

**Implementasi dan Analisis Pipeline *Computer Vision* Dasar: *Filtering*,
Edge Detection, *Feature Points*, dan Geometri Kamera**

Disusun untuk memenuhi tugas pada mata kuliah

IF5152/ Computer Vision

Oleh

Kayla Namira Mariadi

NIM : 13522050



**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO & INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2025

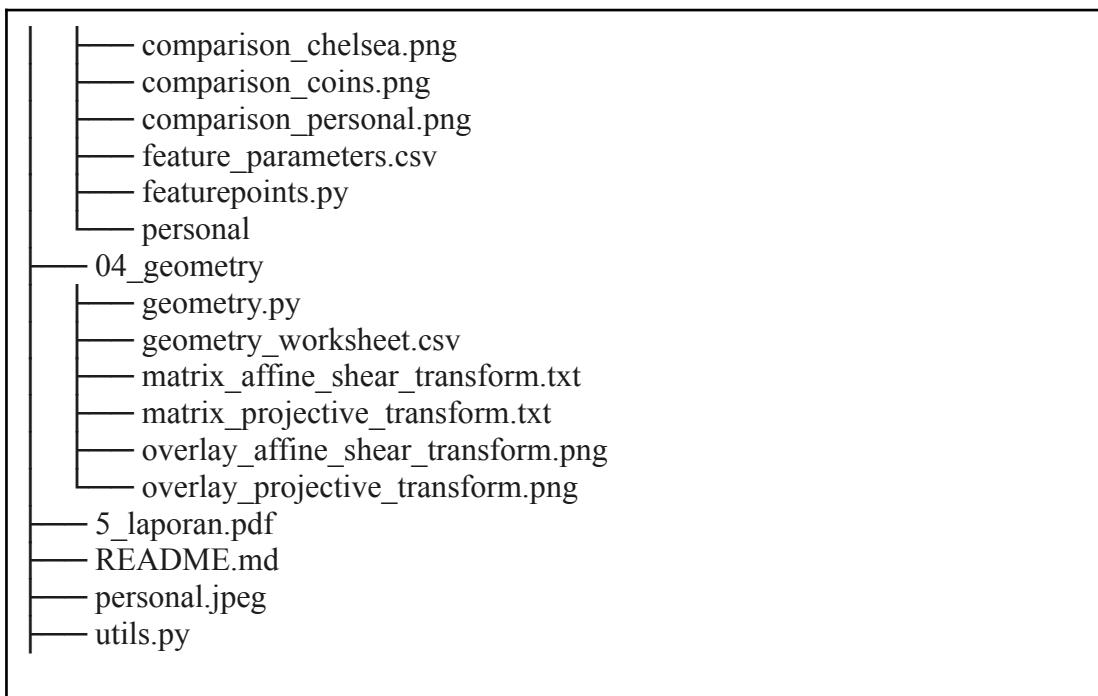
DAFTAR ISI

1. Workflow Pipeline.....	2
1.1 Arsitektur Umum.....	2
2. Proses & Hasil Tiap Fitur.....	4
2.1 Image Filtering.....	4
2.2 Edge Detection & Sampling.....	10
2.3 Feature Points & Corner Detection.....	15
2.4 Camera Geometry & Calibration.....	20
3. Komparasi & Refleksi Pribadi.....	23
3.1 Komparasi Hasil (Gambar Standar vs. Gambar Pribadi).....	23
3.2 Narasi Pilihan Desain Pribadi: Inovasi Kode: Struktur Runner dan Config.....	23
3.3 Refleksi Kendala dan Solusi.....	24

1. Workflow Pipeline

Aplikasi ini terdiri dari satu modul utilitas inti dan tiga modul pemrosesan utama yang dapat dieksekusi secara independen. Berikut struktur file:

```
Kayla Namira Mariadi_13522050_IF5152_CV/
└── 01_filtering
    ├── astronaut
    ├── cameraman
    ├── checkerboard
    ├── chelsea
    ├── coins
    ├── comparison_astronaut.png
    ├── comparison_cameraman.png
    ├── comparison_checkerboard.png
    ├── comparison_chelsea.png
    ├── comparison_coins.png
    ├── comparison_personal.png
    ├── filter_parameters.csv
    ├── filtering.py
    └── personal
└── 02_edge
    ├── astronaut
    ├── cameraman
    ├── checkerboard
    ├── chelsea
    ├── coins
    ├── comparison_astronaut.png
    ├── comparison_cameraman.png
    ├── comparison_checkerboard.png
    ├── comparison_chelsea.png
    ├── comparison_coins.png
    ├── comparison_personal.png
    ├── edge_detection.py
    ├── edge_parameters.csv
    └── personal
└── 03_featurepoints
    ├── astronaut
    ├── cameraman
    ├── checkerboard
    ├── chelsea
    ├── coins
    ├── comparison_astronaut.png
    ├── comparison_cameraman.png
    └── comparison_checkerboard.png
```



1.1 Arsitektur Umum

Arsitektur aplikasi dirancang secara modular untuk memisahkan fungsi-fungsi umum dari logika pemrosesan spesifik.

- `utils.py` (Modul Utilitas Inti):
 - `load_images()`: Bertanggung jawab memuat semua gambar uji (standar dari `skimage.data` dan gambar `personal.jpeg`), serta melakukan pra-pemrosesan awal seperti konversi ke *grayscale*.
 - `save_img()`: Menyimpan gambar hasil olahan ke *disk*.
 - `save_params_to_csv()`: Menyimpan data statistik dan parameter (dalam format `pandas.DataFrame`) ke file `.csv` untuk analisis.
 - `plot_comparison()`: Membuat plot perbandingan (dalam 2 baris) yang menampilkan gambar asli dan semua hasil olahan dalam satu *figure*, lalu menyimpannya.
 - `create_marked_image()` (atau `mark_features`): Fungsi *helper* khusus untuk menggambar penanda fitur pada gambar.
- `run_..._py` (Skrip Runner / Modul Pemroses):
 - Ini adalah titik masuk eksekusi untuk setiap fitur.
 - Setiap skrip import `utils` untuk menggunakan fungsi-fungsi di atas.
 - Setiap skrip mendefinisikan `..._CONFIG` (sebuah *list* berisi *dictionary*), yang mendefinisikan fungsi mana yang akan dipanggil dan parameter apa yang akan digunakan. Ini membuat eksperimen menjadi mudah dikonfigurasi.
 - Setiap skrip memiliki struktur `main() -> run_all_...() -> process_one_image()`.

1.2 Alur Kerja Modul Spesifik

a. Modul Deteksi Tepi (run_edge_detection.py)

1. main() dieksekusi.
2. Direktori output_edges/ dibuat.
3. run_all_edges() dipanggil.
4. utils.load_images() dipanggil untuk memuat semua gambar (misal: cameraman, chelsea, personal, dll.).
5. Aplikasi me-loop setiap gambar yang dimuat.
6. Untuk setiap gambar, process_one_image() dipanggil.
7. Di dalam process_one_image(), aplikasi me-loop setiap konfigurasi di EDGE_CONFIG.
8. Fungsi (filters.sobel atau feature.canny) dipanggil dengan parameter yang sesuai.
9. Hasil gambar individu disimpan ke sub-direktori (misal: output_edges/cameraman/cameraman_sobel.png).
10. Hasil disimpan ke *dictionary comparison_plots*.
11. Data log (parameter dan nama file) ditambahkan ke *list all_params_log*.
12. Setelah *loop* konfigurasi selesai, utils.plot_comparison() dipanggil untuk menyimpan plot perbandingan (misal: comparison_cameraman.png).
13. Setelah *loop* gambar selesai, utils.save_params_to_csv() dipanggil untuk menyimpan all_params_log ke edge_parameters.csv.

b. Modul Deteksi Fitur (run_feature_detection.py)

Alur kerja ini sangat mirip dengan modul pertama, namun difokuskan pada perbandingan detektor fitur (Harris, FAST, ORB, SIFT).

