IMPLEMENTASI JARINGAN SYARAF TIRUAN DENGAN METODE KOHONEN SOM PADA PENGENALAN POLA OBYEK 2 DIMENSI

SKRIPSI

Oleh:

ALFA ANGGAWASITA

0210960005-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2009

IMPLEMENTASI JARINGAN SYARAF TIRUAN DENGAN METODE KOHONEN SOM PADA PENGENALAN POLA OBYEK 2 DIMENSI

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer
dalam bidang Ilmu Komputer

oleh:
ALFA ANGGAWASITA
0210960005-96

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG

2009


UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

IMPLEMENTASI JARINGAN SYARAF TIRUAN DENGAN METODE KOHONEN SOM PADA PENGENALAN POLA OBYEK 2 DIMENSI

oleh:
ALFA ANGGAWASITA
0210960005-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 6 Agustus 2009
dan dinyatakan memenuhi syarat untuk memperoleh gelar
Sarjana Komputer dalam bidang Ilmu Komputer

Pembimbing

Drs. Marji, MT
NIP. 131 993 386

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya

Dr. Agus Suryanto, Msc
NIP. 132 126 049


UNIVERSITAS BRAWIJAYA

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama	**:**	Alfa Anggawasita
NIM	**:**	0210960005-96
Jurusan	**:**	Matematika / Ilmu Komputer
Penulis Skripsi berjudul	**:**	Implementasi Jaringan Syaraf Tiruan dengan Metode Kohonen SOM pada Pengenalan Pola Obyek 2 Dimensi

Dengan ini menyatakan bahwa :

1. **Isi dari Skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam Skripsi ini.**
2. **Apabila dikemudian hari ternyata Skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 6 Agustus 2009
Yang menyatakan,

Alfa Anggawasita
NIM. 0210960005


UNIVERSITAS BRAWIJAYA

IMPLEMENTASI JARINGAN SYARAF TIRUAN DENGAN METODE KOHONEN SOM PADA PENGENALAN POLA OBYEK 2 DIMENSI

ABSTRAK

Pengenalan pola memiliki beberapa cabang, salah satunya adalah menggunakan pendekatan Jaringan Syaraf Tiruan yang akan mencoba meniru cara berpikir manusia dalam mengelompokkan informasi berdasarkan kedekatan subjektif dari data-data tentang Obyek tersebut. Penelitian ini bertujuan untuk mengimplementasikan teknik Jaringan Syaraf Tiruan khususnya metode Kohonen SOM untuk mengelompokkan Obyek-Obyek ke dalam kelompok-kelompok. Kohonen SOM merupakan pembelajaran unsupervised yang menggunakan kedekatan antar Obyek untuk melakukan pengelompokkan. Selain mengenali Obyek penelitian ini juga digunakan untuk melihat hubungan antara variabel-variabel Kohonen SOM terhadap proses pelatihan.

Data Obyek direpresentasikan dengan mengambil properti geometrinya untuk kemudian dilatih dengan menggunakan Kohonen SOM. Obyek yang digunakan disini adalah citra digital dua dimensi sederhana dan untuk pengujiannnya digunakan Obyek yang sama dengan dikenakan rotasi, refleksi, perbesaran dan pengecilan kemudian akan dilihat seberapa tinggi tingkat ketepatannya.

Hasil pengujian terhadap pengenalan Obyek menunjukkan pengambilan properti geometri dari Obyek yang mengalami rotasi dan refleksi dapat menyebabkan perbedaan data bila dibandingkan dengan Obyek semula sehingga menyebabkan pengenalan tidak sepenuhnya akurat. Untuk hubungan antara variabel-variabel Kohonen SOM menunjukkan penggunaan α yang kecil akan sangat baik karena penyesuaian bobot pada tetangga BMU akan dapat mengikuti penyesuaian bobot pada BMU.

ARTIFICIAL NEURAL NETWORK IMPLEMENTATION USING KOHONEN SOM METHODE FOR 2 DIMENSION OBJECT RECOGNITION

ABSTRACT

Pattern recognition had a few branch, one of them using Artificial Neural Network approach which is trying to replicate how human think on the way to organize and clustering information based on the subjective similarity from object's data. This observation aimed to implementing Artificial Neural Network specifically Kohonen SOM clustering objects to clusters. Kohonen SOM is an unsupervised learning that using similarity between the object to do clustering. Besides recognizing this observation also observing the relationship between the Kohonen SOM variables affecting training process.

Object data are represented by geometri property to be trained using Kohonen SOM. Object that being used here are simple two dimension digital image and for testing the same object are used again, but it will have some operation like rotation, reflection, maximize and minimize then observing the accuracy for the recognizing.

The testing result for recognizing showed that geometri property from rotated and reflected object may cause a differentiation of data compared from the original object and resulting less accuracy. For the relationship between the Kohonen SOM variables showed the used of minimum α had a very good result for training because the adjusting weight for neighbour of the BMU could follow the BMU adjusting weight.

KATA PENGANTAR

Alhamdulillah, segala puji dan syukur kehadirat Allah SWT atas nikmat, rahmat hidayah yang telah diberikan-Nya sehingga penulis dapat menyelesaikan Skripsi ini dengan judul **"IMPLEMENTASI JARINGAN SYARAF TIRUAN DENGAN METODE KOHONEN SOM PADA PENGENALAN POLA OBYEK 2 DIMENSI"**

Dalam proses penyusunan Skripsi ini penulis banyak menerima bantuan dari berbagai pihak. Oleh karena itu pada kesempatan ini penulis banyak mengucapkan terima kasih kepada :

1. Bapak Drs. Marji, MT, selaku dosen pembimbing yang telah banyak membantu dan meluangkan waktunya untuk membimbing penulis hingga terselesainya Skripsi ini.
2. Bapak Wayan Firdaus Mahmudy, S.Si, MT, selaku Ketua Program Studi Ilmu Komputer, Jurusan Matematika, Fakultas MIPA, Universitas Brawijaya.
3. Bapak Dr. Agus Suryanto, Msc., selaku Ketua Jurusan Matematika FMIPA Universitas Brawijaya.
4. Segenap bapak dan ibu dosen di Fakultas MIPA Universitas Brawijaya yang selama ini membimbing dan memberikan bekal ilmu kepada penulis.
5. Seluruh staf dan karyawan di Fakultas MIPA Universitas Brawijaya yang telah banyak membantu penulis dalam pelaksanaan penyusunan Skripsi ini.
6. Serta pihak yang secara langsung maupun tidak langsung telah membantu penulis demi kelancaran penulisan Skripsi ini.

Penulis menyadari sepenuhnya bahwa penyusunan Skripsi ini masih jauh dari sempurna dan oleh karenanya dengan segala kerendahan hati penulis menerima tanggapan baik berupa kritik maupun saran yang membangun demi penyempurnaan penulisan Skripsi ini.

Akhirnya penulis berharap agar Skripsi ini dapat memberikan sumbangan serta manfaat bagi semua pihak yang berkepentingan.

Malang, 3 Agustus 2009

Penulis

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR TABEL

Halaman

DAFTAR *SOURCECODE*

Halaman

DAFTAR LAMPIRAN

Halaman

BAB I
PENDAHULUAN

1.1 Latar Belakang

Di dunia ini terdapat bermacam-macam informasi yang beragam. Akan tetapi dari keragaman ini bila diperhatikan lebih lanjut terdapat beberapa kesamaan antara satu dengan yang lainnya. Kesamaan-kesamaan ini sering kali membentuk kelompok-kelompok berdasarkan tingkat kesamaannya tersebut yang selanjutnya disebut dengan pola (Schalkoff, 1992). Pola sendiri adalah entitas yg terdefinisi dan dapat diidentifikasi melalui ciri-cirinya (*features*) (Munir, 2004).

Pengenalan pola itu sendiri khususnya berkaitan dengan langkah pengklasifikasian. Dalam kasus tertentu, sebagaimana dalam jaringan syaraf tiruan, pemilihan ciri-ciri, pengklasifikasian dan akhirnya pemrosesan berdasarkan kelas pengenalan. Contoh pengenalan ini adalah proses identifikasi tulisan tangan, tanda tangan, suara, wajah, pengklasifikasian dan lain sebagainya.

Karena kemampuannya sering kali manusia dapat dengan mudah mengelompokkan pola berdasarkan kemiripannya (Patterson, 1996), tetapi dibalik kemudahan tersebut manusia juga memiliki keterbatasan untuk memberikan hasil yang tepat bila menghadapi situasi-situasi tertentu. Misalkan bila informasi yang dianalisa relatif besar akan sangat memungkinkan untuk terjadinya kesalahan karena ketidaktelitian dan juga penggunaan waktu yang lama sehingga menjadikannya tidak efisien. Untuk itu perlu diciptakan cara yang dapat mengatasi masalah ini sekaligus juga dapat melakukan efisiensi waktu.

Mengingat betapa mudahnya manusia mengelompokkan pola-pola bukan berarti ini adalah suatu hal yang dapat dengan mudah diotomatisasikan. Ini disebabkan betapa rumitnya cara berpikir dan proses belajar manusia untuk diimplementasikan dan ditirukan oleh sebuah mesin. Oleh karena itu berkembang sebuah bidang ilmu yang disebut Jaringan Syaraf Tiruan (JST), yang mencoba mempelajari dan menirukan cara berpikir manusia untuk diimplementasikan pada sebuah alat.

JST digunakan dalam pengenalan pola untuk mengenali obyek sedemikian hingga obyek tersebut masih dapat dikenali meskipun telah mengalami perubahan-perubahan.

JST sendiri memiliki beberapa metode yang dapat digunakan untuk mengelompokkan pola, salah satu metode tersebut adalah Kohonen SOM (*self organizing map*) yang ditemukan oleh Professor Teuvo Kohonen. Ini merupakan jenis pembelajaran tanpa pengawasan (*unsupervised*) yang memungkinkan pembelajaran dilakukan tanpa diberikan *output* yang benar.

Penggunaan SOM untuk pengenalan citra sebelumnya pernah dilakukan dengan judul "Pengenalan Citra Obyek Sederhana dengan Menggunakan Metode Jaringan Saraf Tiruan SOM" (Ang Wei Siong, 1999). Dalam penelitian ini *input* pada SOM diberikan langsung dari tiap-tiap *pixel* tanpa melalui pengolahan lebih dahulu.

Dengan cirinya yang *unsupervised* yaitu pada saat pelatihan tidak disertakan *output* yang diinginkan, sebuah nilai *error* untuk Kohonen merupakan sesuatu yang subjektif sehingga berbeda dengan jenis metode yang lain (Heaton, 2005), bahkan kadangkala *error* tidak diperhitungkan untuk model ini. Untuk Kohonen SOM ini jaringan akan dilatih sehingga bobot pada jaringan relatif tetap, sehingga kemudian akan dapat digunakan dalam pengelompokkan dan pengenalan.

Obyek yang digunakan dalam pengenalan ini adalah bangun-bangun 2 dimensi sederhana yaitu bangun-bangun yang letak pusat massa (momen) obyeknya masih berada di dalam tepi obyeknya, untuk lebih jelasnya tentang momen akan dibahas pada bab selanjutnya.

1.2 Rumusan Masalah

Berdasarkan latar belakang tersebut maka rumusan masalah dalam penelitian yang akan dikerjakan adalah:

1. Bagaimana menerapkan Kohonen SOM untuk mengelompokkan obyek-obyek.
2. Melakukan analisa beberapa aspek dari Kohonen SOM dan ekstraksi ciri sehingga menghasilkan nilai parameter yang efisien untuk menghasilkan jaringan yang konvergen dengan jumlah ekstraksi data tertentu.

1.3 Batasan Masalah

Hal-hal yang dibatasi pada penelitian ini adalah sebagai berikut:

1. Model Jaringan Syaraf Tiruan yang digunakan adalah tipe *unsupervised* Kohonen SOM
2. Mengelompokkan obyek yang belum dikenali berdasarkan pola-pola yang didapat melalui pelatihan jaringan dengan Kohonen SOM.
3. Obyek yang digunakan adalah citra biner dua dimensi dengan pusat momen berada di dalam obyek.
4. Perubahan yang dikenakan pada obyek untuk pengujian adalah rotasi, refleksi, perbesaran atau pengecilan serta objek yang mengalami cacat

1.4 Tujuan Penelitian

Tujuan yang akan dicapai pada penelitian ini adalah:

1. Mengimplementasikan Kohonen SOM untuk melakukan pengelompokan
2. Menganalisa metode Kohonen SOM yang digunakan dalam penelitian ini dan ekstraksi ciri yang dilakukan menggunakan pengolahan citra digital.

1.5 Manfaat Penelitian

Berdasarkan tujuan dari tulisan ini yaitu secara umum adalah mengimplementasikan metode yang diharapkan dapat memudahkan pekerjaan manusia untuk mengelompokkan pola-pola dengan mengotomatisasikan cara kerja sistem yang dapat meniru cara berpikir manusia dan secara khusus menganalisa metode yang digunakan, maka penelitian ini diharapkan dapat menjadi dasar pengembangan untuk penciptaan sistem yang lebih sempurna dengan menghindari kekurangan dari sistem yang dipaparkan pada analisa tulisan ini.


UNIVERSITAS BRAWIJAYA

BAB II
TINJAUAN PUSTAKA

2.1 Pengenalan Pola

2.1.1 Definisi

Dengan kemampuannya manusia dapat dengan mudah membedakan informasi dan mengelompokkannya berdasarkan pola kemiripannya. Hal ini disebabkan manusia memiliki daya ingat dan kemampuan belajar sehingga dapat mengolah informasi-informasi tersebut. Pengenalan pola dapat dikarakteristikkan sebagai reduksi informasi, pemetaan informasi atau proses pelabelan informasi (Patterson, 1996).

Pengenalan pola adalah bidang penelitian yang mempelajari kerja dan desain sistem yang mengenali pola di dalam data. Melingkupi bidang-bidang seperti analisa diskriminan, *feature extraction*, estimasi *error*, analisa *cluster* (yang bersama-sama disebut pengenalan pola *statistical*), inferensi gramatikal dan parsing (kadangkala disebut pengenalan pola *syntactical*). Pengenalan pola secara alami berdasarkan pada pola. Sebuah pola dapat menjadi sekelompok hasil pengukuran atau observasi, kemungkinan diwakili oleh notasi vektor atau matrix. Penggunaan pengukuran juga termasuk tahapan sebelum pemrosesan dan pengukuran kompleksitas system. Pengukuran ini dapat berupa entitas-entitas seperti tekanan darah, usia, jumlah roda dsb. Lebih jauh lagi pola dapat dikonversikan dari representasi yang satu ke representasi yang lain (Schalkoff, 1992).

Pengenalan pola bertujuan menentukan kelompok atau kategori pola berdasarkan ciri-ciri yg dimiliki oleh pola tersebut. Dengan kata lain, pengenalan pola membedakan suatu obyek dengan obyek yang lain. Terdapat dua pendekatan yang dilakukan dalam pengenalan pola: pendekatan secara statistik dan pendekatan secara sintaktik atau struktural (Munir, 2004).

2.1.2 Pengenalan Pola secara Statistik

Pendekatan ini menggunakan teori-teori ilmu peluang dan statistik. Ciri-ciri yang dimiliki oleh suatu pola ditentukan distribusi statistiknya. Pola yang berbeda memiliki distribusi statistik yang berbeda pula sehingga dapat digunakan untuk mengklasifikasikan pola (Munir, 2004).

Terdapat beberapa tahap dalam pengenalan pola:

1. *Feature extraction*, proses pengambilan ciri-ciri yang terdapat pada obyek di dalam citra. Pada proses ini obyek di dalam citra terlebih dahulu dideteksi seluruh tepinya dan dihitung properti-properti obyek yang berkaitan sebagai cirinya.
2. *Learning*, proses belajar membuat aturan klasifikasi sehingga jumlah kelas yang tumpang tindih dibuat sekecil mungkin., proses *learning* disini menggunakan Kohonen SOM.

2.2 *Feature Extraction*

Feature extraction dapat berarti menemukan representasi baru dari ciri (Aksoy, 2007). *Feature extraction* digunakan dalam mensederhanakan penggunaan jumlah data yang dibutuhkan untuk menggambarkan sekumpulan data yang lebih besar tapi dengan hasil yang masih relatif akurat. Saat melakukan analisa data yang kompleks, salah satu kendala adalah besarnya jumlah variabel yang terlibat. Analisa dengan menggunakan jumlah variabel yang besar umumnya membutuhkan komputasi yang juga lebih besar. *Feature extraction* secara umum adalah metode menciptakan kombinasi dari variabel berikut kerumitannya tersebut dalam usaha menggambarkan data dengan keakuratan yang cukup.

Feature extraction dibagi dalam dua proses utama yaitu segmentasi obyek dan penentuan properti geometri.

2.2.1 Segmentasi Obyek

Proses segmentasi bertujuan mengelompokkan *pixel-pixel* obyek agar terpisah dari latar belakang dan obyek-obyek lainnya.

Untuk memisahkan obyek dari latar belakang digunakan segmentasi berdasarkan batas wilayah (tepi dari obyek). *Pixel-pixel* tepi ditelusuri sehingga rangkaian *pixel* yang menjadi batas antara

obyek dengan latar belakang dapat diketahui secara keseluruhan (algoritma *boundary following*) (Munir, 2004). Metode pendeteksian batas wilayah salah satunya adalah secara topologi, yaitu dengan memeriksa setiap tetangga dari *pixel* tepi.

2.2.2 Properti Geometri

Setelah obyek-obyek tersebut disegmentasi akan dilakukan analisa untuk mengenali obyek tersebut. Analisa ini didasarkan pada ciri (*feature*) geometri obyek tersebut. Ada dua kelompok ciri pada obyek (Jain, 1995) :

1. *Global Feature*, yaitu ciri khas keseluruhan obyek.
2. *Local Feature*, yaitu ciri khas bagian tertentu dari obyek.

Yang termasuk dalam *global feature*, antara lain :

1. Luas atau ukuran obyek (*A*)

$$A = \sum_{i=1}^{n}\sum_{j=1}^{m} f(i,j) \tag{2.1}$$

$f(i, j)$ = 1 jika (i, j) adalah pixel obyek

2. Pusat Massa

$$\bar{x} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} j.f(i.j)}{A} \tag{2.2}$$

$$\bar{y} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m} i.f(i.j)}{A} \tag{2.3}$$

Sehingga pusat massa dari obyek tersebut akan berada pada posisi koordinat ($\bar{x}, \bar{y}$).

Sedangkan yang termasuk *local feature* salah satunya adalah *object signature* yaitu menyatakan jarak dari pusat massa ke tepi suatu obyek pada arah 0 sampai 360 derajat.

2.2 *Learning*

Proses *learning* disini menggunakan pendekatan Jaringan Syaraf Tiruan (JST) lebih khususnya sendiri adalah menggunakan model Kohonen *Self Organizing Map* (SOM).

2.2.1 Jaringan Syaraf Tiruan (JST)

Untuk meniru kemampuan belajar manusia ini JST dikembangkan agar dapat dibuat sistem yang dapat mengotomatisasi pekerjaan untuk mengenali pola ini. Untuk mengenali obyek yang identik sama, sistem yang ada sekarang sudah mampu melakukannya dengan relatif sempurna. Tetapi bila obyek yang akan dikenali mengalami sedikit perubahan, sistem yang hanya mengenali obyek secara identik ini tidak akan dapat mengenali lagi. Salah satu peran JST adalah dalam menyelesaikan persoalan seperti tersebut di atas.

Berbicara mengenai JST sendiri sudah merupakan bidang ilmu yang memiliki cakupan yang sangat luas. Karena luasnya cakupan JST sering terjadi tumpang tindih antara bidang JST dengan statistika (Sarle, 2002). Statistika menekankan pada analisa data, yang dalam pengertian JST berarti belajar menggeneralisasi data yang memiliki *noise*. Memang terdapat model-model JST yang hanya dapat belajar secara baik hanya dari data yang bebas *noise*, tapi kebanyakan JST yang dapat belajar untuk menggeneralisasi data ber-*noise* secara efektif hampir sama dengan metode statistika.

JST sendiri memiliki makna meniru cara berpikir otak manusia melalui pengaturan bobot dan koneksi antar *neuron* yang akan menentukan *output*. Cara kerja JST adalah menggunakan sekelompok elemen proses atau *node-node* yang hampir sama dengan otak manusia, *node-node* ini berhubungan dalam sebuah jaringan yang dapat mengidentifikasi pola yang ditunjukkan dalam kumpulan data.

