2. TEORI PENUNJANG

1. COMPUTER VISION DAN IMAGE PROCESSING

Computer Vision mempunyai tujuan utama untuk membuat keputusan yang berguna tentang obyek fisik nyata dan pemandangan (*Scenes*) berdasarkan *image* yang didapat dari sensor. *Computer vision* ingin membangun sebuah mesin pandai yang dapat melihat. Tentunya hal ini bukanlah hal mustahil. Ada berbagai contoh dari aplikasi *computer vision* seperti *Human Computer Interaction* (HCI), *Object Identification, Segmenatation,* dan *Recognition.*

Dalam pengambilan keputusan tentang obyek nyata, ternyata selalu dibutuhkan untuk membangun beberapa deskripsi atau model dari obyek obyek tersebut dari gambar yang ada. Karena ini, banyak ahli akan berkata bahwa tujuan dari *computer vision* adalah konstruksi dari deskripsi pemandangan yang diambil dari gambar yang diperoleh.

Pengertian sederhana dari *image processing* adalah manipulasi dan analisis suatu informasi gambar oleh komputer. Yang dimaksud dengan informasi gambar disini adalah gambar *visual* dalam dua dimensi. Segala operasi untuk memperbaiki, analisa atau pengubahan suatu gambar disebut *image processing*. Konsep dasar dari sistim *image processing* diambil dari kemampuan indera penglihatan manusia yang selanjutnya dihubungkan dengan kemampuan otak manusia. Dalam sejarahnya, *image processing* telah diaplikasikan dalam berbagai bentuk, dengan tingkat kesuksesan yang cukup besar. Seperti berbagai cabang ilmu lainnya, *image processing* menyangkut pula berbagai gabungan cabang–cabang ilmu. Seperti diantaranya optik, elektronik, matematika, fotografi, dan teknologi komputer.

Berbagai bidang telah banyak menggunakan aplikasi dari *image processing* baik dibidang komersil, industri dan medik. Bahkan bidang militer telah menggunakan perkembangan dunia *digital image processing* ini.

Pada umumnya, obyektif dari *image processing* adalah mentransformasikan atau menganalisa suatu gambar sehingga informasi baru tentang gambar dibuat lebih jelas. Ada banyak cara yang dapat diaplikasikan dalam suatu operasi *image processing*.

2. OPEN SOURCE COMPUTER VISION

Open Source Computer Vision Library mulai dikembangkan tiga tahun yang lalu oleh *Visual Interactivity Group* didalam *Intel's Microprocessor Research Lab.* Proyek ini dibuat dengan tujuan untuk mendirikan sebuah komunitas *Open Source Vision* dan menyediakan sebuah situs dimana usaha terdistribusi dari komunitas dapat dikonsolidasi dan *performance*-nya dapat dioptimalkan. *Library* ini ditunjukkan untuk digunakan oleh peneliti dan pengembang *software* komersial. Keunggulan *library* ini adalah semua fungsi–fungsinya telah dapat dioptimasi untuk prosessor Intel sehingga dapat berjalan jauh lebih cepat.

Open Source Computer Vision Libraray Committee terdiri dari beberapa orang diantaranya Dr. Gary Bradski, Prof. Trevor Darrell, Prof. Irfan Essa, Prof. Jitendra Malik, Prof. Pietro Perona, Prof. Stan Sclaroff dan Prof. Carlo Tomasi. Berikut adalah beberapa are dari fungsi umum yang dapat didukung oleh *Open Source Computer Vision Libraray Committee* :

- *Geometric Methods*
- *Recognition*
- *Measures*
- *Segmentation*
- *Utilities*
- *Features*

- *Image Pyramids*
- *Camera*
- *Tracking*
- *Fitting*
- *Matrix*
- *Image Processing.*

Libraray Open Source Computer Vision dalam merepresentasikan suatu gambar (*image*) dalam format IplIMage dari *Intel® Image Processing Libraray* (IPL). Struktur ini hanya dapat mendukung tingkat *depth* sebagai berikut :

