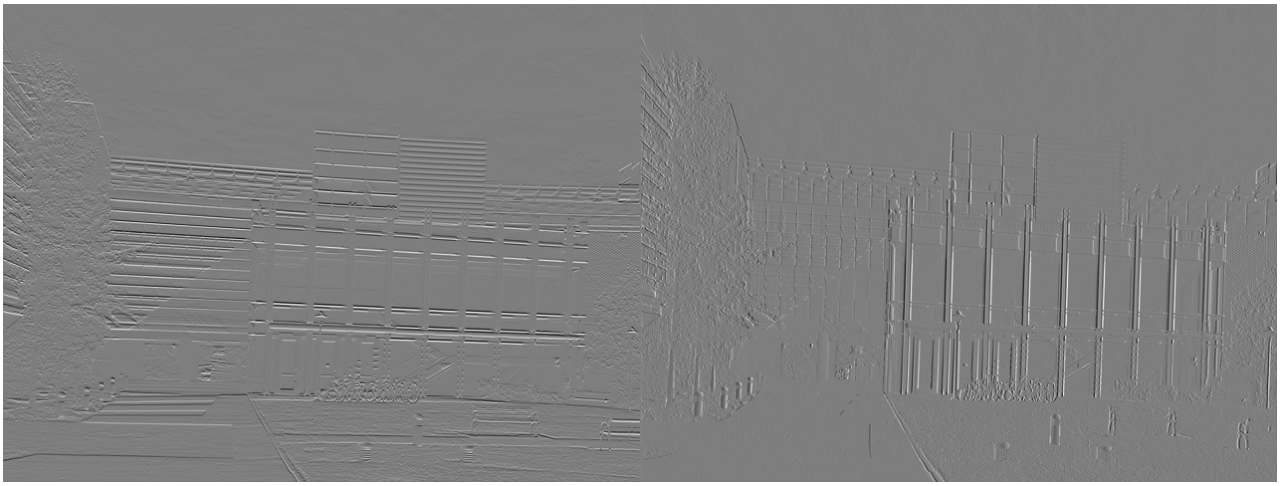
Computer vision - Problem set 4

Nicolas Six

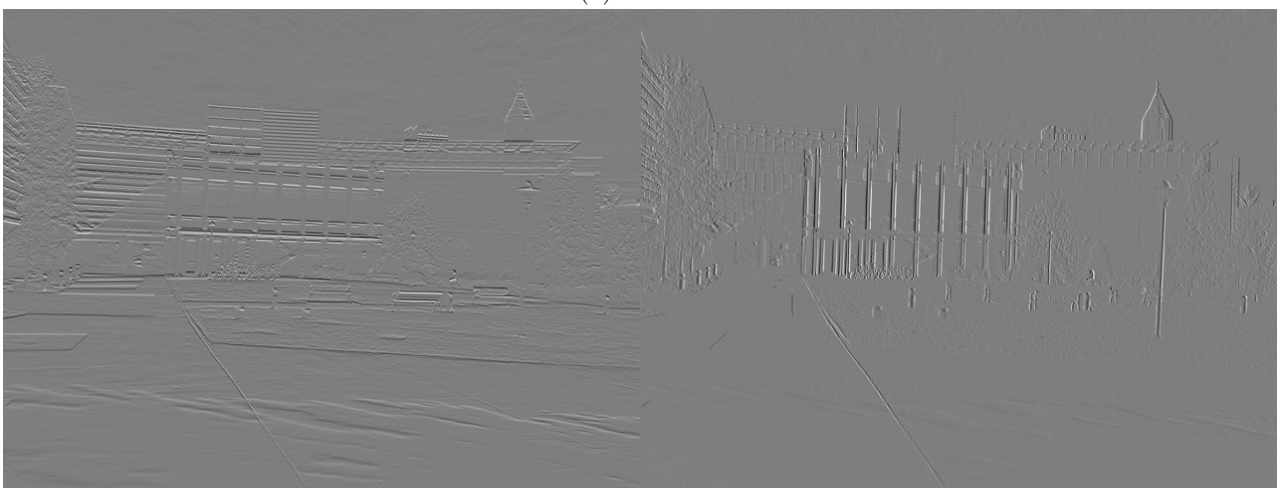
March 22, 2018

Question 1 Harris Corner

1.1 Gradient



(a) transA



(b) simA

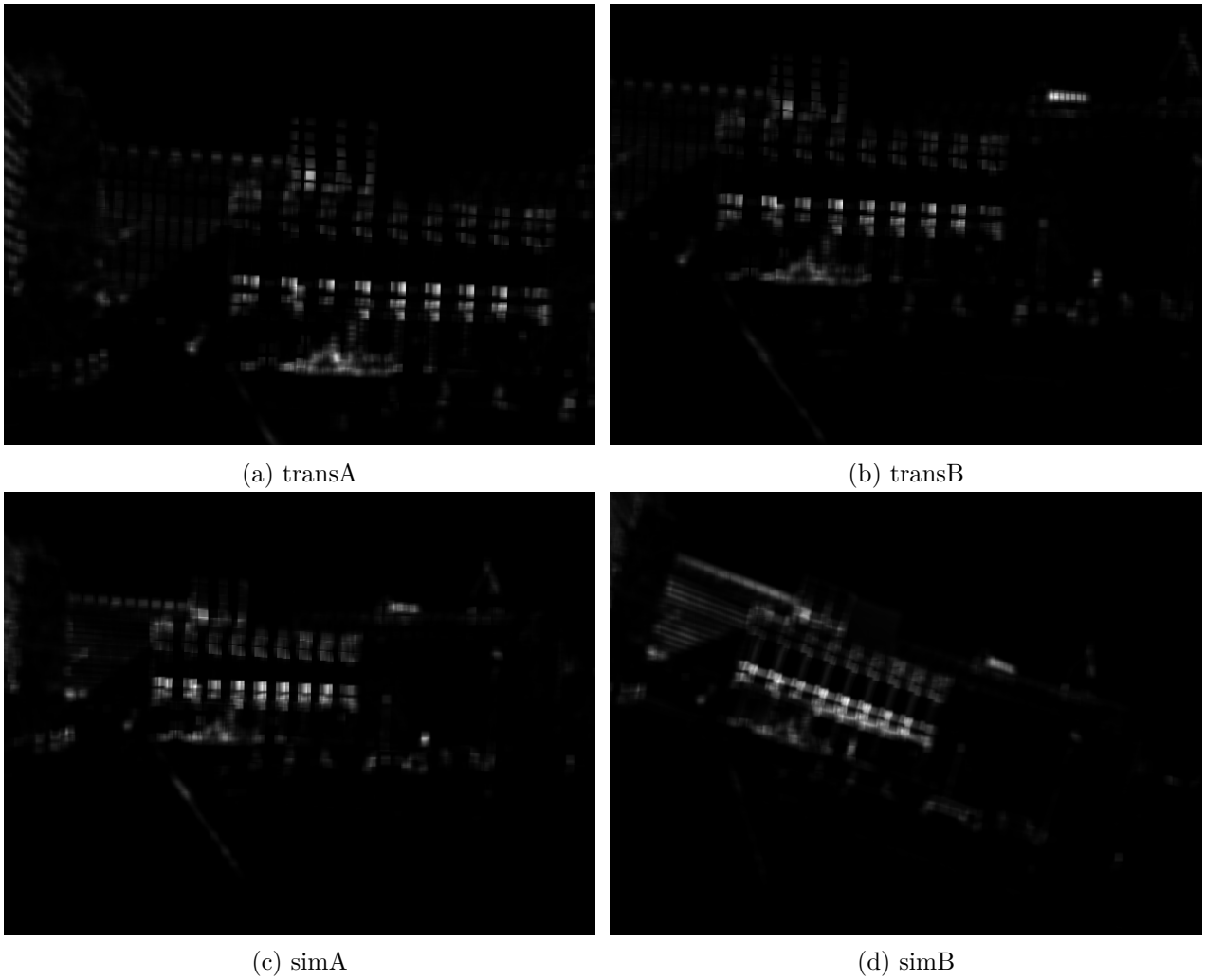
Figure 1: X and Y gradient

```
1 def compute_gradient(img: np.ndarray, direction: str):
```

```
    if direction == "X":
        imgA = np.array(img[:-1, :], dtype=np.int32)
        imgB = np.array(img[1:, :], dtype=np.int32)
    elif direction == "Y":
        imgA = np.array(img[:, :-1], dtype=np.int32)
        imgB = np.array(img[:, 1:], dtype=np.int32)
    else:
        raise ValueError("direction argument is expected to be 'X' or 'Y'
            only")

    return imgA - imgB
```

1.2 Harris detector



(a) transA (b) transB

(c) simA (d) simB

Figure 2: Harris values

```
def harris_transform(img: np.ndarray, win_size: int, alpha=1.0):
    i_x = compute_gradient(img, 'X')
    i_y = compute_gradient(img, 'Y')
```

```
    # reshape both gradient so they have the same size
    xx, xy = i_x.shape
    yx, yy = i_y.shape
    i_x = i_x[:min(xx, yx), :min(xy, yy)]
    i_y = i_y[:min(xx, yx), :min(xy, yy)]

    win = np.ones(shape=(win_size, win_size))

    i_xx_sum = convolve2d(np.power(i_x, 2), win, 'valid')
    i_yy_sum = convolve2d(np.power(i_y, 2), win, 'valid')
    i_xy_sum = convolve2d(i_x * i_y,        win, 'valid')

    return i_xx_sum * i_yy_sum - i_xy_sum**2 + alpha * (i_xx_sum +
        i_yy_sum), i_x, i_y
```