1. main() dieksekusi.
2. Direktori output_features/ dibuat.
3. run_all_features() dipanggil.
4. utils.load_images() dipanggil.
5. Aplikasi me-loop setiap gambar.
6. Untuk setiap gambar, process_one_image() dipanggil.
7. Di dalam process_one_image(), aplikasi me-loop setiap konfigurasi di FEATURE_CONFIG.
8. Logika if/elif menentukan cara memanggil fungsi *helper* (misal _detect_harris yang mengembalikan *coords* dan *responses*, atau _detect_fast yang hanya mengembalikan *coords*).
9. Fungsi detektor (_detect_harris, _detect_orb, _detect_fast, _detect_sift) dipanggil.
10. Statistik utama dikumpulkan: num_features (jumlah fitur) dan mean_response (kekuatan fitur rata-rata).
11. create_marked_image() dipanggil untuk menggambarkan hasil deteksi.
12. Gambar hasil individu disimpan ke sub-direktori.
13. Hasil dan statistik (jumlah fitur) ditambahkan ke comparison_plots.
14. Data log (parameter, num_features, mean_response) ditambahkan ke all_params_log.
15. Setelah *loop* konfigurasi, utils.plot_comparison() dipanggil.

16. Setelah *loop* gambar, `utils.save_params_to_csv()` dipanggil untuk menyimpan `feature_parameters_and_counts.csv`.

c. Modul Transformasi Geometri (`run_geometry.py`)

Modul ini memiliki alur kerja yang berbeda karena tidak memproses *dataset* gambar, melainkan melakukan simulasi spesifik pada satu gambar (checkerboard).

1. `main()` dieksekusi.
2. Direktori `output_geometry/` dibuat.
3. `run_forward_transform()` (Simulasi 1) dipanggil:
 - a. Memuat data.`checkerboard()` secara internal.
 - b. Mendefinisikan 4 titik sumber (lurus) dan 4 titik tujuan (miring).
 - c. Memanggil `estimate_transform()` untuk menghitung Matriks `H_forward`.
 - d. Memanggil `warp()` untuk membuat gambar miring (`warped_image`).
 - e. Memanggil `save_matrix_to_csv()` untuk menyimpan `1_forward_transform_matrix.csv`.
 - f. Memanggil `plot_transform_comparison()` untuk menyimpan `comparison_1_forward_transform.png`.
 - g. Mengembalikan gambar miring dan set titik-titiknya.
4. `run_rectification_transform()` (Simulasi 2) dipanggil, menggunakan output dari langkah 3:
 - a. Titik sumber dan tujuan dibalik (miring -> lurus).
 - b. Memanggil `estimate_transform()` untuk menghitung Matriks `H_rectify`.
 - c. Memanggil `warp()` untuk meluruskan kembali gambar (`rectified_image`).
 - d. Memanggil `save_matrix_to_csv()` untuk menyimpan `2_rectification_transform_matrix.csv`.
 - e. Memanggil `plot_transform_comparison()` untuk menyimpan `comparison_2_rectification.png`.
5. Proses selesai.

2. Proses & Hasil Tiap Fitur

2.1 Image Filtering

a. Teori

Image Filtering adalah teknik dalam pemrosesan citra yang bertujuan untuk memodifikasi atau menyempurnakan gambar. Proses ini umumnya bekerja melalui konvolusi, yaitu dengan menggeser sebuah matriks kecil yang disebut kernel (atau filter) ke seluruh pixel gambar.

Dalam eksperimen ini, tiga jenis filter utama digunakan:

1. Gaussian Filter

Sebuah *low-pass filter* yang bekerja dengan merata-ratakan nilai pixel dengan tetangganya. Pembobotan rata-rata mengikuti distribusi Gaussian, di mana pixel yang lebih dekat ke pusat kernel memiliki bobot lebih besar. Filter ini sangat efektif untuk menghaluskan gambar (*smoothing*) dan mengurangi *noise* Gaussian. Parameter utamanya adalah sigma (standar deviasi), yang mengontrol seberapa besar area "blur".

2. Median Filter

Sebuah non-linear filter. Alih-alih menghitung rata-rata, filter ini mengganti nilai setiap pixel dengan nilai median (nilai tengah) dari piksel-piksel tetangganya. Jendela tetangga ini didefinisikan oleh footprint (misalnya, disk(3)). Filter ini sangat *robust* dalam menghilangkan *salt-and-pepper noise* karena nilai outlier (noise) yang ekstrim akan diabaikan saat proses pengurutan median.

3. Sobel Filter

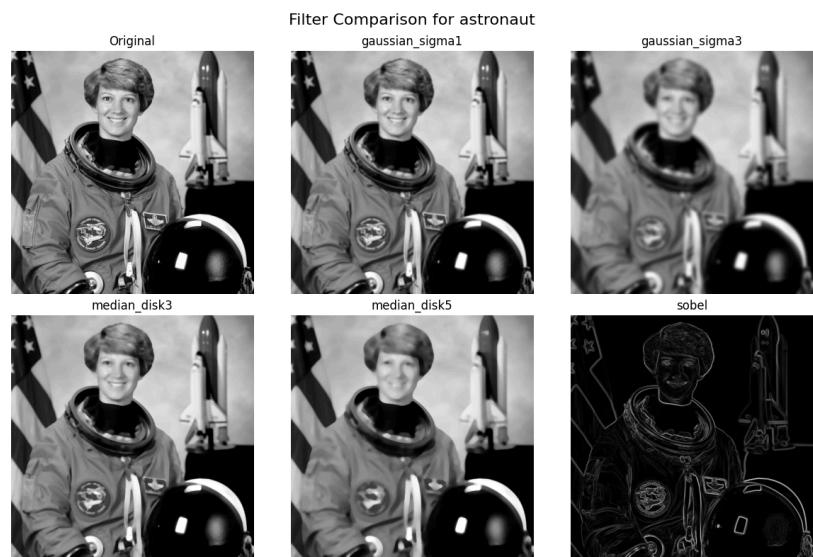
Sebuah *high-pass filter* yang dirancang untuk mendeteksi tepi (edge detection). Filter ini bekerja dengan menghitung aproksimasi gradien (turunan) dari intensitas gambar, baik secara horizontal maupun vertikal. Hasilnya adalah gambar yang berada pada area-area dengan perubahan intensitas yang drastis.

b. Parameter

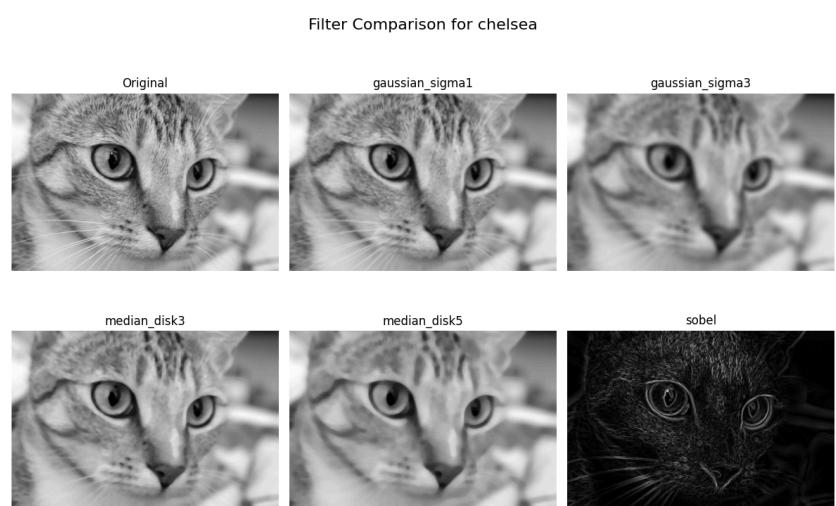
Filter	Nama Parameter	Parameter Uji 1	Parameter Uji 2	Tujuan
Gaussian	sigma	1.0	3.0	Standard smoothing vs Strong smoothing
Median	footprint	disk(3)	disk(5)	Salt-pepper removal, k=3 vs