- IPL_DEPTH_8U – *unsigned 8-bit intereger value (Unsigned char)*
- IPL_DEPTH_8S – S*igned 8-bit intereger value (Signed char* atau *simply char)*
- IPL_DEPTH_16S –S*igned 16-bit intereger value (Short int)*
- IPL_DEPTH_32S – S*igned 32-bit intereger value (int)*
- IPL_DEPTH_32F – 32 *bitfloating point single precision value (float)*

```
typedef struct _IplImage {
    int nSize; /* size of iplImage struct */
    int ID; /* image header version */
    int nChannels;
    int alphaChannel;
    int depth; /* pixel depth in bits */
    char colorModel[4];
    char channelSeq[4];
    int dataOrder;
    int origin;
    int align; /* 4- or 8-byte align */
    int width;
    int height;
    struct _IplROI *roi; /* pointer to ROI if any */
    struct _IplImage *maskROI; /*pointer to mask ROI if any */
    void *imageId; /* use of the application */
   struct _IplTileInfo *tileInfo; /* contains information on tiling

    int imageSize; /* useful size in bytes */
    char *imageData; /* pointer to aligned image */
    int widthStep; /* size of aligned line in bytes */
    int BorderMode[4]; /* the top, bottom, left,
    and right border mode */
    int BorderConst[4]; /* constants for the top, bottom,
                           left, and right border */
    char *imageDataOrigin; /* ptr to full, nonaligned image */
} IplImage;
```

GAMBAR 2.1

STRUKTUR IplImage

Open Source Computer Vision Library dibuat berdasarkan fungsi fungsi dasar dari *Intel® Image Processing Libraray* (IPL) ini. *Intel® Image Processing Libraray* (IPL) menyediakan sekumpulan fungsi–fungsi C yang sangat teroptimasi yang mengimplementasikan fungsi–fungsi *image Processing* pada prosessor berarsitektur Intel.

3. IMAGE PROCESSING LIBRARY

Intel® Image Processing Libraray (IPL) adalah suatu *library* yang menyediakan sekumpulan fungsi–fungsi *low-level* untuk memanipulasi gambar dalam strandart DLL. Fungsi–fungsi tersebut telah dioptimasi untuk

prosessor – prosessor dengan arsitektur *Intel®*, dan sangat efektif dikarenakan menggunakan keunggulan dari teknologi MMX™, Streaming SIMD Exstention (SSE) dan SSE-2. Hingga kini, versi–versinya telah dikembangkan untuk prosessor Intel486™ kompatible, prosessor Pentium®, prosessor Pentium Pro, prosessor Pentium dengan teknologi MMX™, prosessor Pentium II, Prosessor Pentium III dan yang terbaru prosessor Pentium 4. Untuk masing–masing prosesor telah disediakan DLL yang berbeda–beda.

Library ini terdiri dari fungsi–fungsi untuk melakukan proses *filtering*, *thresholding*, *transformasi* (FFT, DCT, geometri) serta untuk operasi-operasi aritmatika dan morfologi juga disediakan. *Library* ini menggunakan format gambar yang fleksible, mendukung gambar dengan 1, 8, dan 16 *channels* dan pixel *integer* atau *floating-point* 32 bit, dimana setiap gambar dapat memiliki sejumlah *channels* yang berbeda. Konversi dari dan ke format gambar windows DIB (*device independent bitmap*), konversi antara gambar berwarna dan *gray-scale* juga disediakan.

Library ini dapat dipakai di beberapa software pemograman umum seperti Borland Delphi, Borland C++, Microsoft Visual Basic, dan Microsoft Visual C++. Untuk saat ini versi yang terbaru dari *Intel® Image Processing Libraray* (IPL) adalah versi 2.5. Salah satu hal yang disayangkan dari *Library* ini adalah sifatnya yang tidak *open source*, sehingga pata pengembang-pengembang *software* agak terbatas jika ingin melakukan perubahan–perubahan pada fungsi *library* ini.

