Tugas Individu Computer Vision

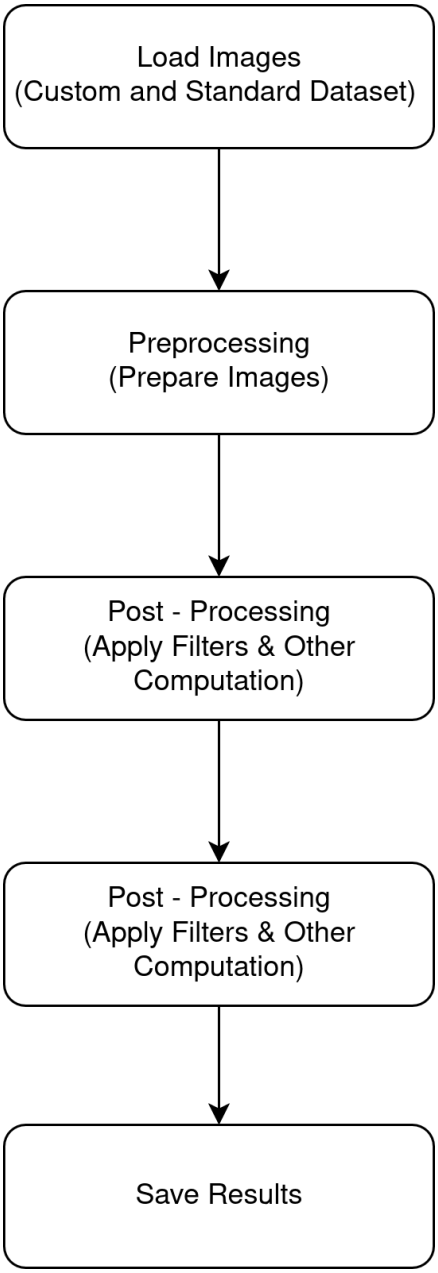
IF5152

Oleh
Christopher Richard Chandra
NIM: 18222057

PROGRAM TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

WORKFLOW PIPELINE



Gambar 1.1 Diagram Pipeline

Pipeline aplikasi pemrosesan citra ini terdiri dari empat tahap utama yang menggambarkan alur kerja sistem secara keseluruhan mulai dari pengambilan data hingga penyimpanan hasil akhir.

1. Input / Akuisisi Data

Tahap pertama merupakan proses pengambilan dan memuat data citra yang akan digunakan dalam sistem.

Aplikasi memanfaatkan dua sumber utama:

- Dataset standar dari Scikit-Image, seperti *cameraman* dan *coins*
- Dataset custom, yaitu citra tambahan seperti *rubik.jpg* dan *dice.jpg* yang diambil dari folder lokal pengguna.

Tujuan utama tahap ini adalah menyiapkan citra mentah agar dapat digunakan dalam tahap pemrosesan berikutnya. Setiap citra kemudian diubah menjadi format yang konsisten (grayscale, tipe data `uint8` atau `float32`) agar kompatibel dengan pustaka seperti NumPy dan OpenCV.

2. Processing / Pemrosesan Citra

Setelah citra diperoleh, sistem melakukan tahap pra proses untuk menyesuaikan citra dengan kebutuhan operasi berikutnya.

Beberapa langkah umum pada tahap ini meliputi:

- Konversi format data antara Scikit-Image, OpenCV, dan NumPy agar dapat saling digunakan.
- Normalisasi nilai piksel untuk memastikan citra memiliki rentang intensitas yang sesuai (misalnya [0, 255]).
- Penyiapan parameter seperti ukuran kernel, sigma, atau ukuran jendela untuk filter dan detektor fitur.

Tahap ini berfungsi sebagai fondasi penting untuk memastikan bahwa hasil pada tahap *post-processing* dapat dihasilkan secara akurat dan stabil.

3. Post Processing

Tahap ini merupakan inti dari pipeline, di mana berbagai operasi analisis citra dan ekstraksi fitur dilakukan.

Beberapa proses utama yang termasuk di dalamnya adalah:

- Filtering, seperti *Gaussian filter* untuk menghaluskan citra dan mengurangi noise.

- Edge detection, misalnya menggunakan *Sobel filter* untuk menonjolkan batas-batas objek pada citra.

- Feature detection, menggunakan algoritma seperti *Harris*, *SIFT*, atau *FAST* untuk mendeteksi titik-titik penting pada citra.

- Kalibrasi kamera, yang melibatkan proses penghitungan parameter intrinsik dan ekstrinsik berdasarkan pola *checkerboard*.

Tahap ini menghasilkan citra yang telah mengalami analisis mendalam dan siap digunakan untuk interpretasi atau langkah lanjutan.

4. Save Results / Penyimpanan Hasil

Pada tahap terakhir, seluruh hasil pemrosesan dan analisis disimpan dalam struktur folder yang terorganisasi.

Setiap jenis operasi (misalnya Gaussian filter, Sobel filter, atau kalibrasi) memiliki direktori tersendiri, seperti:

- numpy_results/gaussian_filter/
- numpy_results/sobel_filter/
- geometry_results/

Hasil yang disimpan mencakup citra keluaran, visualisasi kernel, hingga file CSV yang berisi parameter hasil perhitungan (misalnya matriks kamera dan koefisien distorsi). Tujuannya adalah agar hasil dapat ditinjau dan diverifikasi.

