BAB 2

LANDASAN TEORI

2.1 *Image*

Menurut Pearson (1991, p1), *image* adalah gambaran dua dimensi dari dunia visual. Schalkoff (1989, p9) mendefinisikan *image*, yang biasanya disebut dengan gambar (*picture*), sebagai suatu bentuk dua dimensi yang dihasilkan melalui proses melihat atau proses merasakan sebuah *scene*. *Scene* sendiri didefinisikan Schalkoff (1989, p9) sebagai kumpulan dari objek tiga dimensi. Untuk memudahkan dalam pembahasan selanjutnya maka kata *image* akan disebut dengan gambar.

2.2 *Image Indexing*

Menurut Hendra (1999), *image indexing* diartikan sebagai metode-metode dari pengelompokkan gambar-gambar digital dalam beberapa cara (menggunakan fitur-fitur gambar) sehingga gambar tertentu dapat ditemukan dengan cepat yang selanjutnya digunakan untuk proses pengambilan informasi gambar .

Image indexing dapat didasarkan pada fitur seperti warna, tekstur, bentuk, sketsa dan *spatial relationships*. Berikut ini penjelasan secara singkat yang menerangkan macam-macam metode yang diajukan untuk mengindeks informasi gambar.

2.2.1 *Image Indexing* berdasarkanWarna

Sebuah gambar digital terdiri dari susunan *pixel-pixel* dua dimensi yang membentuk sebuah refleksi visual yang akan dirasakan oleh mata manusia sebagai objek

sebenarnya atau buatan. Tampilan gambar ditentukan oleh warna dari tiap-tiap *pixel*. Dengan kata lain, warna adalah sebuah atribut penting untuk merepresentasikan gambar. Oleh karena itu, banyak peneliti memfokuskan elemen ini sebagai sebuah kunci dalam memanggil kembali gambar dari database gambar.

Sebuah permintaan yang umum *image indexing* berdasarkan warna, sebagai contoh, untuk menemukan seluruh gambar dengan panorama perkebunan, dimana warna biru dan hijau adalah warna dominan (langit dan kebun teh). Contoh permintaan lain adalah mencari gambar yang memiliki 40% warna putih dan 30% warna merah.

2.2.2 *Image Indexing* berdasarkanTekstur

Perhitungan tekstur yang diterapkan adalah sebagai berikut : kekasaran (menghitung butiran halus), kekontrasan (menghitung berdasarkan distribusi tingkat keabuan), dan arah (menghitung tampilan arah gambar tertentu).

Umumnya, mengindeks dan mencari gambar berdasarkan tekstur berguna ketika pemakai tertarik untuk mencari tekstur gambar. Sebagai contoh, seorang perancang busana yang ingin memeriksa apakah tekstur tersebut sudah pernah digunakan sebelumnya (hanya jika dia memiliki database gambar tekstur yang besar).

2.2.3 *Image Indexing* berdasarkan Bentuk

Secara umum, fitur bentuk berguna pada sistem yang memiliki warna dan tekstur dari objek adalah sama. Contohnya mencakup database gambar medis, dimana warna dan tekstur dari objek anatomi sama, dan dalam database gambar *clip-art* yang biasanya mencakup bagian warna yang sama dipisahkan dengan garis yang jelas. Secara umum,

segmentasi gambar dan metode pendeteksi sudut diperlukan untuk mencari *profile* objek. Pengguna dapat juga membantu sistem secara manual memberi outline objek gambar.

2.2.4 *Image Indexing* berdasarkan Sketsa

Cara lain dengan menggambarkan isi dari sebuah gambar menggunakan sebuah sketsa dimana sebuah gambar mengandung outline gambar. Dalam permintaan, pemakai dapat menggambar garis dan sudut yang dominan pada gambar, dan sistem akan mencari gambar yang menyerupai garis atau sudut tersebut.

2.2.5 *Image Indexing* berdasarkan *Spatial Relationships*

Image indexing berdasarkan *spatial relationships* berarti pencarian gambar berdasarkan posisi-posisi dalam suatu ruang. Mencari gambar yang memenuhi permintaan *spatial* adalah sesuatu yang penting dari sistem database gambar. Sebagai contoh pencarian informasi gambar adalah "temukan seluruh gambar yang menunjukkan sebuah pohon di sebelah kiri sebuah rumah". Pada umumnya, kita menganggap permintaan-permintaan untuk mencari seluruh gambar mengandung pola dasar yang diberikan, yaitu seluruh gambar dimana beberapa objek muncul pada suatu aturan *spatial* tertentu. *Chang et al,* telah memperkenalkan suatu metode *indexing iconic* berdasarkan gambaran 2D string. Sebagai langkah pertama untuk mendapat suatu indeks *iconic* suatu gambar nyata, isi secara logik dari gambar dikenali dan suatu gambar simbolik di asosiasikan dengan gambar nyata. Gambar simbolik adalah suatu matriks simbol dua dimensi. Masing-masing objek dari gambar nyata direpresentasikan dengan simbol yang ditempatkan pada objek. Jadi indeks *iconic* gambar adalah representasi

padat dari gambar simbolik yang didapat dengan merancang simbol-simbol dengan arah x dan y. (Rothwell, pp45-46).

