

IF6083Vision

Assignment - 2

Oleh:

Made Arbi Parameswara

(23522002)



**PROGRAM STUDI MAGISTER INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023**

Q.1.2 Correspondences

1. *Degree of freedom (dof)* merupakan jumlah nilai independen yang diperlukan untuk secara unik menentukan matriks. Matriks H merupakan matriks homografi yang berukuran 3×3 yang digunakan untuk memetakan koordinat titik dalam satu bida ke koordinat titik ke dalam bidang lane. Adapun struktur matriks H adalah sebagai berikut:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Terdapat 9 elemen dalam matriks H. Dalam konteks homografi dan koordinat homogen, matriks H hanya ditentukan hingga skala. Sehingga satu dari sembilan nilai dalam matriks tidak independen dan dependent terhadap variable lainnya. Hal ini membuat jumlah 'derajat kebebasan' matriks dari 9 menjadi 8.

2. Matriks homografi H jika diterapkan pada sebuah titik dalam koordinat homogen, menghasilkan titik yang sesuai di pandangan kamera. Diberikan pasangan titik (x_1^i, x_2^i) dimana x_1^i adalah titik dalam pandangan kamera pertama dan x_2^i adalah titik yang sesuai dalam pandangan kamera kedua. Hubungan antara titik-titik ini direpresentasikan oleh persamaan homografi:

$$x_1^i = Hx_2^i \quad (i \in \{1 \dots N\})$$

Jika matriks H didekomposisikan menjadi vektor kolom h . Untuk setiap pasangan titik, maka dapat menulis dua persamaan dari hubungan di atas, untuk koordinat x dan koordinat y pada titik pertama adalah

$$a_{11}h_1 + a_{12}h_2 + \dots + a_{19}h_1 = 0$$

$$a_{21}h_1 + a_{22}h_2 + \dots + a_{29}h_1 = 0$$

a_{ij} adalah koefisien yang berasal dari koordinat titik-titik x_1^i dan x_2^i , dan h_j adalah elemen dari vektor h . Perhatikan bahwa setiap pasangan titik memberikan dua persamaan seperti ini. Berdasarkan jawaban no 1, dof dari matriks H memiliki 8 komponen yang menjadi dof. Mengingat bahwa setiap pasangan titik memberikan dua persamaan, diperlukan setidaknya 4 pasangan titik untuk mendapatkan 8 persamaan. Dengan demikian, 4 pasangan titik adalah jumlah minimum yang diperlukan untuk menyelesaikan h dalam konteks homografi.

3. Jika kita menggunakan dua titik yang saling homogen berdasarkan persamaan $x_1^i = [x_1^i \ y_1^i \ 1]^T$ maka akan didapatkan dua persamaan titik sebagai berikut:

$$x_1^i = [x_1^i \ y_1^i \ 1]^T$$

$$x_2^i = [x_2^i \ y_2^i \ 1]^T$$

Hubungan kedua titik ini dapat direpresentasikan oleh matriks H dengan persamaan:

$$\lambda x_1^i = H x_2^i$$

Dengan λ sebuah faktor skala dalam bentuk konstanta yang merupakan factor dari x_1^i dan x_2^i , jika dilakukan dekomposisi maka:

$$\lambda \begin{bmatrix} x_1^i \\ y_1^i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Dikarenakan titik x_1^i dan x_2^i memiliki hubungan hamografi maka berlaku persamaan:

$$x_1^i x H x_2^i = 0$$

Sehingga:

$$\begin{bmatrix} y_1^i h_3^T x_2^i - h_2^T x_2^i \\ h_1^T x_2^i - x_1^i h_3^T x_2^i \\ x_1^i h_2^T x_2^i - y_1^i h_1^T x_2^i \end{bmatrix} = 0$$

Jika kita atur ulang sehingga persamaan untuk membentuk persamaan matriks $A_i h =$

0 dengan h merupakan matriks dengan komponen $\begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$

$$\begin{bmatrix} 0^T & -x_2^i & y_1^i x_2^i \\ x_2^i & 0^T & x_1^i x_2^i \\ -y_1^i x_2^i & x_1^i x_2^i & 0^T \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} = 0$$

Sehingga nilai dari A_i adalah

$$\begin{bmatrix} 0^T & -x_2^i & y_1^i x_2^i \\ x_2^i & 0^T & x_1^i x_2^i \\ -y_1^i x_2^i & x_1^i x_2^i & 0^T \end{bmatrix}$$

4.

- Solusi trivial untuk h pada persamaan $Ah = 0$ maka $h = 0$. Matrik A bukan merupakan full rank. Agar solusi non trivial ada maka nilai A harus memiliki null space, yang artinya Matrik A bukan singular. Sehingga Matrik A mempunyai *inverse transform*.
- Pada kasus persoalan diatas rank dari A adalah 8 dan dimensi dari $A^T A$ adalah 9×9 . Sehingga dimensi dari $A^T A$ memiliki 0 *eigenvalue* dan setiap nilai yang besesuaian merupakan *eigenvector* yang akan menjadi solusi dari h .

Q2.1.3 Matching Methods

Deskriptor BRIEF termasuk deskriptor biner yang bekerja dengan mengubah area gambar di sekitar titik fitur yang terdeteksi menjadi sekumpulan angka biner. Proses pencocokan fitur menggunakan deskriptor BRIEF sangat bergantung pada pemanfaatan jarak Hamming. Berikut pembahasan mengenai cara kerja jarak Hamming dan metode Nearest Neighbor dalam konteks ini:

1. Menggunakan Jarak Hamming:

Jarak Hamming diukur dengan menentukan berapa banyak posisi yang berbeda antara dua urutan biner. Dalam penggunaan deskriptor BRIEF, jarak Hamming berfungsi sebagai alat ukur perbandingan antara dua representasi fitur. Sebuah jarak Hamming yang kecil menunjukkan tingkat kesamaan yang lebih tinggi antar deskriptor.

2. Pencocokan Melalui Metode Nearest Neighbor:

Dalam proses pencocokan titik minat dengan deskriptor BRIEF, seringkali digunakan metode Nearest Neighbor. Ini melibatkan pencarian deskriptor dalam satu gambar yang paling mendekati atau memiliki jarak Hamming paling kecil terhadap deskriptor di gambar lain. Deskriptor dengan jarak Hamming minimum dianggap sebagai pasangan yang paling cocok.

