

SKRIPSI

PENERAPAN DATA MINING PADA MASALAH PENGENALAN POINT OF INTEREST



Gian Martin Dwibudi

NPM: 6181801015

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN

«tahun»

UNDERGRADUATE THESIS

**APPLICATION OF DATA MINING ON THE PROBLEM OF
RECOGNIZING POINT OF INTEREST**



Gian Martin Dwibudi

NPM: 6181801015

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY**

«tahun»

LEMBAR PENGESAHAN

PENERAPAN DATA MINING PADA MASALAH PENGENALAN POINT OF INTEREST

Gian Martin Dwibudi

NPM: 6181801015

Bandung, «tanggal» «bulan» «tahun»

Menyetujui,

Pembimbing

/KDH

Ketua Tim Penguji

Anggota Tim Penguji

«penguji 1»

«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PENERAPAN DATA MINING PADA MASALAH PENGENALAN POINT OF INTEREST

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal «tanggal» «bulan» «tahun»



Gian Martin Dwibudi
NPM: 6181801015

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini. . . ?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini . . . »

Bandung, «bulan» «tahun»

Penulis

DAFTAR ISI

| | |
|--------------------------------------------------------|-------------|
| KATA PENGANTAR | xv |
| DAFTAR ISI | xvii |
| DAFTAR GAMBAR | xix |
| 1 PENDAHULUAN | 1 |
| 1.1 Latar Belakang | 1 |
| 1.2 Rumusan Masalah | 2 |
| 1.3 Tujuan | 3 |
| 1.4 Batasan Masalah | 3 |
| 1.5 Metodologi | 3 |
| 1.6 Sistematika Pembahasan | 3 |
| 2 LANDASAN TEORI | 5 |
| 2.1 Point of Interest | 5 |
| 2.2 Object Instance Recognition | 6 |
| 2.3 SIFT (Scale Invariant Feature Transform) | 9 |
| 2.3.1 Pencarian Extrema | 9 |
| 2.3.2 Penentuan Skala | 12 |
| 2.3.3 Penentuan Orientasi | 12 |
| 2.3.4 Pembuatan Deskriptor | 14 |
| 2.3.5 SIFT di OpenCV | 14 |
| 2.4 ORB (Oriented FAST and Rotated BRIEF) | 14 |
| 2.4.1 Pencarian Keypoint | 15 |
| 2.4.2 Penentuan Skala | 16 |
| 2.4.3 Penentuan Orientasi | 16 |
| 2.4.4 Pembuatan Deskriptor | 17 |
| 2.4.5 ORB di OpenCV | 19 |
| 2.5 BSIS (Best Score Increasing Subsequence) | 19 |
| 2.5.1 Pairing | 20 |
| 2.5.2 Verification | 20 |
| 2.5.3 Scoring | 22 |
| 2.6 KD-Tree | 22 |
| 2.7 Clustering | 23 |
| 2.7.1 Agglomerative Clustering | 23 |
| 2.7.2 DBSCAN | 24 |
| DAFTAR REFERENSI | 27 |
| A KODE PROGRAM | 29 |
| B HASIL EKSPERIMEN | 31 |

DAFTAR GAMBAR

| | | |
|------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| 1.1 | Contoh POI yang memiliki logo unik. POI seperti ini dapat dikenali dengan melakukkan identifikasi pada logo tersebut. | 1 |
| 1.2 | Empat gambar di atas menunjukkan permasalahan yang dihadapi pada penelitian ini. Gambar (a) dan (b) merupakan gambar dari dua POI yang berbeda tetapi memiliki logo dengan bagian sudut yang mirip, sudut yang mirip tersebut kemungkinan akan menghasilkan fitur lokal yang mirip juga. Sedangkan pada gambar (c) dan (d) merupakan dua gambar dari POI yang sama tetapi banyak objek latar yang berbeda sehingga akan memunculkan fitur lokal yang tidak konsisten. | 2 |
| 2.1 | Salah satu contoh POI | 5 |
| 2.2 | Contoh POI dengan logo unik. | 5 |
| 2.3 | Contoh variasi pada gambar <i>cover</i> buku yang dapat menyebabkan masalah pada OIR | 7 |
| 2.4 | Ilustrasi pasangan fitur lokal yang konsisten dan tidak konsisten. | 8 |
| 2.5 | Kurva Gaussian dan bentuk representasi <i>matrix</i> -nya. | 9 |
| 2.6 | Kurva dan <i>matrix</i> Gaussian pada nilai σ yang berbeda. | 10 |
| 2.7 | Efek nilai σ pada hasil gambar konvolusi. | 10 |
| 2.8 | Operasi DoG pada gambar | 11 |
| 2.9 | Penggunaan DoG pada SIFT | 11 |
| 2.10 | Oktaf pada proses konvolusi SIFT | 12 |
| 2.11 | Ilustrasi pembobotan pada <i>Gaussian Weighting</i> . Titik tengah merupakan <i>keypoint</i> yang diperiksa sedangkan setiap kotak merupakan <i>pixel-pixel</i> di sekitar <i>keypoint</i> . Tanda panah pada tiap kotak menunjukkan <i>magnitude</i> dan orientasi <i>pixel</i> tersebut, panjang panah merupakan nilai <i>magnitude</i> dan arahnya merupakan orientasi | 13 |
| 2.12 | Histogram untuk menentukan orientasi dari <i>keypoint</i> . Bin dengan nilai tertinggi (tanda panah biru) akan digunakan sebagai orientasi dari <i>keypoint</i> . Untuk bin lain yang jumlahnya berada dalam rentang 80% dari bin tertinggi (tanda panah hijau) digunakan untuk membuat <i>keypoint baru</i> | 13 |
| 2.13 | Ilustrasi penentuan <i>keypoint</i> pada ORB. Setiap kotak menunjukkan sebuah <i>pixel</i> pada gambar dan angka di dalamnya merupakan nilai intensitasnya. <i>Pixel</i> di tengah gambar (<i>pixel</i> bernilai 4) merupakan <i>pixel</i> yang akan diperiksa apakah merupakan <i>keypoint</i> . <i>Pixel-pixel</i> yang ditandai dengan kotak putih merupakan 16 <i>pixel</i> yang digunakan untuk memeriksa apakah <i>pixel</i> di tengah merupakan <i>keypoint</i> | 15 |
| 2.14 | <i>Image Pyramid</i> pada ORB | 16 |
| 2.15 | Ilustrasi penentuan orientasi pada ORB. | 17 |
| 2.16 | Ilustrasi penghitungan <i>Integral Image</i> . <i>Matrix I</i> merupakan <i>matrix</i> awal dan <i>Matrix I_{Σ}</i> merupakan <i>Integral Image</i> dari <i>I</i> | 18 |
| 2.17 | Penghitungan nilai total sebuah daerah dengan menggunakan <i>Integral Image</i> | 19 |
| 2.18 | Tahapan verifikasi pada BSIS. | 21 |
| 2.19 | Contoh pohon struktur KD-Tree. Angka di sebelah kanan menunjukkan elemen vektor yang diperiksa pada level tersebut. | 22 |
| 2.20 | Contoh <i>cluster</i> dengan bentuk non- <i>circular</i> . Objek-objek yang tersebar seperti ini dapat tergabung menjadi <i>cluster</i> dengan menggunakan DBSCAN. | 25 |

| | | |
|-----|-------------------|----|
| B.1 | Hasil 1 | 31 |
| B.2 | Hasil 2 | 31 |
| B.3 | Hasil 3 | 31 |
| B.4 | Hasil 4 | 31 |

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Sebuah lokasi atau titik geografis yang memiliki kegunaan tertentu biasa disebut sebagai *Point of interest* (POI). POI ini dapat berupa tempat apa saja yang memiliki ciri khas tertentu dan dapat dikenali dari ciri khas tersebut. POI-POI tertentu dapat dikenali dari logo tempat tersebut atau objek-objek lainnya yang terlihat secara langsung. Seperti ditunjukkan pada Gambar 1.1, kedua POI tersebut memiliki logo unik yang terlihat dengan jelas.

Pada skripsi ini akan dibuat sebuah sistem untuk mengidentifikasi POI dari masukan yang berisi gambar POI tersebut. Proses identifikasi POI dilakukan dengan mendeteksi logo khusus atau objek unik yang ada pada POI tersebut.



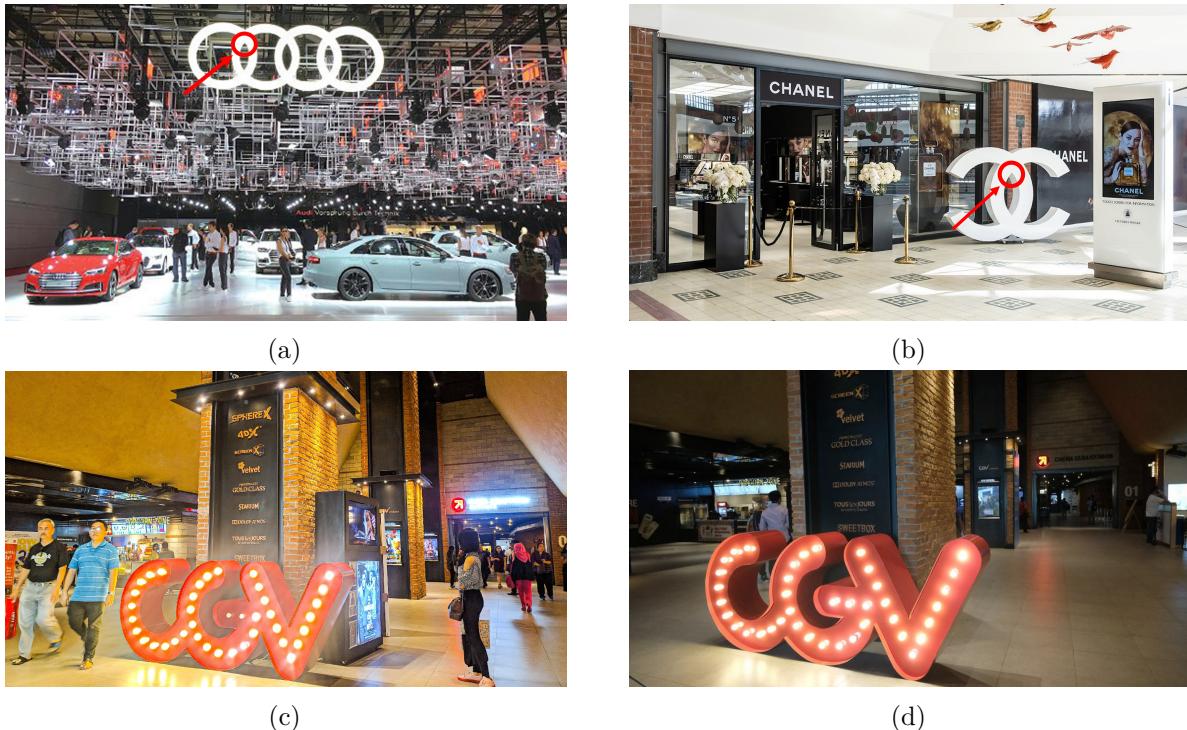
Gambar 1.1: Contoh POI yang memiliki logo unik. POI seperti ini dapat dikenali dengan melakukan identifikasi pada logo tersebut.

Proses identifikasi POI akan dilakukan menggunakan teknik *Object Instance Recognition* (OIR). Teknik OIR merupakan teknik pengenalan objek spesifik. Sebuah algoritma OIR harus dapat mengatasi masalah seperti pencahayaan, sudut pengambilan, objek *background*. Objek harus tetap dapat diidentifikasi walaupun gambar memiliki gangguan-gangguan tersebut. OIR dapat dilakukan dengan memanfaatkan fitur lokal.

Fitur lokal merupakan fitur yang mendeskripsikan sebuah daerah penting (*keypoint*) pada gambar. Salah satu cara mendapatkan *Keypoint* adalah dengan mencari sudut-sudut atau perpotongan garis (*corner*) yang terdapat pada gambar. *Keypoint-keypoint* yang terdeteksi pada gambar akan memiliki sebuah vektor untuk mendeskripsikan daerah di sekitar *keypoint* tersebut yang disebut sebagai vektor deskriptor. Proses pencarian fitur lokal pada penelitian ini akan dilakukan dengan menggunakan metode *Scale Invariant Feature Transform* (SIFT) dan *Oriented FAST and Rotated BRIEF* (ORB). Metode SIFT dan ORB akan menghasilkan vektor deskriptor untuk tiap fitur lokal yang terdeteksi, vektor deskriptor ini dapat digunakan untuk mengidentifikasi fitur lokal.

Salah satu masalah yang ada pada pengenalan POI adalah pada sebuah gambar POI tidak semua fitur lokal yang dideteksi bersifat unik terhadap POI tersebut. Ada fitur lokal yang juga

dimiliki oleh POI lain atau fitur lokal yang berasal dari objek di latar belakang yang sifatnya tidak konsisten. Masalah ini akan mempersulit pada proses OIR untuk mengidentifikasi POI yang tepat. Gambar 1.2 menunjukkan masalah-masalah ini, gambar 1.2a dan 1.2b menunjukkan fitur lokal yang mirip dari dua POI yang berbeda, sedangkan gambar 1.2c dan 1.2d menunjukkan objek-objek latar belakang yang tidak konsisten pada POI. Penelitian ini akan melakukan analisis untuk menemukan dan memisahkan fitur-fitur lokal tersebut agar tidak diproses dalam pembuatan model POI.



Gambar 1.2: Empat gambar di atas menunjukkan permasalahan yang dihadapi pada penelitian ini. Gambar (a) dan (b) merupakan gambar dari dua POI yang berbeda tetapi memiliki logo dengan bagian sudut yang mirip, sudut yang mirip tersebut kemungkinan akan menghasilkan fitur lokal yang mirip juga. Sedangkan pada gambar (c) dan (d) merupakan dua gambar dari POI yang sama tetapi banyak objek latar yang berbeda sehingga akan memunculkan fitur lokal yang tidak konsisten.