1.3 Corner

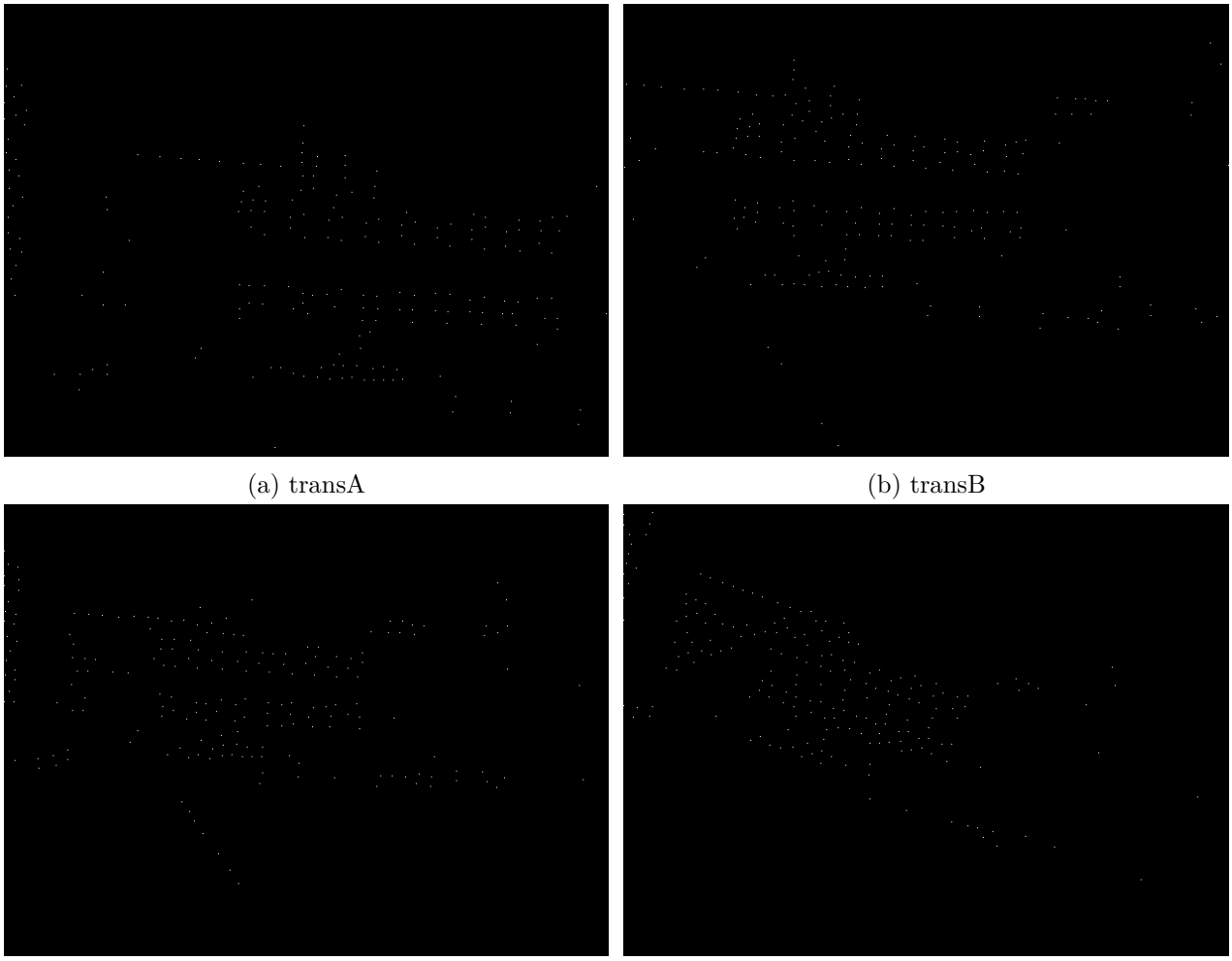
As you can see on Figure 3, all the main corner of the building are detected and have a similar shape. However, some corner appear on one side but note on the other, such as the road which appear on simA but not on simB.

```
def harris_filtering(harris_r: np.ndarray, number_of_points: int,
    win_size=5):
    r = harris_r.copy()
    r_w, r_h = r.shape
    r_filtered = np.zeros(shape=r.shape)
    win_radius = int(floor((win_size - 1) / 2))
    for i in range(number_of_points):
        p = np.argmax(r)
        p_x, p_y = p // r_h, p % r_h
        # r_filtered[p_x, p_y] = r[p_x, p_y]
        r_filtered[p_x, p_y] = 1
        r[max(0, p_x - win_radius):p_x + win_radius + 1, max(0, p_y -
            win_radius):p_y + win_radius + 1] = 0.0
    return r_filtered
```

