Vision – IF6083

Tugas Homework 3 (HW3)

Reza Budiawan
33221040

Program Studi S3 Teknik Elektro & Informatika

INSTITUT TEKNOLOGI BANDUNG
Desember 2022

Indeks Konten

Link Terkait

- Laporan Tugas
- Video Demo
- Link github If6083ComputerVision

Deskripsi Umum Tugas 3

Pada homework 3, terdapat beberapa hal yang dikerjakan, seperti: menemukan keypoints, describe, mencocokkan, dan melakukan stitching keypoint ke dalam sebuah panorama image. Untuk penanganan high level, sudah dilakukan pada method "panorama_image" di file "panorama_image.c". Beberapa task yang perlu dilakukan adalah menentukan corner points menggunakan Harris corner detector. Untuk melakukan estimasi projection dari koordinat citra, digunakan RANSAC.

3.1: Compute the structure matrix

Deskripsi

Isilah body method dari "image structure_matrix(image im, float sigma)" pada file harris_image.c. Terdapat 3 ha yang dilakukan pada method ini: menghitung derivatives, corresponding measures, dan weighted sum menggunakan Gaussian blur.

Solusi

Method ini memiliki 2 input parameter: citra input (im) dan nilai standar deviasi (sigma) yang digunakan untuk weighted sum. Dikarenakan menggunakan Gaussian, kode structure_matrix menggunakan kode dari HW 2 & melakukan proses konvolusi. Method ini membuat structure matrix yang memiliki informasi nilai I_x^2 (channel 1), I_y^2 (channel 2), I_xI_y (channel 3).

```
image structure_matrix(image im, float sigma)
{
    image S = make_image(im.w, im.h, 3);
    image D = make_image(im.w, im.h, 3);

    image gx_filter = make_gx_filter();
    image gy_filter = make_gy_filter();
    image gx = convolve_image(im, gx_filter, 0);
    image gy = convolve_image(im, gy_filter, 0);

    for (int i = 0; i < im.h * im.w; ++i){
        D.data[i] = gx.data[i] * gx.data[i];
        D.data[i + im.h * im.w] = gy.data[i] * gy.data[i];
        D.data[i + 2 * im.h * im.w] = gx.data[i] * gy.data[i];
    }
    S = smooth_image(D, sigma);
    free_image(D);
    free_image(gx_filter);
    free_image(gy_filter);
    free_image(gx);
```

```
    free_image(gy);
    return S;
}
```

3.1b: Compute the structure matrix

Deskripsi

Kodekan body method “image make_1d_gaussian(float sigma)” & “image smooth_image(image im, float sigma)” untuk membentuk filter Gaussian 1xN & melakukan smoothing.

Solusi

```
image make_1d_gaussian(float sigma)
{
    int w = 6 * sigma;
    w = (w > 0) ? ((w % 2 == 1) ? w : w + 1) : 1;
    int offset = w / 2;
    image gaussian_filter = make_image(1, w, 1);
    for (int i = 0 - offset; i < w - offset; ++i)
    {
        float v = 1 / sqrt((TWOPI * sigma * sigma)) * exp((-i * i) /
                                                          (2 * sigma * sigma));
        set_pixel(gaussian_filter, 0, i + offset, 0, v);
    }
    return gaussian_filter;
}

image smooth_image(image im, float sigma)
{
    if (1)
    {
        image g = make_gaussian_filter(sigma);
        image s = convolve_image(im, g, 1);
        free_image(g);
        return s;
    }
    else
    {
        image g = make_1d_gaussian(sigma);
        image s0 = convolve_image(im, g, 1);
        g.w = g.h;
        g.h = 1;
        image s1 = convolve_image(s0, g, 1);
        free_image(g);
        free_image(s0);
        return s1;
    }
}
```

3.2: Computer cornerness from structure matrix

Deskripsi

Isi body method dari image cornerness_response(image S).