Fitur-fitur lokal yang tidak unik dan tidak konsisten tersebut akan dipisahkan dengan menggunakan metode *clustering*. Metode *clustering* merupakan teknik pemrosesan data yang akan mengelompokkan data-data dengan sifat yang mirip ke dalam satu kelompok. Metode *clustering* pada penelitian ini akan menggunakan metode *Agglomerative* dan DBSCAN. Penelitian ini mengasumsikan fitur lokal yang merepresentasikan suatu POI adalah fitur lokal yang muncul secara konsisten di gambar POI tersebut dan relatif unik terhadap POI tersebut.

1.2 Rumusan Masalah

Skripsi ini memiliki rumusan masalah sebagai berikut:

- Bagaimana membuat model pengenalan POI berdasarkan fitur lokalnya menggunakan teknik *data mining*?
- Bagaimana mengidentifikasi POI dalam sebuah gambar berisi POI dengan memanfaatkan model pengenalan POI yang telah dibuat?

1.3 Tujuan

Skripsi ini memiliki tujuan sebagai berikut:

- Membuat perangkat lunak yang akan menghasilkan model pengenalan POI berdasarkan dari *dataset* yang diberikan
- Membuat perangkat lunak yang dapat melakukan identifikasi POI dari sebuah gambar POI dengan menggunakan model yang dihasilkan.

1.4 Batasan Masalah

Berikut batasan-batasan masalah dari skripsi ini:

- Pengambilan fitur lokal untuk analisis dilakukan menggunakan metode SIFT dan ORB dengan implementasi OpenCV pada Python.

1.5 Metodologi

Skripsi ini akan memiliki metodologi sebagai berikut:

1. Melakukan studi literatur tentang metode OIR, teknik ekstraksi fitur lokal SIFT dan ORB, serta teknik-teknik *data mining* yang digunakan pada skripsi ini. Studi literatur dilakukan dengan mencari dan membaca *paper* atau buku yang berkaitan dengan topik tersebut.
2. Mengumpulkan *dataset* gambar POI yang diperlukan untuk penelitian dan pembuatan model identifikasi.
3. Melakukan analisis pada latar belakang masalah pengenalan POI, dengan melihat sifat-sifat fitur lokal pada gambar POI.
4. Menyusun rancangan perangkat lunak.
5. Melakukan implementasi perangkat lunak.
6. Menguji kinerja perangkat lunak.
7. Menulis buku skripsi.

1.6 Sistematika Pembahasan

Sistematika pembahasan yang digunakan pada penelitian ini adalah sebagai berikut:

1. Bab 1 Pendahuluan
Bab ini berisi tentang hal-hal yang menggambarkan skripsi ini secara garis besar. Hal yang dibahas merupakan latar belakang masalah, rumusan masalah, tujuan penelitian, batasan masalah, dan metodologi penelitian.
2. Bab 2 Landasan Teori
Bab ini berisi tentang dasar-dasar teori dari teknik atau metode yang digunakan dalam skripsi ini, yaitu POI, OIR, metode SIFT, metode ORB, *Best Score Increasing Subsequence* (BSIS), KD-Tree, teknik *clustering Agglomerative* dan DBSCAN, serta metode SIFT dan ORB di *library* OpenCV.
3. Bab 3 Analisis
Bab ini berisi analisis pada masalah yang dibahas pada skripsi beserta solusi yang digunakan untuk menyelesaikan masalah tersebut.
4. Bab 4 Perancangan
Bab ini berisi tentang perancangan baik dari metode *clustering* pada pemilihan fitur dan perancangan metode identifikasi POI dengan OIR. Bab juga akan berisi rancangan struktur *file* dan *folder* pada hasil akhir perangkat lunak.
5. Bab 5 Implementasi dan Pengujian
Bab ini berisi implementasi perangkat lunak pada metode pemilihan fitur dan identifikasi POI serta pengujian terhadap kinerja kedua metode tersebut.

6. Bab 6 Kesimpulan dan Saran

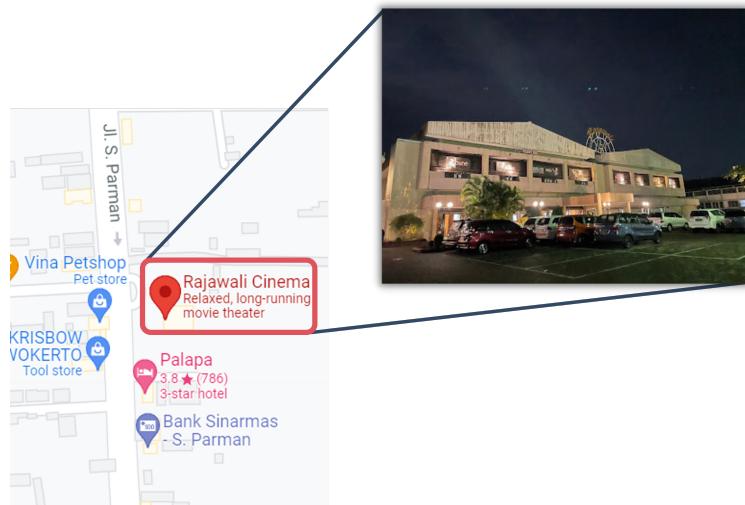
Bab ini berisi kesimpulan yang didapatkan dari hasil analisis serta keseluruhan implementasi dan pengujian yang dilakukan pada penelitian ini.

BAB 2

LANDASAN TEORI

2.1 Point of Interest

Point of Interest (POI) adalah sebuah lokasi geografis yang memiliki kegunaan tertentu. POI biasanya dikenali oleh banyak orang dan memiliki keunikan tertentu pada tampilannya. Salah satu contoh POI dapat dilihat pada Gambar 2.1. POI juga dapat dimanfaatkan untuk menjadi penanda lokasi seseorang. Seseorang dapat mengerti lokasinya dengan melihat POI yang ada di sekitarnya.



Gambar 2.1: Salah satu contoh POI.

Beberapa POI tertentu dapat memiliki logo yang sifatnya unik. POI tersebut dapat dikenali dengan hanya melihat logonya saja. Contoh bisa dilihat pada Gambar 2.2. POI dengan logo unik ini dapat dikenali oleh komputer salah satunya dengan menggunakan teknik *Object Instance Recognition* (OIR).



Gambar 2.2: Contoh POI dengan logo unik.

2.2 Object Instance Recognition

Object Instance Recognition (OIR) adalah teknik pengenalan objek spesifik. Pada teknik OIR deteksi tidak memberikan kelas dari gambar tetapi label yang khusus, seperti contohnya jika objek merupakan sebuah mobil, OIR tidak hanya akan memberikan keluaran bahwa objek merupakan mobil melainkan hal yang spesifik seperti merk dan jenis mobil tersebut. OIR bekerja dengan menerima gambar masukkan dan mencari gambar pada *dataset* yang memiliki paling banyak kemiripan.

Penyelesaian OIR akan mudah bila gambar masukkan merupakan gambar yang sudah bersih. Pada praktiknya sebuah algoritma OIR seharusnya tetap dapat mengenali objek walaupun gambar bervariasi. Beberapa perubahan pada gambar berikut merupakan faktor-faktor yang dapat mempersulit proses OIR:

- Cahaya

Perubahan pada tingkat pencahayaan pada gambar yang mengakibatkan perubahan nilai *pixel-pixel* pada gambar.

- Skala

Jarak diambilnya gambar yang berisi objek. Perbedaan ukuran objek pada gambar akan menyebabkan sudut-sudut pada objek menjadi berbeda.

- Rotasi

Orientasi atau arah pengambilan gambar yang berbeda akan mengakibatkan objek pada gambar jadi terlihat berbeda. Sudut-sudut akan menjadi berbeda karena arah hadapnya berbeda.

- Latar Belakang

Objek-objek lain di sekitar objek yang ingin diidentifikasi akan berpotensi mempersulit pemrosesan. Objek-objek tersebut dapat menghasilkan fitur-fitur lokal yang tidak relevan terhadap objek yang ingin diidentifikasi.

- Bagian Objek Tertutup

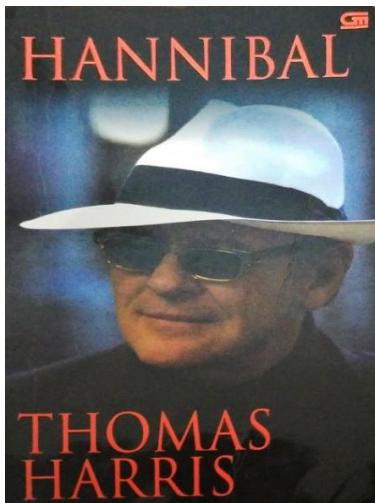
Adanya objek lain yang menutupi sebagian dari objek yang ingin diidentifikasi akan berpotensi menyebabkan beberapa fitur lokal dari objek tidak terdeteksi.

- Sudut Pandang

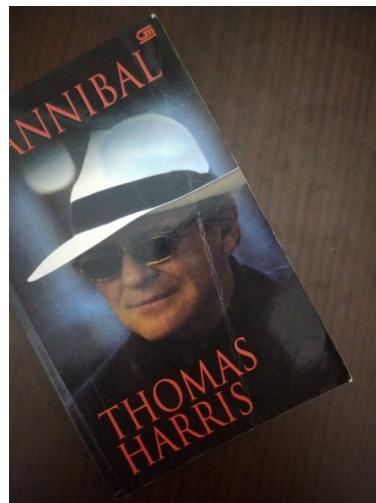
Sudut pengambilan gambar yang berbeda akan memengaruhi pemrosesan. Fitur-fitur lokal dari objek yang ingin diidentifikasi akan menjadi berbeda.

- Translasi

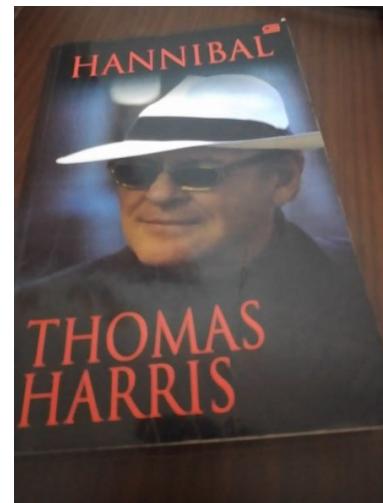
Posisi objek yang ingin diidentifikasi dalam gambar akan memengaruhi pemrosesan. Pencarian pasangan fitur lokal tidak dapat dengan hanya menggunakan posisi di mana fitur lokal tersebut ditemukan pada gambar.



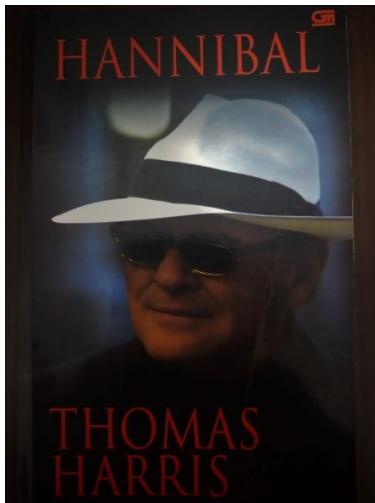
(a) Gambar objek yang ideal, dari sudut tegak lurus



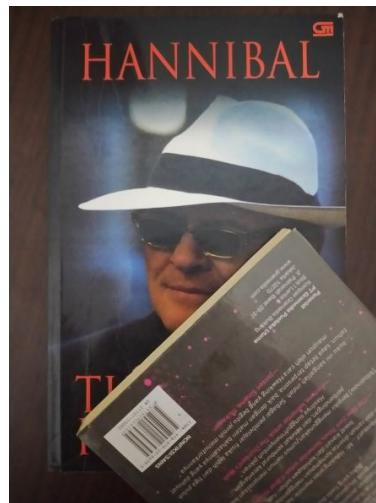
(b) Objek pada gambar mengalami translasi dan rotasi



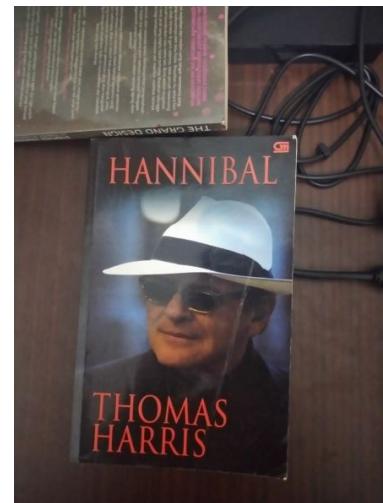
(c) Gambar diambil dari sudut yang berbeda



(d) Gambar dengan pencahayaan yang berbeda



(e) Objek pada gambar terhalang oleh objek lain



(f) Terdapat objek lain di latar belakang dari objek utama

Gambar 2.3: Contoh variasi pada gambar *cover* buku yang dapat menyebabkan masalah pada OIR

OIR dapat dilakukan dengan menggunakan fitur lokal. Fitur lokal sendiri merupakan fitur dalam gambar yang mendeskripsikan sebuah daerah tertentu pada gambar tersebut. Fitur lokal dapat berupa perpotongan garis atau yang biasa disebut *keypoint*. Sebuah *keypoint* akan dapat diidentifikasi dengan menggunakan daerah di sekitar *keypoint* tersebut. Deskripsi *keypoint* ini dibuat dalam sebuah vektor yang dinamakan vektor deskriptor.

Teknik OIR bekerja dengan melakukan deteksi fitur lokal pada gambar masukkan dan pada gambar-gambar di *dataset*. Fitur-fitur lokal dari gambar masukkan tersebut kemudian dipasangkan dengan fitur lokal dari gambar *dataset*. Gambar yang memiliki pasangan terbanyak akan merupakan hasil deteksi gambar masukkan tersebut.

Pengambilan fitur lokal dari gambar dapat dilakukan dengan beberapa metode, seperti SIFT (lihat 2.3) dan ORB (lihat 2.4). Kedua metode tersebut—SIFT dan ORB—sudah menangani masalah perubahan translasi, skala dan rotasi karena fitur lokal yang dihasilkan oleh SIFT dan ORB bersifat invariant terhadap translasi, skala dan rotasi.