Question 2 SIFT

2.1 Key points

```
def compute_keypoint_from_harris(img:np.ndarray, keypoint_number=100,
    win_size=5):
    r, i_x, i_y = harris_transform(img, win_size)
    r_w, r_h = r.shape
    win_radius = int(floor((win_size * 2 - 1) / 2))
    kp_list = []
    for i in range(keypoint_number):
        # find good corner
        p = np.argmax(r)
        p_x, p_y = p // r_h, p % r_h
        # remove good corner neighbors
```



(a) transA (b) transB

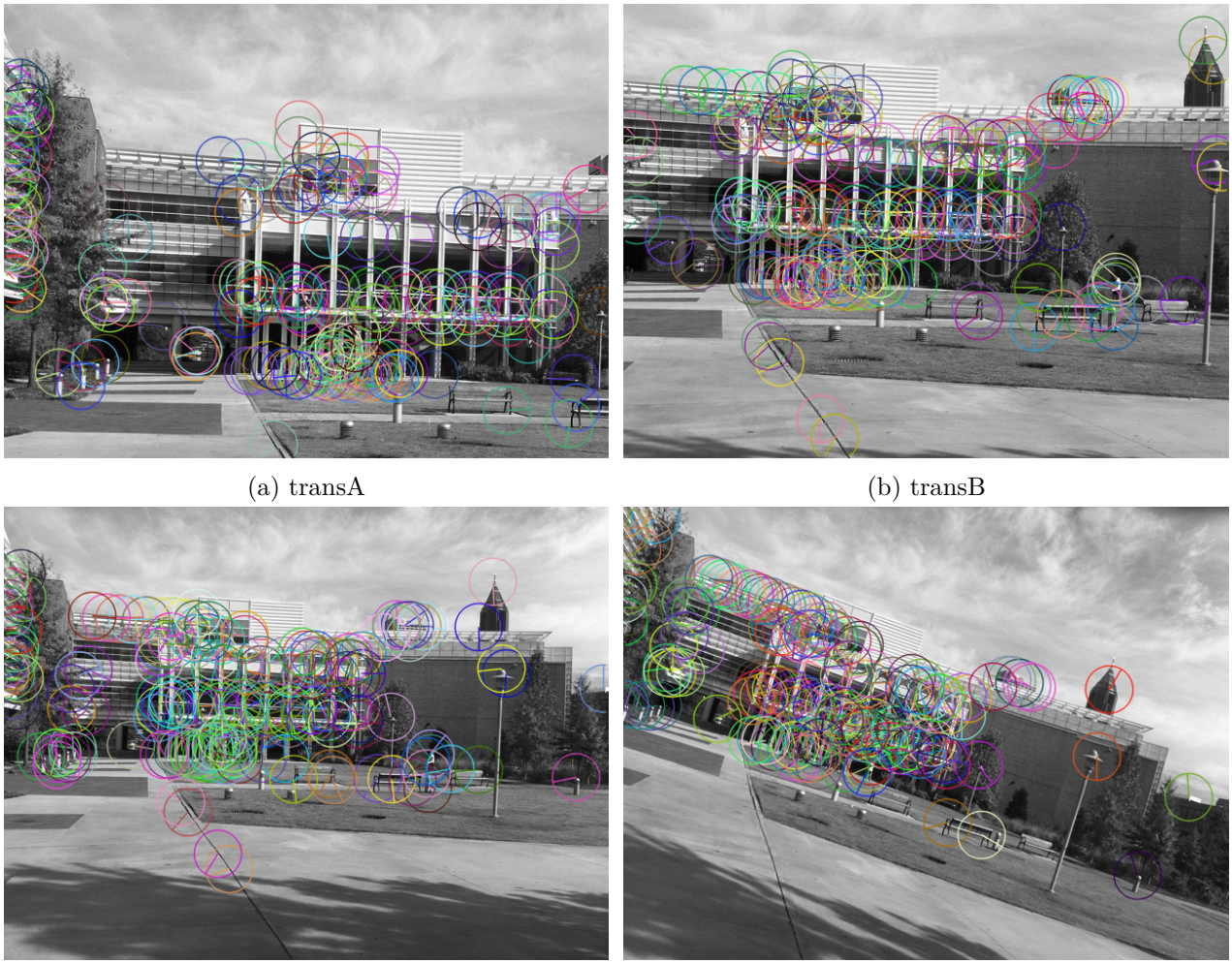
(c) simA (d) simB

Figure 3: Selected Harris corners (you may need to zoom in to properly see the white dot on some pdf viewer, including evince)

```
            r[max(p_x - win_radius, 0):p_x + win_radius + 1, max(p_y -
                win_radius, 0):p_y + win_radius + 1] = 0.0
            # create keypoint
            kp = cv2.KeyPoint()
            kp.angle = atan2(i_y[p_x, p_y], i_x[p_x, p_y]) * 180.0 / np.pi
            kp.size = 50.0
            kp.octave = 0
            kp.pt = (p_y, p_x)
            kp_list.append(kp)
        return kp_list
```