Solusi

Method ini meng-estimasi cornerness dari tiap piksel dari citra S (structure matrix). Hal yang di-return adalah map of cornerness.

```
image cornerness_response(image S)
{
    image R = make_image(S.w, S.h, 1);
    // TODO: fill in R, "cornerness" for each pixel using the structure matrix.
    // We'll use formulation det(S) - alpha * trace(S)^2, alpha = .06.

    float alpha = 0.06;
    for (int i = 0; i < S.w * S.h; ++i)
    {
        float gxx = S.data[i];
        float gyy = S.data[i + S.w * S.h];
        float gxy = S.data[i + 2 * S.w * S.h];
        R.data[i] = gxx * gyy - gxy * gxy - alpha * (gxx + gyy) * (gxx + gyy);
    }
    return R;
}
```

3.3: Non-maximum suppression

Deskripsi

Agar proses matching menjadi lebih mudah, kita mengusahakan hanya local maximum responses yang ada pada corner detector. Hal ini dituliskan pada method “image nms_image(image im, int w)”.

Solusi

NMS atau non-suppretion max dilakukan dengan panduan berikut:

```
for every pixel in the image:
  for neighbors within w:
    if neighbor response greater than pixel response:
      set response to be very low (I use -999999 [why not 0??])
```

Method tersebut memiliki 2 input:

- im: 1-channel image of feature responses
- w: distance untuk melihat larger responses

Nilai return dari method ini adalah citra dengan nilai local-maxima responses pada w pixels

```
image nms_image(image im, int w)
{
    image r = copy_image(im);
    for (int i = 0; i < im.h; ++i){
        for (int j = 0; j < im.w; ++j){
            float v = get_pixel(im, j, i, 0);
            for (int m = -w; m < w + 1; ++m){
                for (int n = -w; n < w + 1; ++n){
                    float u = get_pixel(im, j + m, i + n, 0);
                    if (u > v){
                        set_pixel(r, j, i, 0, -999999);
                        goto label_out;
                    }
                }
            }
            label_out:;
        }
    }
    return r;
}
```

3.4: Complete the Harris detector

Deskripsi

Kodekan body method “harris_corner_detector”. Method ini mengembalikan array of descriptors untuk corners pada image.

Solusi

Method “harris_corner_detector” akan melakukan Harris corner detection & melakukan ekstraksi ciri dari cornes. Terdapat 5 parameter input: citra input, nilai std dev, nilai threshold untuk cornerness, nms, dan pointer terhadap jumlah corners terdeteksi.

```
descriptor *harris_corner_detector(image im, float sigma, float thresh, int nms, int *n)
{
    // Calculate structure matrix
    image S = structure_matrix(im, sigma);

    // Estimate cornerness
    image R = cornerness_response(S);

    // Run NMS on the responses
    image Rnms = nms_image(R, nms);

    // TODO: count number of responses over threshold
    int count = 0;
```

```
    for (int i = 0; i < Rnms.h*Rnms.w; ++i){
      if (Rnms.data[i] > thresh) count++;
    }

    *n = count; // <- set *n equal to number of corners in image.
    descriptor *d = calloc(count, sizeof(descriptor));
    //TODO: fill in array *d with descriptors of corners, use describe_index.
    for (int i = 0; i < Rnms.h*Rnms.w; ++i){
      if (Rnms.data[i] > thresh) {
        *d++ = describe_index(im, i);
      }
    }
    d -= count;

    free_image(S);
    free_image(R);
    free_image(Rnms);
    return d;
}
```

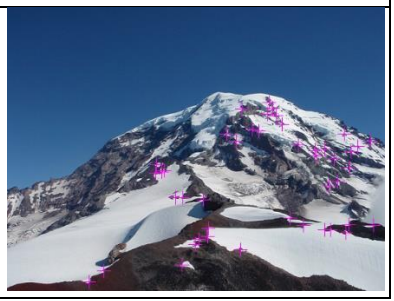
Hasil

Untuk melihat hasil dari Harris corner detection, script berikut dijalankan:

```
im = load_image("data/Rainier1.png")
detect_and_draw_corners(im, 2, 50, 3)
save_image(im, "corners")
```

