

**Laporan Tugas Individu**  
**IF5152 Visi Komputer**

**Aplikasi Sederhana Integratif: Materi Minggu 3-6**



Disusun Oleh  
**Fabian Radenta Bangun 13522105**

Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
Semester I - 2025

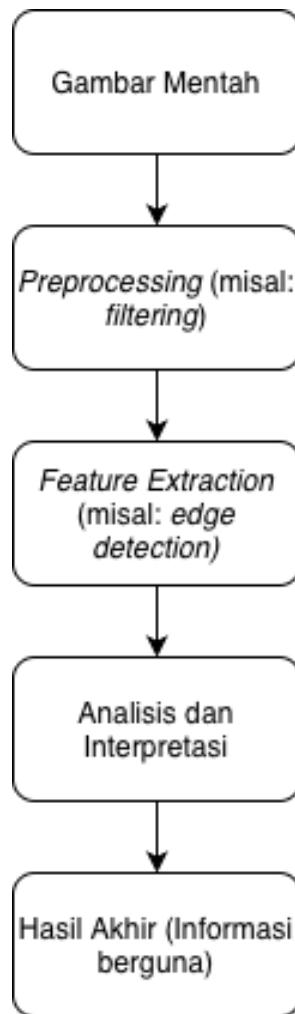
## DAFTAR ISI

Laporan Tugas Individu.....	1
DAFTAR ISI.....	2
WORKFLOW PIPELINE.....	3
Penjelasan Tahapan Pipeline.....	4
Tahap 1: Pra-pemrosesan (Preprocessing).....	4
Tahap 2: Ekstraksi Fitur (Feature Extraction).....	4
Tahap 3: Analisis dan Interpretasi.....	4
Proses dan Hasil Tiap Fitur.....	5
Filtering.....	5
Teori Singkat.....	5
Implementasi dan Hasil.....	6
Analisis Hasil.....	8
Edge Detection.....	9
Teori Singkat.....	9
Implementasi dan Hasil.....	9
Analisis Hasil.....	12
Feature Points.....	13
Teori Singkat.....	13
Implementasi dan Hasil.....	13
Analisis Hasil.....	16
Geometry.....	17
Teori Singkat.....	17
Implementasi dan Hasil.....	17
Analisis Hasil.....	19
Komparasi dan Refleksi Pribadi.....	20
Komparasi Gambar Standar vs. Gambar Pribadi.....	20
Refleksi Proses Penggerjaan.....	21
Lampiran.....	22

## WORKFLOW PIPELINE

Dalam *Computer Vision*, sebuah *pipeline* (alur kerja) merujuk pada serangkaian langkah pemrosesan yang dilakukan secara berurutan untuk mengubah data gambar mentah menjadi informasi yang bermakna. Setiap langkah dalam *pipeline* mengambil *output* dari langkah sebelumnya sebagai *input*, memprosesnya, dan meneruskannya ke langkah berikutnya. Pendekatan modular ini memungkinkan pengembangan sistem yang kompleks namun terstruktur.

Meskipun pada proyek ini masing-masing pemrosesan dipisah dalam beberapa modul, pengimplementasian alur kerja inti dapat disusun menjadi sebuah *pipeline* yang lengkap. Diagram di bawah ini mengilustrasikan alur kerja konseptual yang menunjukkan bagaimana modul-modul yang telah dikembangkan dapat diintegrasikan.



Gambar 1.1 Diagram alur

## Penjelasan Tahapan Pipeline

### Tahap 1: Pra-pemrosesan (*Preprocessing*)

Tahap ini adalah langkah awal dan krusial dalam setiap *pipeline*. Tujuannya adalah untuk meningkatkan kualitas gambar dan mempersiapkannya untuk analisis lebih lanjut. Pada proyek ini, tahap ini direpresentasikan oleh Modul *01\_filtering*.

- Proses: Teknik seperti *Gaussian Blur* diterapkan untuk mengurangi *noise* acak. Sementara, *Median Blur* digunakan untuk menghilangkan *noise* tipe *salt-and-pepper*.
- Hasil: Sebuah gambar yang lebih bersih dan halus, di mana fitur-fitur yang tidak relevan (*noise*) telah diminimalkan.

### Tahap 2: Ekstraksi Fitur (*Feature Extraction*)

Setelah gambar dibersihkan, tahap selanjutnya adalah mengekstraksi informasi atau fitur-fitur penting dari gambar. Fitur ini adalah representasi sederhana dari konten gambar yang lebih kompleks. Proyek ini mengimplementasikan dua jenis ekstraksi fitur utama:

- Deteksi Tepi (Modul *02\_edge*): Menggunakan algoritma seperti *Canny* untuk mengidentifikasi kontur atau batas-batas objek dalam gambar. Hasilnya adalah sebuah *edge map* yang merepresentasikan struktur geometris dari citra.
- Deteksi Titik Fitur (Modul *03\_featurepoints*): Menggunakan algoritma SIFT untuk menemukan titik-titik unik atau "menarik" pada gambar, seperti sudut dan area bertekstur. Outputnya adalah sekumpulan *keypoints* beserta deskriptornya.

### Tahap 3: Analisis dan Interpretasi

Dengan fitur-fitur yang telah diekstraksi, tahap akhir adalah melakukan analisis untuk mencapai tujuan spesifik aplikasi. Meskipun tidak diimplementasikan secara eksplisit sebagai satu kesatuan, modul-modul ini menyediakan dasar untuk analisis tingkat tinggi, seperti:

- Kalibrasi Kamera (Modul *04\_geometry*): Menggunakan fitur sudut dari papan catur untuk mengestimasi parameter kamera.
- Pengenalan Objek: Fitur *SIFT* yang diekstrak dapat dicocokkan dengan *database* fitur untuk mengidentifikasi objek.