2.2 Putative pair

```
def draw_matches(im1, im2, kp1, kp2, matches):
    h1, w1 = im1.shape
    h2, w2 = im2.shape
```



(a) transA (b) transB

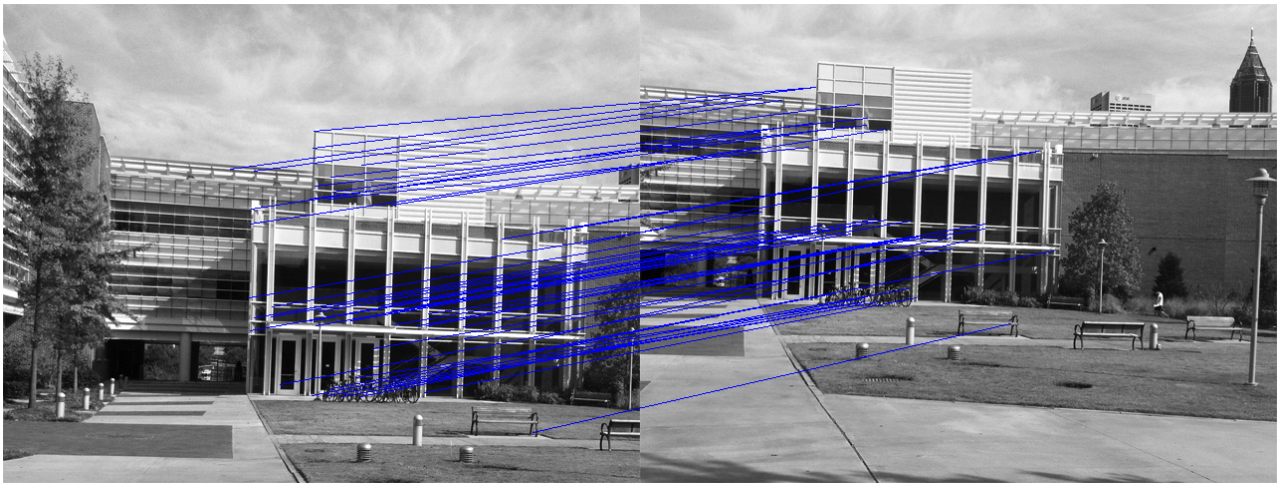
(c) simA (d) simB

Figure 4: Key points

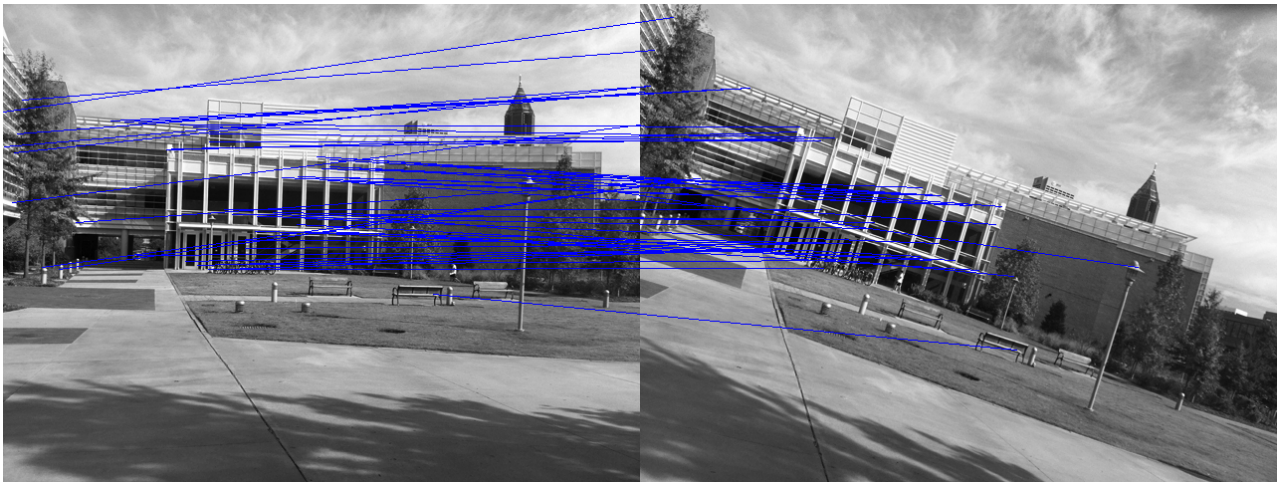
```
    ret = np.zeros(shape=(max(h1, h2), w1 + w2), dtype=np.uint8)
    ret[:h1, :w1] = im1
    ret[:h2, w1:] = im2
    ret = extend_to_three_channels(ret)
    for m in matches:
        x1, y1 = kp1[m[0].queryIdx].pt
        x2, y2 = kp2[m[0].trainIdx].pt
        ret = cv2.line(ret, (int(x1), int(y1)), (int(x2 + w1), int(y2)),
            (255, 0, 0))
    return ret
```