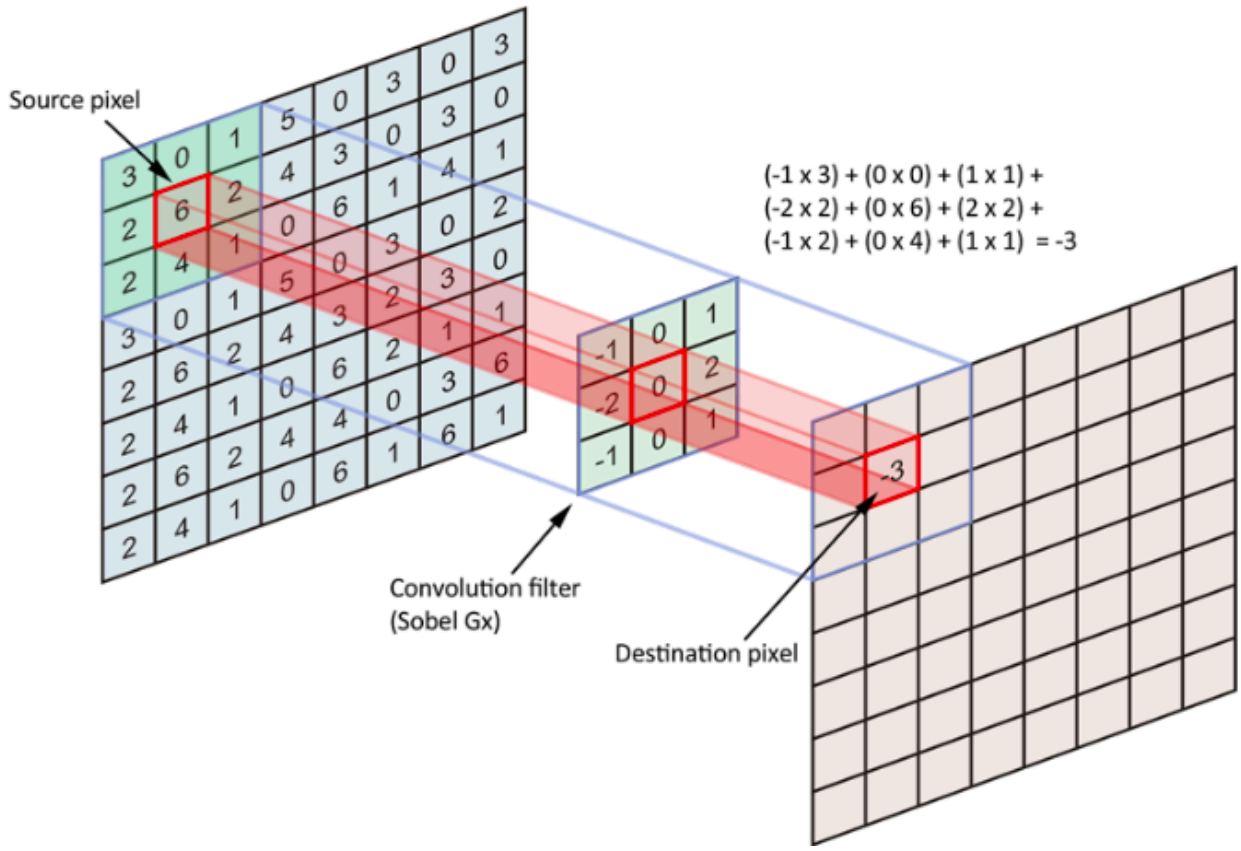
PROSES & HASIL SETIAP FITUR

Part 1 : Filtering

A. Gaussian

1. Teori Singkat

Pada proses *filtering*, kita akan melakukan konvolusi menggunakan sebuah *kernel* dari *filter* pilihan dengan *image* target. Proses konvolusi dilakukan dengan fokus pada bagian tengah dari kernel, misalnya pada kernel 3 x 3, kita fokus untuk menghitung nilai baru untuk satu piksel saja dengan mempertimbangkan sekitarnya.



Gambar 2.1 Ilustrasi Konvolusi dengan Kernel

Pada *filter* ini akan menggunakan fungsi *gaussian* sebagai berikut,

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Fungsi tersebut merupakan fungsi dari Gaussian Kernel. Hasil dari perhitungan tersebut menghasilkan kernel yang kemudian akan dikonvolusikan dengan *image* awal yang mau di-*filter*.

Sigma (σ) pada persamaan tersebut adalah standar deviasi yang menunjukkan semakin besar standar deviasi yang digunakan maka akan mempengaruhi distribusi *weight* ke masing - masing tetangga piksel target. Sedangkan nilai X dan Y pada persamaan tersebut merupakan *mesh grid* dari gaussian 1D.

Hasil dari tersebut berupa *filter* dengan dimensi *n* x *n* yang kemudian digunakan untuk proses konvolusi ke gambar tujuan.

2. Parameter

Pada *script* yang dibuat, terdapat 2 metode pembuatan *filter gaussian*, penerapan manual dengan Numpy dan menggunakan Scikit Image secara langsung.

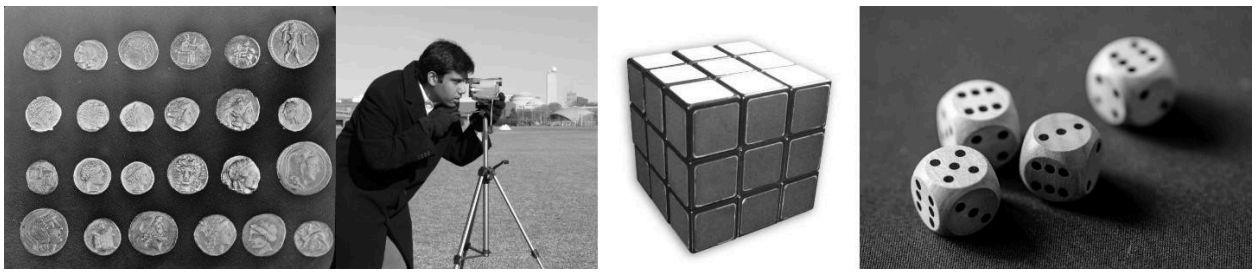
Parameter Scikit Image :

1. Sigma - Standar deviasi yang digunakan untuk pembuatan kernel
   -> Menggunakan standar deviasi sebesar 2 sehingga tidak terlalu besar dan tidak terlalu kecil
2. Image - Gambar target
3. Mode - Mengatur bagaimana border akan diproses ketika melakukan konvolusi

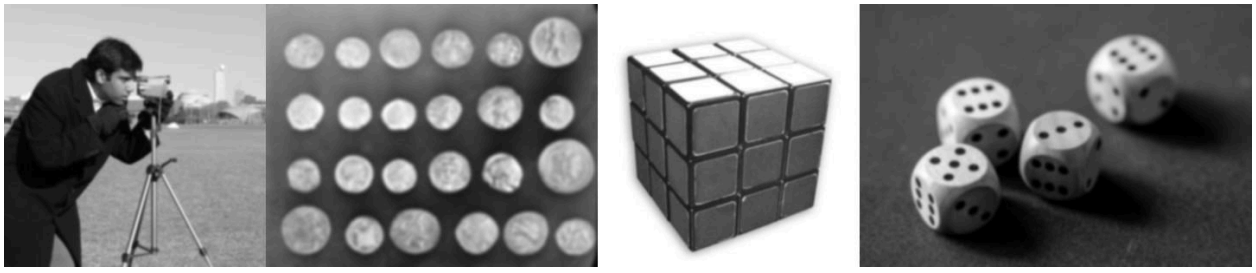
Parameter Numpy Scratch:

1. Size - Ukuran n x n dari kernel yang mau digunakan
2. Sigma - Standar deviasi untuk pembuatan kernel

3. Hasil & Analisis

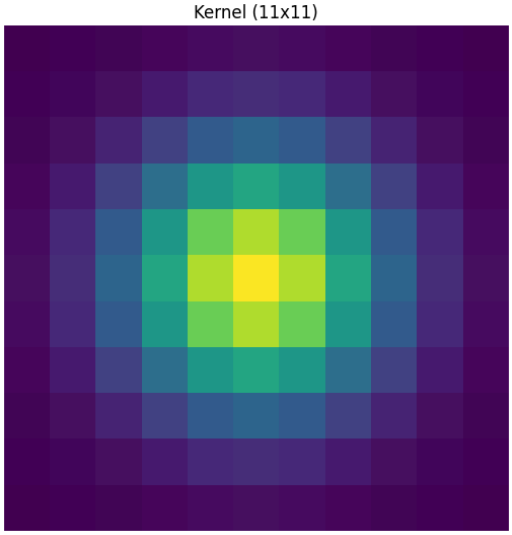


Gambar 2.2 Gambar Dataset (Belum ada perubahan)



Gambar 2.3 Hasil Gaussian Filter

Setelah gambar asli diberikan *gaussian filter* yang terjadi adalah gambar menjadi lebih *smooth*. Hal ini bisa kita lihat pada hasil gambar tersebut gambarnya menjadi lebih blur karena proses *smoothing* yang dilakukan oleh *filter*. Kemudian, untuk mengetahui seperti apa filter yang digunakan dapat melihat gambar 2.4.