Ditinjau dari segi pembelajarannya terdapat dua macam tipe pembelajaran JST yaitu:

1. *Supervised Learning* atau pembelajaran dengan pengawasan, yaitu selama proses pelatihan bobot pada *neuron*, *output* yang benar selalu diikutsertakan sebagai kendali *error*.

2. *Unsupervised Learning* atau pembelajaran tanpa pengawasan, sesuai namanya selama proses pelatihan tidak diberikan *output* yang seharusnya, tipe ini belajar dari perubahan yang terjadi selama pelatihan dan kedekatan hubungan antar *input* yang digunakan selama pelatihan.

Ditinjau dari jumlah *layer* yang digunakan dibagi menjadi:

1. *Single layer*, yaitu tidak terdapat *hidden layer*. *Input* dan *output* terkoneksi langsung tanpa ada perantara.
2. *Multi layer*, terdapat *hidden layer* sehingga *input* dan *output* tidak terhubung langsung. *Layer* ini dapat berjumlah satu atau lebih, biasanya digunakan untuk menyelesaikan permasalahan yang lebih kompleks.

Metode yang digunakan dalam penelitian ini sendiri adalah Kohonen SOM, Kohonen ini tergolong kedalam *unsupervised learning* dan struktur jaringannya tidak menggunakan *hidden layer*.

2.2.2 Kohonen

Teuvo Kohonen adalah salah satu diantara sekian banyak peneliti di bidang *neurocomputing*, dan telah menemukan variasi-variasi jaringan. Tapi banyak penggunaan yang menunjuk pada "jaringan Kohonen" tanpa menspesifikasikan jenis jaringan kohonen yang mana yang dimaksud, ini memberikan informasi yang kurang tepat dan dapat mengarah pada kebingungan (Sarle, 2002).

Frase "jaringan Kohonen" seringkali menunjuk pada salah satu diantara tiga tipe jaringan berikut:

1. *Vector Quantization* (VQ)
2. *Self-Organizing Map* (SOM)
3. *Learning Vector Quantization* (LVQ)

Vector Quantization atau biasa disingkat VQ adalah merupakan tipe jaringan kohonen yang paling mendasar. Jaringan ini adalah jaringan dimana *output* berkompetisi untuk mewakili pola *input*. Unit dengan bobot vector terdekat dengan *input* menjadi pemenang dalam kompetisi yang sering disebut kompetisi "*winner-takes-all*". Neuron yang berkompetisi memperkuat penguatan dirinya melalui hubungan umpan-baliknya sendiri dan mencegah unit kompetisi yang lain

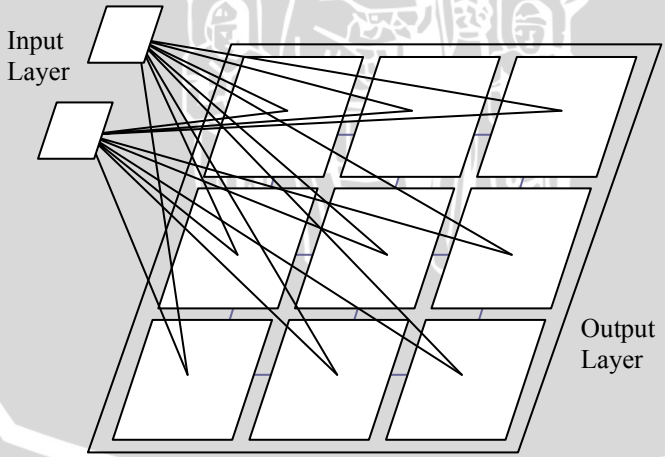
menjadi pemenang. Unit dengan penguatan *input* terkuat memenangkan kompetisi (Patterson, 1996).

Self Organizing Map atau Disingkat SOM merupakan pengembangan dari VQ. Seperti juga VQ, SOM melakukan kompetisi di antara vektor-vektor bobotnya sehingga dihasilkan sebuah neuron pemenang. Letak perbedaan dengan VQ adalah bobot neuron yang mengalami penguatan tidak hanya neuron yang menjadi pemenang tapi juga neuron tetangganya.

Pengembangan dari VQ yang lain adalah Learning Vector Quantization disingkat LVQ. Yang membuat jaringan ini berbeda dengan jaringan kohonen pada umumnya jaringan belajar dengan pengawasan atau disebut juga *supervised* tidak seperti jaringan kohonen lainnya yang bertipe *unsupervised*.

2.2.3 Self-Organizing Map (SOM)

SOM merupakan cabang JST yang pembelajarannya bertipe *unsupervised*. Bila dilihat dari segi arsitekturnya SOM hanya terdiri dari sebuah *layer* tunggal atau sering disebut *single layer*, dengan setiap *input* terhubung ke semua *output layer*. Disertakan pada setiap *node* adalah vektor bobot dengan dimensi yang sama dengan dimensi vektor *input*.



Gambar 2.1 Arsitektur Jaringan Kohonen SOM

SOM tergolong sebagai *competitive learning* atau "*winner-takes-all*" dimana hanya akan ada sebuah *neuron* yang akan keluar sebagai pemenang. Tetapi untuk SOM fokusnya tidak hanya pada *neuron* yang jadi pemenang tapi bobot untuk *neuron-neuron* tetangganya (*neighbour*) juga akan disesuaikan sehingga akan membentuk kelompok (*cluster*) sesuai jumlah pola yang dilatihkan.

Arsitektur jaringan Kohonen SOM sesuai dengan Gambar 2.1 adalah contoh jaringan Kohonen SOM dengan jumlah *node* 3 x 3 yang terhubung dengan *input layer* berdimensi dua. Masing-masing *node* memiliki posisi topologi yang spesifik yaitu koordinat x dan y pada *map* Kohonen dan didalamnya terdapat data yang untuk JST disebut sebagai bobot (*weight*) dengan jumlah dimensi yang sama dengan *input*. Bobot dengan jumlah dimensi tertentu ini selanjutnya akan disebut sebagai *vector*.

Kohonen SOM tidak menspesifikasikan *output* berdasarkan *input* yang diberikan padanya, tapi *node output* dipilih berdasarkan kedekatannya dengan *input* yang diberikan. Kedekatan ini dihitung dengan persamaan jarak *Euclidean* berikut (AI-Junkie, 2007):

$$dist = \sqrt{\sum_{i=0}^{i=n} (v_i - w_i)^2} \qquad (2.4)$$

dimana V_i = *input vector* ke-*i*
W_i = bobot *node vector* ke-*i*
n = dimensi dari *vector*
dist = jarak *input vector* dengan *vector bobot*

Karena SOM tidak memiliki *output* yang ditentukan maka *node* yang paling mendekati *input* akan terpilih yang disebut sebagai *Best Matching Unit* (BMU) untuk kemudian dioptimasi sebanyak *t* iterasi agar *vector* bobotnya mendekati data *vector input*.

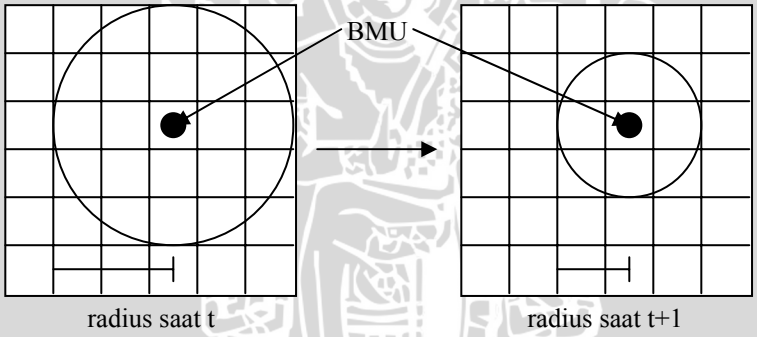
Dalam penyesuaian bobot pada *map* Kohonen ini terdapat variabel yang bernilai tetap dan yang berubah menurut waktu, yang dimaksud waktu disini adalah ukuran suatu perubahan yang dalam penerapannya adalah jumlah iterasi (*t*). Variabel yang tetap disini adalah *Lambda* yang selanjutnya akan disimbolkan sebagai λ, disini

gunanya adalah sebagai konstanta waktu yang akan digunakan oleh variabel-variabel yang berubah mengikuti waktu.

$$\lambda = t_{max} / \log 10(r_0) \qquad (2.5)$$

dimana t_{max} = iterasi maximum
r_0 = radius mula-mula

Sedangkan variabel yang berubah mengikuti waktu adalah radius tetangga dari BMU. Keistimewaan Kohonen SOM adalah selain BMU, tetangga dari BMU itu sendiri juga mendapatkan penyesuaian bobot mendekati bobot dari BMU tempatnya berada. Yang dimaksud dengan tetangga disini adalah *node-node* yang berada di dalam radius dari suatu BMU, dan radius BMU itu sendiri akan menyusut mengikuti waktu seperti diperlihatkan pada Gambar 2.2



Gambar 2.2 Penyusutan radius BMU

Radiusnya sendiri dapat dihitung dengan menggunakan persamaan berikut

$$r_t = r_0 \times 10^{\left(\frac{-t}{\lambda}\right)} \qquad t = 1,2,3... \qquad (2.6)$$

r_0 = radius mula-mula
t = iterasi ke sekian

λ = konstanta waktu dari Persamaan 2.5

Selain itu juga masih terdapat variabel lain yang berubah mengikuti waktu yaitu *learning rate* (α) dengan cara penghitungan yang kurang lebih sama dengan cara perhitungan radius. Rumusnya diberikan oleh Persamaan 2.7

$$\alpha_t = \alpha_0 \times 10^{\left(\frac{-t}{\lambda}\right)} \qquad t = 1,2,3... \qquad (2.7)$$

α_0 = *learning rate* mula-mula
t = iterasi ke sekian
λ = konstanta waktu dari Persamaan 2.5

Dalam penyesuaian bobot terdapat faktor lain yang juga diperhitungkan yaitu kedekatan jaraknya dengan BMU. Dalam Kohonen SOM semakin dekat jarak suatu node tetangga dengan BMU maka *vector* bobotnya akan mendapatkan penyesuaian yang semakin besar demikian pula bila semakin jauh dari BMU penyesuaiannya akan makin kecil. Faktor jarak ini disebut dengan *distance factor* dan akan disimbolkan δ dan persamaannya adalah

$$\delta = 10^{\left(\frac{-dist^2}{2r_t^2}\right)} \qquad t = 1,2,3... \qquad (2.8)$$

dist = jarak node dari BMU
r_t = r pada saat ke-t, didapat dari Persamaan 2.6

Setelah semua variabel yang diperlukan diketahui maka penyesuaian bobot dapat dilakukan.

$$W_{t+1} = W_t + (\delta_t . \alpha_t . (V_t - W_t)) \qquad (2.9)$$

Dimana: V_t = *input vector*
Wt = *vector* bobot

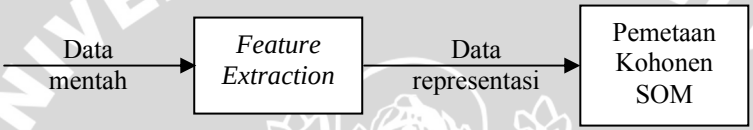
Dari sejumlah distribusi bobot acak dan setelah melalui sekian iterasi Kohonen SOM akhirnya akan membentuk *map* dengan bobot

node yang relatif stabil, hingga akhirnya *input* yang sebelumnya belum pernah diberikan akan menghasilkan *output* menurut *node* dengan bobot yang paling mendekati. Langkah pembelajaran dapat digambarkan sebagai berikut:

1. Inisialisasi acak tiap bobot *node*.
2. *Input vector* dari sekumpulan data pelatihan diberikan *map* Kohonen.
3. Setiap *node* diperiksa untuk menghitung bobot *node* yang mana yang paling mendekati *input vector*, *node* pemenang ini kemudian akan disebut sebagai Best Matching Unit (BMU).
4. Radius tetangga (*neighbour*) dari BMU dihitung, dimulai dengan nilai yang besar yang semakin berkurang mengikuti bertambahnya iterasi. Setiap *node* yang terletak di dalam radius ini disebut tetangga dari BMU.
5. Bobot BMU dan tetangga disesuaikan agar mendekati *vector input*, semakin dekat dengan BMU semakin besar perubahan bobotnya.
6. Ulangi dari langkah 2 untuk N iterasi.

BAB III
METODOLOGI DAN PERANCANGAN SISTEM

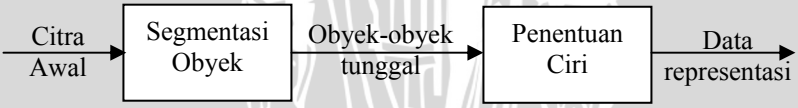
Untuk dapat melakukan pengenalan pola dibutuhkan data yang akan dilatihkan. Data ini didapat dengan melakukan *feature extraction* pada data mentah, tujuannya adalah agar didapat data yang dapat mewakili sekaligus membuang informasi-informasi yang tidak diperlukan. Kemudian data yang mewakili ini dilatihkan dengan menggunakan metode Kohonen SOM.



Gambar 3.1 Tahapan umum proses

3.1 *Feature Extraction*

Proses *feature extraction* menggunakan pengolahan citra digital karena data yang digunakan dalam bentuk citra digital. Pada citra digital sebuah citra tersusun dari kumpulan *pixel-pixel* yang teratur sedemikian rupa hingga membentuk persepsi yang dapat ditangkap oleh indera penglihatan. Proses ini terdiri dari segmentasi obyek dan penentuan properti geometrinya.

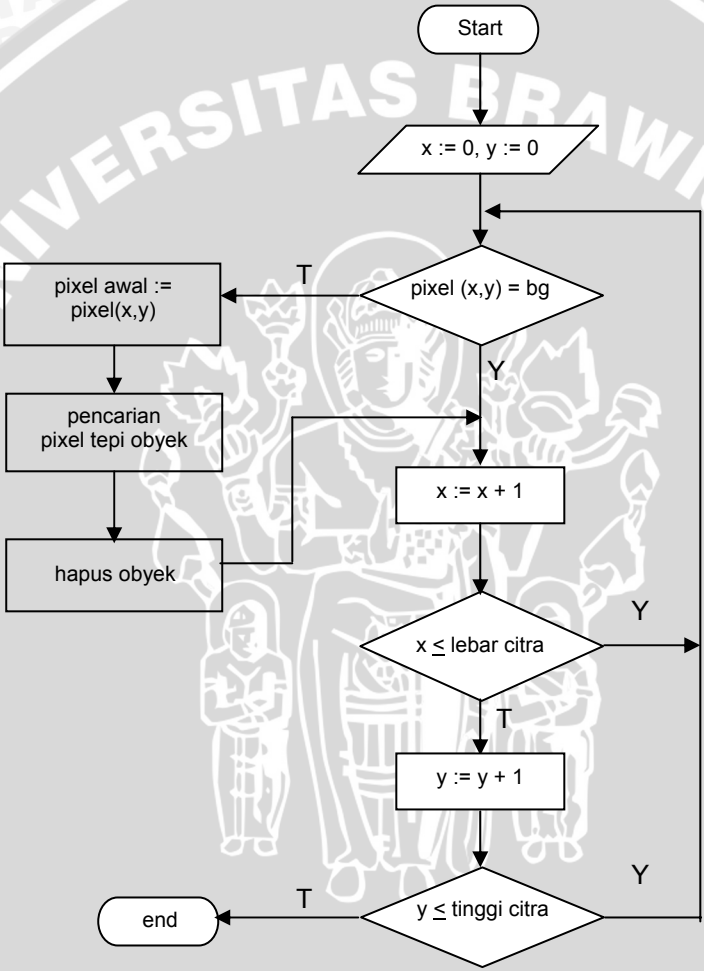


Gambar 3.2 Tahapan *feature extraction*

3.1.1 Segmentasi Obyek

Segmentasi digunakan memisahkan obyek dari latar belakang dengan menelusuri rangkaian *pixel* yang menjadi batas tepi obyek.

Setelah ditemukan *pixel* awal untuk suatu obyek, *pixel* awal tersebut kemudian digunakan sebagai acuan awal untuk mencari *pixel-pixel* tepi yang lainnya. Proses segmentasi lebih jelasnya digambarkan melalui *flowchart* pada Gambar 3.3 berikut:



Gambar 3.3 *flowchart* segmentasi obyek

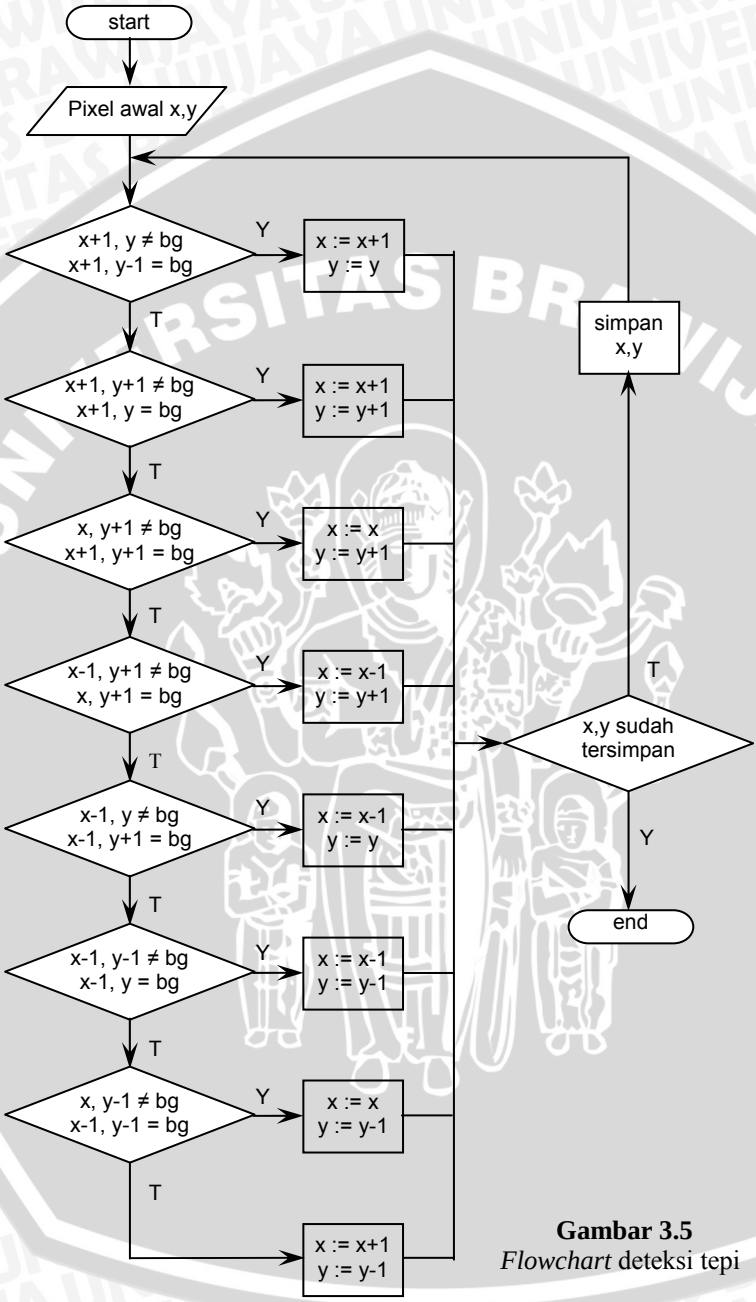
Kelompok 9 *pixel* bertetangga diperiksa sesuai urutan pada Gambar 3.4 dengan *x* adalah *pixel* awal obyek.