Question 3 RANSAC

3.1 Translational case



(a) transA - transB



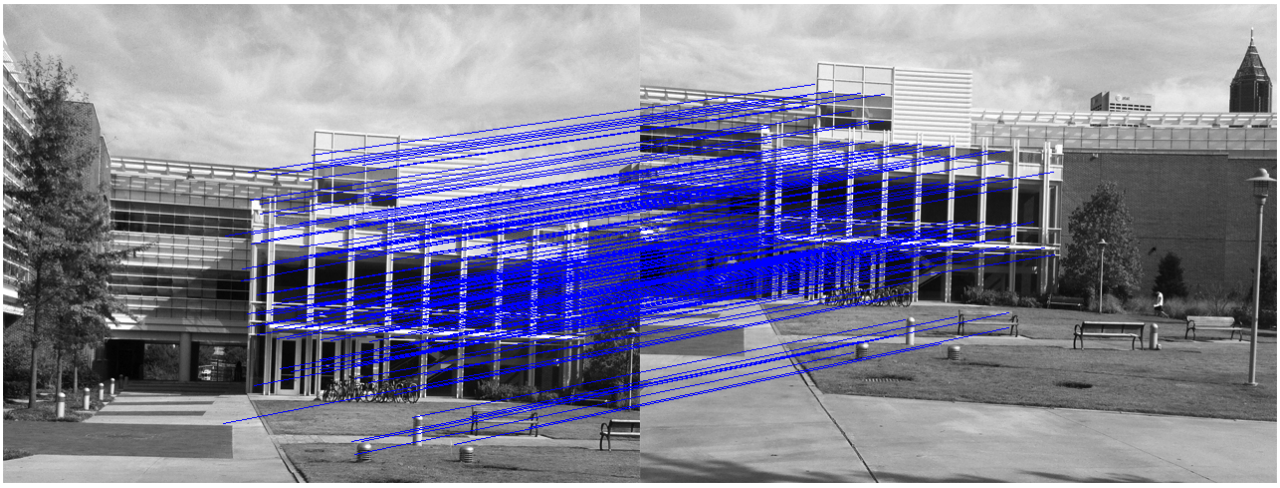
(b) simA - simB

Figure 5: putative-pair-images

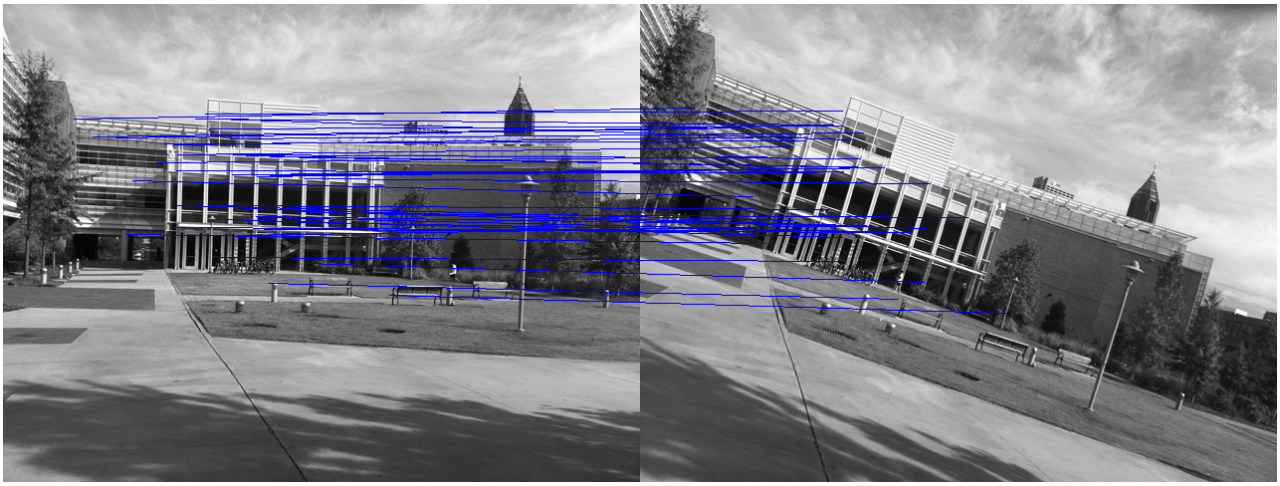
Pair	Vector	Consensus nb (%)
trans	$(-142, -87)$	145 (36%)
sim	$(-12, 5)$	41 (10%)

Table 1: Results of the RANSAC run

```
def randsac_match_trans(kp1, kp2, matches, number_of_try: int, max_dist:
    float):
     matches = list(map(lambda x: x[0], matches))
     con = sqlite3.connect(":memory:")
     con.enable_load_extension(True)
     con.load_extension("./extension-functions")
     con.execute("CREATE TABLE m (p1x INTEGER, p1y INTEGER, p2x INTEGER, p2y
         INTEGER);")
     con.executemany("insert into m(p1x, p1y, p2x, p2y) values (?, ?, ?, ?)",
                     map(lambda x: kp1[x.queryIdx].pt + kp2[x.trainIdx].pt,
                         matches))
```