Script di atas akan mendeteksi corners menggunakan Gaussian window dengan2 sigma, "cornerness" threshold bernilai 100, dan nms distance bernilai 3. Script ini terdapat pada function “draw_corners()” dalam file “tryhw3.py”. Luaran dari script di atas adalah sebuah citra “corners.jpg” yang memperlihatkan corners dari citra input Rainier1.png.

Hasil akhir diperlihatkan sebagai berikut:

Citra input	Citra luaran	Citra (deskripsi pada HW)
		

Berdasarkan hasil di atas, terlihat bahwa citra luaran sudah sesuai dengan yang diharapkan.

3.5: Distance metric

Deskripsi

Untuk membandingkan patches, digunakan L1 distance. Tuliskan body method dari "l1_distance(float *a, float *b, int n)". L1 distance merupakan sum of absolute differences. Method ini terdapat pada file "panorama_image.c".

Soal

```
float l1_distance(float *a, float *b, int n)
{
    float dst = 0;
    for (int i = 0; i < n; ++i){
        dst += fabsf(a[i] - b[i]);
    }
    return dst;
}
```

3.6: Matching descriptors

3.6.1: Find the best matches from a to b

Deskripsi

Tuliskanlah kode pada TODO pertama untuk method "match_descriptors" dalam file "panorama_image.c".

Solusi

```
match *match_descriptors(descriptor *a, int an, descriptor *b, int bn, int *mn)
{
    int i, j;

    // We will have at most an matches.
    *mn = an;
    match *m = calloc(an, sizeof(match));
    for (j = 0; j < an; ++j){
        // TODO: for every descriptor in a, find best match in b.
        // record ai as the index in *a and bi as the index in *b.
        int bind = 0; // <- find the best match
        m[j].distance = l1_distance(a[j].data, b[0].data, a[j].n);
        for (i = 1; i < bn; ++i){
            float dst = l1_distance(a[j].data, b[i].data, a[j].n);
            if (dst < m[j].distance){
                m[j].distance = dst;
                bind = i;
            }
```

```
        }
        m[j].ai = j;
        m[j].bi = bind; // <- should be index in b.
        m[j].p = a[j].p;
        m[j].q = b[bind].p;
    }

    ...
    return m;
}
```

3.6.2: Eliminate multiple matches to the same descriptor in b

Deskripsi

Setiap descriptor a memiliki 1 pasangan terhadap descriptor b. Akan tetapi, terkadang hal ini tidak sesuai dengan yang diinginkan karena adanya duplikasi (duplicate matches). Untuk memastikan one-to-one matches, urutkan setiap matches berdasarkan shortest distance. Berikutnya, lakukan iterasi pada setiap matches. Jika terdapat kesamaan, maka hilangkan hal tersebut (shifting forward). Hal ini dilakukan pada TODO kedua dari method “match_descriptors”.

Solusi

```
match *match_descriptors(descriptor *a, int an, descriptor *b, int bn, int *mn)
{
    int i, j;

    // We will have at most an matches.
    *mn = an;
    match *m = calloc(an, sizeof(match));
    for (j = 0; j < an; ++j){
        ...
        m[j].q = b[bind].p;
    }


    int count = 0;
    int *seen = calloc(bn, sizeof(int));
    // TODO: we want matches to be injective (one-to-one).
    // Sort matches based on distance using match_compare and qsort.
    // Then throw out matches to the same element in b. Use seen to keep track.
    // Each point should only be a part of one match.
    // Some points will not be in a match.
    // In practice just bring good matches to front of list, set *mn.
    qsort(m, an, sizeof(match), &match_compare);
    for (i = 0; i < an; ++i){
        if (seen[m[i].bi]){
            for (j = i; j < an - 1; ++j){
                m[j] = m[j + 1];
```

```
            }
            i = count - 1;
            an--;
        }
        else{
            seen[m[i].bi] = 1;
            count++;
        }
    }
    *mn = count;
    free(seen);
    return m;
}
```