				Strong S&P removal, k=5
Sobel	-	-	-	Gradient magnitude (edge)

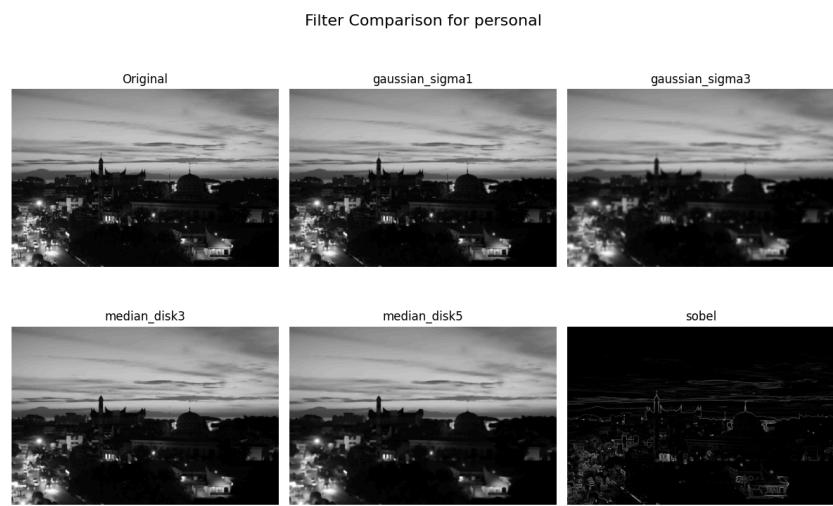
c. Analisis Efek Perubahan Parameter



Gambar 2.1.1 Perbandingan *filtering* pada dataset astronout.

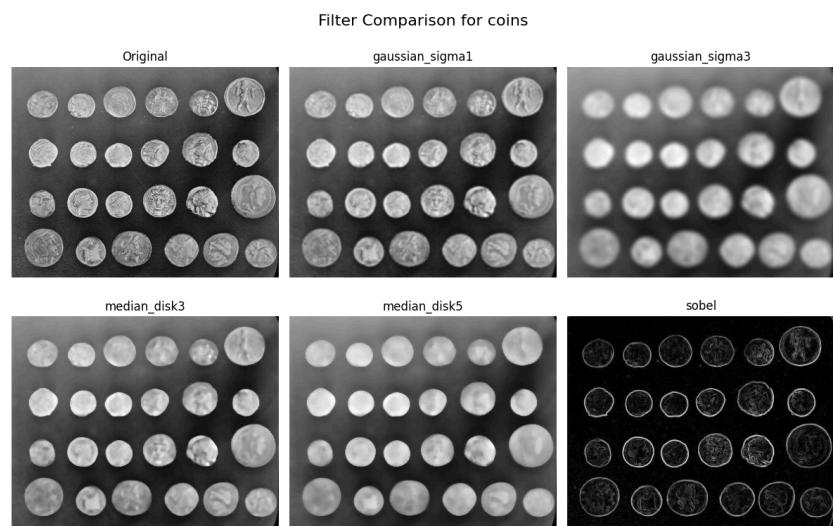


Gambar 2.1.2 Perbandingan *filtering* pada dataset chelsea.

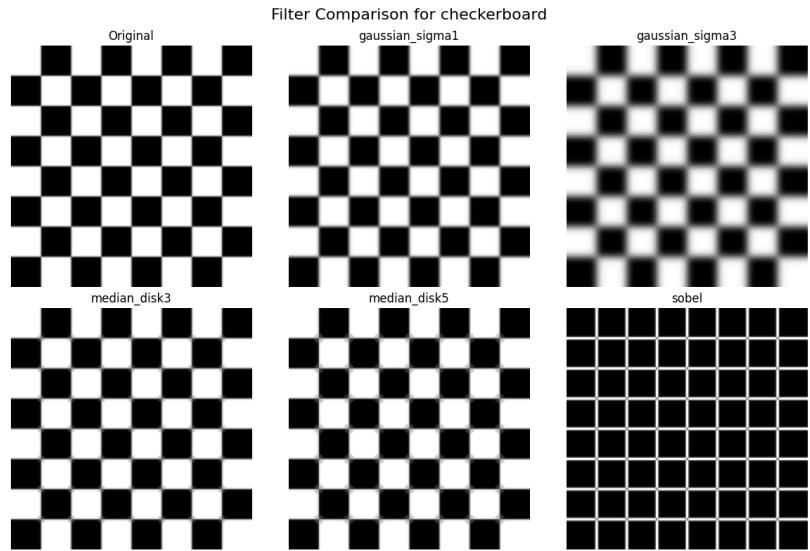


Gambar 2.1.3 Perbandingan *filtering* pada gambar personal.

Sumber: Dokumentasi pribadi penulis



Gambar 2.1.4 Perbandingan *filtering* pada gambar coin.



Gambar 2.1.5 Perbandingan *filtering* pada gambar checkerboard.



Gambar 2.1.6 Perbandingan *filtering* pada gambar cameraman.

1. Analisis sigma pada Gaussian Filter

Perbandingan antara gaussian_sigma1 ($\sigma=1.0$) dan gaussian_sigma3 ($\sigma=3.0$) menunjukkan efek *smoothing* (blur) secara jelas. Pada comparison_cameraman.jpg, $\sigma=1.0$ sudah cukup untuk menghaluskan noise ringan pada langit. Namun, saat σ dinaikkan menjadi 3.0, detail

yang lebih signifikan mulai hilang, tekstur rumput di latar depan menjadi sangat kabur, dan garis-garis pada gedung di latar belakang hampir menyatu.

Pada comparison_astronaut.jpg efek ini sangat terlihat pada tekstur kain baju astronot. $\sigma=1.0$ menghaluskan teksturnya, tetapi polanya masih samar-samar terlihat. $\sigma=3.0$ menghapus tekstur kain tersebut sepenuhnya, membuat gambar terlihat seperti permukaan yang rata dan halus dan lebih blur.

Parameter sigma berbanding lurus dengan kekuatan blur. Peningkatan sigma mengurangi noise, namun dengan risiko kehilangan detail dan tekstur halus pada gambar.

2. Analisis footprint (Kernel Size) pada Median Filter

Perbandingan antara median_disk3 (radius 3 piksel) dan median_disk5 (radius 5 piksel) menunjukkan trade-off antara penghilangan *noise* dan preservasi struktur.

- Pada comparison_coins.jpg, disk(3) berhasil menghaluskan latar belakang sambil tetap menjaga tepi koin tetap tajam. Namun, disk(5) (dengan jendela yang lebih besar) mulai mengaburkan bagian tepi koin, membuatnya terlihat sedikit lebih kecil dan bentuknya sedikit terdistorsi (efek *cartoonish*).
- Pada comparison_cameraman.jpg, Efek "blotchy" (berbercak) atau "airbrushed" sangat terlihat. disk(3) sudah membuat tekstur kulit cameraman terlihat tidak alami. disk(5) memperparah efek ini.