6	7	8
5	***x***	1
4	3	2

Gambar 3.4 Arah Pencarian *Pixel* Tepi

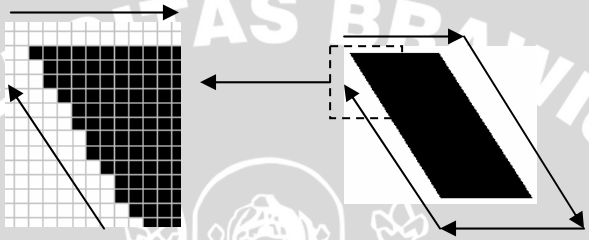
Algoritma pencarian *pixel* tepi:

```
1. if (pixel 1 <> background)
2. and (pixel 8 = background) then
3.    pixel x = pixel 1
4. else if (pixel 2 <> background)
5. and (pixel 1 = background) then
6.    pixel x = pixel 2
7. else if (pixel 3 <> background)
8. and (pixel 2 = background) then
9.    pixel x = pixel 3
10.else if (pixel 4 <> background)
11.and (pixel 3 = background) then
12.   pixel x = pixel 4
13.else if (pixel 5 <> background)
14.and (pixel 4 = background) then
15.   pixel x = pixel 5
16.else if (pixel 6 <> background)
17.and (pixel 5 = background) then
18.   pixel x = pixel 6
19.else if (pixel 7 <> background)
20.and (pixel 6 = background) then
21.   pixel x = pixel 7
22.else if (pixel 8 <> background)
23.and (pixel 7 = background) then
24.   pixel x = pixel 8
```



Gambar 3.5
Flowchart deteksi tepi

Dengan algoritma di atas, arah pencarian *pixel* adalah searah putaran jarum jam. *Pixel* tepi yang baru kemudian diperiksa apakah sudah tersimpan bila belum simpan *pixel* tepi yg baru dan lanjutkan pencarian tapi bila sudah tersimpan hentikan pencarian. Untuk lebih jelasnya *flowchart* pada Gambar 3.5 menunjukkan proses pencarian *pixel* tepi. Gambar 3.6 menunjukkan proses pencarian *pixel* tepi berdasarkan *flowchart* pada Gambar 3.5.

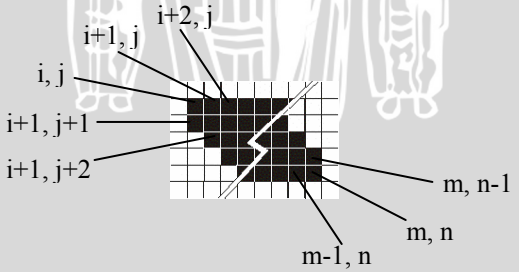


Gambar 3.6 Proses pencarian *pixel* tepi

Setelah seluruh *pixel* tepi obyek tersimpan, obyek akan dihapus dari citra dan pencarian *pixel* awal untuk obyek berikutnya akan dimulai kembali sampai tidak ada lagi obyek yang tersisa.

3.1.2 Penentuan Ciri Karakteristik (*Feature*)

Untuk menentukan ciri obyek disini digunakan properti geometri yaitu menghitung luas obyek dan menentukan pusat massa untuk kemudian dicari *object signature* yang akan dijadikan ciri karakteristik dari masing-masing obyek.



Gambar 3.7 *pixel* obyek beserta koordinatnya

Untuk menghitung luas obyek (*A*) digunakan Persamaan 2.1, dengan luas obyek adalah

$A = pixel\ (i, j) + pixel\ (i+1, j) + pixel\ (i+2, j) + ... + pixel\ (m, n)$
$= 1 + 1 + 1 + ... + 1$

Selanjutnya mencari pusat massa obyek menggunakan Persamaan 2.2 dan Persamaan 2.3

$$\bar{x} = \frac{i + (i+1) + (i+2) + ... + (m-1) + m}{A}$$

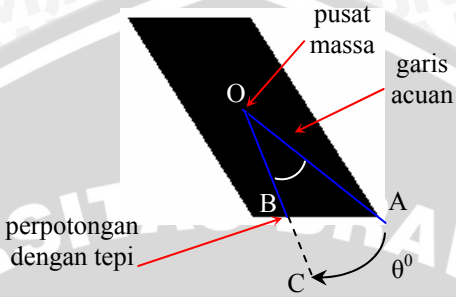
$$\bar{y} = \frac{j + j + j + ... + n + n}{A}$$

Didapatkan pusat massa obyek adalah $(\bar{x}, \bar{y})$

Object signature didapatkan dengan mengukur jarak dari pusat massa ke tepi-tepi obyek dari sudut 0^0 sampai 360^0. Semakin banyak titik yang diambil ciri karakteristik obyek akan semakin mencerminkan obyeknya yang artinya cirinya akan semakin presisi. Jarak antara pusat massa dengan *pixel* obyek terjauh akan dijadikan acuan, selanjutnya dipilih sekian titik yang akan digunakan sebagai ciri untuk merepresentasikan obyek.

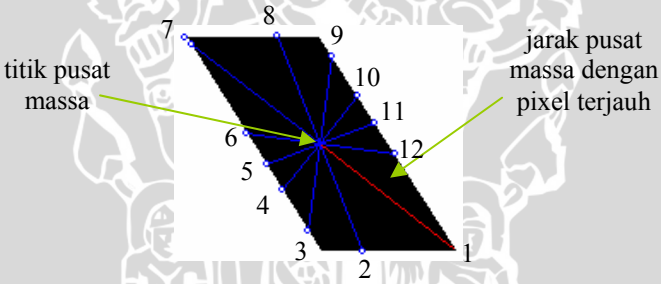
Titik 0^0 digunakan oleh jarak terjauh *pixel* dengan pusat massa dan kemudian dipilih n titik yang membagi 360^0 menjadi sudut-sudut yang sama besar, dalam hal ini adalah $360^0/n = \theta^0$ sudut yang sama besar. Garis acuan dirotasikan sebesar θ^0 dan jarak perpotongan garis acuan dengan tepi akan menjadi salah satu ciri karakteristik obyek. Sesuai Gambar 3.8 garis acuan adalah segmen garis OA dan untuk ciri pertama adalah panjang garis OA dibagi dengan panjang garis acuan yang hasilnya adalah 1. Kemudian garis OA dirotasikan sebesar θ^0 dengan pusat rotasi pada pusat massa hingga didapatkan garis OC. Untuk ciri selanjutnya dilihat perpotongan garis OC dengan tepi obyek dan didapatkan garis OB, maka ciri selanjutnya dihitung dengan membagi panjang garis OB/OA. Nilai ciri

selanjutnya karena merupakan nilai perbandingan mempunyai rentang antara 0 dan 1.



Gambar 3.8 proses pencarian ciri obyek

Contoh pada Gambar 3.9 adalah obyek dengan penggunaan 12 titik sebagai cirinya.



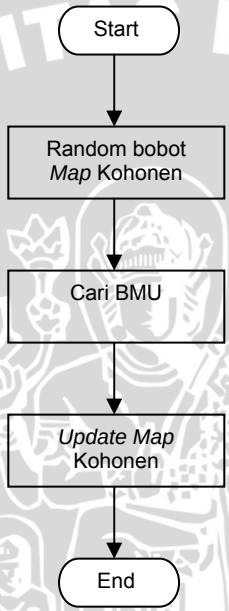
Gambar 3.9 penentuan ciri obyek dengan 12 titik

Dari cara tersebut didapatkan karakteristik obyek misalkan adalah 1.0, 0.7, 0.5, 0.3, 0.3, 0.4, 0.9, 0.7, 0.5, 0.3, 0.3, dan 0.4.

3.2 Pelatihan Bobot Pada Kohonen SOM

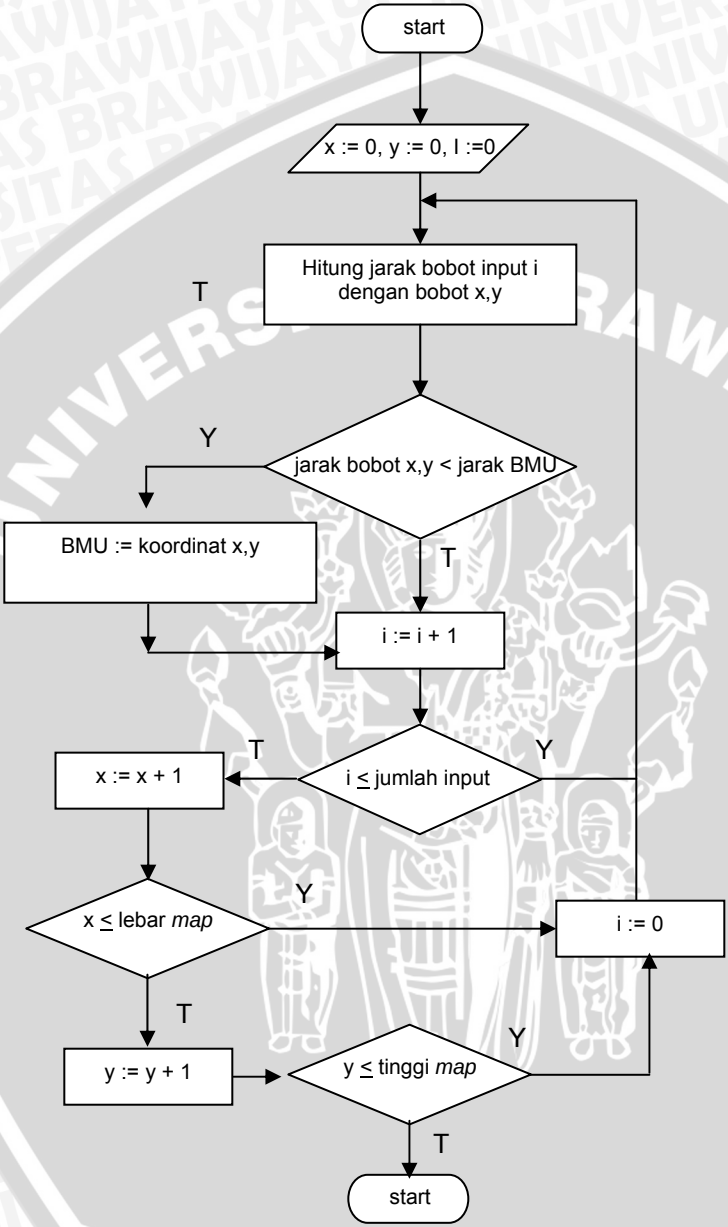
Ciri karakteristik obyek-obyek yang didapatkan melalui proses *feature extraction* akan digunakan sebagai *input training* untuk mengelompokkan *map* Kohonen SOM. *Node* yang paling mendekati *input training* akan menjadi BMU dan *node-node* di sekitarnya yg

masih dalam radius tertentu disebut sebagai tetangganya. Setiap *node* pada *map* Kohonen SOM akan dikelompokkan menjadi sebanyak *input training* yang diberikan dan *vector* bobot dari setiap *node* akan disesuaikan mendekati *input training* dimana *node* tersebut menjadi tetangganya. Proses pelatihan Kohonen SOM secara umum ditunjukkan oleh *flowchart* pada Gambar 3.10



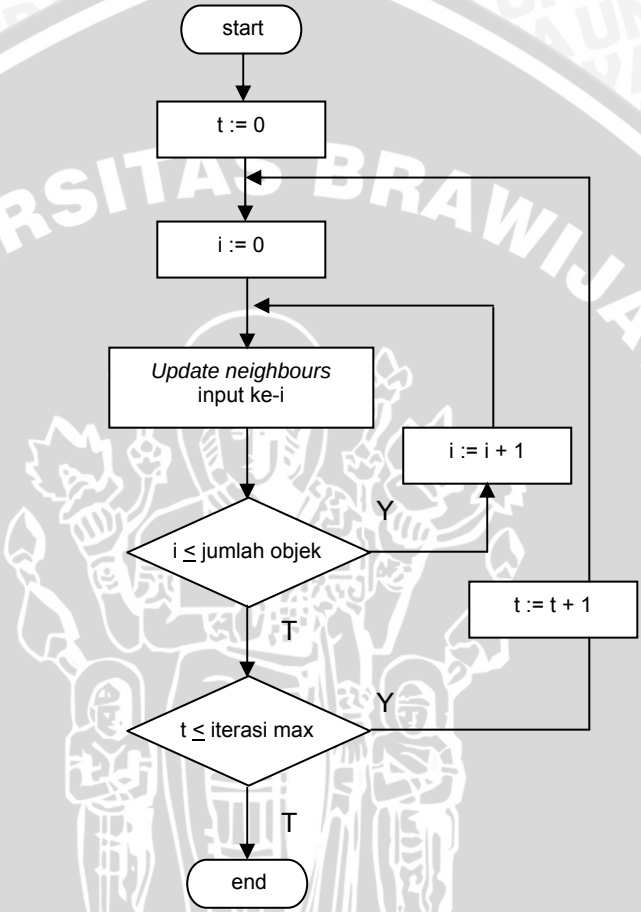
Gambar 3.10 *flowchart* proses pelatihan *map* Kohonen

Proses awal dimulai dengan membuat sebuah *map* Kohonen dengan bobot yang acak. Dari bobot yang acak ini akan dicari nilai-nilai yang paling mewakili dari data input pelatihan yang selanjutnya akan menjadi BMU-nya. Model *map* Kohonen yang digunakan pada penelitian ini adalah 2 dimensi yang berarti masing-masing *node*-nya memiliki koordinat posisi. Proses pencarian yang lebih jelasnya digambarkan pada *flowchart* Gambar 3.11.



Gambar 3.11 *flowchart* pencarian BMU

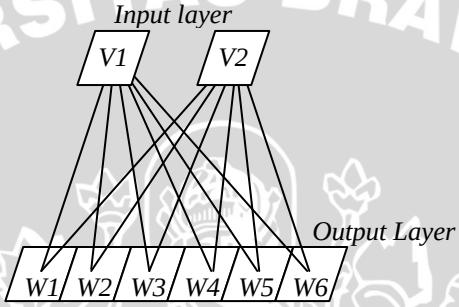
Sedangkan untuk penyesuaian bobot Kohonen dijelaskan pada Gambar 3.12



Gambar 3.12 penyesuaian bobot Kohonen

3.3 Perhitungan Manual

Berikut diberikan contoh perhitungan data pada *map* Kohonen SOM. Perhitungan manual dibuat berdasarkan contoh pada literatur (Muis, 2006) dengan beberapa penyesuaian rumus. *Map* Kohonen SOM yang digunakan adalah yang memiliki 1 dimensi dengan ukuran 6 *node* dan dengan *vector* bobot berdimensi 2 seperti diperlihatkan pada Gambar 3.7



Gambar 3.7 contoh map Kohonen SOM

Diberikan *vector* bobot tiap *node* berturut-turut adalah w_1(0.1 ; 0.2), w_2(0.2 ; 0.4), w_3(0.7 ; 0.3), w_4(0.6 ; 0.5), w_5(0.1 ; 0.8), dan w_6(0.9 ; 0.8). Dengan *learning rate* (α) = 0.4, radius (r) = 4 dan jumlah iterasi maksimum = 10. *Node-node* pada *map* Kohonen SOM akan dikelompokkan berdasarkan 2 buah *vector input* berturut-turut I_1(0.3; 0.4) dan I_2(0.8 ; 0.9).

Diketahui *vector* bobot mula-mula adalah :

w1	0.1	0.2
w2	0.2	0.4
w3	0.7	0.3
w4	0.6	0.5
w5	0.1	0.8
w6	0.9	0.8

Dan input vector adalah :

I_1 (0.3 ; 0.4) I_2 (0.8 ; 0.9)

Pertama-tama akan dicari BMU bagi tiap *input vector* pada *map* Kohonen dengan menggunakan Persamaan 2.4.

Untuk I_1 dicari jaraknya (*Distance*) dengan *node* w_1

$$Dist\ I_1 = \sqrt{(0.3-0.1)^2 + (0.4-0.2)^2}$$
$$= \sqrt{0.2^2 + 0.2^2}$$
$$= 0.282843$$

Dengan cara yang sama dicari jaraknya untuk setiap *node* sehingga didapat berturut-turut :

Dist $w_1 = 0.282843$
Dist $w_2 = 0.1$
Dist $w_3 = 0.412311$
Dist $w_4 = 0.316228$
Dist $w_5 = 0.447214$
Dist $w_6 = 0.781025$

Dari sini dapat dilihat BMU dari I_1 adalah w_2 karena memiliki jarak yang paling kecil.

Demikian juga untuk I_2 dicari jaraknya dengan cara yang sama dan didapatkan :

Dist $w_1 = 0.989949$
Dist $w_2 = 0.781025$
Dist $w_3 = 0.608276$
Dist $w_4 = 0.447214$
Dist $w_5 = 0.707107$
Dist $w_6 = 0.1$

Sehingga didapatkan :

BMU $I_1 = w_2$
BMU $I_2 = w_6$

Menghitung λ, dari data yang diberikan didapatkan:

$$\lambda = \frac{t_{\max}}{\log 10(r_0)}$$
$$= \frac{10}{\log 10(4)}$$
$$= 16.60964 \approxeq 16.61$$

untuk t = 0, didapatkan *r* dan α berturut-turut adalah

$$r = r_0 \cdot \left(10^{\frac{-t}{\lambda}} \right)$$

$$= 4 \cdot \left(10^{\frac{-0}{16.61}} \right)$$

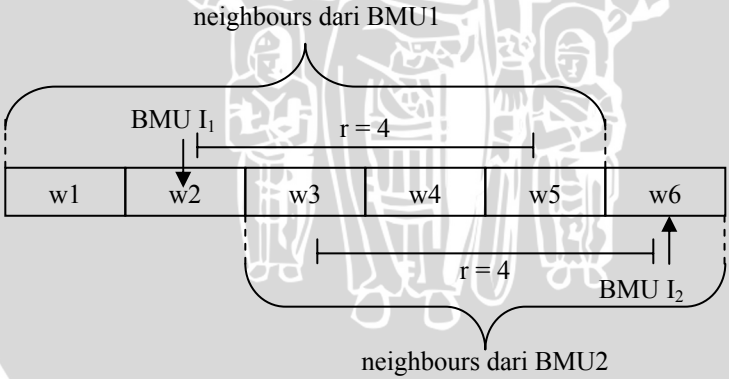
$$= 4$$

$$\alpha = \alpha_0 \cdot \left(10^{\frac{-t}{\lambda}} \right)$$

$$= 0.4 \cdot \left(10^{\frac{-0}{16.61}} \right)$$

$$= 0.4$$

r_0 menunjukkan radius maksimum dari masing-masing BMU, karena yang digunakan adalah *map* Kohonen 1 dimensi maka radiusnya hanyalah maksimum 3 node di kiri dan kanan BMU (1 node lagi adalah BMU itu sendiri). Gambarnya sendiri ditunjukkan oleh Gambar 3.8.



Gambar 3.8 neighbours dari tiap BMU

Sehingga penyesuaian bobot oleh BMU I_1 hanya meliputi w_1, w_2, w_3, w_4 dan w_5. Sedangkan penyesuaian bobot oleh BMU I_2 meliputi w_3, w_4, w_5 dan w_6. Seperti diketahui jauh dekatnya tetangga dengan BMU akan mempengaruhi penyesuaian bobot yang disebut dengan *distance factor*, maka akan dihitung terlebih dahulu *distance factor* relatif untuk masing-masing *node* terhadap BMU-nya.
Untuk w_1 dengan *dist* = 2 (jarak w_1 dengan I_1 berdasarkan Gambar 3.8), *distance factor* terhadap BMU I_1 adalah

$$\delta_{w1} = 10^{\frac{-dist^2}{2r^2}}$$

$$= 10^{\frac{-2^2}{2 \cdot 4^2}}$$

$$= 0.749894$$

Dengan cara yang sama didapat juga *distance factor* w_2, w_3, w_4, w_5 terhadap BMU I_1 adalah

$\delta_{w2} = 0.930572$
$\delta_{w3} = 0.749894$
$\delta_{w4} = 0.523299$
$\delta_{w5} = 0.316228$

Dengan cara yang sama juga didapat *distance factor* w_3, w_4, w_5, w_6 terhadap BMU I_2 adalah

$\delta_{w3} = 0.316228$
$\delta_{w4} = 0.523299$
$\delta_{w5} = 0.749894$
$\delta_{w6} = 0.930572$

Penyesuaian bobot tetangga BMU I_1 adalah meliputi w_1, w_2, w_3, w_4, w_5. Untuk bobot w_1 didapat

$$\text{new } w_1 = w_1 + (\delta_{w1} \,.\, \alpha \,.(I_1 - w_1))$$

untuk *vector* pertama w_1

= 0.1 +(0.749894 * 0.4 * (0.3 – 0.1))

= 0.15999154

untuk *vector* kedua w_1

= 0.2 + (0.749894 * 0.4 * (0.4 – 0.2))

= 0.259992

didapat *vector* bobot baru untuk w_1 adalah:

w_1	0.15999154	0.259992

penyesuaian bobot w_2, w_3, w_4, w_5 dilakukan dengan cara yang sama dan hasilnya adalah

w_1	0.15999154	0.259992
w_2	0.23722288	0.4
w_3	0.58001693	0.329996
w_4	0.53720411	0.479068
w_5	0.12529822	0.749404

Penyesuaian bobot tetangga BMU I_2 kurang lebih sama dengan I_1 hanya saja untuk bobot-bobot yang sudah mengalami penyesuaian tidak lagi menggunakan bobot awal tapi menggunakan bobot penyesuaian dari BMU I_1. Tetangga BMU I_2 meliputi w_3, w_4, w_5 dan w_6. Untuk bobot w3 didapat

new $w_3 = w_3 + (\delta_{w3} . \alpha . (I_2 – w_3))$

untuk *vector* pertama w_3

= 0.58001693 + (0.316228 * 0.4 * (0.8 - 0.58001693))

= 0.607843

untuk *vector* kedua w_3

= 0.329996 +(0.316228 * 0.4 * (0.9 – 0.329996))

= 0.402096

Selanjutnya w_4, w_5 dan w_6 didapatkan dengan cara yang sama
Hasilnya adalah:

w_3	0.607843	0.402096
w_4	0.592212	0.567177
w_5	0.32768	0.794576
w_6	0.862777	0.9

Karena BMU I_2 hanya melakukan penyesuaian pada w_3, w_4, w_5 dan w_6 maka untuk *vector* bobot w_1 dan w_2 digunakan penyesuaian oleh BMU I_1.

