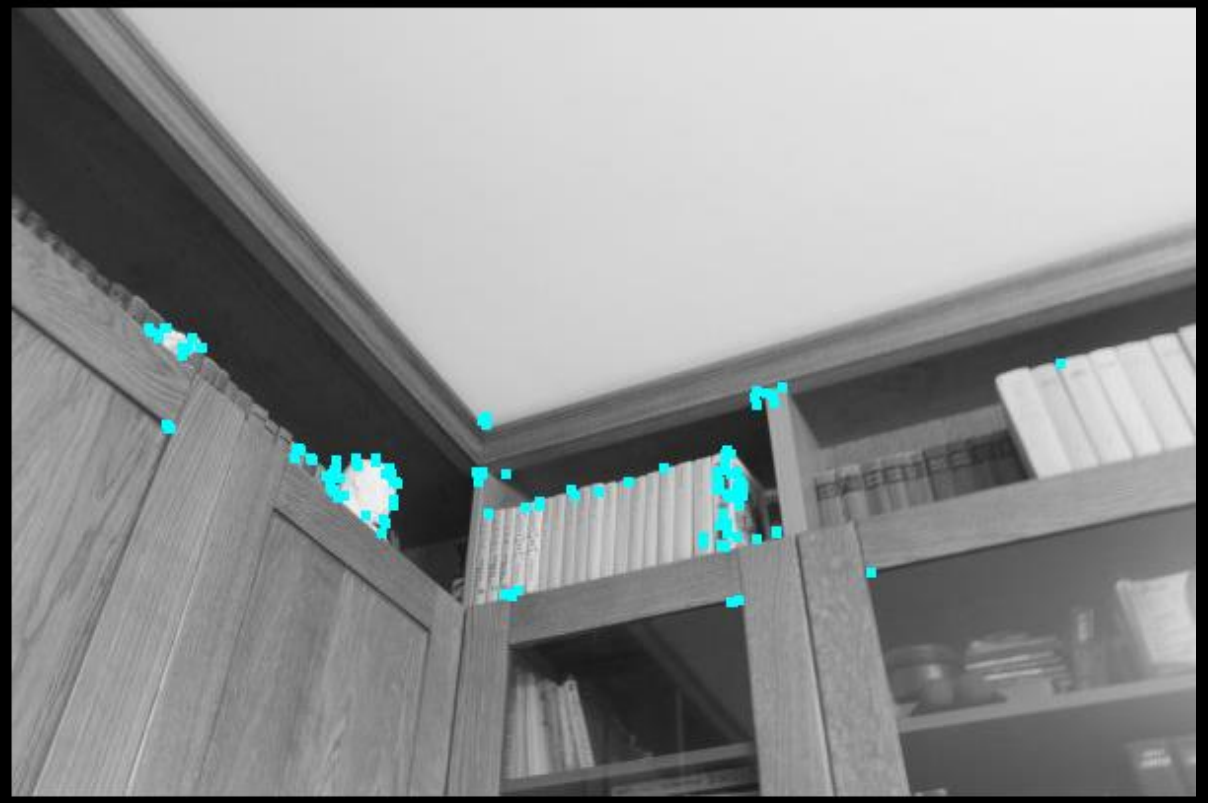
Blending: average

Blending: average

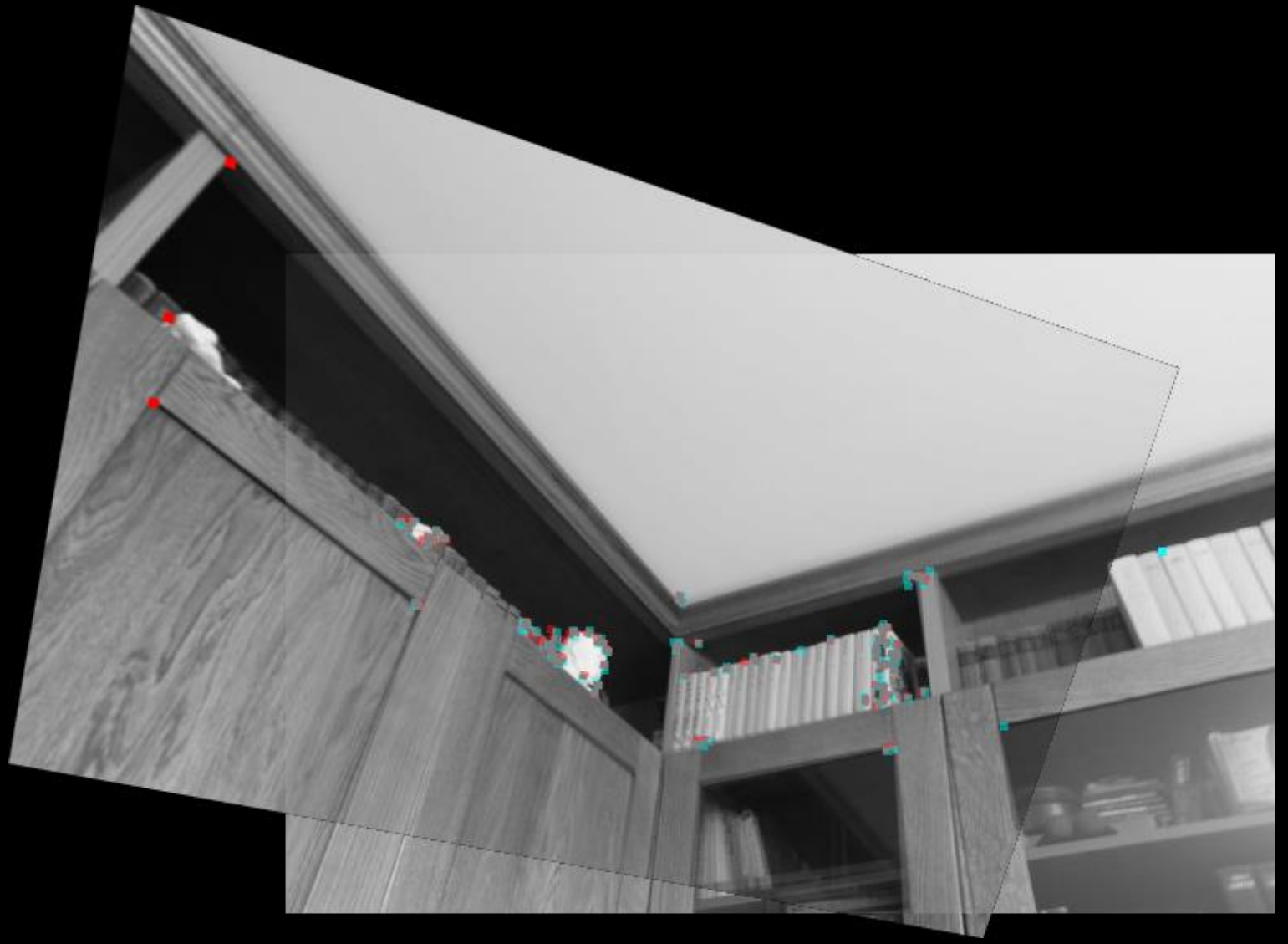
Blending: average



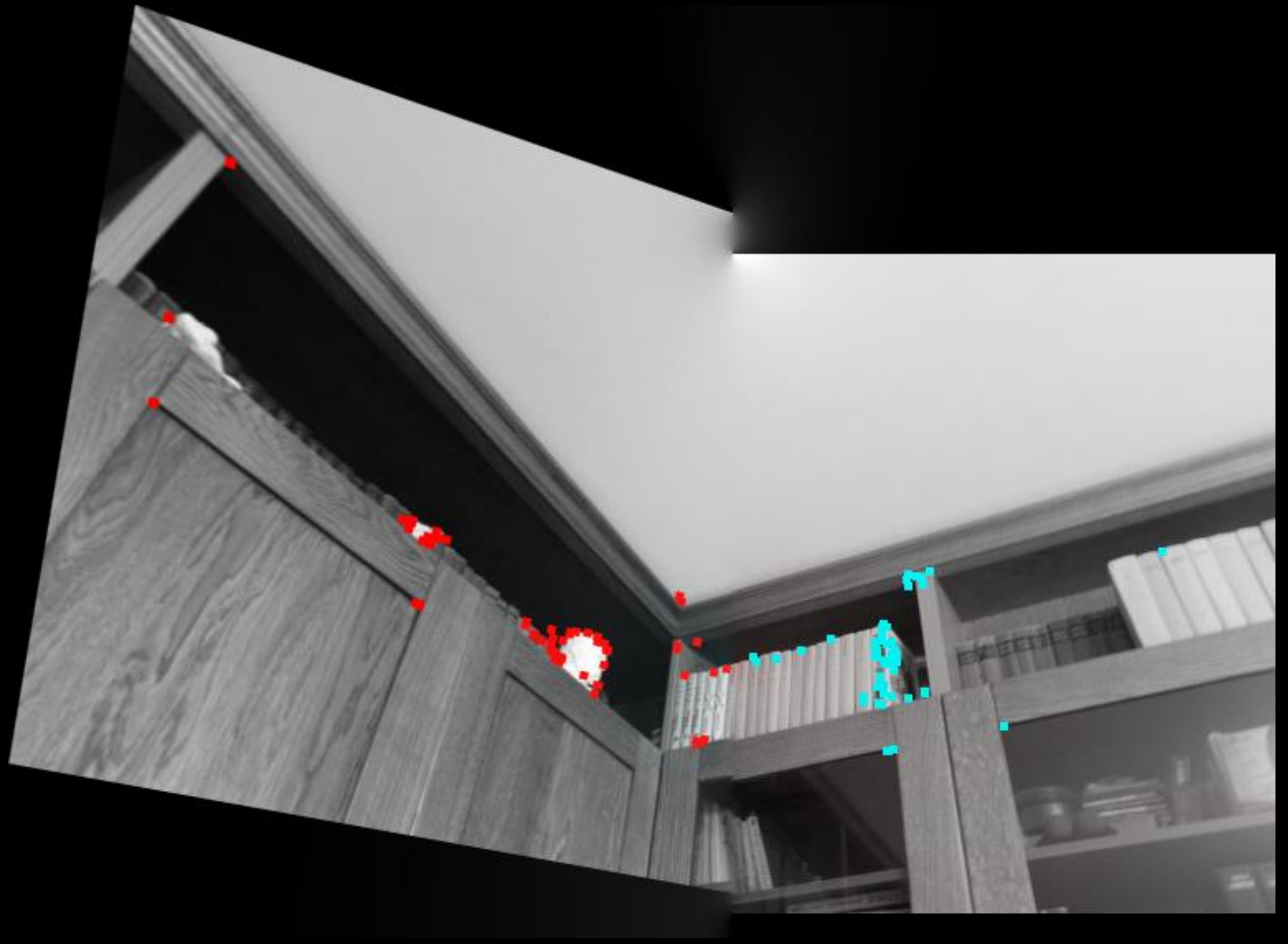
Blending: average



Blending: average



Blending: average



Blending: average

```
def blend_multi_band(left, rght, background_color=(255, 255, 255)):

    left_l, left_r, left_t, left_b = get_borders(left, background_color)
    rght_l, rght_r, rght_t, rght_b = get_borders(rght, background_color)
    border = int((left_r+rght_l)/2)

    mask = numpy.zeros(left.shape)
    mask[:, :border] = 1

    leveln = int(numpy.floor(numpy.log2(min(left.shape[0], left.shape[1]))))

    MP = GaussianPyramid(mask, leveln)
    LPA = LaplacianPyramid(numpy.array(left).astype('float'), leveln)
    LPB = LaplacianPyramid(numpy.array(rght).astype('float'), leveln)
    blended = blend_pyramid(LPA, LPB, MP)

    result = reconstruct_from_pyramid(blended)
    result[result > 255] = 255
    result[result < 0] = 0
    return result
```

