BAB 2
LANDASAN TEORI

2.1 *Artificial Intelligence*

Kecerdasan buatan atau *artificial intelligence* (*AI*) merupakan cabang ilmu yang berusaha memahami kecerdasan manusia. *AI* berusaha membangun entitas-entitas cerdas yang sesuai dengan pemahaman manusia. Entitas-entitas cerdas yang yang dibangun *AI* ini ternyata sangat menarik dan mempercepat proses pemahaman terhadap kecerdasan manusia [16].

Stuart Russel dan Peter Norvig mengelompokan definisi *AI* kedalam empat kategori, yaitu:

1. *Thinking humanly*

   Pendekatan ini dilakukan dengan dua cara, yaitu:

   a. Melalui introspeksi: mencoba menangkap pemikiran kita sendiri pada saat berfikir. Tetapi seorang psikolog barat mengatakan *"how do you know that you understand?"* bagaimana anda tahu bagaimana anda mengerti? Karena pada saat anda menyadari pemikiran anda, ternyata pemikiran tersebut sudah lewat dan digantikan kesadaran anda. Sehingga definisi ini terkesan mengada-ada dan tidak mungkin dilakukan.
   b. Melalui eksperimen psikologi.

2. *Acting humanly*

   Pada tahun 1950, Alan Turing merancang suatu ujian bagi *computer* berintelejensia untuk menguji apakah komputer tersebut mampu mengelabuhi seorang manusia yang menginterogasinya melalui *teletype* (komunikasi berbasis teks jarak jauh).

3. *Thinking rationally*

   Terdapat dua masalah dalam pendekatan ini, yaitu:

   a. Tidak mudah untuk membuat pengetahuan informasi in*form*al dan menyatakan pengetahuan tersebut ke dalam *form*al term yang

diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.

b. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. *Acting rationally*

   Membuat inferensi yang logis merupakan bagian dari suatu rational agent. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka *agent* bisa melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio yakni refleks dan intuitif (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini, maka banyak produk komputasi cerdas saat ini yang tidak laya disebut sebagai produk *AI*.

Definisi *thinking rationally* terasa lebih sempit daripada *acting rationally*. Oleh karena itu, definisi *AI* yang paling tepat untuk saat ini adalah *acting rationally* dengan pendekatan *normal agent*. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut [16].

2.1.1 *Computer Vision*

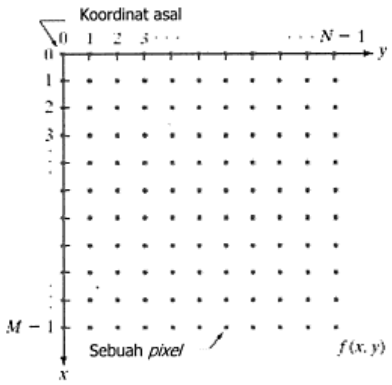
Computer vision merupakan proses otomatis yang mengintegrasikan sejumlah besar proses untuk persepsi visual, seperti akuisisi citra, pengolahan citra, klasifikasi, pengenalan (*recognition*), dan membuat keputusan [17]. Pada hakikatnya, *computer vision* mencoba meniru cara kerja sistem visual manusia (*human vision*). *Human vision* sesungguhnya sangat kompleks. Manusia melihat objek dengan indera penglihatan (mata), lalu citra objek diteruskan ke otak untuk diinterpretasi sehingga manusia mengerti objek apa yang tampak dalam

pandangan matanya. Hasil interpretasi ini mungkin digunakan untuk pengambilan keputusan (misalnya menghindar kalau melihat mobil melaju di depan).

2.2 Citra Digital

Secara umum, pengolahan citra digital menunjuk pada pemrosesan gambar dua dimensi menggunakan komputer. Dalam konteks yang lebih luas, pengolahan citra digital mengacu pada pemrosesan setiap data dua dimensi. Citra digital merupakan sebuah larik (*array*) yang berisi nilai-nilai real maupun komplek yang direpresentasikan dengan deretan bit tertentu [18].

Suatu citra dapat didefinisikan sebagai fungsi *f(x,y)* berukuran M baris dan N kolom, dengan x dan y adalah koordinat spasial, dan amplitudo *f* di titik koordinat (x,y) dinamakan intensitas atau tingkat keabuan dari citra pada titik tersebut. Apabila nilai x, y, dan nilai amplitudo f secara keselurhan berhingga dan bernilai diskrit, maka dapat dikatakan bahwa citra tersebut adalah citra digital. Gambar 2.1 menunjukkan posisi koordinat citra digital.



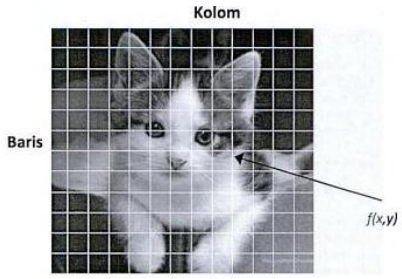
Gambar 2.1. Koordinat Citra Digital [18]

Citra digital dapat ditulis dalam bentuk matrik sebagai berikut.

$$f(x,y)=\begin{bmatrix} f(0,0) & f(0,1) & \cdots & f(0,N-1) \\ f(1,0) & f(1,1) & \cdots & f(1,N-1) \\ \vdots & \vdots & & \vdots \\ f(M-1,0) & f(M-1,1) & \cdots & f(M-1,N-1) \end{bmatrix}$$

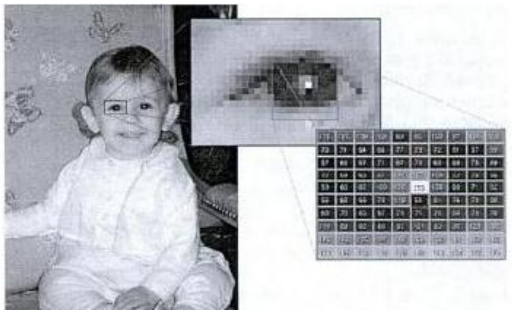
Nilai pada suatu irisan antara baris dan kolom (pada posisi x,y) disebut dengan *picture elements*, *pels*, atau piksels. Istilah terakhir (piksel) paling sering

digunakan pada citra digital. Gambar 2.2 menunjukkan ilustrasi digitalisasi citra dengan M=16 baris dan N=16 kolom.



Gambar 2.2. Ilustrasi digitalisasi citra [18]

Gambar 2.3 menyajikan contoh lain dari suatu citra digital (citra *grayscale*), dengan nilai intensitas dari citra pada area tertentu.



Gambar 2.3. Contoh citra grayscale, *cropping* dan intensitas [18]

2.3 Model Warna Pada Citra

Warna adalah persepsi yang dirasakan oleh sistem visual manusia terhadap panjang gelombang cahaya yang dipantulkan oleh objek [17]. Setiap warna mempunyai panjang gelombang yang berbeda. Warna merah mempunyai panjang gelombang paling tinggi, sedangkan warna ungu (*violet*) mempunyai panjang gelombang paling rendah.

Warna-warna yang diterima oleh mata (sistem visual manusia) merupakan hasil kombinasi cahaya dengan panjang gelombang berbeda. Penelitian memperlihatkan bahwa kombinasi warna yang memberikan rentang warna yang paling lebar adalah *red* (R), *green* (G), dan *blue* (B).

Dalam pembentukan citra digital model warna yang umum digunakan antara lain sebagai berikut :

2.3.1 Citra *RGB*

Citra *RGB*, yang biasa disebut juga citra *true color*, disimpan dalam citra berukuran (mxn)x3 yang mendefinisikan warna merah (*red*), hijau (*green*), dan warna biru (*blue*) untuk setiap pikselnya. Warna pada tiap piksel ditentukan berdasarkan kombinasi dari warna *red*, *green* dan *blue* (*RGB*). *RGB* merupakan citra 24 *bit* dengan komponen merah, hijau, biru yang masing-masing umumnya bernilai 8 *bit* sehingga intensitas kecerahan warna sampai 256 level dan kombinasi warnanya kurang lebih sekitar 16 juta warna sehingga disebut "*true color*" [17].

2.3.2 Citra Keabuan

Citra dengan derajat keabuan berbeda dengan citra *RGB*, citra ini didefinisikan oleh satu nilai derajat warna. Umumnya bernilai 8 *bit* sehingga intensitas kecerahan warna sampai 256 *level* dan kombinasi warnanya 256 varian. Tingkat kecerahan paling rendah yaitu 0 untuk warna hitam dan putih bernilai 255 [17].

Untuk mengkonversikan citra yang memiliki warna *RGB* ke derajat keabuan digunakan rumus:

$$gray = \frac{R+G+B}{3} \qquad (2.1)$$

atau

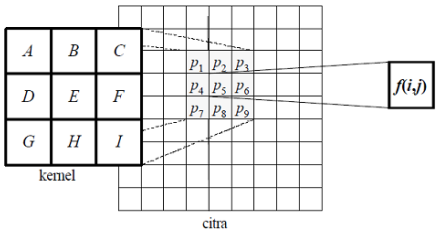
$$gray = 0.299R + 0.587G + 0.114B \qquad (2.2)$$

2.4 *Neighborhood Filtering* (*Convolution*)

Convolution merupakan proses dimana citra dimanipulasi dengan menggunakan eksternal mask atau kernel untuk menghasilkan citra yang baru berdasarkan *neighborhood operator* atau *local operator*. *Convolution* digunakan untuk *local tone adjustment*, *bluring*, mempertajam detail, menonjolkan tepi atau menghilangkan *noise* [19]. Dalam citra yang memiliki fungsi dua dimensi, operasi *convolution* didefinisikan sebagai berikut:

$$f(i,j) = \sum_{k,l} p(i+k, j+l).h(k,l) \qquad (2.3)$$

Fungsi penapis h disebut juga *convolution filter*, *convolution mask*, *convolution kernel*. Ilustrasi *convolution* ditunjukkan pada Gambar 2. 4.



Gambar 2. 4. Ilustrasi *convolution*

Sehingga *convolution* untuk $f(i,j)$ dapat didefinisikan sebagai berikut:

$$f(i,j) = A.p_1 + B.p_2 + C.p_3 + D.p_4 + E.p_5 + F.p_6 + G.p_7 + H.p_8 + I.p_9$$

Operasi *convolution* dilakukan dengan menggeser kernel *convolution* piksel per piksel. Hasil *convolution* disimpan di dalam matriks yang baru.

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

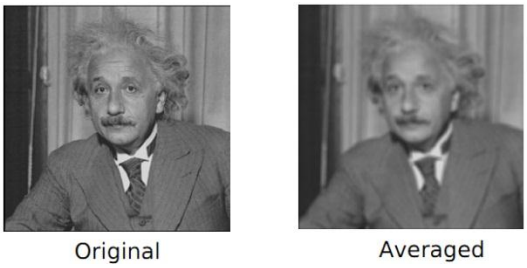
Gambar 2. 5. Ilustrasi hasil operasi *convolution*

2.5 Penghalusan Citra (*Image Smoothing*)

Penghalusan citra (*smoothing*) merupakan prosses untuk meminimalisir *noise* atau mengaburkan citra [19]. Dalam penghalusan citra operasi sederhana yang dapat digunakan adalah *average* atau rata-rata. Citra yang akan diperhalus di konvolusi dengan kernel $h = (mxn)$. Dimana kernel h dapat didefinisikan, sebagai berikut:

$$h = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & 1 & \cdots & 1 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \cdots & 1 \end{bmatrix} \cdot \frac{1}{Z}$$

Hasil dari penghalusan dengan rata-rata ini dapat di lihat pada Gambar 2.6.