Untuk hasil yang lengkap t = 0 adalah

w_1	0.15999154	0.259992
w_2	0.23722288	0.4
w_3	0.607843	0.402096
w_4	0.592212	0.567177
w_5	0.32768	0.794576
w_6	0.862777	0.9

Untuk t = 1 didapat

w_1	0.19333954	0.29334
w_2	0.25710308	0.4
w_3	0.5547555	0.439588
w_4	0.58611853	0.59539
w_5	0.4385726	0.73
w_6	0.84289692	0.9

Untuk t = 2 didapat

w_1	0.212928607	0.312928607
w_2	0.268575756	0.4
w_3	0.507967563	0.432317292
w_4	0.581784501	0.607994637
w_5	0.504951709	0.761221952
w_6	0.831424244	0.9

Untuk t = 3 didapat

w_1	0.224790196	0.324790196
w_2	0.275605074	0.4
w_3	0.479636491	0.427914761
w_4	0.578996514	0.613741297
w_5	0.545145642	0.780127452
w_6	0.824394926	0.9

Untuk t = 4 didapat

w_1	0.23201093	0.33201093
w_2	0.280111199	0.4
w_3	0.462389973	0.425234724
w_4	0.577416069	0.616303029
w_5	0.569613667	0.791636161
w_6	0.819888801	0.9

Untuk t = 5 didapat

w_1	0.236310936	0.336310936
w_2	0.283094098	0.4
w_3	0.452119529	0.42363874
w_4	0.577416069	0.616303029
w_5	0.584184579	0.798489692
w_6	0.816905902	0.9

Untuk t = 6 didapat

w_1	0.2387383	0.3387383
w_2	0.285107469	0.4
w_3	0.446321839	0.422737803
w_4	0.577416069	0.616303029
w_5	0.592409893	0.802358527
w_6	0.814892531	0.9

Untuk t = 7 didapat

w_1	0.239397017	0.339397017
w_2	0.285827566	0.4
w_3	0.444748514	0.422493315
w_4	0.577416069	0.616303029
w_5	0.594642006	0.803408416
w_6	0.814172434	0.9

Untuk t = 8 didapat

w_1	0.239669824	0.339669824
w_2	0.286291365	0.4
w_3	0.444096922	0.42239206
w_4	0.577416069	0.616303029
w_5	0.595566434	0.803843228

w_6	0.813708635	0.9

Untuk t = 9 didapat

w_1	0.239762481	0.339762481
w_2	0.286579851	0.4
w_3	0.443875613	0.42235767
w_4	0.577416069	0.616303029
w_5	0.595880409	0.803990908
w_6	0.813420149	0.9

Untuk t = 10 didapat

w_1	0.239762481	0.339762481
w_2	0.286749604	0.4
w_3	0.443875613	0.42235767
w_4	0.577416069	0.616303029
w_5	0.595880409	0.803990908
w_6	0.813250396	0.9

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Berikut adalah penjelasan tentang proses, *source code* dan tampilan program yang telah dibuat serta analisa terhadap data yang dihasilkan.

4.1 *Application Environment*

Lingkungan (*environment*) pengembangan aplikasi ini terdiri dari perangkat keras (*hardware*) dan perangkat lunak (*software*)

4.1.1 *Hardware Environment*

Hardware yang digunakan dalam pembuatan aplikasi adalah sebagai berikut:

1. Laptop dengan Processor Intel 450 MHz
2. 256 MB RAM
3. 20 GB HDD

4.1.2 Software Environment

Sedangkan untuk *software* yang digunakan adalah:

1. Microsoft Windows XP sebagai sistem operasi
2. Borland Delphi 7.0 sebagai *text editor* dan *compiler*

4.2 Implementasi

Secara keseluruhan sistem ini dibagi menjadi 3 bagian utama yaitu bagian *input training*, pelatihan *map* Kohonen dan pengujian dengan data *input* yang sudah dimodifikasi.

4.2.1 Struktur Data

Struktur data berkaitan dengan pengaturan yang khusus untuk mengorganisir dan menyimpan data. Setiap struktur data didesain untuk mengatur data agar cocok dengan penggunaannya sehingga dapat dikelola dan dipakai secara tepat. Disini struktur data digunakan untuk memodelkan data-data dari bentuk nyata ke bentuk yang dapat dimengerti dan dapat diolah oleh aplikasi.

```
type koordinat = record
    x : integer;
    y : integer;
   end;

type TEdge = ^edge;
  edge = record
    tepi : integer;
    x : integer;
    y : integer;
    next : TEdge;
end;

type TCiriAwal = record
    xTotal, yTotal, n, _x, _y, xFar, yFar : integer;
    r : real;
  end;

type arrayReal = array of real;

type TObyek = ^obyek;
  obyek = record
    n : integer;
    dataEdge : TEdge;
    xMax, xMin, yMax, yMin : integer;
    width, height : integer;
    img : TImage;
    image : TBitmap;
    ciriAwal : TCiriAwal;
    ciriUtama : arrayReal;
    objColor : TColor;
    next : TObyek;
end;

type TAddType = record
    n : integer;
    dataEdge : TEdge;
    xMax, xMin, yMax, yMin : integer;
    width, height : integer;
    img : TImage;
    image : TBitmap;
    ciriAwal : TCiriAwal;
    ciriUtama : arrayReal;
    objColor : TColor;
  end;
```

```
type TKohonen = record
    kord : koordinat;
    dataArray : arrayReal;
    cellColor : TColor;
  end;

type arrayVektor = array of TKohonen;

type kohonen2D = array of array of TKohonen;

type arrInt = array of integer;
```

Gambar 4.1 Struktur Data yang digunakan

Tabel 4.1 keterangan Struktur Data

`koordinat`	Digunakan untuk menyimpan variabel yang berbentuk koordinat, atau yang memiliki dua nilai seperti sebuah posisi dalam peta koordinat.
`TEdge`	Sebuah *linked list* yang digunakan untuk menyimpan seluruh koordinat *pixel-pixel* tepi dari sebuah obyek dan jumlah *pixel* tepinya.
`TCiriAwal`	Untuk menyimpan ciri awal dari obyek yang telah diketahui seluruh *pixel* tepinya. Ciri awal antara lain adalah berisi *global feature*, yaitu berisi luas dan pusat massa obyek.
`TObyek`	Berisi seluruh ciri sebagai representasi data dari sebuah obyek dalam bentuk sebuah *linked list*, memuat tidak hanya ciri awal obyek yaitu *global feature* tapi juga *local feature* yaitu data *signature* obyek.
`TAddType`	Mempunyai struktur yang sama dengan TObyek hanya saja bukan merupakan *linked list*, digunakan untuk menampung nilai kembalian dari variabel bertipe TObyek.

`TKohonen`	Visualisasi data map Kohonen di dalam aplikasi, antara lain berisi kordinat *cell* dan *vector* data yang ditampungnya
`arrayVektor`	Array 1 dimensi dengan tipe data TKohonen untuk menyimpan *signature* dari obyek. Digunakan tipe data TKohonen karena struktur data yang dibutuhkan kurang lebih sama dengan perbedaan tidak terpakainya bagian kordinatnya.
`kohonen2D`	Array 2 dimensi dari tipe data TKohonen, merupakan penggunaan yang sebenarnya dari TKohonen yaitu untuk pembuatan *map* Kohonen. Selanjutnya deklarasi variabel dengan tipe ini adalah merupakan *map* Kohonen.
`arrayReal`	Array 1 dimensi dengan tipe data real, digunakan antara lain untuk menyimpan *signature* obyek.
`ArrInt`	Array 1 dimensi dengan tipe data integer, untuk menyimpan indeks kelas masing-masing obyek yang diujikan.

4.2.2 *Input Training*

Bagian ini merupakan implementasi dari penentuan ciri karakteristik (*feature extraction*) obyek. *Input* yang berupa citra digital diubah ke bentuk representasi data berupa angka-angka. Disini dilakukan operasi citra digital untuk menentukan koordinat-koordinat *pixel* tepi, penentuan luas obyek dan koordinat pusat massa serta pengambilan *signature* obyek. *Signature* obyek adalah bentuk akhir yang akan memodelkan sebuah obyek dan akan berupa nilai-nilai yang kemudian bisa dimasukkan ke dalam pelatihan.

Citra yang akan digunakan diambil dengan operasi *openfile* biasa menggunakan komponen *openfile dialog* untuk kemudian disimpan pada variabel bertipe *TImage*, dari sini pemrosesan citra digital dimulai.

Block proses pencarian *pixel* tepi merupakan bagian segmentasi obyek yang telah dijelaskan melalui *flowchart* pada gambar 3.5 dan untuk potongan *sourcecode* terdapat pada *Sourcecode* 4.1

```
. . .
  pixelAwal := firstPixel(bufImg,0,0, ProgressBar1);

  while ((pixelAwal.x<>0) and (pixelAwal.y<>0)) do
  begin
    Application.ProcessMessages;
    if (Application.Terminated) or (cancelProc) then
      break;

    objCount := objCount + 1;
    lbObjCount.Caption := IntToStr(objCount);
    xMin := pixelAwal.x; yMin := pixelAwal.y;
    xMax := pixelAwal.x; yMax := pixelAwal.y;

  memoProcess.Lines.Clear;
  memoProcess.Lines.Add('cari pixel tepi');
    pixelTepi := nextEdge(pixelAwal.x,pixelAwal.y,
     bufImg.Canvas.Pixels[pixelAwal.x,pixelAwal.y],
      bufImg);
    eraseEdgeList;
  memoProcess.Lines.Clear;
  memoProcess.Lines.Add('simpan pixel tepi');
    addEdgeList(pixelTepi.x,pixelTepi.y);

    while ((pixelTepi.x <> 0)or(pixelTepi.y <> 0)) do
    begin
  memoProcess.Lines.Clear;
  memoProcess.Lines.Add('cari pixel tepi');
      pixelTepi := nextEdge(pixelTepi.x,pixelTepi.y,
       bufImg.Canvas.Pixels[pixelAwal.x,pixelAwal.y],
        bufImg);
      if not((pixelTepi.x=0)and(pixelTepi.y=0)) then
      begin
  memoProcess.Lines.Clear;
  memoProcess.Lines.Add('simpan pixel tepi');
        addEdgeList(pixelTepi.x, pixelTepi.y);
. . .
      end;
    end;
. . .
```

***Sourcecode* 4.1** potongan *sourcecode* deteksi tepi

Deteksi tepi menggunakan sejumlah fungsi dan prosedur yaitu firstPixel, nextEdge dan addEdgeList. *Sourcecode* dari fungsi firstPixel diperlihatkan pada *Sourcecode* 4.2

```
function firstPixel(var image: TImage; x, y: integer;
 progressBar: TProgressBar) : koordinat;
var
  bg, curClr : TColor;
  imWidth, imHeight : integer;
begin

  imWidth := image.Width;  imHeight := image.Height;
  bg := image.Canvas.Pixels[0,0];
  curClr := bg;

  progressBar.Position :=0;
  progressBar.Max := imHeight;

  while ((y<imHeight) and (curClr=bg)) do
  begin
    while ((x<imWidth) and (curClr=bg)) do
    begin
      curClr := image.Canvas.Pixels[x,y];
      if (curClr=bg) then
        x:=x+1
    end;
    if (curClr=bg) then
    begin
      y:=y+1; x:=0;
    end;
    progressBar.Position := y;
  end;
  if ((x<imWidth) and (y<imHeight)) then
  begin
    firstPixel.x := x;
    firstPixel.y := y;
  end
  else
  begin
    firstPixel.x := 0;
    firstPixel.y := 0;
  end;

end;
```

***Sourcecode* 4.2** *sourcecode* fungsi firstPixel

FirstPixel digunakan untuk mencari *pixel* awal dari suatu obyek, nilai kembaliannya adalah sebuah koordinat dan fungsi ini akan mengembalikan koordinat 0,0 bila tidak menemukan *pixel* awal.

Fungsi selanjutnya adalah nextEdge dan addEdgeList, kedua fungsi ini sangat berkaitan karena secara langsung mempengaruhi fungsi yang lainnya. NextEdge mencari tepi-tepi selanjutnya dari suatu obyek setelah diketahui *pixel* awalnya dan menyimpannya dalam bentuk *linked list* dengan prosedur nextEdge. Fungsi nextEdge diperlihatkan pada *Sourcecode* 4.3 dan prosedur addEdgeList pada *Sourcecode* 4.4.

```
function nextEdge(var x, y: integer; col: TColor;
image: TImage) : koordinat;
var
  w1, w2 : TColor;
  a,b : integer;
  cekLL : boolean;
begin

w1 := image.Canvas.Pixels[x+1,y-1];
w2 := image.Canvas.Pixels[x+1,y];
a := x+1; b := y;
cekLL := inEdgeList(a,b);
if ((w1<>col) and (w2=col) and (cekLL=false)) then
begin
  nextEdge.x := x+1;
  nextEdge.y := y;
end
else
begin
  w1 := image.Canvas.Pixels[x+1,y];
  w2 := image.Canvas.Pixels[x+1,y+1];
  a := x+1; b := y+1;
  cekLL := inEdgeList(a,b);
  if ((w1<>col) and (w2=col) and (cekLL=false)) then
  begin
    nextEdge.x := x+1;
    nextEdge.y := y+1;
  end
  else
  begin
    w1 := image.Canvas.Pixels[x+1,y+1];
    w2 := image.Canvas.Pixels[x,y+1];
    a := x; b := y+1;
```

```
cekLL := inEdgeList(a,b);
if ((w1<>col) and (w2=col) and (cekLL=false)) then
begin
  nextEdge.x := x;
  nextEdge.y := y+1;
end
else

begin
  w1 := image.Canvas.Pixels[x,y+1];
  w2 := image.Canvas.Pixels[x-1,y+1];
  a := x-1; b := y+1;
  cekLL := inEdgeList(a,b);
  if ((w1<>col) and (w2=col) and (cekLL=false))
    then
  begin
    nextEdge.x := x-1;
    nextEdge.y := y+1;
  end
  else
  begin
    w1 := image.Canvas.Pixels[x-1,y+1];
    w2 := image.Canvas.Pixels[x-1,y];
    a := x-1; b := y;
    cekLL := inEdgeList(a,b);

    if ((w1<>col) and (w2=col) and (cekLL=false))
      then
    begin
      nextEdge.x := x-1;
      nextEdge.y := y;
    end
    else
    begin
      w1 := image.Canvas.Pixels[x-1,y];
      w2 := image.Canvas.Pixels[x-1,y-1];
      a := x-1; b := y-1;
      cekLL := inEdgeList(a,b);

      if ((w1<>col) and (w2=col) and
        (cekLL=false)) then
      begin
        nextEdge.x := x-1;
        nextEdge.y := y-1;
      end
      else
```

```
            begin
              w1 := image.Canvas.Pixels[x-1,y-1];
              w2 := image.Canvas.Pixels[x,y-1];
              a := x; b := y-1;
              cekLL := inEdgeList(a,b);
              if ((w1<>col) and (w2=col) and
               (cekLL=false)) then
              begin
                nextEdge.x := x;
                nextEdge.y := y-1;
              end

              else
              begin
                w1 := image.Canvas.Pixels[x,y-1];
                w2 := image.Canvas.Pixels[x+1,y-1];
                a := x+1; b := y-1;
                cekLL := inEdgeList(a,b);
                if ((w1<>col) and (w2=col) and
                 (cekLL=false)) then
                begin
                  nextEdge.x := x+1;
                  nextEdge.y := y-1;
                end
                else
                begin
                  nextEdge.x := 0;
                  nextEdge.y := 0;
                end;
              end;
            end;
          end;
        end;
      end;
    end;
  end;
end;
```

***Sourcecode* 4.3** fungsi nextEdge

```
procedure addEdgeList(var x,y:integer);
begin
  new(edgeCur);
  edgeCur.x := x;
  edgeCur.y := y;
  edgeCur.next := nil;
 if edgeHead = nil then
 begin
  edgeCur.tepi := 1;
  edgeHead := edgeCur;
 end
 else
 begin
  edgeCur.tepi := edgeLast.tepi + 1;
  edgeLast.next := edgeCur;
 end;
 edgeLast := edgeCur;
end;
```

***Sourcecode* 4.4** prosedur addEdgeList

Selain nextEdge dan addEdgeList ada satu fungsi lain yang memiliki peran penting yaitu inEdgeList. Fungsi inEdgeList diperlihatkan pada *Sourceode* 4.5.

```
function inEdgeList(var x,y:integer) : boolean;
var
 edgeSearch : TEdge;
begin
 edgeSearch := edgeHead;
 Result := false;

 while (edgeSearch <> nil) do
 begin
  if ((edgeSearch.x = x) and (edgeSearch.y = y)) then
   Result := true;
  edgeSearch := edgeSearch.next;
 end;
end;
```

***Sourcecode* 4.5** fungsi inEdgeList

InEdgeList adalah sebuah fungsi bertipe *boolean* yang digunakan untuk memeriksa apakah sebuah koordinat tepi sudah tersimpan atau belum. Fungsi nextEdge digunakan mencari *pixel* tepi dan tepi baru yang ditemukan kemudian dikonfirmasi oleh fungsi inEdgeList apakah sudah tersimpan atau belum. Bila belum tersimpan digunakan prosedur addEdgeList untuk menyimpan tepi yang baru tersebut pada *linked list*, atau jika ternyata sudah tersimpan fungsi nextEdge akan mengembalikan nilai koordinat 0,0 sebagai indikator tepi obyek seluruhnya sudah tersimpan dan akhirnya hentikan pencarian *pixel* tepi.