Gambar 2.4 Kernel Gaussian yang Digunakan

B. Sobel

1. Teori Singkat

Pada proses ini kita akan mengganti kernel yang awalnya merupakan *gaussian* menjadi *sobel*. Untuk membuat Kernel Sobel jauh lebih rumit apabila dibandingkan dengan Gaussian. Hal ini disebabkan karena kernel sobel dihasilkan seutuhnya dari persamaan polinomial sebagai berikut,

$$Sobel\_x(pic) := \sum_{x\_win=0}^{winsize-1} \sum_{y\_win=0}^{winsize-1} smooth_{y\_win} \cdot diff_{x\_win} \cdot pic_{y\_win,x\_win}$$

(a) *Mx*

$$Sobel\_y(pic) := \sum_{x\_win=0}^{winsize-1} \sum_{y\_win=0}^{winsize-1} smooth_{x\_win} \cdot diff_{y\_win} \cdot pic_{y\_win,x\_win}$$

(b) *My*

Kemudian, beberapa bagian dari persamaan tersebut dituliskan sebagai berikut,

$$smooth_{x\_win} := \frac{(winsize-1)!}{(winsize-1-x\_win)! \cdot x\_win!}$$

$$Pascal(k,n) := \begin{cases} \frac{n!}{(n-k)! \cdot k!} & if(k \geq 0) \cdot (k \leq n) \\ 0 & otherwise \end{cases}$$

$$diff_{x\_win} := Pascal(x\_win, winsize-2) - Pascal(x\_win-1, winsize-2)$$

Persamaan ini bertujuan untuk menghasilkan sebuah fungsi kita bisa menggunakan berbagai ukuran kernel sobel, misal 3x3, 5x5, atau bahkan 11x11. Namun, 3x3 sangat umum digunakan. Setelah mendapatkan kernel untuk X dan Y, dilakukan konvolusi dengan gambar target dan menghasilkan konvolusi horizontal dan vertikal. Hasil konvolusi tersebut kemudian dihitung dengan pitagoras untuk menghasilkan *magnitude*. Hasil berupa *magnitude* tersebut akan menghasilkan gambar yang sudah terfilter dengan sobel.

2. Parameter

Pada *script* yang dibuat, terdapat 2 metode pembuatan *filter* Sobel, penerapan manual dengan Numpy dan menggunakan Scikit Image secara langsung.

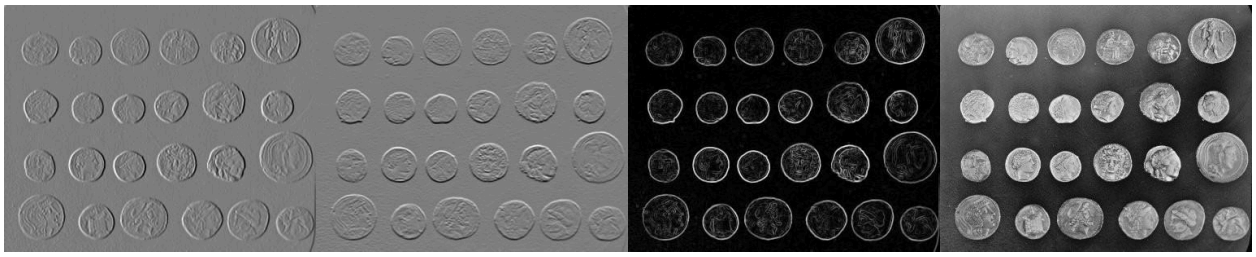
Parameter Scikit Image :

1. Image - Gambar target

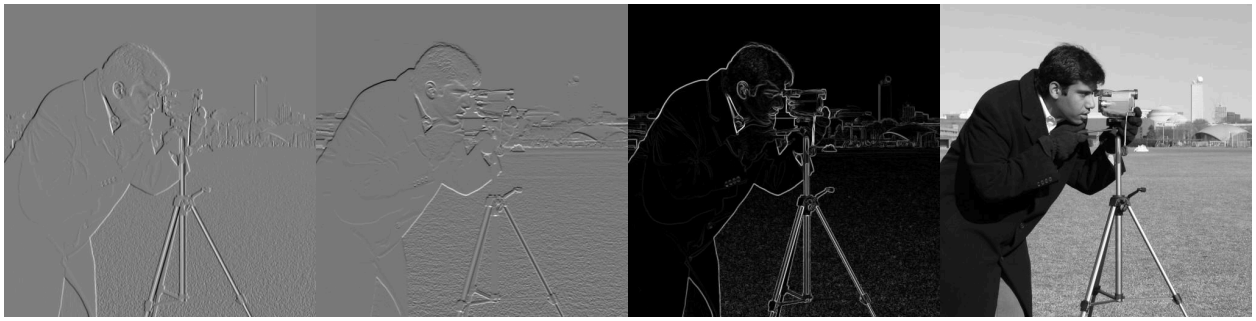
Parameter Numpy Scratch:

1. Sobel Size - Ukuran n x n dari kernel yang mau digunakan

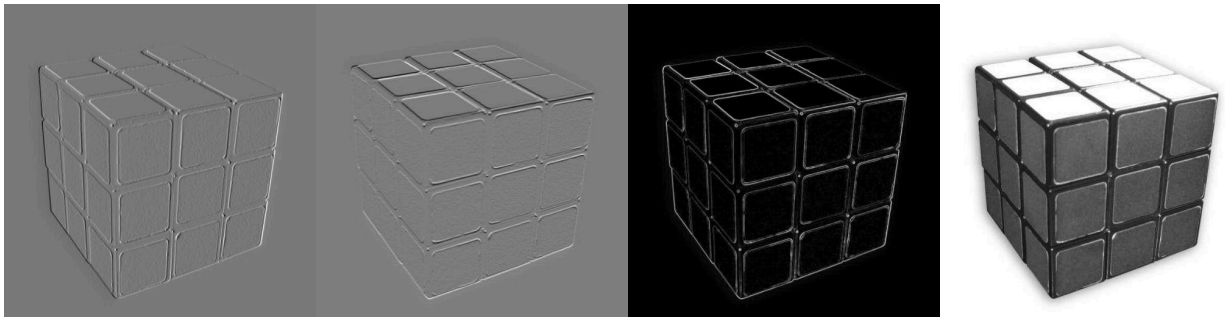
3. Hasil & Analisis



Gambar 2.2.1 Coin dengan Sobel (Horizontal, Vertical, Magnitude, Original)



Gambar 2.2.2 Cameraman dengan Sobel (Horizontal, Vertical, Magnitude, Original)



Gambar 2.2.3 Rubik dengan Sobel (Horizontal, Vertical, Magnitude, Original)

Pada percobaan ini digunakan operator Sobel untuk mendeteksi tepi pada tiga citra uji, yaitu *Cameraman*, *Rubik*, dan *Coin*. Operator Sobel bekerja dengan menghitung gradien intensitas piksel dalam dua arah horizontal (sumbu X) dan vertikal (sumbu Y). Kemudian menggabungkannya menjadi magnitudo gradien untuk menonjolkan tepi secara keseluruhan.