(a) transA - transB



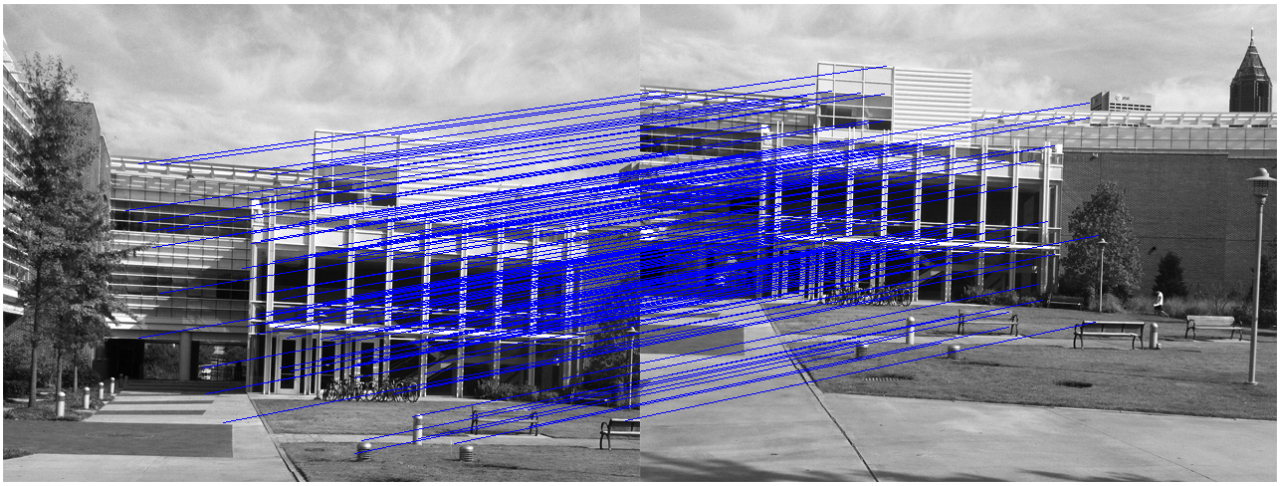
(b) simA - simB

Figure 6: Translational RANSAC matches

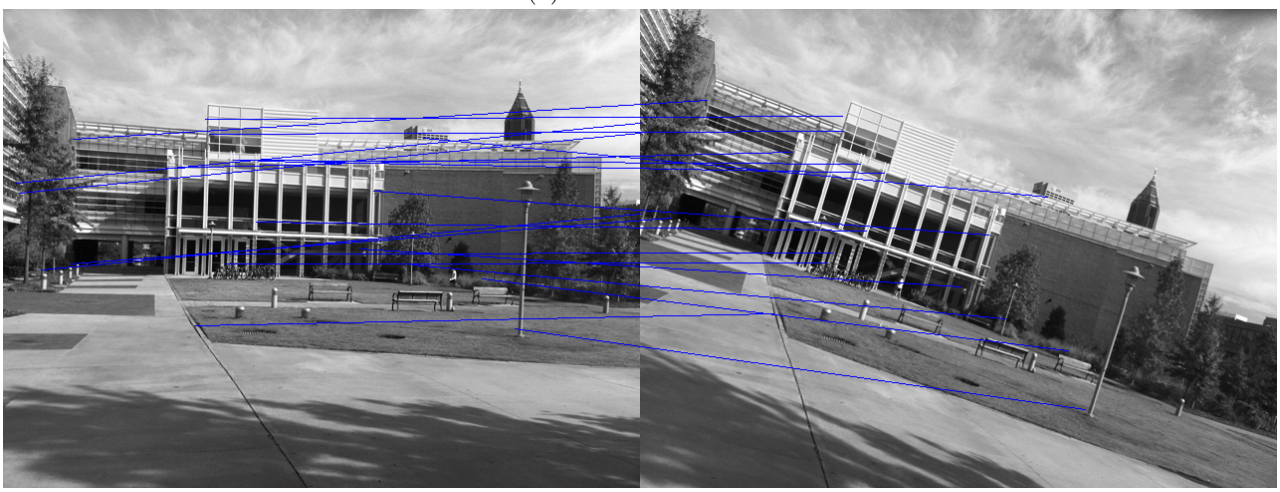
```
    max_count = 0
    sq_max_dist = max_dist**2
    best_move = None
    best_matchs = []
    for i in range(number_of_try):
        m = np.random.choice(matches)
        p1, p2 = kp1[m.queryIdx], kp2[m.trainIdx]
        movement = np.array(p2.pt) - np.array(p1.pt)
        matches_list = []
        for row in con.execute("SELECT m.rowid "
                               # "power(m.p1x + "+str(movement[0])+" -
                                  m.p2x, 2) + "
                               # "power(m.p1y + "+str(movement[1])+" -
                                  m.p2y, 2) as d "
                               # "from m where d <= "+str(sq_max_dist)+";"):
                               "FROM m "
                               "WHERE abs(m.p1x + "+str(movement[0])+" -
                                  m.p2x) <= "+str(sq_max_dist)+" "
```

```
                                    "AND abs(m.p1y + "+str(movement[1])+" -
                                       m.p2y) <= "+str(sq_max_dist)+";"):
                matches_list.append((matches[row[0] - 1],))
            if len(matches_list) > max_count:
                max_count = len(matches_list)
                best_move = movement
                best_matchs = matches_list
        con.execute("DROP TABLE m")
        con.close()
        return best_matchs, best_move, max_count
```