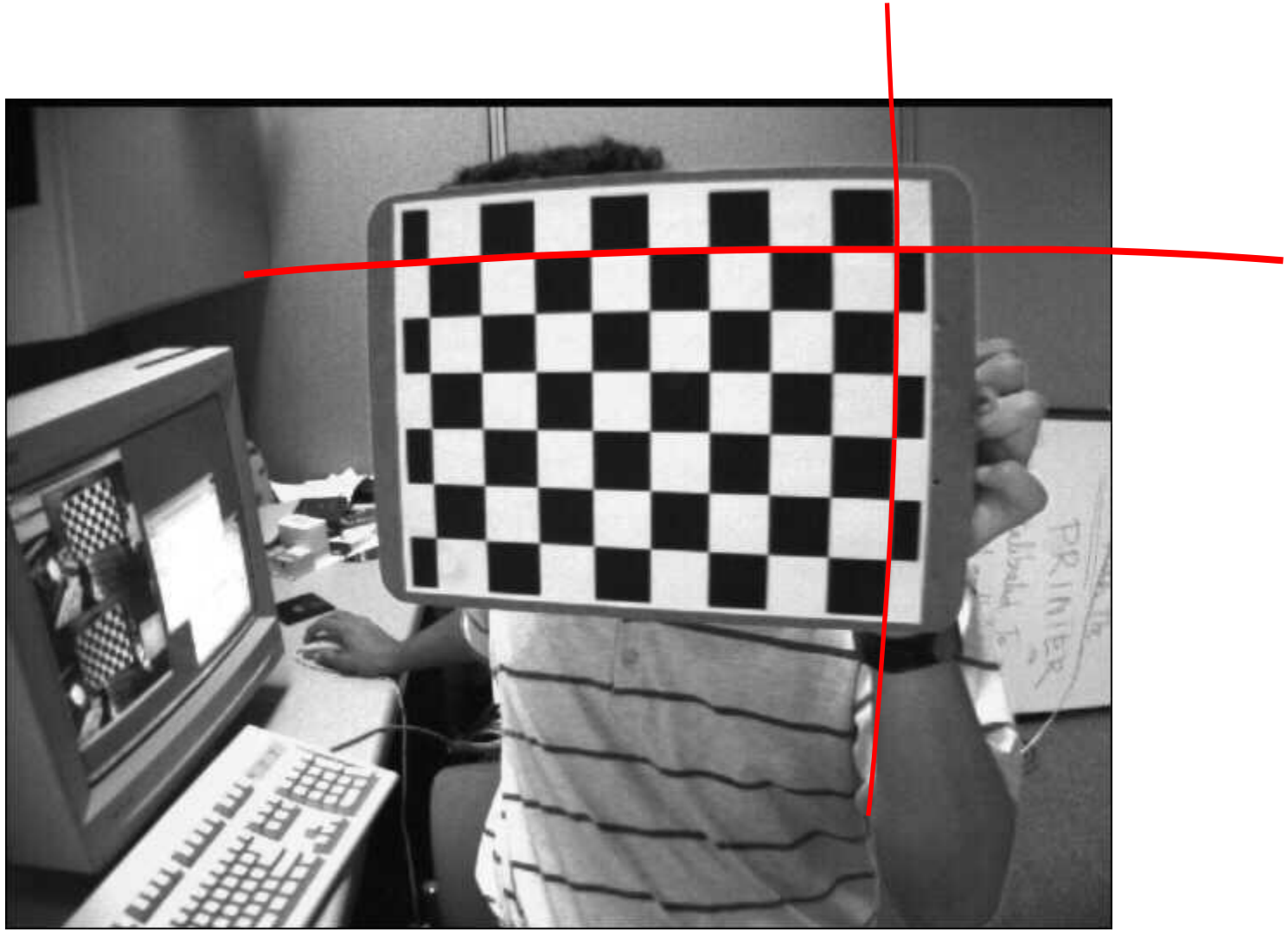
5. Panorama

5. Panorama

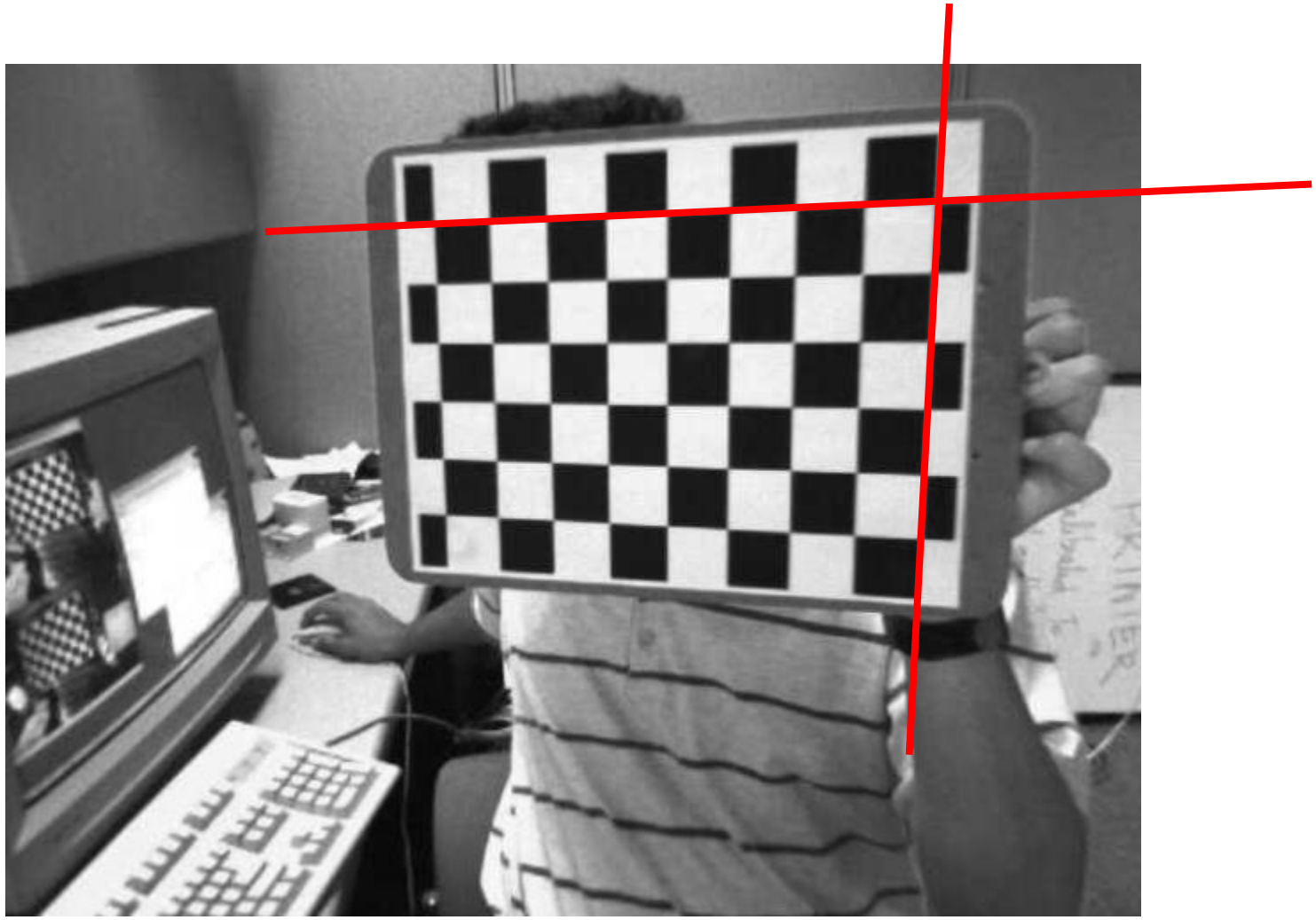
https://github.com/dibya-pati/ComputerVision/tree/master/HW2-Panoramas

6. Camera calibration

Camera calibration



Camera calibration

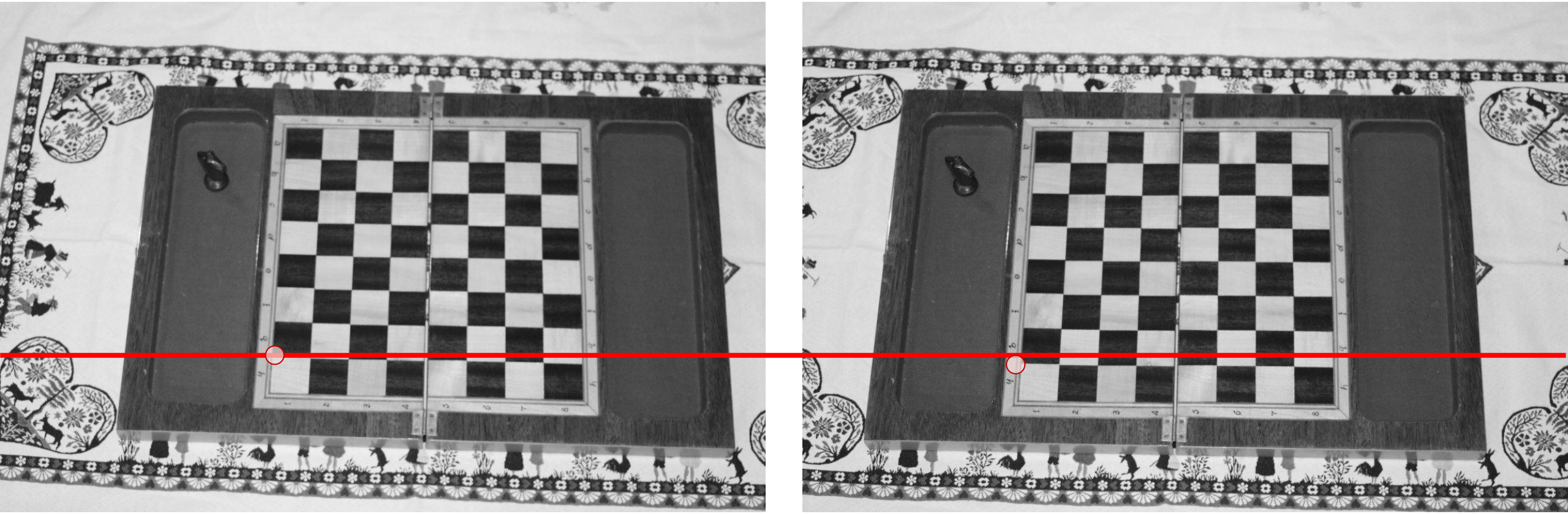


Camera calibration

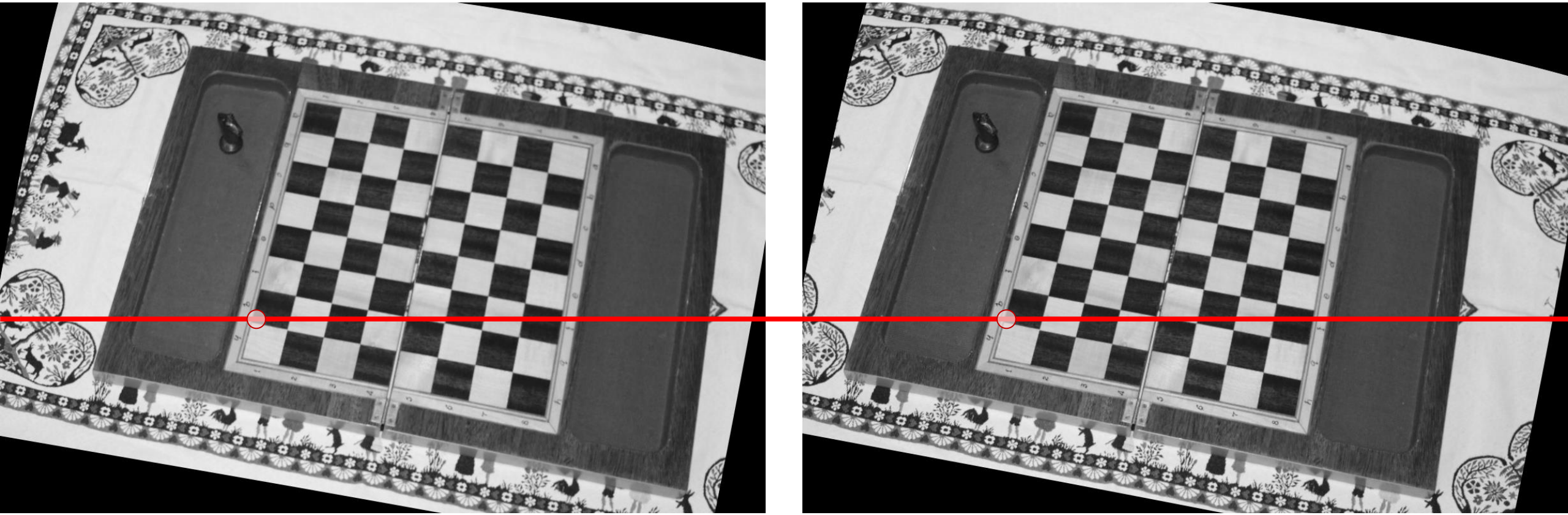
```
def example_01_calibrate_camera():

    filename_input_grid     = '_images/ex01/grid.jpg'
    folder_input_chess      = '_images/ex01/left/'
    filename_input_chess    = folder_input_chess + 'left01.jpg'

    folder_output           = '_images/ex01_out/'
    filename_output_chess   = folder_output + 'undistorted_chess.jpg'
    filename_output_grid    = folder_output + 'undistorted_grid.jpg'
    chess_rows=6
    chess_cols=7

    if not os.path.exists(folder_output):
        os.makedirs(folder_output)
    else:
        tools_IO.remove_files(folder_output)

    cameraMatrix, dist =
Calibrate.get_proj_dist_mat_for_images(folder_input_chess,chess_rows,chess_cols)
    undistorted_chess  = Calibrate.undistort_image(cameraMatrix, dist, filename_input_chess)
    undistorted_grid   = Calibrate.undistort_image(cameraMatrix, dist, filename_input_grid)
    cv2.imwrite(filename_output_chess, undistorted_chess)
    cv2.imwrite(filename_output_grid , undistorted_grid)

    return
```

