BAB 2
LANDASAN TEORI

2.1. *Game*

Game adalah suatu aktifitas yang tidak dilakukan dengan sungguh sungguh [8]. Pengertian *Game* menurut beberapa ahli [9] diantaranya adalah sebagai berikut:

a. Menurut Ivan C Sibero, *Game* merupakan salah satu aplikasi yang paling banyak dipakai dan dinikmati oleh pengguna media elektronik saat ini.
b. Menurut Samuel Henry *Game* adalah sesuatu yang tidak bisa dipisahkan dari keseharian anak anak.
c. Menurut Fauzia A *Game* adalah salah satu bentuk hiburan yang dapat dijadikan sebagai penyegar pikiran dari kepenatan akibat dari padatnya aktivitas sehari hari.
d. Menurut John Naisbitt *Game* merupakan sistem partisipatoris dinamis karena *Game* memiliki tingkat penceritaan yang tidak dimiliki film.
e. Menurut Wiaya Ariyana dan Arifianto *Game* adalah salah satu kebutuhan sekaligus menjadi masalah besar bagi para pengguna komputer, hal tersebut dikarenakan untuk dapat menjalankan suatu *Game* dengan nyaman, maka dibutuhkan spesifikasi komputer yang relatif tinggi dibanding komputer untuk penggunaan biasa.

Dari beberapa pendapat yang dikemukakan oleh para ahli , dapat disimpulkan bahwa *Game* adalah sebuah kebutuhan untuk menghibur dari kepenatan dan tidak dapat dipisahkan dari keseharian anak anak.

2.2. Monopoli

Monopoli adalah salah satu papan permainan yang paling terkenal di dunia, dengan tujuan untuk menguasai semua petak di dalam sistem ekonomi yang disediakan. Awal mula dari permainan monopoli sebenarnya dimaksudkan untuk mendemonstrasikan kejahatan dari pemilik tanah dan mempermudah orang

mengerti bagaimana pemilik tanah memperkaya dirinya dan mempermiskin para penyewa [10]. Permainan monopoli yang standar biasanya memiliki peralatan sebagai berikut:

a. Kartu, Sebanyak 32 kartu kesempatan dan dana umum (16 kartu kesempatan dan 16 kartu dana umum). Dimana pemain mengambil kartu tersebut ketika berada pada petak kesempatan atau dana umum.
b. Properti, 22 properti yang dapat dikembangkan dengan rumah atau hotel, 4 jalur kereta api, 2 utilitas (*utility*).
c. Dadu, sepasang atau 2 buah 6 sisi dadu, pada c penggunaan dadu tergantung dalam waktu dan kondisi tertentu. Seperti perang dunia ke II, dadu digantikan dengan papan putar atau *Spinner* dikarenakan terbatasnya material dari dadu pada jaman tersebut .
d. Rumah dan Hotel, 32 rumah dan 12 hotel terbuat dari kayu dan plastik.
e. Uang, pada permainan monopoli lama terdapat uang dengan total $15,140, sedangkan pada permainan monopoli baru meningkat menjadi $20,580.
f. Serta bidak atau *tokens*, sebagai representasi dari pemain.

2.2.1. Gameplay

Cara memainkan monopoli berbeda beda pada tiap negara maupun tiap daerah. Ada yang menambah peraturannya sendiri dan ada yang mengurangi peraturan yang sudah ada. Peraturan yang resmi dikeluarkan oleh *Parker Brothers* dari artikel "*Monopoly Parker Brothers Real Estate Trading Game* " [11]. itu sendiri adalah sebagai berikut:

1. *The Play*, dimulai dari *Banker*, setiap pemain dalam giliran melempar dadu. Pemain yang mendapatkan nilai tertinggi dari lemparan dadunya, dapat memulai permainanannya dimulai dari pemain tersebut. Taruh bidak pada sisi yang bertanda "*GO*", lempar dadu dan gerakan bidak sesuai dengan nilai pada lemparan dadunya. Setelah giliran pemain selesai dilanjutkan dengan pemain di sebelah kiri pemain tersebut. Bidaknya akan tetap berada di tempat dimana bidak tersebut berdiri dan melanjutkan dari posisi

tersebut ketika giliran pemain itu kembali bermain. Dua atau lebih bidak dapat ditempatkan di posisi yang sama pada waktu yang bersamaan.

2. *GO*, ketika bidak pemain berada atau melewati *GO*, pemain dapat melempar dadu atau mengambil kartu serta *banker* memberi uang sebanyak $200. Uang tersebut diberikan kepada pemain setiap kali melewati atau berada di *GO*, pemain bisa saja mendapatkan uang sebanyak dua kali lipat apabila setelah melewati *GO* dan berhenti di dana umum atau kesempatan yang kemudian mendapatkan kartu untuk maju sampai *GO*.
3. Membeli properti, ketika suatu pemain berada pada properti yang tidak dimiliki oleh pemain lain, pemain tersebut dapat membeli properti tersebut dengan cara membayar uang yang tertera pada harga properti tersebut kepada *Banker*. Pemain yang berada pada properti yang tidak dimiliki oleh pemain lain dan tidak membeli properti tersebut. Maka *banker* akan melakukan pelelangan terhadap properti tersebut, pelelangan dapat dimulai pada harga berapapun.
4. Membayar Sewa, ketika suatu pemain berada pada properti yang dimiliki oleh pemain lain, pemain tersebut harus membayar uang sewa kepada pemain yang memiliki properti tersebut. Apabila properti yang dimiliki digadaikan, maka pemain yang berada pada properti tersebut tidak mengeluarkan uang sewa kepada pemain yang memilikinya. Ketika pemain memiliki semua properti dengan warna group yang sama, maka pemain tersebut dapat menaikan uang sewa menjadi dua kali lipat. Peraturan ini berlaku bagi properti yang tidak digadaikan, walaupun kartu lainya dengan warna yang sama digadaikan.
5. Dana Umum dan Kesempatan, ketika berada pada tempat dana umum atau kesempatan, ambil kartu paling atas dan ikuti petunjuk yang tertera. Kemudian taruh kembali kartu tersebut paling bawah dari tumpukan dimana kartu tersebut diambil.
6. Pajak penghasilan, ketika seorang pemain berada di pajak penghasilan atau *Income Tax* maka pemain dapat membayar pajak sebesar $200 atau 10% dari uang yang dimiliki kepada *banker*.

7. Penjara, seorang pemain akan berada dipenjara apabila bidak pemain berada pada petak bertuliskan "*Go to Jail*", mengambil kartu yang bertuliskan "*Go to Jail*", atau melempar dadu kembar atau *doubles* sebanyak tiga kali berturut turut. Ketika pemain akan dikirimkan kedalam penjara, pemain tersebut tidak mendapatkan uang $200 dalam perjalanannya ke penjara, dimanapun bidak pemain berada, bidak pemain tersebut masuk penjara secara langsung. Apabila pemain yang sebelumnya tidak dipenjara dan berada di petak bertuliskan penjara atau "*Jail*", maka pemain tersebut hanya menjenguk atau melewati penjara saja. Seorang pemain akan keluar dari penjara apabila melempar dadu secara

2.3. Citra

Citra adalah kombinasi antara titik, garis, bidang dan warna untuk menciptakan suatu imitasi dari suatu objek. Citra bisa berupa gambar dua dimensi seperti lukisan, foto, dan berwujud tiga dimensi seperti patung [12]. Gambar atau Citra telah berkembang seiring dengan peradaban manusia, bentuknya tidak lagi hanya sebuah lukisan prasejarah. Sekarang ada foto yang dihasilkan dengan menangkap cahaya pada medium yang telah dilapisi bahan kimia peka cahaya atau *sensor digital* [12].

2.3.1. Pengolahan Citra

Pengolahan citra (*Image Processing*) adalah pemprosesan sinyal dimana yang menjadi masukan adalah gambar seperti foto atau *frame* dari sebuah *video*. Dengan keluaran atau hasil dari pemprosesannya bisa menjadi sebuah gambar atau sebuah karakteristik atau parameter yang berhubungan dengan gambar tersebut [13]. Dalam grafik komputer(*Computer Graphics*), gambar biasanya dibuat manual dari model objek fisik, lingkungan dan cahaya daripada mendapatkannya melalui kamera atau sejenisnya. Sedangkan pada penglihatan komputer(*Computer Vision*) lebih memilih pemprosesan gambar tingkat tinggi, dimana mesin atau komputer ditujukan untuk menguraikan konten fisik dari gambar atau urutan gambar (*Sequence Image*) [13].

2.4. Kecerdasan Buatan

Kecerdasan buatan (*Artificial Intelligence*) atau sering disebut *AI* adalah suatu ilmu yang mempelajari cara membuat komputer melakukan sesuatu seperti yang dilakukan oleh manusia [14]. Kecerdasan buatan memiliki penafsiran yang berbeda beda oleh para ahli namun memiliki makna yang sama, berikut adalah pengertian kecerdasan buatan menurut para ahli [14]:

a. Menurut H. A. Simon menjelaskan bahwa kecerdasan buatan (*Artificial Intelligence*) merupakan kawasan penelitian, aplikasi dan instruksi yang terkait dengan pemrograman komputer untuk melakukan sesuatu hal yang dalam pandangan manusia adalah cerdas.
b. Menurut Rick and Knight menjelaskan bahwa Kecerdasan buatan (*Artificial Intelligence*) merupakan sebuah studi tentang bagaimana membuat komputer melakukan hal-hal yang pada saat ini dapat dilakukan lebih baik oleh manusia, dengan tujuan utama dari kecerdasan buatan itu sendiri adalah membuat mesin menjadi lebih pintar.

2.5. Pengenalan Objek

Pengenalan objek (*Object Recognition*) adalah menemukan objek pada dunia asli dari sebuah gambar menggunakan model [15]. Algoritma pengenalan objek bergantung pada pencocokan, pembelajaran atau algoritma pengenalan pola yang menggunakan tehnik berbasis fitur. Beberapa metode yang populer dalam pengenalan objek dan pendekatannya dalam penelitian ini dapat menggunakan metode *OCR*, *Template Matching* dan yang berbasis *Feature* seperti *FAST*, *SURF* dan *SIFT*.

2.5.1. Pengenalan Karakter Optik (*Optical Character Recognition*)

Pengenalan karakter optik adalah sebuah perangkat lunak yang secara automatis menganalisa teks dan mengubah kedalam sebuah bentuk yang dapat diproses dengan mudah oleh komputer [16]. Dalam sebuah artikel yang berjudul "*Analysing and Improving OCR Accuracy in Large Scale Historic Newspaper Digitisation Programs*", metode *OCR* membutuhkan pemindaian (*Scanning*) yang memiliki resolusi 300 *DPI*(*Dot per Inch*) untuk mendapatkan informasi citra sebanyak

mungkin. Serta menggunakan format *tiff* agar tidak ada informasi yang hilang pada citra tersebut [17]. Pada Tabel 2.1 berikut menjelaskan proses serta faktor yang mempengaruhi pengenalan karakter optik tersebut [17]:

Tabel 2.1 Proses dan faktor yang mempengaruhi pengenalan karakter optik [17]

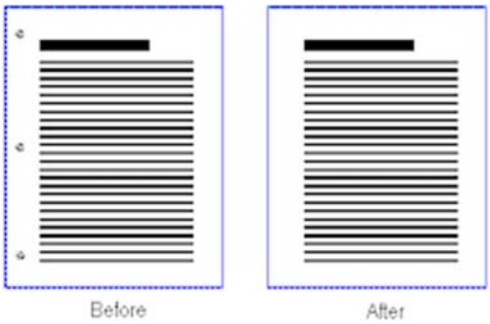
Langkah Proses	Faktor yang mempengaruhi pengenalan karakter optik	Aksi yang disarankan
Mengambil Citra asli	Kualitas dari citra	a. Menggunakan *hard copies* jika dana mencukupi . b. *Hard copies* harus bersih dari debu dan benda yang menghalangi agar dapat mengambil informasi yang banyak
Memindai file	Resolusi pemindaian dan *format* dari *file*	a. Resolusi pemindaian harus lebih atau sama dengan 300 *dot per inch* untuk mengambil informasi gambar sebanyak mungkin. b. *Format* dari *file* harus tidak menggunakan kompresi(*lossless*) agar tidak ada informasi atau data yang hilang ,misalnya *format* tiff.
Membuat kontras yang baik antara hitam dan putih di file citra	a. Kedalaman warna dari gambar. b. pengoptimasian citra atau proses mengubah menjadi biner. c. Kualitas dari sumber atau citra.	a. Memindai citra sebagai citra abu abu (*GreyScale*). b. Pengoptimasian citra untuk pengenalan karakter optik untuk meningkatkan kontras citra melalui perangkat lunak pemindaian atau perangkat lunak lainnya. c. Jika gambar adalah gambar abu abu (*greyScale*) maka ubah kedalam bentuk citra

Langkah Proses	Faktor yang mempengaruhi pengenalan karakter optik	Aksi yang disarankan
		optimasi atau pengubahan kedalam bentuk biner. d. Mengambil citra sumber yang paling baik kualitasnya.
Perangkat Lunak pengenalan karakter optik tahap *layout* dari citra	a. Citra yang tidak tegak lurus atau miring (*Skewed Images*). b. Citra yang memiliki *layout* yang kompleks atau rumit. c. Kecukupan jarak putih diantara baris, klom dan pada batas text pada sisi citra, sehingga dapat dikenali.	a. Meluruskan citra pada langkah pengolahan citra agar baris kata *horizontal* . b. *Layout* dari citra dan jarak putih antar baris tidak bisa diubah.
Perangkat lunak pengenalan karakter optik tahap analisa tiap karakter.	a. Pengoptimasian Citra. b. Kualitas dari citra sumber.	a. Citra di optimasi untuk pengenalan karakter optik agar tepi dari karakter lebih halus jelas dan tajam untuk meningkatkan akurasi dari pengenalan karakter tersebut. b. Mengambil sumber sebaik mungkin.
Perangkat lunak pengenalan karakter optik tahap pencocokan tepi karakter dari citra *Sample* dan membuat keputusan untuk mengidentifikasi	a. Citra *Sample* pada *database* perangkat lunak pengenalan karakter optik. b. Algoritma pada pengenalan karakter optik.	a. Memilih perangkat lunak yang bagus untuk pengenalan karakter optik

Langkah Proses	Faktor yang mempengaruhi pengenalan karakter optik	Aksi yang disarankan
dari suatu karakter.		
Perangkat lunak pengenalan karakter optik tahap mencocokan seluruh kata pada kamus dan mengambil keputusan .	a. Algoritma dan kamus yang ada pada perangkat lunak pengenalan karakter optik.	a. Memilih perangkat lunak yang bagus untuk pengenalan karakter optik



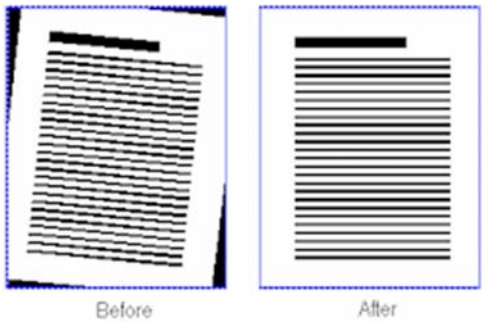
Gambar 2.1 Proses penghapusan *Border* dan *Halftone* [18]



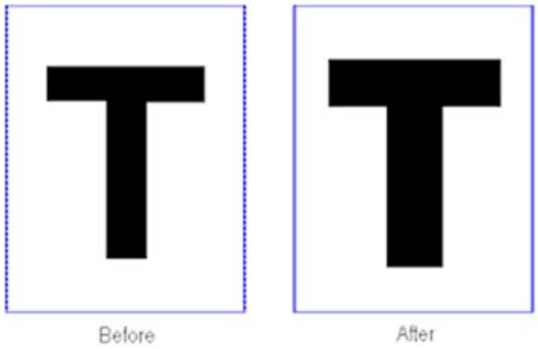
Gambar 2.2 Proses penghapusan lubang [18]



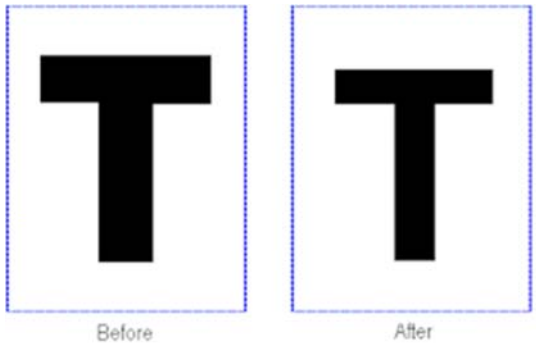
Gambar 2.3 Proses *Cropping* [18]



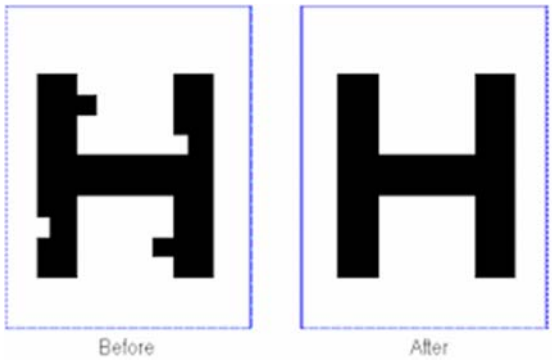
Gambar 2.4 Proses pelurusan gambar [18]



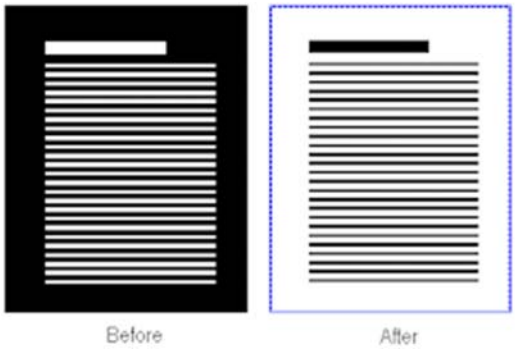
Gambar 2.5 Proses *Dilating* [18]



Gambar 2.6 Proses *Eroding* [18]



Gambar 2.7 Proses penghalusan [18]



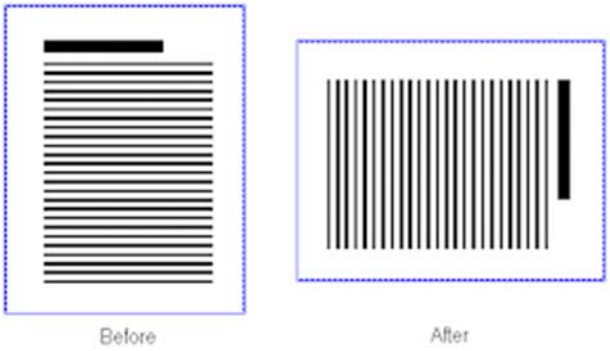
Gambar 2.8 Proses pembalikan warna [18]



Gambar 2.9 Proses penghapusan garis [18]



Gambar 2.10 Proses penghapusan *Noise* [18]



Gambar 2.11 Proses rotasi [18]



Gambar 2.12 Proses skala [18]

Dalam artikel lain yang berjudul "*The DPI You Should Be Scanning Your Paper Photographs*" melakukan penelitian waktu yang digunakan pada pemindaian dengan *DPI* tertentu pada gambar dengan ukuran 3,5 X 3,5". Untuk pemindaian 300 *DPI* membutuhkan waktu 12 detik [18]. Masalah yang timbul saat pendeteksian *OCR* adalah seperti text yang tidak lurus (*Straight Line*), tiap karakter berdempetan, juga resolusi yang kecil [19]. Untuk mengatasi masalah seperti diatas pada *OCR* dapat dilakukan dengan berbagai macam pemprosesan gambar diantaranya adalah *Deskew*(Meluruskan gambar), *Line Removal/Editing* (Menghapus garis), juga *Noise Removal* (Menghilangkan bercak bercak hitam) [20]. *DPI* atau *Dot per Inch* adalah pengukuran kepadatan titik pada cetakan spasial atau video. *Tiff* atau *Tagged Image File Format* adalah format file komputer untuk menyimpan gambar *raster* (*raster image*), serta menggunakan kompresi *loseless* tanpa mengubah kualitas gambar. Proses proses yang terjadi dalam meningkatkan akurasi pada *OCR* yang dijelaskan pada "*Oracle Distributed Document Capture/Oracle Document Capture Administrator's Guide*" diantaranya adalah sebagai berikut :

a. Penghapusan *Border, Halftone* dan lubang

Proses ini untuk menghilangkan sebuah *border*, *background* atau latar belakang seperti pada Gambar 2.1, yang sebelumnya terdapat *border* setelah melalui proses ini akan dihapus *border tersebut*. Dan juga proses penghapusan lubang pada citra pada Gambar 2.2 agar meningkatkan akurasi dari pengenalan karakter optik.

b. *Cropping*

Proses ini membuang bagian yang tidak perlu pada sebuah teks. Seperti pada Gambar 2.3 citra sesudah melalui proses ini akan membuang bagian yang tidak perlu untuk meningkatkan efisiensi dari proses pengenalan karakter optik.

c. Meluruskan gambar yang miring

Proses ini untuk meluruskan gambar yang posisinya tidak lurus atau miring. Pada Gambar 2.4 terlihat setelah melalui proses ini akan lebih mudah mengenali karakter yang *horizontal*.

d. *Dilating and Eroding*

Proses *Dilating* adalah untuk membesarkan area hitam pada huruf. Juga dapat meningkatkan kualitas gambar namun meningkatkan ukuran *file* gambar tersebut. Seperti pada Gambar 2.5 karakter terlihat lebih tebal dari sebelumnya.

Untuk proses *Eroding* digunakan untuk mengecilkan area hitam pada huruf. Proses ini mengecilkan ukuran *file* gambar, namun menurukan detail dari gambar tersebut. Pada Gambar 2.6 karakter terlihat lebih tipis dari sebelumnya.

e. Penghalusan dan pembalikan warna

Proses Penghalusan (*Smoothing*) adalah untuk menghapus benjolan atau gumpalan diluar teks maupun kilauan didalam teks. Pada Gambar 2.7 karakter yang terdapat benjolan atau kilauan diperbaiki menjadi karakter yang normal.

Proses pembalikan warna (*Inverting*) adalah untuk mengubah *pixel* hitam ke *pixel* putih. Seperti pada Gambar 2.8 yang sebelumnya berwarna hitam menjadi warna putih.

f. Penghapusan garis

Proses ini menghapus garis yang ada dalam sebuah gambar. Pada Gambar 2.9 beberapa garis dihilangkan dengan ketebalan tertentu.

g. Penghapusan *Noise*

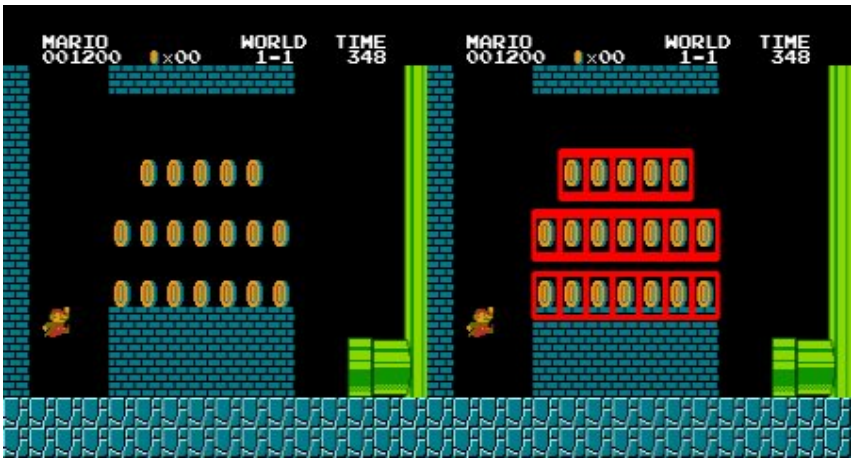
Proses ini menghilangkan *noise* pada sebuah gambar. Pada Gambar 2.10 *noise* dihilangkan untuk meningkatkan akurasi dari pengenalan objek.

h. Rotasi dan Skala

Proses ini untuk merotasikan gambar serta men-*scale* gambar. Pada Gambar 2.11 gambar dirotasi ke arah tertentu. Pada Gambar 2.12 gambar diskala dan dapat digunakan untuk meningkatkan akurasi dari proses pengenalan karakter optiknya.

2.5.2. Pencocokan Sampel

Metode Pencocokan Sampel adalah sebuah tehnik dalam pemprosesan citra untuk menemukan bagian kecil dari sebuah gambar yang cocok dengan gambar sampel [21]. Namun dari metode ini memiliki banyak kelemahan seperti sensitif terhadap distorsi pola, seperti arah penglihatan (*Angle of Viewing*), sensitif terhadap ukuran dan rotasi, serta komputasi dalam komputer yang besar [22]. Contoh dari pencocokan sampel seperti pada Gambar 2.13 berikut.



Gambar 2.13 Contoh *Template Matching* koin [23]

2.5.3. Pendeteksian Berbasis Feature

Metode ini berbasiskan *Feature* dalam pendeteksiannya. Dalam sebuah penelitian yang berjudul " *A Comparative Study of Three Image Matching Algorithms :SIFT, SURF and FAST"*, melakukan pengujian antara ketiga metode pada objek yang berbeda beda. Dari pengujian tersebut pada metode *SIFT* menghasilkan lebih banyak *Feature* daripada *SURF* dan *FAST*, hasilnya tertera dalam Tabel 2.2. Sedangkan perbandingan tiga algoritmanya dalam mendeteksi *Feature* serta efektivitas dari *Feature* yang dicocokan dapat dilihat pada Tabel 2.3, Tabel 2.4 dan Tabel 2.5. Metode *FAST* (*Features From Accelerated Segment Test)*

adalah metode pendeteksi sudut (*Corner Detection)*, yang dapat digunakan untuk menghasilkan *Feature.* Sedangkan *Features* di dalam penglihatan komputer adalah sebuah potongan informasi, yang mungkin menjadi struktur spesifik dalam sebuah gambar seperti *points,* tepi (*Edge)*, atau objek. Deskriptor adalah representasi dari sebuah fitur.

Tabel 2.2 Perbandingan ketiga algoritma dalam mendeteksi *Feature* [3]

Features Detected				
Pair	**Image**	**SIFT**	**SURF**	**FAST**
Window Pair	A	859	770	86
	B	754	734	128
Brick Wall Pair	A	1117	507	13
	B	1106	516	6
Boat Pair	A	484	209	50
	B	604	326	294
Building Pair	A	842	606	216
	B	550	202	52

Tabel 2.3 Efisiensi dari metode *SIFT* [3]

	SIFT		
	Features	**# Matches**	**% Effectiveness**
Window Pair	859	17	2%
	754		
Brick Wall Pair	1177	113	10%
	1106		
Boat Pair	484	23	4%
	604		
Building Pair	842	203	29%
	550		

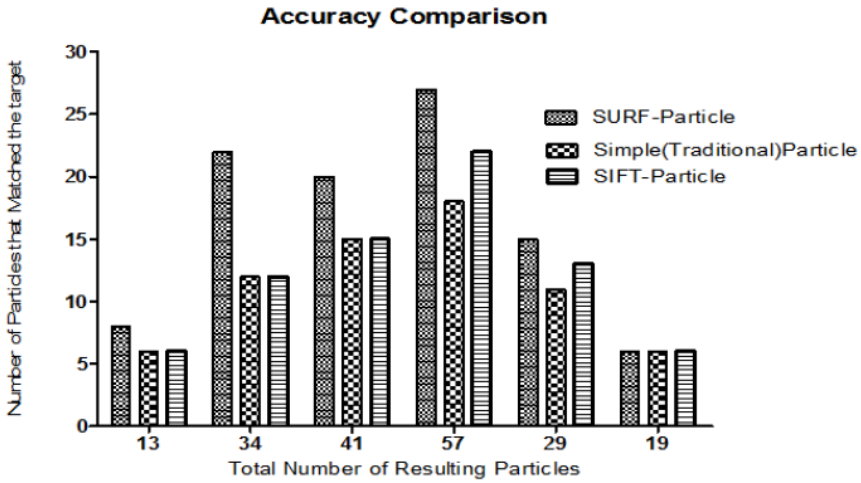
Tabel 2.4 Efisiensi dari metode *SURF* [3]

	SURF		
	Features	**# Matches**	**% Effectiveness**
Window Pair	770	24	3%
	734		
Brick Wall Pair	507	72	14%
	516		
Boat Pair	209	10	4%
	326		
Building Pair	606	102	25%
	202		

Tabel 2.5 Efisiensi dari metode *FAST* [3]

	FAST		
	Features	**# Matches**	**% Effectiveness**
Window Pair	86	17	16%
	128		
Brick Wall Pair	13	5	53%
	6		
Boat Pair	50	18	10%
	294		
Building Pair	216	26	19%
	52		

Pada proses pencarian kesamaan (*Matching Process)*, efisiensi dari metode *SURF* lebih besar daripada *SIFT* dan *FAST* [3] seperti yang terlihat pada Tabel 2.3, Tabel 2.4 dan Tabel 2.5. Pada sebuah penelitian lain yang berjudul "*A Comparative Study between SIFT-Particle and SURF-Particle Video Tracking Algorithms*", pada Gambar 2.14, algoritma *SURF* menghasilkan *Particle* lebih banyak dan lebih akurat daripada metode tradisional dan metode *SIFT*. Metode *SURF* lebih banyak particle yang memiliki kesamaan dengan targetnya [24].



Gambar 2.14 Graph akurasi dan partikel yang dihasilkan tiap metode [3]

Metode *SIFT* atau *Scale Invariant Feature Transform* adalah algoritma untuk mendeteksi den mendeskripsikan fitur lokal dari sebuah gambar. Metode *SURF* atau *Speeded Up Robust Feature* adalah metode yang dikembangkan dari metode *SIFT* dengan fungsi yang sama sebagai algortima pendeteksi fitur yang dapat digunakan dalam penglihatan komputer seperti pengenalan objek (*Object Recognition*) [4]. Perbedaan dari metode *SIFT* dalah penggunaan gambar *integral* (*Integral Image*) sebagai basis atau dasar dari pendeteksiannya [3]. Metode *FAST* atau *Features From Accelerated Segment Test* adalah metode pendeteksi sudut (*Corner Detection)*, yang dapat digunakan untuk menghasilkan *Feature.* Sedangkan *Features* di dalam penglihatan komputer adalah sebuah potongan informasi, yang mungkin menjadi struktur spesifik dalam sebuah gambar seperti *points,* tepi (*Edge)*, atau objek.

2.6. *Speeded Up Robust Feature* (*SURF*)

SURF atau *Speeded Up Robust Feature* adalah metode yang dikembangkan dari metode *SIFT* dengan fungsi yang sama sebagai algortima pendeteksi fitur yang dapat digunakan dalam penglihatan komputer (*Computer Vision*) seperti pengenalan objek (*Object Recognition*). Peningkatan performa utama dalam *SURF* adalah penggunaan pertengahan representasi gambar yang dikenal sebagai gambar *integral*(*Integral Image*) [4]. *SURF* adalah sebuah pendeteksi dan *Descriptor* titik

kunci tingkat tinggi dalam sebuah gambar, dimana gambar diubah kedalam bentuk koordinat menggunakan tehnik *multi-resolution*.

2.6.1. Integral Image

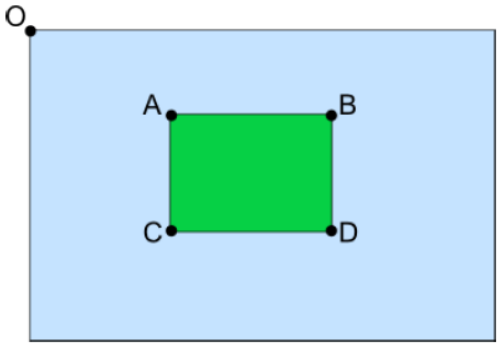
Gambar *integral* atau penjumlahan area tabel (*Summed Area Table*) dikomputasi secara cepat dari gambar masukan dan digunakan untuk mempercepat kalkulasi dari area persegi panjang yang tegak lurus seperti konvolusi dengan *Box Filter* atau *box blur*. Dengan menggunakan gambar *integral* dalam menentukan jumlah intensitas *pixel,* dapat dikalkulasikan dengan rumus 2.1 sebagai berikut dan contoh area komputasi dari gambar *integral* pada gambar 2.17 [4]:

$$\sum = A + D - (C + B) \quad \text{..... (2.1)}$$

Misal terdapat sebuah matriks gambar masukan sebagai berikut pada Gambar 2.15, kemudian akan dicari penjumlahan dari area komputasi tertentu (A,B,C,D) dalam matriks gambar (O) tersebut seperti Gambar 2.16.

1	2	2	4	1
3	4	1	5	2
2	3	3	2	4
4	1	5	4	6
6	3	2	1	3

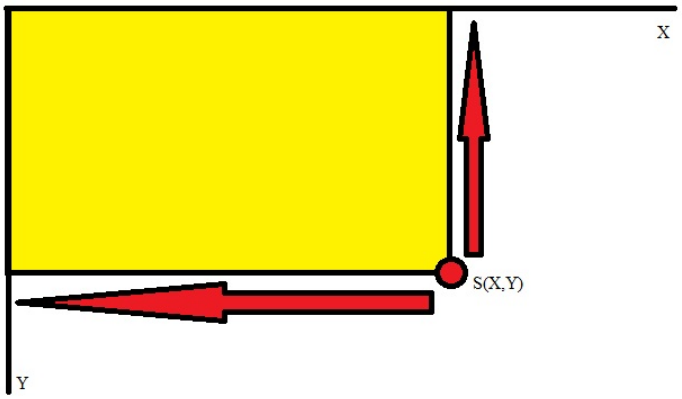
Gambar 2.15 Contoh gambar masukan



Gambar 2.16 Contoh area komputasi [4]

Pertama adalah mencari gambar *integral*nya atau penjumlahan area tabel menggunakan rumus (2.2) dengan cara menjumlahkan nilai mulai dari kiri atas seperti Gambar 2.17. i adalah gambar masukan, S adalah jumlah nilai pada posisi x dan y, x dan y adalah sumbu koordinat.

$$S(x,y) = i(x,y) + S(x-1,y) + S(x,y-1) - S(x-1,y-1) \quad (2.2)$$



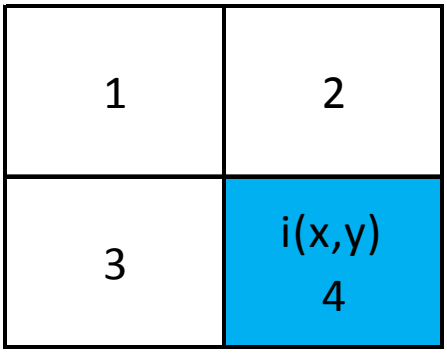
Gambar 2.17 Ilustrasi perhitungan nilai S(x,y) Gambar *Integral*

Untuk menentukan nilai S(x,y) atau gambar *integral* pada posisi x dan y. Dikarenakan x-1, y-1serta x-1 y-1 diluar dari batas angka matriks yang ada, maka secara *default* diberi nilai 0. Tahapan pembentukan gambar *integral* adalah sebagai berikut, apabila titik yang ingin dicari nilainya adalah S(x,y) :

1. Untuk mencari nilai S(x,y) pada Gambar 2.18 berikut, jumlahkan hasil dari tiap tiap S dengan menggunakan Gambar 2.19.
2. Hasilnya dimasukan kedalam rumus (2.2).
3. Didapatlah hasilnya seperti Gambar 2.20.

S(x-1,y-1) 1	S(x,y-1)
S(x-1,y)	S(x,y)

Gambar 2.18 Mencari nilai S(x,y) gambar *integral*



Gambar 2.19 Gambar awal

Kemudian dicari nilainya sampai semuanya terhitung seperti yang terlihat pada Gambar 2.20.

1	3	5	9	10
4	10	13	22	25
6	15	21	32	39
10	20	31	46	59
16	29	42	58	74

Gambar 2.20 Hasil gambar *Integral*

Setelah semua terisi, dimisalkan ingin mencari penjumlahan dari area yang di beri tanda kotak merah pada Gambar 2.21 menggunakan rumus gambar *integral* (*Integral Images*) yaitu rumus (2.1).

1	3	5	9	10
4	10 A	13	22 B	25
6	15	21	32	39
10	20 C	31	46 D	59
16	29	42	58	74

Gambar 2.21 Area yang akan dicari jumlah *pixel*-nya

Sehingga menjadi 46 + 10 – 22 – 20 = 14, 14 adalah jumlah dari nilai yang diberi kotak pada gambar awal pada gambar 3.7, 3 + 2 + 5 + 4 = 14 .

1	2	2	4	1
3	4	1	5	2
2	3	3	2	4
4	1	5	4	6
6	3	2	1	3

Gambar 2.22 Gambar awal

2.6.2. Key Point Detection

Pendeteksian pada *SURF* menggunakan matriks hessian. Dalam pengimplementasian matriks tersebut, digunakan sebuah fungsi f(x,y) kedalam matriks tersebut. Matriks *hessian* adalah turunan parsial dari fungsi f seperti pada rumus (2.3) sebagai berikut.

$$H\big(f(x,y)\big) = \begin{bmatrix} \frac{\partial^2 f}{\partial^2 x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial^2 y^2} \end{bmatrix} \quad \text{.....(2.3)}$$

Determinan dari matrik ini disebut juga sebagai diskriminan dikalkulasikan dengan rumus (2.4)

$$\det(H) = \frac{\partial^2 f}{\partial^2 x^2}\frac{\partial^2 f}{\partial^2 x y^2} - \left(\frac{\partial^2 f}{\partial x \partial y}\right)^2 \quad \text{.....(2.4)}$$

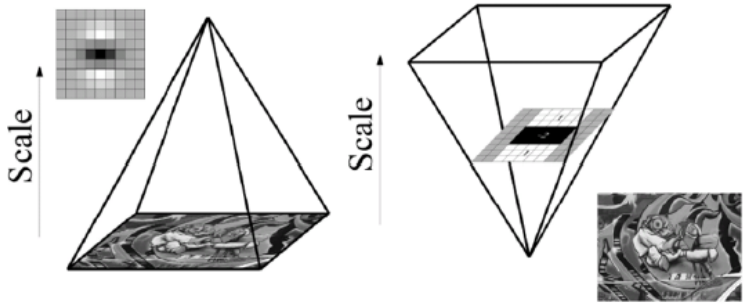
Nilai diskriminan tersebut digunakan untuk mengklasifikasikan *maxima* dan *minima* dari sebuah fungsi turunan orde dua. Dikarenakan determinan adalah produk dari nilai *eigen* pada *hessian*, dapat diklasifikasikan sebuah titik berdasarkan tanda dari hasilnya. Jika determinannya negatif maka nilai *eigen*nya memiliki tanda yang berbeda dan tidak berada pada *extrenum* lokal biasa disebut *maximum* atau *minimum*. Dan dapat berupa lokal atau *relative* dimana *extrenum* tersebut bukanlah yang paling besar atau kecil (*overall*) pada batas yang diberikan [25]. Sebaliknya jika determinannya positif maka antara nilai *eigen* adalah positif atau negatif dan titik tersebut diklasifikasikan sebagai *extrenum*. Untuk menggunakan rumus ini

pada gambar digunakanlah turunan kedua dari *gaussian* sehingga matriks tersebut menjadi seperti rumus (2.5) berikut. Untuk menghitung determinannya metode *SURF* menggunakan pendekatan *hessian* seperti rumus (2.6) berikut. Metode *SURF* memanfaatkan pendekatan *Laplacian of Gaussian* menggunakan *Box Filter* yang sudah ditentukan *kernel*nya. Penggunaan *kernel* dengan beban yang simpel pada *SURF* meningkatkan kemudahan dalam pemprosesannya. *Box Filter* ini dapat dipercepat perhitungannya menggunakan *integral image* [4].

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \quad \text{..... (2.5)}$$

$$\det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \quad \text{..... (2.6)}$$

Ruang skala atau *Scale Space* adalah fungsi yang berkelanjutan atau *continous function* yang digunakan untuk menentukan *extrema* atau biasa disebut *maximum* dan *minimum* dari sebuah fungsi dari segala kemungkinan skala. Dalam penglihatan komputer ruang skala biasanya diimplementasikan sebagai gambar piramid seperti pada Gambar 2.23 [4].



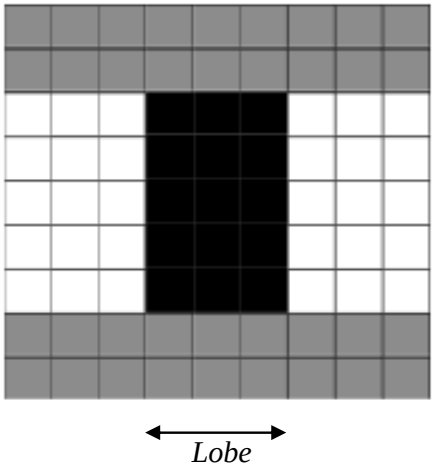
Gambar 2.23 Ruang Skala [4]

Ruang skala tersebut dibagi menjadi beberapa *Octave.* Dimana sebuah *Octave* yaitu serangkaian peta respon yang meliputi penggandaan skala. Untuk menghitung skala pada tiap *filter* dapat menggunakan rumus (2.7) berikut.

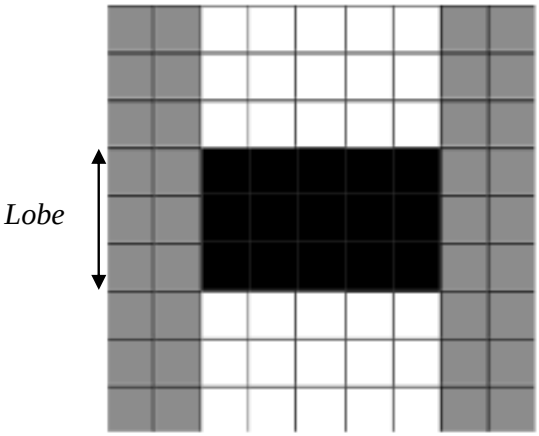
$$\sigma_{approx} = Current\ Filter\ Size * \frac{Base\ Filter\ Scale}{Base\ Filter\ Size} \quad \text{..... (2.7)}$$

Pada *SURF* level yang paling kecil pada ruang skala didapat dari hasil *filter* 9x9. Pada jurnal "*Notes on the OpenSURF Library*" menjelaskan untuk membuat *filter* dengan ukuran yang besar juga perlu diperhatikan berbagai faktor. Seperti

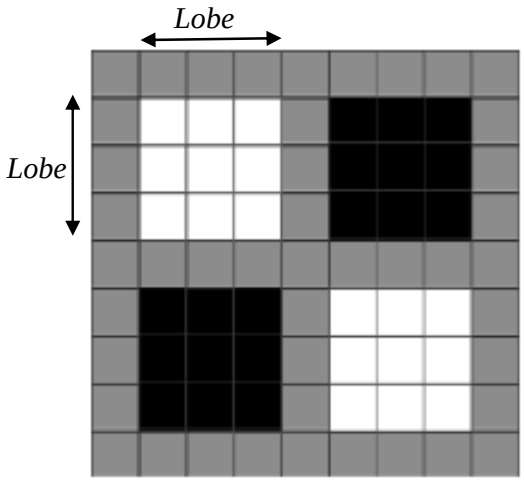
untuk meningkatkan ukurannya dibatasi oleh panjang *lobe* yang direpresentasikan garis hitam pada arah tertentu dari turunan orde dua *gaussian*. Pada pendekatan turunan orde dua *gaussian* ukuran *lobe* di buat satu pertiga dari panjang *filter* seperti pada Gambar 2.24, Gambar 2.25 dan Gambar 2.26.



Gambar 2.24 *Lobe Filter* arah X [4]

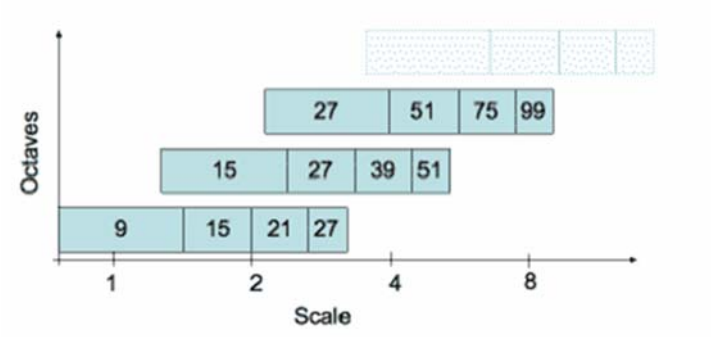


Gambar 2.25 *Lobe Filter* Arah Y [4]



Gambar 2.26 *Lobe Filter* arah XY [4]

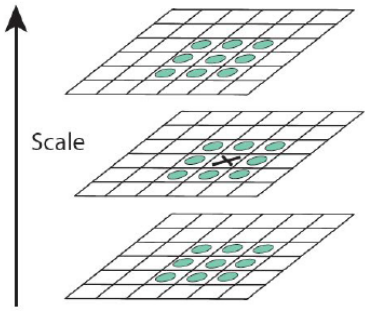
Dikarenakan terdapat 3 buah *lobe* pada tiap *filter* yang ukurannya harus sama, langkah (*step*) terkecil diantara *filter* tersebut adalah 6. Pada *SURF* secara *default* menggunakan 3 *Octave* dan 4 *Interval* pada tiap *Octave* nya seperti pada Gambar 2.27 [4].



Gambar 2.27 *Octave* dan *Interval*/ *Scale* [4]

Untuk mencari penempatan titik kunci yang tetap atau *invariant* pada rotasi dan skala dalam sebuah gambar dapat dilakukan dengan tiga cara. pertama dengan cara *tresholding* atau ambang batas. Ini bertujuan agar semua nilai dibawah ambang batas dihapus sehingga apabila ambang batas dinaikan maka akan terdapat sedikit titik kunci yang terdeteksi begitu juga sebaliknya. Penentuan ambang batas ini dapat diadaptasikan atau ditentukan pada tiap aplikasi. Setelah di *tresholding* dilakukan penekanan *non-maximal* seperti Gambar 2.28 untuk menentukan titik

kandidat atau calon dari titik kunci. Dengan cara membandingkan 6 tetangga yang terdiri atas 8 titik di *Scale* asli dan 9 di tiap *Scale* atas dan bawahnya [4].

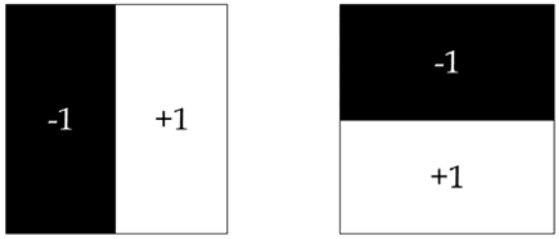


Gambar 2.28 Penekanan Non Maximal [4]

2.6.3. Descriptor Key Point

Deskriptor pada *SURF* menjelaskan bagaimana intensitas *pixel* di distribusikan dalam tiap titik kunci. Pendekatan ini sama dengan apa yang ada pada *SIFT*, namun dalam *SURF* menggunakan gambar *integral* untuk meningkatkan *robustness* dan mengurangi waktu komputasi [4].

Untuk mendapatkan titik yang tetap atau *invariance* pada rotasi setiap titik kunci yang terdeteksi seperti pada Gambar 2.29 dan ditugaskan untuk dapat mereproduksi ulang pada orientasi. Untuk menentukan orientasi *Haar Wavelet* atau urutan yang berbentuk persegi dari ukuran 4σ dihitung untuk beberapa bagian *pixel* di dalam jangkauan 6σ dari *point* yang terdeteksi [4]. Tanggapannya akan diukur dengan *Gaussian* yang terletak ditengah pada titik kunci dan kemudian direpresentasikan sebagai titik pada ruang *vektor* [4].



Gambar 2.29 *Wavelet Response* [4]

2.7. *Fast Library for Approximate Nearest Neighbors*(*FLANN*)

Fast Library for Approximate Nearest Neighbors adalah sebuah *library* untuk melakukan pendekatan yang cepat dalam pencarian tetangga terdekat pada ruang dimensi yang tinggi. *Fast Library for Approximate Nearest* (*FLANN*) ditulis dalam bahasa *C++* dan bisa dieksekusi oleh bahasa lainnya seperti *C*, *Matlab*, dan *Phyton* [5]. *Library FLANN* melakukan pencarian tetangga terdekat menggunakan *linear, brute force search* atau pencarian *linear* secara paksa dari awal sampai akhir.

2.8. *Unified Modelling Languange* (*UML*)

Unified Modelling Languange (*UML*) adalah sebuah bahasa pemodelan standar yang memiliki sintaks dan *semantic*. Pemodelan ini sangat cocok digunakan untuk merancang dan memodelkan sistem berorientasi objek. Diagram pada uml dibagi menjadi dua bagian yaitu *structural Diagram* dan *behavior Diagram*. *Structural Diagram* digunakan untuk mendeskripsikan relasi antar kelas, *Tools* yang digunakan pada bagian ini yaitu *Class Diagram*. Sedangkan *behavior* Diagram digunakan untuk mendeskripsikan interaksi antara aktor dan sebuah *Use Case* (bagaimana seorang aktor menggunakan sistem). *Tools* yang digunakan pada bagian ini yaitu *Use Case Diagram*, *Sequence Diagram*, dan *Activity Diagram* [26].

2.8.1. Diagram Use Case

Diagram *Use Case* digunakan untuk mendeskripsikan kejadian-kejadian apa saja yang dapat dilakukan oleh *user*/aktor dan fungsionalitas-fungsionalitas apa saja yang diharapkan dari sistem yang akan dibangun. Tanpa mendeskripsikan bagaimana sistem menyelesaikannya. Sebuah *Use Case* menggambarkan suatu urutan interaksi antara satu atau lebih aktor dan sistemm dalam tahap *requirements*. Model *Use Case* mengambarkan sistem sebagai sebuah kotak hitam dan interaksi antara aktor dan sistem dalam suatu bentuk naratif. Setiap *Use Case* menggambarkan perilaku sejumlah aspek sistem, tanpa mengurangi struktur *internal*-nya. Selama pembuatan model *Use Case* secara pararel juga harus ditetapkan obyek-obyek yang terlibat dalam setiap *Use Case* [26].

2.8.2. Diagram Sequence

Diagram *Sequence* menggambarkan interaksi antar objek di dalam dan di sekitar sistem yang menekankan pada pengiriman pesan dalam suatu waktu tertentu. Diagram *Sequence* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan dari sebuah event untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara *internal* dan *output* apa yang dihasilkan [26].

2.8.3. Diagram Activity

Diagram *Activity* menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* atau keputusan yang mungkin terjadi, dan bagaimana mereka berakhir. Diagram *Activity* merupakan *state Diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu Diagram *Activity* tidak menggambarkan *behaviour internal* sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari *level* atas secara umum [26].

2.8.4. Diagram Class

Diagram *Class* digunakan untuk menggambarkan keadaan suatu sistem dengan menjelaskan keterhubungan antara suatu *Class* dengan *Class* yang lain yang terdapat pada sistem. Sebuah *Class* terdiri dari nama, atribut dan *method*. Atribut dan *method* dari sebuah *Class* atau kelas mempunyai *visibility*. Ada tiga jenis *visibility* yang digunakan yaitu *private*, *public*, dan *protected*. Setiap kelas pada Diagram *Class* mempunyai hubungan dengan kelas lainnya. Ada beberapa jenis hubungan kelas, yaitu *dependency*, *asosiasi*, *agregasi* dan *generalisasi* [26].

a. *Dependency* merupakan hubungan terlemah antar kelas. *Dependency* bermakna satu kelas menggunakan atau memiliki pengetahuan terhadap kelas lain, namun hubungannya hanya sementara dan tidak ada batas waktu yang jelas.

b. *Asosiasi* mempunyai hubungan yang lebih kuat dari hubungan *dependency*. Dimana suatu *Class* tetap berhubungan dengan kelas lain seterusnya, hubungan *asosiasi* dibagi menjadi dua jenis yaitu *directional* dan *bidirectional*.
c. *Agregasi* merupakan bentuk hubungan yang mengimplikasikan kepemilikan suatu kelas. *Agregasi* juga merupakan bentuk yang lebih kuat dari *asosiasi*. Hubungan *agregasi* dinyatakan dengan simbol *diamond* pada pemilik kelas dan garis utuh berpanah kekelas yang dimiliki.
d. *Generalisasi* merupakan bentuk hubungan antar kelas, dari kelas yang umum dengan kelas yang lebih khusus. Contoh kelas hewan memiliki hubungan *generalisasi* dengan kelas kucing, karena kucing merupakan kelas khusus dari kelas hewan yang lebih umum.

2.9. *Emgu CV*

Emgu CV adalah sebuah *library OpenCV* yang dapat digunakan pada bahasa pemprograman *C#. Emgu CV* dapat di-*compile* pada *Microsoft Visual Studio, Xamarin Studio,* dan *Unity*. *Emgu CV* dapat dijalankan juga pada sistem operasi *linux, windows, Mac OS X, iOS, Android,* dan *Windows Phone* [27].

2.10. *Arduino*

Arduino adalah sebuah perangkat prototipe untuk mengembangkan suatu alat yang baru. *Arduino* menggunakan bahasa pemprograman sendiri pada pengaplikasiannya, yaitu bahasa pemprograman *Arduino* [28].