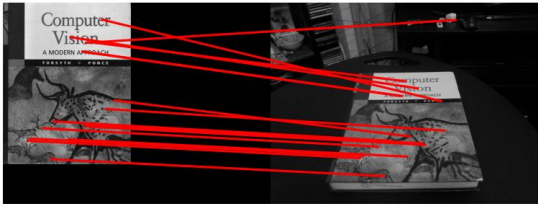
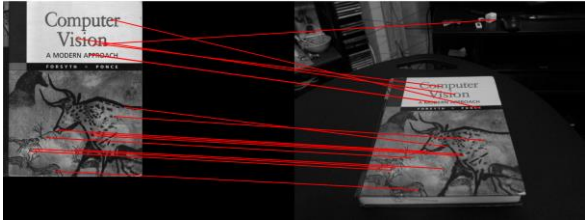
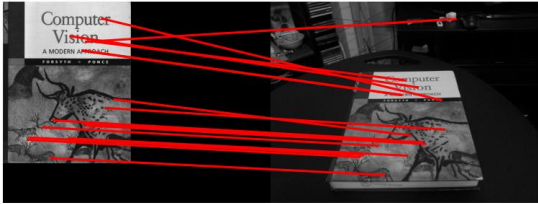
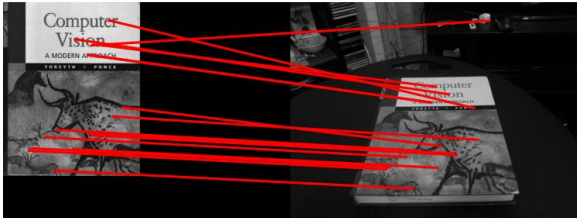
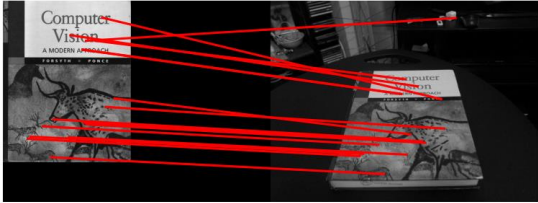
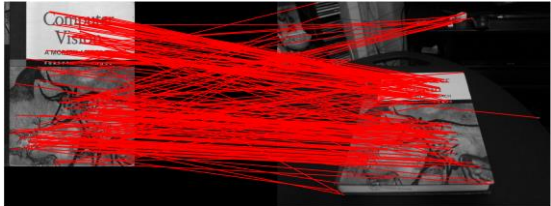
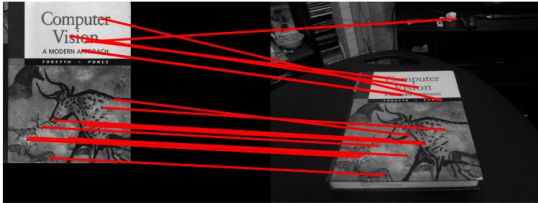
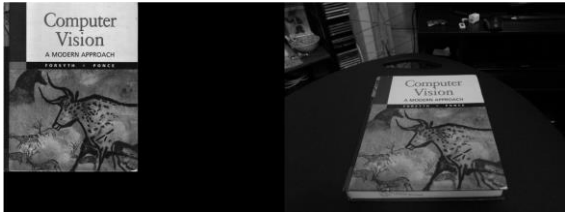
Kelebihan Jarak Hamming Dibandingkan dengan Jarak Euclidean:

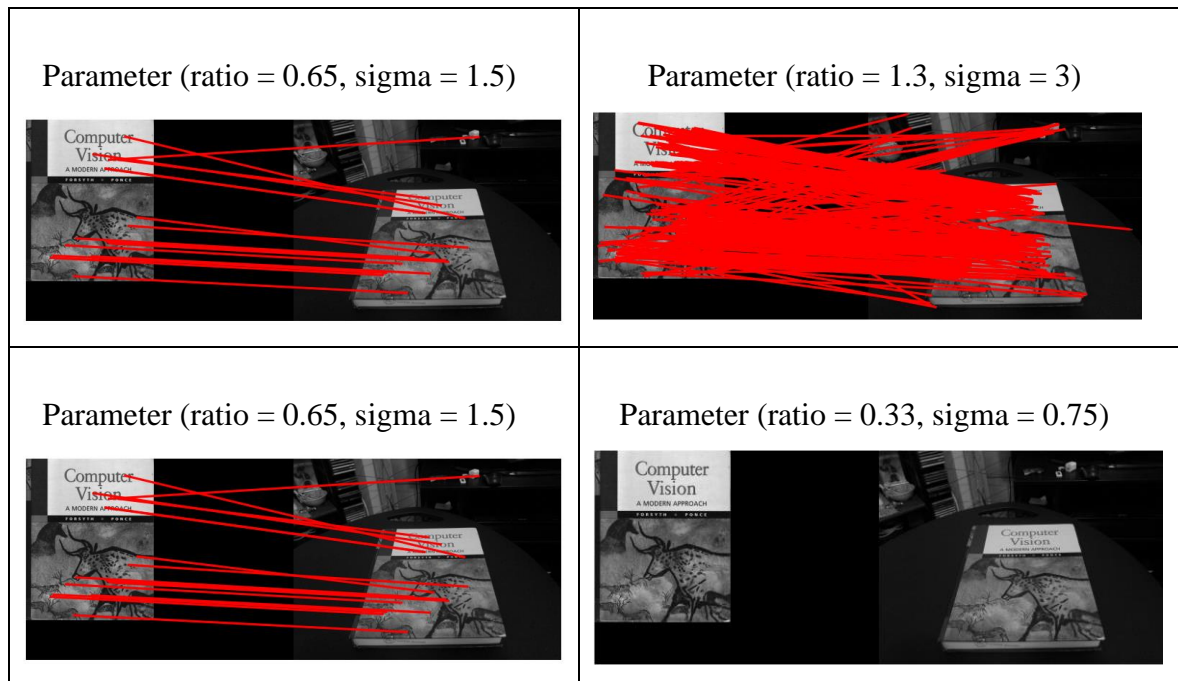
- Efisiensi Dalam Perhitungan
Karena deskriptor BRIEF terdiri dari rangkaian biner, operasi perhitungan menjadi lebih sederhana dan cepat. Penghitungan jarak Hamming lebih efisien daripada penghitungan jarak Euclidean.
- *Robust* Terhadap Variasi Pencahayaan:
Deskriptor BRIEF tetap lebih *robust* terhadap perubahan pencahayaan, hal ini membuat jarak Hamming menjadi kurang sensitif terhadap perubahan dalam intensitas cahaya sehingga dapat tetap lebih robust daripada menggunakan Euclidian.
- Tidak Sensitif Terhadap Panjang Deskriptor
Jarak Hamming tidak terpengaruh oleh panjang deskriptor. Hal ini membuat jarak Hamming memberikan hasil yang konsisten tanpa memandang panjang dari bits.
- Membutuhkan memori yang relative rendah
Deskriptor BRIEF yang memiliki nilai biner menggunakan kapasitas memori yang jauh lebih rendah daripada penggunaan kapasitas deskriptor berbasis nilai *real*.