2.3 Database

Pada *image indexing* dengan *holographic memory* ini digunakan database sebagai tempat menyimpan file gambar dan file keterangannya. McLeod (1995, p305) mendefinisikan database sebagai kumpulan data dalam komputer yang terintegrasi, diatur dengan baik dan disimpan sedemikian adanya sehingga memudahkan pengambilan data tersebut. Date (1990, p5), membagi beberapa elemen dalam suatu struktur database yaitu terdiri dari data yang dimasukkan dan data yang dihasilkan. Jadi database terdiri dari beberapa kumpulan data yang tetap yang dipakai oleh sistem aplikasi terkait.

2.4 Prototype (prototipe)

Seringkali pemakai mendefinisikan apa yang menjadi kebutuhan secara global, tidak secara mendetil terutama dalam masalah input, proses, output yang diinginkan. Pada kasus lainnya, pengembang bisa jadi tidak yakin akan efisiensi algoritma, adaptasi dari sistem operasi, tingkat interaksi pemakai yang diinginkan. Oleh karena itu *prototyping* dapat menjadi salah satu pendekatan yang baik.

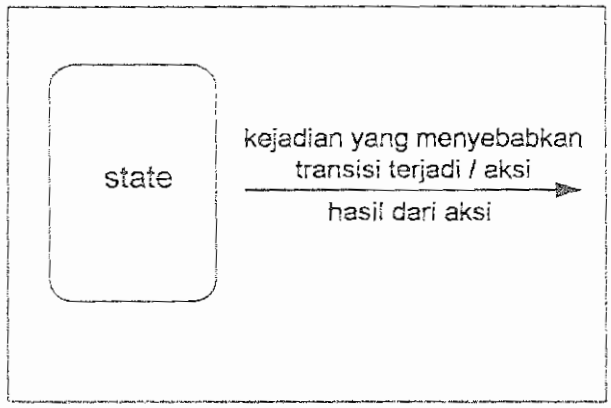
Menurut Pressman (1997, p32), *prototyping* adalah suatu proses yang memungkinkan pembangunan perangkat lunak yang akan dibangun. Prinsip dasar dari *prototyping* adalah pemakai dapat lebih mudah untuk menilai sistem atau produk dalam bentuk model yang sudah ada daripada bentuk teori.

Langkah-langkah untuk mendapatkan prototipe yang baik adalah sebagai berikut :

- mengevaluasi dan menentukan apakah perangkat lunak yang akan dikembangkan merupakan sebuah pilihan yang pantas untuk digunakan .
- membuat representasi dari persyaratan yang disetujui.
- merancang aspek dari persyaratan yang disetujui.
- Tahap pembuatan model tersebut.
- Memperbaiki model yang sudah diuji oleh pemakai.
- Mengulang langkah-langkah diatas sesuai dengan kebutuhan pemakai.

2.5 *State Transition Diagram* (STD)

Untuk membantu perancangan prototipe *holographic memory* ini maka digunakan *State Transition Diagram* (STD). STD (Pressman, 1997, pp300-301) adalah suatu model yang menunjukkan bagaimana sistem menunjukkan reaksi jika ada aksi-aksi dari luar. Untuk mencapai ini, STD menggambarkan *state* dan sistem cara kerja perpindahan yang dibentuk dari suatu state ke state lainnya. Simbol-simbol yang digunakan suatu STD adalah sebagai berikut :



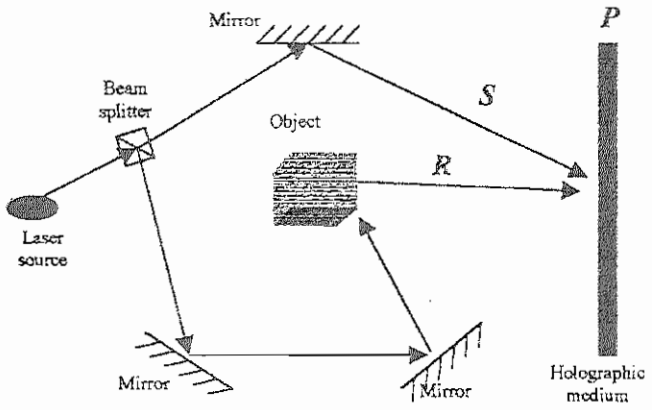
Gambar 2.1 Komponen STD

Keterangan :

- Gambar persegi panjang tumpul menunjukkan state/keadaan dari sistem
- Gambar panah menunjukkan transisi antar sistem. Tiap panah diberi label dengan ekspresi aturan tertentu. Label yang diatas menunjukkan aksi atau kejadian yang menyebabkan transisi terjadi. Label yang dibawah menunjukan akibat dari aksi.

2.6 Optik *Holography*

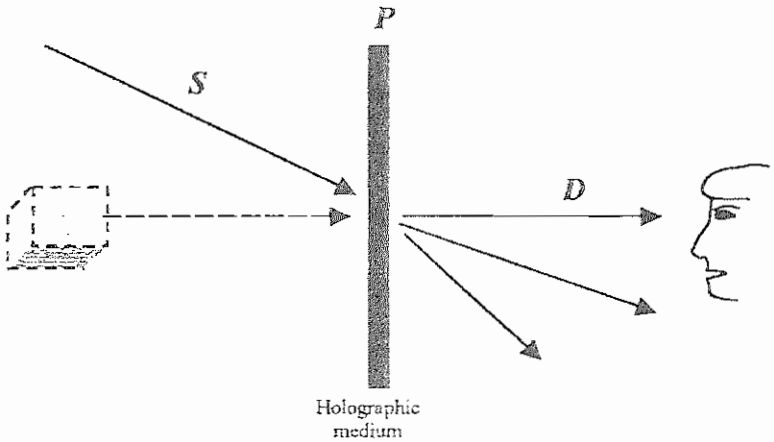
Menurut Hendra (1999, p29) *Holography* adalah suatu metode untuk merekam gambar tiga dimensi dari suatu objek ke dalam suatu media *holographic*. Media yang digunakan dapat berupa bahan atau film khusus untuk fotografi atau bahan kristal khusus. Proses perekamannya dilakukan dengan cara menyinari media tersebut dengan berkas laser yang dipantulkan oleh objek dan seberkas sinar laser referensi, yang berasal dari arah yang lain. Berkas-berkas laser tersebut harus berasal dari satu sumber yang sama, misalnya dengan memantulkan berkas laser yang asli pada sebuah cermin. Kedua berkas laser itu saling berinterferensi dan kemudian membentuk suatu pola pada media perekaman yang digunakan. Gambar 2.2 menggambarkan bagaimana proses pembuatan sebuah hologram. Berkas sinar referensi ditunjukan oleh *S*. Berkas sinar yang dipantulkan oleh objek tersebut ditunjukan oleh *R*.