7. Stereopair rectification

Stereopair rectification



Stereopair rectification

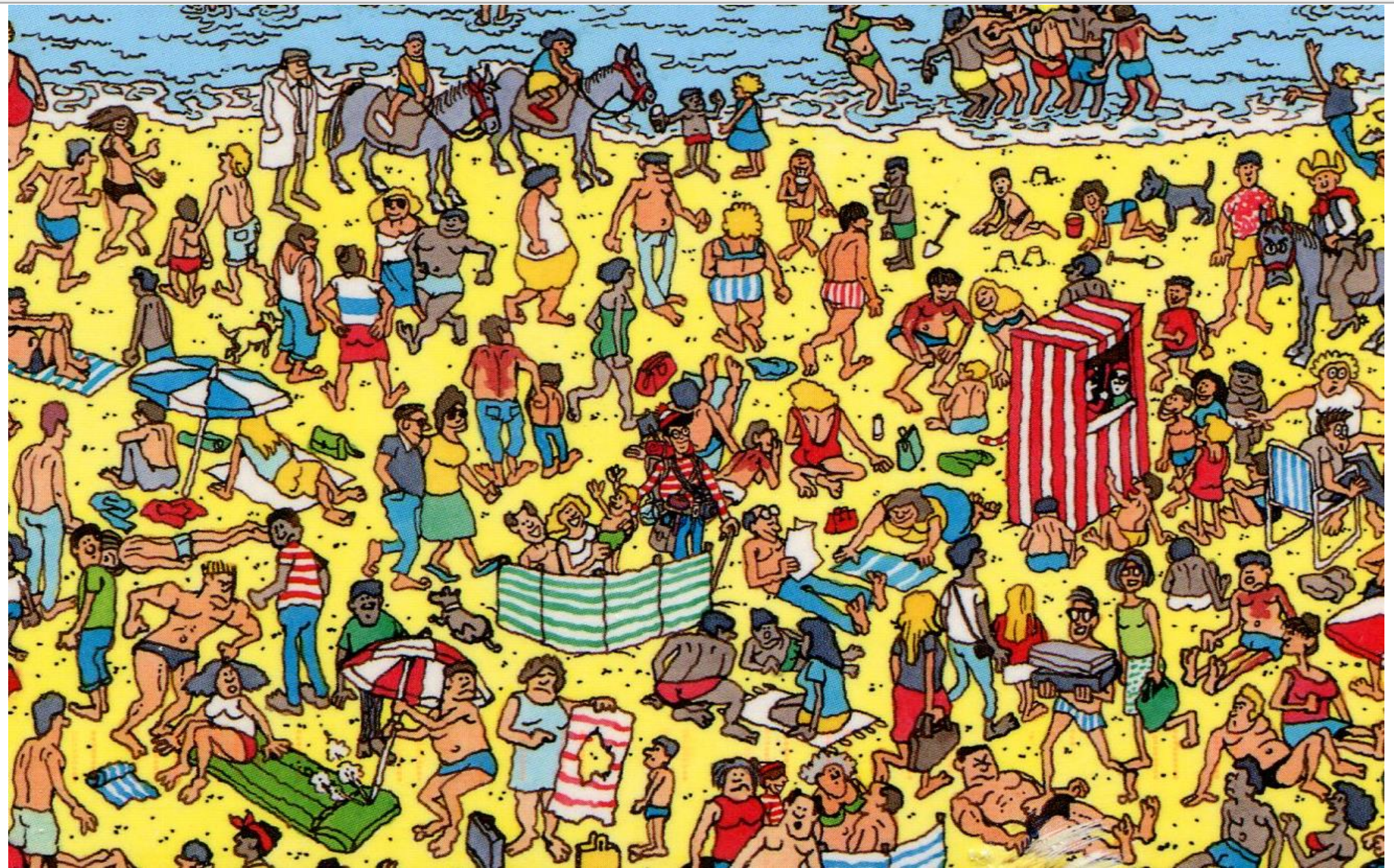


Stereopair rectification

```
def example_02_rectify_pair():

    folder_input_chess = '_images/ex01/left/'
    folder_input       = '_images/ex01/'
    filename_input_1   = folder_input + 'left/left01.jpg'
    filename_input_2   = folder_input + 'right/right01.jpg'
    folder_output      = '_images/ex01_out/'
    filename_output_1  = folder_output + 'left01_rect.jpg'
    filename_output_2  = folder_output + 'right01_rect.jpg'

    chess_rows=6
    chess_cols=7

    if not os.path.exists(folder_output):
        os.makedirs(folder_output)
    else:
        tools_IO.remove_files(folder_output)

        cameraMatrix, dist = tools_calibrate.get_proj_dist_mat_for_images(folder_input_chess, chess_rows, chess_cols)

    im1_remapped, im2_remapped = tools_calibrate.rectify_pair(cameraMatrix, dist, filename_input_1, filename_input_2, chess_rows,
chess_cols)
    cv2.imwrite(filename_output_1, im1_remapped)
    cv2.imwrite(filename_output_2, im2_remapped)

    return
```