Proses pencarian pasangan fitur lokal dari gambar masukkan dan *dataset* dilakukan dengan menggunakan vektor deskriptor dari fitur lokal. Perubahan-perubahan pada gambar walaupun sedikit akan menyebabkan perubahan nilai pada vektor deskriptor. Vektor deskriptor dari fitur

lokal pada gambar masukkan hampir tidak akan persis sama dengan vektor deskriptor dari gambar yang ada di *dataset*. Oleh karena itu pencarian pasangan fitur lokal dilakukan dengan mencari pasangan fitur lokal yang memiliki nilai kemiripan paling tinggi.

Secara garis besar tahapan proses OIR pada penelitian ini adalah sebagai berikut:

1. Ekstraksi Fitur

Pertama akan dilakukan pengambilan fitur-fitur lokal dari gambar masukkan. Dengan menggunakan metode SIFT atau ORB pengambilan fitur lokal akan menghasilkan *tuple* yang berisi *keypoint*. Setiap *keypoint* yang dihasilkan akan memiliki sebuah vektor yang akan digunakan untuk mengidentifikasi *keypoint* tersebut. Vektor ini disebut sebagai vektor deskriptor.

2. Pairing

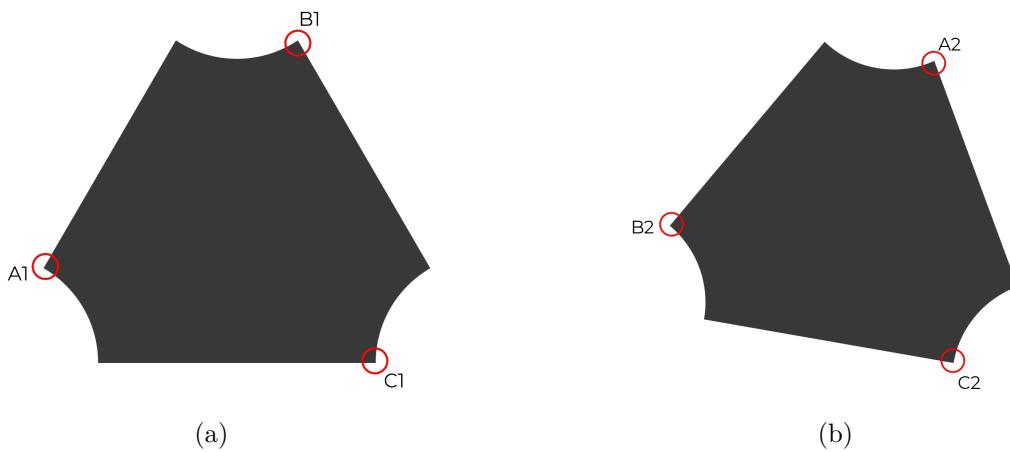
Untuk setiap fitur lokal yang terdeteksi pada gambar masukkan akan dicari beberapa fitur lokal dari gambar pada *dataset* yang paling mirip. Kemiripan fitur lokal ditentukan dengan menghitung jarak (sesuai dengan metrik yang digunakan) antara vektor deskriptor dari kedua fitur lokal tersebut.

3. Verification

Pasangan fitur lokal yang didapat pada tahap sebelumnya merupakan pasangan yang hanya memiliki ciri yang mirip tetapi belum tentu konsisten secara geometris. Pasangan fitur lokal dikatakan konsisten secara geometris jika keduanya memiliki posisi yang sama relatif terhadap pasangan lain di sekitarnya. Penjelasan mengenai pasangan yang konsisten akan dijelaskan pada paragraf di bawah. Hanya pasangan-pasangan fitur lokal yang konsisten secara geometris yang akan diproses lebih lanjut.

4. Scoring

Setelah didapatkan semua pasangan fitur lokal yang paling mirip dan konsisten secara geometris maka dapat ditentukan pasangan gambar yang menjadi label gambar masukkan. Kemudian perlu dihitung nilai kemiripan dari label yang dihasilkan. Kemiripan dapat dihitung dengan menghitung $\frac{1}{d}$ untuk semua pasangan fitur lokal, di mana d merupakan jarak pasangan tersebut. Nilai-nilai $\frac{1}{d}$ tersebut kemudian dihitung totalnya untuk menjadi nilai kemiripan label hasil.



Gambar 2.4: Ilustrasi pasangan fitur lokal yang konsisten dan tidak konsisten.

Ilustrasi untuk menunjukkan pasangan yang tidak konsisten dapat dilihat pada Gambar 2.4. Pada kedua gambar tersebut, fitur lokal A1, B1, dan C1 pada Gambar 2.4a secara berurutan dipasangkan dengan fitur lokal A2, B2, dan C2 pada Gambar 2.4b.

Pada orientasi seperti di gambar, pasangan A1 dengan A2 dan B1 dengan B2 merupakan pasangan yang tidak konsisten. Hal tersebut dikarenakan posisi A1 yang berada di sebelah kiri B1 tetapi pasangan dari A1, yaitu A2 berada di sebelah kiri pasangan dari B2, pasangan dari B1. Sedangkan pasangan C1 dengan C2 merupakan pasangan yang konsisten, karena baik C1 maupun

C2 memiliki posisi relatif yang sama terhadap pasangan fitur lokal lain.

Gambar 2.4b dapat dirotasi sebanyak 180 derajat sehingga B2 berada di sebelah kanan A1. Dengan menggunakan orientasi gambar yang seperti ini, pasangan A1 dengan A2 dan pasangan B1 dengan B2 menjadi konsisten posisinya secara geometris. Tetapi dengan orientasi yang seperti ini pasangan C1 dengan C2 menjadi tidak konsisten karena posisi C2 pada Gambar 2.4b akan berada di sebelah kiri A2 dan B2.

Salah satu metode untuk melakukan verifikasi geometris adalah BSIS (lihat 2.5). BSIS dapat mencari pasangan-pasangan fitur lokal yang posisi relatif konsisten terhadap pasangan lain. Verifikasi yang dilakukan BSIS dilakukan dengan menggunakan beberapa orientasi gambar untuk mendapatkan orientasi yang menghasilkan nilai kemiripan paling tinggi.

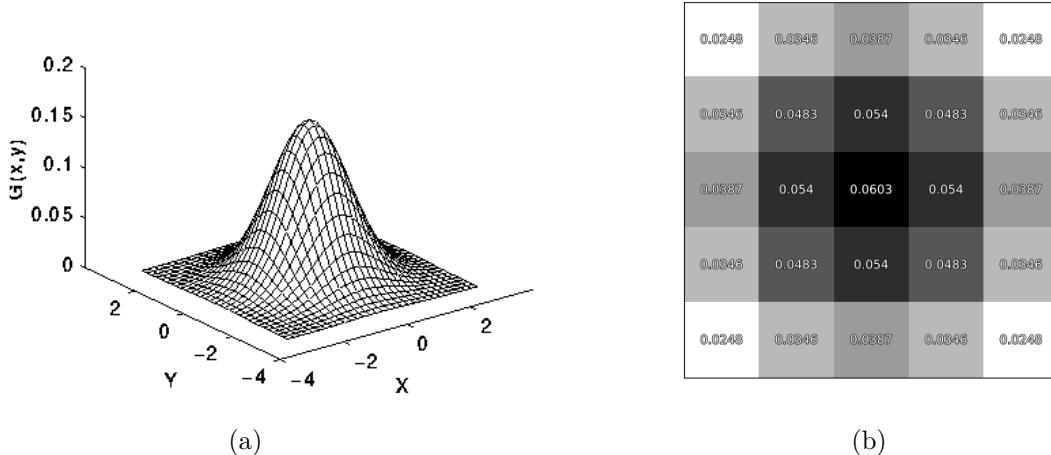
2.3 SIFT (Scale Invariant Feature Transform)

SIFT adalah salah satu metode pencarian fitur lokal yang dicetuskan pada [1]. Fitur lokal yang dihasilkan SIFT bersifat invariant terhadap rotasi, perubahan skala, dan translasi pada gambar. Sifat invariant ini berarti fitur lokal yang sama pada gambar yang telah dirotasi, diubah skalanya, atau ditranslasi akan tetap memiliki ciri yang mirip. Setiap fitur lokal akan memiliki sebuah vektor yang mendeskripsikan daerah area fitur lokal tersebut, vektor ini biasa disebut sebagai deskriptor. Vektor deskriptor SIFT berbentuk vektor bilangan bulat yang memiliki 128 elemen. Tahap pencarian fitur lokal pada SIFT dapat dibagi menjadi 4 langkah yang akan dijabarkan pada subbab-subbab berikut.

2.3.1 Pencarian Extrema

Pada tahap ini akan dicari *pixel-pixel* pada gambar yang merupakan *corner* atau biasa disebut *keypoint*. *Keypoint* pada SIFT dicari dengan memeriksa *pixel-pixel* pada gambar hasil turunan kedua Gaussian. Turunan kedua Gaussian akan dihitung dengan menggunakan *Difference of Gaussian* (DoG).

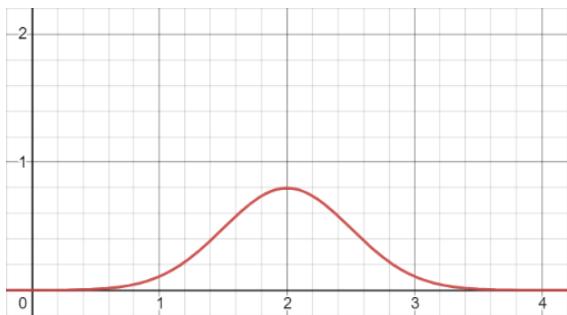
Penghitungan DoG ini dilakukan dengan memanfaatkan sifat dari *Matrix Konvolusi Gaussian*. *Matrix Konvolusi Gaussian* merupakan *matrix* yang memiliki sifat distribusi Gaussian, di mana titik tengah *matrix* memiliki nilai yang tinggi dan nilai-nilai di sekitarnya semakin mengecil semakin mendekati tepi *matrix*. Seperti ditunjukkan pada Gambar 2.5.



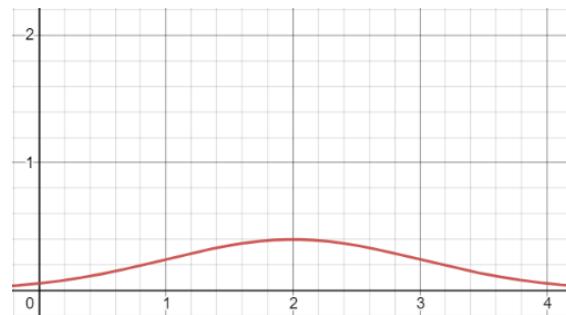
Gambar 2.5: Kurva Gaussian dan bentuk representasi *matrix*-nya.

Pada fungsi Gaussian tingkat penyebaran data dapat diatur dengan mengubah parameter σ yang mengatur nilai deviasi standar. Gambar 2.5a menunjukkan bagaimana nilai σ mengatur bagaimana data tersebar dari *mean*, di mana semakin tinggi nilai σ maka data akan semakin menyebar. Nilai yang semakin menyebar menyebabkan perbedaan nilai antar titik semakin kecil. Efeknya pada

matrix dapat dilihat pada Gambar 2.6. Nilai σ yang tinggi menyebabkan kurva semakin melebar dan pada *matrix* selisih nilai antar titik menjadi semakin kecil.



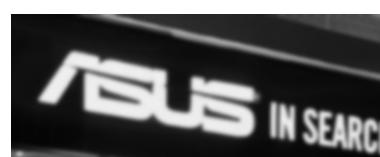
| | | | | |
|--------|--------|--------|--------|--------|
| 0.0001 | 0.0021 | 0.0058 | 0.0021 | 0.0001 |
| 0.0021 | 0.0431 | 0.1171 | 0.0431 | 0.0021 |
| 0.0058 | 0.1171 | 0.3183 | 0.1171 | 0.0058 |
| 0.0021 | 0.0431 | 0.1171 | 0.0431 | 0.0021 |
| 0.0001 | 0.0021 | 0.0058 | 0.0021 | 0.0001 |



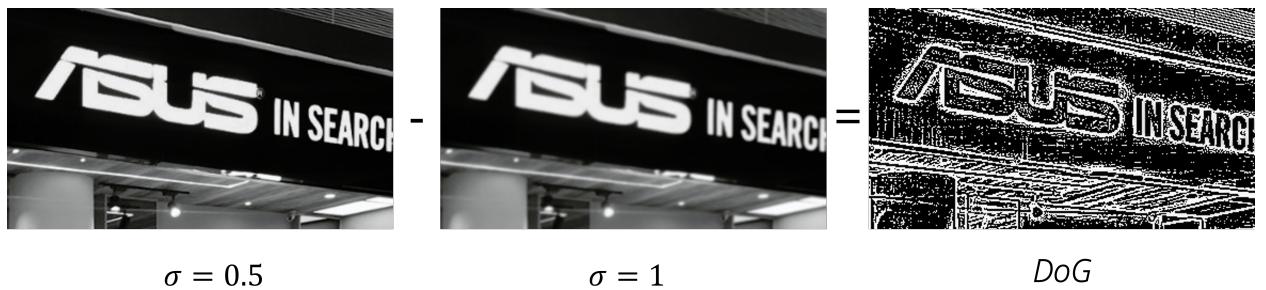
| | | | | |
|--------|--------|--------|--------|--------|
| 0.0125 | 0.0264 | 0.0339 | 0.0264 | 0.0125 |
| 0.0264 | 0.0559 | 0.0718 | 0.0559 | 0.0264 |
| 0.0339 | 0.0718 | 0.0922 | 0.0718 | 0.0339 |
| 0.0264 | 0.0559 | 0.0718 | 0.0559 | 0.0264 |
| 0.0125 | 0.0264 | 0.0339 | 0.0264 | 0.0125 |

(a) $\sigma = 1$ (b) $\sigma = 2$ Gambar 2.6: Kurva dan *matrix* Gaussian pada nilai σ yang berbeda.