Gambar 2.2 Pembuatan Hologram

. Ketika hologram disinari dengan seberkas sinar laser referensi yang identik dengan yang digunakan pada saat pembuatan, dan dari arah yang sama, maka hologram tadi akan memberikan visualisasi dari objek tersebut. Gambar 2.3 menunjukan proses rekonstruksi sebuah hologram. Dengan *S* adalah berkas referensi yang digunakan pada saat pembuatan hologram.

Seorang pengamat yang mengamati dari depan (arah dari mana *D* datang) akan melihat objek yang terekam pada hologram tersebut pada posisi aslinya sebagai sebuah visualisasi tiga dimensi.



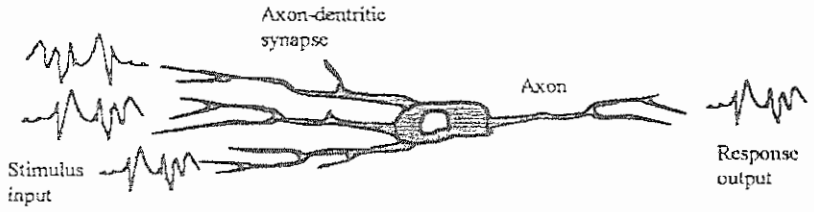
Gambar 2.3 Rekonstruksi sebuah holograph

Yang lebih menarik adalah, holograph mempunyai karakteristik yang unik, yaitu 'mengingat' asosiasi antara berkas referensi dan berkas pantulan objek. Selain bisa direkonstruksi dengan menggunakan berkas referensi, tetapi juga jika hologram tersebut direkonstruksi dengan menggunakan berkas pantulan objek yang bersangkutan (R), maka akan menghasilkan respon yang merupakan replika dari berkas referensi semula. Ide ini pertama kali diperkenalkan oleh Gabor (1969).

Fitur lain yang menarik dari holograph adalah, beberapa pasangan asosiasi antara berkas referensi dan berkas pantulan objek dapat disimpan pada hologram yang sama. Setelah asosiasi yang pertama (S_1,R_1) telah direkam, asosiasi berikutnya (S_i,R_i) bisa disimpan pada hologram yang sama. Dengan adanya R_i, maka S_i akan dapat direkonstruksi, dan sebaliknya.

Salah satu dari fitur yang penting dari holograph adalah kemampuannya untuk melakukan proses rekonstruksi dengan memanfaatkan sebagian dari berkas pantulan objek. Namun berkas referensi yang dihasilkan dari proses rekonstruksi itu banyak mengandung *noise*, tergantung tingkat distorsi yang terjadi ketika berkas tersebut dipantulkan keluar dari objek yang bersangkutan.

Sutherland (1990), mengatakan bahwa ada kemiripan antara *optical holography* dan sistem saraf dalam hal pemrosesan signal yang terjadi dalam *neuron* . Berikut ini adalah gambar sel *neuron* :



Gambar 2.4 Sebuah sel *neuron*

Rangsangan yang masuk *(stimulus)* dan reaksi yang keluar *(response)* saling berasosiasi dan disimpan dalam neuron sebagai sebuah transformasi non-iterasi. Pada model ini, baik *stimulus pattern*, *response pattern*, dan asosiasi antara keduanya bisa direpresentasikan dalam bentuk bilangan kompleks multidimensional. Asosiasi sejenis bisa disimpan dalam jumlah yang besar ke dalam sebuah set yang terdiri dari elemen-elemen kompleks (Hendra, p32).

2.7 *Holographic Memory*

Ide pokok *holographic memory* adalah prinsip-prinsip penyimpanan (*encoding*) dan pengambilan kembali informasi (*decoding)* dengan menggunakan asosiasi *stimulus-response pattern*, dengan berdasarkan pada analogi optik *holography*. Pada model *holographic memory*, secara matematis berkas referensi pada optik *holography* disubtitusikan dengan sebuah *response pattern*, dan berkas pantulan objek dengan sebuah *stimulus pattern*. Model perhitungan untuk membuat *holographic memory* dan bagaimana mengambil kembali informasi yang tersimpan di dalamnya diadaptasi dari Khan (1995, p31).