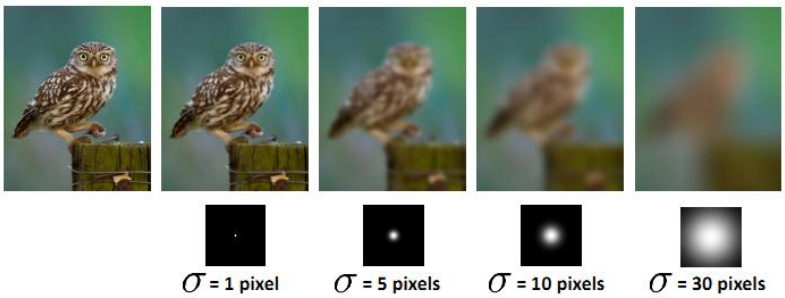


Gambar 2.6. Penghalusan citra einstein dengan *average filter*

Kernel lain yang dapat digunakan adalah kernel *gaussian*. Kernel *gaussian* dinilai baik dalam penghalusan citra karena mementingkan piksel ketentanggan terdekat. Fungsi gaussian dapat didefinisikan sebagai berikut:

$$g(x,y) = \frac{1}{Z} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \qquad (2.4)$$

Dimana $\frac{1}{Z}$ adalah koefisien normalisasi pada fungsi *gaussian filter*. Adapun hasil penghalusan dengan kernel *gaussian* ini dapat dilihat pada Gambar 2.7.



Gambar 2.7. Penghalusan citra burung dengan *gaussian filter*

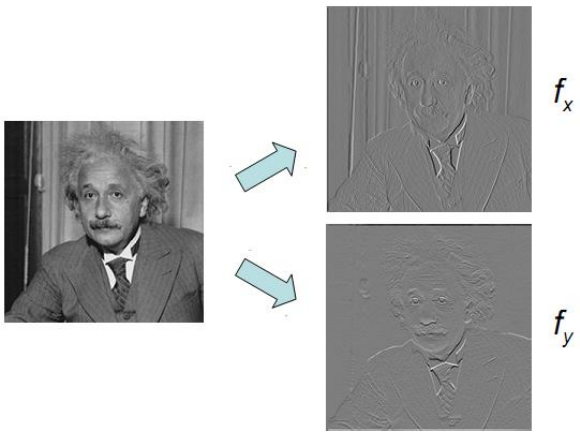
Selain itu terdapat fungsi turunan dari *gaussian* yang disebut *DOG* (*derivative of gaussian*). Fungsi *DOG* dinilai baik dalam mendeteksi tepi pada citra. Fungsi *DOG* berfungsi untuk menghitung turunan citra sehingga akan menghasilkan citra gradasi pada arah *x* (garis vertikal) dan *y* (garis horisontal) Adapun fungsi *DOG* dapat didefinisikan sebagai berikut:

Gradasi citra *x* : $g'_x(x,y) = \frac{-x}{\sigma^2 Z} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$ (2.5)

Gradasi citra *y* : $g'_y(x,y) = \frac{-x}{\sigma^2 Z} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)}$ (2.6)

Magnitude citra : $M(x,y) = \sqrt{{g'_x}^2(x,y) + {g'_y}^2(x,y)}$ (2.7)

Adapun hasil penurunan citra menjadi citra gradasi arah *x* dan *y* dapat dilihat pada Gambar 2.8.



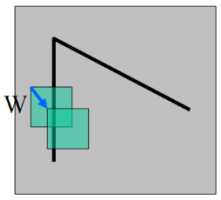
Gambar 2.8. Gradasi citra

2.6 *Corner Detection* (Shi-Tomasi/*Minimum Eigen Value*)

C*orner detection* adalah sebuah pendekatan yang digunakan pada sistem *computer vision* untuk mengekstrak beberapa jenis fitur penting dan menyimpulkan isi suatu gambar.

Shi-Tomasi *corner detection* (*minimum eigen value*) adalah algoritma pendeteksi sudut (titik/fitur penting) yang dikembangkan oleh Jianbo Shi dan Carlo Tomasi pada 1994 [11]. Algoritma ini secara langsung menghitung nilai minimum dari dua nilai eigen yang didapatkan dari dekomposisi matriks sudut. Jika nilai kedua nilai eigen mendekati nol maka menyatakan bahwa tidak terdapat sudut yang terdeteksi. Jika salah satu nilai eigen bernilai besar positif dan yang lainnya mendekati nol, maka sebuah sisi (*edge*) terdeteksi. Jika kedua nilai eigen bernilai besar positif, maka dinyatakan sebagai sudut.

Algoritma Shi-Tomasi dinilai baik untuk dilakukan penelusuran (*tracking*) objek pada citra yang pergeserannya tidak terlalu besar. Nilai eigen minimum digunakan untuk menelusuri titik pada citra lain.



Gambar 2. 9. Pergeseran *patch*

Adapun tahapan pendeteksian sudut Shi-Tomasi adalah sebagai berikut:

1. Menentukan turunan citra

   Turunan citra dihitung menggunakan *DoG* (*derivative of gaussian*), merupakan fungsi turunan dari gaussian dan dilakukan konvolusi (dinotasikan $*$) dengan citra asli A untuk menghitung aproksimasi turunan pada perubahan vertikal I_x dan horisontal I_y.

   $$I_x(x,y) = g'_x * A \text{ , } I_y(x,y) = g'_y * A \quad (2.8)$$

2. Menghitung produk citra turunan ${I^2}_x$, ${I^2}_y$, $I_x I_y$

   $${I^2}_x = I_x . I_x \text{ , } {I^2}_y = I_y . I_y \text{ , } I_x I_y = I_x . I_y \quad (2.9)$$

3. Konvolusi dengan *gaussian*

   $${S^2}_x = g' * {I^2}_x \text{ , } {S^2}_y = g' * {I^2}_y \text{ , } S_{xy} = g' * I_x I_y \quad (2.10)$$

4. Menentukan matriks sudut

   Matriks sudut dirumuskan sebagai

   $$C = \sum_{(x,y)\epsilon W} \begin{bmatrix} {S^2}_x & S_x S_y \\ S_x S_y & {S^2}_y \end{bmatrix} \quad \text{atau}$$

   $$C = \begin{bmatrix} \sum {S^2}_{xi} & \sum S_{xi} S_{yi} \\ \sum S_{xi} S_{yi} & \sum {S^2}_{yi} \end{bmatrix} \quad (2.11)$$

   W adalah *patch image* berukuran $lebarxlebar$.

5. Menghitung nilai eigen $\lambda_{\pm}$

   Nilai eigen diperoleh dengan menyelesaikan persamaan

   $$\det(A - \lambda_{\pm} I) = 0 \quad (2.12)$$

   dimana $A = C$ adalah matriks 2x2, sehingga didapatkan

   $$det \begin{bmatrix} c_{11} - \lambda_{\pm} & c_{12} \\ c_{21} & c_{22} - \lambda_{\pm} \end{bmatrix} = 0 \quad (2.13)$$

   sehingga solusi $\lambda_{\pm}$ adalah

   $$\lambda_{\pm} = \frac{1}{2}\left[(c_{11} + c_{22}) \pm \sqrt{4c_{12}c_{21} + (c_{11} - c_{22})^2}\right] \quad (2.14)$$

6. Mencari respon pada setiap piksel

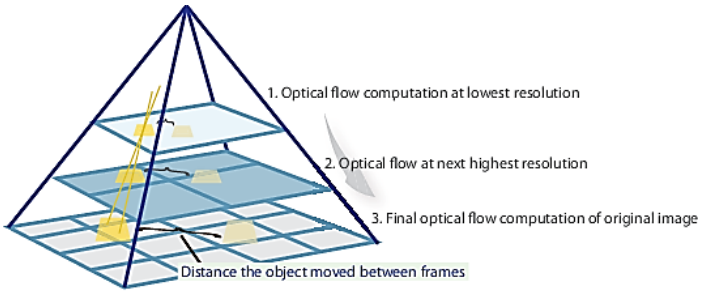
   Sebuah piksel dinyatakan sebagai sudut jika lebih dari *threshold* λ yang ditentukan

   $$\min(\lambda_+, \lambda_-) > \lambda \quad (2.15)$$

2.7 *Feature Tracking* (*KLT*)

Pada *computer vision*, Kanade-Lucas-Tomasi (*KLT*) [20] adalah algoritma penelusuran titik (*feature tracking*) pada citra untuk mencari titik yang bersesuaian pada citra lain. Penelusuran titik ini dapat digunakan di berbagai bidang seperti stabilisasi video, estimasi pergerakan kamera dan penelusuran objek. Algoritma ini bekerja dengan baik pada penelusuran objek yang tidak mengalami perubahan bentuk dan menonjolkan tekstur.

KLT pada dasarnya mengimplementasi metode *pyramidal* yang menghasilkan piramida citra. Setiap *level* pada piramida citra memiliki resolusi lebih kecil setengahnya dari citra sebelumnya. *Level* 0 terdapat di paling bawah dan merupakan citra asli dengan resolusi yang belum direduksi. Penentuan *level* piramida lebih dari satu akan membuat algoritma menelusuri titik pada banyak *level* resolusi citra, yang dimulai dari *level* terendah (resolusi terkecil).



Gambar 2.10. Piramida citra pada *KLT* [21]

Setiap level piramida dibentuk dengan melakukan *down sampling level* sebelumnya dengan faktor sebesar 2 untuk tinggi dan lebar citra. Penelusuran titik dimulai dari level resolusi terendah dan berlanjut sampai terjadi konvergensi. Hasil penelusuran pada level awal dijadikan inisialisasi untuk level berikutnya sebagai hasil tebakan awal lokasi titik. Dengan cara ini, penelusuran diperbaiki pada setiap *level*, sampai ke citra yang asli. Penggunaan piramida ini membuat penelusuran dapat menangani pergerakan piksel yang tinggi, yang jaraknya bisa lebih dari ukuran piksel tetangga.



Gambar 2.11. Reduksi ukuran piramida citra

Jika diketahui u adalah sebuah titik pada citra I, maka tujuan utama dari algoritma *KLT* adalah mencari titik yang bersesuaian v pada citra J. Adapun algoritma *KLT* adalah sebagai berikut.

1. Membuat representasi piramida citra I dan J

   $I: \{ I^L \}_{L=0,\dots,Lm}$ (2.16)

   $J: \{ J^L \}_{L=0,\dots,Lm}$ (2.17)

2. Inisialisasi tebakan piramida

   $g^{Lm} = \begin{bmatrix} g_x^{Lm} & g_y^{Lm} \end{bmatrix}^T = [0 \quad 0]^T$ (2.18)

3. Ulangi $L = L_m$ sampai 0 dengan step -1

   a. Menghitung lokasi titik u pada citra I^L

      $u^L = [p_x \quad p_y]^T = u/2^L$ (2.19)

   b. Turunan citra I^L terhadap arah x

      $I_x(x,y) = \frac{I^L(x+1,y) - I^L(x-1,y)}{2}$ (2.20)

   c. Turunan citra I^L terhadap arah y

      $I_y(x,y) = \frac{I^L(x,y+1) - I^L(x,y-1)}{2}$ (2.21)

   d. Menghitung matriks gradasi spasial

      $$G = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} I_x^2(x,y) & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y^2(x,y) \end{bmatrix}$$