Matrix Konvolusi Gaussian ketika diaplikasikan pada gambar akan menyebabkan perubahan nilai tiap *pixel* pada gambar. Nilai dari setiap *pixel* akan menjadi mirip dengan *pixel* tetangga di dekatnya. Perubahan nilai *pixel* akan paling berpengaruh pada daerah dengan perubahan nilai *pixel* yang tinggi. Tingkat perubahan nilai dipengaruhi oleh nilai σ yang digunakan, nilai σ yang tinggi akan menyebabkan nilai *pixel* yang berdekatan semakin mirip—perubahan nilai *pixel* pada daerah tersebut semakin mengecil. Jika dilihat pada gambar, maka gambar hasil konvolusi akan terlihat kabur (*blur*). Nilai σ menentukan tingkat *blur* gambar.

(a) $\sigma = 1.0$ (b) $\sigma = 1.4$ (c) $\sigma = 1.8$ Gambar 2.7: Efek nilai σ pada hasil gambar konvolusi.

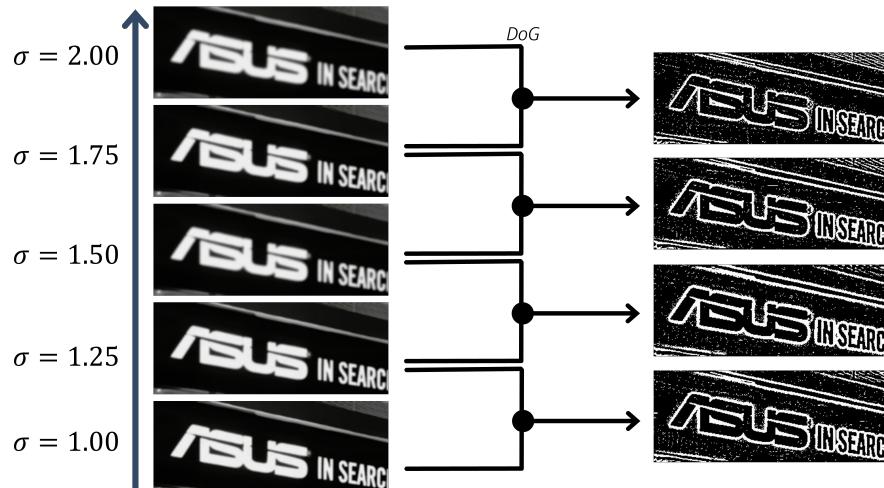
Perubahan nilai σ pada *matrix* konvolusi serta efeknya pada gambar akan dimanfaatkan untuk menghitung *Difference of Gaussian* (DoG). DoG merupakan hasil turunan kedua Gaussian pada gambar. Gambar DoG dapat diperoleh dengan menghitung perbedaan nilai tiap *pixel* dari dua gambar yang telah dikonvolusi oleh *matrix* Gaussian dengan nilai σ yang berbeda. Perbedaan nilai untuk DoG dihitung dengan mengurangi setiap *pixel* pada gambar konvolusi yang memiliki nilai σ yang lebih kecil, dengan setiap *pixel* pada posisi yang sama pada gambar konvolusi yang memiliki nilai σ yang lebih besar. Ilustrasi dapat dilihat pada Gambar 2.8.



Gambar 2.8: Operasi DoG pada gambar

Metode SIFT mencari *keypoint* dengan memanfaatkan konsep DoG. Sebuah gambar akan dikonvolusi dengan *matrix Gaussian* beberapa kali dengan nilai σ yang berbeda. Setelah didapatkan beberapa gambar maka akan dihitung DoG untuk setiap gambar yang nilai σ -nya bersebelahan (Gambar 2.9). Pasangan gambar konvolusi yang berbeda akan menghasilkan gambar DoG yang berbeda juga.

Untuk setiap gambar DoG akan ditentukan *pixel* mana saja yang merupakan *keypoint* dengan mencari *pixel* yang merupakan *extrema*. Sebuah *pixel* merupakan *extrema* jika nilai pixel tersebut lebih besar dari seluruh 26 *pixel* di sekitarnya atau lebih kecil dari seluruhnya. Ke-26 *pixel* tersebut merupakan 8 *pixel* yang mengelilingi, 9 *pixel* pada posisi yang sama dari gambar di atasnya, dan juga 9 *pixel* pada posisi yang sama dari gambar di bawahnya.



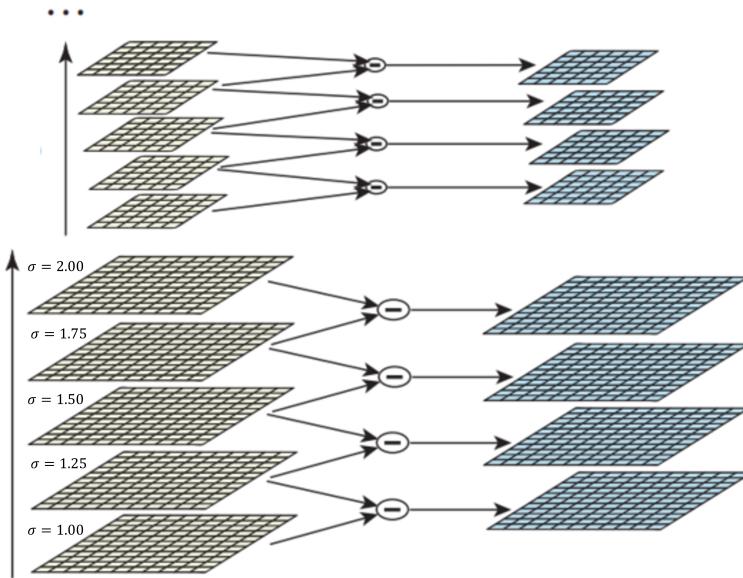
Gambar 2.9: Penggunaan DoG pada SIFT

2.3.2 Penentuan Skala

Pada tahap sebelumnya sudah didapatkan *keypoint-keypoint* dalam gambar. Agar *keypoint* dapat invariant terhadap skala, *keypoint* perlu untuk dapat tetap terdeteksi walaupun ukuran gambar berubah. Untuk setiap *keypoint* perlu untuk dicari skala terkecil di mana *keypoint* tersebut dapat terdeteksi. Untuk mencapai ini SIFT menggunakan lanjutan dari metode pada Gambar 2.9 dengan langkah sebagai berikut (ilustrasi pada Gambar 2.10):

1. Lakukan konvolusi sampai nilai σ sudah mencapai 2 kali nilai awal
2. Perkecil ukuran gambar (*downsample*) menjadi setengah resolusinya
3. Kembalikan nilai σ ke nilai awal
4. Ulang tahap dari langkah 1 hingga gambar sudah terlalu kecil.

Pada langkah di atas setiap siklus ukuran gambar disebut sebagai oktaf, dimulai dari oktaf pertama, lalu kedua, dan seterusnya. Dengan setiap oktaf ukuran gambar akan semakin kecil. Pencarian *keypoint* dilakukan pada tiap oktaf, dan untuk tiap *keypoint* tersebut ditulis nilai oktaf tertinggi (ukuran gambar terkecil) di mana *keypoint* tersebut dapat terdeteksi.



Gambar 2.10: Oktaf pada proses konvolusi SIFT

2.3.3 Penentuan Orientasi

Untuk dapat invariant terhadap rotasi gambar, setiap *keypoint* perlu memiliki orientasi yang konsisten. Untuk mendapatkan orientasi yang sama pada setiap rotasi gambar, orientasi perlu ditentukan dari atribut yang akan selalu sama bagaimanapun gambar dirotasi. Untuk itu orientasi *keypoint* ditentukan dengan menggunakan orientasi yang dominan dari *pixel-pixel* di sekitar *keypoint*. Luas daerah yang digunakan untuk mendapat orientasi ditentukan oleh skala dari *keypoint*.

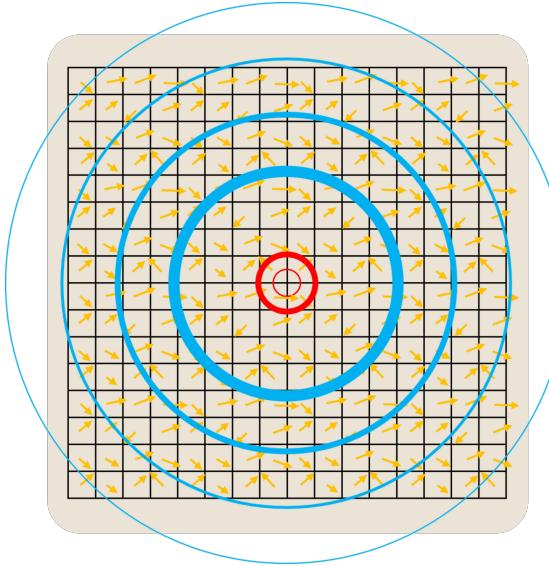
Penentuan orientasi yang dominan dihitung dengan menggunakan *magnitude* ($m(x, y)$) dan orientasi ($\theta(x, y)$) dari *pixel-pixel* dengan menggunakan rumus pada Persamaan 2.1 dan Persamaan 2.2. $L(x, y)$ pada kedua persamaan tersebut merupakan gambar hasil konvolusi.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \quad (2.1)$$

$$\theta(x, y) = \tan^{-1}((L(x, y+1) - L(x, y-1)) / (L(x+1, y) - L(x-1, y))) \quad (2.2)$$

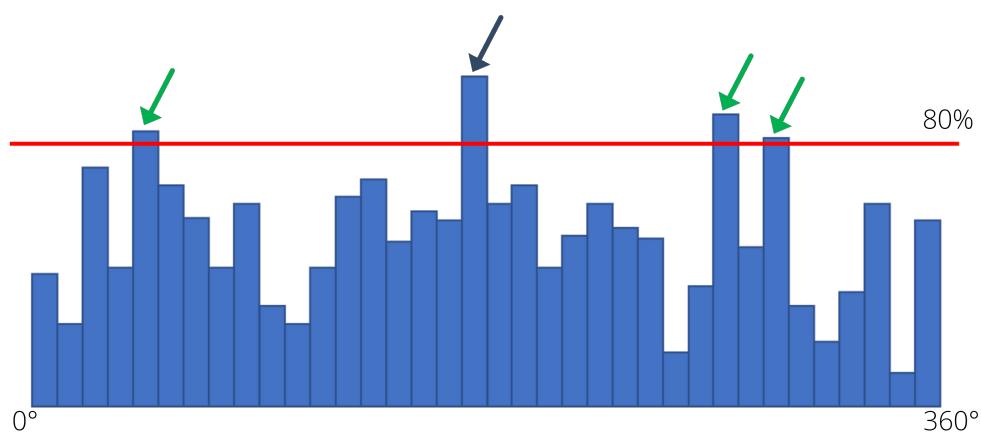
Setiap *pixel* akan dihitung orientasi dan *magnitude*-nya. *Magnitude* akan digunakan sebagai bobot dari *pixel* tersebut. Selain *magnitude*, bobot sebuah *pixel* juga dipengaruhi oleh *Gaussian Weighting*,

yang membuat *pixel* yang posisinya dekat dengan titik pusat (titik *keypoint*) akan memiliki bobot yang lebih tinggi dibanding yang lokasinya jauh dari titik pusat. Ilustrasi pada Gambar 2.11 menunjukkan bagaimana pembobotan dihitung, *pixel-pixel* yang berada dekat dengan *keypoint* (titik tengah) akan diberi bobot yang lebih besar—ditandai dengan lingkaran yang tebal. Sedangkan bobot akan semakin berkurang untuk *pixel* yang jauh dari *keypoint*.



Gambar 2.11: Ilustrasi pembobotan pada *Gaussian Weighting*. Titik tengah merupakan *keypoint* yang diperiksa sedangkan setiap kotak merupakan *pixel-pixel* di sekitar *keypoint*. Tanda panah pada tiap kotak menunjukkan *magnitude* dan orientasi *pixel* tersebut, panjang panah merupakan nilai *magnitude* dan arahnya merupakan orientasi

Setelah setiap *pixel* sudah dihitung orientasi dan bobotnya menggunakan *magnitude* dan *Gaussian Weighting*, nilai bobot tersebut akan dimasukkan ke dalam histogram berdasarkan orientasinya. Histogram yang digunakan memiliki 36 bin yang masing-masing mewakili 10 derajat orientasi. Ilustrasi dapat dilihat pada Gambar 2.12.



Gambar 2.12: Histogram untuk menentukan orientasi dari *keypoint*. Bin dengan nilai tertinggi (tanda panah biru) akan digunakan sebagai orientasi dari *keypoint*. Untuk bin lain yang jumlahnya berada dalam rentang 80% dari bin tertinggi (tanda panah hijau) digunakan untuk membuat *keypoint* baru.

Dari histogram tersebut puncak nilai bin tertinggi akan digunakan sebagai orientasi dari *keypoint*. Untuk puncak-puncak lain yang berada dalam rentang 80% dari puncak tertinggi akan digunakan

untuk membuat *keypoint* baru pada lokasi yang sama dengan orientasi yang berbeda sesuai dengan nilai orientasi pada bin tersebut.

2.3.4 Pembuatan Deskriptor

Setelah didapatkan *keypoint* beserta skala dan orientasinya, perlu untuk diberikan sebuah identitas pada setiap *keypoint*. Pemberian identitas ini berguna untuk mengidentifikasi *keypoint* yang satu dengan yang lainnya, agar dapat ditemukan *keypoint-keypoint* dengan ciri yang mirip. Identifikasi *keypoint* dapat dilakukan dengan membuat sebuah vektor deskriptor. Vektor deskriptor merupakan vektor yang mendeskripsikan daerah di sekitar *keypoint*. Vektor deskriptor pada SIFT berbentuk vektor berjumlah 128 elemen bilangan bulat.