Ukuran footprint (kernel) yang lebih besar (disk(5)) lebih agresif dalam menghaluskan area, tetapi berisiko tinggi merusak struktur tepi objek yang penting dan menciptakan kesan visual "blotchy" yang tidak natural.

3. Analisis Sobel Filter

Filter Sobel tidak memiliki parameter yang divariasikan dalam eksperimen ini, namun berfungsi sebagai *baseline* yang sangat baik untuk deteksi tepi.

- Pada comparison_checkerboard.png
Filter ini menghasilkan garis-garis putih yang jelas di setiap tepi kotak, sementara area datar (interior) menjadi hitam.
- Pada comparison_cameraman.jpg

Sobel berhasil mendeteksi *outline* terlihat, tripod, dan gedung, namun

juga mendeteksi banyak edge pada tekstur rumput, yang menunjukkan sensitivitasnya terhadap *noise* atau tekstur frekuensi tinggi.

2.2 Edge Detection & Sampling

a. Teori

Edge Detection bertujuan untuk menemukan batas-batas objek dalam gambar, yang biasanya ditandai dengan perubahan intensitas yang drastis. Pada tugas ini digunakan 2 metode:

1. Sobel Filter

Filter ini bekerja dengan menghitung gradien atau tingkat perubahan intensitas piksel. Sederhananya, di mana gambar berubah dari gelap ke terang (atau sebaliknya), Sobel akan menghasilkan nilai yang tinggi (terlihat terang). Hasilnya bukan gambar tepi yang bersih (hitam-putih), melainkan peta yang menunjukkan seberapa kuat edge di setiap titik.

2. Canny Edge Detectir

Prosesnya memiliki beberapa tahapan:

1. *Noise Reduction*

Pertama, gambar dihaluskan (di-blur) menggunakan Gaussian filter. Parameter sigma mengontrol seberapa besar blur ini.

2. *Gradient Calculation*

Mirip dengan Sobel, digunakan gradien untuk mendapatkan besar dan arahnya.

3. *Non-Maximum Suppression*

Step ini bertujuan untuk menipiskan *edge* tebal dari langkah sebelumnya menjadi garis tipis 1 pixel.

4. *Hysteresis Thresholding*

Step ini menggunakan dua nilai ambang batas (`low_threshold` dan `high_threshold`) untuk memutuskan mana yang benar-benar *edge* dan mana yang *noise*.

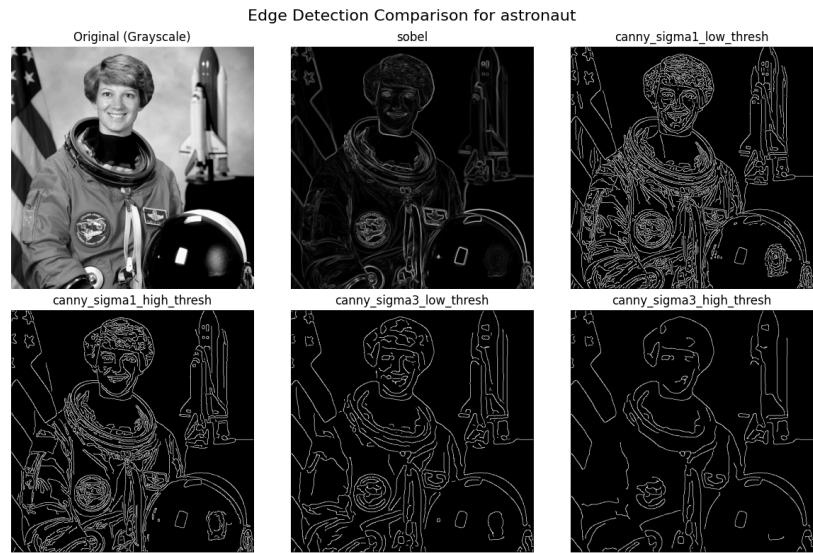
- *Edge* yang lebih kuat dari `high_threshold` pasti diambil

- *Edge* yang lebih lemah dari low_threshold dibuang
- *Edge* yang berada diantara keduanya hanya akan diambil jika ia terhubung dengan *strong edge*.

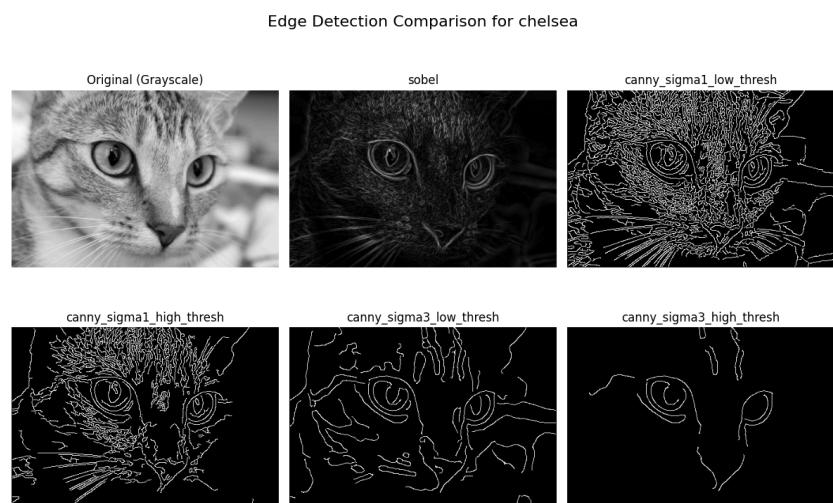
b. Parameter

Nama Konfigurasi	Metode	Parameter	Tujuan
sobel	filters.sobel	(Default)	Metode dasar sebagai pembanding.
canny_sigma1_low_thre sh	feature.canny	sigma: 1.0, low: 0.05, high: 0.15	Sigma rendah (detail tinggi), threshold rendah (sensitif).
canny_sigma1_high_thr esh	feature.canny	sigma: 1.0, low: 0.1, high: 0.3	Sigma rendah (detail tinggi), threshold tinggi (lebih ketat).
canny_sigma3_low_thre sh	feature.canny	sigma: 3.0, low: 0.05, high: 0.15	Sigma tinggi (halus), threshold rendah (sensitif).
canny_sigma3_high_thr esh	feature.canny	sigma: 3.0, low: 0.1, high: 0.3	Sigma tinggi (halus), threshold tinggi (paling ketat/bersih).

c. Analisis Efek Perubahan Parameter

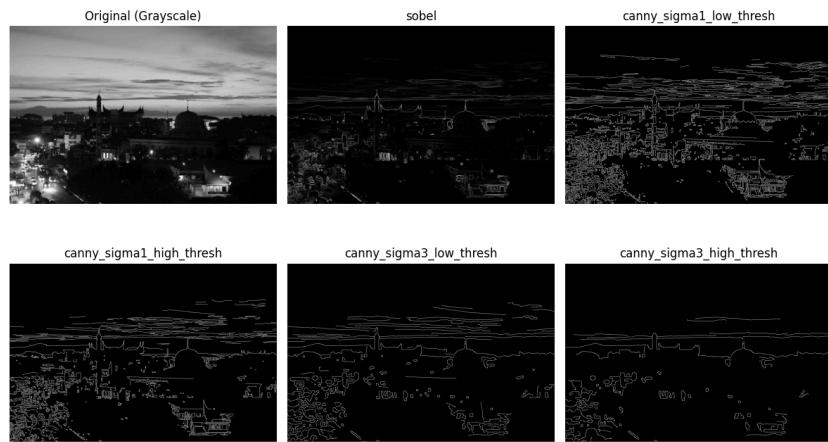


Gambar 2.2.1 Perbandingan *Edge Detection* pada dataset astronout.