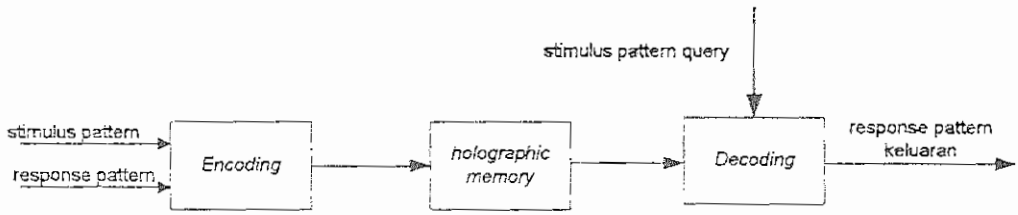
Pada *holographic memory* terdapat dua buah algoritma yang digunakan yaitu algoritma dengan proses *training (trained holograph)* dan algoritma tanpa proses *training (untrained holograph).*

2.7.1 *Untrained Holographic Memory*

Menyimpan *pattern* ke dalam *holographic memory* melibatkan proses *encoding* informasi semua *pattern* tersebut, sehingga dapat disimpan ke dalam *holographic memory* dan kemudian diambil kembali. Secara umum, proses *encoding*, baik pada

trained maupun *untrained holographic*, pertama-tama diawali dengan memetakan informasi suatu gambar ke *stimulus pattern*, setelah itu nomor indeks gambar dipetakan ke *response pattern*. Kemudian, *stimulus pattern* dan *response pattern* diasosiasikan satu sama lain, dan menghasilkan sebuah asosiasi *stimulus-response pattern*. Asosiasi inilah yang dimasukkan ke dalam *holograph*.

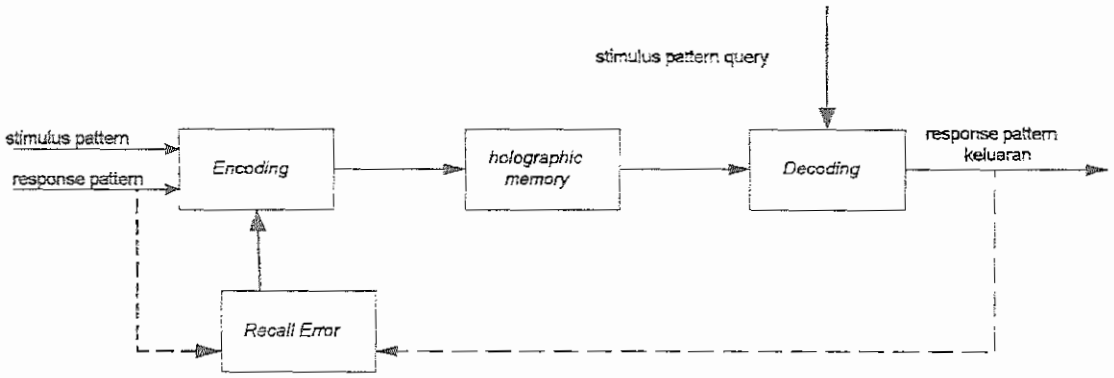
Pada *untrained holograph* tidak ada proses iterasi untuk meminimalisasi *error* sehingga kinerja pada proses *decoding untrained holograph* menjadi kurang baik. Berdasarkan penelitian Hendra (1999), bahwa pada kondisi dan nilai *load factor* (maksimum jumlah gambar yang dapat disimpan dalam *holographic memory*) tertentu akan memberikam kinerja yang baik pula.



Gambar 2.5 Proses *Untrained Holograph*

2.7.2 Trained Holographic Memory

Pada *holographic memory* dengan proses *training* terdapat proses '*learning*' yang melibatkan proses iterasi *encoding* untuk menimalkan *error* yang muncul pada saat *decoding* dengan berdasarkan asosiasi yang terbentuk pada saat *encoding*. Proses "*learning*" ini dilakukan dengan menghitung error yang merupakan selisih antara *response pattern* masukkan dengan *response pattern* keluaran, di-encode ulang untuk memperkecil *error* dan membuat *response pattern* keluaran mendekati atau sama dengan harga pada *response pattern* masukkan.



Gambar 2.6 Proses *Trained Holograph*

2.7.3 Pemetaan *stimulus pattern.*

Sebuah *stimulus pattern* bisa diekspresikan sebagai sebuah himpunan dengan masing-masing elemennya S_k, di mana tiap elemen mewakili sebuah *pixel* ke-k :

$$S = (s_1, s_2, s_3, \ldots, s_k)$$

Secara keseluruhan elemen-elemen ini membentuk suatu pola yang akan disimpan ke dalam memori.

Informasi yang diambil dari masing-masing elemen adalah nilai intensitas *pixel* dan *meta-knowledge-nya. Meta-knowledge* suatu elemen menunjukkan tingkat keyakinan elemen yang telah ditetapkan untuk menentukan *pixel* yang menjadi fokus gambar. Nilai intensitas *pixel* ke-k dan *meta-knowledge*-nya tersebut dituliskan sebagai model berikut ini:

$$s_k = (\alpha_k, \beta_k)$$

Bentuk di atas kemudian ditransformasikan ke dalam gambaran bilangan kompleks, ditunjukkan sebagai sebuah vektor dalam ruang dua dimensi menurut ekspresi berikut ini :

$$s_k = (\alpha_k, \beta_k) \Rightarrow \lambda_k e^{i\theta_k}$$

Dalam transformasi ini, dengan menggunakan fungsi khusus, α_k dipetakan ke dalam elemen fase θ_k dalam batasan 0 - 2π. Sedangkan *meta-knowledge*, β_k, dipetakan ke dalam λ_k (magnitude), dalam interval [0,1]. Nilai satu menyatakan elemen-elemen yang tingkat peranannya lebih tinggi, dan nilai yang lebih rendah menyatakan sebaliknya. Jika nilai semua *magnitude* ditentukan menjadi nol, maka elemen-elemen tersebut tidak terlibat dalam proses-proses selanjutnya.