Hasil keluaran berupa tiga jenis citra untuk setiap gambar:

1. Sobel Horizontal, yang menonjolkan tepi-tepi vertikal dalam citra (perubahan intensitas sepanjang sumbu X).
2. Sobel Vertikal, yang menonjolkan tepi-tepi horizontal (perubahan intensitas sepanjang sumbu Y).
3. Sobel Magnitude, hasil penggabungan keduanya untuk menunjukkan kekuatan tepi secara keseluruhan.

Dari hasil pengujian terlihat bahwa:

- Pada citra *Cameraman*, operator Sobel berhasil menyoroti garis tepi dengan cukup baik
- Pada citra *Rubik*, tepian antar warna dan batas sisi kubus terlihat sangat tegas, terutama pada citra magnitude. Hal ini menunjukkan bahwa citra dengan pola tepi kuat sangat cocok untuk deteksi menggunakan Sobel.
- Pada citra *Coin*, bentuk koin dapat terlihat dengan jelas, yaitu masing - masing berbentuk lingkaran. Tetapi, pada bagian timbul di koin untuk gambar ataupun tulisan tidak begitu terdeteksi dengan baik.

Part 2: Edge

A. Sobel

1. Teori Singkat

Metode Sobel merupakan teknik deteksi tepi berbasis operator gradien yang digunakan untuk menonjolkan perubahan intensitas pada citra. Operator ini menggunakan dua kernel konvolusi, masing-masing untuk mendeteksi tepi pada arah horizontal (Gx) dan vertikal (Gy). Nilai gradien total diperoleh dari kombinasi kedua arah tersebut dengan rumus pitagoras.

Hasil dari metode Sobel menghasilkan peta tepi yang menampilkan batas-batas objek secara jelas, terutama pada area dengan kontras tinggi. Namun, karena metode ini sensitif terhadap noise, biasanya dilakukan filtering Gaussian terlebih dahulu sebelum penerapan operator Sobel.

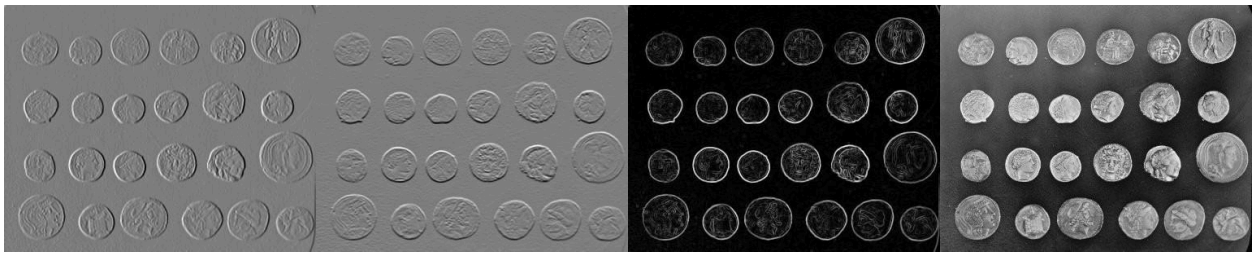
2. Parameter

Pada *script* yang dibuat, pembuatan *filter* Sobel menggunakan Scikit Image secara langsung.

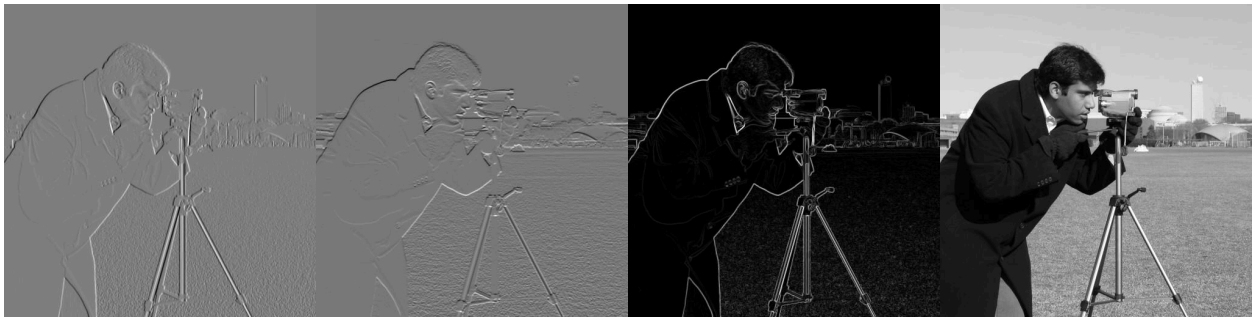
Parameter Scikit Image :

1. Image - Gambar target

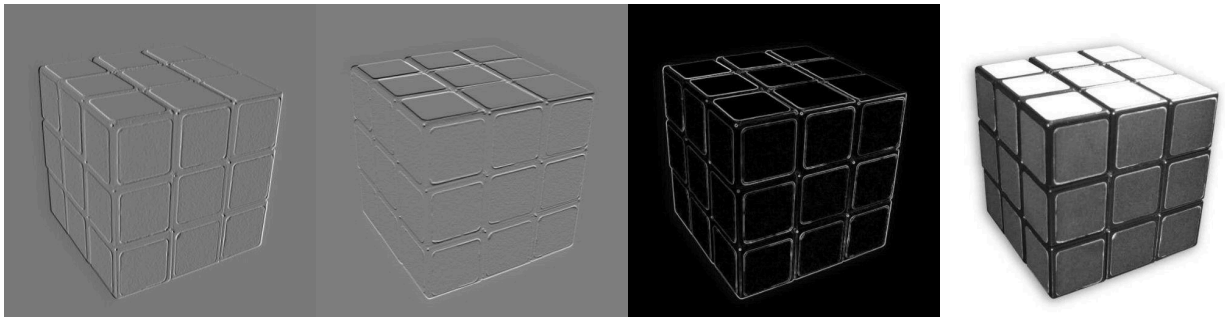
3. Hasil & Analisis



Gambar 2.2.1 Coin dengan Sobel (Horizontal, Vertical, Magnitude, Original)



Gambar 2.2.2 Cameraman dengan Sobel (Horizontal, Vertical, Magnitude, Original)



Gambar 2.2.3 Rubik dengan Sobel (Horizontal, Vertical, Magnitude, Original)

Pada sobel, tidak terdapat parameter yang bisa diubah - ubah apabila menggunakan scikit images. Sehingga, hasil yang ditampilkan sama persis dengan pada proses *filtering* karena tidak terdapat threshold ataupun standar deviasi yang bisa diubah - ubah.