4. MICROSOFT® DIRECTSHOW®

Pekerjaan yang berhubungan dengan *multimedia* memberikan beberapa tantangan utama :

- *Multimedia Stream* mempunyai data dalam jumlah yang banyak yang harus diproses dengan cepat.

- *Audio, Video,* dan *stream* lainnya harus disinkronisasi untuk memulai dan berhenti pada waktu yang sama dan dimainkan pada *rate* yang sama.

- *Stream* dapat berasal dari berbagai sumber termasuk *local media files, network* komputer, *terrestrial broadcasts, video cameras* dan *device media* lainnya.

- *Stream* berasal dari berbagai macam format seperti *Audio-Video Intereaved* (AVI), *Advanced Streaming Format* (ASF), *Motion Picture Experts Group* (MPEG), *Digital Video* (DV) dan *Motion JPG* (MJPEG).

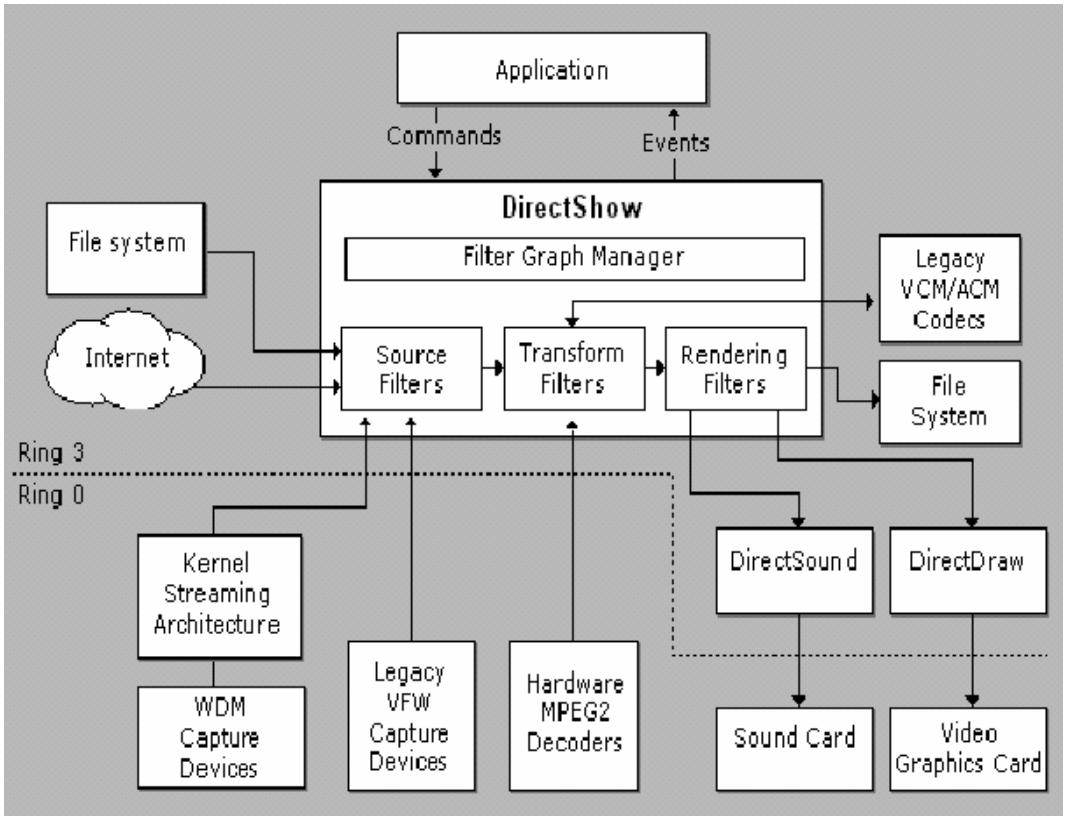
- Sebuah aplikasi tidak mempunyai cara untuk mengetahui *hardware device* yang mana yang disajikan pada *end-user's system*.