Pembuatan vektor dilakukan dengan mengambil daerah di sekitar *keypoint* dari gambar yang terlebih dahulu dirotasi sesuai dengan orientasi *keypoint*. Ukuran daerah tersebut ditentukan berdasarkan dari skala *keypoint*. Daerah tersebut kemudian dibagi menjadi 4×4 subdaerah. Untuk setiap subdaerah dihitung nilai *magnitude* dan orientasi setiap *pixel*-nya dengan diberi bobot menggunakan *Gaussian Weighting* lalu hasilnya dimasukkan ke dalam histogram dengan 8 bin. Setiap bin dalam histogram mewakili 45 derajat orientasi. Nilai dari setiap bin pada histogram akan menjadi satu elemen pada deskriptor. Terdapat total 16 subdaerah dengan setiap daerah menghasilkan 8 elemen, sehingga didapat total sebanyak $16 \times 8 = 128$ elemen untuk vektor deskriptor.

2.3.5 SIFT di OpenCV

Library OpenCV di Python memiliki modul implementasi SIFT. Modul dapat digunakan untuk melakukan ekstraksi fitur lokal dari gambar dan menghasilkan vektor deskriptor untuk tiap fitur lokal. Untuk melakukan deteksi fitur lokal, pertama perlu untuk dibuat objek SIFT dengan menggunakan fungsi `SIFT_create`. Beberapa parameter dari fungsi `SIFT_create` yang relevan terhadap skripsi ini adalah sebagai berikut:

- `nfeatures`: jumlah fitur yang ingin dihasilkan. Dipilih berdasarkan skor yang didapat dari nilai kontras pada daerah sekitar *keypoint*.
- `nOctaveLayers`: jumlah lapisan untuk tiap oktaf. Ditentukan secara otomatis dari resolusi gambar.
- `sigma`: nilai σ Gaussian yang digunakan pada oktaf pertama.

Objek SIFT tersebut lalu digunakan untuk mendeteksi fitur lokal dengan menggunakan fungsi `detectAndCompute`. Fungsi `detectAndCompute` menerima beberapa parameter berikut:

- `image`: gambar masukkan yang ingin dideteksi *keypoint*-nya. Berbentuk *array* 2 dimensi.
- `mask`: *array* 2 dimensi dengan ukuran yang sama dengan gambar, menentukan *pixel* mana yang digunakan dan yang tidak.

Fungsi `detectAndCompute` tersebut akan menghasilkan sebuah *tuple* yang berisi objek *keypoint* dan *array* berjumlah 128 elemen sebanyak *keypoint* yang terdeteksi. Objek *keypoint* yang dihasilkan memiliki atribut sebagai berikut:

- `pt`: atribut ini memiliki dua buah atribut `x` dan `y` yang menunjukkan koordinat dari *keypoint*.
- `size`: diameter daerah di sekitar *keypoint* yang diperiksa.
- `angle`: orientasi dari *keypoint*.
- `octave`: oktaf di mana *keypoint* tersebut didapat.

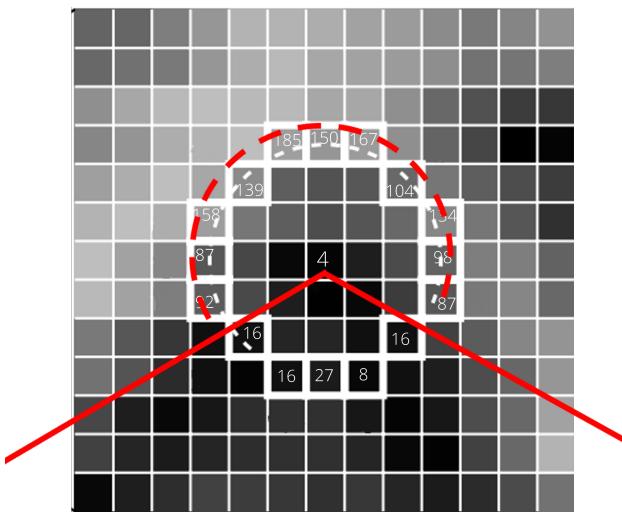
2.4 ORB (Oriented FAST and Rotated BRIEF)

ORB adalah metode pencarian fitur lokal yang dijelaskan pada [2]. ORB dapat menemukan fitur lokal dengan lebih cepat jika dibandingkan dengan SIFT, walaupun fitur lokal yang dihasilkan tidak seakurat yang dihasilkan SIFT. ORB mencari fitur lokal dengan mencari *pixel* yang merupakan sudut perpotongan garis (*Keypoint*). *Keypoint* dalam ORB dicari dengan ide bahwa sebuah *pixel* yang merupakan sudut akan memiliki garis kontinu membentuk lingkaran di sekitarnya dengan

nilai intensitas yang lebih kecil atau lebih besar dari nilai intensitas *pixel* tersebut. Fitur lokal yang dihasilkan ORB bersifat invarian terhadap rotasi, translasi dan invarian sebagian terhadap skala. Fitur lokal ORB juga memiliki sebuah vektor deskriptor yang berbentuk vektor sebanyak 256 elemen bilangan biner. Tahap pencarian fitur lokal pada ORB dibagi menjadi 4 langkah yang dijelaskan pada subbab-subbab berikut.

2.4.1 Pencarian Keypoint

Untuk menentukan apakah sebuah *pixel* dalam gambar merupakan *keypoint*, ORB mengambil nilai *pixel* tersebut dan 16 *pixel* di sekitarnya yang membentuk lingkaran. Dari ke 16 *pixel* tersebut dibandingkan nilai intensitasnya dengan nilai intensitas *pixel* yang berada di tengah (*p*), yaitu *pixel* yang ingin diperiksa apakah merupakan *keypoint*. Sebuah *pixel* merupakan *keypoint* jika dari 16 *pixel* di sekitarnya terdapat setidaknya n *pixel* kontinu yang nilai intensitasnya lebih besar dari nilai $p + t$ atau lebih kecil dari $p - t$. Proses ini diilustrasikan pada Gambar 2.13



Gambar 2.13: Ilustrasi penentuan *keypoint* pada ORB. Setiap kotak menunjukkan sebuah *pixel* pada gambar dan angka di dalamnya merupakan nilai intensitasnya. *Pixel* di tengah gambar (*pixel* bernilai 4) merupakan *pixel* yang akan diperiksa apakah merupakan *keypoint*. *Pixel-pixel* yang ditandai dengan kotak putih merupakan 16 *pixel* yang digunakan untuk memeriksa apakah *pixel* di tengah merupakan *keypoint*.

Pada ilustrasi di Gambar 2.13 *pixel* bernilai 4 yang berada di tengah gambar dapat menjadi *keypoint* tergantung dari parameter n dan t yang digunakan. Di sekitar *pixel* tersebut terdapat 11 *pixel* kontinu yang nilai intensitasnya lebih besar setidaknya 83 satuan dari nilai intensitas *pixel* tersebut.

Keypoint-keypoint yang dihasilkan pada tahap ini belum tentu merupakan sebuah *corner*. Metode yang digunakan ORB akan mendeteksi sebuah titik terang yang dikelilingi lingkaran dengan titik gelap atau sebaliknya menjadi sebuah *keypoint*. Untuk itu perlu untuk disaring untuk mendapatkan hanya *keypoint* yang merupakan *corner*.

Penentuan apakah *keypoint* merupakan *corner* pada ORB dilakukan dengan menggunakan *Harris Corner Measure*. *Harris Corner Measure* akan memberikan sebuah skor untuk sebuah daerah di sekitar *pixel* untuk menunjukkan seberapa daerah tersebut merupakan *corner*.

Harris Corner Measure menentukan apakah sebuah daerah merupakan *corner* dengan menggunakan sebuah *window* yang terpusat pada titik *keypoint*. *Window* tersebut lalu digerakkan ke beberapa arah dan dihitung intensitasnya untuk tiap pergerakan. Daerah tersebut merupakan *corner* jika terdapat perubahan nilai intensitas ke arah manapun *window* tersebut digerakkan.

Teknik *Harris Corner Measure* akan menghasilkan sebuah skor M yang menunjukkan nilai

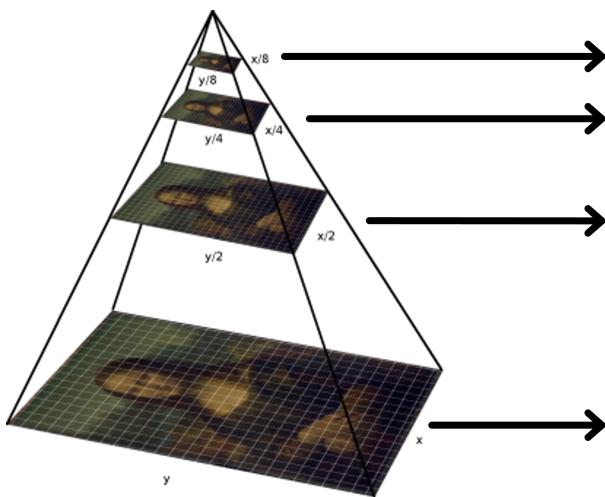
seberapa daerah yang diperiksa merupakan *corner*. ORB akan mengambil N *keypoint* dengan nilai M tertinggi untuk ditentukan sebagai *keypoint*.

2.4.2 Penentuan Skala

Keypoint yang telah dideteksi pada tahap awal perlu untuk dapat terdeteksi juga walaupun ukuran gambar berubah agar sifatnya invariant terhadap skala. *Keypoint* yang invariant terhadap skala adalah *keypoint* dengan ciri yang tetap walaupun dideteksi pada ukuran gambar yang berbeda.

Untuk mencapai sifat ini ORB menggunakan metode *Image Pyramid*. ORB menggunakan *Image Pyramid* dengan cara memperkecil ukuran gambar beberapa kali dan untuk setiap ukuran gambar dilakukan deteksi untuk *keypoint*. Dengan menggunakan cara ini *keypoint* tidak akan benar-benar invariant terhadap perubahan skala, *keypoint* hanya invariant terhadap beberapa skala gambar yang digunakan pada *pyramid*.

Ilustrasi dapat dilihat pada Gambar 2.14. Pada ilustrasi tersebut gambar awal diperkecil beberapa kali dengan membagi panjang dan lebarnya menjadi setengahnya. Untuk setiap ukuran gambar dicari *keypoint-keypoint*-nya.



Gambar 2.14: *Image Pyramid* pada ORB

Tahap ini akan menghasilkan *keypoint-keypoint* dengan skala yang berbeda sesuai dengan ukuran gambar di mana *keypoint* tersebut ditemukan. *Keypoint-keypoint* yang dihasilkan tersebut juga perlu untuk disaring dengan menggunakan *Harris Corner Measure* seperti yang dijelaskan pada subbab sebelumnya.

2.4.3 Penentuan Orientasi

Orientasi *keypoint* pada ORB ditentukan oleh sudut antara sumbu x dan vektor dari *keypoint* menuju titik *Intensity Centroid* dari daerah sekitarnya. *Intensity Centroid* pada sebuah daerah gambar merupakan titik di mana terjadi perubahan nilai intensitas terbesar. Titik *Intensity Centroid* (C) didefinisikan pada Persamaan 2.3.

$$C = \left(\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}} \right) \quad (2.3)$$

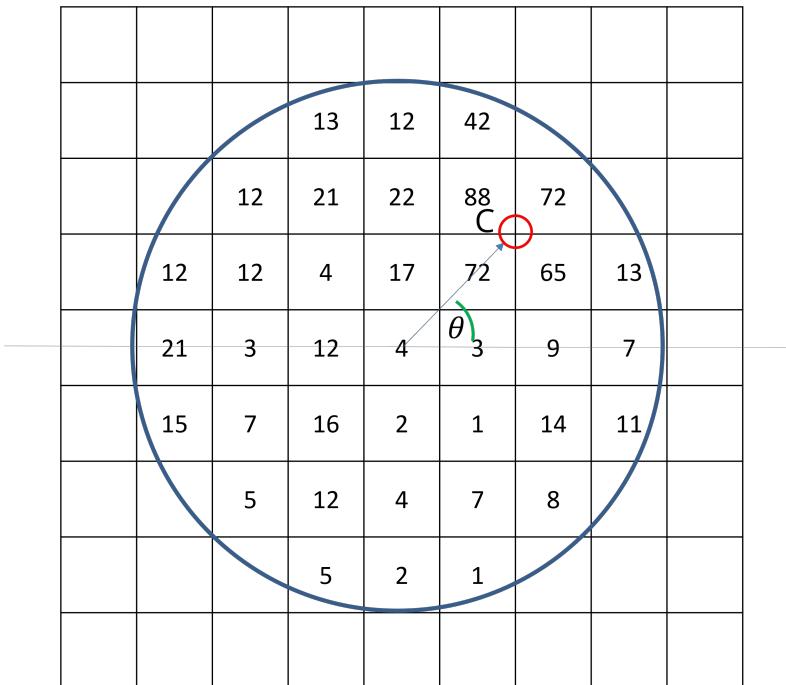
Centroid ditentukan dengan menghitung *moment* pada gambar yang didefinisikan pada Persamaan 2.4.

$$m_{pq} = \sum_{x,y} x^p y^q I(x, y) \quad (2.4)$$

Pada Persamaan 2.3 m_{10} merupakan *moments* gambar dari arah sumbu x . *Moments* dari arah sumbu x merupakan nilai perubahan intensitas yang dihitung dari sumbu tersebut, dihitung dari total seluruh nilai *pixel* dikalikan dengan posisinya pada sumbu x . Sedangkan m_{01} merupakan *moments* dari arah sumbu y yang dihitung dengan cara yang sama. Kedua nilai tersebut— m_{10} dan m_{01} —masing-masing dibagi oleh m_{00} yang didapat dengan menghitung jumlah dari nilai semua *pixel* pada daerah yang diperiksa.

Titik *centroid* dihitung pada daerah yang dikelilingi 16 *pixel* yang digunakan pada tahap Penentuan *Keypoint*. Dari daerah tersebut didapat titik yang merupakan *centroid*. Lalu dibuat garis vektor yang berasal dari titik *keypoint* (titik tengah) menuju titik *centroid*. Orientasi ditentukan dari sudut antara garis lurus sumbu x dengan garis vektor.