B. Canny

1. Teori Singkat

Metode Canny Edge Detection merupakan algoritma deteksi tepi yang dikembangkan oleh John F. Canny pada tahun 1986. Algoritma ini dikenal sebagai salah satu metode deteksi tepi paling efektif karena mampu menghasilkan tepi yang halus, tipis, dan minim noise.

Proses deteksi tepi dengan metode Canny terdiri dari beberapa tahap utama, yaitu:

1. Reduksi Noise
   Gambar terlebih dahulu di filter menggunakan Gaussian filter untuk menghilangkan noise dan variasi kecil pada intensitas piksel yang dapat menimbulkan tepi palsu.

2. Perhitungan Gradien
   Nilai gradien intensitas dihitung untuk mendeteksi perubahan intensitas yang tajam, yang merupakan indikasi adanya tepi.

3. Non-Maximum Suppression
   Proses ini menipiskan hasil deteksi tepi dengan mempertahankan hanya piksel yang merupakan nilai maksimum lokal pada arah gradiennya.

4. Double Thresholding
   Dua nilai ambang digunakan untuk membedakan antara tepi kuat (high threshold) dan tepi lemah (low threshold).

5. Edge Tracking by Hysteresis
   Tepi lemah yang terhubung dengan tepi kuat akan dipertahankan, sedangkan tepi lemah yang terisolasi akan dihapus.

Parameter penting dalam metode ini antara lain:

- σ (sigma) → standar deviasi dari Gaussian filter, yang mengatur tingkat penghalusan gambar.

- low threshold dan high threshold → nilai ambang batas bawah dan atas yang menentukan sensitivitas deteksi tepi.

Metode Canny banyak digunakan karena mampu menghasilkan deteksi tepi yang tajam, tipis, dan stabil terhadap noise serta perubahan intensitas cahaya.

2. Parameter

Pada bagian ini digunakan *feature* Canny dari Scikit Image dengan parameter sebagai berikut, 1

1. img_gray
   Gambar input dalam format grayscale yang akan diproses.
   → Canny hanya bekerja pada citra hitam-putih karena mendeteksi perubahan intensitas piksel.

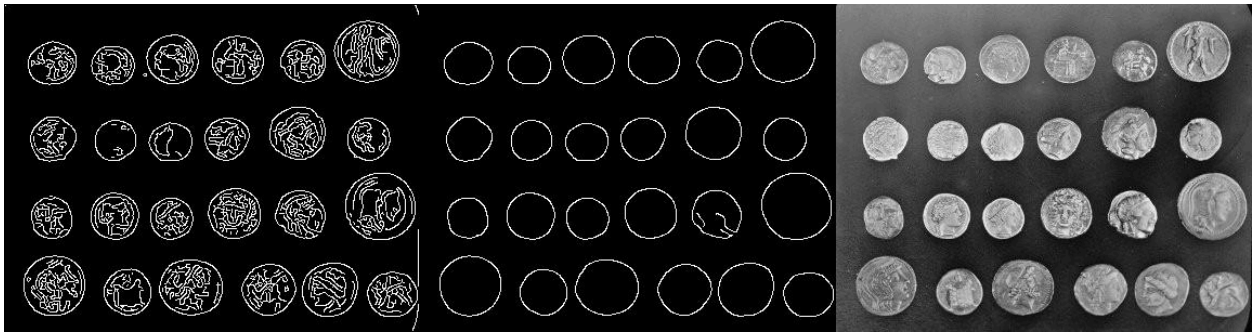
2. sigma
   Nilai standar deviasi untuk Gaussian filter yang digunakan pada tahap awal untuk mereduksi noise.
   → Semakin besar nilai sigma, semakin halus hasil deteksi namun detail kecil bisa hilang.

3. low_threshold
   Nilai ambang bawah untuk menentukan tepi yang lemah.
   → Piksel dengan gradien di bawah nilai ini akan diabaikan (bukan tepi).

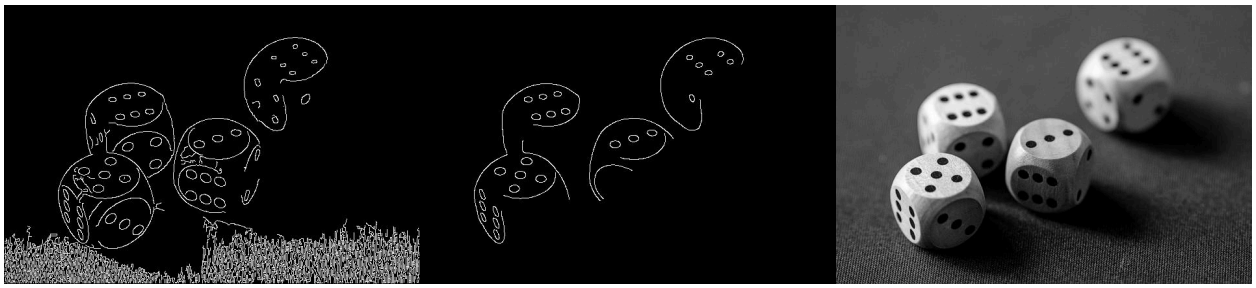
4. high_threshold
   Nilai ambang atas untuk menentukan tepi yang kuat.
   → Piksel dengan gradien di atas nilai ini pasti dianggap sebagai tepi utama (kuat).
   → Piksel di antara low dan high threshold akan dianggap tepi hanya jika terhubung dengan tepi kuat.

3. Hasil & Analisis

Gambar 2.3.1 Canny (σ = 1, Threshold = 0.1 - 0.3), (σ = 2, Threshold = 0.2 - 0.5), Original



Gambar 2.3.2 Coin (σ = 1, Threshold = 0.1 - 0.3), (σ = 2, Threshold = 0.2 - 0.5), Original



Gambar 2.3.2 Dice (σ = 1, Threshold = 0.1 - 0.3), (σ = 2, Threshold = 0.2 - 0.5), Original

Berdasarkan hasil yang diperoleh, apabila dibandingkan dengan Sobel, hasil menggunakan Canny jauh lebih baik karena pada beberapa bagian contohnya seperti di koin terdapat lebih banyak *edge* yang tertangkap pada wajah koin dengan parameter σ = 1, Threshold = 0.1 - 0.3. Namun, ketika σ dinaikkan gambar menjadi lebih *smooth* sehingga banyak *edge* yang terlewat hal ini diperparah dengan *threshold* yang dinaikkan sehingga banyak fitur yang terlewat dan hanya tepi - tepi bentuk pada objek utama saja yang banyak tertangkap.

Part 3: Features Point

A. Harris

1. Teori Singkat

Prinsip utama dari Harris Corner Detection adalah menganalisis perubahan intensitas lokal di sekitar setiap piksel pada gambar. Pada metode ini kita akan menggeser sebuah jendela kecil (window) ke berbagai arah di sekitar piksel tersebut dan mengamati bagaimana intensitas berubah:

- Jika intensitas tidak banyak berubah ke segala arah → area tersebut adalah datar (flat region).
- Jika intensitas berubah kuat hanya ke satu arah → area tersebut adalah tepi (edge).