Dengan menggunakan model ini, masing-masing α_k dipetakan ke dalam suatu himpunan elemen-elemen fase θ_k. Masing-masing θ_k merupakan suatu proyeksi terhadap vektor tersebut.

Sehingga sebuah *stimulus pattern* dengan n elemen bisa direpresentasikan sebagai berikut:

$$\left[S^{\mu}\right]=\left|\lambda_1^{\mu}e^{\left(i\theta_1^{\mu}\right)},\lambda_2^{\mu}e^{\left(i\theta_2^{\mu}\right)},\ldots,\lambda_n^{\mu}e^{\left(i\theta_n^{\mu}\right)}\right| \qquad (2.1)$$

dimana, μ adalah indeks *pattern* tersebut, dan n adalah jumlah elemen pada *stimulus pattern* tersebut. Nilai *magnitude*-nya, λ_k^{μ}, menunjukkan tingkat keyakinan elemen ke-k dalam *pattern* ke-μ.

2.7.4 Pemetaan *Response Pattern.*

Representasi suatu *response pattern* mirip dengan *stimulus pattern* :

$$\left[R^{\mu}\right]=\left|\gamma_1^{\mu}e^{\left(i\phi_1^{\mu}\right)},\gamma_2^{\mu}e^{\left(i\phi_2^{\mu}\right)},\ldots,\gamma_m^{\mu}e^{\left(i\phi_m^{\mu}\right)}\right| \qquad (2.2)$$

dimana, R^{μ} adalah *response pattern* yang berasosiasi dengan *stimulus pattern* S^{μ}, dan m adalah jumlah elemen *response pattern*. Komponen fasa ϕ_k^{μ} mewakili nilai indeks elemen *response pattern*, dan *magnitude* γ_k^{μ} (*magnitude)* mewakili tingkat keyakinan

dari elemen tersebut . Nilai *magnitude* dari setiap *response pattern* umumnya ditentukan menjadi satu, dan melalui proses *decoding* nilai yang diambil kembali diharapkan bisa mencapai angka yang paling dekat dengan itu.

Setiap elemen dalam *response pattern* bisa dianggap sebagai vektor, dimana nilai fasa masing-masing vektor tersebut berada dalam interval 0-2π. Jika q adalah jumlah interval, maka untuk menghitung banyaknya elemen m dalam *response pattern* digunakan perhitungan sebagai berikut:

$$m = \log_q p \tag{2.3}$$

Dengan p adalah maksimum jumlah *pattern* yang dapat di-*encode*.

2.7.5 Proses *encoding*

Setiap *stimulus* dan *response pattern* itu kemudian diasosiasikan satu sama lain untuk membentuk sebuah matriks korelasi [X], yang juga disebut *stimulus-response pattern association*:

$$\left[X^{\mu}\right] = \left[\overline{S}^{\mu}\right]^{T}\left[R^{\mu}\right] \tag{2.4}$$

Di sini, $\left[\overline{S}^{\mu}\right]^{T}$ adalah *transpose* konjugasi dari $[S^{\mu}]$. Karena $[S^{\mu}]$ sebuah matriks dengan n elemen, dan $[R^{\mu}]$ adalah sebuah matriks dengan m elemen, maka $[X^{\mu}]$ adalah matriks dengan ordo n x m.

Beberapa asosiasi *stimulus-response pattern* dapat disimpan dalam sebuah matriks yang sama:

$$\left[X\right] = \sum_{\mu=1}^{p}\left[X^{\mu}\right] = \sum_{\mu=1}^{p}\left[\overline{S}^{\mu}\right]^{T}\left[R^{\mu}\right] \tag{2.5}$$

dimana p adalah jumlah *pattern* (jumlah asosiasi *stimulus-response pattern*). Matriks [X] di sini disebut *holograph matrix.*

Ekspresi matematika di atas adalah untuk *encoding* tanpa proses *training.* Berikut ini adalah ekspresi matematika yang digunakan oleh Khan (1995, p37) untuk meng-*encode holographic memory* dengan proses *training* :

$$[X]=[X]+L[\overline{S}]^{T}\left([R]-\frac{1}{c}[S][X]\right) \tag{2.6}$$

dimana L adalah *learning constant* yang berada antara 0-1. R adalah *response pattern* masukkan, 1/c [S] [X] adalah rumus proses *decoding* untuk memperoleh *response pattern* keluaran (*retrieved response pattern*) yang akan dijelaskan pada subbab berikutnya.

2.7.6 Proses *Decoding*

Proses *decoding* ditujukan untuk mendapat indeks gambar dari sebuah *pattern target* yang cocok dengan *query pattern.* Hal pertama dalam proses *decoding* ialah memetakan *query pattern* ke dalam *stimulus pattern* $[S^q]$ yang didefinisikan dengan cara yang sama seperti pada proses *encoding* :

$$[S^q]=\left[\lambda_1^q e^{(i\theta_1^q)},\lambda_2^q e^{(i\theta_2^q)},\ldots,\lambda_n^q e^{(i\theta_n^q)}\right] \tag{2.7}$$

Hasil dari proses *decoding* adalah *retrieved response pattern.* Untuk menghitung *response pattern*-nya dilakukan dengan cara mengalikan *query stimulus pattern* dengan *holograph matrix.*

$$[R^q]=\frac{1}{c}[S^q][X] \tag{2.8}$$

dengan c adalah nilai faktor normalisasi yang dihitung dengan:

$$c = \sum_{k=1}^{n} \lambda_k^q$$

Setelah itu *response pattern* dipetakan kembali ke sebuah angka indeks.