- Pelacakan Objek: Kontur dari deteksi tepi dapat digunakan untuk melacak pergerakan objek antar *frame* video.

Secara keseluruhan, proyek ini membangun komponen-komponen esensial yang, ketika digabungkan dalam pipeline seperti yang diilustrasikan, dapat digunakan untuk menyelesaikan berbagai masalah kompleks dalam domain *Computer Vision*.

## Proses dan Hasil Tiap Fitur

### *Filtering*

#### Teori Singkat

*Image Filtering* atau penyaringan citra adalah salah satu teknik pra-pemrosesan yang paling fundamental dalam *Computer Vision*. Tujuannya adalah untuk memodifikasi atau meningkatkan atribut sebuah gambar, seperti mengurangi *noise*, menghaluskan (*blurring*), atau menajamkan (*sharpening*) citra. Operasi *filtering* bekerja dengan melakukan konvolusi antara gambar input dengan sebuah *kernel* (atau *filter*), yang merupakan matriks kecil. Nilai-nilai dalam *kernel* ini menentukan efek yang akan dihasilkan pada gambar.

Dalam proyek ini, dua jenis filter linear dan nonlinear yang populer diimplementasikan:

1. **Gaussian Blur**: Sebuah *filter* linear yang mengganti nilai setiap piksel dengan rata-rata terbobot dari piksel-piksel di sekitarnya. Pembobotan ini mengikuti distribusi *Gaussian*, di mana piksel yang lebih dekat ke pusat memiliki pengaruh yang lebih besar. *Filter* ini sangat efektif untuk mengurangi *noise* acak (*Gaussian noise*) dan menghaluskan gambar.
2. **Median Blur**: Sebuah *filter* non-linear yang mengganti nilai setiap piksel dengan nilai median dari semua piksel di dalam jendela *kernel*. Karena menggunakan median, filter ini sangat *robust* terhadap *noise* ekstrem seperti *salt-and-pepper noise* (bintik hitam dan putih) dan cenderung lebih baik dalam mempertahankan ketajaman tepi dibandingkan *Gaussian Blur*.

## Implementasi dan Hasil

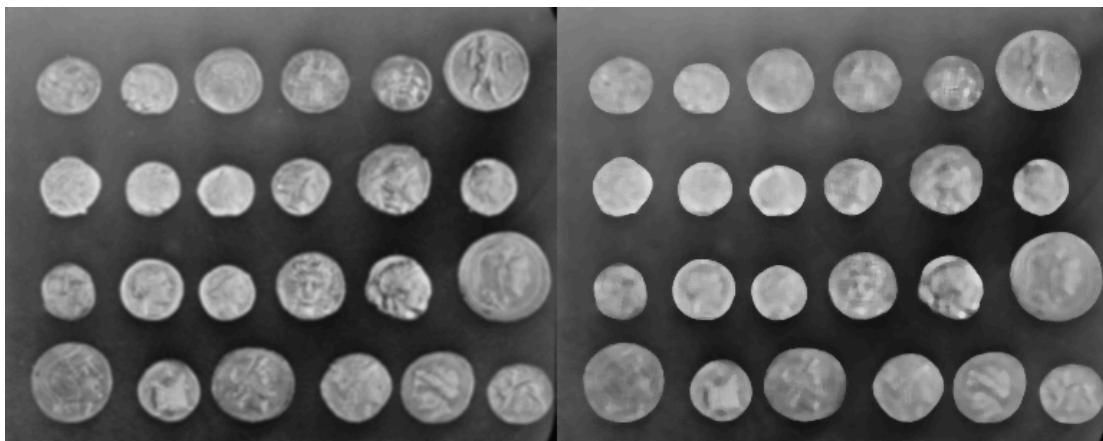
Implementasi *filtering* dilakukan menggunakan fungsi bawaan dari *library OpenCV*, yaitu *cv2.GaussianBlur()* dan *cv2.medianBlur()*. Parameter utama yang diatur adalah ukuran *kernel*, yang menentukan area lingkungan piksel yang dipertimbangkan dalam perhitungan. Tabel berikut merangkum parameter yang digunakan dan *file output* yang dihasilkan untuk setiap gambar.

Gambar Sumber	Filter	Parameter	Nilai Parameter	Output File
cameraman	Gaussian Blur	Ukuran Kernel	(5, 5)	cameraman_gaussian.png
cameraman	Median Blur	Ukuran Kernel	5	cameraman_median.png
coin	Gaussian Blur	Ukuran Kernel	(5, 5)	coin_gaussian.png
coin	Median Blur	Ukuran Kernel	5	coin_median.png
custom	Gaussian Blur	Ukuran Kernel	(5, 5)	free_image_gaussian.png
custom	Median Blur	Ukuran Kernel	5	free_image_median.png

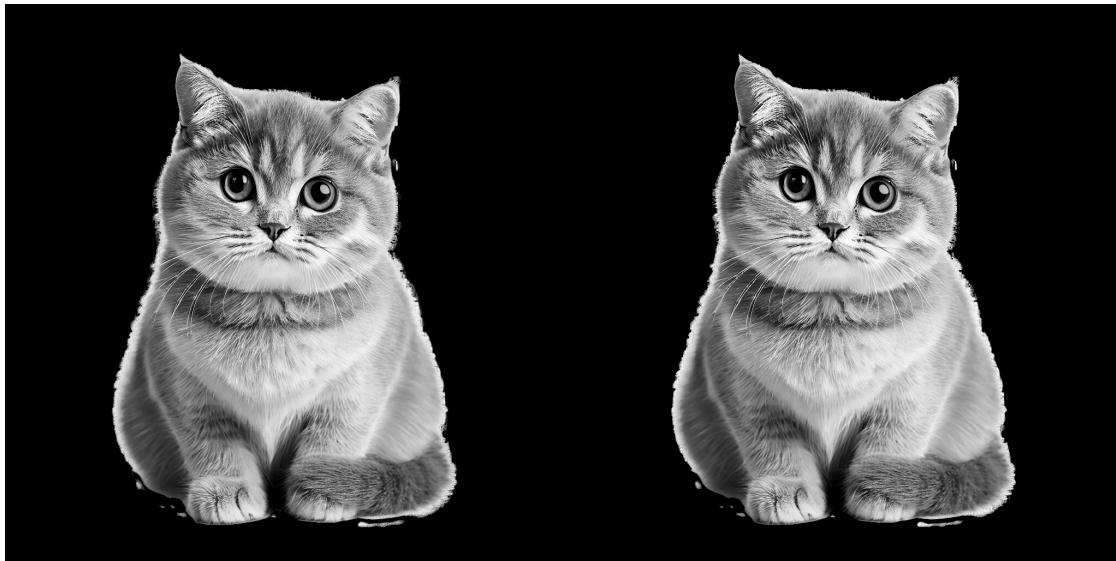
Hasil visual dari penerapan kedua filter pada set gambar uji dapat dilihat pada gambar-gambar di bawah ini.