Q2.1.4 Feature Matching

Hasil *feature matching* jika dibandingkan antara parameter *default* (ratio = 0.65, sigma = 1.5) yang ditentukan penulis dengan *parameter tuning*:

Tabel 1. Hasil Eksperimen *Feature Matching*

<p>Parameter (ratio = 0.65, sigma = 1.5)</p> 	<p>Parameter (ratio = 0.65, sigma = 3)</p> 
<p>Parameter (ratio = 0.65, sigma = 1.5)</p> 	<p>Parameter (ratio = 0.65, sigma = 0.75)</p> 
<p>Parameter (ratio = 0.65, sigma = 1.5)</p> 	<p>Parameter (ratio = 1.3, sigma = 1.5)</p> 
<p>Parameter (ratio = 0.65, sigma = 1.5)</p> 	<p>Parameter (ratio = 0.33, sigma = 1.5)</p> 



Adapun penjelasan berdasarkan eksperimen diatas:

- Sigma

Parameter sigma digunakan untuk mendeteksi sudut atau fitur dalam gambar. Nilai sigma mengontrol ukuran dari Gaussian blur yang diterapkan pada gambar sebelum proses deteksi fitur.

- Peningkatan Sigma (2σ): Menambah nilai sigma akan menyebabkan Gaussian blur yang lebih kuat, sehingga fitur yang lebih kecil mungkin tidak terdeteksi, dan hanya fitur yang lebih besar atau lebih menonjol yang akan diidentifikasi. Pada gaussian 3 lebih sedikit fitur yang terdeteksi dan secara kasat mata hanya fitur yang besar terdeteksi.
- Pengurangan Sigma ($\sigma/2$): Mengurangi nilai sigma menghasilkan Gaussian blur yang lebih lemah, memungkinkan deteksi fitur yang lebih kecil. Hal ini meningkatkan jumlah fitur yang terdeteksi. Pada gaussian 1.5 lebih banyak fitur yang terdeteksi dan banyak fitur-fitur noise yang terdeteksi.

- Ratio

Parameter *ratio* digunakan dalam pencocokan deskriptor fitur antara dua gambar.

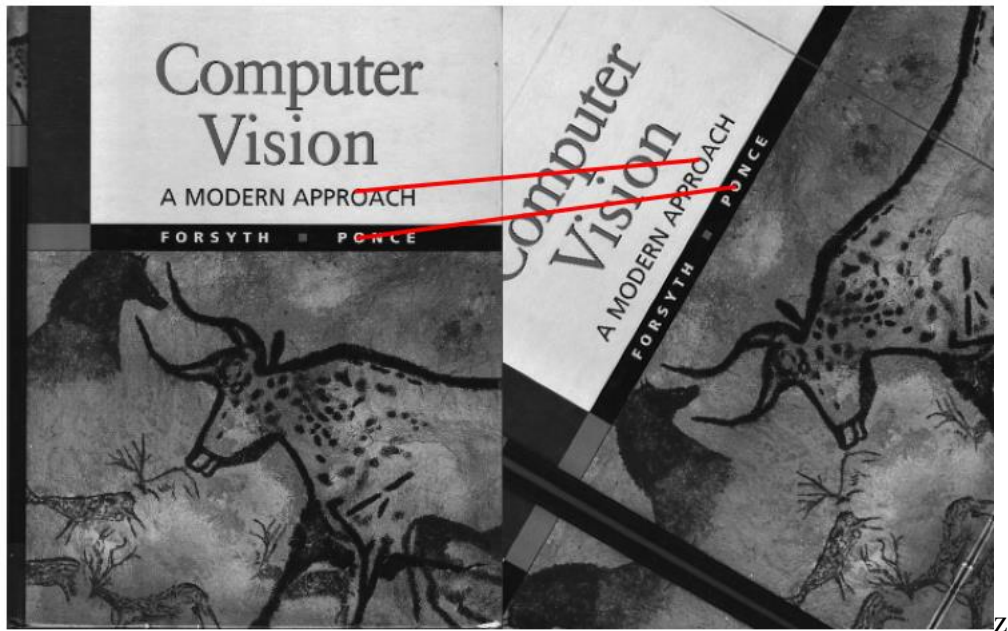
- Peningkatan Ratio ($2 \times \text{ratio}$): Menggandakan nilai ratio akan membuat kriteria pencocokan lebih longgar. Pada ratio 1.3 lebih banyak fitur yang terdeteksi namun banyak *noise* dan kesalahan dari *feature ratio*.
- Pengurangan Ratio ($\text{ratio}/2$): Mengurangi ratio membuat kriteria pencocokan lebih ketat. Ini mungkin mengurangi jumlah *false matches* tetapi juga bisa mengurangi jumlah total pencocokan yang berhasil karena standar yang lebih tinggi untuk dianggap cocok. Pada ratio 0.33 tidak ada fitur yang terdeteksi sehingga nilai 0.33 terlalu tinggi untuk gambar ini.