      (2.22)

   e. Inisialisasi iterasi LK

      $\bar{v}^0 = [0 \;\; 0]^T$ (2.23)

   f. Ulangi $k = 1$ sampai K dengan step 1 (atau)

      sampai $\|\bar{n}^k\| < threshold$

i. Menghitung perbedaan citra

$$\delta I_k(x,y) = I^L(x,y) - J^L\left(x + g_x^L + v_x^{k-1}, y + g_y^L + v_y^{k-1}\right) \quad (2.24)$$

ii. Vektor ketidakcocokan citra

$$\bar{b}_k = \sum_{x=p_x-w_x}^{p_x+w_x} \sum_{y=p_y-w_y}^{p_y+w_y} \begin{bmatrix} \delta I_k(x,y) I_x(x,y) \\ \delta I_k(x,y) I_y(x,y) \end{bmatrix} \quad (2.25)$$

iii. *Optical flow*

$$\bar{n}^k = G^{-1}\bar{b}_k \quad (2.26)$$

iv. Menebak untuk iterasi berikutnya

$$\bar{v}^k = \bar{v}^{k-1} + \bar{n}^k \quad (2.27)$$

g. Hitung *optical flow* akhir pada level L

$$d^L = \bar{v}^K \quad (2.28)$$

h. Menebak untuk level berikutnya $L - 1$

$$g^{L-1} = [g_x^{L-1} \quad g_y^{L-1}]^T = 2(g^L + d^L) \quad (2.29)$$

4. Menghitung vektor *optical flow* akhir

$$d = g^0 + d^0 \quad (2.30)$$

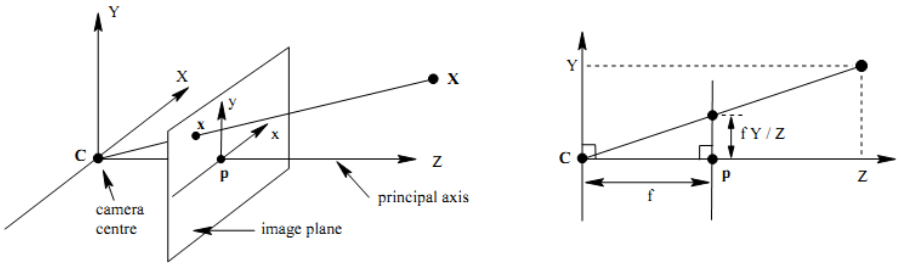
5. Menghitung lokasi titik v pada citra J

$$v = u + d \quad (2.31)$$

2.8 Pembentukan Citra dan Model Kamera

Model sederhana untuk menggambarkan proses pembentukan citra pada sebuah kamera adalah dengan menggunakan model kamera lubang jarum (*pinhole camera model*). Dalam model aslinya, kamera jenis ini tidak lain hanya terdiri dari sebuah lubang yang sangat kecil pada sisi depan sebagai tempat masuk cahaya yang merupakan *pusat proyeksi*, lihat Gambar 2.12. Jarak antara pusat proyeksi dengan bayangan citra dinamakan *nilai fokus* kamera.

2.8.1 Model Dasar Pinhole



Gambar 2.12. Geometri kamera lubang jarum [8]

Pada model kamera *pinhole*, sebuah titik pada ruang dengan koordinat $X = (X, Y, Z)^T$ dipetakan ke titik pada *image plane* (bidang citra) dimana garis yang menghubungkan titik X ke pusat proyeksi bertemu pada *image plane*. Hal ini ditunjukkan pada Gambar 2.12. Kemudian dapat dihitung bahwa titik $(X, Y, Z)^T$ dipetakan ke titik $(fX/X, fY/Z, f)^T$ pada *image plane*. Sehingga dengan sementara mengabaikan hasil akhir koordinat citra, dapat dilihat bahwa

$$(X, Y, Z)^T \mapsto (fX/Z, fY/Z)^T \qquad (2.32)$$

menjelaskan pemetaan pusat proyeksi dari koordinat dunia ke koordinat citra. Inilah pemetaan dari Euclidean 3-*space* ke Euclidean 2-*space*.

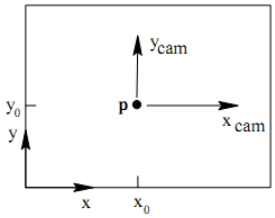
Pusat proyeksi dinamakan *camera centre* (pusat kamera), atau disebut juga *optical centre* (pusat optikal). Garis dari pusat kamera yang tegak lurus dengan bidang citra disebut *principal axis* atau *principal ray* kamera, dan titik pertemuan *principal axis* dengan *image plane* disebut *principal point*. Bidang yang melalui *camera centre* paralel dengan *image plane* disebut *proncipal plane* [8].

Jika koordinat dunia dan citra direpresentasikan kedalam bentuk vektor homogen, maka pusat proyeksi dengan sederhana dapat diekspresikan sebagai pemetaan linier antara koordinat homogen dunia dan citra tersebut. Sehingga persamaan (2.1) dapat ditulis kedalam bentuk perkalian matriks

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad (2.33)$$

Ekspresi matriks pada (2.2) dapat ditulis sebagai $diag(f, f, 1)[I \mid 0]$ dimana $diag(f, f, 1)$ adalah matriks diagonal dan $[I \mid 0]$ adalah matriks identitas 3x3 ditambah kolom vektor nol.

Notasi X untuk titik pada koordinat dunia direpresentasikan dengan 4-vektor homogen $(X, Y, Z, 1)^T$, dan x untuk titik pada koordinat citra yang direpresentasikan dengan 3-vektor homogen, dan P adalah matriks proyeksi kamera $3x4$. Sehingga persamaan (2.33) dapat ditulis dengan singkat sebagai $x = PX$ yang mendefinisikan matriks kamera untuk pusat proyeksi dari model *pinhole* sebagai $P = diag(f, f, 1)[I \mid 0]$.



Gambar 2.13. Sistem koordinat citra dan kamera [8]

Ekspresi (2.32) diasumsikan bahwa koordinat asal pada *image plane* berada pada *principal point*. Pada prakteknya tidaklah demikian, sehingga secara umum terdapat pemetaan

$$(X, Y, Z)^T \mapsto \left(fX/Z + p_x, fY/Z + p_y\right)^T$$

dimana $(p_x, p_y)^T$ adalah koordinat *principal point*. Lihat Gambar 2.13. Persamaan ini dapat diekspresikan menjadi

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad (2.34)$$

kemudian dapat didefinisikan juga K

$$K = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \qquad (2.35)$$

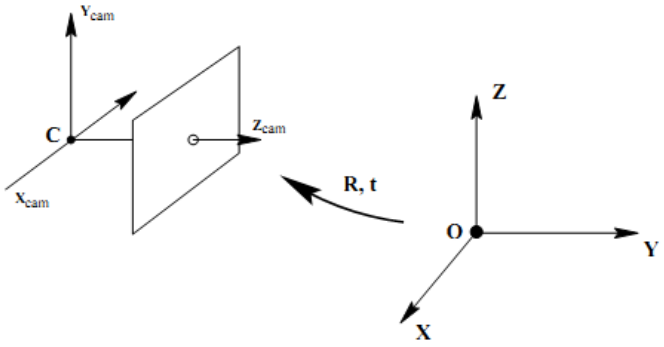
sehingga persamaan (2.33) memiliki bentuk ringkas

$$x = K[\, I \mid 0\,]X_{cam} \qquad (2.36)$$

Matriks K disebut matriks kalibrasi kamera. Pada persamaan (2.36) $(X, Y, Z, 1)^T$ ditulis sebagai X_{cam} untuk menegaskan bahwa kamera diasumsikan berada pada titik asal sistem koordinat *Euclidean* dengan *principal axis* mengarah ke sumbu Z, dan titik X_{cam} diekspresikan pada sistem koordinat tersebut. Sistem

koordinat tersebut dapat disebut sebagai *camera coordinat frame* (frame koordinat kamera).

2.8.2 Rotasi dan Translasi (Pose Kamera)



Gambar 2.14. Transformasi koordinat frame dunia ke kamera [8]

Secara umum, titik pada ruang akan diekspresikan dalam bentuk *world coordinate frame*. Dua koordinat frame berhubungan lewat rotasi dan translasi. Lihat Gambar 2.14. Jika X adalah 3-vektor bukan homogen yang merepresentasikan titik koordinat pada frame dunia, dan X_{cam} merepresentasikan titik yang sama pada koordinat frame kamera, maka dapat ditulis $X_{cam} = R(X - C)$, dimana C merepresentasikan koordinat dari pusat kamera pada koordinat *frame* dunia, dan R adalah matriks rotasi 3x3 merepresentasikan orientasi koordinat frame kamera. Persamaan ini dapat ditulis dalam koordinat homogen sebagai

$$X_{cam} = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} R & -RC \\ 0 & 1 \end{bmatrix} X \quad (2.37)$$

Dengan menggabungkannya dengan persamaan (2.5) akan menghasilkan formula

$$x = KR[\, I \mid -C \,]X \quad (2.38)$$

dimana X sekarang berada pada koordinat frame dunia. Ini adalah pemetaan umum yang diberikan oleh model kamera *pinhole*. Parameter yang terkandung dalam K disebut *parameter internal kamera*, atau orientasi internal kamera. Parameter R dan C yang menghubungkan orientasi kamera dan posisi ke sistem koordinat dunia disebut *parameter eksternal kamera*, atau orientasi ekternal [8].

Biasanya akan lebih mudah untuk tidak membuat pusat kamera eksplisit, dan bukannya merepresentasikan transformasi dunia ke citra sebagai $X_{cam} = RX + t$. Dalam kasus ini secara sederhana matriks proyeksi kamera menjadi,

$$P = K [R | t] \quad (2.39)$$

dimana dari persamaan (2.7), $t = -RC$.

2.8.3 Kamera *CCD*

Charge-coupled device (*CCD*) adalah sebuah perangkat yang biasanya ada dalam kamera digital yang memiliki fungsi mendeteksi cahaya yang masuk dan mengkonversinya menjadi sinyal digital (citra digital) [8]. Model kamera lubang jarum yang dijelaskan sebelumnya memberi asumsi bahwa koordinat citra koordinat *Euclidean* yang memiliki skala yang sama pada kedua arah sumbu. Pada kasus kamera *CCD*, terdapat kemungkinan piksel tidak berbentuk persegi. Jika koordinat citra diukur dengan satuan piksel, maka akan terdapat faktor skala pada setiap arah. Terutama jika jumlah piksel per satuan jarak dalam koordinat citra adalah m_x dan m_y pada arah *x* dan *y*, maka transformasi dari koordinat dunia ke koordinat citra (dalam piksel) diperoleh dengan mengalikan persamaan (2.4) di sebelah kiri dengan faktor tambahan $diag(m_x, m_y, 1)$. Sehingga, bentuk umum dari matriks kalibrasi kamera CCD adalah

$$K = \begin{bmatrix} a_x & & x_0 \\ & a_y & y_0 \\ & & 1 \end{bmatrix} \quad (2.40)$$

dimana $a_x = fm_x$ dan $a_y = fm_y$ merepresentasikan *focal length* kamera dalam dimensi piksel. Sama juga untuk (x_0, y_0) adalah *principal point* dalam dimensi piksel, dengan koordinat $x_0 = m_x p_x$ dan $y_0 = m_y p_y$.

Bentuk yang lebih umum dari matriks kalibrasi kamera pada proyeksi kamera adalah

$$K = \begin{bmatrix} a_x & s & x_0 \\ & a_y & y_0 \\ & & 1 \end{bmatrix} \quad (2.41)$$

dimana *s* adalah parameter *skew* (miring atau tidak simetris). Parameter *skew* akan bernilai nol pada kebanyakan kamera. Namun dalam beberapa kasus tertentu (misalnya: hasil fotografi difoto kembali), nilai *skew* bisa tidak nol.

2.8.4 Matriks Proyeksi Kamera

Matriks proyeksi kamera atau matriks kamera adalah matriks yang memproyeksikan titik pada sistem koordinat dunia kedalam sistem koordinat citra (piksel) [8]. Hasil dari proyeksi ini memiliki satuan piksel.

$$x = PX \qquad (2.42)$$

Namun secara matematis sebenarnya matriks proyeksi pertama memproyeksikan terlebih dahulu ke sistem koordinat kamera dengan melakukan rotasi dan translasi, kemudian hasilnya diproyeksikan kedalam sistem koordinat citra menggunakan matriks kalibrasi. Adapun persamaannya adalah sebagai berikut.

Pertama, proyeksi ke sistem koordinat kamera sesuai persamaan.

$$X_{cam} = [R|t]X \qquad (2.43)$$

Kedua, proyeksi ke sistem koordinat citra.

$$x = KX_{cam} \qquad (2.44)$$

Maka proyeksi titik dunia menjadi titik citra secara umum dapat ditulis,

$$x = K[R|t]X \qquad (2.45)$$

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{bmatrix} fm_x & s & m_x p_x \\ 0 & fm_y & m_y p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad (2.46)$$

atau

$$\begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{bmatrix} a_x & s & x_0 \\ 0 & a_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad (2.47)$$

Dimana,

x : titik pada citra (satuan piksel) berupa matriks 3x1 pada sistem koordinat homogen.