Gambar 3.1.1 Perbandingan hasil filter pada cameraman.png. (Kiri: Gaussian Blur, Kanan: Median Blur)



Gambar 3.1.2 Perbandingan hasil filter pada coin.png. (Kiri: Gaussian Blur, Kanan: Median Blur)



Gambar 3.1.3 Perbandingan hasil filter pada free\_image.png. (Kiri: Gaussian Blur, Kanan: Median Blur)

### Analisis Hasil

Berdasarkan hasil visual yang disajikan pada Gambar 3.1.1, 3.1.2, dan 3.1.3, dapat diamati bahwa kedua *filter* berhasil menghaluskan gambar. Pada gambar cameraman.png, *Gaussian Blur* menghasilkan efek keburaman yang lebih merata dan natural, namun dengan konsekuensi hilangnya sebagian detail halus pada jaket dan kamera. *Median Blur* pada gambar yang sama juga menghasilkan efek penghalusan, namun secara visual tampak sedikit kurang kabur dan lebih baik dalam mempertahankan beberapa struktur tepi.

Pada gambar coins.png (Gambar 3.2), yang merupakan citra yang relatif bersih dan memiliki kontras tinggi, efek dari kedua *filter* tidak terlalu signifikan. Namun, dapat terlihat bahwa *Gaussian Blur* sedikit menghaluskan permukaan logam dari koin, yang dapat berguna sebagai langkah pra-pemrosesan sebelum deteksi tepi untuk mengurangi deteksi tepi palsu akibat tekstur.

Sebuah pengamatan menarik ditemukan saat menerapkan filter pada gambar *custom* (Gambar 3.1.3). Berbeda dengan gambar lainnya yang menunjukkan perubahan signifikan, efek *Gaussian Blur* dan *Median Blur* pada gambar kucing hampir tidak terlihat dengan *kernel* 5x5. Hal ini dapat diatribusikan pada dua faktor utama: kualitas gambar yang sangat tinggi dengan *noise* minimal, dan tekstur objek (bulu kucing) yang secara inheren sudah halus. Ini

membuktikan bahwa efektivitas dan dampak visual dari sebuah filter sangat bergantung pada konten dan karakteristik awal dari citra input. Untuk menghasilkan efek blur yang terlihat, diperlukan ukuran kernel yang jauh lebih besar.

### ***Edge Detection***

#### **Teori Singkat**

Deteksi tepi (*Edge Detection*) adalah proses mengidentifikasi titik-titik dalam citra digital di mana kecerahan gambar berubah secara tajam. Perubahan tajam ini, atau diskontinuitas, merupakan indikator adanya batas-batas objek, perubahan tekstur, atau garis. Tepi adalah fitur fundamental yang sangat penting karena mengandung informasi struktural utama dari sebuah gambar.

Proyek ini mengimplementasikan dua metode deteksi tepi yang berbeda secara fundamental:

1. **Operator Sobel:** Metode ini bekerja dengan menghitung gradien intensitas gambar pada setiap piksel. Ia menggunakan dua kernel konvolusi 3x3 untuk mendeteksi perubahan intensitas secara horizontal dan vertikal. Hasil dari kedua gradien ini kemudian digabungkan untuk menghasilkan edge map yang merepresentasikan magnitudo perubahan. Kekurangan utama *Sobel* adalah hasilnya cenderung tebal dan rentan terhadap *noise*.
2. **Algoritma Canny:** Dianggap sebagai standar industri untuk deteksi tepi. *Canny* adalah algoritma multi-tahap yang jauh lebih canggih. Tahapannya meliputi: (1) Reduksi noise dengan filter Gaussian, (2) Perhitungan magnitudo dan arah gradien, (3) *Non-maximum suppression* untuk menipiskan tepi menjadi setebal satu piksel, dan (4) *Hysteresis thresholding* dengan dua ambang batas (atas dan bawah) untuk menghilangkan tepi yang lemah dan menghubungkan tepi yang kuat. Hasilnya adalah *edge map* yang bersih, tipis, dan memiliki kontinuitas yang baik.