Gambar 2.2.2 Perbandingan *Edge Detection* pada dataset chelsea.

Edge Detection Comparison for personal



Gambar 2.2.3 Perbandingan *Edge Detection* pada gambar personal.

Sumber: Dokumentasi pribadi penulis

1. Perbandingan Sobel vs. Canny

Seperti yang terlihat di kedua gambar, sobel menghasilkan garis *edge* yang lebih tebal dan menangkap banyak *noise*. Sementara itu, hasil Canny menghasilkan gambar dengan garis *edge* yang lebih tipis sehingga lebih mudah diinterpretasi sebagai garis batas objek.

2. Efek perubahan sigma (tingkat gaussian blur)

Parameter sigma menentukan seberapa banyak gambar dihaluskan sebelum edge dideteksi.

- sigma=1.0 (Blur Rendah):

Sigma yang lebih rendah ini lebih sensitif terhadap detail halus dan juga *noise*. Pada gambar Chelsea (*canny_sigma1_low_thresh*), kita bisa melihat hampir setiap helai bulu terdeteksi sebagai *edge*. Pada gambar Astronaut, detail pada pakaian, wajah, dan latar belakang roket tertangkap dengan jelas. Pada gambar Personal (pemandangan kota),

hampir semua detail arsitektur bangunan terdeteksi, serta juga banyak noise di area langit.

- $\sigma=3.0$ (Blur Tinggi):

Sigma yang lebih tinggi lebih memiliki efek *smoothing*, noise dan detail-detail kecil jadi lebih minim. Pada gambar Chelsea, tekstur bulu sangat berkurang dan hanya tersisa garis besar wajahnya (*canny_sigma3_low_thresh*). Pada gambar Astronaut, fitur wajah dan lipatan baju menjadi jauh lebih sederhana dan halus. Pada gambar Personal, detail-detail kecil di gedung hilang dan mayoritas tersisa *outline* gedungnya saja, langit juga lebih bersih dari *noise*.

3. Efek Perubahan *low_threshold* dan *high_threshold*

Thresholds menentukan seberapa kuat sebuah perubahan pixel harus dianggap sebagai edge.

- Threshold Rendah (low: 0.05, high: 0.15):

Edge yang lebih lemah akan lebih mudah lolos dan disambungkan dengan edge yang kuat. Hasilnya, gambar cenderung memiliki lebih banyak garis. Jika dibandingkan dengan sigma yang sama (misal, *canny_sigma1_low_thresh* vs *canny_sigma1_high_thresh*), versi *low_thresh* terlihat lebih ramai.

- Threshold Tinggi (low: 0.1, high: 0.3):

Hanya edge yang benar-benar kuat yang akan lolos. Hasilnya adalah gambar yang lebih bersih dengan lebih sedikit garis *noise*. Efek ini paling jelas terlihat pada $\sigma=3.0$ (*canny_sigma3_low_thresh* dan *canny_sigma3_high_thresh*) pada gambar Chelsea. Di versi *high_thresh*, hanya tersisa fitur mata dan hidung. Hal yang sama terjadi pada gambar Personal, di mana versi *high_thresh* menghasilkan garis yang lebih

terputus-putus karena hanya bagian *edge* yang paling kuat yang terdeteksi.

2.3 Feature Points & Corner Detection

a. Teori

Feature Points (atau *Interest Points*) adalah titik-titik spesifik pada gambar yang memiliki keunikan dan mudah dilacak. Titik-titik ini dapat dikenali lagi bahkan jika gambar diputar, diubah ukurannya, atau mengalami perubahan pencahayaan.

1. Harris Corner Detector

Ini adalah detektor sudut (corner) klasik. Detektor ini bekerja dengan cara “meletakkan” sebuah jendela kecil di atas pixel. Jika menggeser jendela itu ke arah manapun menghasilkan gradien/perubahan besar, maka itu adalah *corner*.

2. FAST (Features from Accelerated Segment Test)

Sesuai namanya, ini adalah detektor yang sangat cepat. Detektor ini bekerja dengan melihat lingkaran 16 pixel di sekitar satu titik. Jika ada beberapa pixel berurutan di lingkaran itu yang semuanya jauh lebih terang atau lebih gelap dari titik pusat, maka itu dianggap sebagai fitur. Implementasi yang digunakan juga menyertakan Non-Maximum Suppression (NMS) untuk memastikan detektor hanya mendapatkan corner, bukan edge.

3. ORB (Oriented FAST and Rotated BRIEF)

Ini adalah detektor yang lebih modern. Detektor ini menggunakan FAST untuk menemukan *keypoints* dengan cepat, lalu menambahkan orientasi (arah) pada setiap fitur. Ini membuatnya *robust* (tangguh) terhadap rotasi.

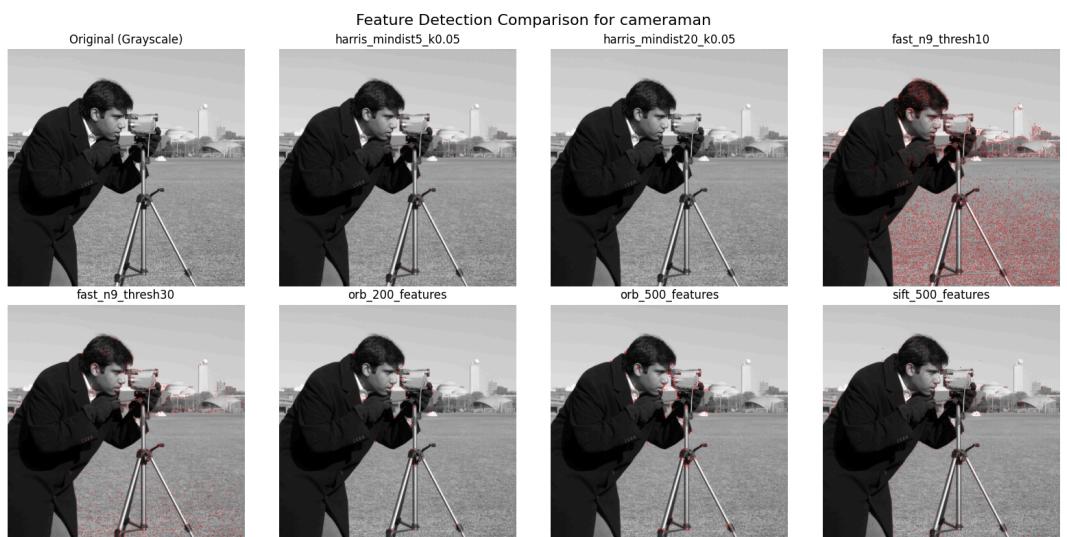
4. SIFT (Scale-Invariant Feature Transform)

Ini adalah salah satu detektor paling kuat. Keunggulan utamanya adalah *scale-invariant*, artinya ia bisa menemukan fitur yang sama baik saat gambar di *zoom in* maupun *zoom out*. Detektor ini juga *robust* terhadap rotasi dan perubahan pencahayaan.