- Dalam konteks perubahan kombinasi sigma dan ratio:

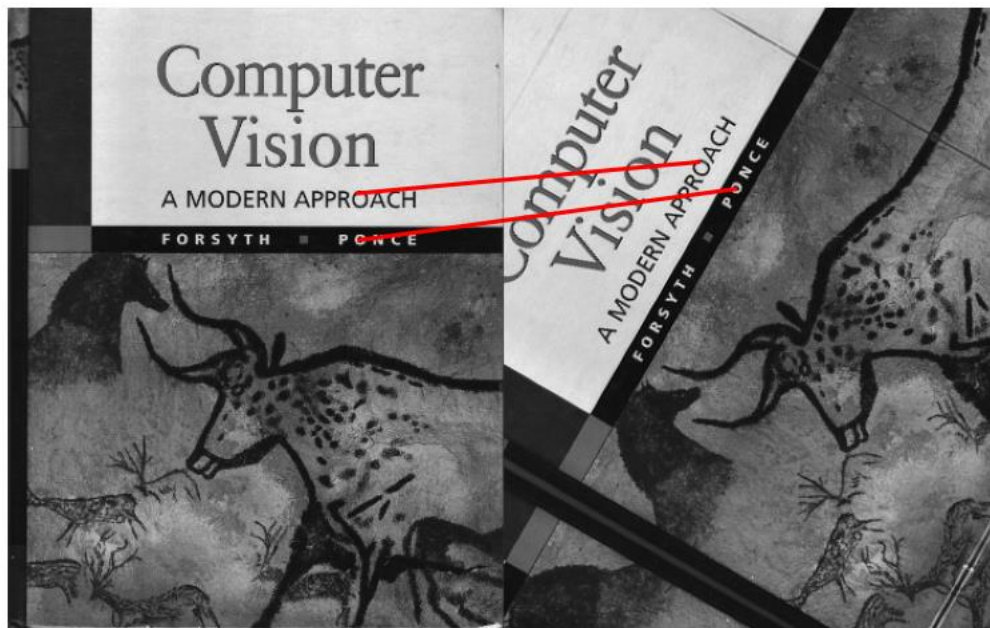
- Meningkatkan Keduanya (2σ , $2 \times \text{ratio}$): Akan menghasilkan deteksi fitur yang lebih kasar dengan pencocokan yang lebih longgar, hal ini meningkatkan jumlah total pencocokan tetapi meningkatkan *false matches*.
- Mengurangi Keduanya ($\sigma/2$, $\text{ratio}/2$): Akan menghasilkan deteksi fitur yang lebih detail dengan pencocokan yang lebih ketat, hal ini menghasilkan pencocokan yang lebih akurat tetapi mengurangi jumlah total pencocokan. Bahkan pada eksperimen ini tidak ada pencocokan yang terjadi.

Q2.1.5 BRIEF and Rotations

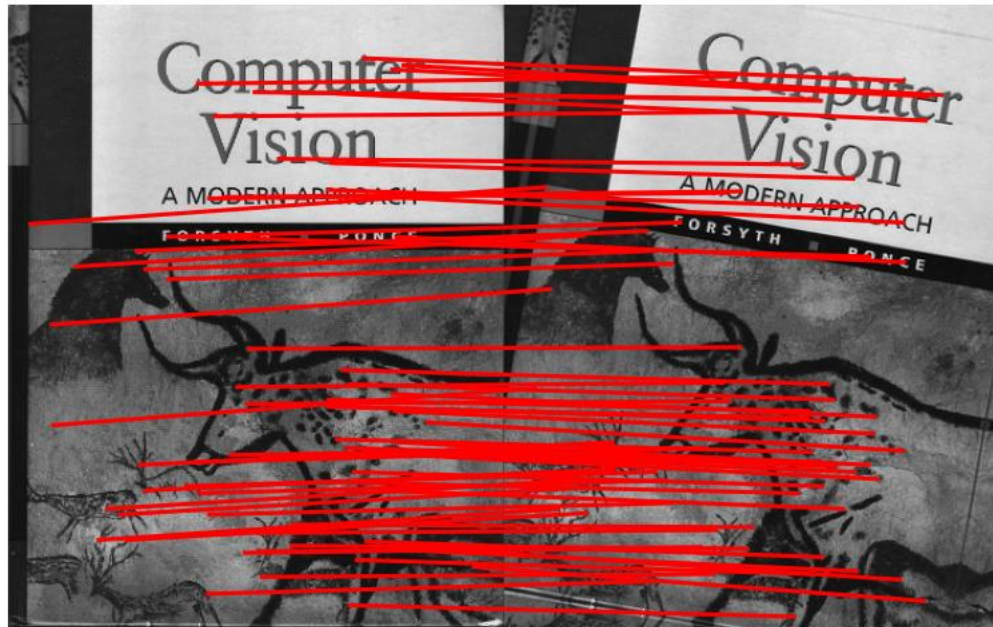
Berikut merupakan contoh penggunaan BRIEF pada gambar yang mengalami rotasi.



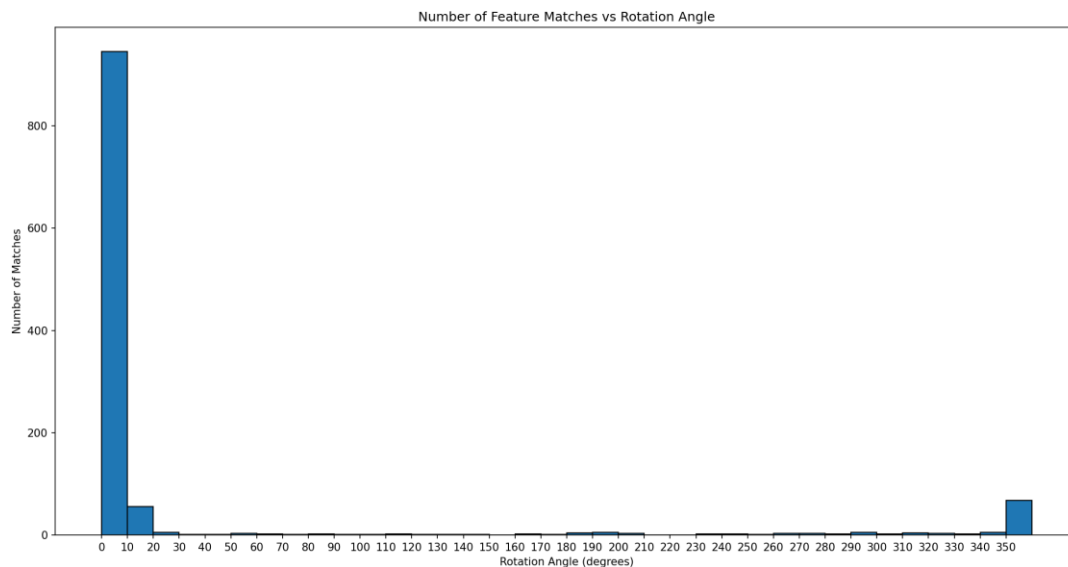
Gambar 1. Hasil BRIEF pada rotasi 60 derajat