#### **Implementasi dan Hasil**

Implementasi *edge detection* menggunakan fungsi *cv2.Sobel()* dan *cv2.Canny()* dari library *OpenCV*. Parameter yang digunakan untuk setiap metode dirangkum dalam tabel di

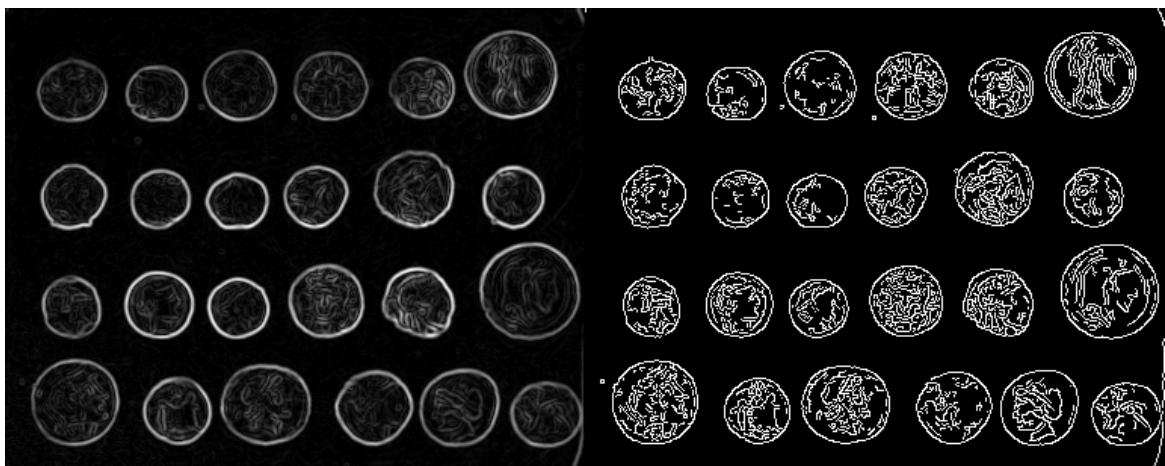
bawah ini. Untuk *Canny*, digunakan nilai threshold bawah 100 dan threshold atas 200, yang merupakan nilai umum yang baik untuk berbagai jenis gambar.

Gambar Sumber	Metode	Parameter	Nilai Parameter	Output File
cameraman	Sobel	Ukuran Kernel	5x5	cameraman_sobel.png
cameraman	Canny	Threshold (Bawah, Atas)	(100, 200)	cameraman_canny.png
coin	Sobel	Ukuran Kernel	5x5	coin_sobel.png
coin	Canny	Threshold (Bawah, Atas)	(100, 200)	coin_canny.png
custom	Sobel	Ukuran Kernel	5x5	free_image_sobel.png
custom	Canny	Threshold (Bawah, Atas)	(100, 200)	free_image_canny.png

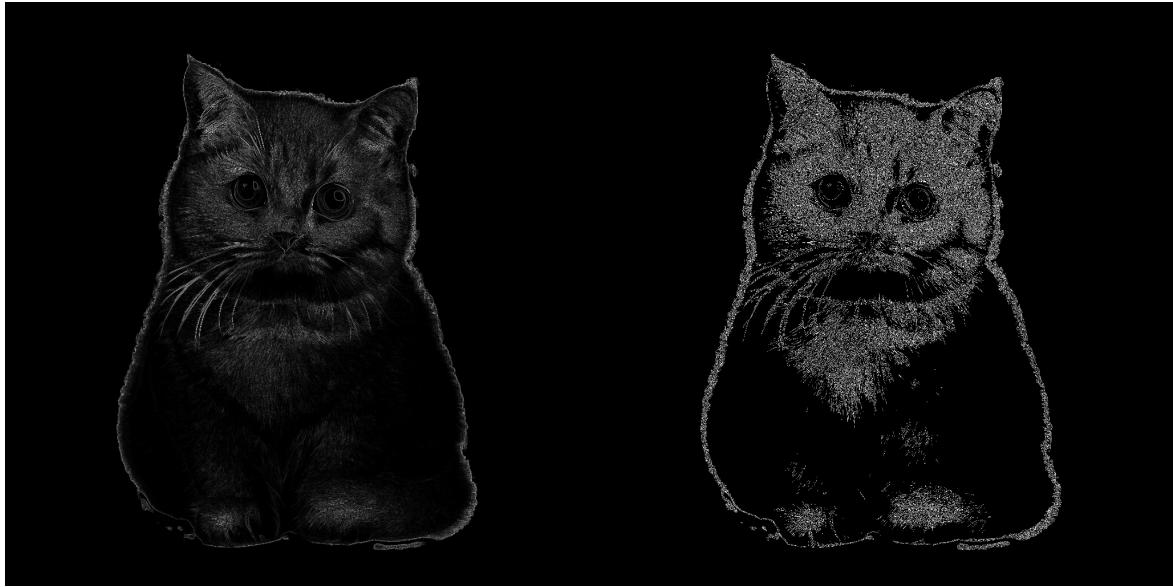
Hasil visual dari kedua metode pada set gambar uji disajikan di bawah ini.



Gambar 3.2.1 Perbandingan deteksi tepi pada cameraman.png. (Kiri: Sobel, Kanan: Canny)



Gambar 3.2.2 Perbandingan deteksi tepi pada coins.png. (Kiri: Sobel, Kanan: Canny)



Gambar 3.2.3 Perbandingan deteksi tepi pada free\_image.png. (Kiri: Sobel, Kanan: Canny)

### Analisis Hasil

Dari perbandingan visual, keunggulan algoritma *Canny* atas Operator *Sobel* sangat jelas terlihat di semua kasus. Pada gambar *coins.png* (Gambar 3.2.2), *Canny* berhasil mengidentifikasi kontur melingkar dari setiap koin dengan garis yang bersih, tipis, dan terhubung dengan baik. Sebaliknya, *Sobel* menghasilkan tepi yang tebal, kabur, dan bahkan mendeteksi "tepi ganda" yang disebabkan oleh bayangan dan highlight pada koin.

Pada gambar *cameraman.png* (Gambar 3.2.1), *Canny* secara presisi menyoroti garis-garis lurus dari tripod kamera dan kontur bangunan di latar belakang. Sementara itu, *Sobel* cenderung lebih sensitif terhadap tekstur, sehingga menghasilkan banyak respons pada area jaket dan rumput, yang membuat tepi objek utama menjadi kurang menonjol.