- Jika intensitas berubah kuat ke dua arah berbeda → area tersebut adalah sudut (corner).

Dengan kata lain, corner adalah titik di mana perubahan arah tepi bertemu, seperti ujung kotak, jendela, atau pertemuan dua dinding pada gambar.

Algoritma ini dimulai dengan menghitung perubahan intensitas (first-order derivative) dari gambar baik secara horizontal (Ix) maupun vertikal (Iy) menggunakan operator gradien, seperti Sobel. Nilai-nilai gradien ini kemudian digunakan untuk membentuk sebuah matriks kovarian lokal yang disebut M matrix (atau struktur tensor).

Matriks ini merepresentasikan seberapa besar perubahan intensitas terjadi di sekitar suatu piksel dalam dua arah (x dan y). Dengan menganalisis matriks ini, kita dapat menilai apakah suatu titik merupakan area datar, tepi, atau sudut.

Setelah matriks M terbentuk, langkah selanjutnya adalah menghitung response value (R) menggunakan fungsi respon Harris.

Nilai R inilah yang menentukan jenis fitur:

- Jika R kecil atau negatif → area datar (flat region).
- Jika R negatif besar → tepi (edge).
- Jika R besar positif → sudut (corner).

Tahap akhir dari algoritma adalah melakukan thresholding untuk memilih piksel dengan nilai R yang cukup tinggi dan menerapkan non-maximum suppression agar hanya titik-titik dengan respons paling kuat yang dianggap sebagai sudut

2. Parameter

Pada bagian ini terdapat beberapa paremeter yang bisa diatur untuk Canny sebagai berikut:

1. img_gray (np.ndarray)
Gambar input dalam format grayscale. Gambar ini digunakan untuk mendeteksi sudut (corner) karena Harris berfokus pada intensitas dari sebuah gambar.

2. block_size
Ukuran area lokal (window) yang digunakan untuk menghitung matriks kovarian (M) di sekitar setiap piksel. Nilai yang lebih besar memperhitungkan area yang lebih luas tetapi dapat mengurangi sensitivitas terhadap detail kecil.

3. ksize
Ukuran kernel Sobel yang digunakan untuk menghitung gradien intensitas (Ix dan Iy). Semakin besar nilai ini, semakin halus gradien yang dihasilkan.

4. k

Parameter empiris untuk fungsi respon Harris. Nilai ini mengontrol sensitivitas algoritma terhadap sebuah sudut / *corner*. Nilai lebih kecil membuat deteksi lebih sensitif terhadap sebuah *corner*.

5. thresh
Threshold untuk menentukan titik mana yang dianggap sebagai corner. Nilai ini merepresentasikan bagian dari maksimum nilai respon R.

3. Hasil & Analisis

Gambar 3.1.1 Cameraman with Harris & Before Images

Gambar 3.1.2 Room with Harris & Before Images

Berdasarkan hasil deteksi menggunakan metode Harris Corner Detection, tidak semua titik sudut (*corner*) pada citra dapat terdeteksi dengan baik. Pada citra *cameraman*, deteksi sudut terlihat kurang optimal karena perbedaan intensitas antar piksel tidak terlalu kontras, sehingga algoritma sulit mengenali perubahan arah tepi secara jelas. Sebaliknya, pada gambar ruangan hasil deteksi terlihat sedikit lebih baik karena terdapat perbedaan intensitas yang lebih jelas antara dinding, lantai, dan objek di dalam ruangan.

Meskipun demikian, titik-titik sudut yang terdeteksi masih belum sepenuhnya sesuai dengan posisi sudut sebenarnya dan cenderung tersebar secara acak. Hal ini kemungkinan dipengaruhi oleh faktor seperti tingkat kontras lokal, pencahayaan, serta pengaturan parameter seperti *block size*, *k*, dan *threshold* yang belum optimal. Secara keseluruhan, metode Harris menunjukkan hasil yang lebih efektif pada citra dengan kontras intensitas yang tinggi, namun kurang stabil pada citra dengan perbedaan intensitas yang lemah seperti *cameraman*.

B. SIFT

1. Teori Singkat

Metode SIFT (Scale-Invariant Feature Transform) digunakan untuk mendeteksi dan mendeskripsikan fitur lokal pada citra yang bersifat invarian terhadap skala, rotasi, dan perubahan pencahayaan. Artinya, fitur yang dideteksi tetap dapat dikenali meskipun ukuran objek berubah, citra diputar, atau pencahayaannya berbeda.

Proses SIFT dimulai dengan membangun ruang skala (scale-space) menggunakan serangkaian filter Gaussian untuk mendeteksi titik-titik ekstrem (keypoints) pada berbagai tingkat skala melalui perbedaan Gaussian (Difference of Gaussian – DoG). Titik-titik ini kemudian disaring untuk menghilangkan noise dan titik yang tidak stabil.

Selanjutnya, setiap keypoint diberikan orientasi dominan berdasarkan arah gradien di sekitarnya agar tetap konsisten terhadap rotasi. Setelah itu, dibuat deskriptor fitur berupa vektor yang mewakili pola intensitas lokal di sekitar keypoint, sehingga fitur tersebut dapat dibandingkan antar gambar secara robust.

Dengan kemampuannya yang stabil dan akurat, SIFT sering digunakan untuk pengenalan objek, pencocokan citra, dan rekonstruksi 3D, meskipun secara komputasi tergolong lebih kompleks dibandingkan metode seperti Harris atau FAST.

2. Parameter

1. nfeatures (default = 0)
Jumlah maksimum titik fitur (*keypoints*) yang akan disimpan setelah proses deteksi.

- Jika diatur ke 0 → semua fitur yang ditemukan akan digunakan.
- Semakin besar nilai ini, semakin banyak fitur yang dideteksi namun proses menjadi lebih lambat.

2. nOctaveLayers (default = 3)
Jumlah *layer* per *octave* dalam piramida skala (*scale-space*).

- Tiap *octave* mewakili gambar dengan tingkat blur berbeda.
- Nilai lebih tinggi menghasilkan deteksi fitur yang lebih halus tetapi membutuhkan waktu komputasi lebih lama.

3. contrastThreshold (default = 0.04)
Ambang batas minimum kontras agar suatu titik dianggap sebagai fitur yang valid.

- Nilai kecil → mendeteksi lebih banyak fitur, tapi lebih rentan terhadap noise.
- Nilai besar → hanya mendeteksi fitur dengan kontras tinggi, bisa kehilangan detail halus.

4. edgeThreshold (default = 10)
Batas untuk menekan deteksi di area tepi gambar.

- Nilai kecil → lebih ketat dalam menyaring area tepi, bisa kehilangan fitur yang sah.
- Nilai besar → lebih toleran terhadap tepi, namun berisiko mendeteksi fitur palsu.

