BAB 2
LANDASAN TEORI

2.1 Teori-teori Umum

Teori-teori umum yang berkaitan dengan topik penelitian ini adalah tentang *Computer Vision*, Teori dasar citra digital, Pengolahan citra digital, *Biometrik,* Ruas jari tangan, OOP dan lain-lain.

2.1.1 *Computer Vision*

Computer Vision merupakan bidang yang berkembang pesat dikhususkan untuk menganalisa, memodifikasi, dan pemahaman gambar tingkat tinggi. Tujuannya adalah untuk menentukan apa yang terjadi di depan kamera dan menggunakan pemahaman tersebut untuk mengendalikan komputer, sistem robot, atau untuk memberikan citra baru yang lebih informatif dari pada gambar kamera asli. Aplikasi area untuk komputer visi teknologi termasuk video surveillance, biometrik, otomotif, fotografi, produksi film, pencarian web, obat-obatan, game augmented reality, antarmuka pengguna yang baru, dan banyak lagi [7].

2.1.2 Teori Dasar Citra Digital

Dalam bidang pengolahan citra (image processing), citra yang diolah adalah citra digital, yaitu citra kontinu yang telah diubah ke dalam bentuk diskrit, baik koordinat ruangnya maupun intensitas (kecerahan) cahayanya melalui proses sampling dan kuantisasi [8]. Citra dapat diartikan sebagai suatu fungsi kontinu dari intensitas cahaya (x,y) dalam bidang dua dimensi, dengan (x,y) menyatakan suatu koordinat spasial dan nilai f pada setiap titik (x,y) menyatakan intensitas atau tingkat kecerahan atau derajat keabuan (brightness/gray level).

Piksel (0,0) terletak pada sudut kiri atas pada citra, indeks x bergerak ke kanan dan indeks y bergerak ke bawah. Secara matematis persamaan untuk fungsi intensitas *f(x,y)* adalah:

$$0 \leq f(x,y) < \infty \tag{2.1}$$

Misalkan f merupakan sebuah citra digital 2 dimensi berukuran *MxN*. Maka representasi f dalam sebuah matriks dapat dilihat pada gambar berikut :

$$f(x,y) = \begin{matrix} f(0,0) & f\ 0,1 & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ . & . & . & . \\ f(M-0) & f(M-1) & \dots & f(M-1,N-1) \end{matrix}$$

Gambar 2.1 Matriks Citra Digital

2.1.2.1 Citra RGB

RGB merupakan model warna yang terdiri dari merah (*Red*), hijau (*Green*), dan biru (*Blue*) yang digabungkan dan membentuk suatu susunan warna yang luas. Setiap warna dasar memiliki rentang nilai antara 0 smpai 255 [9].

2.1.2.2 Citra *Grayscale*

Grayscale adalah warna-warna piksel yang berada dalam rentang gradasi warna hitam dan putih. Intensitas *grayscale* biasanya disimpan sebagai data citra 8 bit atau 256 intensitas warna *gray* dari nilai 0 (hitam) dan 255 (putih) [9]. Citra abu-abu dapat dihasilkan dari citra warna RGB dengan mengalikan ketiga komponen warna pokok merah, hijau, dan biru dengan suatu koefisien yang jumlahnya satu (a+b+c = 1).

$$Y = a \times R + b \times G + c \times B$$

$$Gray[x,y] = 0.3*R + 0.59*G + 0.11*B = ... \quad (2.2)$$

2.1.2.3 Citra Biner

Citra biner diperoleh melalui proses pemisahan piksel-piksel berdasarkan derajat keabuan yang dimilikinya. Piksel yang memiliki derajat keabuan lebih kecil dari nilai batas (*threshold*) yang ditentukan akan diberikan nilai nol (hitam). Sedangkan piksel yang memiliki derajat keabuan lebih besar dari nilai *threshold* akan diubah menjadi satu (putih) [9].

2.1.3 Pengolahan Citra Digital

Pengolahan citra digital bertujuan untuk memanipulasi dan menganalisis citra dengan bantuan komputer. Pengolahan citra digital dapat dikelompokkan dalam dua jenis kegiatan:

1) Memperbaiki kualitas suatu gambar, sehingga dapat lebih mudah diinterpretasi oleh mata manusia.
2) Mengolah informasi yang terdapat pada suatu gambar untuk keperluan pengenalan objek secara otomatis.

Secara umum tahapan pengolahan citra digital meliputi akusisi citra, peningkatan kualitas citra, segmentasi citra, representasi dan uraian, pengenalan dan interpretasi [9].

2.1.3.1 Akusisi Citra

Pengambilan data dapat dilakukan dengan menggunakan berbagai media seperti kamera analog, kamera digital, *handycam*, *scanner*, *optical reader* dan sebagainya.

2.1.3.2 Peningkatan Kualitas Citra

Pada tahap ini dikenal dengan *pre-processing* dimana dengan meningkatkan kualitas citra dapat meningkatkan kemungkinan dalam keberhasilan pada tahap pengolahan citra digital berikutnya.

2.1.3.3 Pemotongan Citra

Pemotongan citra (*cropping*) ialah mengambil sebagian dari citra yang akan diproses dengan membuang piksel yang tidak diperlukan.