Proses penentuan orientasi ini diilustrasikan pada Gambar 2.15. Pada gambar tersebut titik dengan lingkaran hijau merupakan titik ditemukannya *keypoint*. Penentuan orientasi dilakukan dengan mencari titik *centroid* pada daerah yang dikelilingi lingkaran biru. Setelah didapat titik *centroid* (lingkaran merah) maka orientasi merupakan garis lurus dan vektor dari titik *keypoint* ke *centroid*, ditunjukkan oleh θ .



Gambar 2.15: Ilustrasi penentuan orientasi pada ORB.

2.4.4 Pembuatan Deskriptor

Untuk setiap *keypoint* yang dihasilkan perlu untuk dibuat sebuah vektor yang mendeskripsikan daerah di sekitar *keypoint* tersebut. Vektor deskriptor berguna untuk melakukan identifikasi pada *keypoint* tersebut. Pada ORB, vektor deskriptor berbentuk vektor biner berjumlah 256 elemen. Vektor didapat dengan melihat perbandingan nilai intensitas *pixel* pada daerah di sekitar *keypoint*.

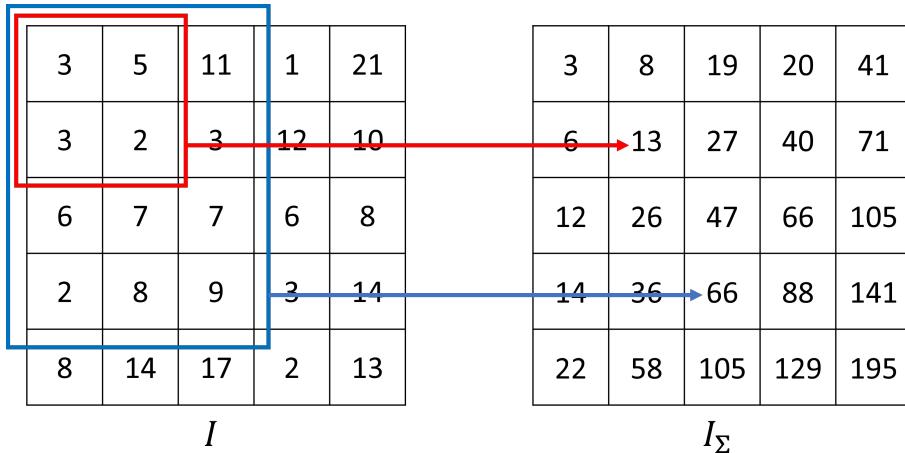
Agar deskriptor bersifat invarian terhadap rotasi gambar terlebih dahulu dirotasi sesuai dengan orientasi yang sudah didapat sebelumnya. Dari gambar yang telah dirotasi diambil daerah berukuran 31×31 untuk dilakukan *binary test*. *Binary test* adalah fungsi yang membandingkan nilai intensitas antar dua titik pada gambar. Fungsi *binary test* τ didefinisikan sebagai berikut:

$$\tau(p; x, y) = \begin{cases} 1 : p(x) < p(y), \\ 0 : p(x) \geq p(y) \end{cases} \quad (2.5)$$

Fungsi tersebut akan menghasilkan nilai 1 atau 0 bergantung pada nilai intensitas dari dua titik yang dibandingkan.

Hasil dari *binary test* yang dilakukan dengan membandingkan nilai dari pasangan-pasangan *pixel* pada gambar akan sangat terpengaruh oleh *noise* yang ada pada gambar. Jika pada saat melakukan *binary test* digunakan *pixel* yang merupakan *noise* maka hasilnya akan tidak representatif terhadap sifat daerah sebenarnya. Untuk menangani masalah ini perlu terlebih dahulu dilakukan *smoothing* pada gambar tersebut. ORB melakukan *smoothing* dengan menggunakan *integral image*.

Integral image adalah sebuah *matrix* 2 dimensi yang dapat digunakan untuk menghitung hasil penjumlahan semua nilai *pixel* di sebuah daerah pada gambar. *Integral Image* dapat mempercepat proses penghitungan total nilai *pixel* pada sebuah daerah dari gambar dengan ukuran berapapun. Nilai sebuah elemen pada koordinat (x, y) di *Integral Image* adalah total penjumlahan semua nilai yang posisi koordinatnya lebih kecil atau sama dengan (x, y) , seperti diilustrasikan pada Gambar 2.16.



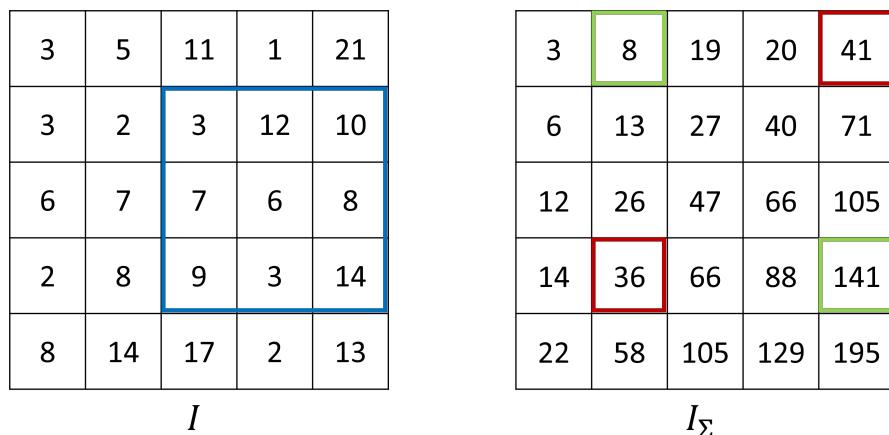
Gambar 2.16: Ilustrasi penghitungan *Integral Image*. *Matrix I* merupakan *matrix* awal dan *Matrix I_{Σ}* merupakan *Integral Image* dari *I*.

Gambar 2.16 menunjukkan cara menghitung nilai elemen pada sebuah koordinat. Kotak pada *Matrix I* merupakan daerah yang dihitung total nilainya untuk mendapatkan nilai pada *Matrix I_{Σ}* yang ditunjuk oleh tanda panah.

Matrix Integral Image yang sudah dihasilkan dapat digunakan untuk menghitung nilai total dari sebuah daerah berukuran berapapun dengan melakukan operasi penjumlahan dan pengurangan pada 4 angka. Sebuah daerah pada Gambar I yang sudah dibuat *Integral Image*-nya (I_{Σ}) yang berada pada persegi dengan titik pojok kiri atas (T_x, T_y) dan pojok kanan bawah (B_x, B_y) dapat dihitung nilai total elemennya dengan persamaan berikut:

$$Su(T_x, T_y, B_x, B_y) = I_{\Sigma}(B_x, B_y) - I_{\Sigma}(B_x, T_y - 1) - I_{\Sigma}(T_x - 1, B_y) + I_{\Sigma}(T_x - 1, T_y - 1) \quad (2.6)$$

Proses penghitungan nilai total sebuah daerah menggunakan *Integral Image* diilustrasikan pada Gambar 2.17. Total nilai elemen dari daerah yang di dalam kotak biru pada Gambar I dapat dihitung dengan menjumlahkan elemen pada Gambar I_{Σ} yang diberi kotak hijau dan menguranginya dengan elemen yang diberi kotak merah. Daerah di dalam kotak biru tersebut memiliki nilai total $141 - 41 - 36 + 8 = 72$.



Gambar 2.17: Penghitungan nilai total sebuah daerah dengan menggunakan *Integral Image*

Smoothing pada ORB dilakukan dengan mengubah *binary test* yang dilakukan. *Binary test* pada ORB dilakukan dengan membandingkan nilai total *pixel* dari dua daerah berukuran 5×5 dalam daerah 31×31 yang digunakan untuk menghitung deskriptor. Penghitungan nilai total pada daerah 5×5 dilakukan dengan menggunakan *Integral Image* untuk mempercepat proses.

Binary test pada ORB dilakukan sebanyak 256 kali untuk menghasilkan 256 elemen untuk vektor. ORB menggunakan 256 pasangan *binary test* yang sudah ditentukan sebelumnya. Pasangan-pasangan tersebut didapat dari eksperimen yang dilakukan pada [2].

2.4.5 ORB di OpenCV

Library OpenCV di Python memiliki implementasi untuk metode ORB. Implementasi ORB ini memiliki kegunaan dan cara penggunaan yang mirip dengan implementasi SIFT. Untuk melakukan deteksi fitur lokal perlu untuk terlebih dahulu dibuat objek ORB dengan fungsi `ORB_create`. Beberapa parameter fungsi `ORB_create` yang relevan terhadap skripsi ini adalah:

- `nfeatures`: jumlah maksimum fitur lokal yang dihasilkan.
- `scaleFactor`: rasio pengurangan resolusi pada setiap level *pyramid*.
- `nlevels`: jumlah tingkatan (*level*) pada *pyramid*.
- `firstLevel`: tingkat dalam *pyramid* sebagai tempat gambar asli.
- `patchSize`: ukuran daerah yang digunakan untuk membuat deskriptor.

Objek ORB tersebut lalu dapat digunakan untuk mendeteksi fitur lokal dalam gambar dengan menggunakan fungsi `detectAndCompute`. Fungsi `detectAndCompute` menerima beberapa parameter berikut:

- `image`: gambar masukkan yang ingin dideteksi *keypoint*-nya. Berbentuk *array* 2 dimensi.
- `mask`: *array* 2 dimensi dengan ukuran yang sama dengan gambar, menentukan *pixel* mana yang digunakan dan yang tidak.

Fungsi tersebut menghasilkan *tuple* berisi objek *keypoint* dan array vektor deskriptor untuk tiap objek *keypoint*. Setiap vektor merupakan vektor berisi bilangan biner berjumlah 256 elemen. Objek *keypoint* memiliki atribut sebagai berikut:

- `pt`: atribut ini memiliki dua buah atribut `x` dan `y` yang menunjukkan koordinat dari *keypoint*.
- `size`: diameter daerah di sekitar *keypoint* yang diperiksa.
- `angle`: orientasi dari *keypoint*.
- `octave`: lapisan *pyramid* di mana *keypoint* tersebut didapatkan.

2.5 BSIS (Best Score Increasing Subsequence)

Pada tahapan OIR (lihat 2.2) sebuah pasangan fitur lokal perlu untuk memiliki sifat yang mirip (dilihat dari vektor deskriptornya) dan juga konsisten secara geometris. Pasangan yang konsisten

secara geometris adalah pasangan yang memiliki posisi spasial yang konsisten terhadap pasangan lain di sekitarnya (contoh dapat dilihat pada Gambar 2.4 di ??). BSIS [?] adalah salah satu metode yang dapat digunakan untuk menentukan apakah pasangan fitur lokal bersifat konsisten secara geometris.

BSIS merupakan modifikasi metode WLIS [4] yang merupakan implementasi dari OIR. Modifikasi terdapat pada tahap *Pairing*, *Verification*, dan *Scoring*.

Tahapan dari BSIS dilakukan setelah fitur lokal telah diekstrak dari gambar *query* dan gambar *training* (gambar yang ada di *dataset*). Tahapan BSIS ini dilakukan untuk setiap pemasangan gambar *query* dan gambar *training*. Pasangan gambar dengan skor tertinggi ditentukan sebagai pasangan yang benar dari gambar *query*. Rincian tahapan dijabarkan pada subbab-subbab di bawah ini.

2.5.1 Pairing

Pada tahap awal setiap fitur lokal pada gambar *query* akan dicari N fitur lokal dari gambar *training* yang nilai kemiripan vektor deskriptornya paling tinggi. Pencarian N pasangan paling mirip dilakukan dengan menggunakan KD-Tree (lihat 2.6) untuk mempercepat waktu komputasi. Pasangan-pasangan ini akan memiliki sebuah skor kemiripan yang dihitung dari vektor deskriptor kedua fitur lokal. Tidak semua dari N pasangan paling mirip akan digunakan dalam proses BSIS.

Untuk setiap fitur lokal, BSIS hanya akan mengambil beberapa pasangan yang benar-benar mirip atau paling berpeluang menjadi pasangan yang benar. Pasangan yang benar-benar mirip pada BSIS adalah pasangan yang nilai kemiripannya tinggi dan memiliki perbedaan yang cukup jauh dengan nilai kemiripan pasangan lain.

BSIS menganggap pasangan-pasangan yang sangat mirip adalah pasangan yang nilai kemiripannya tinggi dan merupakan *outlier*. Penentuan *outlier* dilakukan dengan menggunakan *mean* dan deviasi standar. Penggunaan *mean* dan deviasi standar untuk menentukan *outlier* karena BSIS mengasumsikan bahwa nilai kemiripan pasangan-pasangan dari sebuah fitur lokal terdistribusi secara normal.

BSIS menentukan *outlier* seperti pada Persamaan 2.7. Untuk sebuah fitur lokal gambar *training* P_Q yang dipasangkan dengan gambar *training* P_T , pasangan tersebut hanya akan digunakan jika nilai kemiripannya lebih dari *mean* (m) ditambah konstanta K dikali deviasi standar (σ). *Mean* merupakan rata-rata dari nilai kemiripan dari N pasangan P_Q dan σ merupakan nilai deviasi standarnya sedangkan untuk konstanta digunakan nilai $K = 4$.

$$\text{similarity}(P_Q, P_T) > m + (K \times \sigma) \quad (2.7)$$

Pasangan-pasangan yang telah dipilih (merupakan *outlier*) tersebut lalu diberi bobot berdasarkan nilai kemiripannya. Bobot sebuah pasangan didapatkan dengan menghitung seberapa jauh nilai kemiripan pasangan tersebut terhadap rata-rata dengan memperhatikan penyebaran nilainya. Bobot (P_w) untuk sebuah pasangan dari fitur lokal P_Q dan P_T didefinisikan pada Persamaan 2.8.

$$P_w(P_Q, P_T) = \left(\frac{\text{similarity}(P_Q, P_T) - m}{\sigma} \right)^2 \quad (2.8)$$

P_Q merupakan fitur lokal gambar *query* dan P_T merupakan fitur lokal dari gambar *training*. Nilai P_w yang semakin tinggi menunjukkan bahwa pasangan tersebut semakin berpotensi untuk menjadi pasangan yang tepat.