5. sigma (default = 1.6)
Nilai standar deviasi (*standard deviation*) dari Gaussian blur yang digunakan pada skala awal.

- Nilai besar → membuat gambar lebih halus dan tahan terhadap noise, tetapi fitur kecil bisa hilang.
- Nilai kecil → lebih sensitif terhadap detail kecil, namun bisa terpengaruh noise.

3. Hasil & Analisis

Gambar 3.2.1 Cameraman with SIFT & Before Images

Gambar 3.2.2 Room with Harris & Before Images

Pada hasil deteksi fitur menggunakan algoritma SIFT (Scale-Invariant Feature Transform), terlihat bahwa terdapat banyak titik fitur yang terdeteksi pada gambar. Setiap titik tersebut biasanya ditandai dengan lingkaran-lingkaran berukuran berbeda. Lingkaran ini menggambarkan skala atau ukuran fitur lokal yang berhasil dideteksi oleh algoritma.
Artinya, lingkaran besar menunjukkan bahwa fitur tersebut terdeteksi pada area dengan ukuran yang lebih besar (misalnya pola atau tekstur yang lebar), sedangkan lingkaran kecil menunjukkan fitur dengan detail halus atau tekstur kecil.

Dari hasil yang diperoleh, terlihat bahwa jumlah titik fitur yang terdeteksi oleh SIFT jauh lebih banyak dibandingkan dengan metode Harris. Hal ini menunjukkan bahwa SIFT lebih sensitif dalam mendeteksi berbagai jenis fitur, baik pada area dengan kontras tinggi maupun rendah. Namun, pada hasil deteksi juga terlihat beberapa titik yang tidak benar-benar berada pada tepi atau sudut objek. Hal ini bisa terjadi karena SIFT tidak hanya mendeteksi sudut (corner), tetapi juga area dengan perubahan intensitas yang signifikan di berbagai skala.

C. FAST

1. Teori Singkat

FAST (Features from Accelerated Segment Test) adalah algoritma deteksi sudut (corner detection) yang dirancang agar sangat cepat dan efisien dalam mendeteksi fitur pada citra. Prinsip dasarnya adalah membandingkan intensitas piksel pusat dengan piksel-piksel di sekitarnya yang terletak pada lingkaran berjumlah 16 piksel (seperti pola pada jam).

Sebuah piksel dianggap sebagai corner jika terdapat sejumlah piksel yang berurutan pada lingkaran tersebut yang memiliki intensitas lebih terang atau lebih gelap secara signifikan dibandingkan piksel pusat. Untuk mempercepat prosesnya, FAST menggunakan pengujian

awal (segment test) dengan hanya memeriksa beberapa piksel kunci terlebih dahulu sebelum melakukan pengecekan penuh — inilah yang membuatnya jauh lebih cepat dibandingkan metode seperti Harris atau SIFT.

FAST sangat cocok untuk aplikasi real-time seperti pelacakan objek dan sistem visi robot karena kecepatan tinggi dan efisiensinya, meskipun tidak sekuat SIFT atau Harris dalam hal ketahanan terhadap rotasi dan perubahan skala.

2. Parameter

1. threshold (int, default = 10)
Menentukan seberapa besar perbedaan intensitas antara piksel pusat dengan piksel di sekitarnya agar dianggap sebagai corner (sudut). Nilai lebih besar menghasilkan lebih sedikit fitur terdeteksi (lebih ketat), sedangkan nilai lebih kecil membuat detektor lebih sensitif terhadap perubahan kecil.

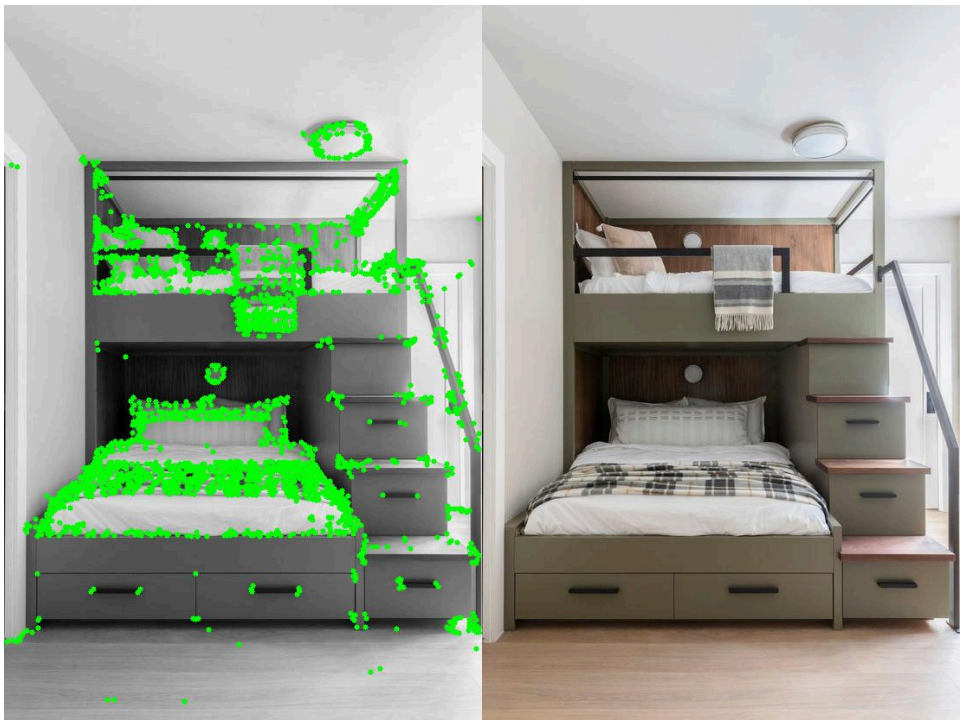
2. nonmaxSuppression (bool, default = True)
Jika diaktifkan (True), hanya mempertahankan corner dengan nilai respons maksimum di area lokalnya. Hal ini membantu menghindari deteksi berlebih dan menghasilkan hasil yang lebih bersih. Jika dimatikan (False), semua titik kandidat corner akan ditampilkan, termasuk yang lemah.

3. type (cv2.FAST_FEATURE_DETECTOR_TYPE_5_8, TYPE_7_12, atau TYPE_9_16)
Menentukan jumlah piksel berurutan di sepanjang lingkaran Bresenham yang digunakan untuk mendeteksi corner.

- TYPE_5_8: Menggunakan 8 piksel.
- TYPE_7_12: Menggunakan 12 piksel.
- TYPE_9_16: Menggunakan 16 piksel (paling umum dan default).

3. Hasil & Analisis

Gambar 3.2.1 Cameraman with FAST & Before Images



Gambar 3.2.1 Room with SIFT & Before Images

Pada hasil deteksi fitur menggunakan metode FAST (Features from Accelerated Segment Test), terlihat bahwa algoritma ini menghasilkan banyak titik fitur (keypoints) yang tersebar di seluruh area gambar.

Metode ini memang dikenal sangat sensitif terhadap perubahan intensitas lokal sehingga area seperti rumput pada gambar cameraman menghasilkan banyak titik-titik kecil karena variasi tekstur yang tajam.