2.1.3.4 *Filter*

Untuk mengurangi pengaruh derau (*noise*) pada citra maka dilakukan penapisan.

2.1.4 Biometrik

Biometrik merupakan suatu ilmu yang menentukan identifikasi seseorang berdasarkan fisik, kimiawi atau tingkah laku suatu individu. Secara harfiah, biometrik atau *biometrics* berasal dari kata *bio* dan *metrics*. *Bio* berarti sesuatu yang hidup, dan *metrics* berarti mengukur. Biometrik berarti mengukur karakteristik pembeda (*distinguishing traits*) pada badan atau perilaku seseorang yang digunakan untuk melakukan pengenalan secara otomatis terhadap identitas orang tersebut, dengan membandingkannya dengan karakteristik yang sebelumnya telah disimpan dalam suatu *database*.

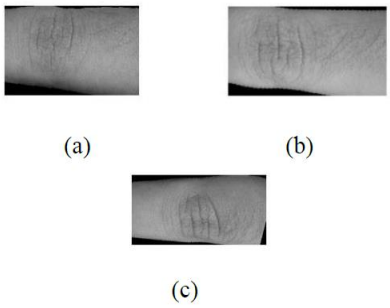
Tidak semua bagian tubuh atau perilaku seseorang dapat digunakan untuk biometrik. Ada tujuh faktor yang menentukan kesesuaian fisik atau tingkah laku dalam penerapan biometrik [10], yaitu :

1) *Universality* (universal), yaitu karakteristik yang dipilih harus dimiliki oleh setiap orang.
2) *Uniqueness* (keunikan), yaitu karakteristik yang dipilih memiliki sifat yang dapat membedakan antar individu.
3) *Permanence* (permanen), yaitu karakteristik yang dipilih tidak cepat berubah dalam periode waktu yang lama.
4) *Measurabillity* (ukuran), yaitu karakteristik yang dipilih mudah untuk diperoleh dan mudah untuk di ukur.
5) *Performance* (Unjuk kerja), yaitu karakteristik yang dipilih dapat memberikan unjuk kerja yang bagus baik dari segi akurasi maupun kecepatan.
6) *Acceptability* (dapat diterima), yaitu setiap orang dapat menerima karakteristik yang digunakan oleh sistem biometrika.
7) *Circumvention* (tidak mudah dikelabui), yaitu karakteristik yang dipilih tidak mudah ditiru dengan berbagai macam bentuk.

Masing-masing biometrika memiliki kelebihan dan kekurangan, tergantung bidang penerapannya.

2.1.5 Ruas Jari Tangan

Permukaan ruas jari tangan dikenal sebagai permukaan punggung jari tangan, pada setiap manusia mempunyai ruas jari tangan dengan pola yang berbeda-beda. Pembentukan pola dari ruas jari tangan manusia sangat unik sejak manusia tersebut dilahirkan dan stabil terhadap perubahan kondisi fisik. Anatomi jari-jari kita memungkinkan ruas jari tangan untuk menekuk ke depan dan menahan gerak mundur. Hal ini menyebabkan pola kulit pada jari-buku jari tangan sangat kaya akan tekstur karena lipatan kulit, dan karenanya, dapat dianggap sebagai identifier biometrik khas dan memiliki tingkat kemanan yang tinggi jika digunakan dalam sistem autentikasi, keamanan dan lain-lain, walaupun belum banyak menarik perhatian para peneliti. Selain kaya fitur tekstur, ruas jari tangan juga mudah diakses, invarian terhadap emosi dan aspek perilaku lainnya seperti kelelahan, fitur yang stabil dan penerimaan *user* sangat tinggi karena, tidak seperti sidik jari tangan, tidak ada stigma potensi investigasi kriminal yang terkait dengan pendekatan ini [11]. Ruas jari tangan yang digunakan diambil dari jari telunjuk, tengah dan manis seperti pada Gambar 2.2 .



Gambar 2.2 Ruas punggung jari tangan

2.1.6 Basis Data

Sistem basis data adalah sistem terkomputerisasi yang tujuan utamanya adalah memelihara data yang sudah diolah atau informasi dan membuat informasi tersedia saat dibutuhkan [12]. Pada intinya basis data adalah media untuk menyimpan data agar dapat diakses dengan mudah dan cepat.

Sistem informasi tidak dapat dipisahkan dengan kebutuhan akan basis data apa pun bentuknya, entah berupa *file* teks ataupun *Database Management System* (DMBS). Kebutuhan basis data dalam sistem informasi meliputi :

1) Masukkan, menyimpan, dan mengambil data.
2) Membuat laporan berdasarkan data yang telah disimpan.

2.1.7 Pemrograman Berorientasi Objek

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya [12]. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientas objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek.

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur, jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini beragam dengan *platform* yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut :

1) Meningkatkan produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).

2) Kecepatan pengembangan

Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.

3) Kemudahan pemeliharaan

Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.

4) Adanya konsistensi

Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.

5) Meningkatkan kualitas perangkat lunak

Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

2.1.7.1 Konsep Dasar Berorientasi Objek

Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya) [12]. Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata.

Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dan abstraksi objek, kelas, hubungan antar kelas sampai abstraksi sistem. Saat mengabstraksikan dan memodelkan objek, data dan proses-proses yang dipunyai oleh objek akan dienkapsulasi (dibungkus) menjadi suatu kesatuan.

Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancangan, pemrograman, dan pengujian

perangkat lunak. Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu.

Sistem berorientasi objek merupakan sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (dienkapsulasi) menjadi kelompok data dan fungsi. Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dan komponen lainnya, dan dapat berinteraksi satu sama lain.

Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek:

1. Kelas (*Class*)

   Kelas adalah sekumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statis dan himpunan objek yang sama yang mungkin lahir atau diciptakan dan kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (metode/operasi), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan dan kelas yang lain, dimana atribut dan kelas semula dapat diwariskan ke kelas yang baru. Secara teknis kelas adalah sebuah struktur dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek.

2. Objek (*object*)

   Objek adalah abstraksi dari sesuatu yang mewakili dunia nyata benda, manusia, satu organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstrak. Objek merupakan suatu *entitas* yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan.

   Secara teknis, sebuah kelas saat program dieksekusi maka akan dibuat sebuah objek. Objek dilihat dari segi teknis adalah elemen pada saat *runtime* yang akan diciptakan, dimanipulasi, dan dihancurkan saat eksekusi sehinga sebuah objek hanya ada saat sebuah program dieksekusi. Jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat *runtime* (saat sebuah program dieksekusi),

yang kita punya adalah objek, di dalam teks program yang kita lihat hanyalah kelas.

3. Metode (*method*)

   Operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri. Operasi atau metode merupakan fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek.
   Metode atau operasi dapat berasal dari *event*, aktifitas atau aksi keadaan, fungsi, atau kelakuan dunia nyata. Contoh metode atau operasi misalnya *Read, Write, Move, Copy,* dan sebagainya.

4. Atribut (*attribute*)

   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat berupa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.

5. Abstraksi (*abstraction*)

   Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapsulasi (*encapsulation*)

   Pembungkusan atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerja-nya.

7. Pewarisan (*inheritance*)

   Mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dan dirinya.

8. Antarmuka (*interface*)

   Antarmuka sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah *interface* dapat diimplementasikan oleh kelas lain.

9. *Reusability*

   Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

10. Generalisasi dan Spesialisasi

    Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil, motor, dan kereta.

11. Komunikasi Antarobjek

    Komunikasi antarobjek dilakukan lewat pesan (*message*) yang dikirim dari satu objek ke objek lainnya.

12. Polimorfisme (*polymorphism*)

    Kemampuan suatu objek digunakan di banyak tujuan yang berbeda dengan nama yang sehingga menghemat baris program.

13. *Package*

    Package adalah sebuah kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam *package* yang berbeda.

2.1.7.2 Pengenalan UML

Pada perkembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang di berbagai negara dapat mengerti pemodelan perangkat lunak. Seperti yang kita ketahui bahwa menyatukan banyak kepala untuk menceritakan sebuah ide dengan tujuan untuk memahami, merupakan hal yang tidak mudah, oleh karena itu diperlukan sebuah bahasa pemodelan perangkat lunak yang dapat dimengerti oleh banyak orang.

Banyak orang yang telah membuat bahasa pemodelan pembangunan perangkat lunak sesuai dengan teknologi pemrograman yang berkembang pada saat itu, misalnya yang sempat berkembang dan digunakan banyak pihak adalah *Data Flow Diagram* (DFD) untuk memodelkan perangkat lunak yang menggunakan pemrograman prosedural atau struktural, kemudian juga ada *State Transition Diagram* (STD) yang digunakan untuk memodelkan sistem *real time* (waktu nyata). Pada perkembangan teknik pemrograman berorientasi objek, munculah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

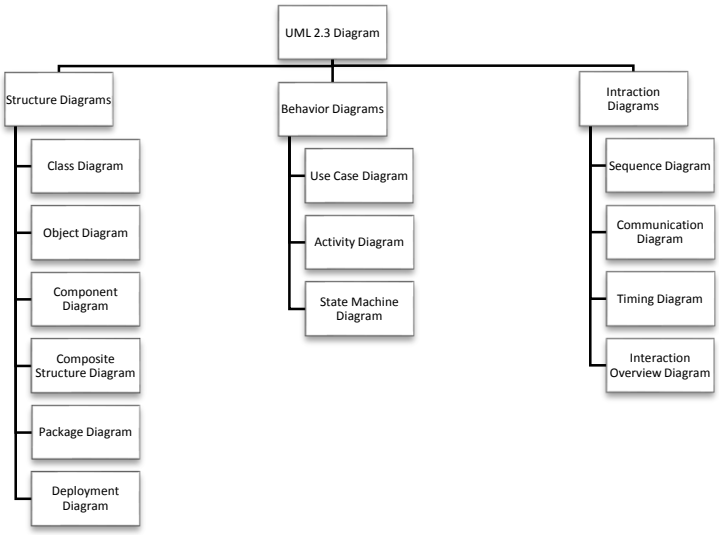
UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

Seperti yang kita ketahui di dunia sistem informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan dan konteksnya. Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunaannya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin digambarkan. Secara analogi jika dengan bahasa yang digunakan sehari-hari, belum

tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar.