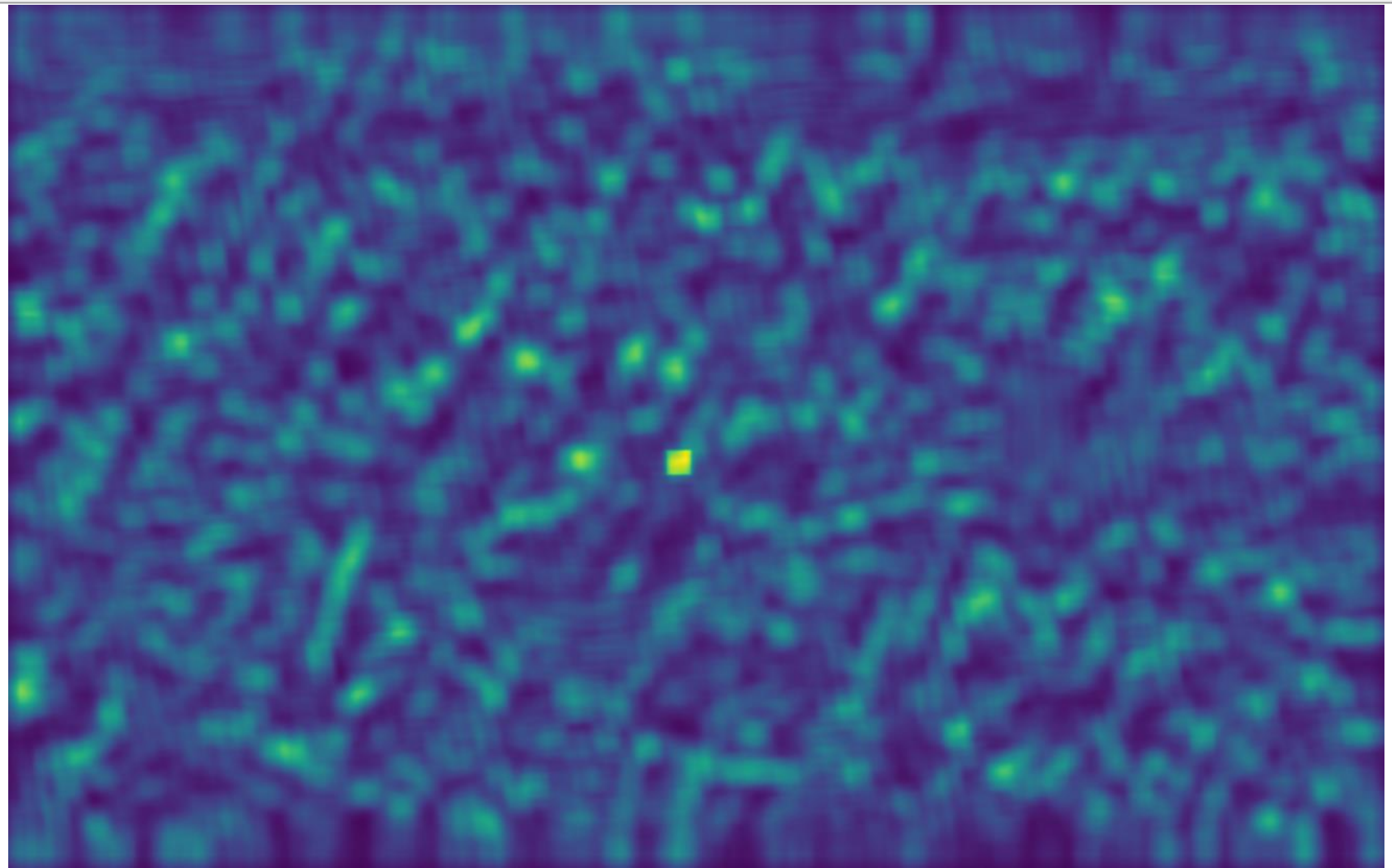
8. Template matching

Template Matching

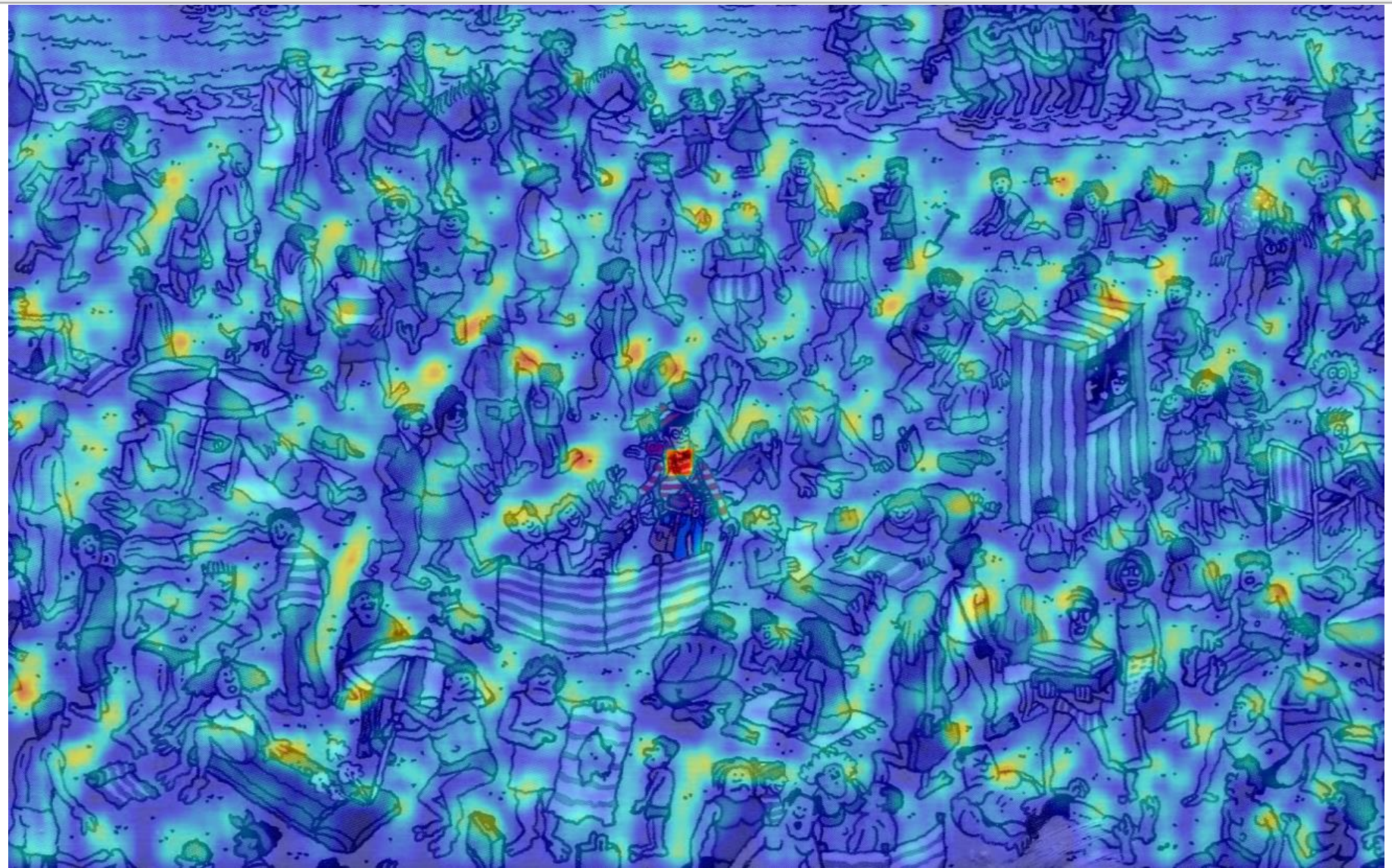


Template Matching

Template Matching



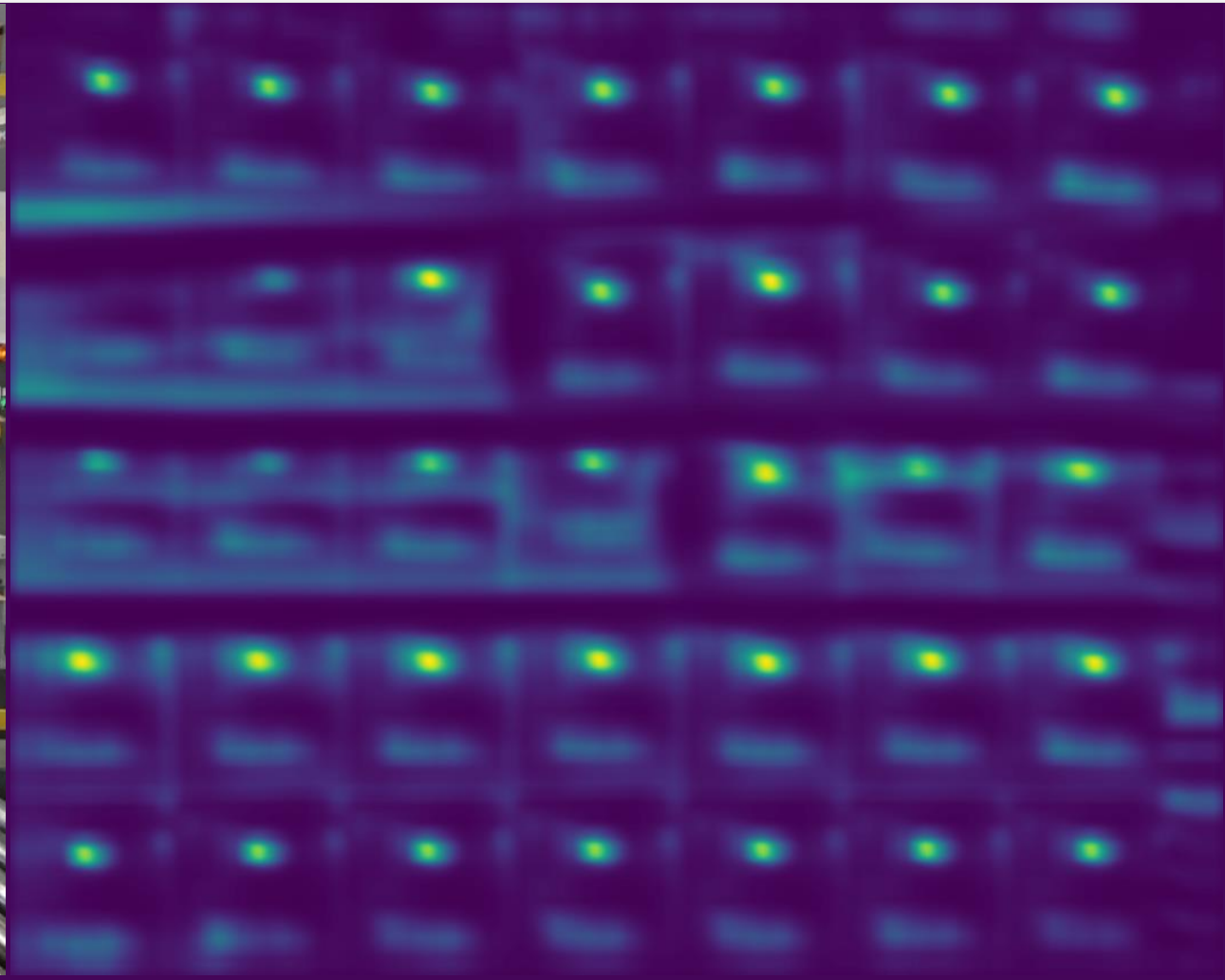
Template Matching



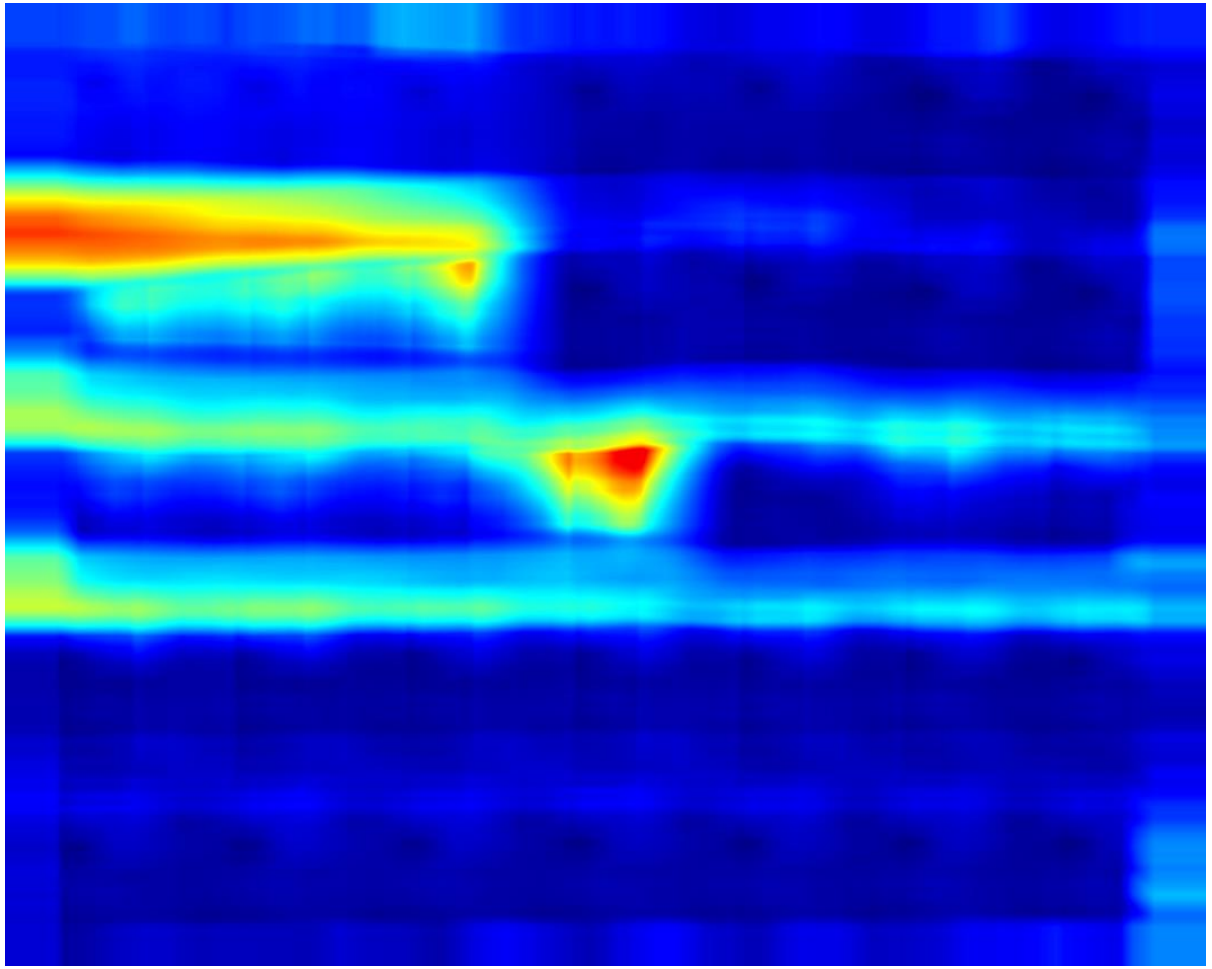
Template Matching

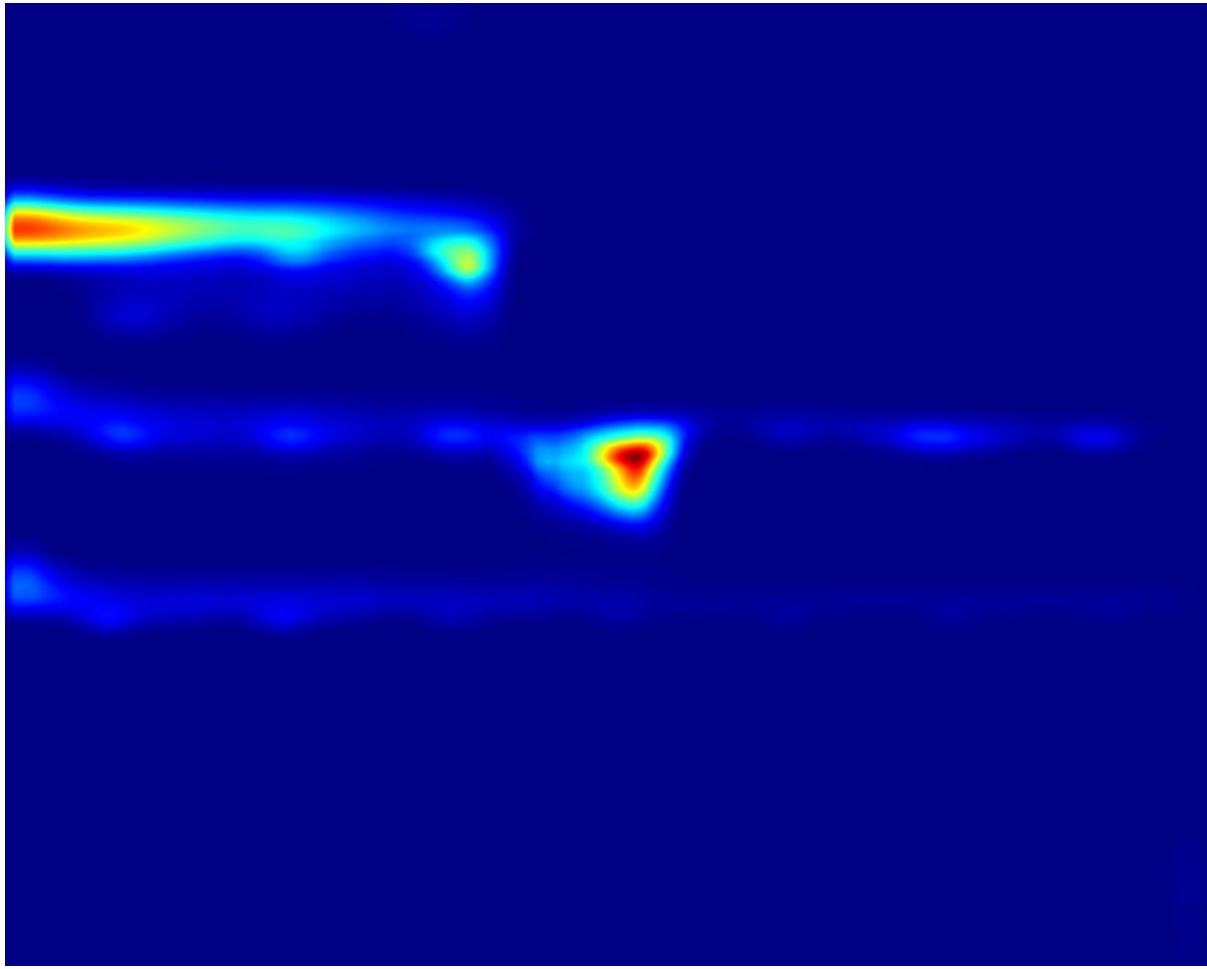
Template Matching

Template Matching

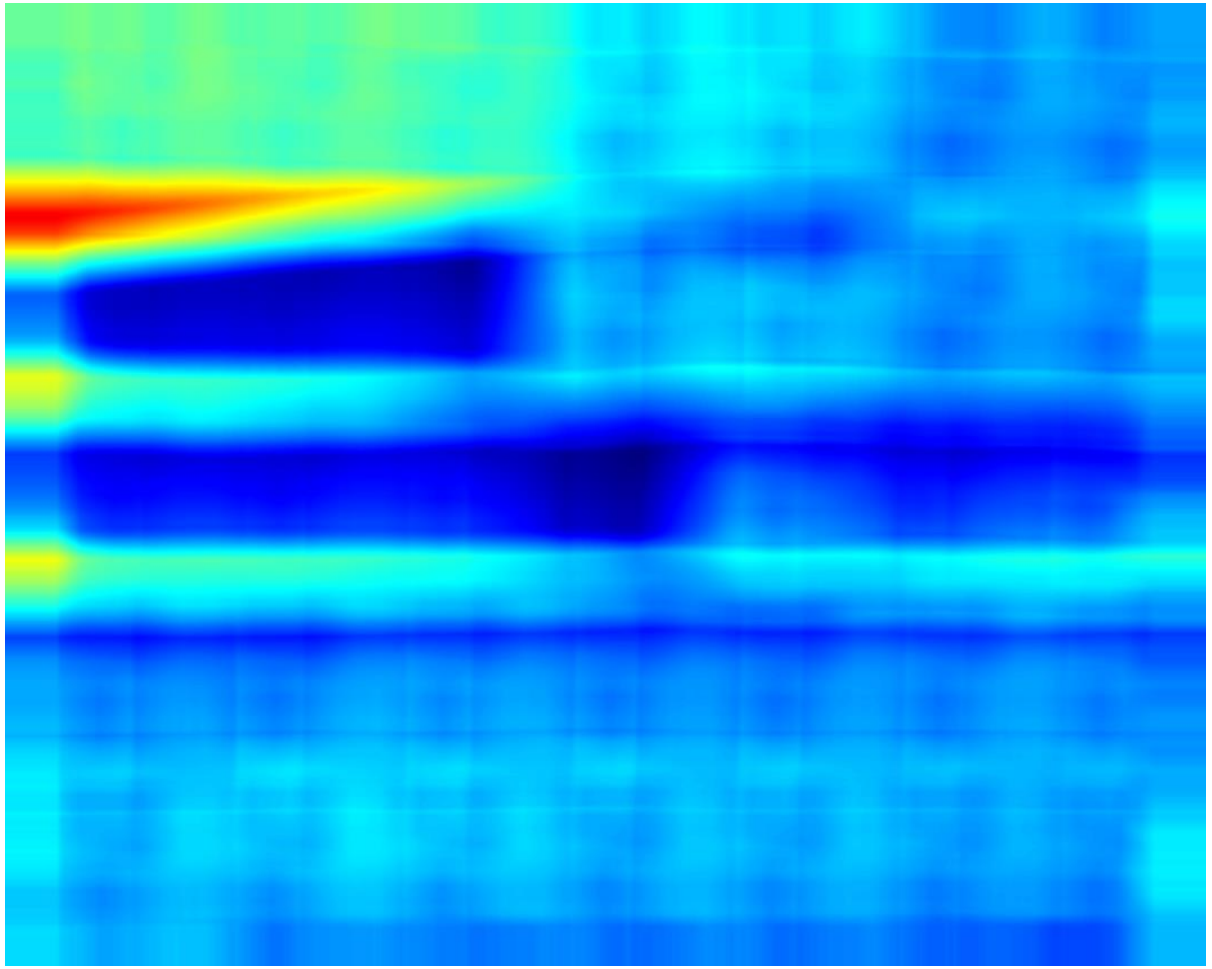


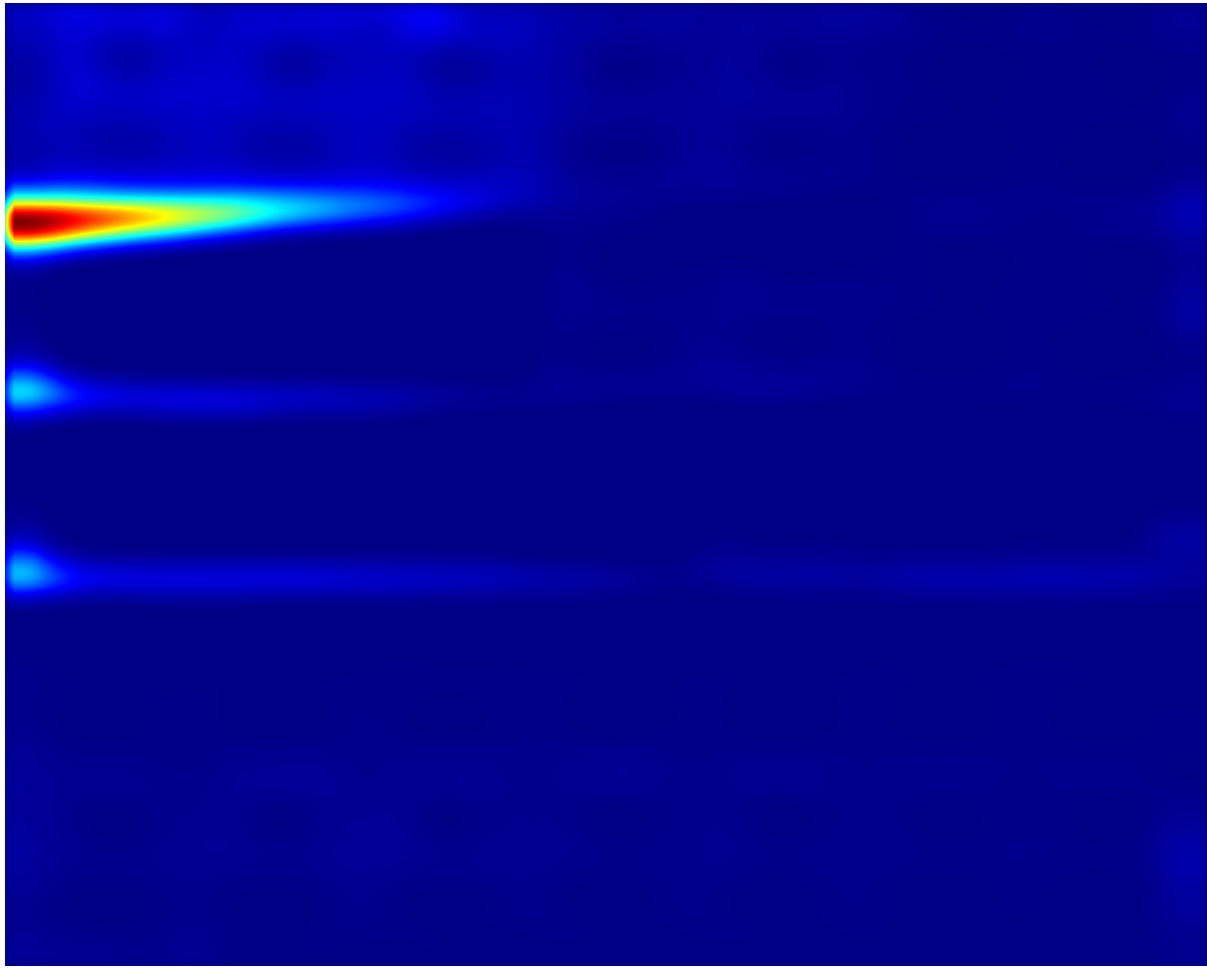
Template Matching: TM_SQDIFF_NORMED



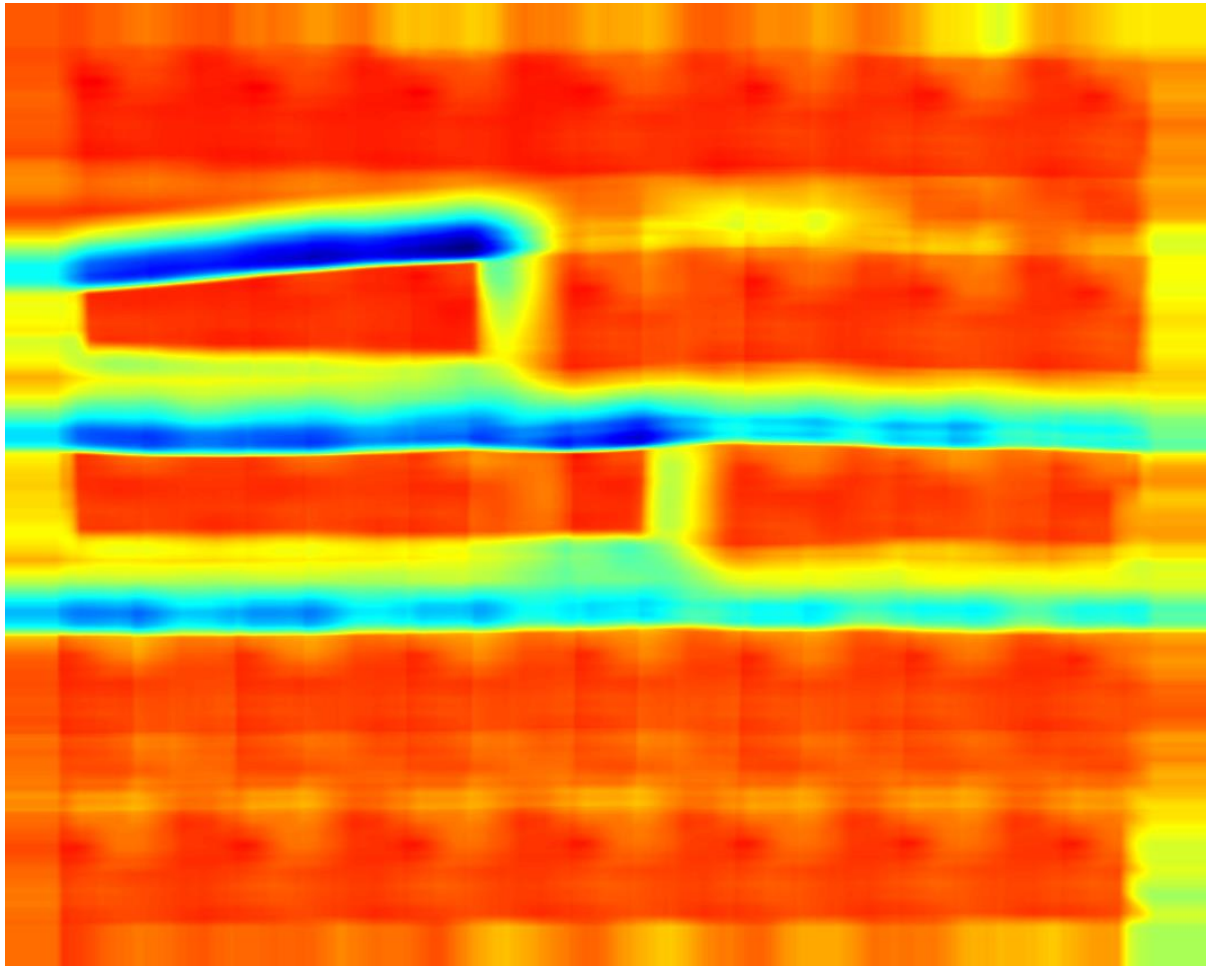


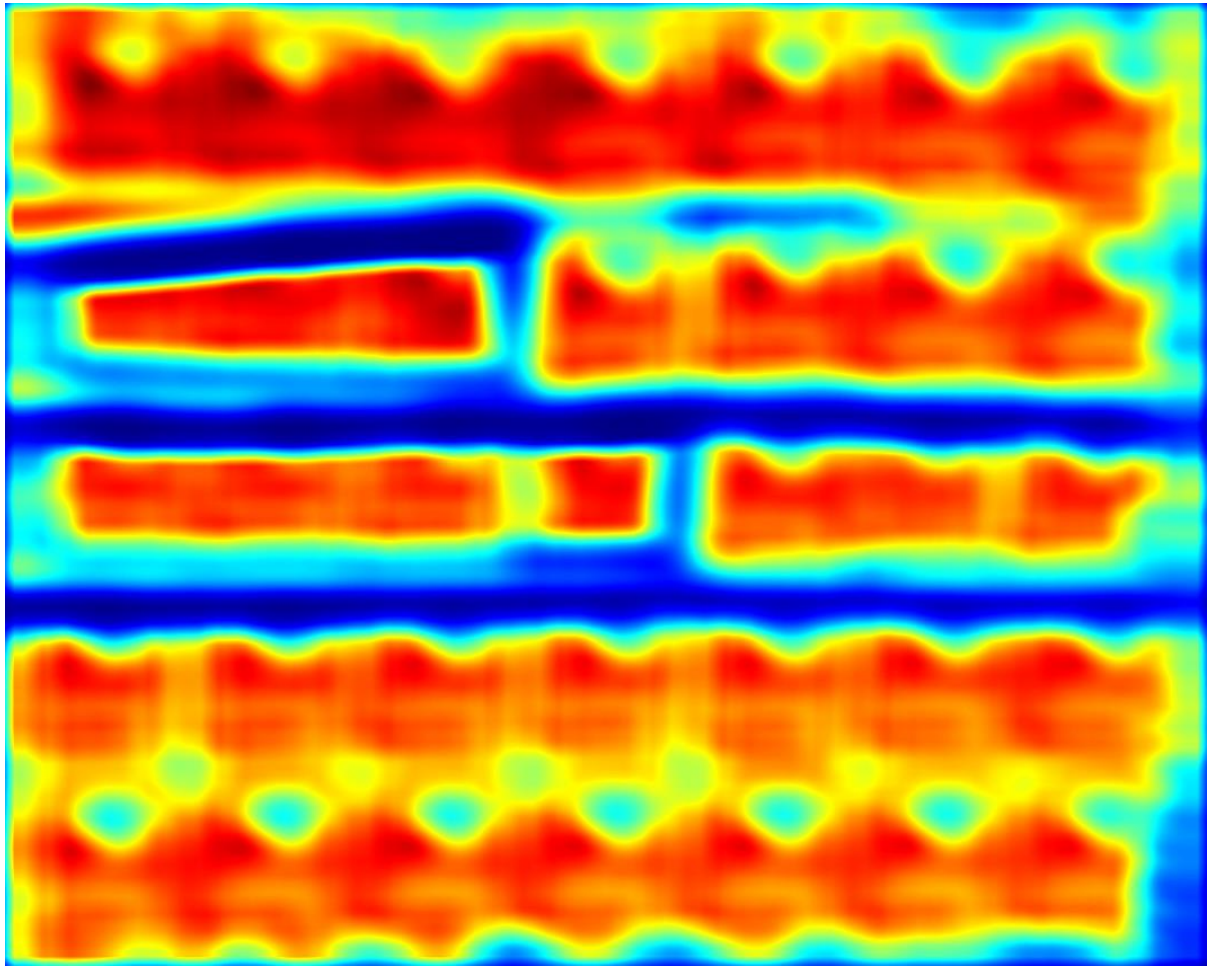
Template Matching: TM_CCORR



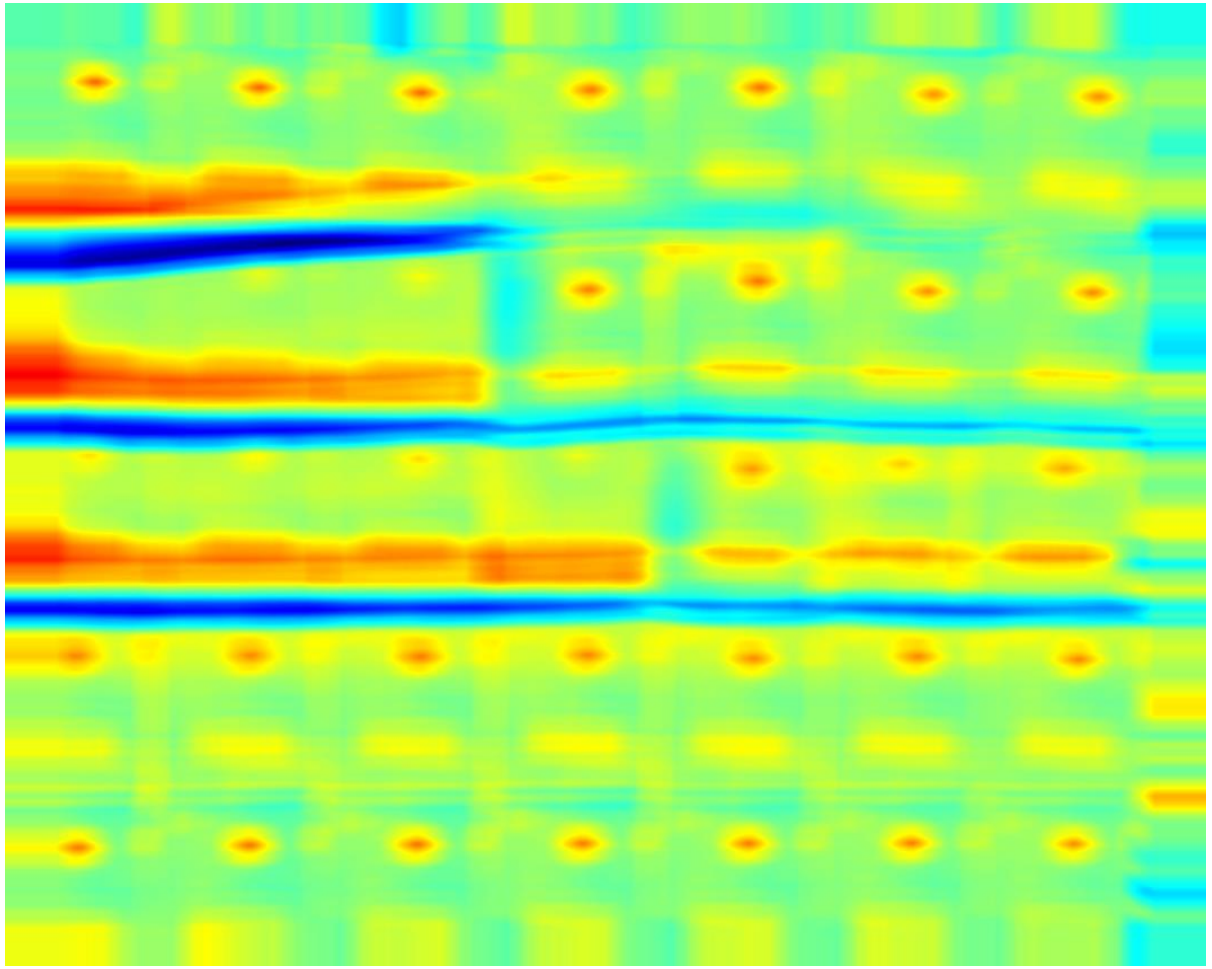


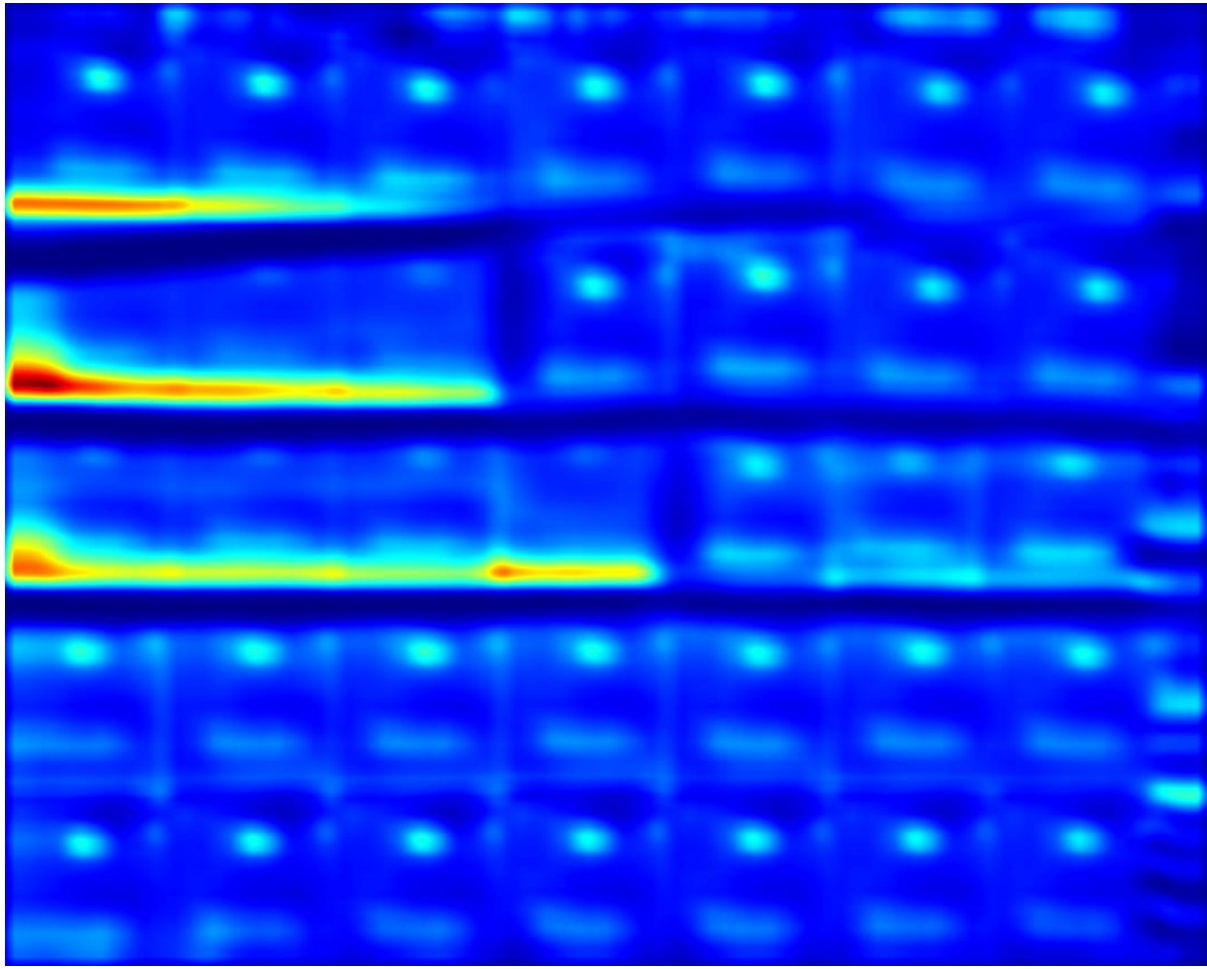
Template Matching: TM_CCORR_NORMED



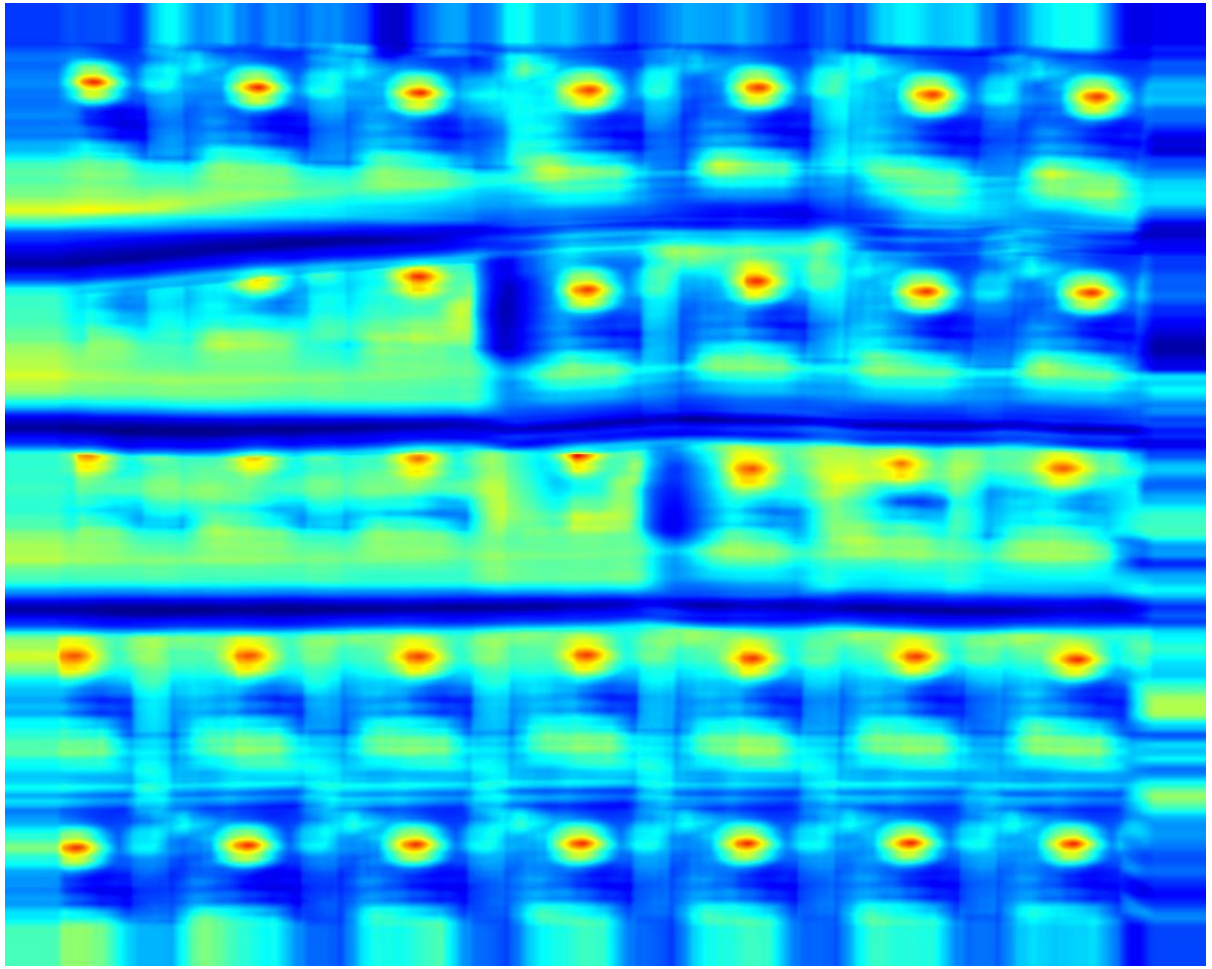