Gambar 2. Hasil BRIEF pada rotasi 90 derajat



Gambar 3. Hasil BRIEF pada rotasi 350 derajat



Gambar 4. Histogram dari Jumlah Pasangan BRIEF terhadap Sudut Rotasi

Berdasarkan hasil chart histogram pada gambar 4. dapat disimpulkan bahwa BRIEF adalah metode *feature matching* yang kurang baik dalam memproses gambar rotasi. Ini berarti ketika sebuah gambar diputar, pola yang ditangkap oleh deskriptor BRIEF pada orientasi asli gambar dapat berubah secara signifikan, yang mengakibatkan penurunan jumlah kecocokan fitur ketika sudut rotasi berbeda dari orientasi asli.

Berdasarkan histogram diatas kecocokan fitur memiliki nilai yang cukup signifikan pada rotasi 10, 20, 350 derajat. Dikarenakan BRIEF bekerja dengan membandingkan intensitas piksel di sekitar titik kunci. Rotasi gambar yang mengubah posisi relatif piksel-piksel ini membuat BRIEF sulit mengenali fitur match diantara dua gambar diluar rotasi 10, 20, 350 derajat.

Homography Computation

Q2.2.1 Computing the Homography

Homografi adalah transformasi geometris yang menghubungkan dua bidang gambar, memungkinkan transformasi perspektif dari satu set titik dua dimensi ke set titik lain dalam ruang dua dimensi. Fungsi untuk menghitung homografi terdapat pada code planarH.py:

```

1  def computeH(x1, x2):
2
3
4      # Periksa apakah jumlah titik di kedua set sama
5
6      assert len(x1) == len(x2), "Jumlah titik di x1 dan x2 harus sama."
7
8      # Jumlah titik
9      n = len(x1)
10
11     # Membangun matriks A, setiap korespondensi titik memberikan dua baris
12     A = np.zeros((2 * n, 9))
13     for i in range(n):
14         A[2*i] = [-x2[i, 0], -x2[i, 1], -1, 0, 0, 0, x1[i, 0] * x2[i, 0], x1[i, 0] * x2[i, 1], x1[i, 0]]
15         A[2*i + 1] = [0, 0, 0, -x2[i, 0], -x2[i, 1], -1, x1[i, 1] * x2[i, 0], x1[i, 1] * x2[i, 1], x1[i, 1]]
16     ]
17
18     # Melakukan Dekomposisi Nilai Singular
19     U, S, Vh = np.linalg.svd(A)
20
21     # Homografi adalah kolom terakhir dari V (atau baris terakhir dari Vh)
22     homography = Vh[-1, :].reshape(3, 3)
23
24     return homography

```

Homography Normalization

Q2.2.2 Homography with normalization

Normalisasi digunakan sehingga hasil dari computing homography menjadi lebih stabil.

Fungsi compute_norm untuk menghitung homografi terdapat pada code planarH.py:

```
1 def computeH_norm(x1, x2):
2
3     assert len(x1) == len(x2), "Array harus memiliki panjang yang sama."
4
5
6
7     # Hitung centroid dari titik-titik
8     x1_centroid = np.mean(x1, axis=0)
9     x2_centroid = np.mean(x2, axis=0)
10
11     # Geser centroid titik-titik ke asal
12     x1_shifted = x1 - x1_centroid
13     x2_shifted = x2 - x2_centroid
14
15     # Normalisasi titik-titik agar jarak rata-rata dari asal sama dengan sqrt(2)
16     dist1 = np.sqrt(np.sum(x1_shifted**2, axis=1))
17     dist2 = np.sqrt(np.sum(x2_shifted**2, axis=1))
18     avg_dist1 = np.mean(dist1)
19     avg_dist2 = np.mean(dist2)
20     scale1 = np.sqrt(2) / avg_dist1
21     scale2 = np.sqrt(2) / avg_dist2
22     x1_normalized = x1_shifted * scale1
23     x2_normalized = x2_shifted * scale2
24
25     # Membangun transformasi kemiripan
26     T1 = np.array([[scale1, 0, -x1_centroid[0] * scale1],
27                   [0, scale1, -x1_centroid[1] * scale1],
28                   [0, 0, 1]])
29     T2 = np.array([[scale2, 0, -x2_centroid[0] * scale2],
30                   [0, scale2, -x2_centroid[1] * scale2],
31                   [0, 0, 1]])
32
33     # Hitung homografi dari koordinat yang dinormalisasi
34     homography = computeH(x1_normalized, x2_normalized)
35
36     # Denormalisasi matriks homografi
37     homography = np.linalg.inv(T1) @ homography @ T2
38
39     return homography
```