2.1.7.3 Diagram UML

Pada UML 2.3 terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada Gambar 2.3 .



Gambar 2.3 Diagram UML 2.3

Berikut ini penjelasan singkat dari pembagian kategori tersebut :

1) *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2) *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.
3) *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antarsubsistem pada suatu sistem.

2.1.7.4 *Use Case Diagram*

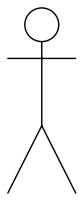
Use Case atau diagram *Use Case* merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use Case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *Use Case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu.

Syarat penamaan pada *Use Case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *Use Case* yaitu pendefinisian apa yang disebut aktor dan *Use Case*.

1.1.7.4.1 Aktor

Aktor adalah sesuatu (entitas) yang berhubungan dengan sistem dan berpartisipasi dalam *use case*. Aktor menggambarkan orang, sistem atau *entitas eksternal* yang secara khusus membangkitkan sistem dengan *input* atau masukan kejadian-kejadian, atau menerima sesuatu dari sistem. Aktor dilukiskan dengan peran yang mereka mainkan dalam *use case*, seperti Staff, Kurir dan lain-lain.

Dalam *use case diagram* terdapat satu aktor pemulai atau *initiator actor* yang membangkitkan rangsangan awal terhadap sistem, dan mungkin sejumlah aktor lain yang berpartisipasi atau *participating actor*. Akan sangat berguna untuk mengetahui siapa aktor pemulai tersebut.

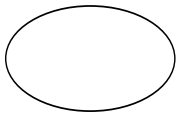


Gambar 2.4 Aktor

1.1.7.4.2 *Use Case*

Use case yang dibuat berdasarkan keperluan aktor merupakan gambaran dari "apa" yang dikerjakan oleh sistem, bukan "bagaimana" sistem mengerjakannya.

Use case diberi nama yang menyatakan apa hal yang dicapai dari interaksinya dengan aktor.

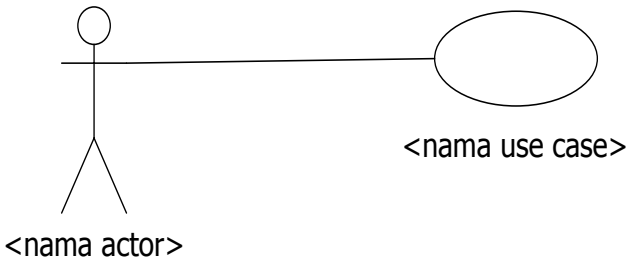


Gambar 2.5 Use case

1.1.7.4.3 *Relationship*

Relasi (*relationship*) digambarkan sebagai bentuk garis antara dua simbol dalam *use case diagram*. Relasi antara *actor* dan *use case* disebut juga dengan asosiasi (*association*). Asosiasi ini digunakan untuk menggambarkan bagaimana hubungan antara keduanya.

Relasi-relasi yang terjadi pada *use case diagram* bisa antara *actor* dengan *use case* atau *use case* dengan *use case*.



Gambar 2.6 *Relationship*

Relasi antara *use case* dengan *use case* :

a. Include, pemanggilan *use case* oleh *use case* lain atau untuk menggambarkan suatu *use case* termasuk di dalam *use case* lain (diharuskan). Contohnya adalah pemanggilan sebuah fungsi program. Digambarkan dengan garis lurus berpanah dengan tulisan <<include>>.
b. Extend, digunakan ketika hendak menggambarkan variasi pada kondisi perilaku normal dan menggunakan lebih banyak kontrol *form* dan mendeklarasikan ekstension pada *use case* utama. Atau dengan kata lain

adalah perluasan dari *use case* lain jika syarat atau kondisi terpenuhi. Digambarkan dengan garis berpanah dengan tulisan <<extend>>.

c. Generalization/Inheritance, dibuat ketika ada sebuah kejadian yang lain sendiri atau perlakuan khusus dan merupakan pola berhubungan *base-parent use case*. Digambarkan dengan garis berpanah tertutup dari *base use case* ke *parent use case*.

2.1.7.5 *Activity Diagram*

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut ini :

1) Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan
2) Urutan atau pengelompokan tampilan dari sistem/*user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan
3) Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya

2.1.7.6 *Sequence Diagram*

Diagram *sequence* menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram

sequence sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.

Penomoran pesan berdasarkan urutan interaksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu.

2.1.7.7 *Class Diagram*

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsi-fungsi sesuai dengan kebutuhan sistem. Susunan struktur kelas yang baik pada diagram kelas ebaiknya memiliki jenis-jenis kelas berikut :

1) Kelas Main

   Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.
2) Kelas yang menangani tampilan sistem

   Kelas yang mendefinisikan dan mengatur tampilan ke pemakai.
3) Kelas yang diambil dari pendefinisian *usecase*

   Kelas yang menangani fungsi-fungsi yang harus ada diambil dari pendefinisian *use case*.
4) Kelas yang diambil dari pendefinisian data

   Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data.