Namun, perlu dicatat bahwa pada area dengan kepadatan tekstur yang sangat tinggi, seperti pada wajah kucing, algoritma *Canny* dengan *threshold* standar (100, 200) cenderung menghasilkan respons yang berlebihan. Seperti yang terlihat pada Gambar 3.2.3 (kanan), sebagian besar area wajah menjadi putih, menandakan bahwa hampir setiap perubahan tekstur kecil pada bulu terdeteksi sebagai tepi. Fenomena ini terjadi karena tingginya jumlah gradien yang melampaui *low threshold* dan saling terhubung melalui proses *hysteresis*. Ini menunjukkan sebuah *trade-off* penting: sementara *Canny* sangat baik dalam mengekstraksi kontur utama,

pemilihan *threshold* harus disesuaikan dengan tujuan. Jika tujuannya adalah untuk mendapatkan kontur wajah secara keseluruhan, maka diperlukan pra-pemrosesan dengan blur yang lebih kuat atau menaikkan nilai *threshold Canny* untuk menekan respons terhadap detail tekstur yang halus

### **Feature Points**

#### **Teori Singkat**

Deteksi titik fitur (*feature point detection*) adalah proses mengidentifikasi lokasi-lokasi spesifik pada gambar yang unik dan mudah dikenali. Titik-titik ini, yang juga dikenal sebagai keypoints atau interest points, biasanya adalah sudut, *blobs* (gumpalan), atau daerah dengan tekstur yang sangat kaya. Keunikan dari titik-titik ini adalah mereka harus dapat ditemukan kembali secara konsisten bahkan jika gambar mengalami transformasi seperti perubahan skala (*zoom in/out*), rotasi, atau perubahan pencahayaan.

Salah satu algoritma deteksi fitur yang paling kuat dan berpengaruh adalah SIFT (*Scale-Invariant Feature Transform*). SIFT tidak hanya mendeteksi lokasi *keypoints*, tetapi juga menghasilkan sebuah descriptor (vektor numerik) untuk setiap *keypoint*. Deskriptor ini merangkum penampilan visual dari area di sekitar *keypoint*, sehingga sangat robust dan dapat digunakan untuk mencocokkan fitur antara gambar yang berbeda. Keunggulan utama SIFT adalah invariansinya terhadap skala dan rotasi, yang membuatnya sangat berguna untuk aplikasi seperti *image stitching* (pembuatan panorama), pengenalan objek, dan pelacakan 3D.

#### **Implementasi dan Hasil**

Deteksi fitur SIFT diimplementasikan menggunakan kelas *cv2.SIFT\_create()* dari *library opencv-contrib-python*. Setelah *keypoints* terdeteksi, fungsi *cv2.drawKeypoints()* digunakan untuk memvisualisasikannya di atas gambar asli. Lingkaran yang digambar merepresentasikan lokasi dan skala dari fitur, sementara garis di dalamnya menunjukkan orientasi fitur.

Jumlah *keypoints* yang terdeteksi pada setiap gambar dihitung dan disimpan dalam sebuah *file feature\_statistics.csv*. Data ringkasannya disajikan pada tabel di bawah ini.

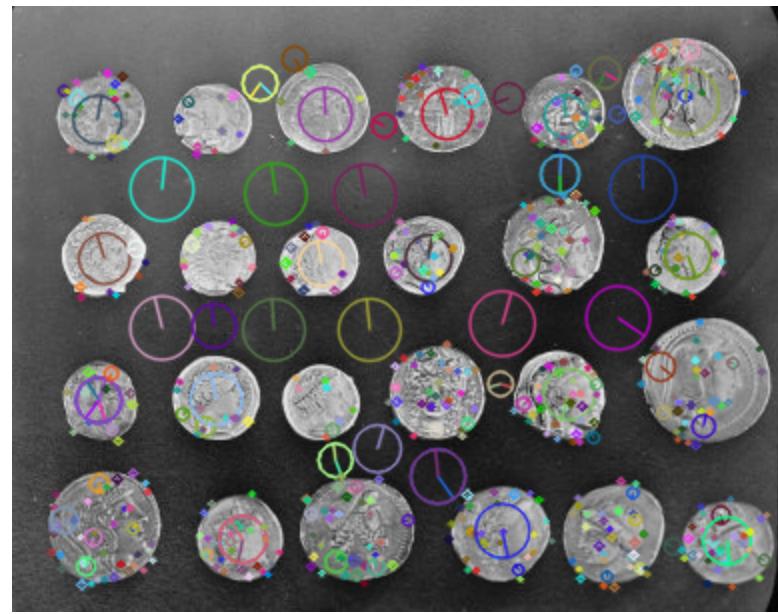
Sumber Gambar	Metode	Jumlah Fitur Terdeteksi	Output File
cameraman	SIFT	791	cameraman_sift.png