Setelah proses segmentasi obyek selesai dilakukan selanjutnya bisa dilakukan proses *feature extraction* (pembentukan ciri karakteristik), untuk itu terlebih dahulu perlu ditentukan berapa jumlah ciri dari obyek yang akan digunakan. Jumlah ciri selanjutnya akan digunakan untuk menentukan sudut rotasi garis terjauh dengan pusat massa sesuai Gambar 3.5. Sudutnya dihitung dengan membagi 360^0 dengan jumlah ciri yang diinginkan. *Sourcecode* 4.5 menunjukkan potongan *sourcecode* pembentukan ciri karaketeristik obyek.

```
. . .
  memoProcess.Lines.Clear;
  memoProcess.Lines.Add('simpan obyek');
    addObyekList2(angle0);

. . .
  SetLength(vektorObyek, objCount);

  for i:= 0 to objCount-1 do
  begin
  memoProcess.Lines.Clear;
  memoProcess.Lines.Add('isikan vektor input : ' +
     IntToStr(i));
    j := i;
    bufCiri := obyekByNumber(j);
    vektorObyek[i].dataArray := bufCiri.ciriUtama;
    vektorObyek[i].cellColor := bufCiri.objColor;
  end;
. . .
```

***Sourcecode* 4.6** potongan *sourcecode feature extraction*

Pembuatan ciri karakteristik obyek dilakukan oleh prosedur addObyekList2, prosedur ini memiliki nilai *input* bernama angle0 yaitu nilai ukuran sudut yang dipilih. *Sourcecode* addObyekList2 dapat dilihat pada *Sourcecode* 4.6.

```
procedure addObyekList2(angle0 : real);
var
  bufHead, bufLast, bufCur : TEdge;
  xMin, xMax, yMin, yMax : integer;
  data : TCiriAwal;
  height, width : integer;
  bg, clr : TColor;
  angleN : real;
  dataCount : integer;
  r, rMax : real;
  dX, dY : integer;
  i : integer;
begin
  bufLast := nil;
  bufHead := nil;
  new(objCur);
  edgeCur := edgeHead;

  xMin := edgeCur.x; xMax := edgeCur.x;
  yMin := edgeCur.y; yMax := edgeCur.y;

  while edgeCur<>nil do
  begin
    if (edgeCur.x <= xMin) then
     xMin := edgeCur.x;
    if (edgeCur.x >= xMax) then
     xMax := edgeCur.x;
    if (edgeCur.y <= yMin) then
     yMin := edgeCur.y;
    if (edgeCur.y >= yMax) then
     yMax := edgeCur.y;
    new(bufCur);
      bufCur.tepi := edgeCur.tepi;
      bufCur.x := edgeCur.x;
      bufCur.y := edgeCur.y;
      bufCur.next := nil;
    if bufHead = nil then
      bufHead := bufCur
    else
      bufLast.next := bufCur;
```

```
    bufLast := bufCur;
    edgeCur := edgeCur.next;
  end;

  objCur.dataEdge := bufHead;
  objCur.xMax := xMax;
  objCur.xMin := xMin;
  objCur.yMax := yMax;
  objCur.yMin := yMin;
  objCur.width := xMax - xMin;
  objCur.height := yMax - yMin;

  objCur.image := TBitmap.create;
  objCur.image.Width := objCur.width + 5;
  objCur.image.Height := objCur.height + 5;
  edgeCur := edgeHead;
  while edgeCur<>nil do
  begin
    objCur.image.Canvas.Pixels[edgeCur.x - xMin+2,
     edgeCur.y - yMin+2] := clBlue;
    edgeCur := edgeCur.next;
  end;

  objCur.image.Canvas.Brush.Color := clBlue;
  objCur.image.Canvas.FloodFill(
   objCur.image.Width div 2,
   objCur.image.Height div 2, clBlue, fsBorder);

  data.xTotal := 0; data.yTotal := 0; data.n := 0;
  bg := objCur.image.Canvas.Pixels[0,0];
  for height:= 0 to objCur.image.Height-1 do
    for width:= 0 to objCur.image.Width-1 do
    begin
      clr:= objCur.image.Canvas.Pixels[width,height];
      if clr<>bg then
      begin
        data.xTotal := data.xTotal + width;
        data.yTotal := data.yTotal + height;
        data.n := data.n + 1;
      end
    end;

  data._x := data.xTotal div data.n;
  data._y := data.yTotal div data.n;
  objCur.ciriAwal.xTotal := data.xTotal;
  objCur.ciriAwal.yTotal := data.yTotal;
```

```
objCur.ciriAwal.n := data.n;
objCur.ciriAwal._x := data._x;
objCur.ciriAwal._y := data._y;

data.xFar := 0; data.yFar := 0;
data.r := 0;
bufCur := objCur.dataEdge;
while bufCur<>nil do
begin
  r:=sqrt(sqr(data._x - (bufCur.x - xMin + 2))
   + sqr(data._y - (bufCur.y - yMin + 2)));
  if (r >= data.r) then
  begin
    data.xfar := bufCur.x - xMin + 2;
    data.yfar := bufCur.y - yMin + 2;
    data.r := r;
  end;
  bufCur := bufCur.next;
end;

objCur.ciriAwal.xFar := data.xFar;
objCur.ciriAwal.yFar := data.yFar;



rMax:=distant(objCur.ciriAwal._x,
 objCur.ciriAwal._y,objCur.ciriAwal.xFar,
 objCur.ciriAwal.yFar, objCur.image);
objCur.ciriAwal.r := rMax;

dataCount := round(360/angle0);
SetLength(objCur.ciriUtama, dataCount);
objCur.ciriUtama[0] := rMax/rMax;
for i := 1 to dataCount-1 do
begin
  angleN:=angle0*i;
  dx := round(rotateX(data._x, data._y, data.xFar,
   data.yFar, angleN));
  dy := round(rotateY(data._x, data._y, data.xFar,
   data.yFar, angleN));
  r:= distant(data._x, data._y,dx,dy,objCur.image);
  objCur.ciriUtama[i] := r/rMax;
end;
objCur.objColor := convert2RGB(objCur.ciriUtama);

objCur.next := nil;
```

```
    if objHead=nil then
    begin
      objCur.n := 0;
      objHead := objCur;
    end
    else
    begin
      objCur.n := objLast.n + 1;
      objLast.next := objCur;
    end;

    objLast := objCur;
  end;
```

Sourcecode 4.7 *sourcecode* addObyekList2

Prosedur addObyekList2 akan menyimpan data *pixel* tepi yang didapat dari proses segmentasi obyek sebelumnya. Disini obyek akan direkonstruksi kembali agar dapat dicari *global feature* yang terdiri dari luas dan pusat massa obyek, kemudian dicari *pixel* yang memiliki jarak terjauh dengan pusat massa sebagai garis acuan. *Pixel* terjauh selanjutnya akan dirotasikan sejumlah titik yang telah dipilih dan panjangnya akan dibagi dengan garis acuan dengan demikian didapatkanlah ciri karakteristik obyek. Langkah terakhir adalah menyimpan semua data-data tersebut mulai dari koordinat tepi sampai dengan ciri karakteristik ke dalam sebuah *linked list* bertipe TObyek yang digunakan untuk menyimpan data suatu obyek.

AddObyekList2 sendiri menggunakan beberapa fungsi-fungsi kecil yaitu rotateX, rotateY, distant dan convert2RGB. Fungsi rotateX dan rotateY sendiri digunakan untuk merotasikan koordinat *pixel* terjauh dengan sumbu rotasi adalah pusat massa obyek, nilai parameter yang digunakan oleh kedua fungsi ini adalah koordinat titik terjauh yang akan dirotasikan, koordinat titik pusat rotasi dan yang terakhir adalah besarnya sudut titik tersebut akan dirotasikan. Kedua fungsi ini diperlihatkan pada *Sourcecode* 4.7 dan 4.8. Fungsi distant digunakan untuk menghitung jarak pusat massa ke tepi obyek setelah *pixel* terjauh dirotasikan, *sourcecode*-nya diperlihatkan pada *Sourcecode* 4.9. Dan yang terakhir adalah mencari kode warna dari ciri karakteristik obyek yang telah didapat sebagai representasi ciri

tersebut pada *map* Kohonen dengan *sourcecode* yang dapat dilihat pada *Sourcecode* 4.10..

```
function rotateX(var _x,_y,xfar,yfar:integer;
 angle:real):real;
var
  cosA, sinA, rotX : real;
begin
  angle := angle/180*PI;
  cosA := cos(angle);
  sinA := sin(angle);
  rotX := 0;
  if (_x <= xfar) and (_y > yfar) then
  begin
    rotX:=(xfar - _x)*cosA - (-(_y - yfar))*sinA;
    rotX:=rotX + _x;
  end
  else if (_x <= xfar) and (_y <= yfar) then
  begin
    rotX:=(xfar - _x)*cosA - (yfar - _y)*sinA;
    rotX:=rotX + _x;
  end
  else if (_x > xfar) and (_y <= yfar) then
  begin
    rotX:=(-(_x - xfar)) * cosA - (yfar - _y) * sinA;
    rotX:=rotX + _x;
  end
  else if (_x > xfar) and (_y > yfar) then
  begin
    rotX:=(-(_x - xfar))*cosA - (-(_y-yfar))*sinA;
    rotX:=rotX + _x;
  end;
  Result := rotX;
end;
```

***Sourcecode* 4.8** *sourcecode* fungsi rotateX

```
function rotateY(var _x,_y,xfar,yfar:integer;
 angle:real):real;
var
  cosA, sinA, rotY : real;
begin
  angle := angle/180*PI;
  cosA := cos(angle);  sinA := sin(angle);
  rotY := 0;
  if (_x <= xfar) and (_y > yfar) then
  begin
    rotY := (xfar - _x)*sinA + (-(_y - yfar))*cosA;
    rotY := rotY + _y;
  end
  else if (_x <= xfar) and (_y <= yfar) then
  begin
    rotY := (xfar - _x)*sinA + (yfar - _y)*cosA;
    rotY := rotY + _y;
  end
  else if (_x > xfar) and (_y <= yfar) then
  begin
    rotY := (-(_x - xfar))*sinA + (yfar - _y)*cosA;
    rotY := rotY + _y;
  end
  else if (_x > xfar) and (_y > yfar) then
  begin
    rotY:= (-(_x - xfar))*sinA + (-(_y - yfar))*cosA;
    rotY := rotY + _y;
  end;
  Result := rotY;
end;
```

***Sourcecode* 4.9** *sourcecode* fungsi rotateY

```
function distant(var _x,_y,dx,dy:integer;
 image:TBitmap) : real;
var
  m,x,y,c,n_range : real;
begin
  n_range := 0;  x:=_x; y:=_y;
  image.Canvas.Pen.Color := clLime;
  if (_x - dx) = 0 then
  begin
    if(_y<dy) then
      while (image.Canvas.Pixels[round(x),
       round(y)]<>clWhite) do
```

```
        begin
          y := y + 1;
          n_range := y - _y;
        end
      else
        while (image.Canvas.Pixels[round(x),
         round(y)]<>clWhite) do
        begin
          y := y - 1;
          n_range := _y - y;
        end;
    image.Canvas.MoveTo(round(_x),round(_y));
    image.Canvas.LineTo(round(x),round(y));
      Result := n_range;
      Exit;
    end;

    m := (_y-dy)/(_x-dx); //gradien
    c := _y - (m*_x); // pers.-> y = mx + c
    if (_x<dx) then
    begin
      while (image.Canvas.Pixels[round(x),
       round(y)]<>clWhite) do
      begin
        y := m * x + c;
        x := x + 0.005;
        n_range:=sqrt(sqr(_x-x) + sqr(_y-y));
      end;
    end
    else if (_x>dx) then
    begin
      while (image.Canvas.Pixels[round(x),
       round(y)]<>clWhite) do
      begin
        y := m * x + c;
        x := x - 0.005;
        n_range := sqrt(sqr(_x-x) + sqr(_y-y));
      end;
    end;
    image.Canvas.MoveTo(round(_x),round(_y));
    image.Canvas.LineTo(round(x),round(y));

    Result := n_range;
  end;
```

***Sourcecode* 4.10** *sourcecode* fungsi distant

```
function convert2RGB(var vektorObyek: array of real):
  TColor;
var
  i : integer;
  rTotal, gTotal, bTotal : integer;
  r,g,b : real;
  rConv, gConv, bConv : byte;
begin
  r:=0; g:=0; b:=0;
  rTotal:=0; gTotal:=0; bTotal:=0;
  for i := 0 to high(vektorObyek) do
  begin
    if (i mod 3 = 0) then
    begin
      r := r + vektorObyek[i];
      rTotal := rTotal+1;
    end
    else if (i mod 3 = 1) then
    begin
      g := g + vektorObyek[i];
      gTotal := gTotal + 1;
    end
    else if (i mod 3 = 2) then
    begin
      b := b + vektorObyek[i];
      bTotal := bTotal + 1;
    end;
  end;

  rConv := round(275 - (r/rTotal)*255);
  gConv := round(275 - (g/gTotal)*255);
  bConv := round(275 - (b/bTotal)*255);

  result :=RGB(rConv,gConv,bConv);
end;
```

***Sourcecode* 4.11** *sourcecode* fungsi convert2RGB

Demikian didapatkan *input* untuk *training* yang akan disebut sebagai *input* vektor

4.2.3 *Training Map* Kohonen

Setelah didapatkan ciri karakteristik obyek maka pelatihan menggunakan Kohonen SOM dapat dilakukan. Pertama akan dibuat

vektor bobot random untuk *map* Kohonen dengan prosedur randomWeight.

```
procedure TFormMain.randomWeight(n : integer);
var
  x,y,i : integer;
begin
  SetLength(weightKohonen,n);
  for x:=Low(weightKohonen) to high(weightKohonen) do
  begin
    SetLength(weightKohonen[x],n);
    for y: = Low(weightKohonen[x]) to
     High(weightKohonen[x]) do
    begin
      weightKohonen[x,y].kord.x := x;
      weightKohonen[x,y].kord.y := y;
      SetLength(weightKohonen[x,y].dataArray,
       (360 div StrToInt(radioGrSudut.Items.Strings
       [radioGrSudut.itemIndex])));
      Randomize;
      for i := Low(weightKohonen[x,y].dataArray)
       to High(weightKohonen[x,y].dataArray) do
        if i=0 then
          weightKohonen[x,y].dataArray[i] := 1
        else
          weightKohonen[x,y].dataArray[i] :=

           Random(1000) / 1000;
      weightKohonen[x,y].cellColor :=
       convert2RGB(weightKohonen[x,y].dataArray);
    end;
  end;
end;
```

***Sourcecode* 4.12** *sourcecode* prosedur randomWeight

Untuk menyimpan vektor bobot *map* Kohonen digunakan sebuah variabel bernama weightKohonen yaitu array 2 dimensi dengan tipe TKohonen.

Setelah bobot awal Kohonen tercipta selanjutnya akan dibuat visualisasi dari map Kohonen dengan prosedur createKohonenMap.

```
procedure TFormMain.createKohonenMap(i:integer);
var
  x, y : integer;
begin
  panelKohonen.Free;

  panelKohonen := TPanel.Create(Self);
  with panelKohonen do
  begin
    Name := 'panelKohonen';
    caption := '';
    Parent := pgMain.Pages[1];
    top := 1;
    left := 1;
    Width := i*8 + 17;
    Height := i*8 + 17;
    Visible := true;
  end;

  mapKohonen := TImage.Create(self);
  with mapKohonen do
  begin
    Name := 'imgMapKohonen';
    Parent := panelKohonen;
    top := 8;
    left := 8;
    Width := i * 8;
    Height := i * 8;

   Canvas.Pen.Color:=clBlack;
    for i := 0 to i do
    begin
      Canvas.MoveTo((i+1)*8,0);
      Canvas.LineTo((i+1)*8,Height);
      Canvas.MoveTo(0,(i+1)*8);
      Canvas.LineTo(Width,(i+1)*8);
    end;
    OnMouseMove := cellKohonenMouseMove;
    OnMouseUp := cellKohonenMouseUp;
    Visible := true;
    //Hint := 'hint map kohonen';
  end;

  for x:=low(weightKohonen) to high(weightKohonen) do
    for y := low(weightKohonen[x]) to
     high(weightKohonen[x]) do
```

```
      begin
        mapKohonen.Canvas.Brush.Color :=
          weightKohonen[x,y].cellColor;
        mapKohonen.Canvas.FloodFill(
        ((x*8)+4),((y*8)+4), clBlack,fsBorder);
      end;
  end;
```

***Sourcecode* 4.13** *sourcecode* prosedur createKohonenMap

Bagian utama dari prosedur ini adalah membuat sebuah *map* Kohonen dengan menggunakan komponen TImage, nama variabel TImagenya sendiri bernama mapKohonen. MapKohonen ini akan terbagi menjadi persegi-persegi kecil sesuai dengan jumlah kuadrat *input* yang diberikan. Persegi ini yang selanjutnya disebut dengan *cell* akan mewakili sebuah nilai vektor bobot tertentu dan mempunyai warna sesuai dengan vektor bobot tersebut.

Dari nilai-nilai random tersebut akan dicari masing-masing satu vektor bobot yang memiliki nilai paling dekat dengan *input-input* vektor, kedekatan ini diukur jarak Euclideannya sesuai Persamaan 2.4 dan bobot tersebut disebut sebagai *Best Matching Unit* (BMU). Berikut adalah potongan-potongan *sourcecode* pencarian BMU, penyesuaian BMU dan tetangganya serta perubahan warna pada *map* Kohonen sebagai visualisasi penyesuaian bobot.

```
  . . .
     objCount := length(vektorObyek);
      for i := 0 to objCount-1 do
      begin
        vektorObyek[i].kord :=
         findBMU(vektorObyek[i].dataArray,
         weightKohonen);
      end;
  . . .
    for t:=0 to loopMax do
    begin
      for n:=0 to length(vektorObyek)-1 do
      begin
        weightKohonen := updateNeighbours(loopMax,
         radiusMax, t, alpha0, vektorObyek[n].kord,
         max, weightKohonen, vektorObyek[n].dataArray);
      end;
  . . .
```

```
  end;

      for x:= 0 to length(weightKohonen)-1 do
      begin
        for y := 0 to length(weightKohonen[x])-1 do
        begin
          clr :=
          convert2RGB(weightKohonen[x,y].dataArray);
          mapKohonen.Canvas.Brush.Color := clr;
          mapKohonen.Canvas.FloodFill(x*8+4,y*8+4,
            clBlack,fsBorder);
        end;
      end;
      mapKohonen.Repaint;
  . . .
```

***Sourcecode* 4.14** *sourcecode* proses Kohonen SOM

Proses ini menyatukan beberapa proses lain yaitu mencari BMU dari *map* Kohonen. *Sourcecode* pencarian BMU bernama findBMU dan diperlihatkan pada *Sourcecode* 4.14. Fungsi ini mengembalikan nilai berupa koordinat dalam *map* Kohonen dan *input* yang diberikan adalah data *input* vektor yang akan dicari BMU-nya.

```
function findBMU(var dataObyek : array of real;
 weightKohonen : kohonen2D) : koordinat;
var
  width, height : integer;
  i : integer;
  dataKohonen : TKohonen;
  kord : koordinat;
  euclidean, buf : real;
begin
  euclidean := 0;
  for height := low(weightKohonen) to
   high(weightKohonen) do
  begin
    for width := low(weightKohonen[height]) to
   high(weightKohonen[height]) do
    begin
      buf := 0;
      dataKohonen := weightKohonen[width,height];
      for i := 0 to length(dataObyek)-1 do
```

```
      begin
        buf := buf + sqr(dataObyek[i] –
        dataKohonen.dataArray[i]);
      end;
      buf := sqrt(buf);
      if ((height=0) and (width=0)) then
        euclidean := buf;
      if (buf<=euclidean) then
      begin
        euclidean := buf;
        kord.x := dataKohonen.kord.x;
        kord.y := dataKohonen.kord.y;
      end;
    end;
  end;

  Result := kord;
end;
```

***Sourcecode* 4.15** *sourcecode* fungsi findBMU

Sourcecode 4.14 diberikan *sourcecode* findBMU dan *Sourcecode* 4.15 berisi *sourcecode* updateNeighbours. UpdateNeighbours berisi proses penyesuaian bobot BMU dan tetangga-tetangganya.

```
function updateNeighbours(loopMax, radiusMax,
  t : integer; alpha0 : real;
 BMU, mapMax : koordinat; weightKohonen : kohonen2D;
 inputVektor : arrayReal) : kohonen2D;
var
  lambda, radius, alpha : real;
  min, max : koordinat;
  i, j : integer;
  dist, distFactor : real;
begin
  lambda := findLambda(loopMax, radiusMax);
  radius := updateRadiusNeighbours(radiusMax, t,
   lambda);
  alpha := updateAlpha(loopMax, t, alpha0);

  if BMU.x <= Round(radius) then
    min.x := 0
  else min.x := BMU.x - Round(radius);
  if BMU.y <= Round(radius) then
```

```
    min.y := 0
  else min.y := BMU.y - Round(radius);
  if (BMU.x + Round(radius)) >= mapMax.x then
    max.x := mapMax.x
  else max.x := BMU.x + Round(radius);
  if (BMU.y + Round(radius)) >= mapMax.y then
    max.y := mapMax.y
  else max.y := BMU.y + Round(radius);

  for i := min.y to max.y do
    for j := min.x to max.x do
    begin
      dist := sqrt(sqr(j-BMU.x) + sqr(i-BMU.y));
      if dist <= radius then
      begin
        distFactor := distanceFactor(dist,radius);
        weightKohonen[j,i].dataArray :=
         updateWeight(weightKohonen[j,i].dataArray,
          inputVektor, distFactor, alpha);
        weightKohonen[j,i].cellColor :=
         convert2RGB(weightKohonen[j,i].dataArray);
      end;
    end;
  Result := weightKohonen;
end;
```