RANSAC

Q2.2.3 Implement RANSAC for computing a homography

Fungsi `computeH_ransac(locs1, locs2, opts)` merupakan implementasi algoritma RANSAC untuk menemukan homografi terbaik antara dua set titik yang cocok dari dua gambar berbeda. RANSAC adalah pendekatan yang kuat dan sering digunakan untuk mengatasi masalah outlier. Berikut merupakan kode untuk mengimplementasikan RANSAC untuk menghitung homografi.

```

1  def computeH_ransac(locs1, locs2, opts):
2      max_iters = opts.max_iters # Jumlah iterasi untuk menjalankan RANSAC
3      inlier_tol = opts.inlier_tol # Nilai toleransi untuk menganggap suatu titik sebagai inlier
4
5      assert len(locs1) == len(locs2),
6
7      # Tukar kolom di locs karena dalam bentuk [y, x] yang dikembalikan oleh matchPics
8      x1 = locs1[:, [1, 0]]
9      x2 = locs2[:, [1, 0]]
10
11     # Inisialisasi variabel untuk melacak homografi terbaik dan inlier
12     bestH2to1 = None
13     best_inliers = np.zeros(len(x1), dtype=bool)
14     best_inliers_count = 0
15
16     for iter in range(max_iters):
17         # Pilih secara acak 4 pasang titik
18         sample_indices = np.random.choice(len(x1), 4, replace=False)
19         x1_samples = x1[sample_indices]
20         x2_samples = x2[sample_indices]
21
22         # Hitung homografi dari sampel
23         H2to1 = computeH_norm(x1_samples, x2_samples)
24
25         # Terapkan homografi ke semua titik di x2
26         x2_homo = np.concatenate((x2, np.ones((len(x2), 1))), axis=1)
27         x2_transformed = (H2to1 @ x2_homo.T).T
28         x2_transformed /= x2_transformed[:, 2:3]
29
30         # Hitung inlier di mana titik-titik yang ditransformasi berada
31         dalam toleransi inlier
32         inliers_binary = np.linalg.norm(x1 - x2_transformed[:, :2], axis=1) <= inlier_tol
33         inliers_count = np.sum(inliers_binary)
34
35         # Perbarui homografi terbaik jika ditemukan lebih banyak inlier

```

```
36     if inliers_count > best_inliers_count:
37         bestH2to1 = H2to1
38         best_inliers = inliers_binary
39         best_inliers_count = inliers_count
40
41     # Jika semua titik adalah inlier, kita telah menemukan homografi terbaik
42     if inliers_count == len(x1):
43         break
44
45     return bestH2to1, best_inliers.astype(int)
```

Automated Homography Estimation and Warping

Q2.2.4 Putting it together & Q2.2.5 Parameter Tuning

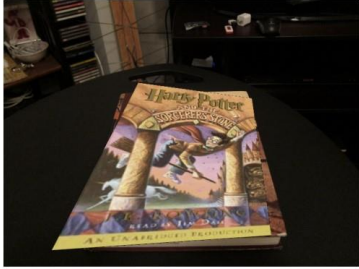
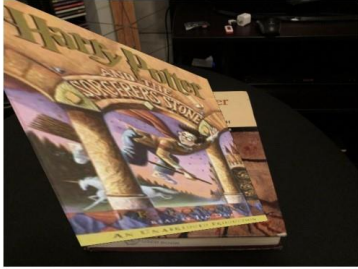
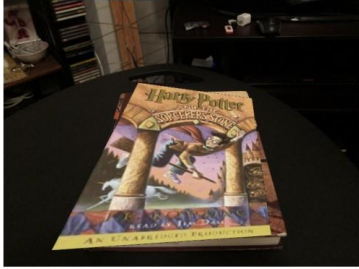
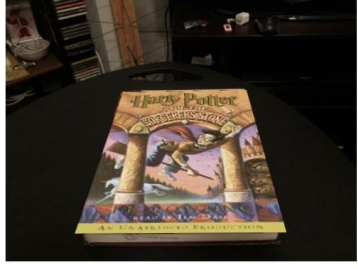
Code (Putting it together)

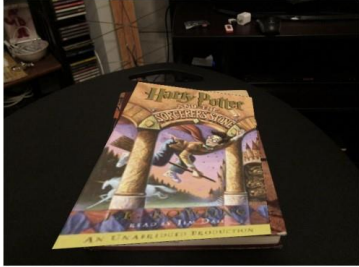
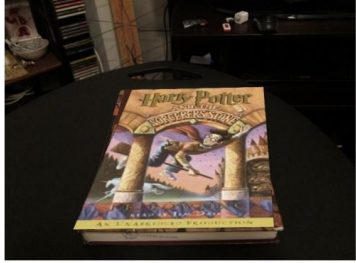
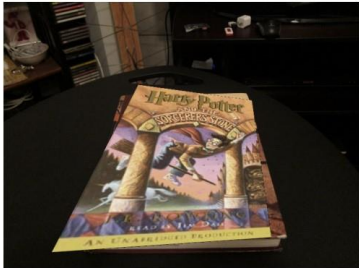
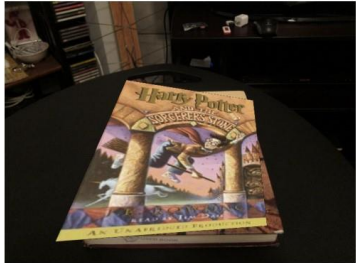
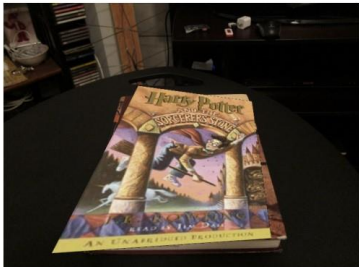
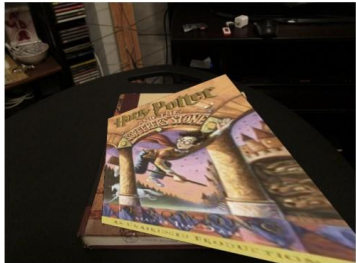
```
1  import numpy as np
2  import cv2
3  import skimage.io
4  import skimage.color
5  from matplotlib import pyplot as plt
6
7  from opts import get_opts
8  from MatchPics import matchPics
9  from planarH import computeH_ransac
10 from planarH import compositeH
11 from helper import plotMatches
12
13 # Impor fungsi-fungsi yang diperlukan (Dengan asumsi fungsi-fungsi seperti get_opts,
14 matchPics, computeH_ransac, compositeH, dll., sudah didefinisikan di tempat lain)
15
16 def read_and_resize_images(cover_path, desk_path, new_cover_path):
17     original_cover = cv2.imread(cover_path)
18     desk_image = cv2.imread(desk_path)
19     new_cover = cv2.imread(new_cover_path)
20
21     # Ubah ukuran gambar sampul baru agar sesuai dengan ukuran sampul asli
22     new_cover_resized = cv2.resize(new_cover, (original_cover.shape[1],
23 original_cover.shape[0]))
24     return original_cover, desk_image, new_cover_resized
25
26 def process_images(cv_cover, cv_desk, hp_cover, opts, max_iters, tol_values):
27     matches, locs1, locs2 = matchPics(cv_cover, cv_desk, opts)
28     # plotMatches(cv_cover, cv_desk, matches, locs1, locs2)
29
```