$\hat{x}$: titik pada frame / sistem koordinat kamera (satuan fisik) berupa matriks 3x1 pada sistem koordinat homogen.

X : titik pada dunia (satuan fisik mm, cm, dsb) berupa matriks 4x1.

K : 3x3 matriks kalibrasi (parameter internal kamera)

R : 3x3 matriks rotasi kamera (parameter eksternal kamera)

t : 3x1 matriks translasi kamera (parameter eksternal kamera)

f : *focal length* kamera (satuan fisik)

s : nilai *skew* piksel (satuan piksel)

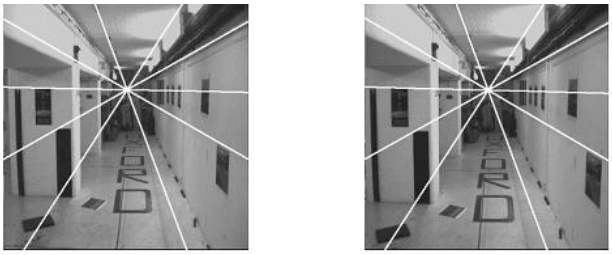
m_x, m_y : jumlah piksel persatuan fisik (satuan piksel/*length*)

x_0, y_0 : titik tengah / *principal point* citra (satuan piksel)

a_x, a_y : *focal length* kamera (satuan piksel)

2.9 Ambiguitas Rekonstruksi

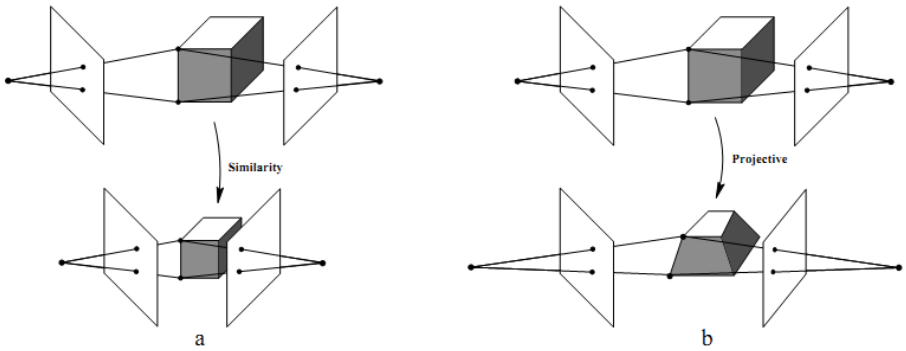
Tanpa adanya pengetahuan tentang penempatan dan posisi scene terhadap koordinat frame 3D, maka secara umum rekonstruksi posisi dan orientasi absolut suatu scene tidak mungkin dapat dilakukan dari sepasang gambar (bahkan dari banyak gambar sekalipun). Hal ini benar jika memang tidak tersedianya informasi parameter internal kamera maupun penempatan dan posisi relatif objek. Sebagai contoh, pada Gambar 2.15 menggambarkan sebuah koridor, namun informasi latitude dan longitude tidak dapat dihitung jika hanya dari gambar, juga tidak mungkin ditentukan kearah mana (utara-selatan atau timur-barat) koridor tersebut sebenarnya berada. Hal ini berarti bahwa sebuah scene hanya dapat direkonstruksi sebatas kepada transformasi (translasi dan rotasi) Euclidean yang relatif terhadap frame dunia. Selain itu, skala sebuah scene juga tidak dapat ditentukan. Pada gambar sebuah koridor, lebar pintu pada gambar tersebut tidak dapat ditentukan, apakah itu 1 meter atau 2 meter. Bahkan bisa saja bahwa pada kenyataanya gambar koridor tersebut hanyalah sebuah rumah-rumahan yang lebarnya sekitar 20 cm. Informasi tambahan seperti ini adalah sebuah contoh suatu cabang pengetahuan dari *scene* yang tidak terdapat pada citra. Tanpa pengetahuan tersebut, sebuah *scene* hanya dapat ditentukan atau dibuat dari citra sebatas kepada transformasi kesamaan atau *similarity transformation* (rotasi, translasi dan eskalasi).



Gambar 2.15. Sepasang citra *Koridor* dengan garis epipolar

Menurut Longuet-Higgins yang dibahas kembali oleh Hartley [8], ambiguitas juga dapat terjadi pada kamera terkalibrasi. Pada kamera terkalibrasi, rekonstruksi dapat dilakukan sebatas transformasi kesamaan (*similarity transformation*). Hal ini diilustrasikan pada Gambar 2.16a.

Jika parameter kamera tidak diketahui, dan juga posisi relatif kamera terhadap kamera lainnya tidak diketahui, maka ambiguitas rekonstruksi menjadi transformasi proyektif yang sembarang. Hal ini diilustrasikan pada Gambar 2.16b.



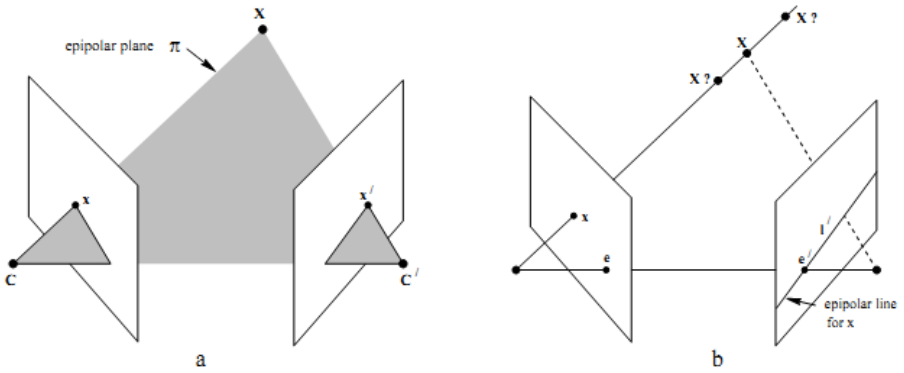
Gambar 2.16. Ambiguitas Rekonstruksi

2.10 *Epipolar Geometry*

Geometri epipolar (*epipolar geometry*) adalah sebuah geometri proyeksi intrinsik antara dua tampilan. Geometri epipolar hanya bergantung kepada parameter internal kamera dan pose relatif, tidak bergantung pada struktur *scene* [8].

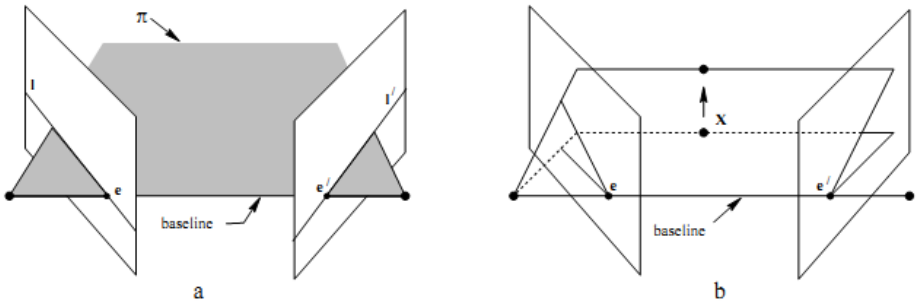
Geometri epipolar antara dua tampilan (bidang citra) pada dasarnya sebuah geometri yang menggambarkan perpotongan antara bidang gambar dengan bidang pensil dimana *baseline*-nya sebagai sumbu (*baseline* adalah garis yang menghubungkan titik tengah kamera). Pada Gambar 2.17 menjelaskan bahwa geometri epipolar berhubungan dengan korespondensi titik $x \leftrightarrow x'$ pada dua citra

dalam satu bidang epipolar π. *Baseline* yang berpotongan dengan bidang citra membentuk epipole e, e', dan bidang epipolar π yang berpotongan dengan bidang gambar membentuk garis epipolar l, l'.



Gambar 2.17. Geometri korespondensi titik

Dimisalkan sebuah titik X pada ruang dimensi tiga dicitrakan pada dua tampilan, x pada citra pertama, dan x' pada citra kedua. Terdapat relasi antara titik yang bersesuaian x dan x'. Seperti pada Gambar 2.17, titik citra x dan x', titik pada dimensi tiga X, dan pusat kamera adalah sebidang (*coplanar*). Bidang ini dinotasikan π. Sehingga, sinar yang diproyeksikan kembali dari citra, x dan x', berpotongan di X, dan sinarnya adalah sebidang pada π.



Gambar 2.18. Geometri epipolar

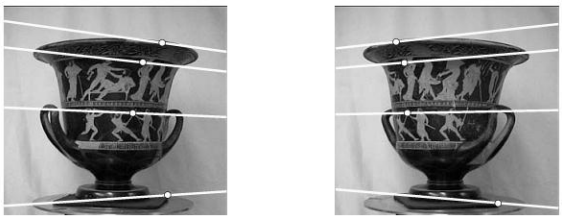
Jika yang diketahui hanya x, maka terdapat batasan untuk menemukan titik yang bersesuaian x'. Bidang π ditentukan oleh *baseline* dan sinar sebagai x. Dari Gambar 2.18 dapat diketahui bahwa sinar yang merupakan titik yang belum diketahui x' berada pada π, juga titik x' berada pada perpotongan garis l' dari π dengan bidang citra kedua. Garis l' ini adalah citra pada tampilan kedua dari sinar yang diproyeksikan kembali dari x. Garis tersebut adalah garis epipolar (*epipolar*

line) terhadap x. Sehingga untuk mencari titik yang bersesuaian x' pada citra kedua tidak dilakukan pada seluruh bidang citra, namun dapat dibatasi dalam garis l'.

Terminologi geometri epipolar berdasarkan Gambar 2.18 adalah sebagai berikut [8]:

a. *Epipole* adalah titik perpotongan dari garis yang menghubungkan pusat kamera (*baseline*) dengan bidang citra. Dengan kata lain, epipole adalah sebuah tampilan pusat kamera lainnya pada citra. Itu juga disebut sebagai titik hilang dari arah *baseline*.
b. *Epipolar plane* atau bidang epipolar adalah sebuah bidang yang didalamnya terdapat *baseline*. Terdapat sebuah pensil dari bidang epipolar.
c. *Epipolar line* atau garis epipolar adalah perpotongan bidang epipolar dengan bidang citra. Semua garis epipolar berpotongan dengan epipole. Bidang epipolar berpotongan dengan bidang citra kiri dan kanan pada garis epipolar, dan menjelaskan kesesuaian antar garis.

Contoh geometri epipolar dapat dilihat pada Gambar 2.19. Sepasang citra digambarkan beserta titik-titik yang bersesuaian dan garis epipolar. Pergerakan yang terjadi antar tampilan adalah translasi dan rotasi. Pada setiap citra, arah kamera lain dapat ditengarai dari perpotongan dari pensil garis epipolar. Dalam hal ini, kedua epipole berada diluar dari citra yang terlihat.



Gambar 2.19. Contoh geometri epipolar pada sepasang citra

2.10.1 Matriks Fundamental (F)

Matriks fundamental F adalah representasi aljabar dari geometri epipolar [8]. Matriks fundamental memiliki hubungan pemetaan dari titik x citra ke garis epipolar l' di citra lain. Titik x' manapun pada citra kedua yang bersesuaian dengan x harus berada pada garis epipolar l'. Garis epipolar adalah proyeksi sinar

dari x pada citra kedua yang melalui pusat kamera C dari kamera pertama. Sehingga terdapat pemetaan $x \mapsto l'$ dari titik di citra satu ke garis epipolar yang bersesuaian di citra lainnya. Sehingga pemetaan ini adalah sebuah korelasi singular, yang mana pemetaan proyeksi dari titik ke garis, yang diwakilkan dengan matriks fundamental F.