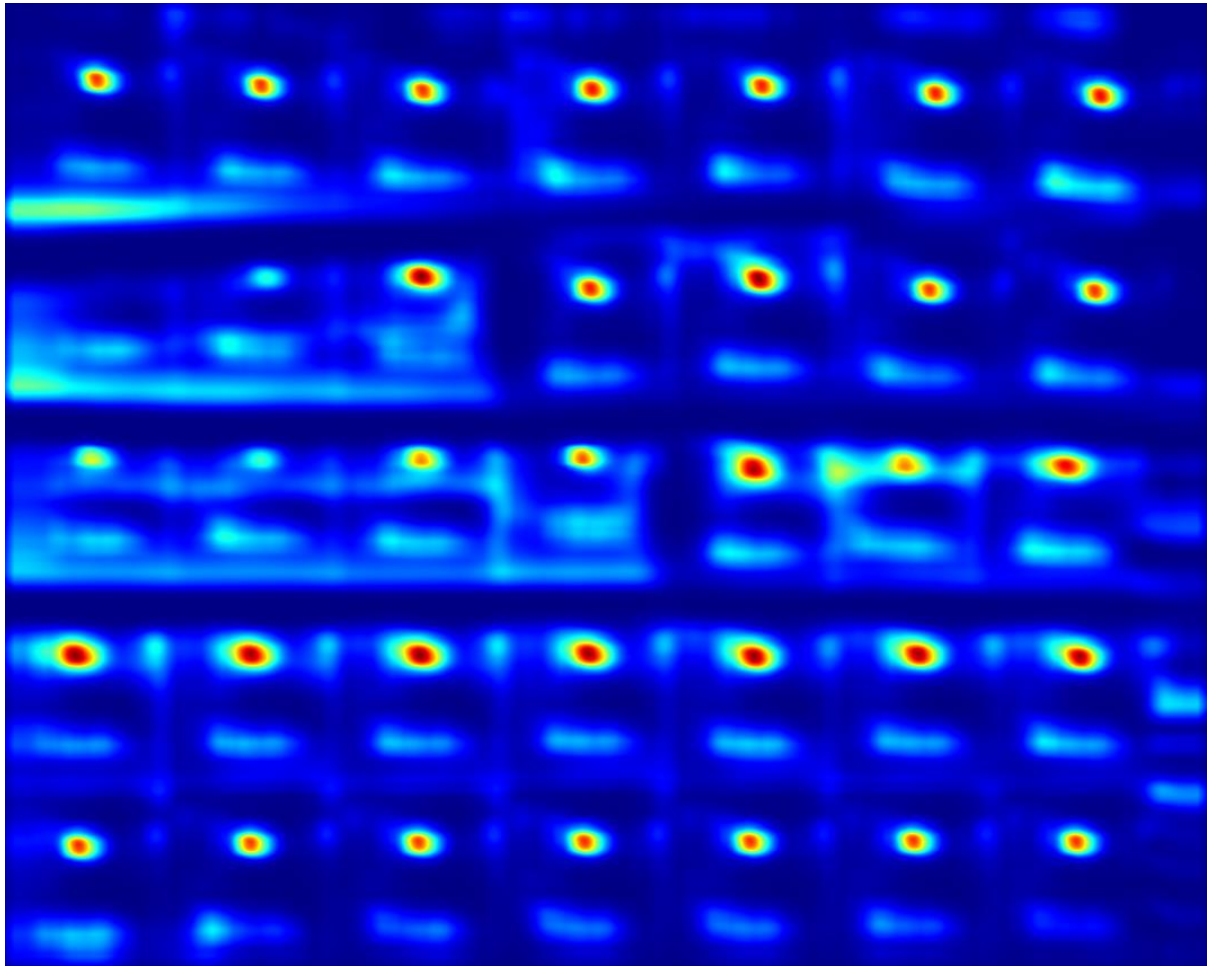
Template Matching: TM_CCOEFF





Template Matching: TM_CCOEFF_NORMED





Template Matching

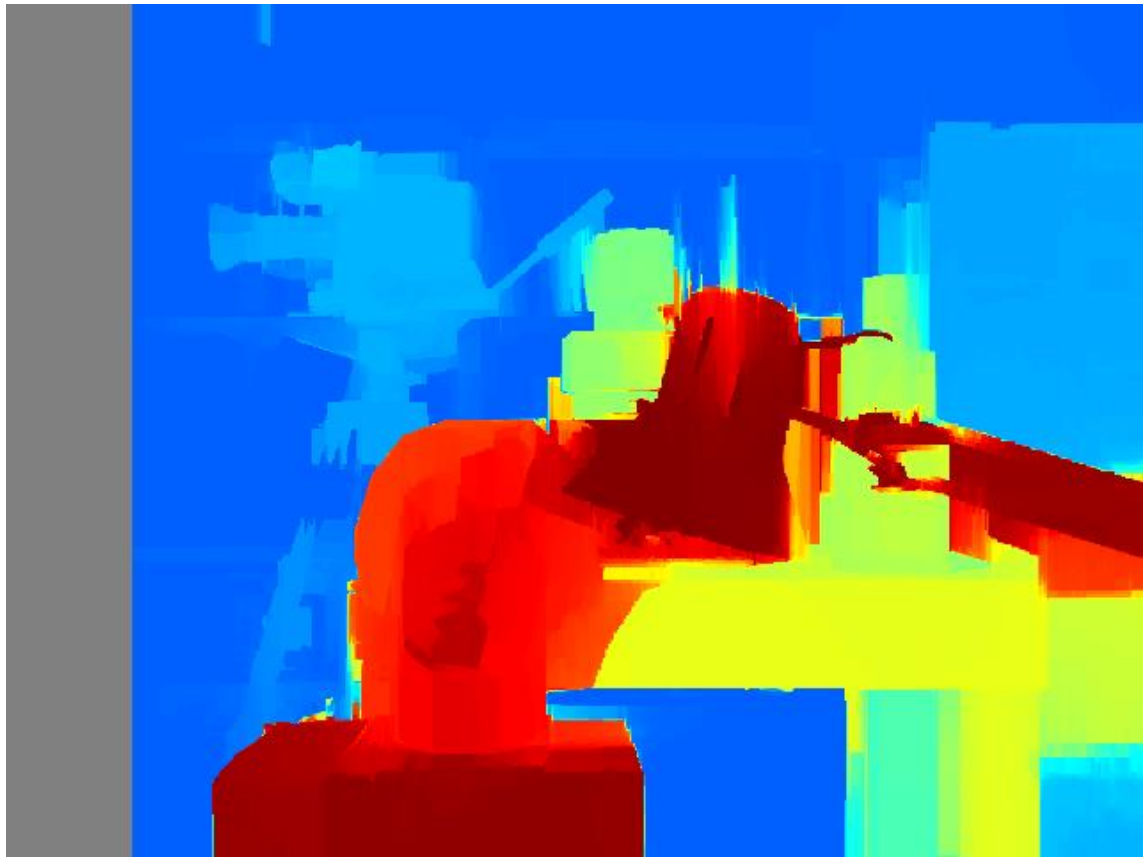
```
def calc_hit_field_basic(image_base, image_pattern,mask=None):

    hitmap_gray = cv2.matchTemplate(image_base, image_pattern, method=cv2.TM_CCOEFF_NORMED)

    min = hitmap_gray.min()
    max = hitmap_gray.max()
    hitmap_gray = (225 * (hitmap_gray - min) / (max - min)).astype(numpy.uint8)
    hitmap_gray = numpy.reshape(hitmap_gray,(hitmap_gray.shape[0],hitmap_gray.shape[1],1))

    hitmap_gray2 = numpy.zeros((image_base.shape[0],image_base.shape[1],1)).astype(numpy.uint8)
    shift_x = int((image_base.shape[0] - hitmap_gray.shape[0])/2)
    shift_y = int((image_base.shape[1] - hitmap_gray.shape[1])/2)
    hitmap_gray2[shift_x:shift_x+hitmap_gray.shape[0], shift_y:shift_y + hitmap_gray.shape[1], 0] =
hitmap_gray[: , :, 0]


    hitmap_gray2[:shift_x                              , shift_y:shift_y + hitmap_gray.shape[1], 0] =
hitmap_gray[0 , :, 0]
    hitmap_gray2[shift_x + hitmap_gray.shape[0]:       , shift_y:shift_y + hitmap_gray.shape[1], 0] =
hitmap_gray[-1, :, 0]

    for row in range(0,hitmap_gray2.shape[0]):
        hitmap_gray2[row,:shift_y                            , 0] = hitmap_gray2[row, shift_y
, 0]
        hitmap_gray2[row, shift_y + hitmap_gray.shape[1]:, 0] = hitmap_gray2[row, shift_y +
hitmap_gray.shape[1]-1 , 0]


    hitmap_2d = numpy.reshape(hitmap_gray2, (hitmap_gray2.shape[0], hitmap_gray2.shape[1]))
```