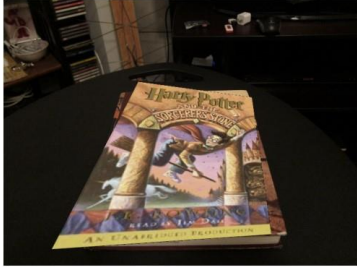
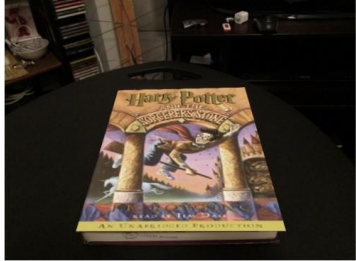
```
30     for max_iter in max_iters:
31         for tol in tol_values:
32             opts.max_iters = max_iter
33             opts.inlier_tol = tol
34             bestH2to1, inliers = computeH_ransac(locs1[matches[:,0]], locs2[matches[:,1]],
35 opts)
36             composite_img = compositeH(bestH2to1, hp_cover, cv_desk)
37
38             plt.figure()
39             plt.axis('off')
40             plt.imshow(cv2.cvtColor(composite_img, cv2.COLOR_BGR2RGB))
41             plt.savefig(f'./result/pic_{max_iter}_{tol}.png')
42
43 # Inisialisasi opsi (parameter untuk berbagai fungsi)
44 opts = get_opts()
45
46 # Baca dan ubah ukuran gambar
47 cv_cover, cv_desk, hp_cover = read_and_resize_images('./data/cv_cover.jpg',
48 './data/cv_desk.png', './data/hp_cover.jpg')
49
50 # Kisaran hiperparameter
51 max_iterations = [1250, 2500, 5000]
52 tolerances = [2, 10, 50]
53
54 # Proses gambar dengan hiperparameter RANSAC yang berbeda
55 process_images(cv_cover, cv_desk, hp_cover, opts, max_iterations, tolerances)
56
```


Gambar dibawah ini merupakan gambar dengan script HarryPoterized.py dengan menggunakan default parameter max_iter=500 dan inliner_tolerance= 2.

Tabel 2. Eksperimen script HarryPoterized.py

<p>Parameter (max_iterations = 2500, tolerance = 10)</p> 	<p>Parameter (max_iterations = 2500, tolerance = 50)</p> 
<p>Parameter (max_iterations = 2500, tolerance = 10)</p> 	<p>Parameter (max_iterations = 2500, tolerance = 2)</p> 

<p>Parameter (max_iterations = 2500, tolerance = 10)</p> 	<p>Parameter (max_iterations = 5000, tolerance = 10)</p> 
<p>Parameter (max_iterations = 2500, tolerance = 10)</p> 	<p>Parameter (max_iterations = 1250, tolerance = 10)</p> 
<p>Parameter (max_iterations = 2500, tolerance = 10.0)</p> 	<p>Parameter (max_iterations = 5000, tolerance = 50)</p> 

<p>Parameter (max_iterations = 2500, tolerance = 10)</p> 	<p>Parameter (max_iterations = 1250, tolerance = 2)</p> 
--	--

Pengaturan max_iterations dan tolerance pada algoritma) dalam proses ini memiliki dampak signifikan terhadap akurasi dan robustness hasil distorsi gambar. Adapun hasil yang didapatkan dari eksperimen sederhana ini:

- Meningkatkan tolerance (toleransi) mengizinkan lebih banyak poin untuk dianggap sebagai inliers dalam penghitungan homografi. Namun berdasarkan hasil eksperimen tolerances yang terlalu tinggi membuat gambar menjadi terdistorsi.
- Menurunkan tolerances menjadi (0.2x):
Mengurangi tolerance meningkatkan ketepatan dalam menentukan inliers, tetapi juga meningkatkan risiko menolak poin yang seharusnya merupakan inliers. Berdasarkan hasil eksperimen tolerances ini menghasilkan gambar yang tidak terdistorsi.
- Peningkatan Max_iterations Dua Kali Lipat (2x):
Meningkatkan max_iterations memungkinkan RANSAC lebih banyak kesempatan untuk menemukan subset data (inliers) yang memberikan estimasi homografi terbaik. Namun peningkatan ini akan meningkatkan waktu komputasi untuk mencari estimasi homografi terbaik.

- Max_iterations Setengah (0.5x):
Mengurangi max_iterations mengurangi jumlah iterasi yang dilakukan RANSAC, sehingga mengurangi kesempatan menemukan inliers yang optimal dan menghasilkan distorsi pada gambar.
- Kombinasi Max_iterations (2x) dan Tolerances Dua Kali Lipat (5x):
Kombinasi ini menghasilkan hasil yang kurang baik dikarenakan tolerances yang terlalu tinggi, max_iterations yang ditingkatkan sebanyak 2x ternyata tidak mampu membuat gambar tanpa distorsi.
- Kombinasi Max_iterations dan Tolerances Setengah (0.5x):
Hasil ini mirip dengan pengurangan max_iterations menjadi setengah, menemukan inliers yang optimal menjadi berkurang dan menghasilkan distorsi pada gambar.