***Sourcecode* 4.16** *sourcecode* fungsi updateNeighbours

Fungsi updateNeighbours berisi fungsi-fungsi kecil lainnya yang digunakan untuk menghitung nilai-nilai lain yang dibutuhkan untuk proses penyesuaian bobot. Diantaranya adalah menghitung λ sesuai Persamaan 2.5 dengan nama fungsi findLambda pada *Sourcecode* 4.16

```
function findLambda(var loopMax,
  radiusMax : integer) : real;
var
  l : real;
begin
  l := loopMax / Log10(radiusMax);
  Result := l;
end;
```

***Sourcecode* 4.17** *sourcecode* fungsi findLambda

Fungsi updateRadiusNeighbours yang digunakan menghitung radius tetangga setelah iterasi ke t sesuai dengan Persamaan 2.6. Fungsi ini adalah fungsi yang berubah mengikuti waktu sehingga membutuhkan parameter masukan t.

```
function updateRadiusNeighbours(radiusMax,
  t : integer; lambda : real) : real;
var
  r : real;
begin
  r := radiusMax * Power(10,(-t / lambda));
  result := r;
end;
```

***Sourcecode* 4.18** *sourcecode* fungsi updateRadiusNeighbours

Fungsi updateAlpha yaitu menghitung *learning rate* setelah iterasi ke t sesuai Persamaan 2.7. Serupa dengan updateRadiusNeighbours, fungsi ini juga membutuhkan parameter waktu t.

```
function updateAlpha(loopMax, t : integer;
  alpha0 : real) : real;
var
  a : real;
begin
  a := alpha0 * Power(10,(-t/loopMax));
  Result := a;
end;
```

***Sourcecode* 4.19** *sourcecode* fungsi updateAlpha

Dan nilai terakhir yang dibutuhkan yaitu faktor jarak (*distance factor*) yang dihitung sesuai Persamaan 2.8

```
function distanceFactor(dist, radius : real) : real;
var
  d, dist_2, radius_2 : real;
begin
  dist_2 := dist*dist;
  radius_2 := radius*radius;
  d := Power(10,(-(dist_2)/(2*radius_2)));
  Result := d;
end;
```

***Sourcecode* 4.20** *sourcecode* fungsi distanceFactor

Setelah semua nilai-nilai tersebut terkumpul penyesuaian bobot untuk sebuah *cell* dapat dilakukan dengan fungsi updateWeight. Fungsi updateWeight merupakan implementasi dari Persamaan 2.9.

```
function updateWeight(neighbours,
  inputVektor : arrayReal;
 distFactor, alpha : real) : arrayReal;
var
  n, arrayLength : integer;
  newWeight : arrayReal;
begin
  arrayLength := length(neighbours);
  SetLength(newWeight,arrayLength);
  for n := low(neighbours) to high(neighbours) do
  begin
    newWeight[n] := neighbours[n]
     + (distFactor * alpha
     * (inputVektor[n] - neighbours[n]));
  end;
  Result := newWeight;
end;
```

***Sourcecode* 4.21** *sourcecode* fungsi updateWeight

4.2.4 Pengujian Kohonen SOM

Pengujian dilakukan pada *map* Kohonen yang sudah dilatih dengan menggunakan obyek-obyek pengujian. Obyek pengujian yang digunakan adalah obyek-obyek pada *input training* yang sudah mengalami perubahan, yaitu rotasi, diperbesar ataupun diperkecil.

Proses awalnya merupakan tipikal dari proses *input training* yaitu menentukan ciri karakteristik dari obyek-obyek yang diujikan. Setelah didapatkan ciri obyeknya, pada *map* Kohonen yang telah dilatih akan dicari satu vektor bobot yang paling mewakili dari sekian banyak vektor bobot. Dengan demikian dapat terlihat hubungan antara BMU dari *input* vektor dengan vektor bobot yang menjadi representasi dari obyek-obyek yang diujikan.

Untuk mengetahui di dalam kelompok mana suatu obyek yang diujikan berada, vektornya akan dibandingkan dengan vektor pusat kelompok yang dalam hal ini adalah BMU. Proses pencariannya dilakukan pada potongan *sourcecode* pada Gambar 4.21.

```
. . .
  for i := 0 to length(trainResult)-1 do
  begin
    min := vectorDistance(vektorObyek[0].dataArray,
     vektorTest[i].dataArray);
    trainResult[i] := 0;
    for j := 0 to length(vektorobyek)-1 do
    begin
      buf := vectorDistance(vektorObyek[j].dataArray,
       vektorTest[i].dataArray);
      if (buf < min) then
      begin
        min := buf;
        trainResult[i] := j;
      end;
    end;
  end;
. . .
```

***Sourcecode* 4.22** *sourcecode* pencarian jarak antara dua vektor

Pada proses ini dipanggil sebuah fungsi yang bernama vectorDistance, fungsi inilah yang digunakan untuk menghitung

jarak antara dua vektor. *Sourcecode* 4.22 adalah isi dari fungsi vectorDistance.

```
function vectorDistance(vectObj,
  vectTest : arrayReal) : real;
var
  buf : real;
  i : integer;
begin
  buf := 0;
  for i := low(vectObj) to high(vectObj) do
  begin
    buf := buf + sqr(vectObj[i] - vectTest[i]);
  end;

  Result := sqrt(buf);
end;
```

***Sourcecode* 4.23** *sourcecode* fungsi vectorDistance

Pencarian vektor bobot pada *map* Kohonen ini diperlihatkan oleh potongan sourcecode pada *Sourcecode* 4.23

```
. . .
  for i := 0 to testCount-1 do
  begin
    vektorTest[i].kord :=
     findBMU(vektorTest[i].dataArray, weightKohonen);
  end;
. . .
```

***Sourcecode* 4.24** *sourcecode* pencarian vektor bobot untuk obyek test

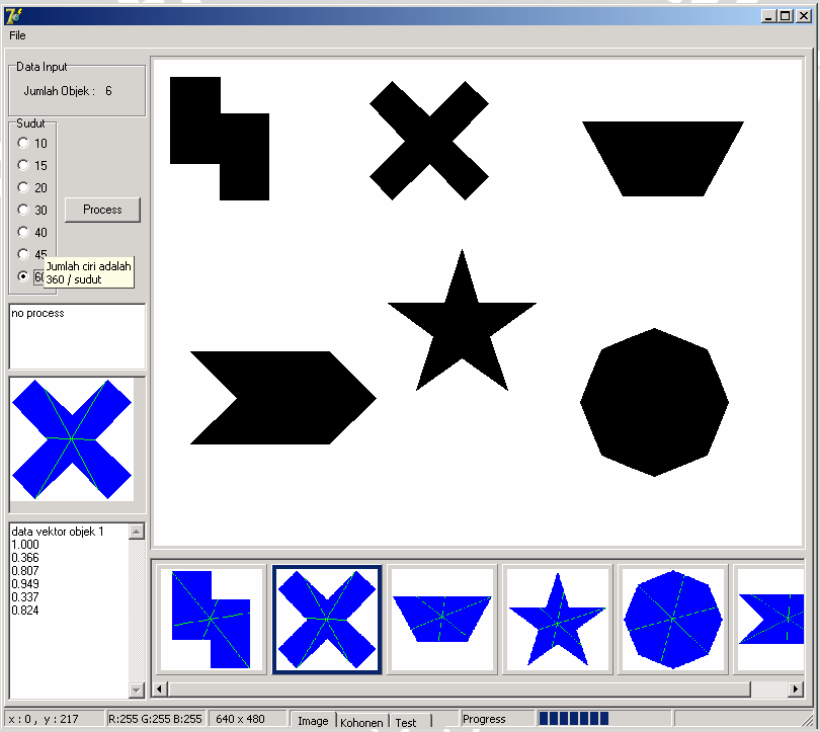
Karena proses pencariannya sama dengan pencarian BMU untuk *input* vektor, fungsi yang digunakan juga sama yaitu fungsi findBMU. Hasil pengujian akan memperlihatkan bahwa BMU dari kelompok akan berada pada kelompok yang sama dengan vektor bobot obyek yang diujikan ataukah tidak.

4.3 Penerapan Aplikasi

Aplikasi ini terdiri dari 3 bagian sebagaimana terdapatnya 3 proses utama *input training*, pelatihan dengan Kohonen SOM, dan pengujian *map* Kohonen.

4.3.1 *Page Input Training*

Halaman pertama aplikasi ini digunakan untuk mengambil *input training* yang akan digunakan pada pelatihan.

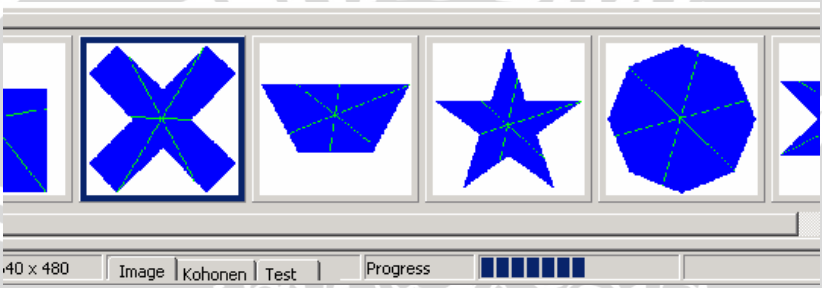


Gambar 4.2 Proses *input training*

Citra yang akan digunakan sebagai *input training* diambil dengan operasi *open file* melalui menu dan Citra yang dipilih akan ditampilkan menggunakan komponen *TImage*. Jumlah ciri obyek yang diinginkan dipilih dengan menggunakan komponen *radio*

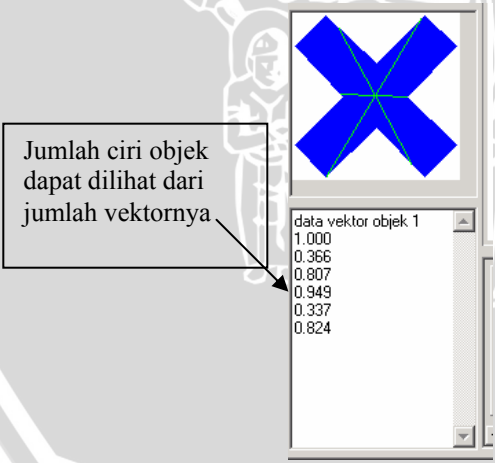
group, jumlah ciri yang akan didapat adalah 360^0 dibagi besarnya sudut yang dipilih pada *radio group*. Selanjutnya proses pengambilan ciri dapat dilakukan dengan menekan *button Process*.

Pengambilan *input training* ini menggunakan terdiri dari proses segmentasi dan pengambilan ciri karakteristik obyek. Dari citra yang utuh setelah disegmentasi akan menghasilkan obyek-obyek tunggal. Obyek-obyek tunggal ini diperlihatkan pada sederetan komponen-komponen *image* untuk menampilkannya.



Gambar 4.3 obyek-obyek yang sudah disegmentasi

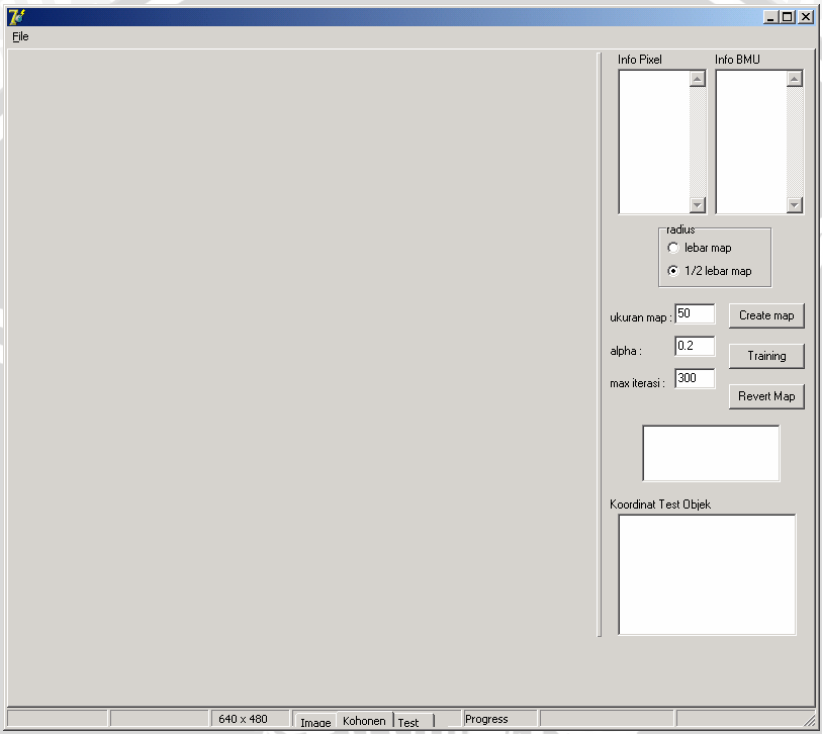
Dan untuk ciri karakteristik dapat dilihat dengan melakukan *click* pada salah satu obyek tersebut sehingga didapat tampilan berikut.



Gambar 4.4 data ciri obyek

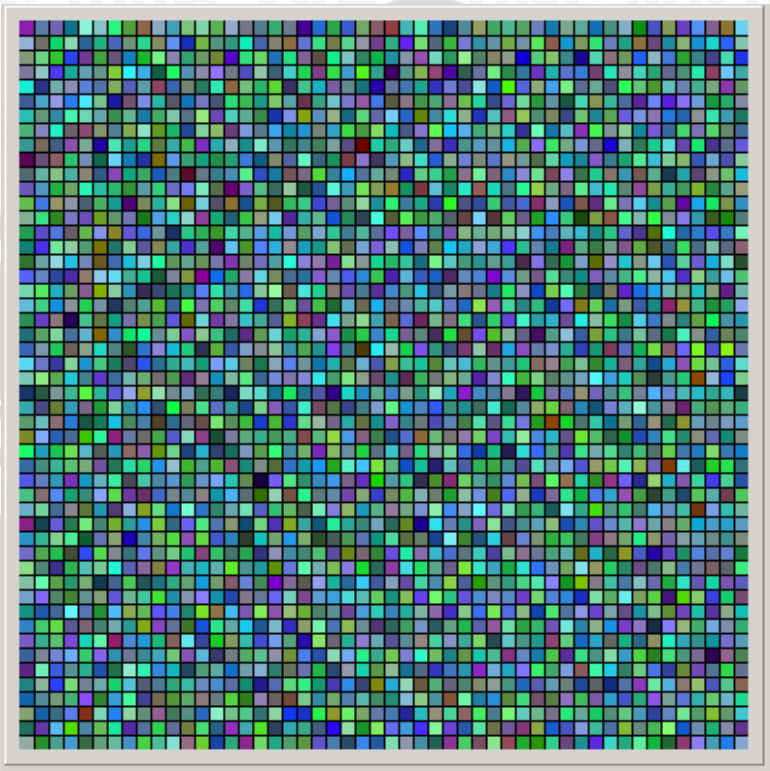
4.3.2 *Page Map* Kohonen

Setelah ciri karakteristik obyek didapatkan maka proses selanjutnya adalah melakukan pelatihan dengan Kohonen SOM. Berikut adalah tampilan awal halaman Kohonen.



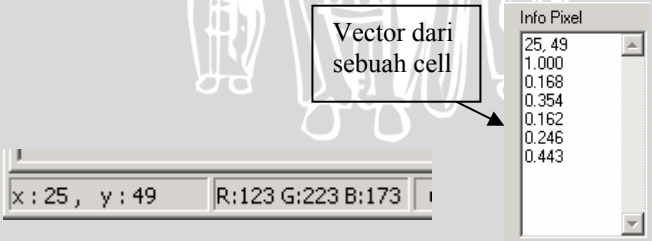
Gambar 4.5 *page* Kohonen

Untuk membuat *map* Kohonen dilakukan dengan menekan *button Create map* setelah terlebih dahulu memilih ukuran *map* yang diinginkan melalui komponen *edit* di sebelahnya. Pertama kali *map* Kohonen yang didapatkan akan memiliki vektor bobot yang acak, ini diperlihatkan dari penyebaran warna yang tidak sama pada *map* tersebut. Koordinat *cell* pada *map* itu dapat dilihat dengan melewatkan *pointer* pada *cell* yang ingin diketahui, dan bila ingin diketahui nilai vektor bobotnya dapat dilakukan dengan *click* pada *cell* tersebut.



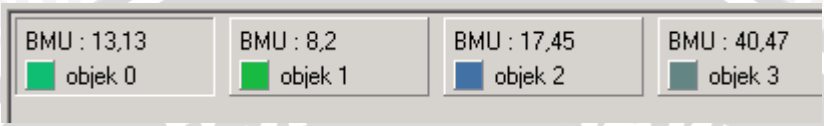
Gambar 4.6 *map* Kohonen dengan bobot acak

Koordinat *cell* serta kode warnanya dapat dilihat pada *StatusBar* dan vektor bobotnya dapat dilihat pada *Memo*.



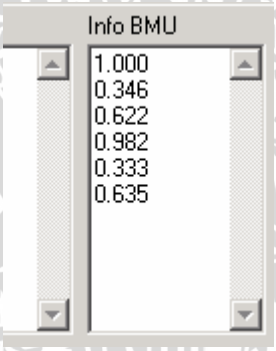
Gambar 4.7 keterangan *cell*

Pada saat *map* Kohonen dengan bobot yang masih acak selesai dibuat, seketika itu juga dicari BMU dari *input-input training*. Koordinat BMU ditampilkan pada *panel* seperti diperlihatkan pada Gambar 4.8.



Gambar 4.8 panel keterangan BMU

Input training memiliki nilai vektor sendiri yang kurang lebih agak berbeda dari BMU-nya, untuk melihat nilai vektor dari *input training* dapat dilakukan dengan *click* pada panelnya yang hasilnya ditampilkan pada sebuah komponen *Memo*.



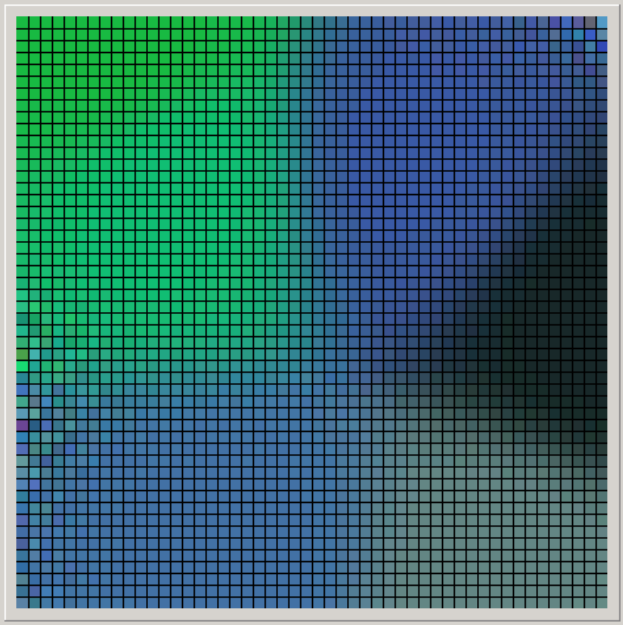
Gambar 4.9 data vektor input

Memo untuk input vektor berada disamping *Memo* info *cell* agar memudahkan membandingkan vektor bobot awal BMU dengan *input* vektornya.

Setelah persiapan-persiapan awal selesai dilakukan, pelatihan menggunakan Kohonen SOM dapat dilakukan. Pada pelatihan terdapat beberapa variabel yang dapat dipilih antara lain radius awal tetangga, *learning rate* dan jumlah iterasi maximum.