Jenis-jenis kelas diatas juga dapat digabungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada struktur kelas tetap ada. Susunan kelas juga dapat ditambahkan kelas utilitas seperti koneksi ke basis data, membaca *file* teks, dan lain sebagainya sesuai kebutuhan.

Dalam mendefinisikan metode yang ada di dalam kelas perlu memperhatikan apa yang disebut dengan *cohesion* dan coupling. *Cohesion* adalah ukuran seberapa

dekat keterkaitan instruksi di dalam sebuah metode terkait satu sama lain sedangkan *coupling* adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas. Sebagai aturan secara umum maka sebuah metode yang dibuat harus memiliki kadar *choesion* yang kuat dan kadar *coupling* yang lemah.

Dalam class diagram terdapat beberapa relasi (hubungan antar class) yaitu:

1. *Generalization and Inheritence*

   Diperlukan untuk memperlihatkan hubungan pewarisan (inheritance) antar unsur dalam diagram kelas. Pewarisan memungkinkan suatu kelas mewarisi semua atribut, operasi, relasi, dari kelas yang berada dalam hirarki pewarisannya.
2. *Associations*

   Hubungan statis antar class. Umumnya menggambarkan class yang memiliki atribut berupa class lain, atau class yang harus mengetahui ekstensi class lain. Dalam notasi UML kita mengenal asosiasi 2 arah (bidirectional) dan 1 arah (undirectional).
3. *Aggregation*

   Hubungan antar-class dimana class yang satu (part class) adalah bagian dari class lainnya (whole class).
4. *Composition*

   Aggregation dengan ikatan yang lebih kuat. Di dalam composite aggregation, siklus hidup part class sangat bergantung pada whole class sehingga bila objek instance dari whole class dihapus maka objek instance dari part calss juga akan terhapus.
5. *Depedency*

   Hubungan antar-class dimana sebuah class memiliki ketergantungan pada class lainnya tetapi tidak sebaliknya.
6. *Realization*

   Hubungan antar-class dimana sebuah class memiliki keharusan untuk mengikuti aturan yang ditetapkan class lainnya. Biasanya realization

digunakan untk menspesifikasikan hubungan antara sebuah interface dengan class yang mengimplementasikan interface tersebut.

2.1.8 Rekayasa Perangkat Lunak

Rekayasa perangkat lunak (*software engineering*) merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi, dipercaya dan bekerja secara efisien menggunakan mesin. Rekayasa perangkat lunak lebih fokus pada praktik pengembangan perangkat lunak dan mengirimkan perangkat lunak yang bermanfaat bagi pengguna. Rekayasa perangkat lunak fokus pada bagaimana membangun perangkat lunak yang memenuhi kriteria berikut:

1) Dapat terus dipelihara setelah perangkat lunak selesai dibuat seiiring berkembangnya teknologi dan lingkungan.
2) Dapat diandalkan dengan proses bisnis yang dijalankan dan perubahan yang terjadi.
3) Efisien dari segi sumber daya dan penggunaan.
4) Kemampuan untuk dipakai sesuai dengan kebutuhan.

Pekerjaan yang terkait dengan rekayasa perangkat lunak dapat dikategorikan menjadi tiga buah kategori umum tanpa melihat area dari aplikasi, ukuran proyek atau kompleksitas perangkat lunak yang akan dibangun, yaitu:

a. Fase Pendefinisian (*Definition Phase*)

   Fase pendefinisian fokus pada "*what*" yang artinya harus mencari tahu atau mengidentifikasi informasi apa yang harus diproses, seperti apa fungsi dan performansi yang diinginkan, seperti apa prilaku sistem yang diinginkan, apa kriteria validasi yang dibutuhkan untuk mendefinisikan sistem.

b. Fase Pengembangan (*Development Phase*)

   Fase pengembangan fokus pada "*how*" yang artinya selama tahap pengembangan perangkat lunak seorang perekayasa perangkat lunak (*software engineer*) berusaha untuk mendefinisikan bagaimana data diinstrukturkan dan bagaimana fungsi-fungsi yang dibutuhkan

diimplementasikan, bagaimana karakter antarmuka tampilan, bagaimana desanin ditranslasikan ke dalam bahasa pemrograman dan bagaimana pengujian dijalankan.

c. Fase Pendukung (*Support Phase*)

Fase pendukung fokus pada perubahan yang terasosiasi pada perbaikan kesalahan (*error*), adaptasi yang dibutuhkan pada lingkungan perangkat lunak yang terlibat dan perbaikan yang terjadi akibat perubahan kebutuhan pengguna. Fase pendukung terdiri dari empat tipe perubahan antara lain:

1) Koreksi (*Correction*), yaitu pemeliharaan dengan melakukan perbaikan terhadap kecacatan perangkat lunak.
2) Adaptasi (*Adaptation*), yaitu merupakan tahap untuk memodifikasi perangkat lunak guna mengakomodasi perubahan lingkungan luar dimana perangkat lunak dijalankan.
3) Perbaikan (*Enchancement*), pemeliharaan perfektif atau penyempurnaan melakukan eksekusi atau penambahan pada kebutuhan fungsional sebelumnya.
4) Pencegahan (*Prevention*), pencegahan atau sering disebut juga dengan rekayasa ulang sistem (*software reengineering*) harus dikondisikan untuk mempu melayani kebutuhan yang diinginkan pemakainya.