coins	SIFT	655	coins_sift.png
custom	SIFT	62593	free_image_sift.png

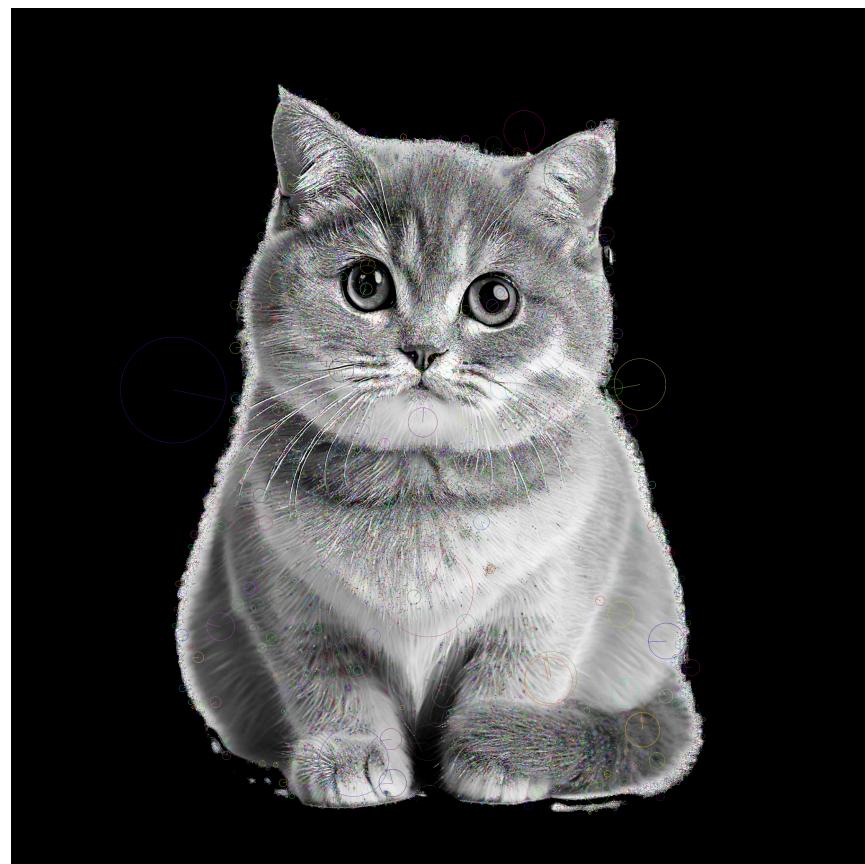
Visualisasi dari keypoints yang terdeteksi pada setiap gambar uji ditunjukkan di bawah ini.



Gambar 3.3.1 Visualisasi fitur SIFT yang terdeteksi pada cameraman.png.



Gambar 3.3.2 Visualisasi fitur SIFT yang terdeteksi pada coins.png.



Gambar 3.3.3 Visualisasi fitur SIFT yang terdeteksi pada free\_image.png.

## **Analisis Hasil**

Hasil deteksi fitur SIFT, yang dirangkum dalam Tabel dan divisualisasikan pada Gambar 3.3.1 hingga 3.3.3 secara jelas menunjukkan korelasi kuat antara kompleksitas tekstur sebuah gambar dengan jumlah keypoints yang terdeteksi. Terdapat perbedaan jumlah fitur yang sangat drastis antar gambar, yang memberikan wawasan mendalam tentang cara kerja algoritma SIFT.

Pada gambar cameraman.png, SIFT berhasil mendeteksi 791 fitur. Seperti yang terlihat pada visualisasi, keypoints ini terkonsentrasi pada area yang kaya akan detail dan kontur, seperti wajah, kamera, tripod, dan bangunan di latar belakang. Sebaliknya, area dengan tekstur homogen seperti langit dan sebagian besar area rumput memiliki sangat sedikit fitur. Ini adalah perilaku klasik SIFT yang secara efektif mengidentifikasi daerah yang informatif secara visual.

Hasil pada gambar coins.png juga menunjukkan perilaku serupa, dengan 655 fitur terdeteksi. Keypoints secara eksklusif ditemukan pada detail ukiran di permukaan koin (wajah, huruf, dan angka). SIFT mengabaikan tepi lingkaran yang mulus dan latar belakang yang polos, karena area tersebut tidak memiliki gradien lokal yang unik. Hal ini membuktikan bahwa SIFT lebih fokus pada "apa yang ada di dalam" objek (tekstur) daripada bentuk geometris luarnya.

Analisis yang paling menonjol datang dari gambar pribadi, free\_image.png. Gambar kucing ini menghasilkan 62.593 fitur. Visualisasinya menunjukkan bahwa hampir seluruh permukaan tubuh kucing dipenuhi oleh keypoints berukuran kecil. Fenomena ini disebabkan oleh dua faktor utama:

1. Resolusi Tinggi dan Tekstur Mikro: Gambar kucing memiliki resolusi tinggi dan tekstur bulu yang sangat kompleks. Setiap helai bulu menciptakan perubahan gradien lokal yang sangat kecil.
2. Sensitivitas SIFT: Algoritma SIFT dirancang untuk mendeteksi variasi lokal ini. Bagi SIFT, setiap gumpalan kecil bulu adalah "fitur" yang unik dan dapat diidentifikasi. Akibatnya, ia mendeteksi puluhan ribu keypoints pada skala mikro.

Sebagai perbandingan, gambar checkerboard (tidak divisualisasikan) hanya menghasilkan 134 fitur menurut data CSV. Ini karena fitur pada papan catur hanya ada di sudut-sudutnya yang sangat berbeda, sementara area hitam dan putih yang luas sama sekali tidak memiliki fitur.

Hasil ini secara mengilustrasikan bahwa jumlah fitur SIFT adalah proksi dari kompleksitas tekstur sebuah citra. Gambar dengan area detail yang jelas seperti cameraman dan coins menghasilkan jumlah fitur yang moderat. Sebaliknya, gambar dengan tekstur berfrekuensi sangat tinggi seperti bulu kucing dapat menyebabkan ledakan deteksi fitur. Hal ini menunjukkan bahwa untuk beberapa aplikasi, mungkin diperlukan langkah tambahan untuk menyaring atau memilih hanya keypoints yang paling kuat agar tidak kewalahan oleh jumlah fitur yang sangat besar.

## Geometry

### Teori Singkat

Analisis geometri dalam Computer Vision berfokus pada pemahaman hubungan spasial dan sifat-sifat geometris dari objek dalam sebuah gambar. Salah satu aplikasi fundamentalnya adalah kalibrasi kamera, yaitu proses untuk mengestimasi parameter-parameter kamera (seperti panjang fokus, pusat optik, dan distorsi lensa) yang memodelkan bagaimana sebuah titik 3D di dunia nyata diproyeksikan ke sebuah titik 2D pada gambar.