2.10.1.1 Kondisi Kesesuaian

Sebelumnya telah dijelaskan bahwa pemetaan $x \mapsto l'$ didefinisikan oleh F. Matriks fundamental memenuhi kondisi bahwa untuk setiap pasangan korespondensi titik $x \leftrightarrow x'$ pada dua citra, maka berlaku:

$$x'^T F x = 0$$

Hal ini benar, karena jika titik x dan x' bersesuaian, maka x' berada pada garis epipolar $l' = Fx$ yang bersesuaian dengan x. Dengan kata lain, $0 = x'^T l' = x'^T$. Sehingga, jika titik citra memenuhi hubungan $x'^T F x = 0$ maka sinar yang merupakan titik-titik tersebut adalah sebidang. Ini adalah kondisi yang dibutuhkan untuk titik-titik yang ada agar bersesuaian. Dengan melihat hubungan yang ada, ini memberikan karakteristik bahwa matriks fundamental dapat diperoleh tanpa mengetahui matriks proyeksi kamera, melainkan dengan titik-titik citra yang bersesuaian. Hal ini memungkinkan F dapat dihitung dari kesesuaian citra sendiri. Masalah selanjutnya adalah berapa banyak titik yang bersesuaian yang dibutuhkan untuk menghitung matriks fundamental dari $x'^T F x = 0$.

2.10.1.2 Properti Matriks Fundamental

Jika terdapat dua citra yang diambil menggunakan kamera, maka matriks fundamental F adalah matriks unik $3x3$ yang memenuhi

$$x'^T F x = 0 \qquad (2.48)$$

untuk semua titik-titik yang bersesuaian $x \leftrightarrow x'$.

Berikut ini adalah properti dari matriks fundamental [8]:

a. *Transpose*: jika F adalah matriks fundamental dari pasangan kamera (P, P'), maka F^T adalah matriks fundamental dari pasangan kebalikannya (P', P).

b. *Epipolar lines*: untuk setiap titik x pada citra pertama, garis epipolar yang bersesuaian adalah $l' = Fx$. Hal ini juga sama pada $l = F^T x'$ yang mewakili garis epipolar yang bersesuaian dengan x' pada citra kedua.
c. *Epipole*: untuk setiap x (selain e) garis epipolar $l' = Fx$ mengandung *epipole* e'. Sehingga e' memenuhi ${e'}^T(Fx) = ({e'}^T F)x = 0$ untuk semua x. Hal ini mengikuti bahwa ${e'}^T F = 0$ dan $Fe = 0$.
d. F memiliki 7 derajat kebebasan: matriks homogen 3x3 memiliki 8 rasio independen. F juga memenuhi batasan $\det F = 0$ yang menghilangkan satu derajat kebebasan.
e. F adalah sebuah korelasi, peta proyeksi yang membawa titik ke garis. Dalam hal ini, sebuah titik pada citra pertama x berada pada garis $l' = Fx$ di citra kedua, yang merupakan garis epipolar dari x. Jika l dan l' bersesuaian maka setiap titik x pada l dipetakan ke garis l' yang sama. Hal ini berarti tidak terdapat pemetaan kebalikan (*inverse*).

2.10.1.3 Perhitungan Matriks Fundamental

Berdasarkan persamaan (2.48), matriks fundamental didefinisikan sebagai persamaan ${x'}^T Fx = 0$ untuk setiap pasangan titik yang bersesuaian $x \leftrightarrow x'$ pada dua citra. Jika diberikan sejumlah titik yang bersesuaian $x_i \leftrightarrow {x'}_i$ (setidaknya 7), persamaan (2.48) dapat digunakan untuk menghitung matriks F yang belum diketahui. Maka dengan menulis $x = (\mathrm{x}, \mathrm{y}, 1)^T$ dan $x' = (\mathrm{x}', \mathrm{y}', 1)^T$setiap titik memberikan satu persamaan linier dalam masukan F. Koefisien persamaan ini dengan mudah ditulis dalam bentuk koordinat x dan x' yang diketahui. Sehingga, persamaan yang bersesuaian dengan pasangan titik $(\mathrm{x}, \mathrm{y}, 1)$ dan $(\mathrm{x}', \mathrm{y}', 1)$ adalah

$$\mathrm{x}'\mathrm{x}f_{11} + \mathrm{x}'\mathrm{y}f_{12} + \mathrm{x}'f_{13} + \mathrm{y}'\mathrm{x}f_{21} + \mathrm{y}'\mathrm{y}f_{22} + \mathrm{y}'f_{23} + \mathrm{x}f_{31} + \mathrm{y}f_{32} + f_{33} = 0 \quad (2.49)$$

Dinotasikan dengan f, 9-vektor terdiri dari masukan F dalam urutan baris-mayor. Maka (2.17) dapat diekspresikan sebagai produk vektor *inner*

$$(\mathrm{x}'\mathrm{x},\ \mathrm{x}'\mathrm{y},\ \mathrm{x}',\ \mathrm{y}'\mathrm{x},\ \mathrm{y}'\mathrm{y},\ \mathrm{y}',\mathrm{x},\mathrm{y}, 1)f = 0$$

Dari himpunan n titik yang bersesuaian, dapat diperoleh himpunan persamaan linier dalam bentuk

$$\mathbf{Af} = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} \begin{pmatrix} f_{11} \\ f_{12} \\ f_{13} \\ f_{21} \\ f_{22} \\ f_{23} \\ f_{31} \\ f_{32} \\ f_{33} \end{pmatrix} = \mathbf{0} \tag{2.50}$$

Matriks fundamental tidak bergantung pada struktur *scene* yang ada (rotasi dan translasi) [8]. Matriks fundamental hanya memberikan batasan hubungan titik yang bersesuaian dari dua bidang citra dalam satu bidang epipolar. Oleh karena itu matriks fundamental dapat dicari hanya dengan mengetahui beberapa informasi titik yang saling bersesuaian. Teknik yang cukup akurat dan mudah digunakan adalah algoritma 8-titik [22]. Algoritma ini membutuhkan 8 pasang titik yang saling berkorespondensi antar dua citra sehingga memenuhi persamaan $Af = 0$. Untuk mencari solusi f dilakukan dekomposisi matriks A dengan metode *SVD*.

$$A = USV^T \tag{2.51}$$

Solusi matriks fundamental F adalah kolom V dengan nilai singular terkecil (atau nol) matriks S, atau dengan kata lain kolom terakhir dari V yang disusun menjadi matriks 3x3. Untuk mendapatkan nilai F terbaik, nilai singular terkecil pada matriks S diubah menjadi 0, sehingga $S(3,3) = 0$. Kemudian matriks F kembali dihitung dengan persamaan,

$$F = USV^T \tag{2.52}$$

2.10.2 Matriks Esensial (E)

Matriks esensial E adalah spesialisasi dari matriks fundamental dimana matriks ini menormalkan koordinat citra. Sejarahnya matriks esensial ini diperkenalkan (oleh Lounget-Higgins) sebelum matriks fundamental, dan matriks

fundamental diketahui sebagai generalisasi dari matriks esensial yang mana asumsi (bukan esensial) kalibrasi kamera dihilangkan [8].

Pertimbangkan sebuah matriks kamera didekomposisi sebagai $P = K[R \mid t]$, dan $x = PX$ adalah titik pada citra. Jika matriks kalibrasi K diketahui, maka dapat diterapkan inverse-nya kepada titik x untuk memperoleh $\hat{x} = K^{-1}x$. Maka $\hat{x} = [R \mid t]X$, dimana $\hat{x}$ adalah titik citra yang diekspresikan dalam bentuk koordinat yang dinormalisasi (*normalized coordinates*). Hal itu dapat dianggap sebagai citra dari titik X terhadap kamera $[R \mid t]$ memiliki matriks identitas I sebagai matriks kalibrasinya. Matriks kamera $K^{-1}P = [R|t]$ disebut matriks kamera dinormalisasi (*normalized camera matrix*), dimana matriks kalibrasi yang diketahui sudah dihilangkan.

Jika diketahui sepasang matriks kamera dinormalisasi $P = [I|0]$ dan $P' = [R|t]$. Matriks fundamental yang bersesuaian dengan pasangan matriks kamera tersebut disebut matriks esensial, yang mana memilki bentuk

$$E = [t]_{\times}R = R[R^T t]_{\times} \qquad (2.53)$$

sehingga persamaan yang mendefinisikan matriks esensial adalah

$$\hat{x}'^T E\hat{x} = 0 \qquad (2.54)$$

dalam kondisi koordinat citra dinormalisasi untuk setiap titik bersesuaian $x \leftrightarrow x'$.

Dengan mensubtitusi $\hat{x}$ dan $\hat{x}'$ memberikan $x'^T K'^{-T} E K^{-1} x = 0$. Dengan membandingkannya dengan hubungan $x'^T Fx = 0$ untuk matriks fundamental, maka hubungan antara matriks fundamental dan esensial adalah

$$E = K'^T FK \qquad (2.55)$$

Matriks fundamental bekerja pada citra dengan satuan piksel, sedangkan matriks esensial bekerja pada bidang kamera dengan satuan fisik. Matriks esensial mengandung parameter eksternal kamera, yaitu rotasi dan translasi. Namun untuk mendapatkkannya harus melakukan dekomposisi matriks esensial dengan metode *SVD*.

2.10.2.1 Ekstraksi Kamera Dari Matriks Esensial

Matriks esensial dapat dihitung secara langsung dari persamaan (2.22) menggunakan koordinat citra dinormalisasi, atau cara lainnya dihitung dari

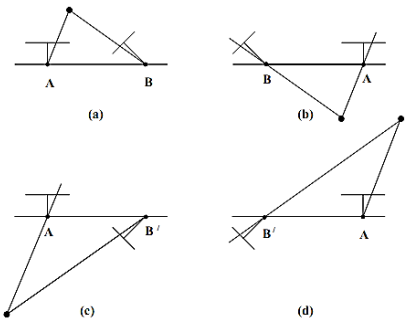
matriks fundamental menggunakan persamaan (2.23). Setelah matriks esensial diketahui, matriks kamera dapat diperoleh dari E. Berbeda dengan kasus pada matriks fundamental, dimana terdapat keambiguan proyeksi, matriks kamera dapat diperoleh dari matriks esensial sampai batas skala tertentu. Sehingga akan terdapat empat solusi yang mungkin.

Jika diasumsikan matriks kamera pertama adalah $P = [I|0]$. Untuk menghitung matriks kamera kedua P', maka E harus difaktorkan kedalam hasil SR dari matriks *skew-symmetric* dan matriks rotasi. Jika dianggap SVD dari E adalah $U\ diag(1,1,0)V^T$, maka terdapat empat kemungkinan pilihan untuk matriks kamera kedua P' yaitu

$$P' = [UWV^T \mid +u_3] \text{ atau}$$
$$[UWV^T \mid -u_3] \text{ atau}$$
$$[UW^TV^T \mid +u_3] \text{ atau}$$
$$[UW^TV^T \mid -u_3] \qquad (2.56)$$

Jelas bahwa perbedaan antara dua solusi pertama adalah arah vektor translasi dari yang pertama ke yang kedua adalah kebalikannya. Hubungan antara solusi yang pertama dan ketiga lebih rumit, yaitu merupakan rotasi sebesar 180^o terhadap garis yang menghubungkan pusat kedua kamera. Dua solusi ini disebut *twisted pair*.

Empat solusi pada Gambar 2.20, dimana digambarkan bahwa rekonstruksi titik X akan berada di depan kedua kamera hanya pada satu dari empat kemungkinan solusi yang ada. Maka pemeriksaan dengan satu titik untuk menentukan jika itu berada di depan kedua kamera diperlukan untuk menentukan diantara empat solusi berbeda untuk matriks kamera P'.



Gambar 2.20. Empat solusi kemungkinan rekonstruksi dari E [8]

2.11 Algoritma 8-titik Dinormalisasi

Algoritma 8-titik [22] adalah metode paling sederhana dalam menghitung matriks fundamental, dan melibatkan persamaan linier. Jika dilakukan dengan benar, maka algoritma ini dapat bekerja sangat baik. Algoritma ini ditemukan oleh Longuet-Higgins. Kunci sukses pada algoritma 8-titik ini adalah menerapkan normalisasi data masukan sebelum membangun penyelesaian persamaan. Dalam hal algoritma 8-titik, transformasi sederhana (translasi dan eskalasi) titik pada citra sebelum memformulasikan persamaan akan berpengaruh stabilitas hasil yang didapatkan [8].