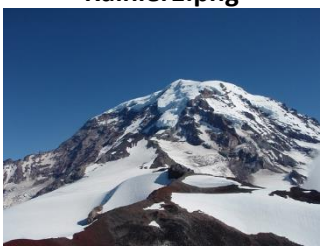
Hasil

Untuk melihat hasil matches, jalankan script python berikut:

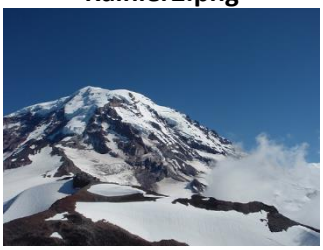
```
a = load_image("data/Rainier1.png")
b = load_image("data/Rainier2.png")
m = find_and_draw_matches(a, b, 2, 50, 3)
save_image(m, "matches")
```

Script di atas akan menghasilkan gambar matches.jpg yang akan mencocokkan gambar Rainier1.png & Rainier2.png. Script ini tercantum pada function “draw_matches()” dalam file “tryhw3.py”. Hasil diperlihatkan pada gambar di bawah ini.

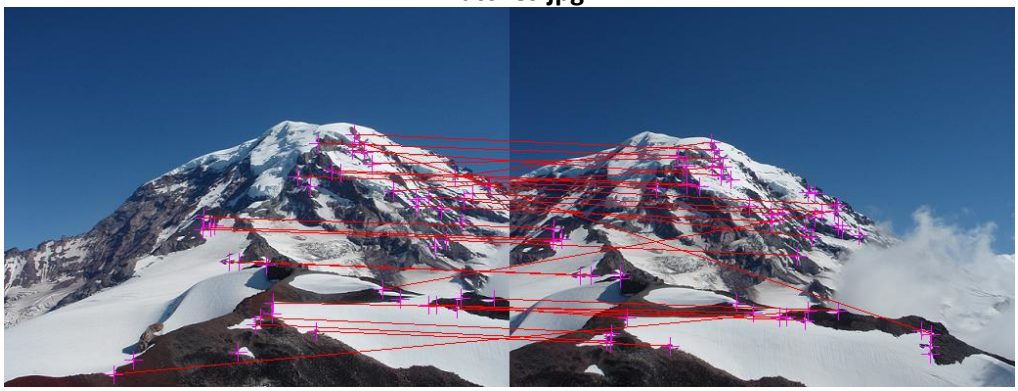
Rainier1.png



Rainier2.png



Matches.jpg



3.7: Fitting our projection to the data

3.7.1: Projecting points with a homography

Deskripsi

Implementasikan method “point project_point(matrix H, point p)” untuk melakukan proyeksi menggunakan matrix input H.

Solusi

```
point project_point(matrix H, point p)
{
    matrix c = make_matrix(3, 1);
    // TODO: project point p with homography H.
    // Remember that homogeneous coordinates are equivalent up to scalar.
    // Have to divide by.... something...
    c.data[0][0] = p.x;
    c.data[1][0] = p.y;
    c.data[2][0] = 1;
    matrix x = matrix_mult_matrix(H, c);
    point q = make_point(x.data[0][0] / x.data[2][0],
                         x.data[1][0] / x.data[2][0]);
    free_matrix(c);
    free_matrix(x);
    return q;
}
```

Hal yang dilakukan adalah menggunakan method “matrix_mult_matrix” yang sudah disediakan.

3.7.2: Calculate distances between points

Deskripsi

Isilah body method “point_distance(point p, point q)” menggunakan L2 distance.