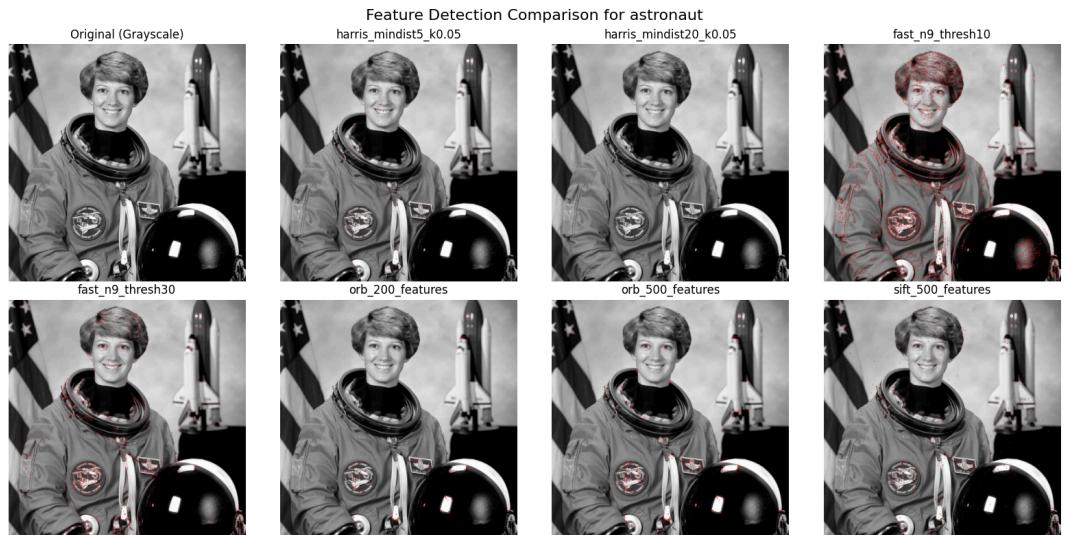
b. Parameter

Detektor	Parameter yang Diuji	Nilai 1	Nilai 2	Tujuan
Harris	min_distance	5	20	Mengontrol berdekatan seberapa fitur boleh terdeteksi.
FAST	threshold	0.1	0.3	Mengontrol seberapa "jelas" fitur harus menonjol.
ORB	n_keypoints	200	500	Menargetkan jumlah fitur "terbaik" yang ingin ditemukan.
SIFT	n_keypoints	500	-	Menargetkan 500 fitur "terbaik".

c. Analisis Efek Perubahan Parameter



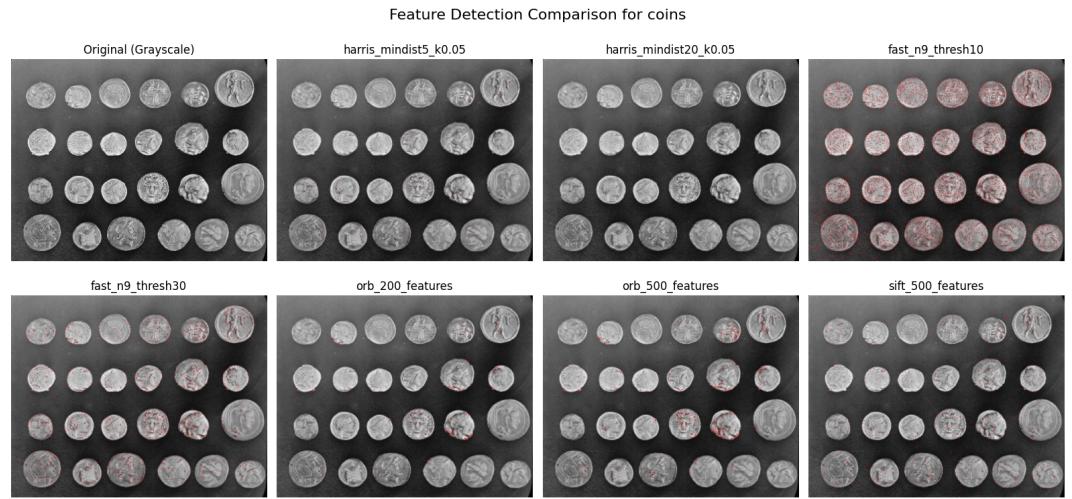
Gambar 2.3.1 Perbandingan min_distance pada dataset cameraman.



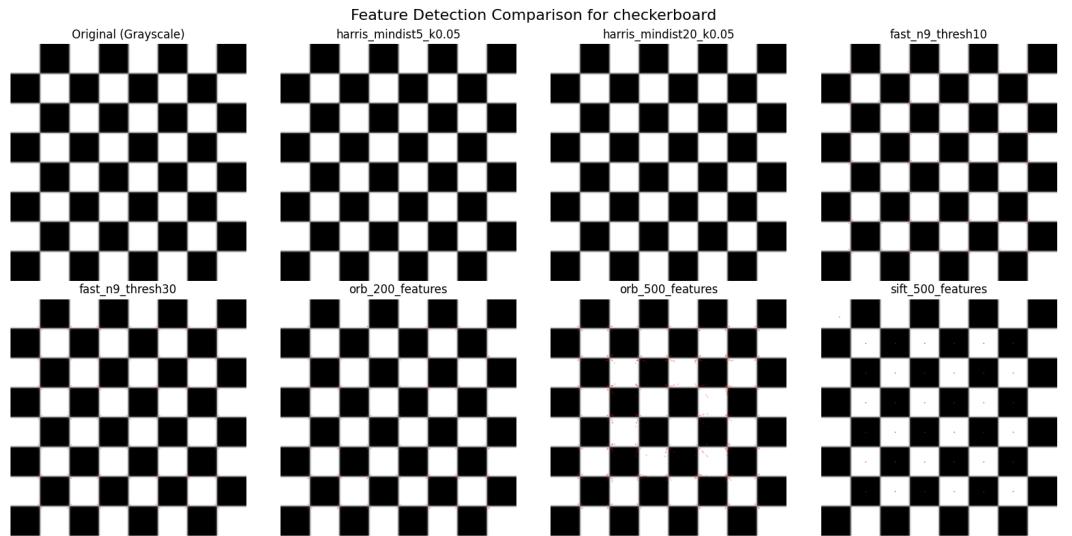
Gambar 2.3.2 Perbandingan min_distance pada dataset chelsea.



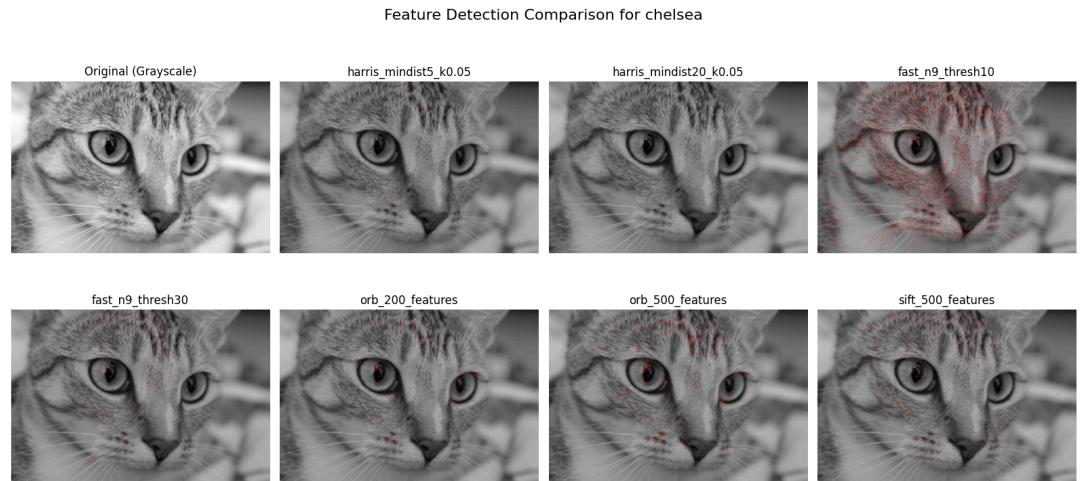
Gambar 2.3.3 Perbandingan min_distance pada dataset personal.