2.1.9 Perangkat Lunak Pendukung

Adapun beberapa perangkat lunak pendukung dalam pembangunan implementasi metode di aplikasi identifikasi ruas jari tangan sebagai berikut :

2.1.9.1 Net Framework Bahasa Pemrograman C#

1. .Net Framework

Microsoft mengumumkan C# pada tahun 2000 bersamaan dengan pengumuman platform .NET Framework. .NET Framework adalah platform yang merupakan perwujudan teknologi modern. Framework .NET diciptakan untuk dapat memecahkan masalah yang banyak dihadapi dunia pemrograman secara lebih

efisien. Metodologi perancangan berorientasi objek dan konsep software sebagai komponen tertanam dengan kuat di Framework.NET.

2. Bahasa Pemrograman C#

C# (C sharp) adalah sebuah bahasa pemrograman berbasis objek yang didukung oleh microsoft .NET framework. Microsoft .NET framework adalah perantara agar aplikasi dengan bahasa pemrograman yang didukung dapat berkomunikasi dengan sistem operasi yang digunakan oleh komputer yang digunakan. Selain itu, .NET framework juga memungkinkan C# berkomunikasi dengan bahasa pemrograman lainnya yang juga didukung oleh .NET framework seperti VB. NET, F# atau C++. Dengan kata lain, aplikasi yang kita buat dapat menggunakan komponen - komponen lain yang dibuat dengan menggunakan VB.NET, J#, F#, atau C++.

Banyak orang berkata bahwa C# hanya dapat digunakan pada sistem operasi windows. Namun sesungguhnya aplikasi C# dapat digunakan dalam berbagai macam sistem operasi baik windows (dengan menggunakan .NET framework), Mac OS, LINUX, serta sistem operasi berbasis UNIX lainnya (dengan menggunakan mono framework). Bahasa pemrograman C# juga banyak digunakan untuk membangun aplikasi seperti aplikasi web, aplikasi desktop, aplikasi zune, aplikasi permainan (desktop dan XBOX) dan berbagai jenis aplikasi lainnya.

Beberapa kelebihan dari bahasa C# :

1) Termasuk bahasa pemrograman .NET. dengan demikian *user* dapat menggunakan komponen - komponen yang dibangun dengan bahasa pemrograman .NET lainnya (integrasi antar bahasa).
2) Bahasa pemrograman C# memiliki *language integrated query* (LINQ) yang merupakan *sintaks query* yang dapat digunakan pada setiap kumpulan data.
3) *Windows Presentation Foundation* (WPF) dapat digunakan untuk membuat tampilan aplikasi dengan sangat kreatif.

4) Microsoft memberikan IDE (software yang digunakan untuk membangun sebuah program) secara gratis, yaitu Microsoft Visual Studio Express Edition IDE inilah yang akan digunakan untuk membangun aplikasi C#.

3. Microsoft Visual Studio 2010

Microsoft Visual Studio merupakan sebuah perangkat lunak yang dapat digunakan untuk melakukan pengembangan aplikasi, baik aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya dalam bentuk aplikasi console, aplikasi Windows, ataupun aplikasi Web. Visual Studio mencakup kompiler, SDK, *Integrated Development Environment* (IDE), dan *Library*. Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, C#, VB.NET, J++ dan J#.

Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *unmanaged*/*native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed* code (dalam bentuk Microsoft Intermediate Language di atas .NET Framework). Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi Silverlight, aplikasi Windows Mobile (yang berjalan di atas .NET Compact Framework).

Beberapa kelebihan dari Microsoft Visual Studio :

1) Dapat membuat aplikasi untuk beberapa platform termasuk Windows 7,Windows Server 2008 R2, SQL Server 2008, SharePoint. Office, Windows Azure, dan Windows Phone.
2) Lingkungan pemrograman dapat dipersonalisasi.
3) Dukungan *Application Lifecycle Management* dapat memudahkan kolaborasi, melakukan *testing* ataupun *debugging*.

2.1.10 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan suatu investigasi yang dilakukan untuk mendapatkan informasi mengenai kualitas dari produk atau layanan yang sedang diuji. Pengujian perangkat lunak juga memberikan pandangan mengenai perangkat lunak secara obyektif dan independen, yang bermanfaat dalam operasional bisnis untuk memahami tingkat risiko pada implementasinya. Teknik-teknik pengujian mencakup, namun tidak terbatas pada, proses mengeksekusi suatu bagian program atau keseluruhan aplikasi dengan tujuan untuk menemukan "bug" perangkat lunak. Bug merupakan suatu kesalahan desain pada suatu perangkat keras komputer atau perangkat lunak komputer yang menyebabkan peralatan atau program itu tidak berfungsi semestinya. Bug umumnya lebih umum dalam dunia perangkat lunak dibandingkan dengan perangkat keras.