Solusi

L2 distance merupakan akar dari jumlah kuadrat kedua vector (x_1,y_1) dan (x_2,y_2). Hal ini dapat dituliskan sebagai formula berikut:

$$distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Sehingga, pada kode dapat dituliskan sebagai berikut:

```
float point_distance(point p, point q)
{
    return sqrtf((p.x-q.x)*(p.x-q.x) + (p.y-q.y)*(p.y-q.y));
}
```

3.7.3: Calculate model inliers

Deskripsi

Tuliskanlah body method untuk “int model_inliers(matrix H, match *m, int n, float thresh)”.

Solusi

```
int model_inliers(matrix H, match *m, int n, float thresh)
{
    int i;
    int count = 0;
    // TODO: count number of matches that are inliers
    // i.e. distance(H*p, q) < thresh
    // Also, sort the matches m so the inliers are the first 'count' elements.
    for (i = 0; i < n; ++i){
      point p1 = project_point(H, m[count].p);
      float dst = point_distance(p1, m[count].q);
      if (dst < thresh){
        count++;
      } else{
        match match_temp = m[count];
        m[count] = m[n-1-i+count];
        m[n-1-i+count] = match_temp;
      }
    }
    return count;
}
```

3.7.4: Fitting the homography

Deskripsi

Implementasikan TODO dari “matrix compute_homography(match *matches, int n)” untuk melakukan homography pada matrix.

Solusi

```
matrix compute_homography(match *matches, int n)
{
    matrix M = make_matrix(n * 2, 8);
    matrix b = make_matrix(n * 2, 1);

    int i;
    for (i = 0; i < n; ++i)
    {
        double x = matches[i].p.x;
        double xp = matches[i].q.x;
        double y = matches[i].p.y;
        double yp = matches[i].q.y;
```

```
        // TODO: fill in the matrices M and b.
        double arr0[8] = {x, y, 1, 0, 0, 0, -x*xp, -y*xp};
        double arr1[8] = {0, 0, 0, x, y, 1, -x*yp, -y*yp};
        memcpy(*(M.data+2*i),   arr0, sizeof(arr0));
        memcpy(*(M.data+2*i+1), arr1, sizeof(arr1));
        b.data[2*i][0] = xp;
        b.data[2*i+1][0] = yp;
    }
    matrix a = solve_system(M, b);
    free_matrix(M);
    free_matrix(b);

    // If a solution can't be found, return empty matrix;
    matrix none = {0};
    if (!a.data)
        return none;

    matrix H = make_matrix(3, 3);
    // TODO: fill in the homography H based on the result in a.
    for (i = 0; i < M.cols; ++i){
      H.data[i / 3][i % 3] = a.data[i][0];
    }
    H.data[2][2] = 1;

    free_matrix(a);
    return H;
}
```

3.8: Randomize the matches

Deskripsi

Saat mengimplementasikan RANSAC, diperlukan langkah random matches untuk mengestimasi new model. Salah satu cara untuk melakukannya yaitu melakukan operasi shuffle pada array. Untuk itu, implementasikan Fisher-Yutes shuffle pada method “void randomize_matches(match *m, int n)”.

Solusi

```
void randomize_matches(match *m, int n)
{
    int i;
    for (i = n - 1; i > 0; --i){
      int j = rand() % (i - 0 + 1) + 0;
      match match_temp = m[i];
      m[i] = m[j];
      m[j] = match_temp;
    }
}
```

3.9: Implement RANSAC

Deskripsi

Implementasikan method “matrix RANSAC(match *m, int n, float thresh, int k, int cutoff)” sesuai pseudocode yang disediakan.