Normalisasi yang disarankan adalah translasi dan eskalasi setiap citra sehingga *centroid* titik referensi berada pada koordinat asal (origin) dan RMS jarak titik dari origin sama dengan $\sqrt{2}$.

Adapun untuk menghitung *centroid* dapat menggunakan persamaan

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \qquad \text{dan} \qquad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{2.57}$$

dan jarak rata-rata ke *centroid* adalah

$$\bar{d} = \frac{\sum_{i=1}^{n}\sqrt{(x_i-\bar{x})^2+(y_i-\bar{y})^2}}{n} \tag{2.58}$$

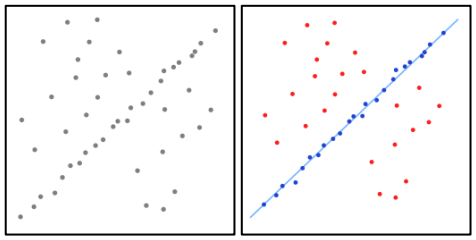
sehingga titik yang dinormalisasi menjadi

$$\begin{bmatrix}\hat{x}_i\\ \hat{y}_i\\ 1\end{bmatrix} = \begin{bmatrix}(x_i-\bar{x})/\bar{d}\\ (y_i-\bar{y})/\bar{d}\\ 1\end{bmatrix} = \begin{bmatrix}1/\bar{d} & 0 & -\bar{x}/\bar{d}\\ 0 & 1/\bar{d} & -\bar{y}/\bar{d}\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}x_i\\ y_i\\ 1\end{bmatrix} \tag{2.59}$$

2.12 *RANSAC* (Menghitung Matriks Fundamental)

RANSAC (*random sample consensus*) adalah algoritma yang diusulkan oleh Fischler dan Bolles [23] merupakan metode iteratif untuk mengestimasi parameter model matematika yang dirancang untuk mengatasi sebagian besar *outlier* dalam data masukan. Tidak seperti kebanyakan teknik estimasi umum lainnya yang *robust*, seperti *m-estimator* dan *least median square* yang diadopsi oleh komunitas *computer vision* dari literatur statistika, *RANSAC* dikembangkan dari dalam komunitas *computer vision* sendiri.

Contoh sederhana adalah pada penyesuaian garis pada ruang dua dimensi seperti pada Gambar 2.21. Diasumsikan bahwa sekumpulan titik mengandung *inliers* yang secara pendekatan tertentu dapat ditentukan sebuah garis, dan titik *outliers* yang tidak dapat digunakan untuk membuat garis. *RANSAC* dapat menghasilkan model yang hanya dihitung dari data titik *inliers*, asalkan probabilitas hanya *inliers* yang dipilih dalam pemilihan data cukup tinggi.



Gambar 2.21. Penyesuaian garis menggunakan *RANSAC*

RANSAC adalah teknik *resampling* (pengambilan sampel ulang) yang menghasilkan solusi kandidat dengan menggunakan jumlah minimum pengamatan (titik data) yang diperlukan untuk mengestimasi parameter model yang mendasari. Tidak seperti teknik *sampling* konvensional yang menggunakan sebanyak mungkin data untuk memperoleh solusi awal dan kemudian dilakukan penghilangan *outlier*, *RANSAC* menggunakan himpunan data sekecil mungkin dan berlanjut kepada penggunaan himpunan dengan titik data yang konsisten.

Algoritma *RANSAC* secara mendasar dapat dijelaskan sebagai berikut.

1. Pilih secara acak sejumlah titik minimal yang dibutuhkan untuk menentukan model parameter.
2. Selesaikan perhitungan model parameter tersebut.
3. Tentukan berapa banyak titik dari semua titik yang sesuai dengan batas toleransi yang telah ditentukan $\in$.
4. Jika jumlah *inliers* dari total jumlah titik melebihi batas yang ditentukan τ, lakukan estimasi ulang model parameter menggunakan titik-titik yang teridentifikasi sebagai *inliers* lalu hentikan.
5. Jika tidak, ulangi langkah 1 sampai 4 (maksimum *N* kali).

Hartley [8] mengusulkan teknik menghitung matriks fundamental secara otomatis menggunakan algoritma *RANSAC* bersama dengan algoritma 7-titik.

Namun teknik berikut ini akan menggunakan algoritma 8-titik untuk menyederhanakan perhitungan menjadi linier.

Perhitungan model parameter berdasarkan algoritma *RANSAC* secara umum adalah perhitungan matriks fundamental. Matriks fundamental dihitung menggunakan algoritma 8-titik dinormalisasi. *RANSAC* digunakan untuk menghitung matriks fundamental sekaligus menghilangkan *outliers* pada titik yang bersesuaian. Penggunaan algoritma *RANSAC* bersamaan dengan algoritma 8-titik dinormalisasi disebabkan karena algoritma 8-titik sendiri harus menjamin data masukan tidak mengandung *outliers*, namun pada kenyataannya hal tersebut tidak dapat dihindari. Sehingga penggunaan bersama algoritma *RANSAC* bertujuan untuk mencari titik-titik *inliers* dan menggunakannya untuk menghitung matriks fundamental.

Adapun tahapan algoritma *RANSAC* dan algoritma 8-titik dinormalisasi dalam perhitungan matriks fundamental adalah sebagai berikut.

1. Hitung titik penting p1 pada citra pertama.
2. Hitung titik bersesuaian p2 pada citra kedua.
3. Inisialisasi matriks fundamental F menjadi matriks nol 3x3.
4. Tentukan n = 0 dan jumlah iterasi N awal.
5. Ulangi langkah berikut sampai n < N
    a. Pilih 8 pasang titik bersesuaian secara acak p1 dan p2
    b. Hitung matriks fundamental, f, menggunakan 8 pasang titik tersebut menggunakan algoritma 8-titik dinormalisasi.
    c. Hitung *fitness* untuk matriks f untuk semua titik p1 dan p2

    $$d(u_i, v_i) = (v_i^T F u_i)^2 . \left(\frac{1}{(Fu_i)_1^2 + (Fu_i)_2^2} + \frac{1}{(F^T v_i)_1^2 + (F^T v_i)_2^2} \right) \quad (2.60)$$

    $$r = \frac{\sum_i^N sgn(d(u_i, v_i), t)}{N} \quad (2.61)$$

    Dimana $sgn(a, b) = 1$ jika $a \leq b$ dan 0 jika selainnya. t adalah nilai *threshold* (dalam piksel) yang ditentukan.
    d. Jika $fitness(f) > fitness(F)$ ganti F dengan f.

e. Perbarui jumlah iterasi N

$$N = \min\left(N, \frac{log(1-p)}{log(1-r^8)}\right) \quad (2.62)$$

Dimana p adalah parameter *confidence* atau tingkat keyakinan yang diinginkan untuk menemukan jumlah maskimum inliers, dengan rentang $0 - 1.0$.

f. $n = n + 1$

2.13 Algoritma *PnP*

PnP atau *perspective-n-point* adalah algoritma untuk menghitung matriks proyeksi kamera. Jika sebelumnya matriks proyeksi dihitung menggunakan dekomposisi matriks esensial dimana memerlukan hubungan titik-titik 2D yang bersesuaian, maka berbeda dengan *PnP*. *PnP* memerlukan hubungan antara titik 3D dan titik 2D citra. Umumnya hubungan ini didapat dari titik 3D yang sebelumnya pernah direkonstruksi pada citra lain. Jika citra lain tersebut memiliki hubungan titik yang bersesuaian pada citra yang akan dihitung matriks proyeksinya, maka titik 3D yang dimilikinya juga berhubungan dengan titik 2D pada citra tersebut. Algoritma ini membutuhkan setidaknya 6 titik citra dan titik 3D yang berhubungan.

Adapun tahapan perhitungan matriks proyeksi menggunakan algoritma *PnP* adalah sebagai berikut.

1. Normalisasi titik

   Normalisasi yang disarankan adalah translasi dan eskalasi setiap citra sehingga *centroid* titik referensi berada pada koordinat asal (*origin*).

   Adapun untuk menghitung *centroid* dapat menggunakan persamaan

   $$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \text{dan} \quad \bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \quad (2.63)$$

   dan jarak rata-rata ke *centroid* adalah

   $$\bar{d} = \frac{\sum_{i=1}^{n}\sqrt{(x_i - \bar{x})^2 + (y_i - \bar{y})^2}}{n} \quad (2.64)$$

sehingga titik yang dinormalisasi menjadi

$$\begin{bmatrix}\hat{x}_i\\ \hat{y}_i\\ 1\end{bmatrix} = \begin{bmatrix}(x_i-\bar{x})/\bar{d}\\ (y_i-\bar{y})/\bar{d}\\ 1\end{bmatrix} = T\begin{bmatrix}x_i\\ y_i\\ 1\end{bmatrix} \tag{2.65}$$

$$T = \begin{bmatrix}1/\bar{d} & 0 & -\bar{x}/\bar{d}\\ 0 & 1/\bar{d} & -\bar{y}/\bar{d}\\ 0 & 0 & 1\end{bmatrix} \tag{2.66}$$

Sedangkan normalisasi titik 3D dilakukan sebagai berikut.

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \ , \ \bar{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i \quad \text{dan} \quad \bar{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i \tag{2.67}$$

$$\bar{d} = \frac{\sum_{i=1}^{n}\sqrt{(X_i-\bar{X})^2+(Y_i-\bar{Y})^2+(Z_i-\bar{Z})^2}}{n} \tag{2.68}$$

sehingga titik yang dinormalisasi menjadi

$$\begin{bmatrix}\hat{X}_i\\ \hat{Y}_i\\ \hat{Z}_i\\ 1\end{bmatrix} = \begin{bmatrix}(X_i-\bar{X})/\bar{d}\\ (Y_i-\bar{Y})/\bar{d}\\ (Z_i-\bar{Z})/\bar{d}\\ 1\end{bmatrix} = U\begin{bmatrix}X_i\\ Y_i\\ Z_i\\ 1\end{bmatrix} \tag{2.69}$$

$$U = \begin{bmatrix}1/\bar{d} & 0 & 0 & -\bar{X}/\bar{d}\\ 0 & 1/\bar{d} & 0 & -\bar{Y}/\bar{d}\\ 0 & 0 & 1/\bar{d} & -\bar{Z}/\bar{d}\\ 0 & 0 & 0 & 1\end{bmatrix} \tag{2.70}$$

2. Dekomposisi *SVD* matriks A sehingga $AP = 0$

$$AP = 0 \tag{2.71}$$

$$\begin{bmatrix} X_1 & Y_1 & Z_1 & 1 & 0 & 0 & 0 & 0 & -x_1X_1 & -x_1Y_1 & -x_1Z_1 & -x_1\\ 0 & 0 & 0 & 0 & X_1 & Y_1 & Z_1 & 1 & -y_1X_1 & -y_1Y_1 & -y_1Z_1 & -y_1\\ & & & & & & & \vdots & & & & \\ & & & & & & & \vdots & & & & \\ X_i & Y_i & Z_i & 1 & 0 & 0 & 0 & 0 & -x_iX_i & -x_iY_i & -x_iX_i & -x_i\\ 0 & 0 & 0 & 0 & X_i & Y_i & Z_i & 1 & -y_iX_i & -y_iY_i & -y_iZ_i & -y_i \end{bmatrix}\begin{bmatrix}p_{11}\\ p_{12}\\ p_{13}\\ p_{14}\\ p_{21}\\ p_{22}\\ p_{23}\\ p_{24}\\ p_{31}\\ p_{32}\\ p_{33}\\ p_{34}\end{bmatrix} = 0 \tag{2.72}$$

$A = USV^T$

Solusi P adalah kolom matriks V dengan nilai eigen terkecil (kolom terakhir). Kemudian P disusun ulang menjadi matriks 3x4.