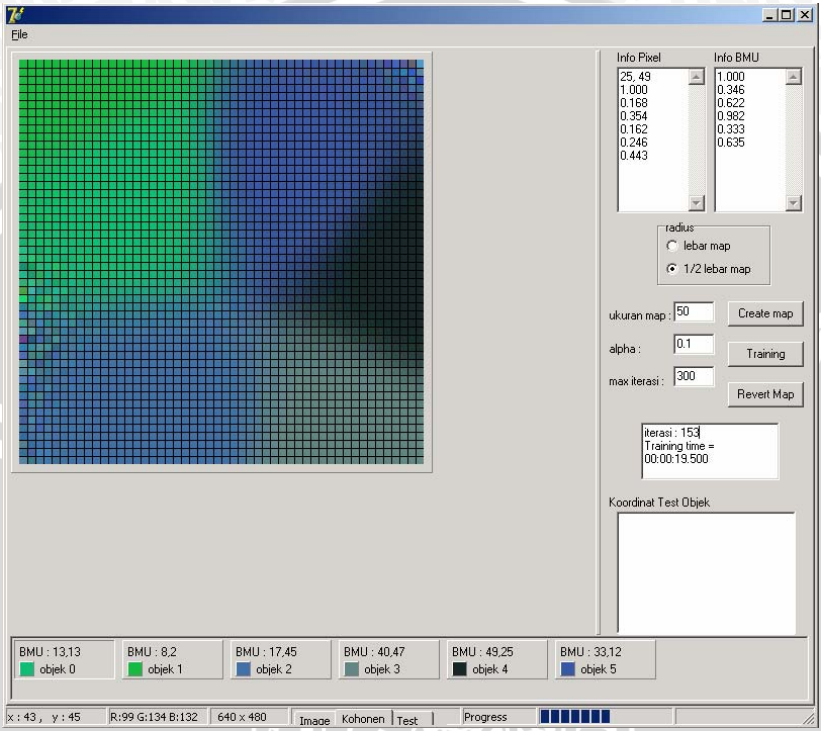
Gambar 4.10 variabel untuk proses pelatihan

Prosesnya sendiri dilakukan dengan menekan *button training* dan dengan menggunakan nilai-nilai variabel sesuai dengan Gambar 4.10 didapatkan hasil pada Gambar 4.11



Gambar 4.11 *map* Kohonen hasil pelatihan

Secara keseluruhan *page* Kohonen akan memiliki tampilan akhir seperti berikut.

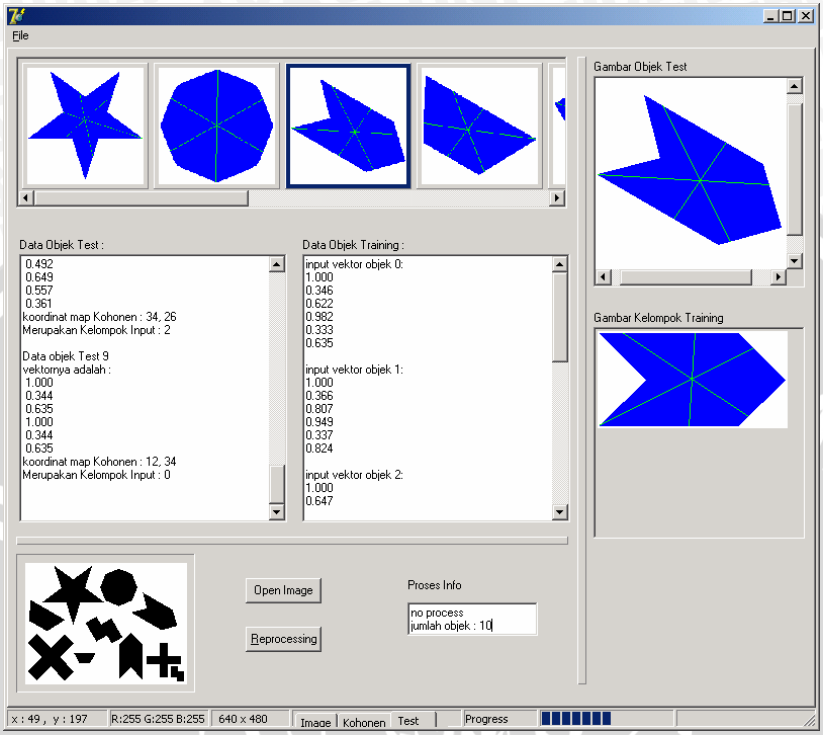


Gambar 4.12 *page* Kohonen setelah pelatihan

4.3.3 Pengujian *Map* Kohonen

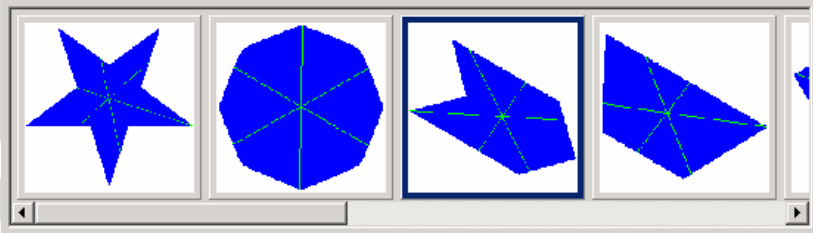
Proses awal pada *page* terakhir ini kurang lebih sama dengan proses pada *page* pertama, yaitu gambar citra yang utuh disegmentasi berdasarkan obyek-obyeknya sampai didapatkan ciri karakteristik. Ciri karakteristik ini akan dicarikan representasinya pada map Kohonen dan dicocokkan dengan vektor *input* yang paling mendekati dan kemudian ditampilkan gambarnya.

Tampilan *page* pengujian yang sudah dilakukan pengambilan ciri karakteristik dapat dilihat pada Gambar 4.13.



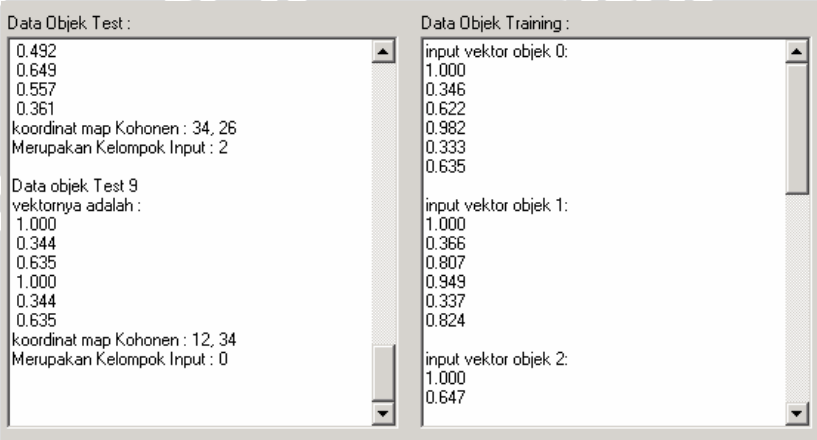
Gambar 4.13 *page* pengujian

Citra yang akan digunakan untuk pengujian diambil dengan menekan *button open image* dan akan ditampilkan pada komponen *Image* kecil hanya untuk sekedar dapat diketahui citra aslinya.



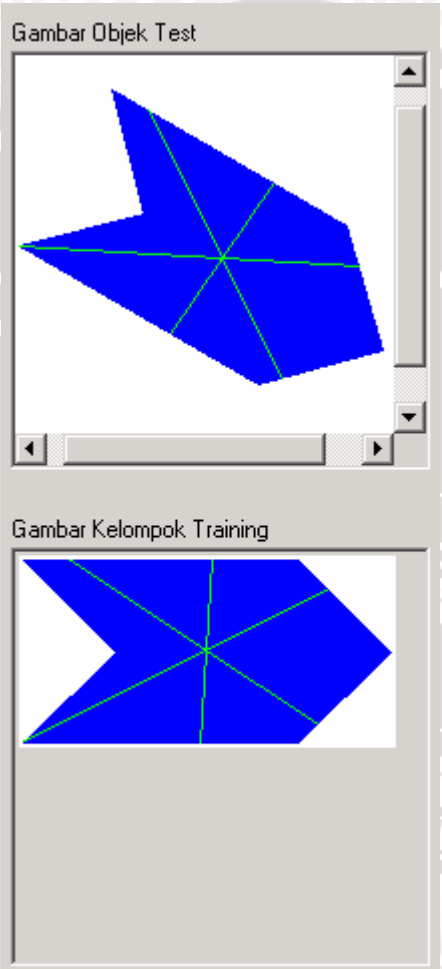
Gambar 4.14 obyek-obyek test

Setelah disegmentasi dan ditentukan cirinya, obyek-obyek test ditampilkan pada deretan image-image agar dapat dilihat tiap-tiap obyeknya. Pada *Memo* data obyek test ditampilkan data seluruh obyek test dari ciri karakteristiknya, representasi koordinatnya pada *cell* Kohonen dan pada kelompok mana obyek tersebut digolongkan. Disebelahnya juga ditampilkan sebuah *Memo* data ciri karakteristik dari *input training* sebagai pembanding.



Gambar 4.15 tampilan data seluruh obyek test

Kesimpulan hasil pada kelompok mana obyek test dikelompokkan dapat dilihat pada *Memo* Data Obyek Test, tetapi agar lebih mudah dipahami dapat ditampilkan juga gambar obyek trainingnya. Ini dilakukan dengan *click* pada obyek test yang ingin diketahui, dan akan didapatkan tampilan pada Gambar 4.16



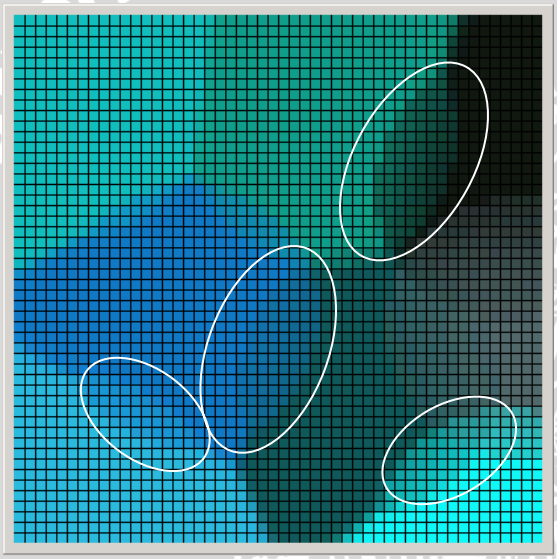
Gambar 4.16 pencocokkan kelompok obyek test

Dengan hasil berupa tampilan gambar obyeknya ini dapat lebih mudah melihat kelompoknya.

4.4 Pengujian Sistem

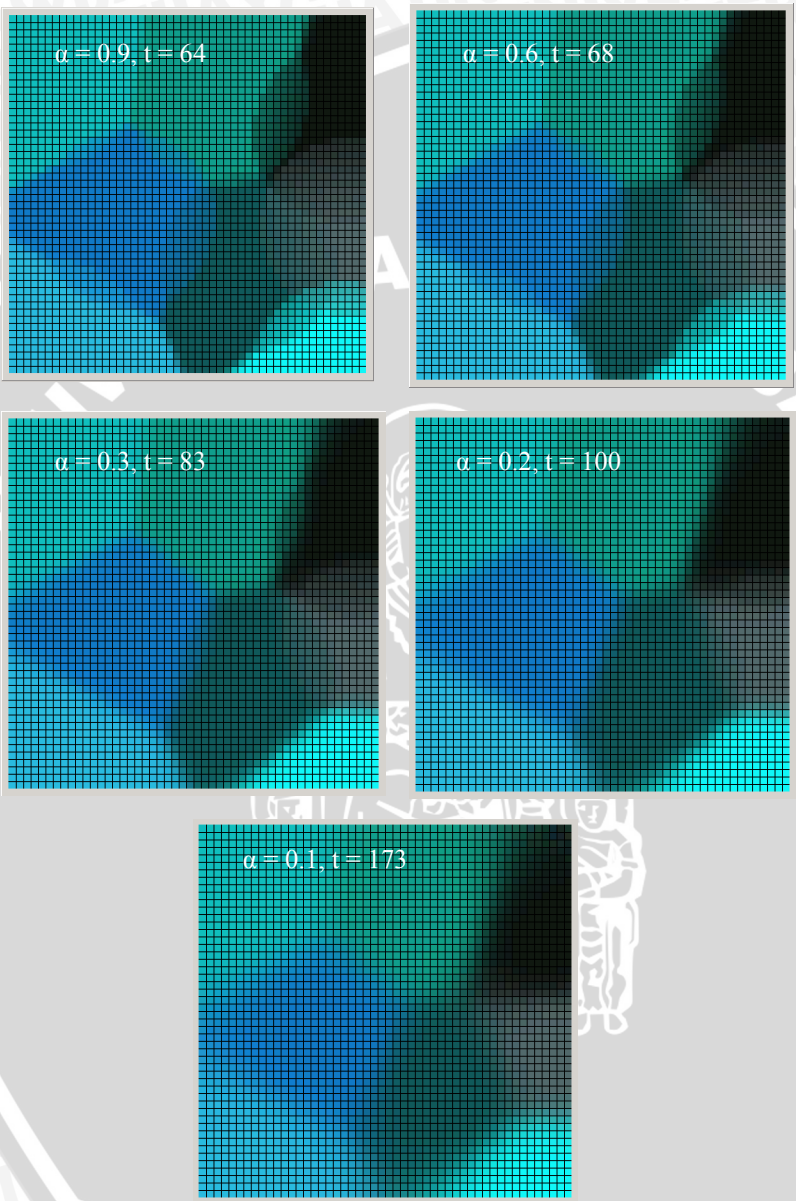
4.4.1 Pengujian Proses Pelatihan Kohonen SOM

Untuk mengetahui nilai *learning rate* (α) yang tepat untuk digunakan pada proses pelatihan dilakukan uji coba berikut. Pengujian berikut ini dilakukan dengan 8 *input training*, pada *map* Kohonen dengan ukuran 50 x 50 *cell*, radius mula-mula adalah setengah lebar *map* dan maximum iterasi (t) adalah 300 serta toleransi maksimum yang diperbolehkan untuk untuk perbedaan antara vektor BMU dengan vektor *input training* adalah 0.001.



Gambar 4.17 bobot dengan penyesuaian tidak sempurna

Terdapat daerah dengan vektor bobot yang saling tumpang tindih karena penyesuaian beberapa BMU yang berada di sekitarnya, *cell-cell* yang berada pada daerah ini tidak jelas berada pada kelompok BMU yang mana. Seperti diperlihatkan pada Gambar 4.41, dengan menggunakan α = 0.9 ternyata proses sudah berhenti pada iterasi t = 64 tapi ternyata menghasilkan banyak daerah yang seperti ini, untuk itu perlu dicari α yang mampu menyesuaikan vektor bobot *map* Kohonen dengan lebih sempurna.



Gambar 4.18 penggunaan α yang berbeda pada *map* Kohonen

Gambar 4.18 memperlihatkan *map* Kohonen yang sama tapi menggunakan α yang berbeda-beda. Seperti diperlihatkan semakin kecil nilai α, daerah dengan penyesuaian bobot yang tidak sempurna akan semakin sedikit dan sampai dengan α = 0.1 relatif tidak terjadi.

4.4.2 Pengujian Konvergensi

Untuk mengetahui iterasi maksimum yang optimal agar bobot-bobot pada *map* Kohonen konvergen dengan nilai toleransi perbedaan vektor BMU dengan vektor *input training* adalah maksimal 0.001, dilakukan pengujian berikut. *Input training* yang digunakan berjumlah 8 pada *map* Kohonen berukuran 50 x 50, dengan radius mula-mula setengah lebar *map* Kohonen dan saat ini α yang digunakan adalah 0.1. Percobaan ini dilakukan dengan menggunakan 3 macam vektor bobot random yang berbeda, Tabel 4.2 memperlihatkan perbedaan maksimal vektor BMU dengan vektor *input training* pada tiap iterasi ke-t.

Tabel 4.2 perbedaan maksimal pada tiap iterasi

iterasi	Data random I	Data random II	Data random III
50	0.05794	0.07504	0.06857
100	0.01785	0.02586	0.01824
150	0.00485	0.00864	0.00543
200	0.00139	0.00291	0.00196
250	0.00099	0.001	0.00099
300	0.00099	0.00098	0.00099

Seperti yang diperlihatkan perbedaan vektor bobot input dengan vektor BMU sudah mendekati batas yang diperbolehkan yaitu dibawah nilai maksimal 0.001 pada iterasi 250 sampai 300, sehingga dapat disimpulkan jumlah iterasi maksimum yang dibutuhkan sistem untuk mencapai bobot yang konvergen adalah 300 iterasi.

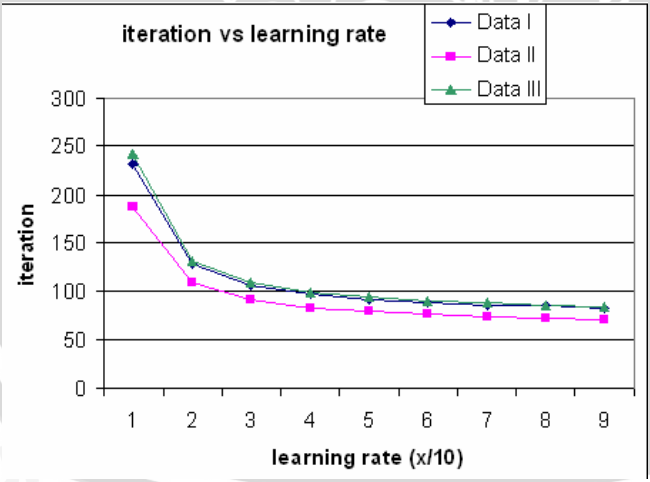
Terakhir akan dibandingkan jumlah iterasi sebenarnya yang diperlukan agar beda maksimal antara vektor bobot dengan BMU berada di bawah 0.001. Pengujian ini menggunakan 3 vektor bobot random yang berbeda, dilakukan menggunakan α dari 0.1 sampai 0.9

pada *map* Kohonen berukuran 50 x 50, radius awal adalah setengah lebar *map*, diuji dengan 8 *input training* dan iterasi maksimum yang digunakan adalah 300. Hasilnya diperlihatkan pada Tabel 4.3

Tabel 4.3 perbandingan α dengan iterasi

α	Iterasi total Data Random I	Iterasi total Data Random II	Iterasi total Data Random III
0.1	232	188	243
0.2	129	109	132
0.3	107	91	109
0.4	98	83	99
0.5	92	79	94
0.6	89	76	90
0.7	86	74	88
0.8	85	72	86
0.9	83	71	84

Untuk lebih jelasnya perbandingan ini dapat dibuat menjadi sebuah grafik pada Gambar 4.19



Gambar 4.19 hubungan iterasi total dengan α

4.4.3 Pengujian Pengenalan Obyek

Untuk pengenalan obyek pengujian dilakukan dengan cara mencocokkan obyek yang digunakan untuk *input training* dengan obyek yang sama tapi sudah mengalami perubahan kondisi. Perubahan kondisi yang digunakan disini adalah rotasi, refleksi, perbesaran 50% dan pengecilan 50% obyek. Pengujian akan bernilai benar apabila jawaban yang diberikan sesuai dengan *input training* yang diberikan.

Data akan dilatihkan dengan memberikan 7 obyek dari *input training* yang masing-masing dari ke 7 obyek ini akan mengalami 5 jenis perubahan, yaitu terdiri 2 rotasi, 1 refleksi, 1 perbesaran dan 1 pengecilan. Setelah diuji didapatkan hasilnya pada Tabel 4.4 dan kesimpulan pada Tabel 4.5

Tabel 4.4 hasil pengujian

Data	*Output* Hasil Rotasi 30^0	*Output* Hasil Rotasi 90^0	*Output* Hasil Refleksi	*Output* Hasil Perbesaran	*Output* Hasil Pengecilan
1	benar	benar	benar	benar	benar
2	benar	benar	salah	benar	benar
3	benar	benar	benar	benar	benar
4	benar	benar	benar	benar	benar
5	benar	benar	benar	benar	benar
6	benar	benar	salah	benar	benar

Tabel 4.5 kesimpulan hasil pengujian

Jenis Uji	Hasil benar	Hasil salah
Rotasi	6	0
Refleksi	4	2
Perbesaran	6	0
Pengecilan	6	0

Selain pengujian sebelumnya dilakukan juga sebuah percobaan dengan kondisi obyek test yang mengalami cacat. Pengujian

dilakukan untuk melihat pengaruh faktor jumlah ciri yang digunakan untuk mengenali objek.