Jika semua elemen pada *query stimulus pattern* $[S^q]$ cocok dengan *stimulus pattern* awal $[S^t]$, maka nilai *response pattern* yang didapatkan $[R^q]$ akan mendekati nilai *response pattern* awal$[R^t]$, dalam hal ini adalah nilai fasa dan tingkat keyakinan dari tiap-tiap elemen.

Jika hanya sebagian elemen pada *query stimulus pattern* tersebut yang cocok dengan *stimulus pattern* awal, maka nilai tingkat kepercayaan yang diharapkan akan lebih kecil dari satu. Nilai tersebut bahkan bisa jauh lebih rendah jika *query stimulus pattern* tersebut sama sekali tidak cocok dengan *stimulus pattern* yang asli.

2.8 Komponen *Principal* dan *Crosstalk*

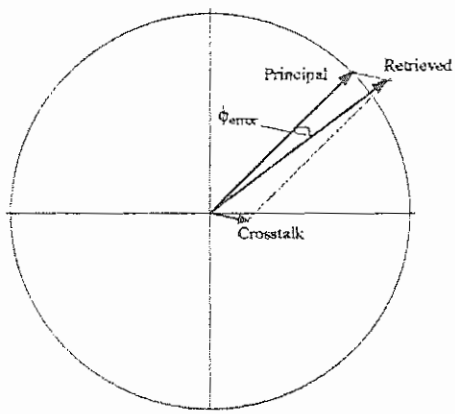
Untuk memahami bagaimana proses memperoleh *response pattern*, persamaan 2.8 dibagi menjadi dua bagian. Bagian kiri dikelompokan sebagai komponen-komponen *principal*, dan bagian kanan sebagai komponen-komponen *crosstalk*, Khan (1995, p34), dengan cara mensubstitusikan [X] pada Persamaan 2.5 ke dalam Persamaan 2.8 seperti berikut:

$$\begin{aligned} [R^q] &= \frac{1}{c}[S^q][\bar{S}^t]^T[R^t] + \frac{1}{c}\sum_{\substack{\mu=1 \\ \mu \neq t}}^{p}[S^q][\bar{S}^\mu]^T[R^\mu] \\ &= [R^q_{\text{principal}}] + [R^q_{\text{crosstalk}}]. \end{aligned} \tag{2.9}$$

Di sini, $[S^t]$ adalah *stimulus pattern* awal yang hampir seluruhnya cocok dengan *query stimulus pattern* $[S^q]$, dan $[R^t]$ adalah *response pattern* awal yang berasosiasi dengan *stimulus pattern* $[S^t]$ pada proses *encoding*. Ketika *query stimulus pattern* secara keseluruhan cocok dengan *stimulus pattern* yang asli, maka $\frac{1}{c}[S^q][\overline{S}^t]^T$ menjadi satu kesatuan, dan karena itu komponen *principal* tidak lain adalah sama dengan *response pattern* awal.

Sedangkan *crosstalk*, dijelaskan sebagai *noise* atau interferensi yang berasal dari asosiasi *stimulus-response pattern* yang lain, yang dikarenakan *query stimulus pattern* $[S^q]$ tidak cocok dengan *stimulus pattern* lain yang di-*enconde* $[S^\mu]$, dimana $1 \leq \mu \leq p$, $\mu \neq t$.

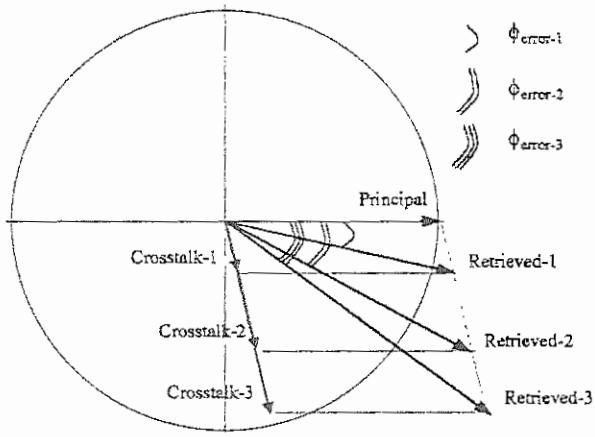
Karena elemen-elemen pada *stimulus pattern* dan *response pattern* dapat dianggap sebagai vektor, maka elemen-elemen untuk komponen *principal* dan *crosstalk* dapat dianggap sebagai resultan dari kedua vektor tersebut. Dengan demikian penjelasan di atas dapat digambarkan sebagai berikut:



Gambar 2.7 *Principal*, *crosstalk* dan vektor *retrieved response* yang dihasilkan