3. Denormalisasi P

$P = T^{-1}PU$ (2.73)

2.14 Triangulasi

Triangulasi pada geometri epipolar adalah proses estimasi untuk mendapatkan kembali titik pada sistem koordinat dunia dari korespondensi titik pada citra [8]. Triangulasi melibatkan informasi rotasi dan translasi pada dua kamera sehingga menghasilkan titik pertemuan ketika diproyeksikan balik ke sistem dunia.

2.14.1 *Direct Linear Transformation* (*DLT*)

DLT adalah metode triangulasi sederhana yang bersifat linear. Titik yang diestimasi pada ruang 3D tidak sepenuhnya memenuhi hubungan geometris, sehingga metode estimasi ini dianggap kurang optimal. Setiap pasang titik yang bersesuaian pada dua citra dan matriks proyeksinya dapat dituliskan melalui persamaan $x = PX$ dan $x' = P'X$, kemudian dua persamaan tersebut dapat digabungkan kedalam bentuk persamaan $AX = 0$, yang mana persamaan linear di X. Solusi X dapat dicari dengan dekomposisi matriks A menggunakan *SVD*.

Adapun langkah triangulasi adalah sebagai berikut.

1. Hitung matriks A , dimana x adalah x_{cam} dan p^{iT} adalah baris ke i pada matriks proyeksi P.

$$A = \begin{bmatrix} xp^{3T} - p^{1T} \\ yp^{3T} - p^{2T} \\ x'p'^{3T} - p'^{1T} \\ y'p'^{3T} - p'^{2T} \end{bmatrix} \quad (2.74)$$

2. Normalisasi setiap baris A, dimana $A1$ adalah baris pertama, $A2$ baris kedua, dan seterusnya.

$norm(Z) = \sqrt{{z_1}^2 + {z_2}^2 + {z_3}^2 + {z_4}^2}$ (2.75)

$$A1 = \left[\frac{a_{11}}{norm(A1)} \quad \frac{a_{12}}{norm(A1)} \quad \frac{a_{13}}{norm(A1)} \quad \frac{a_{14}}{norm(A1)}\right]$$

$$A2 = \left[\frac{a_{21}}{norm(A2)} \quad \frac{a_{22}}{norm(A2)} \quad \frac{a_{23}}{norm(A2)} \quad \frac{a_{24}}{norm(A2)}\right]$$

$$A3 = \left[\frac{a_{31}}{norm(A3)} \quad \frac{a_{32}}{norm(A3)} \quad \frac{a_{33}}{norm(A3)} \quad \frac{a_{34}}{norm(A3)}\right]$$

$$A4 = \left[\frac{a_{41}}{norm(A4)} \quad \frac{a_{42}}{norm(A4)} \quad \frac{a_{43}}{norm(A4)} \quad \frac{a_{44}}{norm(A4)}\right] \qquad (2.76)$$

$$A = \begin{bmatrix} A1 \\ A2 \\ A3 \\ A4 \end{bmatrix} \qquad (2.77)$$

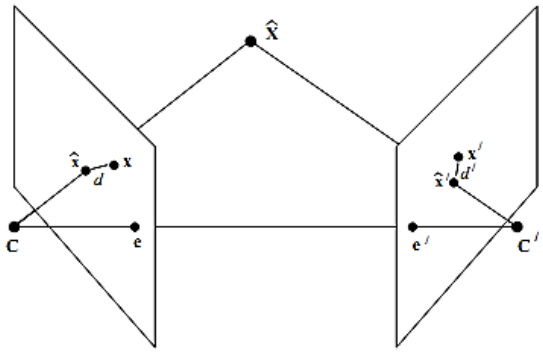
3. Dekomposisi *SVD* matriks A menjadi $A = USV^T$
4. Pilih kolom V dengan vektor eigen terkecil (kolom terakhir).

$$P = \begin{bmatrix} v_{14} \\ v_{24} \\ v_{34} \\ v_{44} \end{bmatrix} \qquad (2.78)$$

5. Normalisasi P dengan membagi masing-masing elemen dengan elemen terakhir, v_{44}, dan elemen terakhir dihilangkan sehingga P berukuran 3x1.

$$P = \begin{bmatrix} v_{14}/v_{44} \\ v_{24}/v_{44} \\ v_{34}/v_{44} \end{bmatrix} \qquad (2.79)$$

2.15 *Sampson Approximation*



Gambar 2.22. *Error* geometri

Dari berbagai hasil pengamatan, korespondensi titik $x \leftrightarrow x'$ tidak memenuhi batasan epipolar, seperti Gambar 2.22. Pada kenyataannya titik yang

tepat seharusnya berada pada korespondensi titik $\hat{x} \leftrightarrow \hat{x}'$ yang mendekati titik yang diestimasi $x \leftrightarrow x'$ dan memenuhi batasan epipolar tepat $\bar{x}'^T F \bar{x} = 0$.

Aproksimasi *sampson* melibatkan matriks fundamental dan pasangan titik yang berkorespondensi pada dua citra. Hasil dari aproksimasi ini adalah korespondensi titik yang baru yang mendekati titik proyeksi sebenarnya dan mendekati batasan epipolar. Aproksimasi titik menggunakan teknik sampson dapat dilakukan dengan menggunakan persamaan berikut ini [8],

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{x}' \\ \hat{y}' \end{bmatrix} = \begin{bmatrix} x \\ y \\ x' \\ y' \end{bmatrix} - \frac{x_2^T F x_1}{(Fx_1)_1^2 + (Fx_1)_2^2 + (F^T x_2)_1^2 + (F^T x_2)_2^2} \begin{bmatrix} (F^T x_2)_1 \\ (F^T x_2)_2 \\ (Fx_1)_1 \\ (Fx_1)_2 \end{bmatrix} \qquad (2.80)$$

dimana $F^T x_2 = f_{11}x_2 + f_{21}y_2 + f_{31}$

2.16 *Geometric Error Cost Function*

Perhitungan *error* dibutuhkan untuk mengetahui akurasi dari metode triangulasi yang digunakan. Korespondensi titik pada dua citra yang didapatkan dari hasil pencarian kesamaan fitur merupakan titik perkiraan yang digunakan untuk memproyeksikan balik titik estimasi dari citra ke ruang 3D. Setelah titik estimasi pada ruang 3D diperoleh dengan triangulasi, maka dengan memproyeksikan kembali titik estimasi 3D tersebut ke 2D, dapat diperoleh selisih antara titik perkiraan sebelumnya dengan titik proyeksi yang baru. Perbedaan yang terjadi merupakan sebuah *error* reproyeksi, seperti pada Gambar 2.22. Hal ini sangat sering terjadi karena *noise* pada citra. Untuk menghitung *error* reproyeksi tersebut dapat digunakan fungsi *cost*. Fungsi ini menghitung jarak *eucledian* antara dua titik, titik perkiraan di awal dengan titik hasil proyeksi ulang. Adapun *cost function* untuk menghitung *error* tersebut disajikan dalam persamaan berikut [8]:

$$C(x, x') = d(x, \hat{x})^2 + d(x', \hat{x}')^2 \qquad (2.81)$$

dan *error* reproyeksi dihitung menggunakan nilai rerata *cost function* semua titik perkiraan dan titik estimasi (reproyeksi) sebagai berikut

$$Err = \frac{1}{2n} \sum_{i=1}^{n} (d(x, \hat{x})^2 + d(x', \hat{x}')^2) \qquad (2.82)$$

dimana,

n : jumlah titik 3D

x, x' : titik bersesuaian pada citra

$\hat{x}, \hat{x}'$: titik reproyeksi

2.17 Dekomposisi Matriks *SVD*

Dalam teori matriks, dikenal beberapa teorema dekomposisi, di antaranya teorema faktorisasi *LU* dan teorema faktorisasi *QR*. Selanjutnya, terdapat dekomposisi yang dikenal dengan Dekomposisi Nilai Singular (*Singular Value Decomposition* atau *SVD*). *SVD* terkait dengan nilai eigen dan nilai singular [13].

Suatu proses dekomposisi akan memfaktorkan sebuah matriks menjadi lebih dari satu matriks. Demikian halnya dengan Dekomposisi Nilai Singular (*Singular Value Decomposition*) atau yang lebih dikenal sebagai *SVD*, adalah salah satu teknik dekomposisi berkaitan dengan nilai singular (*singular value*) suatu matriks yang merupakan salah satu karakteristik matriks tersebut [13].
Dekomposisi nilai singular matriks riil A_{mxn} adalah faktorisasi

$$A = USV^T \tag{2.83}$$

Teorema tersebut juga menyatakan bahwa matriks A_{mxn} dapat dinyatakan sebagai dekomposisi matriks yaitu matriks U, S dan V. Matriks $\sum$ merupakan matriks diagonal dengan elemen diagonalnya berupa nilai-nilai singular matriks A, sedangkan matriks U dan V merupakan matriks-matriks yang kolom-kolomnya berupa vektor singular kiri dan vektor singular kanan dari matriks A untuk nilai singular yang bersesuaian.

Menentukan *SVD* meliputi langkah-langkah menentukan nilai eigen dan vektor eigen dari matriks AA^T atau A^TA. Vektor eigen dari A^TA membentuk kolom V, sedangkan vektor eigen dari AA^T membentuk kolom U. Nilai singular dalam S adalah akar pangkat dua dari nilai-nilai eigen matriks AA^T atau A^TA. Nilai singular adalah elemen-elemen diagonal dari S dan disusun dengan urutan menurun.

2.18 *Matlab*

Matlab (*matrix laboratory*) adalah sebuah lingkungan komputasi numerikal dan bahasa pemrograman komputer generasi keempat. Dikembangkan oleh *The MathWorks*, *Matlab* memungkinkan manipulasi matriks, pem-*plot*-an fungsi dan data, implementasi algoritma, pembuatan antarmuka pengguna, dan peng-antarmuka-an dengan program dalam bahasa lainnya. Meskipun hanya bernuansa numerik, sebuah *toolbox* yang menggunakan mesin simbolik *MuPAD*, memungkinkan akses terhadap kemampuan aljabar komputer. Sebuah paket tambahan, Simulink, menambahkan simulasi grafis multiranah dan *model-based design* untuk sistem *embedded* dan dinamis.

Matlab menyediakan berbagai metode komputasi numerik untuk data analisis, pengembangan algoritma, dan pembuatan model. Bahasa *Matlab* mencakup fungsi matematika yang mendukung operasi *scientific* dan teknik secara umum. Fungsi matematika utama menggunakan pustaka prosesor-teroptimasi untuk menyediakan eksekusi vektor dan kalkulasi matriks secara cepat. *Matlab* juga menyediakan fungsi khusus seperti statistik, optimasi, *signal analysis*, dan *machine learning*. Metode komputasi numerik pada *Matlab* mencakup [24]:

1. Interpolasi dan regresi
2. Diferensiasi dan integrasi
3. Persamaan sistem linear
4. Analisis *fourier*
5. *Eigen* dan *singular value*
6. *Ordinary differential equations* (ODEs)
7. Matiks jarang

Matlab menyediakan peralatan untuk akuisisi, analisis, dan visualisasi data, sehingga membuat pengguna dapat mengetahui lebih dalam tentang data yang ada dibandingkan dengan menggunakan *spreadsheet* atau bahasa pemrograman tradisional lain. *Matlab* menyediakan kemampuan untuk mendokumentasikan dan membagikan hasilnya menjadi plot dan laporan atau dipublikasi menjadi kode *Matlab*.

Matlab memungkinkan untuk mengelola, menyaring, dan *preprocess* data. Pengguna dapat melakukan analisis data eksplorasi untuk mengungkap tren, uji asumsi, dan membangun model deskriptif. *Matlab* menyediakan fungsi untuk *filtering*, *smoothing*, interpolasi, konvolusi, dan *Fourier Fast Trasform* (*FFTs*). Selain itu, *Matlab* menyediakan kemampuan untuk *curve fitting*, statistik *multivariat*, analisis spektral, analisis citra, identifikasi sistem, dan tugas analisis lainnya.