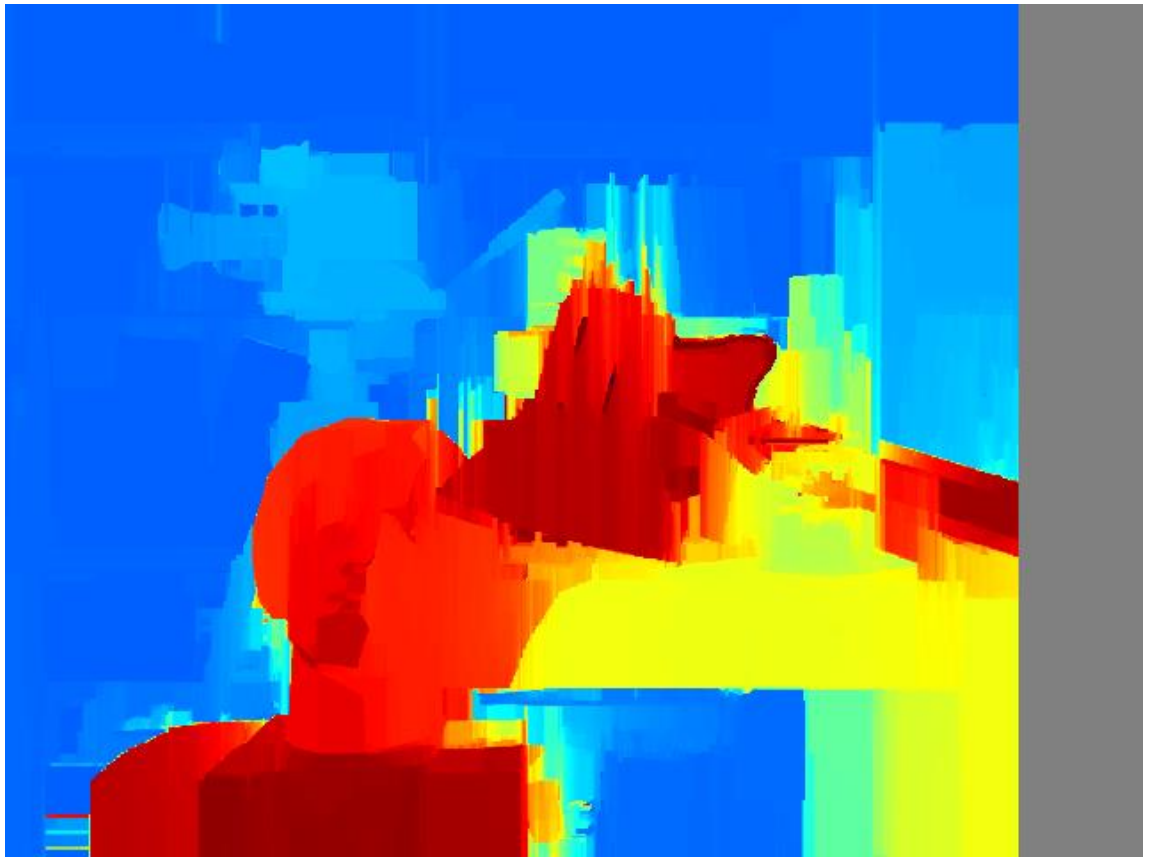
9. Stereo Vision

Stereo vision

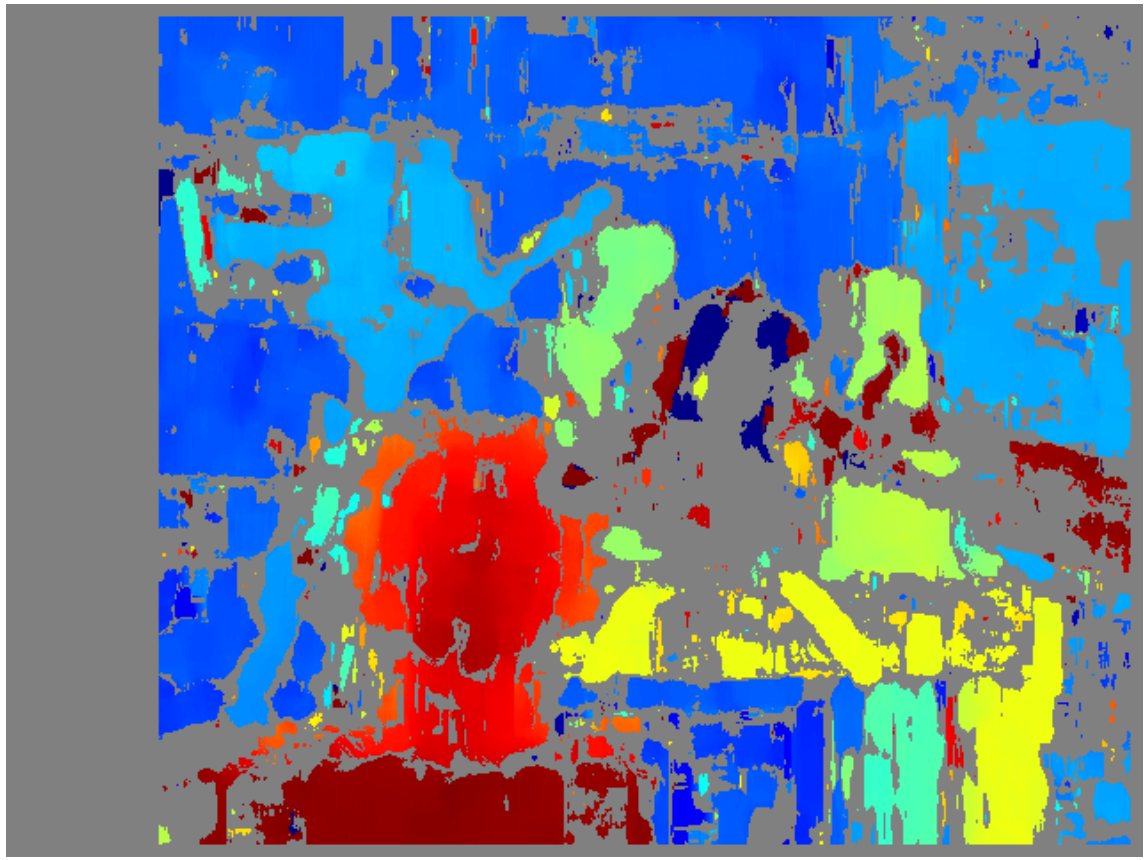
Stereo vision

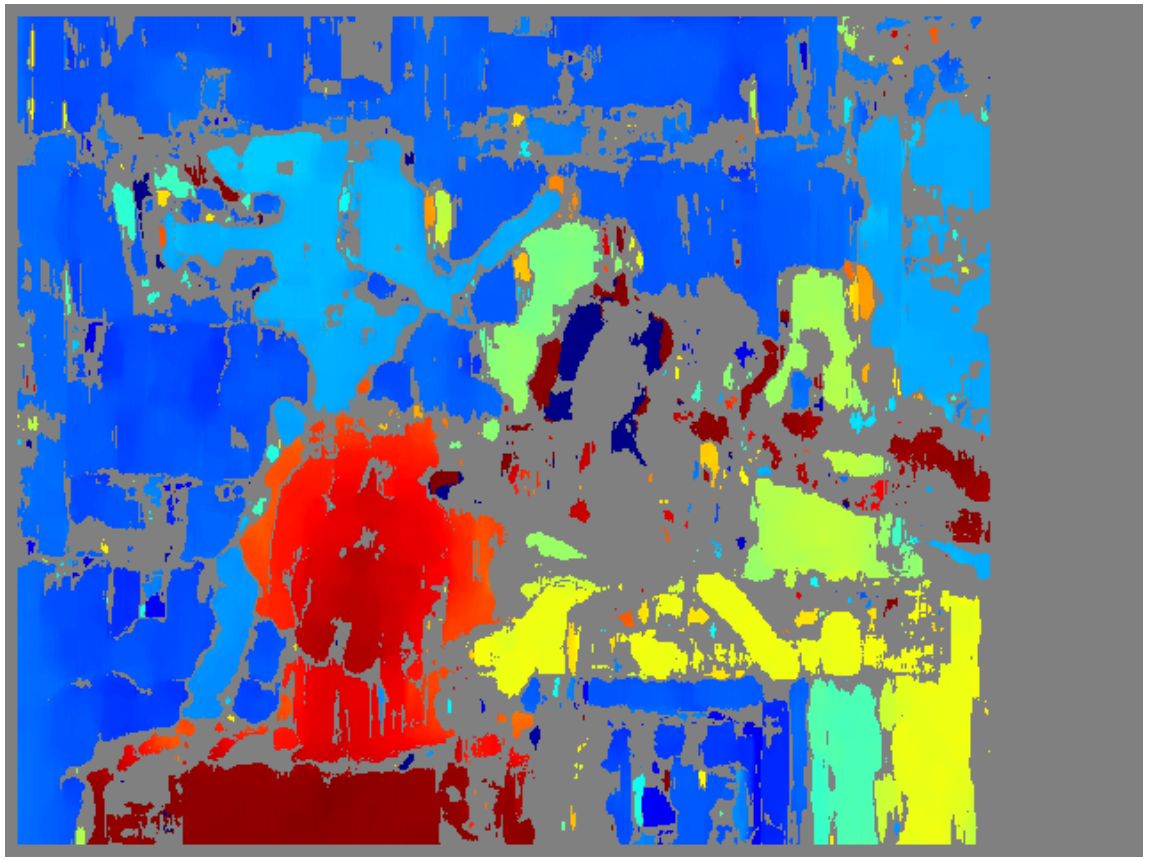
Stereo vision





Stereo vision





Stereo vision

```
def get_disparity_v_01(imgL, imgR, disp_v1, disp_v2, disp_h1, disp_h2):
    window_size = 7
    left_matcher = cv2.StereoSGBM_create(
        minDisparity=disp_h1,
        numDisparities=int(0 + (disp_h2 - disp_h1) / 16) * 16,
        blockSize=5,
        P1=8 * 3 * window_size ** 2,
        P2=32 * 3 * window_size ** 2,
        disp12MaxDiff=disp_h2,
        uniquenessRatio=15,
        speckleWindowSize=0,
        speckleRange=2,
        preFilterCap=63,
        mode=cv2.STEREO_SGBM_MODE_SGBM_3WAY
    )

    right_matcher = cv2.ximgproc.createRightMatcher(left_matcher)
    dispr = right_matcher.compute(imgR, imgL)
    displ = left_matcher.compute(imgL, imgR)

    wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=left_matcher)
    wls_filter.setLambda(80000)
    wls_filter.setSigmaColor(1.2)

    filteredImg_L = wls_filter.filter(displ, imgL, None, dispr)
    wls_filter = cv2.ximgproc.createDisparityWLSFilter(matcher_left=right_matcher)
    filteredImg_R = wls_filter.filter(dispr, imgR, None, displ)
```

Stereo vision

```
def get_disparity_v_02(imgL, imgR, disp_v1, disp_v2, disp_h1, disp_h2):
    max = numpy.maximum(math.fabs(disp_h1), math.fabs(disp_h2))
    levels = int(1 + (max) / 16) * 16
    stereo = cv2.StereoBM_create(numDisparities=levels, blockSize=15)
    displ = stereo.compute(imgL, imgR)

    dispr = numpy.flip(stereo.compute(numpy.flip(imgR, axis=1), numpy.flip(imgL, axis=1)), axis=1)
    return -displ / 16, -dispr / 16
```