Direct Show didesain khusus untuk mengatasi setiap tantangan diatas. Tujuan utama dari desain adalah untuk menyederhanakan tugas dari pembuatan *multimedia* aplikasi pada Windows® *platform* dengan mengisolasi aplikasi dari kompleksitas data transport, perbedaan *hardwear* dan masalah sinkronisasi.

Microsoft® DirectShow® adalah sebuah arsitektur untuk mengalirkan (*Streaming*) pada *platform* Windows®. Direct Show menyediakan *capture* dengan kualitas tinggi dan *playback* dari *multimedia stream*. DirectShow juga mendukung berbagai macam format termasuk *Audio-Video Intereaved* (AVI), *Advanced Streaming Format* (ASF), *Motion Picture Experts Group* (MPEG), MPEG *Audio Layer-3* (MP3), dan WAV files. DirectShow mendukung pula *capture* menggunakan *Windows Driver model* (WDM) *device* atau *device Video for window* yang lebih tua. DirectShow terintegrasi dengan teknologi DirectX yang lainnya.

DirectShow menyederhanakan *playback media*, melakukan format konversi dan tugas–tugas *capture*. Pada saat yang sama, DirectShow juga menyediakan akses ke aplikasi yang membutuhkan solusi yang diinginkan (*custom solution*). DirectShow dibuat berdasarkan *Component Object Model* (COM).

Untuk mencapi hasil yang diinginkan dalam *Streaming Video* dan *audio*, DirectShow menggunakan DirectDraw dan DirectSound untuk me-*render* data secara efisien ke sistem suara dan *graphic card*. Sinkronisasi dicapai dengan men-*encapsulate* multimedia data dalam *time-stamped media samples*. Untuk menangani berbagai sumber, format dan *hardware device*, DirectShow menggunakan sebuah arsitektur modular yang menggunakan sistem komponen disebut *filter* yang dapat dicampur dan dicocokkan (*mixed and matched*) unutk menyediakan dukungan untuk berbagai skenario yang berbeda.



Gambar 2.2

Blok Diagram DirectShow

DirectShow memungkinkan aplikasi untuk memainkan *file* dan *stream* dari berbagai sumber termasuk lokal file, local CD, dan DVD drives, *remote file* pada *network*, TV-Turner yang terbaru dan Video Capture card berdasarkan pada *Window Driver Model* (WDM), dan *Legacy Video For Windows® video Capture Cards*. DirectShow mempunyai *native compressor* dan *decompressor* untuk beberapa file format dan banyak *third-party hardware* dan *software*

decoders yang komptible dengan DirectShow. *Playback* menggunakan secara penuh DirectDraw *hardware acceleration* dan kemampuan DirectSound ketika hardware mendukungnya.

5. PRINCIPAL COMPONENT ANALYSIS

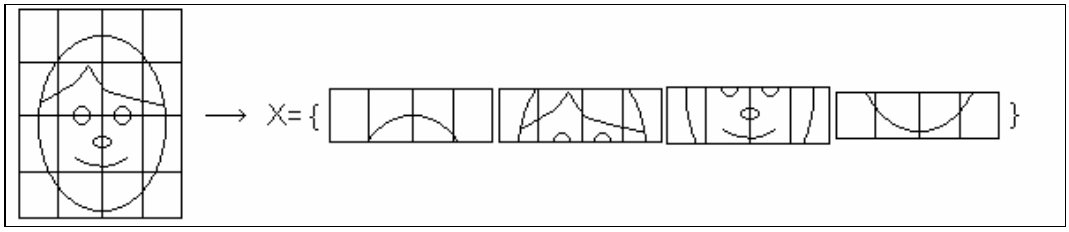
Principal Component Analysis (PCA) digunakan untuk mereduksi dimensi dari sekumpulan atau ruang gambar sehingga basis atau sistem koordinat yang baru dapat menggambar model yang khas dari kumpulan tersebut dengan lebih baik. Dalam hal ini model yang diinginkan merupakan sekumpulan wajah-wajah yang dilatihkan (*training facenumerik*). Fiture yang baru tersebut akan dibentuk melaluui kombinasi linear. Komponen dari fiture ruang karakter ini tidak akan saling berkolerasi dan akan memaksimalkan perbedaan yang ada di dalam variable aslinya.