3.2 Similarity case



(a) transA - transB



(b) simA - simB

Figure 7: Similarity RANSAC matches

```
def randsac_match_affine(kp1, kp2, matches, number_of_try: int, max_dist:
    float):
     matches = list(map(lambda x: x[0], matches))
```

Pair	Matrix	Consensus nb (%)
trans	$\begin{bmatrix} 1.11456415 & 2.67395567e-03 & 1.13670153e+02 \\ -2.67395567e-03 & 1.11456415 & 6.33567651e+01 \end{bmatrix}$	140 (35%)
sim	$\begin{bmatrix} 0.94727273 & 0.27818182 & -21.46545455 \\ -0.27818182 & 0.94727273 & 65.65090909 \end{bmatrix}$	20 (5%)

Table 2: Results of the RANSAC run

```
    con = sqlite3.connect(":memory:")
    con.enable_load_extension(True)
    con.load_extension("./extension-functions")
    con.execute("CREATE TABLE m (p1x INTEGER, p1y INTEGER, p2x INTEGER, p2y
        INTEGER);")
    con.executemany("insert into m(p1x, p1y, p2x, p2y) values (?, ?, ?, ?)",
                    map(lambda x: kp1[x.queryIdx].pt + kp2[x.trainIdx].pt,
                        matches))
    max_count = 0
    best_move = None
    best_matchs = []
    for i in range(number_of_try):
        while True:
            m = np.random.choice(matches, 2, replace=False)
            p1, p2 = list(), list()
            for mi in m:
                p1.append(np.array(kp1[mi.queryIdx].pt))
                p2.append(np.array(kp2[mi.trainIdx].pt))
            if np.sum(p1[0] == p1[1]) != 2 and np.sum(p2[0] == p2[1]) != 2:
                break
        m = np.mat(
            [
                [p2[0][0], -p2[0][1], 1, 0],
                [p2[0][1],  p2[0][0], 0, 1],
                [p2[1][0], -p2[1][1], 1, 0],
                [p2[1][1],  p2[1][0], 0, 1]
            ]
        )
        v = np.mat(
            [
                [p1[0][0]],
                [p1[0][1]],
                [p1[1][0]],
                [p1[1][1]]
            ]
        )
        t = np.array(np.linalg.inv(m) * v)
        a, b, c, d = [t[i][0] for i in range(4)]
        # compute inverse transformation
        det = 1 / (a**2 + b**2)
        ai = a * det
        bi = -b * det
        ci = -(a * c + b * d) * det
        di = (b * c - a * d) * det
        matches_list = []
```

```
        for row in con.execute("SELECT m.rowid, "
                               "" + str(a) + " * m.p2x - " + str(b) + " *
                                  m.p2y +" + str(c) + " as u,"
                               "" + str(b) + " * m.p2x + " + str(a) + " *
                                  m.p2y +" + str(d) + " as v,"
                               "" + str(ai) + " * m.p1x - " + str(bi) + " *
                                  m.p1y +" + str(ci) + " as ui,"
                               "" + str(bi) + " * m.p1x + " + str(ai) + " *
                                  m.p1y +" + str(di) + " as vi "
                               "FROM m "
                               # "WHERE power(kp1.x-u, 2) + power(kp1.y-v,
                                  2) <= " + str(sq_max_dist) + " "
                               # "AND power(m.p2x-ui, 2) + power(m.p2y-vi,
                                  2) <= " + str(sq_max_dist) + ";"):
                               "WHERE abs(m.p1x-u)<=" + str(max_dist) + "
                                  AND abs(m.p1y-v) <= " + str(max_dist) + "
                                  "
                               "AND abs(m.p2x-ui)<=" + str(max_dist) + "
                                  AND abs(m.p2y-vi)<=" + str(max_dist) +
                                  ";"):
            matches_list.append((matches[row[0] - 1],))
        if len(matches_list) > max_count:
            max_count = len(matches_list)
            best_move = t
            best_matchs = matches_list
    con.execute("DROP TABLE m")
    con.close()
    return best_matchs, best_move, max_count
```