Matlab menyediakan fungsi untuk memplot 2D dan 3D, serta visualisasi volume. Fungsi tersebut digunakan untuk memvisualisasikan dan memahami data dan mengkomunikasikan hasilnya. Plot dapat disesuaikan baik secara interaktif maupun dengan pemrograman.

Matlab menyediakan bahsa dan alat pengembangan tingkat tinggi yang memungkinkan pengguna dengan cepat mengembangkan dan menganalisis algoritma dan aplikasi. Bahasa *MATLAB* menyediakan dukungan asli untuk vektor dan matriks operasi yang penting untuk memecahkan masalah ilmiah dan teknik, memungkinkan pengembangan cepat dan eksekusi. Dengan bahasa *MATLAB*, penulisan program dan pengembangan algoritma lebih cepat dibandingkan dengan bahasa tradisional karena tidak perlu melakukan tugas-tugas seperti pada bahasa pemrograman tingkat rendah seperti mendeklarasikan variabel, menentukan tipe data, dan mengalokasikan memori. Dalam banyak kasus, dukungan untuk vektor dan matriks operasi menghilangkan kebutuhan untuk iterasi *for-loop*. Oleh karena itu, satu baris kode *MATLAB* sering dapat menggantikan beberapa baris kode *C* atau *C++*.

MATLAB menyediakan fitur bahasa pemrograman tradisional, termasuk kontrol aliran, penanganan *error*, dan pemrograman berorientasi objek (*OOP*). Pengguna dapat menggunakan tipe data dasar atau struktur data tingkat lanjut, atau menentukan tipe data kustom.

Pengguna dapat menghasilkan hasil secara langsung dengan mengeksekusi perintah satu per satu secara interaktif. Pendekatan ini memungkinkan Anda dengan cepat menjelajahi beberapa pilihan dan iterasi untuk solusi optimal.

Pengguna dapat menangkap langkah interaktif menjadi sebuah *script* dan fungsi untuk menggunakan kembali dan mengotomatisasi pekerjaan.

Matlab menyediakan algoritma untuk pemrosesan sinyal dan komunikasi, gambar dan pemrosesan video, sistem kontrol, dan banyak domain lainnya. Dengan menggabungkan algoritma ini, pengguna dapat membangun program kompleks dan aplikasi.

Matlab mencakup berbagai alat untuk pengembangan algoritma yang efisien, mencakup:

1. *Command Window*, memungkinkan pengguna secara interaktif memasukkan data, menjalankan perintah dan program, dan menampilkan hasil.
2. *Matlab Editor*, menyediakan fitur *editing* dan *debugging*, seperti pengaturan *breakpoints* dan melangkah melalui jalur individu kode.
3. *Code Analyzer*, secara otomatis memeriksa kode untuk masalah dan merekomendasikan modifikasi untuk memaksimalkan kinerja dan perawatan.
4. *Matlab Profiler*, mengukur kinerja program *Matlab* dan mengidentifikasi bidang kode untuk memodifikasi untuk perbaikan.

Matlab menyediakan berbagai pilihan untuk mengembangkan dan menyebarkan aplikasi. Pengguna dapat berbagi algoritma individu dan aplikasi dengan pengguna *Matlab* lain atau menempatkannya menjadi bebas royalti kepada orang lain yang tidak memiliki *Matlab*.

GUIDE (*Graphical User Interface Development Environment*) memungkinkan pengguna dapat membuat, desain, dan mengedit kustom grafis antarmuka pengguna. Pengguna dapat menyertakan kontrol umum seperti *list box*, *pull-down menu*, dan tombol, serta *Matlab plot*. Grafis antarmuka pengguna juga dapat dibuat dengan pemrograman menggunakan fungsi *Matlab*.

Untuk mendistribusikan aplikasi langsung ke pengguna *Matlab* lain, pengguna dapat mempaketnya sebagai aplikasi *Matlab*, yang menyediakan satu file untuk distribusi. Untuk berbagi aplikasi dengan orang lain yang tidak memiliki *Matlab*, pengguna dapat menggunakan produk penyebaran aplikasi.

Produk tambahan tersebut secara otomatis menghasilkan aplikasi mandiri, *shared library*, dan komponen perangkat lunak untuk integrasi dalam *C, C ++*, *Java*, *.NET*, dan lingkungan *Excel*. Berkas *executables* dan komponennya dapat didistribusikan dengan bebas royalti.

Pengguna dapat menggunakan *Matlab Coder* ™ untuk menghasilkan kode *C* mandiri dari kode *Matlab*. *Matlab Coder* mendukung subset dari bahasa *Matlab* biasanya digunakan oleh para insinyur desain untuk mengembangkan algoritma sebagai komponen dari sistem yang lebih besar. Kode ini dapat digunakan untuk eksekusi mandiri, untuk integrasi dengan aplikasi perangkat lunak lain, atau sebagai bagian dari suatu aplikasi yang tertanam.

2.19 Metode *Object Oriented*

Object Oriented (OO) merupakan paradigma baru dalam rekayasa *software* yang didasarkan pada objek dan kelas [25]. *Object oriented* merupakan metodologi terbaik dalam rekayasa *software* dan mencakup bidang aplikasi yang sangat luas.

Karena luasnya cakupan *object oriented* maka terdapat beberapa hal yang membingungkan berkenaan dengan istilah dan konsep *object oriented*. Istilah dan konsep yang berkenaan dengan dengan *object oriented* ini adalah:

1. *Object Oriented Analysis* adalah metode analisis yang memeriksa *requirement* (syarat/keperluan yang harus dipenuhi suatu sistem) dari sudut pandang kelaskelas dan objek-objek yang ditemui dalam ruang lingkup permasalahan.
2. *Object Oriented Design* adalah metode untuk mengarahkan arsitektur *software* yang didasarkan pada manipulasi objek-objek sistem atau subsistem.

Object oriented analysis and desain merupakan pendekatan yang menekankan pada solusi *logic* berbasis objek. Beberapa konsep dasar dalam OOAD:

1. Objek, adalah “benda”, secara fisik atau konseptual dapat kita temui disekeliling kita. *Hardware*, *software*, dokumen, manusia dan bahkan konsep semuanya adalah konsep objek. Sebuah objek memiliki

keadaan sesaat (*state*) dan perilaku (*behaviour*). *State* dari sebuah objek adalah kondisi objek tersebut atau himpunan dari keadaan yang menggambarkan objek tersebut. Sedangkan *behaviour* suatu objek mendefinisikan bagaimana sebuah objek bertindak dan memberi reaksi. *Behaviour* ditentukan oleh himpunan semua atau beberapa operasi yang dapat dilakukan dalam objek itu sendiri.

2. Kelas, adalah definisi umum untuk himpunan objek sejenis. Kelas menetapkan sfesifikasi prilaku dan atibut objek-objek tersebut.

Object oriented merupakan metode yang paling baik dalam rekayasa *software* diantaranya *procedure-oriented, object-oriented, data structure-oriented, data flow-oriented,* dan *constraint-oriented.* Sehingga dengan metode *object oriented* ini dapat diaplikasikan dalam seluruh ruang lingkup rekayasa *software*. *Object oriented* memandang *software* bagian per bagian, dan menggambarkan satu bagian tersebut dalam satu objek. Satu objek dalam sebuah model merupakan suatu fokus selama dalam proses analisis, desain, dan implementasi dengan menekankan pada *state,* perilaku (*behavior*), dan interaksi objek-objek dalam model tersebut.

Konsep *object oriented* memiliki karakteristik utama yaitu:

1. Enkapsulasi (*Encapsulation*), sebuah prinsip yang digunakan ketika membangun struktur program secara keseluruhan yang mana setiap komponen dari program dibungkus (pembungkusan atribut dan metode dalam satu *event*).
2. Pewarisan (*Inheritance*), merupakan sarana untuk menghilangkan penulisan ulang terhadap kode yang dapat digunakan berulang kali.
3. Kebanyakrupaan (*Polimorpishm*), suatu kondisi dimana dua objek atau lebih mempunyai antarmuka yang identik namun mempunyai perilaku yang berbeda.

2.20 *UML* (*Unified Modeling Language*)

1. Pengertian *UML*

UML adalah sebuah bahasa untuk menetukan, visualisasi, kontruksi, dan mendokumentasikan *artifact* bagian dari informasi yang digunakan atau

dihasilkan dalam suatu proses pembuatan perangkat lunak. *Artifact* dapat berupa model, deskripsi atau perangkat lunak dari sistem perangkat lunak, seperti pada pemodelan bisnis dan sistem non perangkat lunak lainnya [26].

UML merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan sistem yang besar dan kompleks. *UML* tidak hanya digunakan dalam proses pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.

2. Bagian-bagian *UML*

Bagian-bagian utama dari *UML* adalah *view*, *diagram*, *model element*, dan *general mechanism* [26].

a. *View*

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis view dalam *UML* antara lain: *use case view*, *logical view*, *component view*, *concurrency view*, dan *deployment view*.

1) *Use case view*

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. *Actor* yang berinteraksi dengan sistem dapat berupa user atau sistem lainnya. *View* ini digambarkan dalam *use case diagrams* dan kadang-kadang dengan *activity diagrams*. *View* ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

2) *Logical view*

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class*, *object*, dan *relationship*) dan kolaborasi dinamis yang terjadi ketika *object* mengirim pesan ke *object* lain dalam suatu fungsi tertentu. *View* ini digambarkan dalam *class diagrams* untuk struktur statis dan dalam *state*, *sequence*, *collaboration*, dan *activity diagram*

untuk model dinamisnya. *View* ini digunakan untuk perancang (*designer*) dan pengembang (*developer*).

3) *Component view*

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari *code module* diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi *administrative* lainnya. *View* ini digambarkan dalam *component view* dan digunakan untuk pengembang (*developer*).

4) *Concurrency view*

Membagi sistem ke dalam proses dan prosesor. *View* ini digambarkan dalam diagram dinamis (*state*, *sequence*, *collaboration*, dan *activity diagrams*) dan diagram implementasi (*component* dan *deployment diagrams*) serta digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

5) *Deployment view*

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. *View* ini digambarkan dalam *deployment diagrams* dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

b. *Diagram*

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk *view* tertentu. Adapun jenis *diagram* antara lain:

1) *Use Case Diagram*

Use case adalah abstraksi dari interaksi antara sistem dan *actor*. *Use case* bekerja dengan cara mendeskripsikan tipe interaksi antara user sebuah sistem dengan sistemnya sendiri melalui sebuah cerita

bagaimana sebuah sistem dipakai. *Use case* merupakan konstruksi untuk mendeskripsikan bagaimana sistem akan terlihat di mata user. Sedangkan *use case diagram* memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan *client*.

2) *Class Diagram*

Class adalah dekripsi kelompok objek-objek dengan *property*, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class diagram* dapat memberikan pandangan global atas sebuah sistem. Hal tersebut tercermin dari *class-class* yang ada dan relasinya satu dengan yang lainnya. Sebuah sistem biasanya mempunyai beberapa *class diagram*. *Class diagram* sangat membantu dalam visualisasi struktur kelas dari suatu sistem.

3) *Component Diagram*

Component software merupakan bagian fisik dari sebuah sistem, karena menetap di komputer tidak berada di benak para analis. Komponen merupakan implementasi *software* dari sebuah atau lebih *class*. Komponen dapat berupa *source code*, komponen biner, atau *executable component*. Sebuah komponen berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*. Sehingga *component diagram* merepresentasikan dunia nyata yaitu component *software* yang mengandung *component*, *interface* dan *relationship*.

4) *Deployment Diagram*

Menggambarkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware*, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executable* component dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh node tertentu dan ketergantungan komponen.

5) *State Diagram*

Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu *object* dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa *object* lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh state yang berbeda.

6) *Sequence Diagram*

Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah *scenario*. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.

7) *Collaboration Diagram*

Menggambarkan kolaborasi dinamis seperti *sequence diagrams*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*.

8) *Activity Diagram*

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi.