BAB 2

LANDASAN TEORI

2.1 Teori *Neuro Fuzzy*

Neuro-fuzzy sebenarnya merupakan penggabungan dari dua studi utama yaitu *fuzzy logic* dengan aplikasi *neuro computing*. Masing-masing memiliki cara dan proses tersendiri, akan tetapi tetap dapat dipadukan sehingga menghasilkan performa kerja yang selaras. Dalam hal ini, penggunaan *neuro computing* dan *fuzzy logic* dapat dipadukan menghasilkan *Neuro Fuzzy* atau *Fuzzy Neural Network*.

Penggabungan ini dilakukan karena manusia memiliki nalar dan proses pembelajaran yang bisa dikatakan memiliki nilai kekaburan. Nilai kekaburan ini membuat penilaian manusia akan suatu hal menjadi tidak terlalu konstan atau kaku untuk sebuah kondisi atau objek. Dengan hal ini, komputer akan mampu untuk melakukan proses pembelajaran yang lebih baik lagi.

2.1.1 *Fuzzy Logic*

Fuzzy Logic adalah suatu cara yang tepat untuk memetakan suatu ruang *input* ke dalam suatu ruang *output*. *Fuzzy logic* juga memiliki himpunan *fuzzy* yang mana pada dasarnya, teori himpunan *fuzzy* merupakan perluasan dari teori himpunan klasik (Kusumadewi, 2005, p13). Dengan menggunakan *fuzzy logic*, hasil yang keluar tidak

akan selalu konstan dengan input yang ada. Menggunakan himpunan *fuzzy*, hasil yang keluar juga akan menjadi output yang rasional sesuai dengan penalaran manusia.

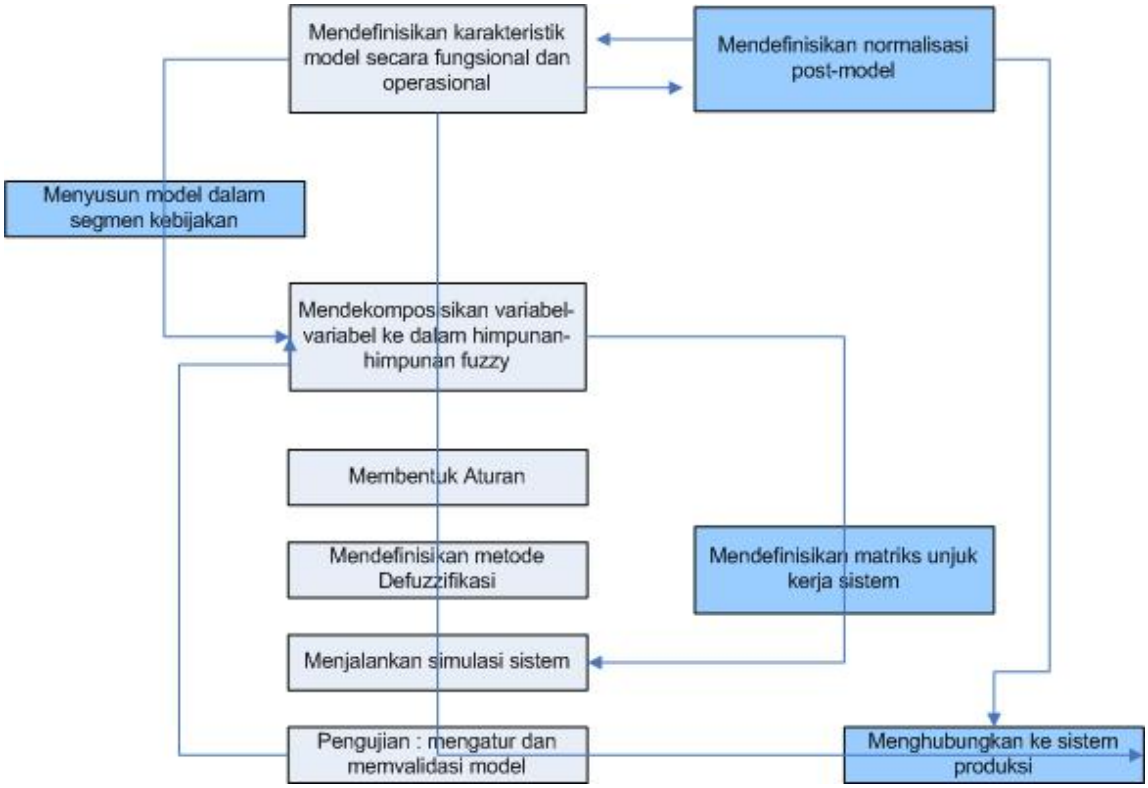
Sebelum memulai dengan input, proses fuzzifikasi, dan output, kita lebih baik mengenal beberapa *scope* dan variabel yang digunakan dalam proses tersebut. Ada beberapa hal yang perlu diketahui dalam memahami sistem *fuzzy* (Kusumadewi, 2003, p158):

a. Variabel *fuzzy,* merupakan variabel yang hendak dibahas dalam suatu sistem *fuzzy*.
b. Himpunan *fuzzy*, merupakan suatu grup yang mewakili suatu kondisi atau keadaan tertentu dalam suatu variabel *fuzzy*.
c. Semesta pembicaraan, merupakan keseluruhan nilai yang diperbolehkan untuk dioperasikan dalam suatu variabel *fuzzy*.
d. Domain, merupakan keseluruhan nilai yang diijinkan dalam semesta pembicaraan dan boleh dioperasikan dalam suatu himpunan *fuzzy*.

Cara kerja aplikasi *fuzzy logic* secara garis besar terdiri input, porses, dan output. Input dalam *fuzzy logic* merupakan perpaduan dari *membership function* dan *variable* yang telah disesuaikan agar sesuai dengan fungsi yang dipakai dalam suatu aplikasi *fuzzy*.

Dalam penalaran *fuzzy*, dikenal proses penalaran Mamdani dan Sugeno. Dalam proses penalaran Mamdani, baik *Input* maupun *Output* merupakan sistem himpunan *fuzzy*. Sedangkan dalam proses penalaran Sugeno, mirip dengan metode Mamdani hanya *output* sistem tidak berupa himpunan *fuzzy*, melainkan berupa konstanta atau persamaan linear.

Arsitektur pengembangan sistem fuzzy secara umum digambarkan seperti di bawah ini :



Gambar 2.1 - *Arsitektur Pengembangan Sistem Fuzzy*

2.1.2 *Neuro Computing*

Neuro Computing merupakan implementasi jaringan syaraf menggunakan program komputer yang mampu menyelesaikan sejumlah proses perhitungan selama proses pembelajaran (Fausett, 1994). Pengimplementasian jaringan syaraf ini dikarenakan manusia mampu melakukan pembelajaran atas suatu keadaan dengan bantuan proses pembelajaran yang dilakukan ole sel syaraf. Dengan melakukan

pembelajaran, manusia mampu mengenali sebuah obyek walaupun dengan keadaan data input yang sedikit berbeda.

Dalam sistem syaraf manusia, tiap sel syaraf memiliki satu inti sel yang nantinya akan bertugas untuk melakukan proses informasi. Informasi tersebut diteruskan ke dendrit, yang juga menyertai axon sebagai keluaran dari suatu pemrosesan informasi. Informasi ini akan menjadi masukan bagi sel syaraf lainnya yang mana antar dendrit kedua sel tersebut disatukan dengan sinapsis. Informasi ini akan diaktifkan oleh sel syaraf lainnya jika dan hanya jika rangsangan tersebut melewati nilai *threshold* atau batasan tertentu.

Neuro Computing menggunakan sel syaraf manusia untuk diaplikasikan pada program komputer. Dengan hal ini, diharapkan komputer nantinya akan mampu melakukan pembelajaran melalui aplikasi sel syaraf sehingga mampu melakukan apa yang manusia sekarang ini lebih baik dalam mengerjakannya.

Artificial Neural Network (ANN) atau jaringan syaraf tiruan memiliki karakteristik yang mirip dengan jaringan syaraf pada manusia. Keserupaan ini dapat terlihat dengan adanya:

a) Pola koneksi di antara neuron (arsitektur)
b) Metode untuk mengukur berat / *weight* dari tiap koneksi (*learning* atau *training algorithm*)
c) *Activation function*

Jaringan syaraf tiruan adalah sistem pemrosesan informasi yang mempunyai karakteristik serupa dengan jaringan neural biologis. Karakteristik yang diadopsi antara lain:

a) Jumlah yang besar dari *processing element* / *neuron*.
b) Neuron-neuron yang bekerja secara paralel.
c) Memiliki sifat *Fault Tolerance*.

Jaringan syaraf tiruan sering pula digunakan dengan maksud arti yang sama dengan *Neural Network*. Fungsi dan kinerja *Neural Network* sebagai sebuah sistem sangat bergantung dari 3 hal berikut:

1. Karakteristik Neuron yang terkait dengan fungsi aktivasi yang digunakan.
2. Topologi jaringan yaitu bagaimana sejumlah neuron dalam system / model *Neural Network* dihubungkan.
3. *Learning rules* yaitu aturan-aturan pembelajaran yang digunakan.

Arsitektur *Neural Network*

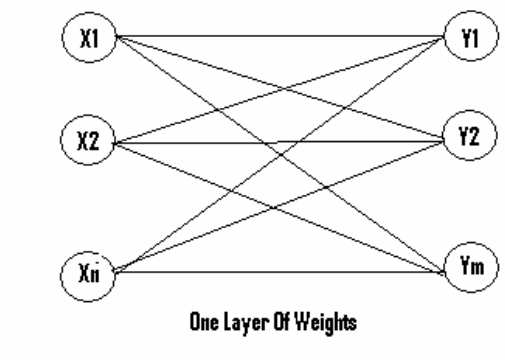
Arsitektur *neural Network* adalah rancangan bangun dari sistem *neural Network* itu sendiri. Rancang bangun ini terdiri dari lapisan yang akan dilalui sebuah proses dengan fungsi aktivasi. Hal ini dikarenakan dalam *artificial neural Network*, sering diatur dalam bentuk lapisan atau *layer*. Lapisan-lapisan ini akan melakukan proses pembelajaran yang terhubung melalui bobot-bobot pada lapisan yang berbeda-beda.

Ada beberapa arsitektur *neural Network*, diantaranya adalah *single layered Network*, *multi layered Network*, dan *recurrent Network*. *Single layered Network*, seperti namanya, adalah sebuah arsitektur *neural Network* yang hanya terdiri dari satu *layer* sebelum akhirnya masuk ke dalam fungsi aktivasi dan menghasilkan output. *Multi layered Network* mirip dengan *single layered Network* dalam hal menghasilkan output, akan tetapi input akan melalui beberapa lapis fungsi aktivasi sebelum akhirnya menghasilkan output. *Recurrent Network* adalah sebuah arsitektur *neural Network* dimana aliran proses yang dilalui input akan melalui banyak lapis dikarenakan aliran prosesnya melakukan timbal-balik yang memungkinkan sebuah proses dilakukan berulang kali pada *layer* yang berbeda.

Pengaturan neuron-neuron ke dalam *layer-layer* dan pola-pola koneksinya disebut *arsitektur neural Networks*. *Neural Network* sering diklasifikasikan sebagai *single layer* atau *multi layer*. Dalam penentuan jumlah *layer*, input unit tidak dihitung sebagai *layer*, karena tidak melakukan komputasi.

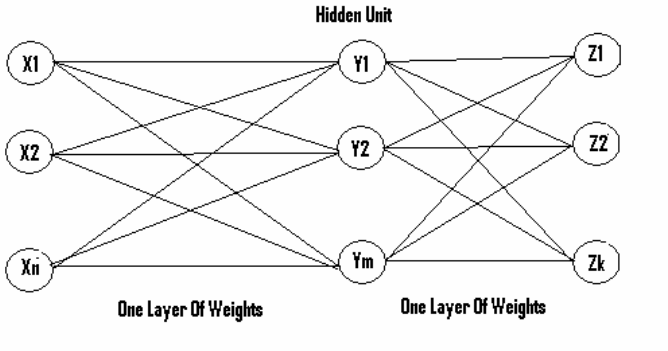
Dengan demikian, jumlah *layer* pada neural *Network*s dapat didefinisikan sebagai "Jumlah *layer-layer* koneksi bobot di antara 2 unit lapisan". Hal ini mudah dipahami karena koneksi bobot berisi informasi yang sangat penting.

Suatu *single layer Network* memiliki satu *layer* koneksi bobot seperti terlihat pada ilustrasi sebagai berikut:



Gambar 2.2 – *Single Layer Network*

Sedangkan pada *multi layer Network*, koneksi bobot terdiri dari 2 atau lebih *layer*. Dengan kata lain, *multi layer Network* memiliki satu atau lebih *hidden unit*. Dengan demikian, *multi layer Network* memiliki satu input unit, satu atau lebih hidden unit, dan satu output unit. *Multi layer Network* dapat memecahkan permasalahan-permasalahan kompleks dibandingkan dengan *single layer Network*. Namun demikian, *training* pada *multi layer Network* lebih sulit dilakukan, walaupun lebih besar peluang kebnerhasilannya. *Ilustrasi dari multi layer Network* adalah:

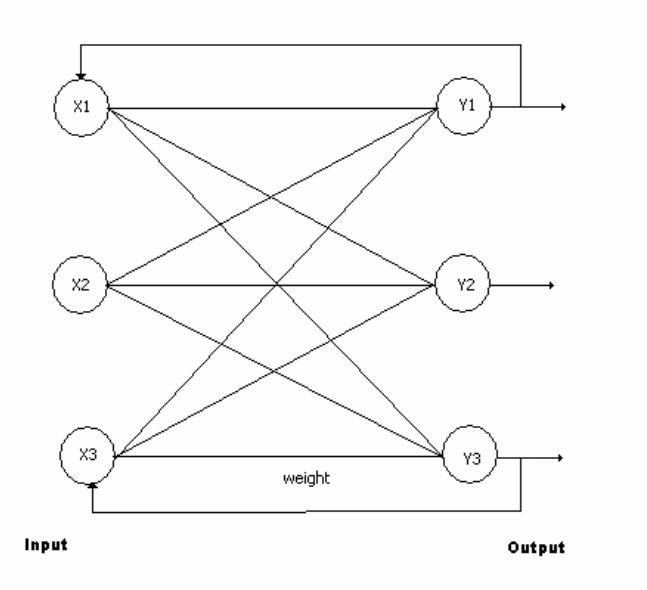


Gambar 2.3 – *Multi Layer Network*

*Network*s yang telah dibahas sebelumnya tidak terdapat koneksi timbal balik, yaitu koneksi yang melalui output suatu *layer* menuju ke input dari *layer* yang sama atau sebelumnya, dan disebut *non recurrent* atau *feed forward Network*.

Secara umum *Network* yang memiliki koneksi timbal-balik disebut *recurrent Network*. *Non recurrent Network* tidak memiliki *memory*, dimana outputnya ditentukan oleh input dan nilai-nilai dari bobot. Dalam beberapa konfigurasi, *recurrent Networks* memutar kembali outputnya menjadi input.

Pada *recurrent Network*, *layer* yang ada terkadang akan kembali ke *layer* sebelumnya dan melakukan komputasi ulang. Sehingga dalam *recurrent Network*, terkadang input akan mengalami proses yang cukup lama sebelum menghasilkan sebuah output. Demikian ilustrasi dari *recurrent Networks*:



Gambar 2.4 - *Recurrent Network*

Learning / Training

Konsep *Neural Network Learning* pun telah dikembangkan untuk ANN dengan tujuan agar input dari *Network* dapat menghasilkan output yang diinginkan (akurat), atau paling tidak menghasilkan output yang konsisten. Input yang didapat pada sebuah ANN merupakan sebuah vektor. *Training* tersebut dilakukan dengan cara mengaplikasikan vektor-vektor input secara berurutan. Selama *training*, bobot *Network* secara bertahap mencapai konvergen ke suatu nilai sedemikian rupa sehingga setiap input menghasilkan output yang diinginkan.

Adapun 2 bentuk learning, yaitu:

A. ***Supervised learning***, memerlukan pasangan setiap input dan output. Pasangan ini disebut juga dengan *training pair*. Biasanya *Network* di-*train* dengan sejumlah *training pair*. 1 input vector diaplikasikan, output dihitung dan dibandingkan dengan target output. Selisihnya dikembalikan ke *Network* dan sekaligus bobotnya disesuaikan berdasarkan suatu algoritma yang cenderung meminimasi error. Vektor-vektor dari *training set* diaplikasikan seluruhnya secara berurutan, error dihitung, bobot disesuaikan sampai seluruh *training set* menghasilkan error sekecil-kecilnya.

B. ***Unsupervised learning***, tidak memerlukan target output. *Training set* hanya terdiri dari vektor-vektor input, tanpa pasangan output. Nantinya algoritma *training* merubah bobot *Network* untuk menghasilkan output yang konsisten. Aplikasi dari vektor-vektor yang cukup serupa akan menghasilkan pola output yang sama. Proses *training* menghasilkan sifat-sifat statistik dalam bentuk

pengelompokkan vektor-vektor dalam beberapa kelas. Aplikasi suatu vektor dari suatu kelas sebagai input akan menghasilkan vektor output yang spesifik.

Fungsi Aktivasi

Fungsi aktivasi adalah suatu fungsi yang akan mengubah nilai input yang telah dikolaborasikan dengan nilai bobot, dan akan menentukan apakah nilai tersebut cukup untuk mencapai sebuah *threshold* tertentu. Kegunaan untuk mencapai nilai *threshold* adalah agar nilai tersebut mampu untuk diaktifkan ke dalam *neuron* berikutnya atau agar komputer dapat melakukan respons apabila nilai tersebut telah melewati nilai batasan tertentu.

Dalam ANN, dikenal beberapa fungsi aktivasi, antara lain:

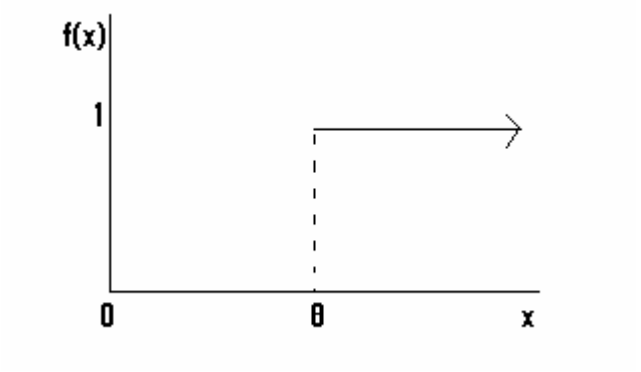
A. Fungsi identitas, dapat ditulis sebagai f(x) = x, dan digunakan pada neuron-neuron input unit. Nilai yang didapat berupa fungsi linear.

B. Fungsi tangga biner, yang mengadopsi nilai *threshold* didalamnya (Θ). Fungsi ini digunakan untuk mengubah input bersih (non input) yang merupakan variable kontinyu, menjadi output bernilai biner (0 atau 1). Fungsi ini dapat ditulis sebagai:

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ 0 & \text{if } x < \theta \end{cases}$$

Gambar 2.5 - Persamaan fungsi tangga biner

Nilai *threshold* menjadi garis pemisah antara daerah denagn respons aktivasi positif dan negatif.



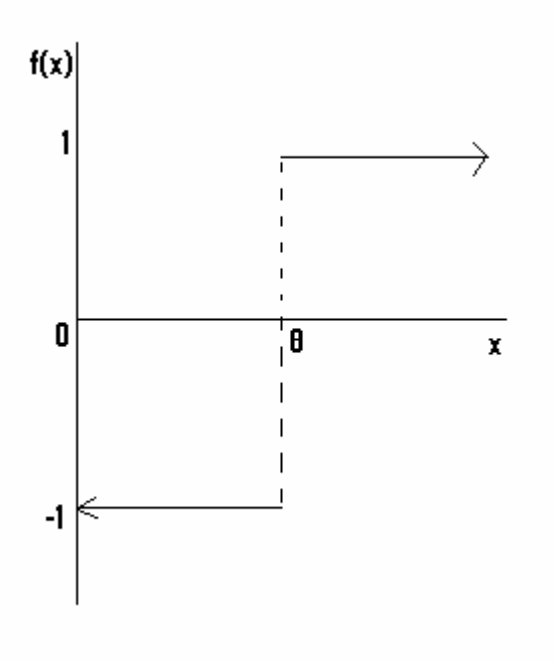
Gambar 2.6 - Grafik fungsi tangga biner

C. Fungsi tangga Bipolar, mirip dengan fungsi tangga biner, tapi dalam Bipolar nilainya tidak hanya berkisar antara 0 dan 1 – melainkan -1 sampai dengan 1. Fungsinya adalah :

$$f(x) = \begin{cases} 1 & \text{if } x \geq \theta \\ -1 & \text{if } x < \theta \end{cases}$$

Gambar 2.7 - Persamaan fungsi tangga Bipolar

Dengan grafik fungsinya sebagai berikut :



Gambar 2.8 - Grafik fungsi tangga Bipolar

D. Fungsi sigmoid Biner, mencakup fungsi-fungsi berbentuk kurva S yang sering digunakan dalam fungsi logistik, karena memiliki kelebihan dalam melatih (*training*) pada *Neural Network* yang menggunakan algoritma *Back Propagation*.

$$f(x) = \frac{1}{1 + e^{-\sqrt{\gamma}x}}$$

Fungsi logistik yang dimaksud :

Dimana ð merupakan parameter kecuraman yang diberikan, biasanya 1.

E. Fungsi sigmoid Bipolar, bisa diskalakan sehingga memiliki daerah hasil pada sembarang interval sesuai dengan permasalahan yang diberikan. Yang paling umum adalah daerah hasil dari -1 sampai dengan 1. Fungsi hasil perskalaan ini disebut dengan fungsi sigmoid Bipolar.

Backpropagation

Backpropagation merupakan algoritma pembelajaran yang terawasi dan biasanya digunakan oleh *perceptron* dengan banyak lapisan untuk mengubah bobot-bobot yang terhubung dengan neuron-neuron yang ada pada lapisan tersembunyinya (Kusumadewi, 2003, p236). *Backpropagation* mampu membawa proses pembelajaran dari *neuro computing* agar makin mendekati hasil atau output yang diinginkan dengan proses *forward* dan *backward approach* sekaligus. Dengan proses ini, nilai dari bobot yang ada pada arsitektur *neural Network* akan dapat diadaptasikan sehingga akan memiliki nilai output yang diinginkan.

Algoritma *backpropagation* menggunakan *error* output untuk mengubah nilai bobot-bobotnya dalam arah mundur (*backward*). Untuk mendapatkan nilai *error* ini, tahap perambatan maju (*forward propagation*) harus dikerjakan terlebih dahulu. Pada saat perambatan maju, *neuron-neuron* diaktifkan dengan menggunakan fungsi aktivasi *sigmoid*, yaitu → f(X) = 1 / 1+ e ^ -x.

2.2 Teori *Computer Vision* sebagai pendukung sistem

Computer vision merupakan salah satu bidang studi yang berkaitan dengan AI, yang mempelajari bagaimana sebuah citra mampu diproses dan dikenali oleh sebuah program komputer. Dengan bantuan AI, *computer vision* akan mampu dikembangakn menjadi sistem intelijen visual. Pengembangan *computer vision* telah diperluas untuk proses *biometric*, robotik, dan pengenalan visual.

Computer vision sebenarnya adalah gabungan dari pengolahan citra (*Image processing*) dan pengenalan pola (*Patter recognition*). Sebuah citra yang diambil perlu dilakukan proses terlebih dahulu, agar nantinya komputer akan lebih mudah untuk mengenali pola yang diberikan.

2.2.1 *Image Processing*

Suatu citra yang diambil, akan diproses menggunakan cara-cara yang telah dijelaskan diatas. Secara garis besar, sebuah citra yang ditangkap melalui sebuah kamera akan diteruskan ke komputer untuk diproses melalui *image processing* dan *pattern recognition*. Proses pertama yang dilakukan oleh komputer adalah mengubah citra tersebut menjadi citra biner. Dengan mengubah citra tersebut, waktu yang diperlukan komputer tersebut untuk memproses sebuah citra akan dipersingkat. Citra biner akan memberikan nilai *grayscale* dan nilai intensitas, yang keduanya diperlukan untuk pemrosesan lebih lanjut. Proses selanjutnya adalah *edge detection*, dimana proses ini secara umum akan mengenali bentuk dari suatu objek yang ditangkap. Setelah melalui proses tersebut, citra yang diambil tadi akan memiliki *value* yang nantinya akan menjadi input bagi aplikasi *Fuzzy Logic* yang nanti secara lebih lanjut akan dijelaskan.

Aplikasi *computer vision* yang digunakan dalam pengenalan pola sebelum dilakukannya deteksi objek, adalah proses mengubah citra tersebut agar dapat diproses oleh komputer. Mengubah citra yang didapat dan mengubahnya menjadi citra biner akan menjadi mudah untuk komputer memprosesnya. Citra biner umumnya sebuah citra yang terdiri dari 2 nilai saja didalamnya. Dengan menggunakan nilai batasan atau *threshold*,

intensitas dalam sebuah citra akan dapat diubah menjadi dua nilai saja – lebih atau sama dengan dari *threshold* dan kurang dari *threshold*.

2.2.2 *Pattern Recognition*

Salah satu bentuk aplikasi *computer vision* lainnya adalah kemampuannya untuk mampu melakukan *edge detection*, yaitu bagaimana citra yang menjadi input akan mampu dikenali oleh komputer untuk melakukan pelacakan tepi. Menurut Sri Kusumadewi (2005, p196) tahapan yang dilakukan oleh *edge detection* adalah melalui tiga tahap. Tahap pertama adalah, citra yang diambil dilakukan *smoothing* – atau pengaburan – guna meningkatkan intentsitas *gray scale* yang telah terkontaminasi gangguan *noise*. Pengaburan ini berguna untuk mengurangi efek *noise* untuk terdeteksi, sehingga citra original atau citra yang ingin kita deteksi tidak akan mengalami gangguan. Tahap kedua adalah penguatan pixel, dimana setelah citra telah mengalami proses *smoothing*, citra tersebut akan kehilangan bentuk semula dan akan sussah untuk dideteksi kembali. Penguatan dilakukan pada perubahan intensitas pada lingkungan suatu titik, dimana terdapat perbedaan nyata antara intensitas lokal dan biasanya dilakukan dengan menghitung besaran gradien menurut Sri Kusumadewi (2005, p197). Tahap ketiga adalah pelacakan tepi, dimana titik-titik yang diinginkan adalah titik-titik dengan informasi tepi yang kuat.