Solusi

```
matrix RANSAC(match *m, int n, float thresh, int k, int cutoff)
{
    int e;
    int best = 0;
    matrix Hb = make_translation_homography(256, 0);
    // TODO: fill in RANSAC algorithm.
    // for k iterations:
    //     shuffle the matches
    //     compute a homography with a few matches (how many??)
    //     if new homography is better than old (how can you tell?):
    //         compute updated homography using all inliers
    //         remember it and how good it is
    //         if it's better than the cutoff:
    //             return it immediately
    // if we get to the end return the best homography
    matrix H = make_matrix(3, 3);
    e = 4;
    for (int i = 0; i < k; ++i){
      randomize_matches(m, n);
      H = compute_homography(m, e);
      int inliners = model_inliers(H, m, n, thresh);
      if (inliners > best){
        best = inliners;
        Hb = compute_homography(m, inliners);
        if (best > cutoff){
          free_matrix(H);
          return Hb;
        }
      }
    }
    free_matrix(H);
    return Hb;
}
```

3.10: Combine the images with a homography

Deskripsi

Tuliskan method “combine_images”. Method ini menggabungkan 2 citra input dengan homography (matrix H).

Solusi

```
image combine_images(image a, image b, matrix H)
{
    matrix Hinv = matrix_invert(H);

    // Project the corners of image b into image a coordinates.
    point c1 = project_point(Hinv, make_point(0,0));
    point c2 = project_point(Hinv, make_point(b.w-1, 0));
    point c3 = project_point(Hinv, make_point(0, b.h-1));
    point c4 = project_point(Hinv, make_point(b.w-1, b.h-1));

    // Find top left and bottom right corners of image b warped into image a.
    point topleft, botright;
    botright.x = MAX(c1.x, MAX(c2.x, MAX(c3.x, c4.x)));
    botright.y = MAX(c1.y, MAX(c2.y, MAX(c3.y, c4.y)));
    topleft.x = MIN(c1.x, MIN(c2.x, MIN(c3.x, c4.x)));
    topleft.y = MIN(c1.y, MIN(c2.y, MIN(c3.y, c4.y)));

    // Find how big our new image should be and the offsets from image a.
    int dx = MIN(0, topleft.x);
    int dy = MIN(0, topleft.y);
    int w = MAX(a.w, botright.x) - dx;
    int h = MAX(a.h, botright.y) - dy;

    // Can disable this if you are making very big panoramas.
    // Usually this means there was an error in calculating H.
    /*
    if(w > 7000 || h > 7000){
        fprintf(stderr, "output too big, stopping\n");
        return copy_image(a);
    }
    */

    int i,j,k;
    image c = make_image(w, h, a.c);

    // Paste image a into the new image offset by dx and dy.
    for(k = 0; k < a.c; ++k){
        for(j = 0; j < a.h; ++j){
            for(i = 0; i < a.w; ++i){
                // TODO: fill in.
                set_pixel(c, i-dx, j-dy, k, get_pixel(a, i, j, k));
```

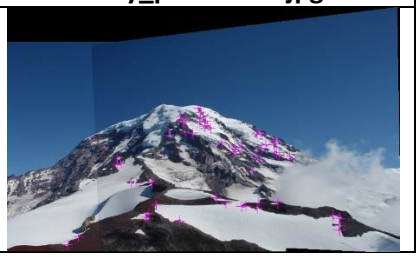
```
            }
        }
    }

    // TODO: Paste in image b as well.
    // You should loop over some points in the new image (which? all?)
    // and see if their projection from a coordinates to b coordinates falls
    // inside of the bounds of image b. If so, use bilinear interpolation to
    // estimate the value of b at that projection, then fill in image c.
    for (k = 0; k < a.c; ++k){
      for (j = topleft.y; j < botright.y; ++j){
        for (i = topleft.x; i < botright.x; ++i){
          point p = project_point(H, make_point(i, j));
          if (p.x >= 0 && p.x < b.w && p.y >= 0 && p.y < b.h){
            float v = bilinear_interpolate(b, p.x, p.y, k);
            set_pixel(c, i-dx, j-dy, k, v);
          }
        }
      }
    }
    return c;
}
```