Langkah pertama yang esensial dalam banyak prosedur kalibrasi adalah deteksi pola kalibrasi yang diketahui, di mana pola yang paling umum digunakan adalah papan catur (chessboard). Dengan mendeteksi lokasi presisi dari sudut-sudut internal pada papan catur, kita dapat membangun korespondensi antara titik-titik 3D yang diketahui di dunia nyata (sudut-sudut pada papan catur fisik) dan titik-titik 2D yang teramat pada gambar. Informasi korespondensi inilah yang menjadi dasar untuk menghitung parameter kamera.

### Implementasi dan Hasil

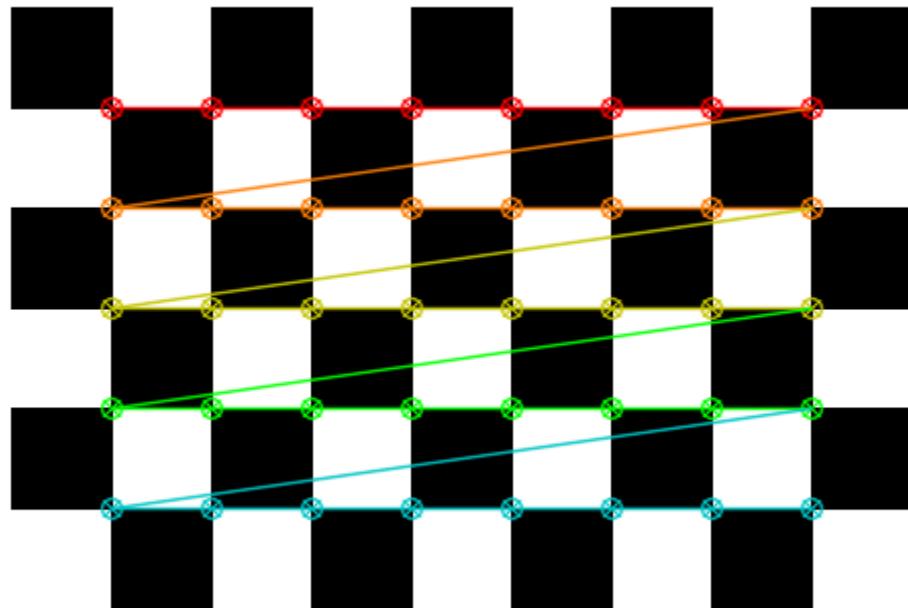
Untuk modul ini, sebuah gambar papan catur sintetis di-generate secara internal menggunakan fungsi OpenCV untuk memastikan kompatibilitas dan deteksi yang robust. Fungsi cv2.findChessboardCorners() kemudian digunakan untuk mendeteksi lokasi semua sudut internal pada pola papan catur yang telah ditentukan.

Implementasi ini berhasil mendeteksi pola papan catur dengan ukuran 8x5 sudut internal. Lokasi presisi dari setiap sudut (total 40 sudut) kemudian divisualisasikan dan disimpan sebagai gambar output. Selain itu, koordinat (x, y) dari setiap sudut diekstraksi dan disimpan ke dalam

sebuah file teks, yang berfungsi sebagai "matriks parameter" atau data input untuk langkah kalibrasi lebih lanjut.

Gambar Sumber	Metode	Parameter	Nilai Parameter	Output File
Papan Catur Sintetis	Deteksi Sudut Papan Catur	pattern_size	(8,5)	checkerboard_corners.png
Papan Catur Sintetis	Ekstraksi Koordinat	-	-	checkerboard_matrix.txt

Hasil visualisasi dan contoh data koordinat yang dihasilkan ditunjukkan di bawah ini.



Gambar 3.4.1 Visualisasi sudut yang berhasil terdeteksi pada papan catur sintetis.

```

# Koordinat Sudut (x, y) untuk pola (8, 5)
100.0000,100.0000
150.0000,100.0000
200.0000,100.0000
250.0000,100.0000
300.0000,100.0000
350.0000,100.0000
400.0000,100.0000
450.0000,100.0000
100.0000,150.0000
150.0000,150.0000
200.0000,150.0000
250.0000,150.0000
300.0000,150.0000
350.0000,150.0000
400.0000,150.0000
450.0000,150.0000
100.0000,200.0000
150.0000,200.0000
200.0000,200.0000
250.0000,200.0000
300.0000,200.0000
350.0000,200.0000
400.0000,200.0000
450.0000,200.0000
100.0000,250.0000
150.0000,250.0000
200.0000,250.0000
250.0000,250.0000
300.0000,250.0000
350.0000,250.0000
400.0000,250.0000
450.0000,250.0000
100.0000,300.0000
150.0000,300.0000
200.0000,300.0000
250.0000,300.0000
300.0000,300.0000
350.0000,300.0000
400.0000,300.0000
450.0000,300.0000

```

Gambar 3.4.2 Contoh data koordinat (x, y) yang disimpan dalam checkerboard\_matrix.txt.

## Analisis Hasil

Seperti yang ditunjukkan pada Gambar 3.4.1, fungsi *cv2.findChessboardCorners()* berhasil mengidentifikasi semua 40 (8x5) sudut internal pada gambar papan catur yang di-generate. Visualisasi dengan garis-garis yang terhubung menunjukkan bahwa semua sudut terdeteksi dalam urutan yang benar dan dengan presisi tinggi. Keberhasilan deteksi ini sangat bergantung pada kualitas gambar input; penggunaan gambar sintetis dengan kontras yang jelas dan border putih yang cukup di sekelilingnya memastikan bahwa algoritma dapat bekerja secara optimal.