2.5.2 Verification

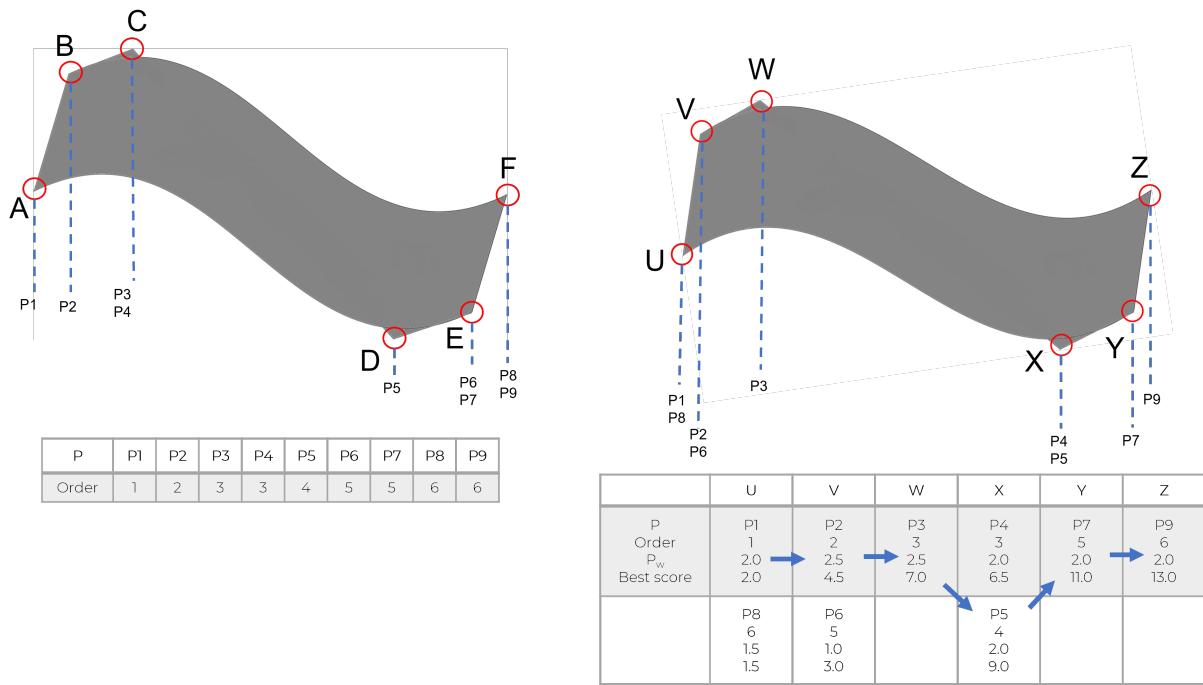
Pada tahap *Pairing* telah didapat beberapa pasangan antara fitur lokal gambar *query* dan gambar *training* yang paling berpotensi merupakan pasangan yang benar. Dari tahap sebelumnya setiap fitur lokal pada gambar *query* dapat dipasangkan dengan lebih dari 1 fitur lokal pada gambar *training*. Pasangan-pasangan tersebut akan diperiksa pada tahap ini untuk mencari pasangan mana

saja yang konsisten secara geometris. Setiap fitur lokal baik dari gambar *query* maupun *training* hanya akan berada dalam satu pasangan setelah dilakukan verifikasi.

Pasangan yang konsisten secara geometris adalah pasangan-pasangan yang memiliki posisi spasial yang konsisten terhadap pasangan fitur lokal lain di sekitarnya. Sifat konsisten ini ditunjukkan oleh posisi fitur-fitur lokal pada gambar *query* dan gambar *training*. Sebagai contoh, dua buah fitur lokal dari gambar *query* PQ_1 dan PQ_2 dipasangkan dengan fitur lokal PT_1 dan PT_2 dari gambar *training*. Jika PQ_1 berada di sebelah kiri PQ_2 (PQ_1 muncul lebih dulu dalam proyeksi sumbu x) maka PT_1 juga harus berada di sebelah kiri PT_2 untuk agar kedua pasangan tersebut dikatakan konsisten secara geometris.

Tahap *Verification* pada BSIS bertujuan untuk menemukan pasangan-pasangan fitur lokal yang konsisten secara geometris sesuai dengan penjelasan di atas. BSIS melakukan verifikasi dengan memproyeksikan seluruh fitur lokal dari gambar *query* dan *training* pada sumbu x dan mencari pasangan yang konsisten dan memiliki skor yang paling tinggi. Langkah-langkah verifikasi pada BSIS secara rinci adalah sebagai berikut:

1. Ambil semua fitur lokal dari gambar *query* dan proyeksikan pada sumbu x . Setiap fitur lokal akan diberi sebuah *order* sesuai dengan urutan kemunculannya.
2. Ambil semua fitur lokal dari gambar *training* dan proyeksikan pada sumbu x . Urutkan fitur lokal tersebut sesuai dengan urutan kemunculannya.
3. Buat tabel untuk fitur lokal pada gambar *training* dengan setiap kolom merupakan fitur lokal yang diurutkan berdasarkan kemunculannya pada sumbu x ;
4. Isi setiap kolom dengan pasangan-pasangan fitur lokal yang melibatkan fitur lokal kolom tersebut. Untuk tiap isi kolom, catat nomor *order* dari fitur lokal gambar *query* pasangan tersebut dan bobot (P_w) pasangan tersebut. Kolom pada tabel merupakan fitur lokal pada gambar *training* dan tiap barisnya merupakan fitur lokal pada gambar *query*.



Gambar 2.18: Tahapan verifikasi pada BSIS.

5. BSIS hanya mengambil satu pasangan dari setiap fitur lokal di gambar *query* dengan bobot pasangan yang tertinggi. Untuk itu buat sebuah *subsequence* dengan mengambil satu baris dari tiap kolom berurutan mulai dari kolom paling kiri. *Order* menunjukkan urutan kemunculan pasangan tersebut pada gambar *query*, untuk itu *order* dari *subsequence* perlu untuk terus bertambah tiap elemen. *Subsequence* yang dicari adalah yang memiliki total nilai bobot

pasangan paling banyak.

6. Verifikasi juga perlu dilakukan untuk sumbu y . Fitur lokal yang telah dipilih pada langkah sebelumnya (fitur lokal yang masuk dalam *subsequence*) akan dilakukan verifikasi ulang menggunakan urutannya pada sumbu y .

Tahapan di atas dilakukan beberapa kali pada orientasi fitur lokal gambar *training* yang berbeda untuk mencari orientasi gambar yang memberikan total skor pasangan paling tinggi.

2.5.3 Scoring

Setelah dipilih dan didapat pasangan fitur lokal yang konsisten secara geometris akan dihitung nilai kemiripan gambar masukkan dengan gambar *training* yang terpilih. Tingkat kemiripan ditentukan dengan menghitung total P_w dari pasangan yang terpilih pada tahap *Verification*. Nilai kemiripan ini dapat digunakan untuk menentukan apakah pasangan gambar cukup mirip untuk menjadi pasangan yang benar.

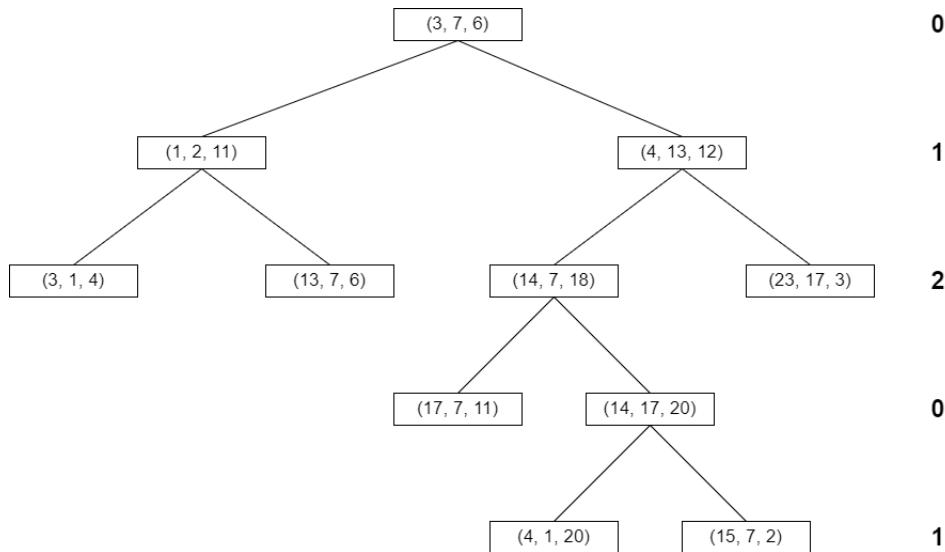
2.6 KD-Tree

KD-Tree [5] merupakan sebuah struktur data yang dapat digunakan untuk mencari vektor yang paling mirip dengan waktu proses yang relatif lebih singkat. Struktur KD-Tree digunakan untuk mencari sejumlah pasangan fitur lokal yang sifatnya paling mirip pada tahapan *Pairing* di BSIS 2.5.

Secara umum KD-Tree merupakan sebuah pohon pencarian biner di mana setiap *node*-nya merupakan sebuah vektor berjumlah k -elemen. Setiap *level* pada pohon membandingkan elemen vektor yang berbeda. *Level* pertama pohon akan menggunakan elemen pertama dari vektor sebagai pembanding, elemen kedua pada *level* kedua dan seterusnya. Saat *level* dari pohon sudah mencapai $k - 1$, maka akan kembali ke elemen pertama. Elemen yang diperiksa pada *level* l di KD-Tree dengan jumlah k elemen didefinisikan pada Persamaan 2.9.

$$\text{target}(l, k) = l \bmod k \quad (2.9)$$

Gambar 2.19 menunjukkan contoh pohon KD-Tree dengan jumlah 3 elemen.



Gambar 2.19: Contoh pohon struktur KD-Tree. Angka di sebelah kanan menunjukkan elemen vektor yang diperiksa pada level tersebut.

KD-Tree membuat pohon dari sekumpulan vektor di *dataset*. Vektor pertama yang masuk akan menjadi *root* dari pohon. Vektor selanjutnya akan dibandingkan nilai elemen pertamanya dengan

elemen pertama dari *root*, jika nilainya lebih kecil maka vektor tersebut akan menjadi *child* sebelah kiri dan menjadi *child* sebelah kanan jika sebaliknya. Proses berlanjut untuk vektor selanjutnya, elemen di vektor yang masuk akan dibandingkan dengan vektor pada *node* pohon sesuai dengan *level* pohon saat itu.

Setelah terbentuk pohon dari semua vektor pada *dataset*, pohon tersebut dapat digunakan untuk pencarian tetangga terdekat (*nearest neighbor search*) dari sebuah vektor masukkan baru. Tahapan untuk mencari tetangga terdekat dari vektor Q pada sebuah pohon KD-Tree K adalah sebagai berikut:

1. Jelajahi pohon K seperti biasa, dimulai dari *node root*. Bandingkan elemen pada *node* tersebut dengan elemen pada Q .
2. Untuk semua *node* yang diperiksa hitung jaraknya ke Q . Simpan jarak tersebut ke dalam variabel *closest*. Jika pada *node* selanjutnya didapat jarak yang lebih kecil maka ganti *closest* dengan jarak tersebut.
3. Jelajahi sampai sudah mencapai *node* yang merupakan *leaf*. Pada titik ini masih mungkin ada *node* pada bagian pohon yang tidak dijelajahi yang jaraknya lebih kecil dari *closest*.
4. Untuk semua *node* bukan *leaf* yang dilewati hitung jarak elemen ke- n dari Q ke elemen ke- n *node* tersebut. Jika jaraknya lebih kecil dari *closest* maka jelajahi *child* dari *node* tersebut yang sebelumnya tidak dipilih.

2.7 Clustering

Clustering adalah salah satu teknik pengolahan data dalam *machine learning*. Pada dasarnya *clustering* akan membagi objek-objek pada *dataset* menjadi beberapa kelompok (*cluster*) berdasarkan sifatnya. Objek-objek yang memiliki sifat mirip akan masuk kedalam satu kelompok. Sebuah pembagian *cluster* yang baik adalah di mana setiap objek dalam *cluster* memiliki sifat yang mirip dan antar *cluster* memiliki sifat yang sangat berbeda.

Beberapa contoh metode *clustering* yang ada adalah *Agglomerative Clustering* dan DBSCAN. Kedua teknik tersebut menggunakan metode dasar yang berbeda dan akan menghasilkan *cluster* dengan ciri yang berbeda juga. Kedua metode tersebut dijelaskan pada dua subbab berikut. Penjelasan berdasarkan pada [6]

2.7.1 Agglomerative Clustering

Teknik *Agglomerative Clustering* adalah salah satu teknik *clustering* yang berbasis hierarki (*hierarchical*). Teknik ini membentuk *cluster* dengan menyusun hierarki atau tingkatan antar objek berdasarkan kemiripannya. Pasangan objek dengan jarak yang paling kecil akan berada di tingkatan terendah, jarak terkecil selanjutnya akan berada di tingkat atasnya, dan seterusnya hingga semua objek telah tercakup.

Agglomerative Clustering adalah teknik *clustering* berbasis hierarki yang menggunakan metode *bottom-up*. Metode *bottom-up* memulai tahapan dengan membentuk *cluster-cluster* kecil dan kemudian menggabungkan *cluster* kecil tersebut menjadi *cluster* yang lebih besar.

Tahapan *Agglomerative Clustering* dimulai dengan terlebih dahulu menghitung jarak antar tiap objek. Setelah itu ambil pasangan dengan jarak paling kecil dan gabungkan menjadi satu *cluster* dan lakukan juga untuk jarak paling kecil berikutnya. Jika jarak terkecil yang ada adalah antara objek yang sudah berada di dalam *cluster*, maka gabungkan kedua *cluster* tersebut menjadi *cluster* yang lebih besar.

Langkah-langkah pada tahapan tersebut dilakukan hingga semua objek sudah berada di dalam satu *cluster* yang sama. *Cluster* besar tersebut lalu dapat dibagi berdasarkan dari jaraknya dengan menggunakan *threshold* yang dapat ditentukan secara manual.

Algoritma *clustering Agglomerative* dapat dibagi menjadi beberapa tipe dibedakan dari cara menghitung jarak antar *cluster-cluster*-nya. Teknik *single-linkage* menghitung jarak antar *cluster*

dengan jarak objek terdekat antar kedua *cluster* tersebut. Sedangkan teknik *complete-linkage* menghitung jarak dengan jarak objek terjauh antar dua *cluster*.

Proses penggabungan objek menjadi *cluster* pada *Agglomerative Clustering* ditunjukkan pada [Pseudocode 1](#).

Pseudocode 1: Agglomerative Clustering

Input:

- D : *dataset* berisi n buah objek
- t : *threshold* untuk menghentikan penggabungan

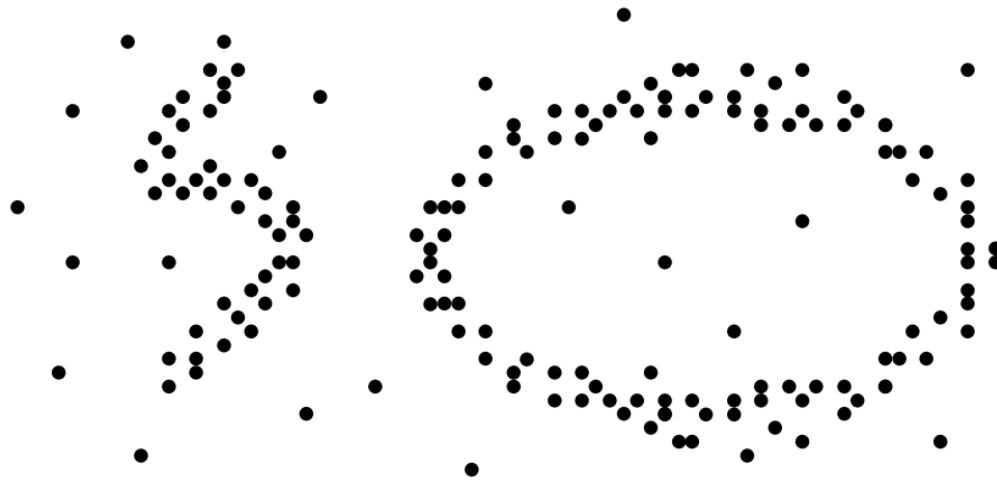
Output: set berisi *cluster-cluster*

- 1: Buat semua objek dalam D menjadi *cluster* dengan 1 anggota
 - 2: Hitung jarak untuk tiap *cluster*
 - 3: **while** jumlah *cluster* dalam $D > 1$ **do**
 - 4: Ambil jarak dari *cluster* r dan s , di mana r dan s adalah dua cluster dalam D dengan jarak terkecil.
 - 5: Gabungkan r dan s menjadi satu *cluster* t .
 - 6: Hitung jarak t ke semua *cluster* lain, sesuai dengan metode *linkage* yang digunakan.
 - 7: **end while**
-

2.7.2 DBSCAN

DBSCAN atau *Density-Based Spatial Clustering of Applications with Noise* adalah salah satu metode *clustering* yang berbasis kepadatan (*density based*). Pada DBSCAN sebuah *cluster* merupakan sebuah daerah *dense* yang dipisahkan oleh daerah *sparse*. Daerah *dense* adalah sebuah kumpulan yang terdiri dari beberapa objek, di mana objek-objek tersebut memiliki ciri yang mirip. Sedangkan daerah *sparse* adalah daerah di mana hanya terdapat sedikit objek dengan sifat yang saling mirip.

Cluster yang dihasilkan oleh DBSCAN tidak selalu berbentuk *circular* (contoh seperti pada [Gambar 2.20](#)) dan objek-objek yang berada dalam satu *cluster* tidak selalu merupakan data yang mirip. Selama sebuah daerah *dense* memiliki daerah *dense* lain di dekatnya, kedua daerah tersebut akan terus bergabung menjadi satu daerah *dense* yang lebih besar. Untuk sebuah daerah *dense* D dapat bergabung ke daerah lain yang terdiri dari lebih dari satu daerah *dense* C , daerah *dense* D hanya perlu untuk dekat dengan salah satu daerah *dense* dari C . Karena bentuknya yang tidak *circular* dan tidak tentu miripnya objek dalam satu *cluster*, maka *cluster* tidak dapat direpresentasikan dengan sebuah titik tengah atau *centroid*.



Gambar 2.20: Contoh *cluster* dengan bentuk non-*circular*. Objek-objek yang tersebar seperti ini dapat tergabung menjadi *cluster* dengan menggunakan DBSCAN.

Penyusunan *cluster* pada DBSCAN dimulai dengan mencari objek yang merupakan *core object*. *Core object* merupakan objek yang memiliki setidaknya $MinPts$ objek lain pada radius ϵ yang berpusat pada objek tersebut. $MinPts$ dan ϵ adalah parameter yang ditentukan secara manual. Nilai $MinPts$ dan ϵ akan memengaruhi hasil *cluster* yang dihasilkan.

Setiap objek yang merupakan *core object* beserta anggotanya (objek lain pada radius ϵ) akan menjadi satu *cluster*. Jika dalam radius ϵ objek tersebut terdapat objek lain yang juga merupakan *core object*, maka akan digabungkan menjadi satu *cluster*. *Core object* yang saling berdekatan ini akan terus digabungkan menjadi *cluster* yang besar.

Proses pencarian *cluster* pada DBSCAN dimulai dengan pertama menandai semua objek pada dataset, D dengan *unvisited*. Dari D diambil sebuah objek, p secara acak dan ditandai sebagai *visited*. Jika p bukan merupakan *core object* maka objek tersebut merupakan *noise*. Sebaliknya jika p adalah *core object* maka p akan dimasukkan ke *cluster C* dan semua objek pada daerah ϵ -nya dimasukkan ke dalam set N . Objek-objek pada N akan diiterasi dan dimasukkan ke *cluster C*. Jika ditemukan objek yang merupakan *core object* maka semua objek lain pada daerah ϵ objek tersebut akan dimasukkan juga ke N . Proses berlanjut sampai tidak ada lagi objek di N . Objek-objek yang sudah diperiksa tersebut akan ditandai sebagai *visited*. Setelah itu akan diambil lagi sebuah objek secara acak yang masih bertanda *unvisited* dan proses diulang untuk membentuk *cluster* baru.

Proses didefinisikan pada *Pseudocode 2* berikut:

Pseudocode 2: DBSCAN**Input:**

- D : dataset berisi n buah objek
- ϵ : parameter radius untuk menghitung daerah
- $MinPts$: batas kepadatan suatu daerah.

Output: set berisi *cluster-cluster*

```
1: tandai semua objek sebagai unvisited
2: while masih terdapat objek yang unvisited do
3:   pilih objek  $p$  secara acak
4:   tandai  $p$  sebagai visited
5:   if dalam radius  $\epsilon$  dari  $p$  terdapat setidaknya  $MinPts$  objek then
6:     buat cluster baru  $C$ , masukkan  $p$  ke dalam  $C$ 
7:     buat variabel  $N$ , berisi semua objek dalam radius  $\epsilon$  dari  $p$ 
8:     for semua objek  $p'$  di  $N$  do
9:       if objek  $p'$  unvisited then
10:        tandai  $p'$  sebagai visited
11:        if dalam radius  $\epsilon$  dari  $p'$  terdapat setidaknya  $MinPts$  objek then
12:          masukkan semua objek tersebut ke dalam  $N$ 
13:        end if
14:        if  $p'$  bukan anggota dari cluster manapun then
15:          masukkan  $p'$  ke dalam  $C$ 
16:        end if
17:      end if
18:    end for
19:    output  $C$ 
20:  else
21:    tandai  $p$  sebagai noise
22:  end if
23: end while
```

DAFTAR REFERENSI

- [1] Lowe, G. (2004) Sift-the scale invariant feature transform. *Int. J.*, **2**, 2.
- [2] Rublee, E., Rabaud, V., Konolige, K., dan Bradski, G. (2011) Orb: An efficient alternative to sift or surf. *2011 International conference on computer vision*, pp. 2564–2571. Ieee.
- [3] Kusuma, G. P., Harjono, K. D., dan Putra, M. T. D. (2019) Geometric verification method of best score increasing subsequence. *IEEE* , ?
- [4] Kusuma, G. P., Szabo, A., Yiqun, L., dan Lee, J. A. (2012) Appearance-based object recognition using weighted longest increasing subsequence. *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 3668–3671. IEEE.
- [5] Bentley, J. L. (1975) Multidimensional binary search trees used for associative searching. *Communications of the ACM*, **18**, 509–517.
- [6] Han, J., Pei, J., dan Kamber, M. (2011) *Data mining: concepts and techniques*. Elsevier.

LAMPIRAN A

KODE PROGRAM

Kode A.1: MyCode.c

```
1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[1] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = ( --aaa + &daa ) / ( bbb++ - ccc % 2 );
14             strcpy(a,"hello_$.@");
15     }
16     count = ~mask | 0x00FF00AA;
17 }
18
19 // Fonts for Displaying Program Code in LATEX
20 // Adrian P. Robson, nepswb.co.uk
21 // 8 October 2012
22 // http://nepswb.co.uk/docs/progfonts.pdf
```

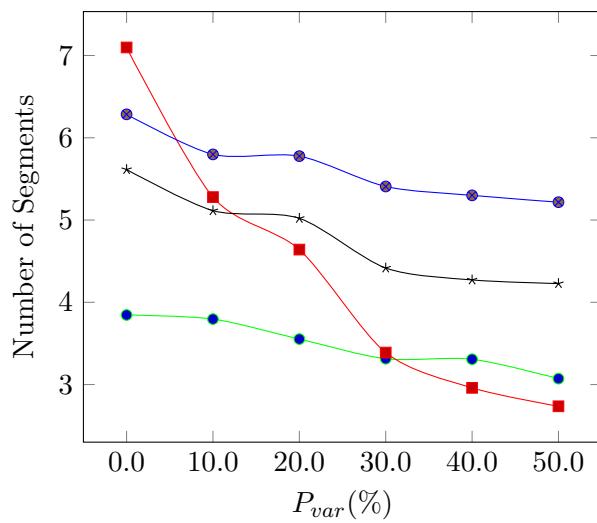
Kode A.2: MyCode.java

```
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the furthest edge
9     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID;           //store the ID of all vertices
12    protected ArrayList<Double> closeDist;          //store the distance of all vertices
13    protected int totaltrj;                          //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i <totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35}
36}
```

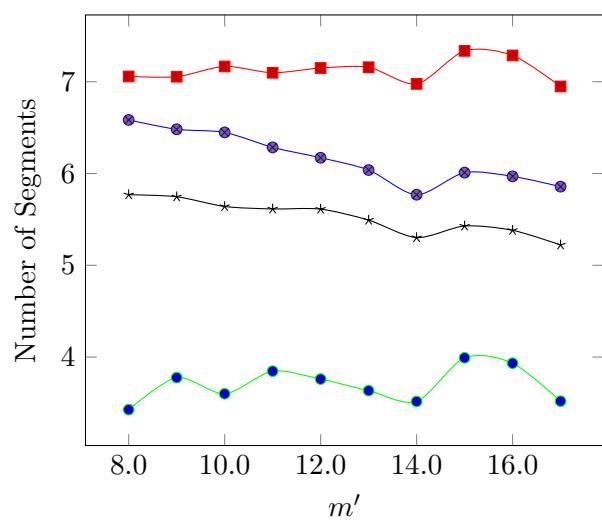

LAMPIRAN B

HASIL EKSPERIMENT

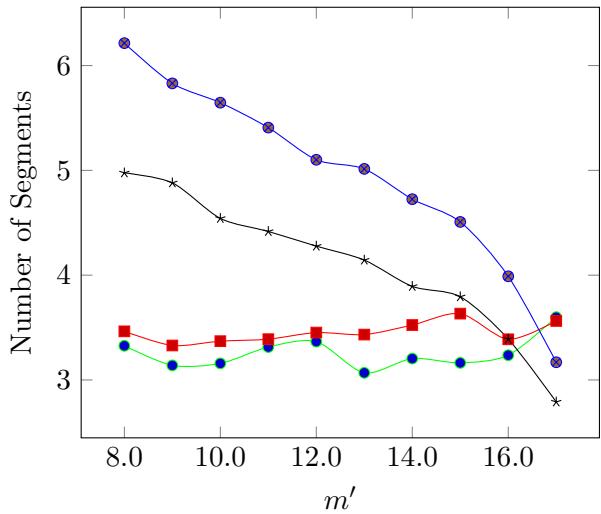
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



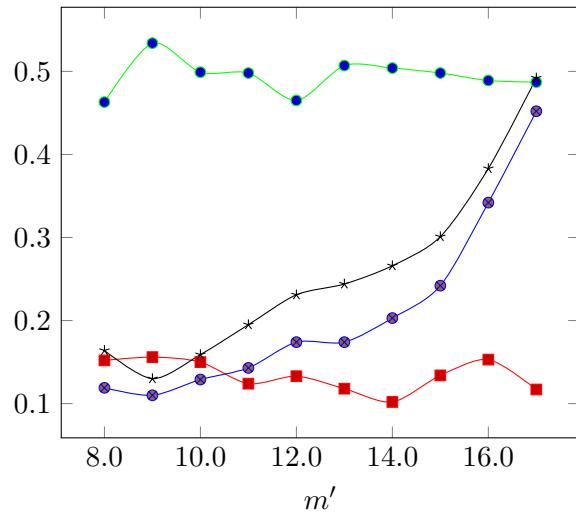
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4