10. Pose estimation

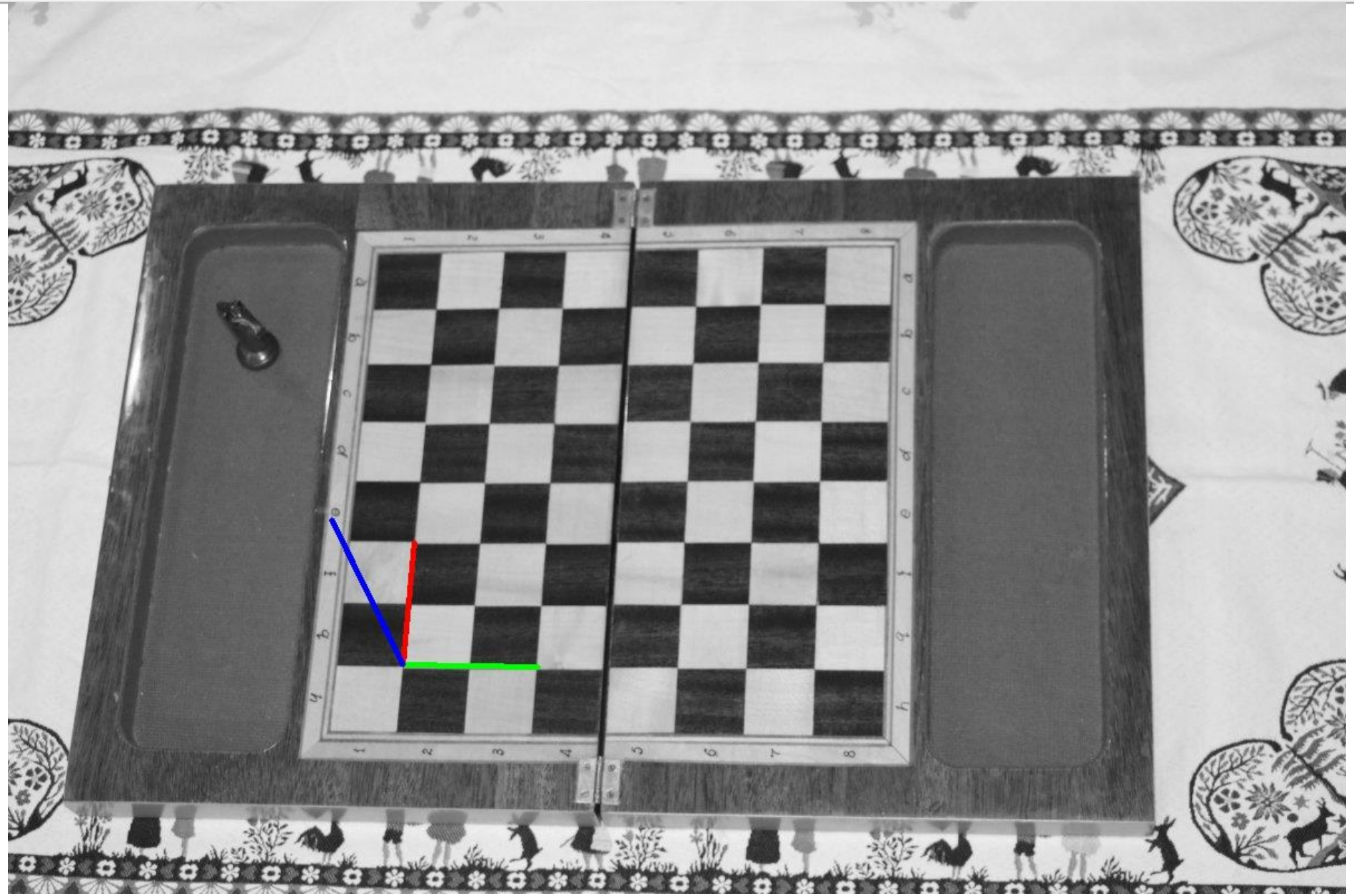
Pose estimation

```
def get_rvecs_tvecs(img,chess_rows, chess_cols,cameraMatrix, dist):
    corners_3d = numpy.zeros((chess_rows * chess_cols, 3), numpy.float32)
    corners_3d[:, :2] = numpy.mgrid[0:chess_cols, 0:chess_rows].T.reshape(-1, 2)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners_2d = cv2.findChessboardCorners(gray, (chess_cols, chess_rows), None)

    rvecs, tvecs=numpy.array([]),numpy.array([])

    if ret == True:
        corners_2d = cv2.cornerSubPix(gray, corners_2d, (11, 11), (-1, -1),(cv2.TERM_CRITERIA_EPS, 30,
0.001))
        _, rvecs, tvecs, inliers = cv2.solvePnPRansac(corners_3d, corners_2d, cameraMatrix, dist)

    return rvecs, tvecs
```

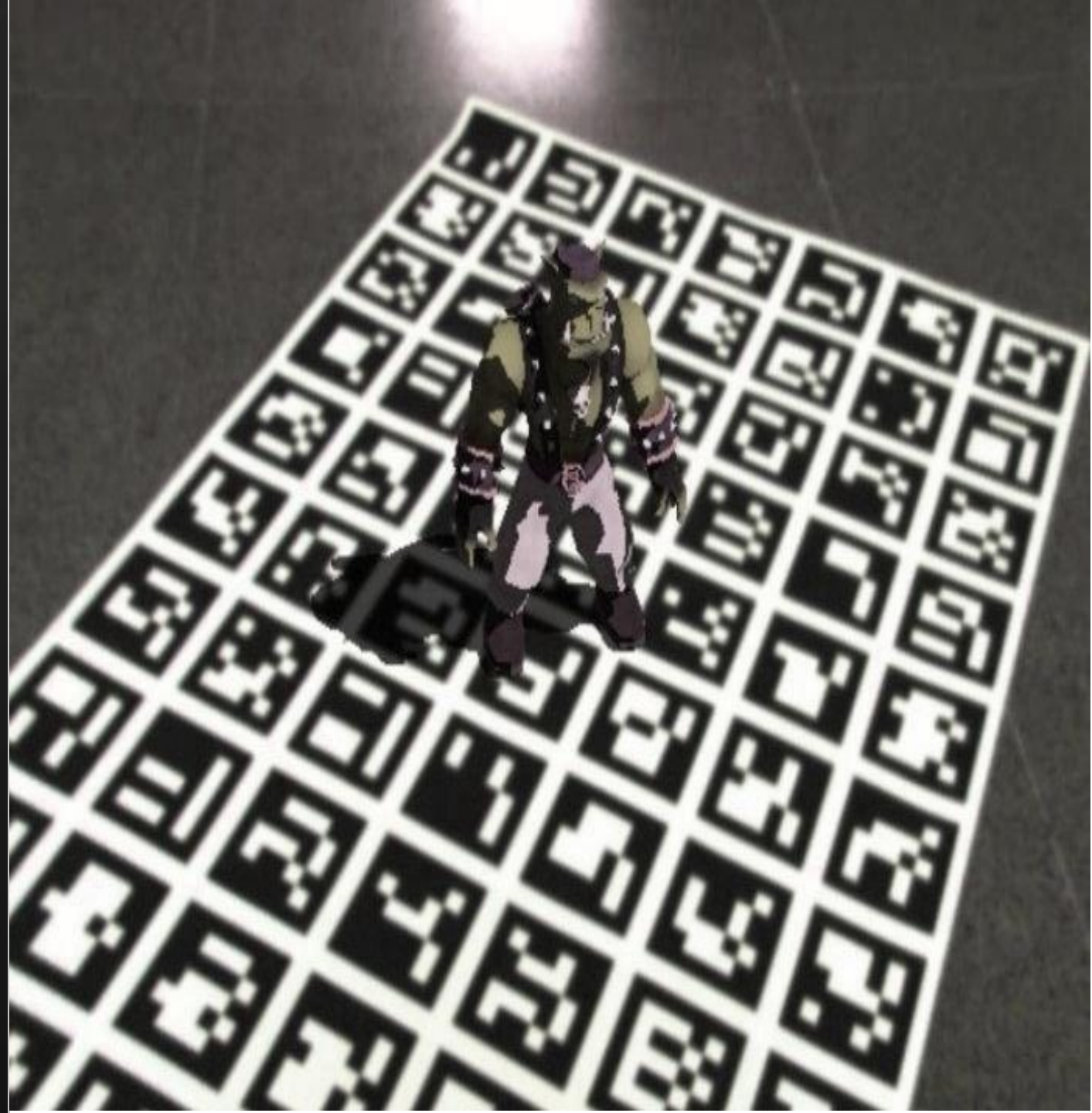
Pose estimation

```
img = cv2.imread(folder_input + image_name)
rvecs, tvecs = tools_calibrate.get_rvecs_tvecs(img,chess_rows, chess_cols,cameraMatrix, dist)

if rvecs.size!=0:

    axis_2d_end, jac   = cv2.projectPoints(axis_3d_end  , rvecs, tvecs, cameraMatrix, dist)
    axis_2d_start, jac = cv2.projectPoints(axis_3d_start, rvecs, tvecs, cameraMatrix, dist)
    img = tools_draw_numpy.draw_line(img, axis_2d_start[0,0,1], axis_2d_start[0,0,0], axis_2d_end[0,0,1],axis_2d_end[0,0,0],
(0,0,255))
    img = tools_draw_numpy.draw_line(img, axis_2d_start[0,0,1], axis_2d_start[0,0,0], axis_2d_end[1,0,1],axis_2d_end[1,0,0],
(0,0,255))
    img = tools_draw_numpy.draw_line(img, axis_2d_start[0,0,1], axis_2d_start[0,0,0], axis_2d_end[2,0,1],axis_2d_end[2,0,0],
(0,0,255))
```

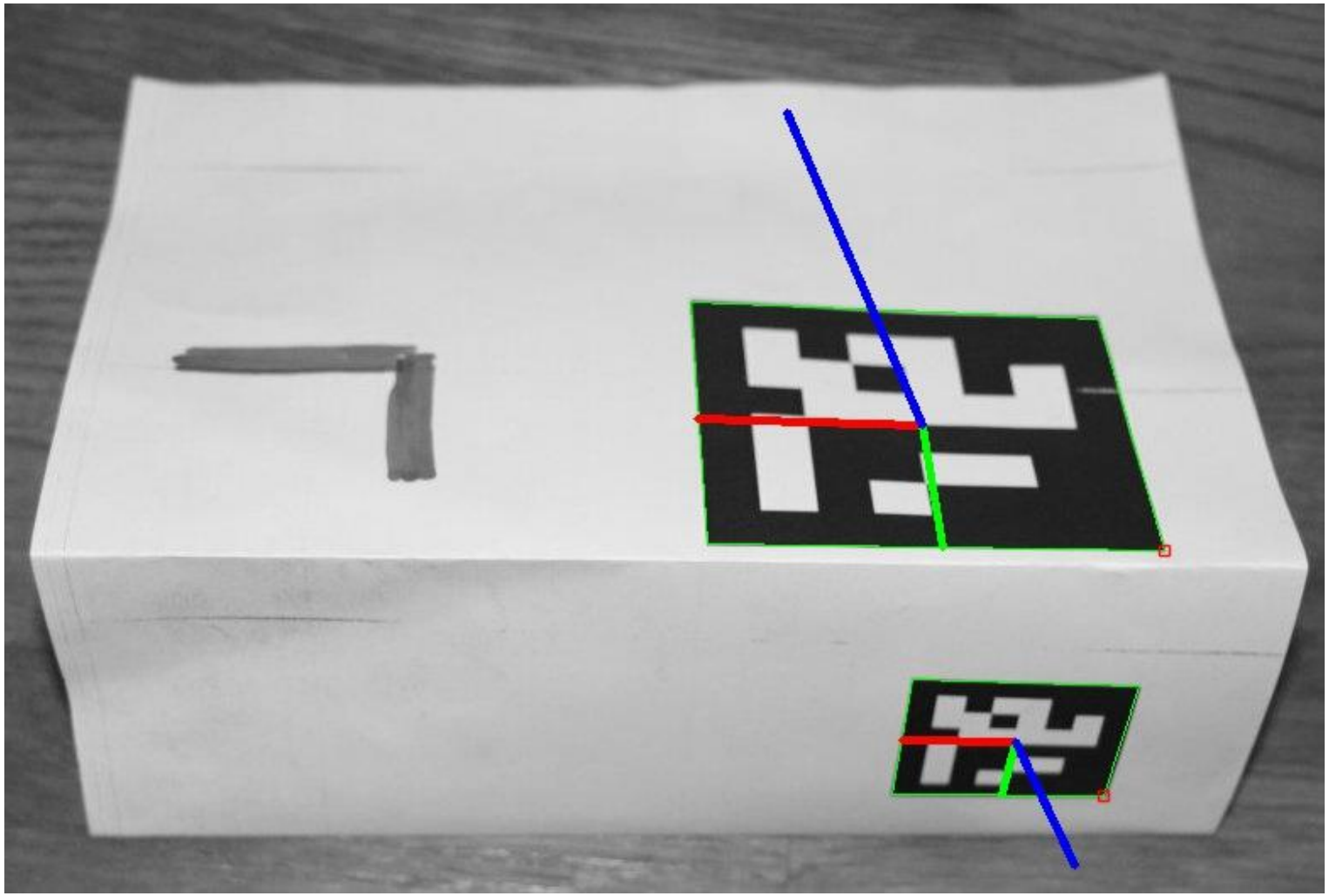
Pose estimation



11. Aruco



Aruco



Aruco

```
def demo_aruco_01(camera_matrix,dist):

    cap = cv2.VideoCapture(0)
    aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)

    while (True):
        ret, frame = cap.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        corners, ids, rejectedImgPoints = aruco.detectMarkers(gray, aruco_dict)

        if len(corners) > 0:
            frame = aruco.drawDetectedMarkers(frame, corners,ids)

            rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners[0], 0.05, camera_matrix,dist)
            aruco.drawAxis(frame, camera_matrix, dist, rvec[0], tvec[0], 0.1)

        cv2.imshow('frame', frame)
        if cv2.waitKey(1) & 0xFF == 27:
            break


    cap.release()
    cv2.destroyAllWindows()


    return
```

12. Augmented reality

Augmented Reality

```
frame = cv2.imread(filename_in)
frame, rvec, tvec = tools_aruco.detect_marker_and_draw_axes(frame, marker_length, cameraMatrix, dist)

glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
glDisable(GL_DEPTH_TEST)
glDrawPixels(frame.shape[1], frame.shape[0], GL_BGR, GL_UNSIGNED_BYTE, cv2.flip(frame, 0).data)
glEnable(GL_DEPTH_TEST)

glMatrixMode(GL_PROJECTION)
glLoadIdentity()

left = -principalX / fx
right = (image_shape[1] - principalX) / fx
bottom = (principalY - image_shape[0]) / fy
top = principalY / fy
glFrustum(left, right, bottom, top,near, far)

if numpy.count_nonzero(rvec)>0:
    glMatrixMode(GL_MODELVIEW)
    glLoadMatrixf(tools_aruco.compose_GL_MAT(rvec, tvec))
    tools_aruco.draw_native_axis(marker_length/2)
    glPushMatrix()
    tools_aruco.draw_cube(size=marker_length/4)
    glPopMatrix()

glutSwapBuffers()
glutPostRedisplay()
```

Augmented Reality