Pengujian perangkat lunak merupakan suatu tahapan penting dalam pembangunan perangkat lunak. Pengujian dilakukan dengan cara mengevaluasi konfigurasi perangkat lunak yang terdiri dari spesifikasi kebutuhan, deskripsi perancangan, dan program yang dihasilkan. Hasil evaluasi kemudian dibandingkan dengan hasil uji yang diharapkan. Jika ditemukan kesalahan, maka perbaikan perangkat lunak harus dilakukan untuk kemudian diuji kembali.

Pengujian perangkat lunak adalah proses menjalankan dan mengevaluasi sebuah perangkat lunak secara manual maupun otomatis untuk menguji apakah perangkat lunak sudah memenuhi persyaratan atau belum, atau untuk menentukan perbedaan antara hasil yang diharapkan dengan hasil sebenarnya. Peksanaan pengujian perangkat lunak biasanya disesuaikan dengan metodologi pembangunan perangkat lunak yang digunakan.

Pengujian dikatakan berhasil apabila pengujian yang dilakukan memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya.

2.1.10.1 Teknik Pengujian

Teknik pengujian yang dapat digunakan dalam menguji perangkat lunak sebagai berikut :

a. Pengujian *White Box :* Pengujian *white box* adalah pengujian yang didasarkan pada pengecekan terhadap detail perancangan, menggunakan struktur kontrol dari desain program secara procedural untuk membagi pengujian ke dalam beberapa kasus pengujian. Secara sekilas dapat diambil kesimpulan *white box testing* merupakan petunjuk untuk mendapatkan program yang benar secara 100%.

Penggunaan metode pengujian *white box* dilakukan untuk :

1) Memberikan jaminan bahwa semua jalur independen suatu modul digunakan minimal satu kali
2) Menggunakan semua keputusan logis untuk semua kondisi true atau false
3) Mengeksekusi semua perulangan pada batasan nilai dan operasional pada setiap kondisi.
4) Menggunakan struktur data internal untuk menjamin validitas jalur keputusan.

Persyaratan dalam menjalankan strategi *White Box Testing* sebagai berikut :

1) Mendefinisikan semua alur logika
2) Membangun kasus untuk digunakan dalam pengujian
3) Mengevaluasi semua hasil pengujian
4) Melakukan pengujian secara menyeluruh

b. Pengujian *Black Box* : Pengujian *black box* digunakan untuk menguji fungsi-fungsi khusus dari perangkat lunak yang dirancang. Pada teknik ini, kebenaran perangkat lunak yang diuji hanya dilihat berdasarkan keluaran yang dihasilkan dari data atau kondisi masukan yang diberikan untuk fungsi yang ada tanpa melihat bagaimana proses untuk mendapatkan keluaran tersebut. Dari keluaran yang dihasilkan, kemampuan program dalam memenuhi

kebutuhan pemakai dapat diukur sekaligus dapat diiketahui kesalahan-kesalahannya.

Beberapa jenis kesalahan yang dapat diidentifikasi :

1) Fungsi tidak benar atau hilang
2) Kesalahan antar muka
3) Kesalahan pada struktur data (pengaksesan basis data)
4) Kesalahan inisialisasi dan akhir program
5) Kesalahan performasi

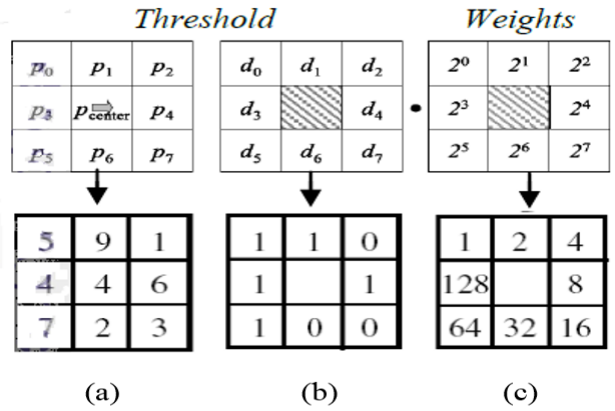
Walaupun sulit untuk menelusuri kesalahan yang mungkin didapat, teknik pengujian black box lebih sering dipilih untuk menguji perangkat lunak karena kemudahan dalam pelaksanaannya.

2.2 Teori-teori Khusus

Teori-teori khusus yang berkaitan dengan topik penelitian ini adalah tentang ***Local Binary Pattern,*** *Histogram equalization* dan *Euclidean distance.*

2.2.1 *Local Binary Pattern*

LBP didefinisikan sebagai perbandingan nilai biner piksel pada pusat citra dengan 8 nilai piksel disekelilingnya. Misal pada sebuah citra berukuran 3x3, nilai biner pada pusat citra dibandingkan dengan nilai sekelilingnya. Dengan cara mengurangkan nilai piksel disekelilingnya dengan nilai piksel pada pusat citra, jika hasilnya lebih atau sama dengan 0 maka biberi nilai 1 dan jika hasilnya kurang dari 0 maka diberi nilai 0. Setelah itu, menyusun 8 nilai biner searah jarum jam atau sebaliknya dan merubah 8 bit biner kedalam nilai desimal untuk menggantikan nilai piksel ada pusat citra.