Gambar 2.3.4 Perbandingan min_distance pada dataset coin.



Gambar 2.3.5 Perbandingan min_distance pada dataset checkerboard.



Gambar 2.3.6 Perbandingan min_distance pada dataset chelsea.

1. Analisis min_distance (Harris)

Parameter min_distance sangat efektif untuk mendistribusikan fitur agar tidak menumpuk di satu tempat. Pada gambar cameraman, harris_mindist5 menemukan banyak fitur di area tripod, wajah, dan gedung, serta beberapa tipik di rumput. Sementara itu, harris_mindist20 mengurangi semua *noise* di rumput, dan hanya menyisakan fitur-fitur yang paling kuat dan tersebar di area tripod, bahu cameraman, dan gedung.

2. Analisis threshold (FAST)

Parameter threshold pada FAST mengontrol seberapa selektif detektornya. Pada gambar cameraman, fast_cv2_thresh_10 (threshold rendah) sangat sensitif. Ia mendekripsi ratusan fitur, mendekripsi hampir semua sudut kecil di tripod, gedung, dan bahkan tekstur di rumput. fast_cv2_thresh_30 (threshold tinggi) mengurangi jumlah fitur. Hanya corner yang paling jelas dan paling tajam yang lolos, seperti beberapa titik di tripod dan outline gedung.

Pada gambar astronaut, thresh_10 mendekripsi banyak titik di area mata, hidung, baju, dan rambut. thresh_30 jauh lebih selektif, hanya fokus pada sudut mata, rambut, dan area baju yang paling kontras.

3. Analisis n_keypoints (ORB & SIFT)

Parameter n_keypoints pada ORB dan SIFT bertujuan untuk mencari dan mengembalikan N fitur terbaik yang bisa ditemukannya. Pada gambar personal, orb_200_features berhasil menemukan 200 fitur terkuat, yang sebagian besar terkonsentrasi di area atap gedung. orb_500_features menemukan 500 fitur. Terlihat bahwa 300 fitur tambahan adalah fitur-fitur yang lebih lemah dan menyebar ke area jalan.

Pada gambar astronaut, hal yang sama terjadi. orb_200 fokus pada area logo pada baju yang kelihatan bertekstur, sementara orb_500 dan sift_500 menyebar lebih luas untuk mencakup tekstur kain di baju, mata, dan area leher baju. Hal ini menunjukkan bahwa SIFT dan ORB sangat baik dalam menemukan fitur di area bertekstur, tidak hanya di corner yang tajam seperti Harris.

2.4 Camera Geometry & Calibration

a Teori

Transformasi Geometri adalah proses mengubah hubungan spasial antar pixel dalam sebuah gambar. Ini adalah dasar proses untuk meluruskan gambar, membuat panorama, atau mensimulasikan perubahan sudut pandang.

Saya menggunakan koordinat homogen dan matriks transformasi 3x3 untuk melakukan operasi ini. Dengan representasi ini, kita dapat menggabungkan operasi kompleks (seperti rotasi, translasi, dan perspektif) ke dalam satu perkalian matriks.

Dalam eksperimen ini, saya mensimulasikan dua tipe transformasi utama pada gambar checkerboard:

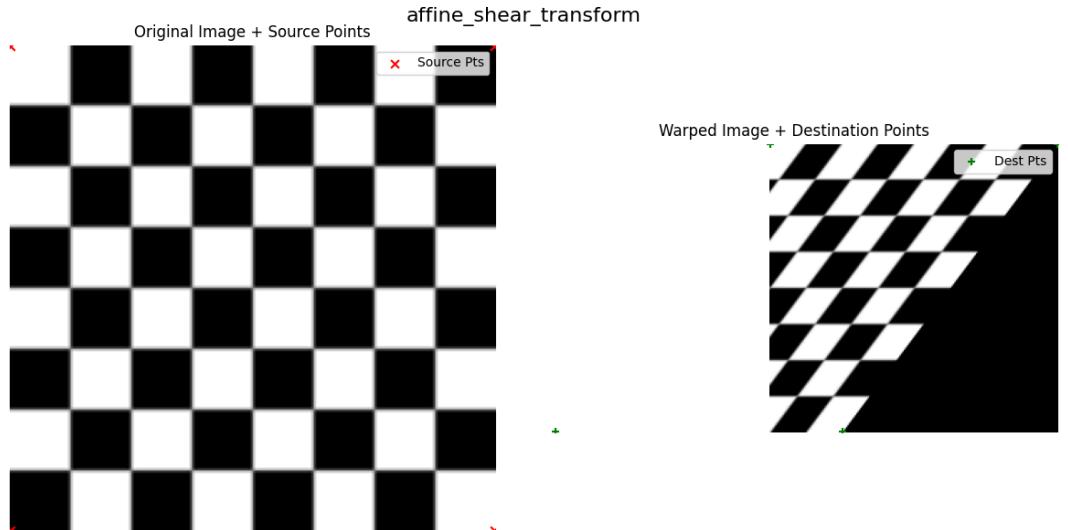
1. *Affine Transform*

Ini adalah transformasi yang lebih sederhana (6 derajat kebebasan). Transformasi ini menjaga garis-garis paralel agar paralel. Ia bisa melakukan rotasi, translasi, scaling (pembesaran), dan shear (efek condong). Untuk mengestimasinya, kita hanya perlu 3 pasang titik (sumber dan tujuan).

2. Projective Transform (Homography):

Ini adalah transformasi yang lebih umum dengan 8 derajat kebebasan. Transformasi ini tidak menjaga garis paralel. Untuk mengestimasinya, kita perlu 4 pasang titik.

b Hasil Eksperimen dan Analisis



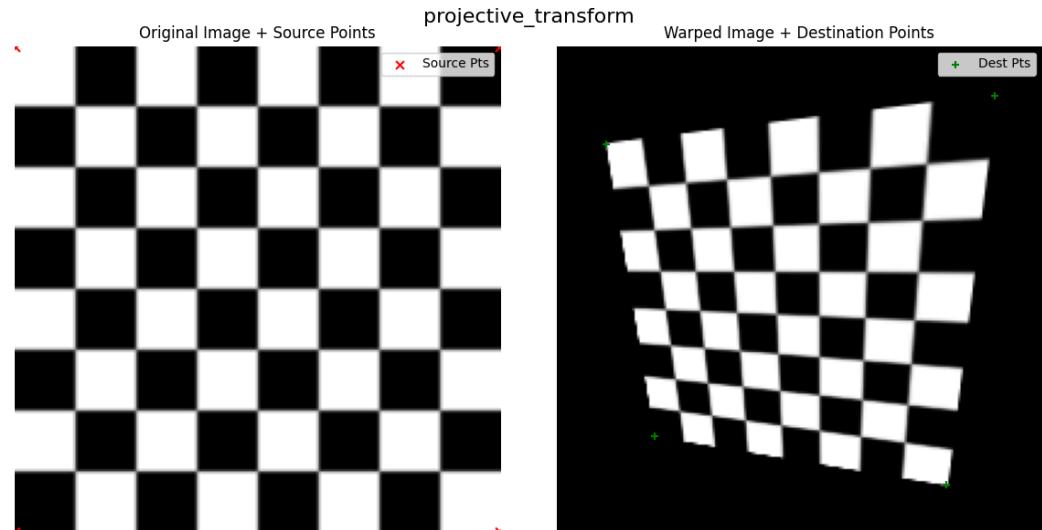
Gambar 2.4.1 *Overlay* hasil affine_shear_transform. Kiri: Gambar asli + 3 titik sumber (merah). Kanan: Hasil warp + 3 titik tujuan (hijau).