Tabel 4.6 Hasil Pengujian

Data	Jumlah Ciri						
	6	8	9	12	18	24	36
1	benar	benar	benar	benar	benar	benar	benar
2	benar	benar	benar	benar	benar	benar	benar
3	benar	benar	benar	benar	benar	benar	benar
4	benar	benar	benar	benar	benar	benar	benar
5	benar	benar	benar	benar	benar	benar	benar
6	benar	benar	benar	benar	benar	benar	benar
7	salah	salah	benar	salah	benar	benar	benar
8	benar	benar	benar	benar	benar	benar	benar
9	benar	benar	benar	benar	benar	benar	benar

4.5 Analisa Hasil

4.5.1 Analisa Kohonen SOM

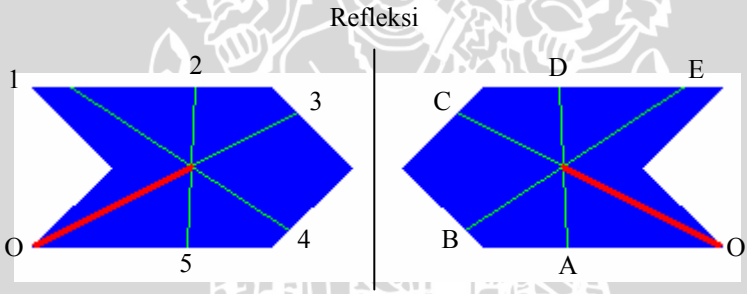
Penggunaan α yang besar dapat mempercepat penyesuaian vektor bobot dari BMU seperti diperlihatkan pada Gambar 4.18 iterasi yang dibutuhkan hanyalah 64. Tapi penggunaan α yang besar ini menyebabkan kecenderungan timbulnya daerah yang mengalami penyesuaian bobot yang tidak sempurna, ini disebabkan kecepatan penyesuaian bobot pada BMU tak mampu diikuti oleh penyesuaian bobot pada tetangga-tetangganya. Sehingga ketika vektor bobot dari BMU sudah mencapai nilai toleransi maksimum yang diizinkan ternyata penyesuaian vektor bobot milik tetangga-tetangganya belum sempurna. Solusinya adalah dengan menurunkan nilai α, dan seperti yang diperlihatkan Gambar 4.18 pada nilai α = 0.1 penyesuaian vektor bobot tetangga-tetangga sudah relatif mampu mengikuti penyesuaian BMU.

Untuk pemilihan iterasi maksimum dipilih 300 iterasi sesuai dengan Tabel 4.2 karena perbedaan vektor *input* dengan vektor bobot BMU sudah relatif konvergen dengan nilai bobot dibawah 0.001 sebelum iterasi ke 300.

Hubungan antara penggunaan α dengan iterasi diperlihatkan pada Tabel 4.3 dan grafiknya pada Gambar 4.19. Semakin besar nilai α maka akan semakin sedikit iterasi yang dibutuhkan, yang artinya α dengan iterasi berbanding terbalik. Dari ke tiga data *random* tersebut didapat pola grafik yang kurang lebih sama, perbedaan posisi data yang terjadi kemungkinan hanya disebabkan oleh data yang *random*.

4.5.2 Analisa Pengenalan Obyek

Seperti diperlihatkan pada Tabel 4.4 dan 4.5, pengujian tidak memberikan hasil yang akurat 100% bila obyek mengalami refleksi. Hasil yang tidak akurat pada obyek yang mengalami refleksi bisa dijelaskan sebagai berikut, obyek yang tidak simetris atau obyek simetris yang direfleksikan tidak pada sumbu simetrinya memiliki kemungkinan untuk terjadinya perubahan urutan pengambilan ciri karakteristik setelah dilakukan proses refleksi, untuk lebih jelasnya hasil diperlihatkan pada Gambar 4.20



Gambar 4.20 obyek pelatihan dan refleksinya

Pada Gambar 4.20 obyek di sebelah kiri adalah obyek mula-mula yang digunakan untuk *input training*, titik 0 adalah acuan awal dan ciri karakteristik obyek diambil searah jarum jam yaitu 0, 1, 2, 3, 4, dan 5 sehingga didapatkan vektor obyek berturut-turut adalah 1.000, 0.823, 0.453, 0.671, 0.652 dan 0.453. Sedangkan obyek sebelah kanan adalah obyek yang sama tapi telah direfleksikan pada sumbu y, karena proses pembulatan pada saat refleksi saat ini acuan untuk obyek di sebelah kanan adalah titik 0. Dengan pengambilan ciri

searah jarum jam yaitu 0, A, B, C, D dan E didapatkan vektor obyek adalah 1.000, 0.450, 0.648, 0.661, 0.443, dan 0.803.

Tabel 4.7 perbandingan obyek awal dengan refleksinya

Vektor obyek awal	Vektor obyek refleksi
1.000	1.000
0.823	0.450
0.453	0.648
0.671	0.661
0.652	0.443
0.453	0.803

Berdasarkan perbandingan pada Tabel 4.6 dapat dilihat karena proses refleksi, obyek yang sama dapat dikenali sebagai dua obyek yang berbeda. Perbedaannya dapat dilihat dari nilai vektor-vektornya tersebut yang memiliki selisih yang jauh, dan bila terdapat obyek pelatihan lain dengan vektor yang lebih dekat dengan obyek yang terotasi maka akan digolongkan sebagai anggota kelompok pelatihan yang lain.


UNIVERSITAS BRAWIJAYA

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Kesimpulan yang diambil dari penelitian ini adalah:

1. Pengenalan obyek dapat dilakukan dengan menggunakan Jaringan Syaraf Tiruan Kohonen *Self Organizing Map* walaupun tidak sepenuhnya akurat.
2. Penggunaan learning rate yang kecil akan sangat baik dalam proses pembelajaran tapi akan berbanding terbalik dengan jumlah iterasi proses yang dilakukan.
3. Penggunaan *signature object* dapat menurunkan jumlah data yang dibutuhkan untuk merepresentasikan suatu obyek.
4. Penggunaan *signature object* kadangkala dapat menimbulkan masalah untuk situasi tertentu sehingga diharapkan dapat menggunakan metode lain yang dapat merepresentasikan obyek dengan lebih akurat.
5. Penggunaan jumlah ciri yang besar pada *signature object* tidak selalu merepresentasikan objek dengan lebih tepat.
6. Dengan terdapatnya visualisasi *map* oleh jaringan Kohonen pengelompokkan suatu obyek dapat dilihat langsung dari letak topografinya.

5.2 Saran

Beberapa permasalahan sistem ini yang diharapkan bisa diperbaiki dan dikembangkan lebih lanjut:

1. Dapat mengembangkan jaringan yang dapat melatih diri sendiri bila diujikan dengan obyek lain yang dianggap berbeda dengan ambang batas tertentu.
2. Nilai ambang batas yang dapat digunakan untuk menilai suatu obyek termasuk ke dalam pengelompokkan adalah besarnya jarak Euclidean obyek tersebut dengan *node-node* pada *map* Kohonen, tapi seberapa besar nilai batas yang pasti masih membutuhkan penelitian lebih lanjut.


UNIVERSITAS BRAWIJAYA

Daftar Pustaka

AI-junkie. *Kohonen's Self Organizing Feature Maps*. http://www.ai-junkie.com/ann/som/ . Tanggal akses : 13 Maret 2007

Aksoy, Selim. *Introduction to Pattern Recognition*. http://www.cs.bilkent.edu.tr/~saksoy/courses/cs551/slides/cs551_intro.pdf : 2007. Tanggal akses : 15 Mei 2007

Ang Wei Siong, Resmana. Pengenalan Citra Obyek Sederhana dengan Menggunakan Metode Jaringan Saraf Tiruan SOM. http://fportfolio.petra.ac.id/user_files/91-024/som-image-recog.doc : 1999. Tanggal akses : 28 Juni 2009

Heaton, Jeff. *Introduction to Neural Networks with Java*. http://www.heatonresearch.com/articles/series/1/ : Wednesday, November 16, 2005 05:15 PM. Tanggal akses: 15 Maret 2007

Jain, Ramesh, *Machine Vision*, Penerbit McGraw-Hill. 1995

Muis, Saludin. *Teknik Jaringan Syaraf Tiruan* : Maret 2006. Penerbit Graha Ilmu. Yogyakarta: 2006

Patterson, Dan W. *Artificial Neural Network Theory and Application*. Prentice Hall, Singapore: 1996

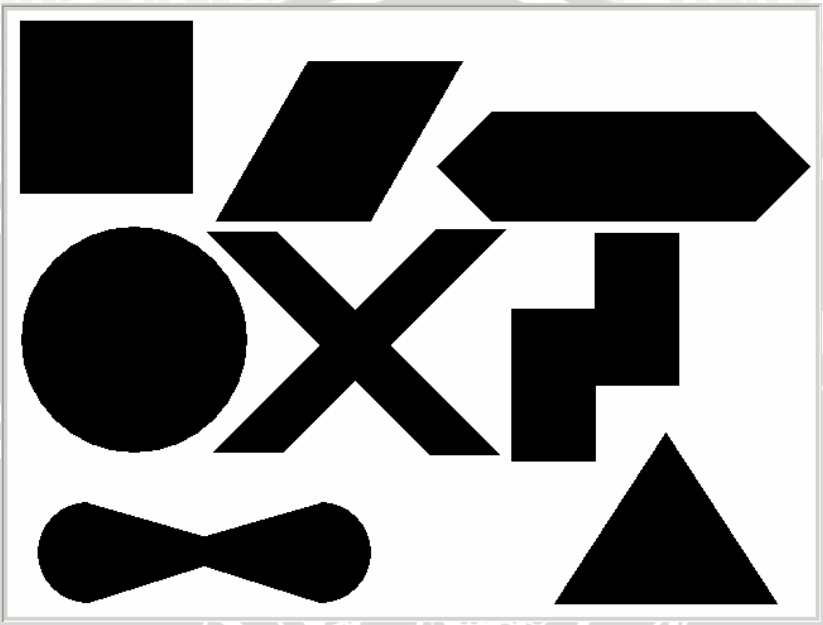
Munir, Rinaldi. *Pengolahan Citra Digital*. Penerbit Informatika Bandung. Bandung 2004

Schalkoff, Robert. *Pattern Recognition: Statistical, Structural and Neural Approaches*. Penerbit John Wiley & Sons. Singapore: 1992

Sarle, Warren S. *ANN_FAQ*. ftp://ftp.sas.com/pub/neural/FAQ.html: 2002-05-17. Tanggal akses: 8 Maret 2007

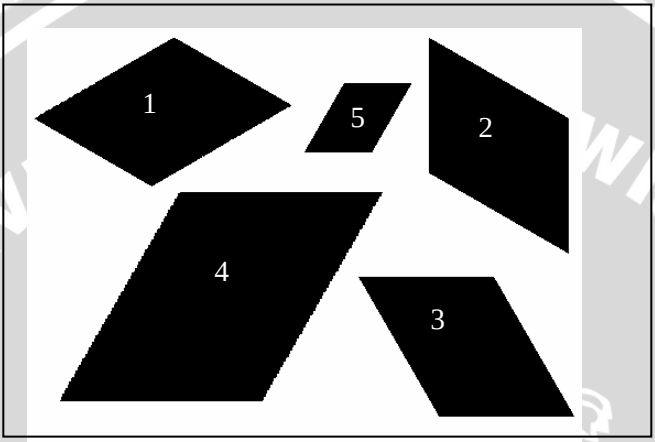

UNIVERSITAS BRAWIJAYA

Lampiran 1 Sample Data Pelatihan

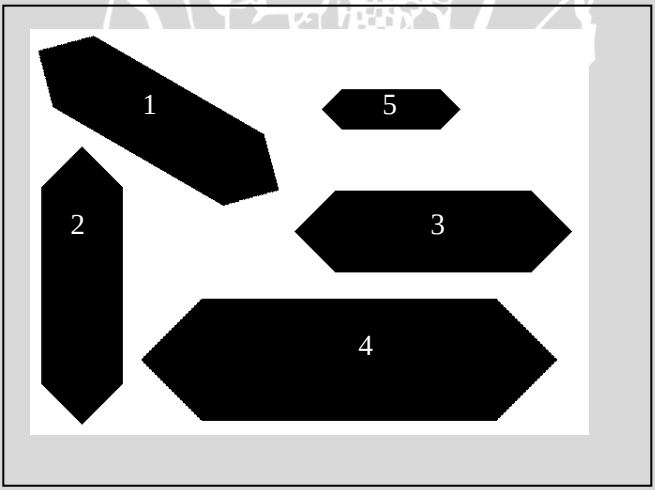


Lampiran 2 Sample Data Pengujian Rotasi, Refleksi, Perbesaran dan Pengecilan dan Hasilnya

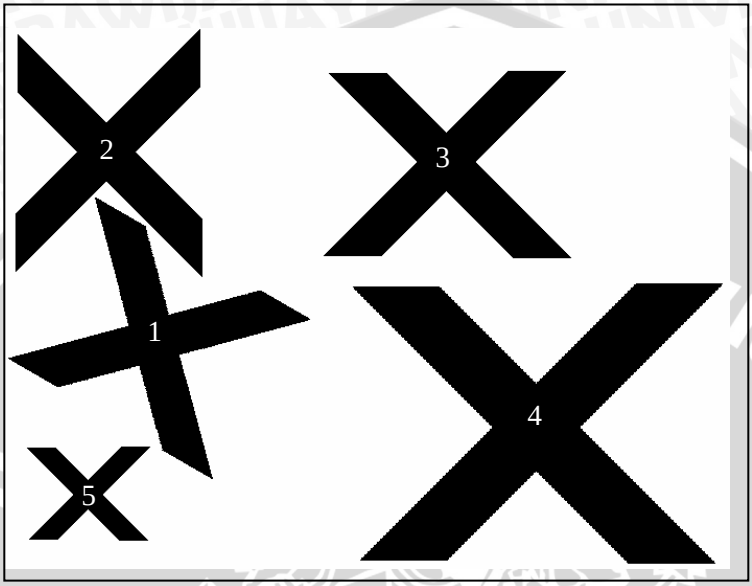
Test Data Set 1:



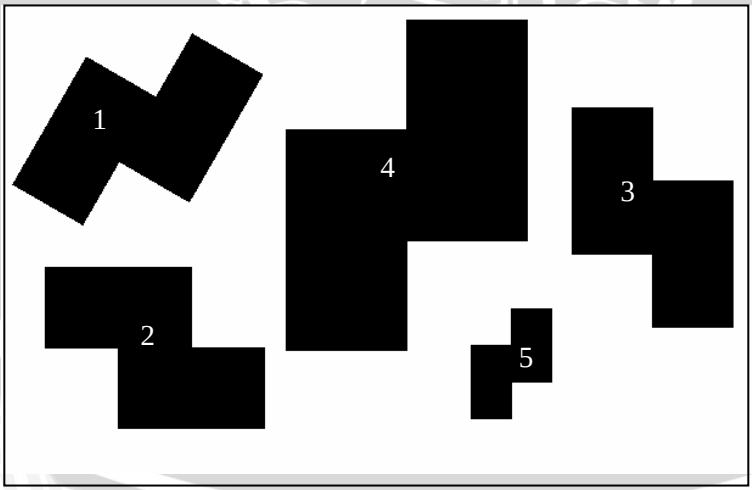
Test Data Set 2:



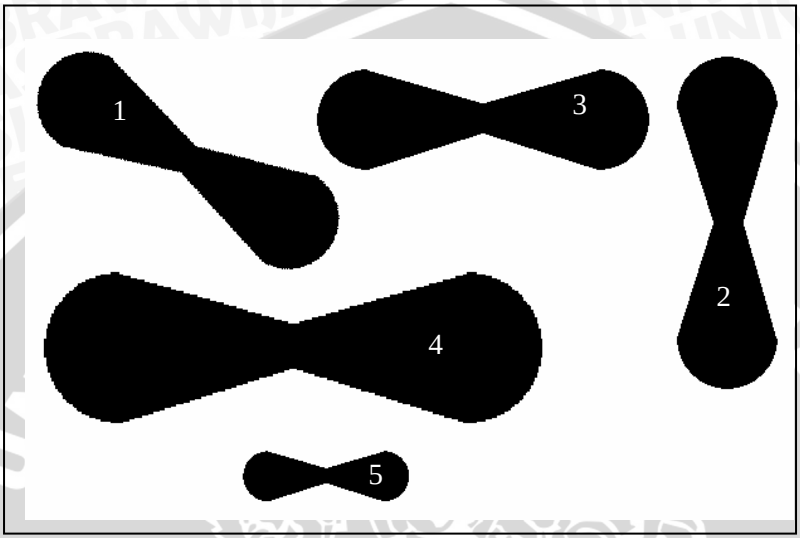
Test Data Set 3



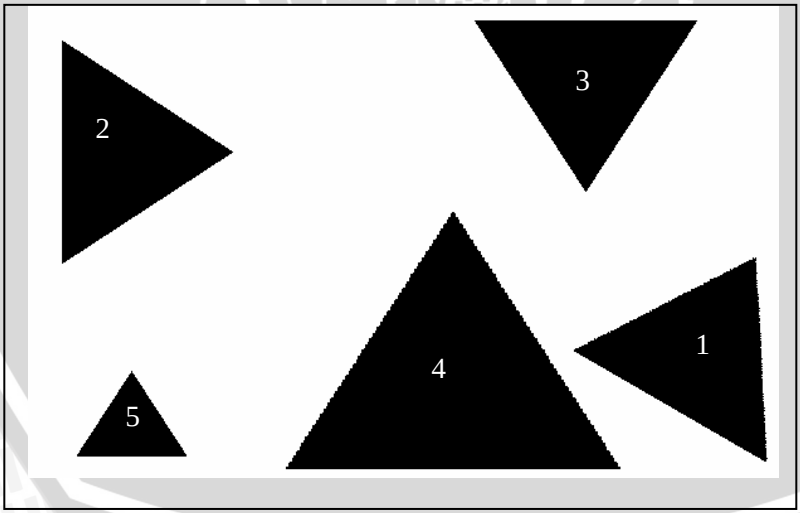
Test Data Set 4



Test Data Set 5



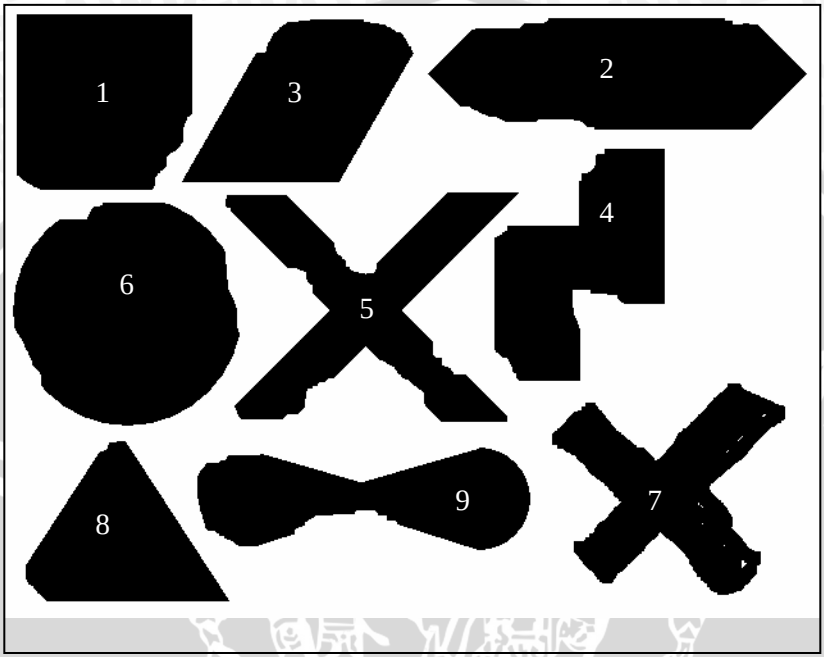
Test Data Set 6



Hasil pengujian menggunakan dengan data set di atas menggunakan jumlah ciri 6

Data	*Output* Hasil Rotasi 30^0(1)	*Output* Hasil Rotasi 90^0(2)	*Output* Hasil Refleksi (3)	*Output* Hasil Perbesaran (4)	*Output* Hasil Pengecilan (5)
1	benar	benar	benar	benar	benar
2	benar	benar	salah	benar	benar
3	benar	benar	benar	benar	benar
4	benar	benar	benar	benar	benar
5	benar	benar	benar	benar	benar
6	benar	benar	salah	benar	benar

Lampiran 3 Sample Data Pengujian yang cacat dan hasilnya



Data	Jumlah Ciri						
	6	8	9	12	18	24	36
1	benar	benar	benar	benar	benar	benar	benar
2	benar	benar	benar	benar	benar	benar	benar
3	benar	benar	benar	benar	benar	benar	benar
4	benar	benar	benar	benar	benar	benar	benar
5	benar	benar	benar	benar	benar	benar	benar
6	benar	benar	benar	benar	benar	benar	benar
7	salah	salah	benar	salah	benar	benar	benar
8	benar	benar	benar	benar	benar	benar	benar
9	benar	benar	benar	benar	benar	benar	benar