3. Creating your Augmented Reality application

Implementasi Code (Error):

```
1  #Error
2
3  import numpy as np
4  import cv2
5  import multiprocessing
6  from matplotlib import pyplot as plt
7  from opts import get_opts
8  from loadVid import loadVid
9  from MatchPics import matchPics
10 from planarH import compositeH
11 from helper import computeH_ransac_2
12 from pathlib import Path
13
14 def cropBlackBar(img):
15     gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
16     gray = 255 - gray
17     blur = cv2.GaussianBlur(gray, (3,3), 0)
18     res = cv2.threshold(blur, 235, 255, cv2.THRESH_BINARY)[1]
19     kernel = np.ones((5,5), np.uint8)
20     res = cv2.morphologyEx(res, cv2.MORPH_CLOSE, kernel)
21     res = cv2.morphologyEx(res, cv2.MORPH_OPEN, kernel)
22     contours, _ = cv2.findContours(res, cv2.RETR_EXTERNAL,
23 cv2.CHAIN_APPROX_SIMPLE)
24     x, y, w, h = cv2.boundingRect(contours[0])
```

```

25     return x, y, w, h
26
27 def homographyWarp(args):
28     i, opts, template, cv_cover, cv_book = args
29     template = cv2.resize(template, (cv_cover.shape[1], cv_cover.shape[0]))
30     matches, locs1, locs2 = matchPics(cv_cover, cv_book, opts)
31     bestH2to1, _ = computeH_ransac_2(locs1[matches[:, 0]], locs2[matches[:, 1]], opts)
32     composite_img = compositeH(bestH2to1, template, cv_book)
33     print(f"Frame {i} Processed")
34     return composite_img
35
36 if __name__ == '__main__':
37     opts = get_opts()
38     opts.sigma = 0.12
39     opts.ratio = 0.7
40     opts.inlier_tol = 1.4
41     n_worker = multiprocessing.cpu_count()
42
43     ar_source_mov_path = '.\data\ar_source.mov'
44     ar_source_npy_path = Path('.\ar_source.npy')
45     cv_cover = cv2.imread('.\data\cv_cover.jpg')
46     ar_book_mov_path = '.\data\book.mov'
47     ar_book_npy_path = Path('.\data\book_source.npy')
48
49     ar_source_load = loadVid(ar_source_mov_path)
50     ar_book_load = loadVid(ar_book_mov_path)
51     np.save(ar_source_npy_path, ar_source_load)
52     np.save(ar_book_npy_path, ar_book_load)
53
54     ar_source = np.load(ar_source_npy_path, allow_pickle=True)
55     ar_book = np.load(ar_book_npy_path, allow_pickle=True)
56
57     frame0 = ar_source[0]
58     x, y, w, h = cropBlackBar(frame0)
59     frame0 = frame0[y:y+h, x:x+w]
60     H, W = frame0.shape[:2]
61     width = cv_cover.shape[1] * H / cv_cover.shape[0]
62     wStart, wEnd = np.round([W/2 - width/2, W/2 + width/2]).astype(int)
63     frame0 = frame0[:, wStart:wEnd]
64     new_source = np.array([frame0[y:y+h, x:x+w][:, wStart:wEnd] for f in ar_source])
65
66     args = [(i, opts, new_source[i], cv_cover, ar_book[i]) for i in range(len(new_source))]
67
68     with multiprocessing.Pool(processes=n_worker) as p:
69         ar = p.map(homographyWarp, args)
70

```

```
71 ar = np.array(ar)
72 writer = cv2.VideoWriter('./result/ar.avi', cv2.VideoWriter_fourcc(*'MJPG'), 25,
73 (ar.shape[2], ar.shape[1]))
74
75 for i, f in enumerate(ar):
76     writer.write(f)
77     plt.figure()
78     plt.axis('off')
79     plt.imshow(cv2.cvtColor(f, cv2.COLOR_BGR2RGB))
80     plt.savefig(f'./result/frame_{i}.png')
81     plt.close()
82
83 writer.release()
```

4. Extra Credit

Q4.1x: Make Your AR Real Time

Q4.2x: Create a Simple Panorama

Implementasi code:

```
1 import numpy as np
2 import cv2
3 import sys
4 sys.path.append('./python')
5 from matplotlib import pyplot as plt
6 from MatchPics import matchPics
7 from planarH import computeH_ransac, compositeH
8 from opts import get_opts
9
10 # Konfigurasi untuk pencocokan fitur dan penghitungan homografi
11 opts = get_opts()
12 opts.sigma = 0.12
13 opts.ratio = 0.7
14 opts.inlier_tol = 1.4
15
16 # Muat gambar kiri dan kanan dari panorama
17 left = cv2.imread('./data/pano_left.jpg')
18 right = cv2.imread('./data/pano_right.jpg')
19
20 if left is None or right is None:
21     print("Kesalahan saat memuat gambar")
22 else:
23     print("Ukuran gambar kiri:", left.shape)
```

```
24     print("Ukuran gambar kanan:", right.shape)
25
26     # Lakukan pencocokan fitur antara gambar kiri dan kanan
27     matches, locs1, locs2 = matchPics(left, right, opts)
28
29     # Hitung matriks homografi
30     H, _ = computeH_ransac(locs1[matches[:, 0]], locs2[matches[:, 1]], opts)
31
32     # Dapatkan dimensi dari gambar kiri dan kanan
33     hl, wl = left.shape[:2]
34     hr, wr = right.shape[:2]
35
36     # Warp gambar kanan agar sejajar dengan gambar kiri dan jahit bersama
37     res = cv2.warpPerspective(right, H, (wr + wl, max(hr, hl)))
38     res[:hl, :wl] = left
39
40     # Ubah hasilnya menjadi skala abu-abu dan terapkan thresholding untuk mengidentifikasi
41     area hitam
42     gray = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
43     gray = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY)[1]
44
45     # Temukan kontur dan hitung persegi pembatas untuk memotong area hitam
46     contours, _ = cv2.findContours(gray, cv2.RETR_EXTERNAL,
47     cv2.CHAIN_APPROX_SIMPLE)
48     x, y, w, h = cv2.boundingRect(max(contours, key=cv2.contourArea))
49
50     # Simpan gambar panorama yang telah dipotong
51     cv2.imwrite('./result/panorama.jpg', res[y:y+h, x:x+w])
```


Contoh hasil panorama

Table 3. Contoh Hasil Panorama menggunakan image dari supplements

Left: 	Right 
Panorama 	

Panorama menggunakan kamera HP

Table 4. Contoh Hasil Panorama menggunakan image dari kamera HP

Left:	Right
	
Panorama 	

Dapat dilihat hasil panorama untuk kedua gambar cukup smooth dengan distorsi yang secara sekilas tak terlihat.

Made Arbi Parameswara (23522002)

Keterangan: adapun sumber belajar dari proses coding pada penyelesaian tugas ini berasal dari github.