Hasil

Untuk melihat hasil dari panorama, jalankan script berikut yang juga terdapat pada file “tryhw3.py” function “easy_panorama()”:

```
im1 = load_image("data/Rainier1.png")
im2 = load_image("data/Rainier2.png")
pan = panorama_image(im1, im2, thresh=50)
save_image(pan, "easy_panorama")
```

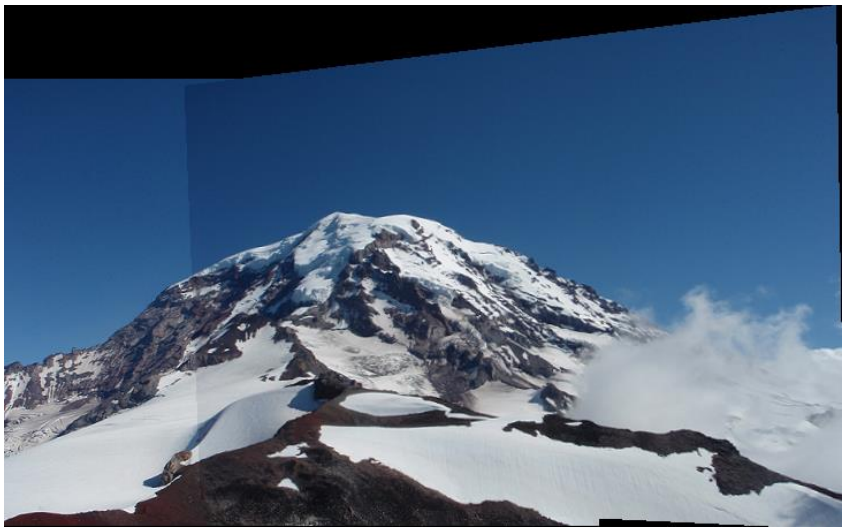
Script di atas akan melakukan penggabungan antara citra “Rainier1.png” & “Rainier2.png”. Hasil penggabungan berupa sebuah file bernama “easy_panorama”.

Rainier1.png	Rainier2.png	easy_panorama.jpg
		

Hasil di atas memperlihatkan hasil penggabungan kedua citra untuk membentuk citra panorama. Untuk menghilangkan marker corner, comment kode berikut pada method “panorama_image” di file “panorama_image.c”:

```
    if (1){
        mark_corners(a, ad, an);
        mark_corners(b, bd, bn);
        image inlier_matches = draw_inliers(a, b, H, m, mn, inlier_thresh);
        save_image(inlier_matches, "inliers");
    }
```

Maka hasil yang didapatkan seperti yang terlihat di bawah ini.



Hasil Skenario Test

Untuk memastikan setiap method yang diimplementasikan/dituliskan body method-nya sesuai harapan, jalankan command “uwimg test hw3” di command prompt/terminal. Testing ini terdiri dari 6 skenario test, dalam pengujian 4 method berikut:

- test_structure();
- test_cornerness();
- test_projection();
- test_compute_homography();

Jika sudah dikodekan dengan benar, maka hasil test menunjukkan status passed untuk setiap kasus uji.

```
C:\Users\rezab\Documents\Studi Lanjut\Sem 3\Vision - IF6083\If6083ComputerVision>uwimg test hw3
passed: [test_structure] testing [same_image(s, gt, EPS)] in ./src/test.c, line 452
passed: [test_cornerness] testing [same_image(c, gt, EPS)] in ./src/test.c, line 465
passed: [test_projection] testing [same_point(project_point(H, make_point(0,0)), make_point(12.4, -3.2
), EPS)] in ./src/test.c, line 477
passed: [test_projection] testing [same_point(p, make_point(-0.66544, 0.326017), EPS)] in ./src/test.c
, line 491
passed: [test_compute_homography] testing [same_matrix(H, d10)] in ./src/test.c, line 508
passed: [test_compute_homography] testing [same_matrix(H, Hp)] in ./src/test.c, line 525
6 tests, 6 passed, 0 failed
```