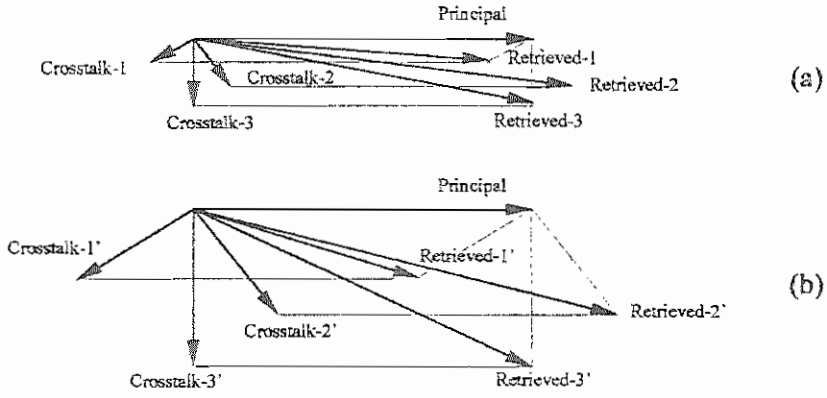
Setiap elemen pada *query response pattern* dapat dihitung tanpa bergantung dengan elemen yang lain. jadi suatu elemen ke-j dari *query response pattern* $[R^q]$ berdiri sendiri. Di sini, elemen ke-j dari $[R^q]$ adalah resultan dari elemen ke-j pada $[R^q_{\text{principal}}]$ dan $[R^q_{\text{crosstalk}}]$. Gambar 2.7 menggambarkan elemen ke-j dari $[R^q]$, $[R^q_{\text{principal}}]$ dan $[R^q_{\text{crosstalk}}]$ sebagai vektor pada suatu ruang dua dimensi. Disini ϕ_{error} (*recall error*) adalah selisih antara nilai fasa yang diperoleh antara *retrieval* dengan vektor komponen *principal*.

Jika *magnitude* (besar vektor) *crosstalk* mendekati angka nol, berarti keterlibatan vektor *crosstalk* pada proses penghitungan vektor *response* lebih kecil dibandingkan dengan vektor *principal*. Karena itu vektor *response* yang didapat cenderung untuk mendekati vektor *principal*. Di sisi lain, jika yang terjadi adalah sebaliknya, yaitu *magnitude* (besar vektor) *crosstalk* relatif besar, maka keterlibatannya menjadi lebih dominan. Ini akan mengakibatkan vektor *response* yang didapat bergerak menjauh dari vektor *principal* sejauh nilai tertentu. Gambar 2.8 mengilustrasikan vektor *crosstalk* dengan nilai fasa yang sama dan *magnitude* yang berbeda. Gambar tersebut menggambarkan vektor *response* yang bergerak menjauh dari vektor *principal* sebagai akibat dari tingginya nilai *magnitude* pada vektor *crosstalk*. Pada gambar tersebut, $|crosstalk\text{-}3| > |crosstalk\text{-}2| > |crosstalk\text{-}1|$, maka $\phi_{\text{error-3}} > \phi_{\text{error-2}} > \phi_{\text{error-1}}$.



Gambar 2.8 Tingginya nilai *magnitude* pada komponen *crosstalk* mengakibatkan vektor *response* yang didapat bergerak menjauh dari komponen *principal*.

Arah vektor *crosstalk* relatif terhadap vektor *principal* juga mempengaruhi vektor *response*. Namun demikian, jika nilai *magnitude* dari vektor *crosstalk* relatif kecil, maka vektor *response* akan tetap mendekat ke arah vektor *principal* tidak peduli ke manapun arah vektor *crosstalk* tersebut. Sebagai contoh pada Gambar 2.9 terdapat dua kasus dengan dua vektor *crosstalk* masing-masing dengan nilai *magnitude* yang berbeda. *Crosstalk* pada Gambar 2.9 (a) mempunyai nilai *magnitude* yang lebih kecil daripada *crosstalk* pada Gambar 2.9 (b) dengan arah yang sama. Dari situ terlihat bahwa vektor respons yang didapat pada (a) lebih dekat dengan vektor *principal* dibandingkan dengan vektor *response* pada (b).



Gambar 2.9 : Efek arah *crosstalk* dan *magnitude*.

Arah *crosstalk* pada (a) dan (b) adalah sama, tetapi nilai *magnitude* dari *crosstalk* pada (b) lebih besar dibanding dengan yang ada pada (a).

2.9 *Load Factor*

Jumlah *pattern* yang secara efektif bisa disimpan dalam suatu *holograph* disebut sebagai *holograph capacity* (kapasitas dari *holograph*).

Secara teori, jumlah *pattern* (p) yang bisa disimpan dalam *holographic memory* untuk kinerja yang baik, tidak melebihi jumlah elemen *stimulus pattern* (n). Ini disebut sebagai *load factor* (L), yaitu perbandingan p terhadap n, untuk $p \leq n$:

$$L = \frac{p}{n}. \qquad (2.10)$$

Nilai L berada dalam batasan 0-1.0.

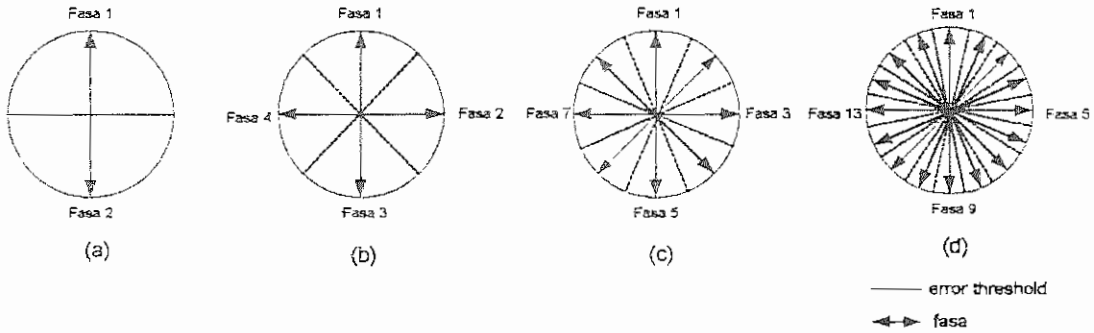
Seperti penjelasan sebelumnya, komponen *crosstalk* mempengaruhi *response pattern* yang dihasilkan. Selain itu, vektor *crosstalk* juga akan menjadi semakin besar seiring dengan bertambahnya jumlah *pattern* yang di-*encode* ke dalam *holograph*, yang berakibat membatasi jumlah *pattern* yang bisa di-*encode* ke dalam suatu *holograph* agar proses pengambilan kembali *pattern* tadi tetap baik kinerjanya. Dengan kata lain,