Pattern : 11010011

Nilai LBP : 1 + 2 + 8 + 64 +128 = 203

Gambar 2.7 Gambaran proses LBP

Setiap pixel memiliki nilai hasil *grayscale*, kemudian dilakukan *threshold* berpusat pada titik tengah. Pixel yang memiliki nilai sama atau lebih dibandingkan dengan titik tengah diberi nilai 1 selain itu diberi nilai 0. Kemudian nilai LBP didapat dari penjumlahan dua pangkat nilai angka yang bernilai satu. Berikut rumusnya :

$$\mathrm{LBP_{P,R}}(x_c, y_c) = \sum_{p=0}^{p-1} s\,(g_p - g_c)2^p \qquad (2.3)$$

Dan rumus perbandingan piksel fungsi s(x) didefinisikan sebagai berikut:

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \qquad (2.4)$$

Kemudian dibuat suatu cara agar pola LBP tidak sensitif terhadap perubahan rotasi *(rotation invariant*).

$$LBP^{ri}_{P,R} = \min\{\mathrm{ROR}(LBP_{P,R}.i) \mid i=0,1,\ldots,\mathrm{P}\text{-}1\} \qquad (2.5)$$

ROR(x,i) di atas menerangkan rotasi sirkular kanan dari suatu kelompok bit x sebanyak i tahap.

Pola-pola LBP tertentu memiliki karakteristik utama dari suatu tekstur. Pola-pola yang memiliki informasi penting ini dinamakan "*uniform patterns*". LBP dikatakan uniform jika struktur melingkar pola-pola binernya paling banyak terdiri atas dua transisi bit dari 0 ke 1 atau sebaliknya. *Uniform patterns* berfungsi untuk

mengidentifikasi noda (spot), flat area atau dark spot, sudut, dan tepi. Hampir 90 persen dari tekstur merupakan *uniform patterns* [13].



Gambar 2.8 Contoh tekstur *uniform patterns*

Di dalam pemetaan *uniform* LBP ini, terdapat pelabelan yang terpisah untuk setiap pola-pola yang *uniform* dan semua pola-pola yang *non uniform* dimasukkan ke dalam satu label. Sebuah rumusan sederhana untuk menghitung jumlah label pola biner *uniform* dapat terlihat dari P(P-1)+3. P di sini berarti jumlah titik biner yang digunakan. Misalnya untuk pola biner 8 bit, berarti terdapat 59 label pola *uniform* yang berbeda dan 243 label untuk 16 bit.

Penggabungan antara *uniform patterns* dengan *rotation invariant* dilambangkan $LBP_{P,R}^{riu2}$. Notasi ri menunjukkan *rotation invariant* dan u2 untuk *uniform patterns* pada sampling points P dan radius R. $LBP_{P,R}^{riu2}$ merupakan ukuran ketidaksensitifan (invariant) terhadap perubahan *grayscale* dan merupakan ukuran yang digunakan untuk pola spasial [5].

2.2.2 *Histogram Equalization*

Informasi suatu citra dapat diwakili dengan suatu histogram, yaitu suatu fungsi yang menunjukkan jumlah titik yang ada dalam suatu citra untuk setiap tingkat warna tertentu. *Histogram equalization* merupakan suatu proses pemerataan histogram, dimana distribusi citra pada derajat keabuan dibuat merata.

Berikut merupakan rumus yang digunakan dengan citra dengan skala keabuan k – bit :

$$K = round\ \frac{(c_i(2^k-1))}{(w.h)} \tag{2.6}$$

Keterangan :

Ci : distribusi kumulatif nilai skalar keabuan ke- i dari citra asli.

Round : fungsi pembulatan kebilangan terdekat, misalnya 35.4 menjadi 35.

K : nilai keabuan hasil *histogram equalization*.

W : lebar citra.

h : tinggi citra.

Rumus histogram ditulis kembali sebagai peluang :

$$P_r(r_k) = \frac{n_k}{n} \quad \text{dimana} \quad r_k = \frac{k}{L-1} \quad ,0 \le k \le L-1 \tag{2.7}$$

K : nilai hasil *grayscale.*

$\mathbf{n_k}$: total nilai K yang muncul.

L : derajat keabuan.

N : jumlah nilai pixel.

Perhitungan perataan histrogram ditulis sebagai berikut :

$$s_0 = T(r_0) = \sum_{j=0}^{0} P_r(r_j) = P_r(r_0) = ...$$

$$s_1 = T(r_1) = \sum_{j=0}^{1} P_r(r_j) = P_r(r_0) + P_r(r_1) = s_0 + ? = ...$$

$$s_2 = T(r_2) = \sum_{j=0}^{2} P_r(r_j) = P_r(r_0) + P_r(r_1) + P_r(r_2) = s_0 + s_1 + ? = ...$$

(2.8)

$P_r(r_0)$: nilai pixel pertama yang muncul di histogram.

2.2.3 *Euclidean Distance*

Euclidean distance adalah matrik yang paling sering digunakan untuk menghitung kesamaan 2 vektor. *Euclidean distance* menghitung akar dari kuadrat perbedaan 2 vektor.

Rumus dari *euclidean distance*:

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2} \quad (2.9)$$

d_{ij} : jarak perbandingan ke 2 vektor.

X_{ik} : nilai vektor 2

X_{jk} : nilai vektor 1