Meskipun demikian, pada bagian objek utama seperti pria yang memegang kamera, titik-titik fitur terdeteksi dengan lebih akurat dan konsisten, menandakan bahwa FAST mampu menangkap struktur nyata dari objek dengan kontras yang cukup.

Hal serupa juga terlihat pada gambar ruangan di mana deteksi fitur cenderung lebih banyak dan padat karena perbedaan intensitas antar permukaan cukup jelas. Namun, walaupun jumlah titik yang terdeteksi lebih banyak, sebagian besar berada di lokasi yang relevan dengan tepi dan sudut yang benar, menunjukkan kinerja FAST yang baik dalam mengenali fitur penting meskipun terkadang agak berlebihan pada area bertekstur halus.

Part 4: Calibration

1. Teori Singkat

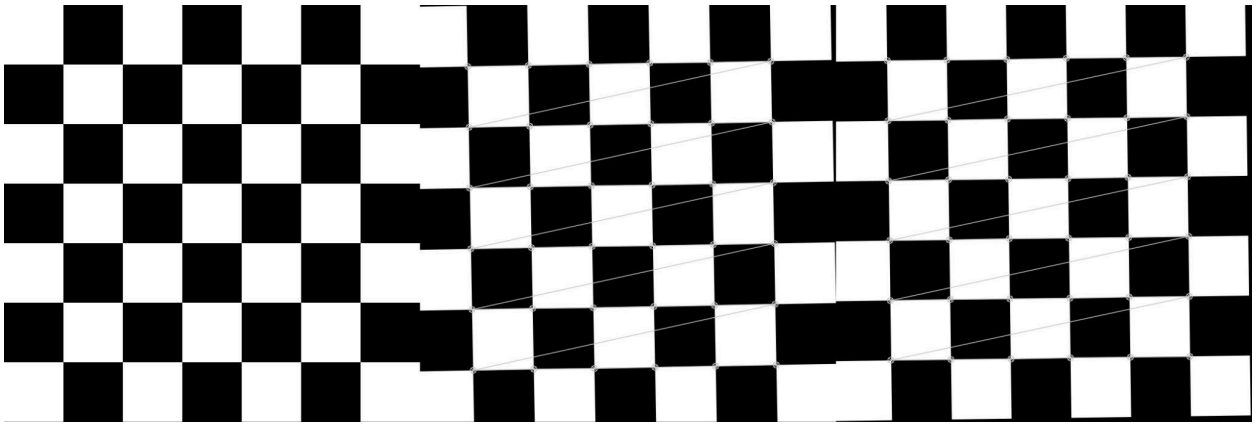
Kalibrasi kamera adalah proses untuk menentukan parameter intrinsik dan ekstrinsik kamera agar hubungan antara koordinat dunia nyata dan koordinat citra dapat diketahui secara akurat. Salah satu metode yang paling umum digunakan adalah dengan pola checkerboard (papan catur) karena pola kotak-kotaknya memiliki sudut (corner) yang mudah dideteksi secara otomatis oleh algoritma visi komputer.

Proses kalibrasi dimulai dengan mengambil beberapa gambar checkerboard dari berbagai sudut dan posisi. Setiap gambar digunakan untuk mendeteksi posisi sudut-sudut (corner) pada pola tersebut. Dari hasil deteksi ini, OpenCV menggunakan fungsi cv2.calibrateCamera() untuk menghitung:

- Parameter intrinsik, seperti *focal length*, *optical center*, dan *koefisien distorsi lensa*, yang menggambarkan karakteristik internal kamera.
- Parameter ekstrinsik, seperti *rotasi* dan *translasi*, yang menggambarkan posisi dan orientasi kamera terhadap objek di dunia nyata.

Dengan hasil kalibrasi ini, sistem dapat mengoreksi distorsi lensa dan melakukan pemetaan antara dunia nyata dan citra dengan lebih presisi.

2. Hasil



Gambar 4.1 Original Images, View 1, View 2

Pada gambar tersebut dapat terlihat terdapat garis - garis yang digunakan oleh CV untuk melakukan kalibrasi kamera. Hal ini dilakukan dengan mencari *corner* pada masing - masing gambar. Kemudian, hasil pemetaan tersebut digunakan untuk menentukan parameter intrinsic dan ekstrinsik dari kamera yang digunakan.

Pada percobaan ini, terdapat beberapa kali skenario dimana CV tidak berhasil menemukan *corner* pada *chessboard* tersebut. Hal ini disebabkan karena translasi ataupun rotasi yang terlalu ekstrem pada gambar original sehingga membuat proses kalibrasi kamera menjadi gagal (tidak mendapatkan parameter intrinsik maupun ekstrinsik). Hasil dari parameter ini kemudian dapat digunakan untuk melakukan *mapping* dari 2D ke 3D ataupun sebaliknya dengan akurasi yang lebih baik.

Refleksi Pribadi

Pada pengerjaan tugas ini digunakan berbagai library seperti OpenCV dan Scikit-Image untuk melakukan deteksi fitur, tepi, serta kalibrasi kamera. Selain itu, beberapa bagian seperti Gaussian filter dan Sobel operator juga diimplementasikan secara manual menggunakan NumPy untuk memahami proses dasar di balik fungsi bawaan pustaka tersebut.

Dari proses implementasi, saya menyadari bahwa penerapan teori ke dalam kode ternyata jauh lebih kompleks dibandingkan hanya memahami konsepnya saja. Namun, justru melalui proses implementasi ini saya menjadi lebih memahami bagaimana algoritma bekerja secara mendalam mulai dari proses konvolusi, pengaruh parameter terhadap hasil, hingga interpretasi hasil deteksi fitur dan tepi.

Bagian kalibrasi kamera juga menjadi hal yang sangat menarik karena memberikan gambaran nyata bagaimana hubungan antara dunia tiga dimensi dan citra dua dimensi dapat dimodelkan secara matematis.

Menurut saya, tahap lanjutan yang menarik untuk dilakukan adalah mencoba pemetaan 3D atau rekonstruksi posisi objek menggunakan parameter hasil kalibrasi.

Dari sisi penulisan kode, saya juga berupaya untuk membuat struktur program yang rapi, modular, dan mudah dibaca, dengan fungsi-fungsi terpisah, dokumentasi yang jelas, serta mengikuti konvensi PEP 8 agar dapat digunakan atau dikembangkan lebih lanjut dengan mudah.

Secara keseluruhan, tugas ini memberikan pengalaman yang sangat berharga karena tidak hanya melatih kemampuan pemrograman dan analisis citra, tetapi juga memperdalam pemahaman saya terhadap hubungan antara teori dan praktik dalam pengolahan citra digital.

Lampiran

Kode dapat di-*download* melalui link sebagai berikut:
https://github.com/Flame25/Christopher-Richard_18222057_IF5152_TugasIndividuCV-.git

Cara Mendownload:

1. Melakukan git clone

git clone https://github.com/Flame25/Christopher-Richard_18222057_IF5152_TugasIndividuCV-.git