Simulasi ini mengaplikasikan efek "condong" atau shear pada gambar. Seperti yang terlihat pada Gambar 2.4.1, dua titik atas gambar tetap di tempatnya. Namun, titik kanan-bawah ([199, 199]) titik digeser secara horizontal ke kiri ke posisi [50, 199]. Hasilnya adalah gambar yang condong ke kiri, di mana garis-garis vertikal menjadi miring, tetapi garis-garis horizontal tetap paralel.

```
04_geometry > matrix_affine_shear_transform.txt
1  Matriks Estimasi (affine_shear_transform)
2  Tipe:AffineTransform
3
4  1.0000 -0.7487 0.0000
5  0.0000 1.0000 0.0000
6  0.0000 0.0000 1.0000
```

Gambar 2.4.2 Matriks hasil affine_shear_transform

Matriks ini menjelaskan efek shear. Nilai -0.7487 pada baris 1, kolom 2 adalah faktor *shear* horizontal. Artinya, koordinat x baru dihitung sebagai $x' = 1 \cdot x - 0.7487 \cdot y + 0$. Semakin besar nilai y (semakin ke bawah pixelnya), semakin besar pula pergeserannya ke kiri (karena dikali faktor negatif), yang persis seperti efek "condong" yang terlihat pada gambar.



Gambar 2.4.3 Overlay hasil projective_transform. Kiri: Asli + 4 titik sumber. Kanan: Hasil warp + 4 titik tujuan

Simulasi ini mengaplikasikan perubahan sudut pandang (perspektif) pada gambar. Gambar 2.4.3 menunjukkan bagaimana gambar checkerboard yang persegi sempurna diubah bentuknya menjadi trapesium. Garis-garis yang tadinya paralel kini tidak lagi paralel. Garis-garis horizontal dan vertikal pada checkerboard hasil warp terlihat "menyempit" dan akan bertemu di vanishing point di luar gambar. Ini adalah ciri khas dari transformasi perspektif (Homografi).

```

04_geometry > matrix_projective_transform.txt
1 Matriks Estimasi (projective_transform)
2 Tipe: ProjectiveTransform
3
4 0.5456 0.1579 20.0000
5 -0.1292 0.8327 40.0000
6 -0.0014 0.0014 1.0000

```

Gambar 2.4.4 Matriks hasil projective_transform.

Pada row terakhir, tidak seperti matriks Affine ([0, 0, 1]), matriks ini memiliki nilai [-0.0014, 0.0014]. Nilai-nilai yang bukan nol ini bertanggung jawab atas pembagian perspektif. Merekalah yang secara matematis menyebabkan garis-garis paralel menjadi konvergen (bertemu di satu titik), sehingga berhasil mensimulasikan perubahan sudut pandang 3D.

Tabel berikut merangkum titik-titik yang digunakan untuk mengestimasi setiap matriks transformasi, berdasarkan file geometry Worksheet.csv.

transform_name	type	src_points_used	dst_points_used
projective_transform	Projective Transform	[[0, 0], [199, 0], [199, 199], [0, 199]]	[[20.0, 40.0], [180.0, 20.0], [160.0, 180.0], [40.0, 160.0]]
affine_shear_transform	AffineTransform	[[0, 0], [199, 0], [199, 199]]	[[0, 0], [199, 0], [50, 199]]

3. Komparasi & Refleksi Pribadi

3.1 Narasi Pilihan Desain Pribadi: Inovasi Kode: Struktur Runner dan Config

Pilihan desain utama saya adalah untuk membuat arsitektur modular meliputi:

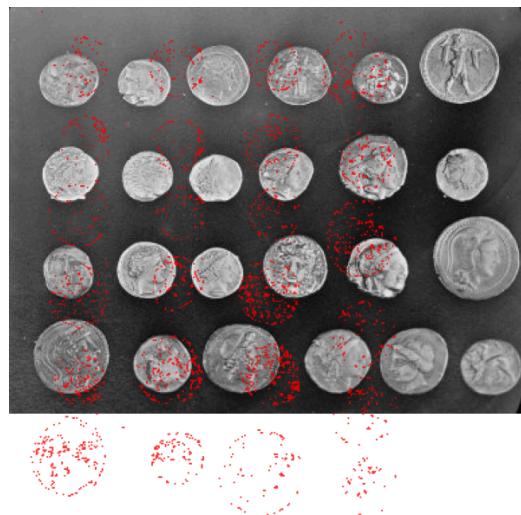
- utils.py: Satu file helper pusat untuk menangani semua logika berulang (memuat gambar, menyimpan file, membuat plot perbandingan, dan menggambar penanda fitur).
- Runner & Config: Setiap modul (filtering.py, feature_points.py, dll.) adalah runner yang logikanya dikendalikan oleh list dictionary ..._CONFIG di bagian atas file. Untuk bereksperimen dengan parameter baru (misal: mengubah sigma Canny atau threshold FAST), saya hanya perlu mengedit CONFIG, tanpa menyentuh logika looping atau penyimpanan file.

3.3 Refleksi Kendala dan Solusi

Kendala awal dalam proyek ini adalah membuat detektor FAST berfungsi. Untuk mencoba FAST, saya mencoba skimage.feature.corner_fast. Hasilnya aneh, detektor malah mendeteksi tepi (edges), bukan sudut (corners). Saya menyadari bahwa skimage.corner_fast (dalam mode default) hanya mengembalikan mask mentah dari semua piksel yang memenuhi kriteria FAST, tanpa Non-Maximum Suppression (NMS) untuk menyaring dan hanya menyisakan corner terbaik.

Daripada mencoba mengimplementasikan NMS secara manual, saya memutuskan untuk pindah library. Karena saya sudah mengimpor opencv-contrib-python untuk SIFT, saya beralih ke cv2.FastFeatureDetector_create. cv2 juga tidak langsung bekerja. Saya harus menyelesaikan 2 masalah:

- TypeError: cv2 mewajibkan threshold berupa integer (misal: 10, 30), bukan float (0.01) seperti skimage.
- Bahkan setelah threshold diset sebagai integer, masih didapatkan hasil yang tidak diharapkan



Gambar 3.2.1 Detektor coin dengan Fast, n=9, threshold=0.1, pada dataset coin.

Solusinya adalah menemukan dan mengaktifkan parameter nonmaxSuppression=True saat membuat detektor.

Kendala selanjutnya yaitu tipe data yang digunakan saat mencoba FAST (float vs. uint8). Di awal, detektor FAST saya tidak menemukan fitur apa pun pada gambar, bahkan ketika threshold di set 0. Saya menemukan bahwa skimage memuat cameraman sebagai uint8 (rentang 0-255), tetapi astronaut dan gambar pribadi saya sebagai float (rentang 0.0-1.0). Parameter di CONFIG (misal threshold_rel=0.01) diatur untuk float dan gagal total pada uint8. Solusi yang saya coba yaitu dengan menstandardisasi semua pemrosesan. Di awal process_one_image untuk deteksi fitur, saya secara eksplisit mengubah semua gambar input menjadi float menggunakan img_gray = utils.to_gray(img_as_float(img)). Dengan begini, semua detektor menerima tipe data yang konsisten.