semakin banyak jumlah *pattern* yang di-*encode* ke dalam suatu *holograph*, semakin menurun kinerja proses *retrieval*-nya (proses pengambilan kembali suatu *pattern*).Hal ini terjadi pada *untrained*, namun dengan *trained holograph* hal ini telah teratasi, sehingga jumlah pattern yang disimpan dapat maksimum.

2.10 Perhitungan *Recall Error* dan *interval*

Dalam perhitungan *recall error* pada eksperimen, setiap indeks dipetakan ke sebuah *response pattern* yang unik dalam bentuk kombinasi nilai-nilai fasa, maka nilai *recall error* dihitung berdasarkan selisih nilai fasa antara kedua vektor tersebut (vektor *response* dan vektor *retrieved response*). Jumlah interval untuk setiap nilai fasa pada elemen-elemen *response pattern* menentukan tingkat *error threshold* yang diijinkan dalam proses *retrieval* tersebut.

Jika jumlah interval untuk sebuah *response pattern* adalah q, maka selisih fasa antara dua interval adalah $\frac{2\pi}{q}$. Tingkat *error threshold*, ϕ_T berjarak setengahnya, yaitu $\frac{\pi}{q}$.



Gambar 2.10 (a) *Interval* 2, (b) *Interval* 4, (c) *Interval* 8, (d) *Interval* 16

interval	q = 2	q = 4	q = 8	q = 16
Selisih fasa	180°	90°	45°	22.5°
Error Threshold	90°	45°	22.5°	11.25°

Tabel 2.1 Perbandingan Selisih fasa dan *Error Threshold* antara keempat *interval* yang berbeda

Berdasarkan hasil-hasil di atas, semakin sedikit jumlah *interval* akan memberikan nilai *threshold* yang lebih tinggi. Tingkat kesalahan untuk tiap *interval* yang berbeda juga berbeda. Karena itu, semakin sedikit jumlah *interval* semakin banyak jumlah *retrieval* yang tepat, karena sebagian besar dari *recall error* berada dibawah *error threshold*.

Asumsikan ϕ_i^t, sebagai nilai fasa untuk vektor *response pattern* yang asli, dan ϕ_i^q, sebagai nilai fasa untuk vektor *response pattern* yang didapat, untuk elemen yang ke-*i*, dimana $1 \le i \le m$ dan *m* adalah jumlah elemen di dalam *response pattern*. *Recall error* untuk elemen ke-*i*, dilambangkan dengan ϕ_i^e, dihitung dengan:

$$\phi_i^e = \min\left\{\left|\phi_i^t - \phi_i^q\right|, 2\pi - \left|\phi_i^t - \phi_i^q\right|\right\}$$

2.11 Pengukuran Unjuk Kerja

Dalam proses *encoding*, nilai *magnitude* seluruh vektor *response original* diberi nilai satu. Secara umum, diharapkan nilai *magnitude* dari vektor *retrieved response* yang didapatkan akan mendekati angka tersebut. Berdasarkan nilai *magnitude* dari

vektor *retrieved response* yang didapat, pengukuran yang disebut *Mean Normalized Confidence* (MNC) dihitung sebagai berikut ini:

$$MNC = \frac{\sum_{i}^{m} \gamma_i^{T(\mu)}}{\sum_{i}^{m} \gamma_i^{\mu}} \tag{2.11}$$

dimana, m menyatakan jumlah elemen dalam *response pattern*, dan μ menyatakan nomor urut *pattern* tersebut. Kemudian, $\gamma_i^{T(\mu)}$ dan γ_i^{μ} secara berurutan menyatakan *magnitude* elemen ke-*i* pada *retrieved response pattern* dan *original response pattern*. Karena *magnitude* pada *original response pattern* adalah satu, maka persamaan 2.11 dapat disederhanakan menjadi berikut ini:

$$MNC = \frac{\sum_{i}^{m} \gamma_i^{T(\mu)}}{m}$$

MNC akan bernilai mendekati satu, jika sebuah *query pattern* yang diberikan seutuhnya cocok dengan *pattern* yang di-*encode*.

2.12 Kompleksitas waktu algoritma

Beberapa parameter menentukan kompleksitas waktu *holographic memory* dengan proses training. P menunjukkan banyaknya gambar, n menunjukkan banyaknya elemen *stimulus pattern*, dan m menunjukkan banyaknya elemen *response pattern* dimana: $m = \log p$

Untuk setiap gambar, kompleksitas pemetaan *stimulus pattern*-nya adalah $O(n)$, untuk *response pattern* $O(m)=O(\log p)$, dan kompleksitas asosiasi *stimulus-response pattern* adalah $O(mn) = O(p \log p)$.

Kompleksitas waktu algoritma *encoding* dengan proses training untuk p gambar:

$$O(p)\,O(n) + O(n \log p) + O(p \log p) \approx O(pn \log p)$$

Kompleksitas waktu algoritma *decoding* dengan proses training :

$$O(n) + O(n \log p) + O(p \log p) \approx O(n \log p)$$