Output sekunder, *checkerboard\_matrix.txt*, berisi daftar koordinat piksel sub-piksel yang presisi untuk setiap sudut. Data ini sangat berharga karena ia membentuk setengah dari pasangan data yang dibutuhkan untuk kalibrasi: kita memiliki titik-titik gambar 2D. Setengah lainnya adalah titik-titik objek 3D, yang dalam kasus ini dapat kita definisikan secara manual dalam sebuah sistem koordinat lokal (misal, sudut pertama di (0,0,0), sudut kedua di (1,0,0), dst.). Dengan kedua set data ini, fungsi seperti *cv2.calibrateCamera()* dapat dipanggil untuk menyelesaikan proses kalibrasi.

Dengan demikian, modul ini berhasil mendemonstrasikan langkah pertama dan paling krusial dalam pipeline kalibrasi kamera, yaitu ekstraksi fitur geometris dari pola kalibrasi yang diketahui.

## **Komparasi dan Refleksi Pribadi**

Bab ini bertujuan untuk memberikan analisis komparatif antara hasil yang diperoleh dari gambar standar dan gambar pribadi. Selain itu, bab ini juga berisi refleksi mengenai proses penggerjaan proyek, termasuk tantangan yang dihadapi, pilihan desain yang diambil, dan pelajaran yang dapat dipetik.

### **Komparasi Gambar Standar vs. Gambar Pribadi**

Penggunaan gambar pribadi (free\_image.png, sebuah foto kucing) di samping gambar standar (cameraman, coins) memberikan wawasan berharga tentang bagaimana performa algoritma Computer Vision sangat bergantung pada karakteristik citra input. Beberapa perbandingan yang paling menonjol adalah sebagai berikut:

1. Pada Modul Filtering, efek filter pada gambar kucing hampir tidak terlihat dengan kernel 5x5. Hal ini disebabkan oleh kualitas gambar yang tinggi dengan noise minimal dan tekstur bulu yang secara alami sudah halus. Ini sangat kontras dengan gambar cameraman yang lebih tua dan memiliki grain lebih jelas, di mana efek penghalusan dari filter lebih mudah teramat. Ini menunjukkan bahwa pra-pemrosesan seperti filtering tidak selalu memberikan dampak visual yang dramatis pada citra berkualitas tinggi.
2. Pada Modul Edge Detection, gambar kucing menjadi studi kasus yang luar biasa. Operator Sobel menghasilkan peta gradien yang padat dan lebih merepresentasikan tekstur bulu daripada kontur. Sebaliknya, algoritma Canny menunjukkan keunggulannya dengan berhasil mengisolasi siluet utama kucing dari detail tekstur yang rumit. Namun, pada area wajah yang sangat padat fitur, Canny menunjukkan respons berlebih, mengklasifikasikan hampir semua detail sebagai tepi. Fenomena ini tidak terjadi pada gambar coins yang memiliki bentuk geometris sederhana dan kontras tinggi.
3. Pada Modul Feature Points, perbedaan hasilnya sangat drastis. Gambar kucing menghasilkan 62,593 fitur SIFT, jumlah yang jauh melampaui gambar cameraman (791 fitur) dan coins (655 fitur). "Ledakan fitur" ini disebabkan oleh kombinasi resolusi tinggi

dan tekstur mikro pada bulu kucing, di mana setiap gumpalan bulu kecil dianggap sebagai fitur unik oleh SIFT. Hal ini mengilustrasikan bahwa SIFT adalah detektor tekstur yang sangat kuat, dan untuk gambar dengan kompleksitas tekstural yang ekstrim, mungkin diperlukan langkah penyaringan fitur untuk aplikasi praktis.

Secara keseluruhan, komparasi ini menegaskan bahwa tidak ada algoritma "satu untuk semua". Sebuah algoritma yang bekerja sempurna pada gambar sintetis atau industrial (seperti coins) mungkin menunjukkan perilaku yang sangat berbeda pada gambar naturalistik yang kompleks.

### **Refleksi Proses Pengerjaan**

Proses pengerjaan tugas ini memberikan pengalaman belajar yang komprehensif, tidak hanya dalam implementasi kode tetapi juga dalam analisis dan pemecahan masalah.

Salah satu tantangan teknis utama yang dihadapi adalah pada Modul 4: Geometry. Awalnya, upaya untuk mendeteksi sudut pada gambar checkerboard yang disediakan oleh library skimage secara konsisten gagal. Setelah melakukan investigasi dan debugging, ditemukan bahwa algoritma cv2.findChessboardCorners sangat sensitif terhadap kualitas gambar, terutama keberadaan border (pinggiran) putih yang cukup di sekitar pola. Gambar dari skimage tidak memenuhi kriteria ini. Solusi yang diambil adalah dengan membuat fungsi untuk men-generate gambar papan catur sintetis sendiri yang dijamin kompatibel dengan OpenCV. Tantangan ini memberikan pelajaran penting tentang pentingnya memahami prasyarat dan batasan dari fungsi-fungsi library yang digunakan.

Dalam pemilihan metode, SIFT dipilih untuk deteksi fitur karena reputasinya sebagai algoritma yang sangat robust terhadap perubahan skala dan rotasi. Meskipun ada alternatif yang lebih cepat seperti ORB atau FAST, SIFT dipilih untuk tujuan edukatif guna memahami salah satu algoritma paling fundamental dalam deteksi fitur.

Pelajaran utama yang dipetik dari proyek ini adalah bahwa Computer Vision adalah bidang yang sangat iteratif dan eksperimental. Hasil yang diperoleh sangat dipengaruhi oleh tiga faktor utama: karakteristik gambar input, pilihan algoritma, dan penyetelan parameter. Memahami interaksi antara ketiga faktor ini adalah kunci untuk membangun sistem Computer Vision yang efektif dan andal.

## **Lampiran**

Tautan menuju Repository GitHub:

[https://github.com/fabianradenta/Fabian-Radenta-Bangun\\_13522105\\_IF5152\\_TugasIndividuCV](https://github.com/fabianradenta/Fabian-Radenta-Bangun_13522105_IF5152_TugasIndividuCV)