Sasaran dari *Principal Component Analysis* adalah untuk menangkap variasi total didalam kumpulan karakter yang dilatihkan dan untuk menjelaskan variasi ini dengan sedikit variable. Suatu observasi yang dijabarkan dengan variable yang sedikit akan lebih mudah untuk ditangani dan dimengerti daripada jika dijabarkan dengan variable yang banyak. Hal ini tidak hanya berarti mengurangi kompleksitas dan waktu komputasi tetapi juga mengatur skala tiap variable sesuai dengan keudukan dan kepetingan relatifnya didalam menjabarkan observasi tersebut.

Apabila didefiniskan sebuah obyek U = {u1, u2, u3, ..., un} sebagai vektor pada n dimensi. Obyek U (Gambar 2.3) dapat berupa suatu gambar dan mempunyai komponen u1, u2, ..., un dimana u1, u2, ..., un adalah nilai pixel dari gambar tersebut. Dengan kondisi ini maka n dapat diartikan sebagai panjang * lebar jumlah pixel yang terdapat dalam gambar. Kemudian apabila objek tersebut ditambah dengan objek–objek yang lain hingga menjadi sekumpulan atau sekelompok objek maka :

$U^i = \{u^i_1, u^i_2, ..., u^i_n)$ dimana i=1, ...,m dan m<<n. Nilai rata rata atau *mean* object (gambar 2.3) : $U = \{ U_1, U_2, ..., U_n\}$ dapat dideskripsikan sebagai berikut :

$$u_i = \frac{1}{m}\sum_{k=1}^{m} u_i^k \qquad (2.1)$$

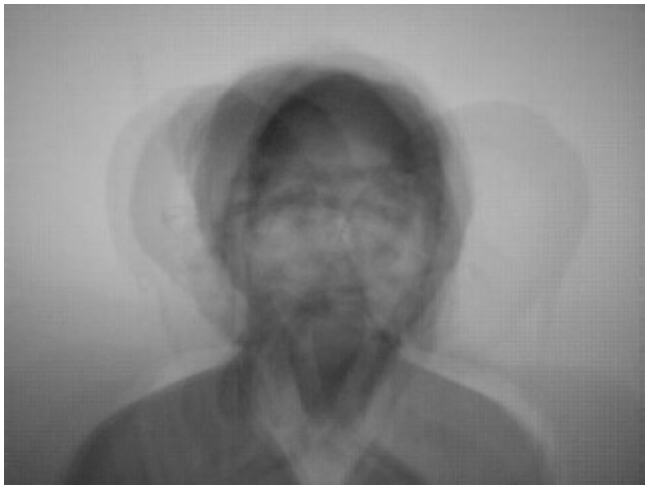


Gambar 2.3

Variasi Obyek dengan menggunakan wajah seseorang

Covariance Matrix $C = [\ C_{ij}\ ]$ adalah sebuah matrix simetris $m \times m$, dimana elemen umumnya mempunyai persamaan:

$$C_{ij} = \sum_{l=1}^{n}\left(u_l^i - \bar{u}_l\right)\left(u_l^j - \bar{u}_l\right) \qquad (2.2)$$



Gambar 2.4

Averange Object dari obyek U

Fitur *Eigen Object* $u^i = \{u^i_1, u^i_2, ..., u^i_n)$, i=1, ..., m_1=m (gambar 2.5) dari *input* grup obyek. Dimana ?i dan $v^i = \{v^i_1, v^i_2, ..., v^i_n)$ adalah *eigen value* dan *eigen vector* dari matrik c. Maka dapat dihitung berikut:

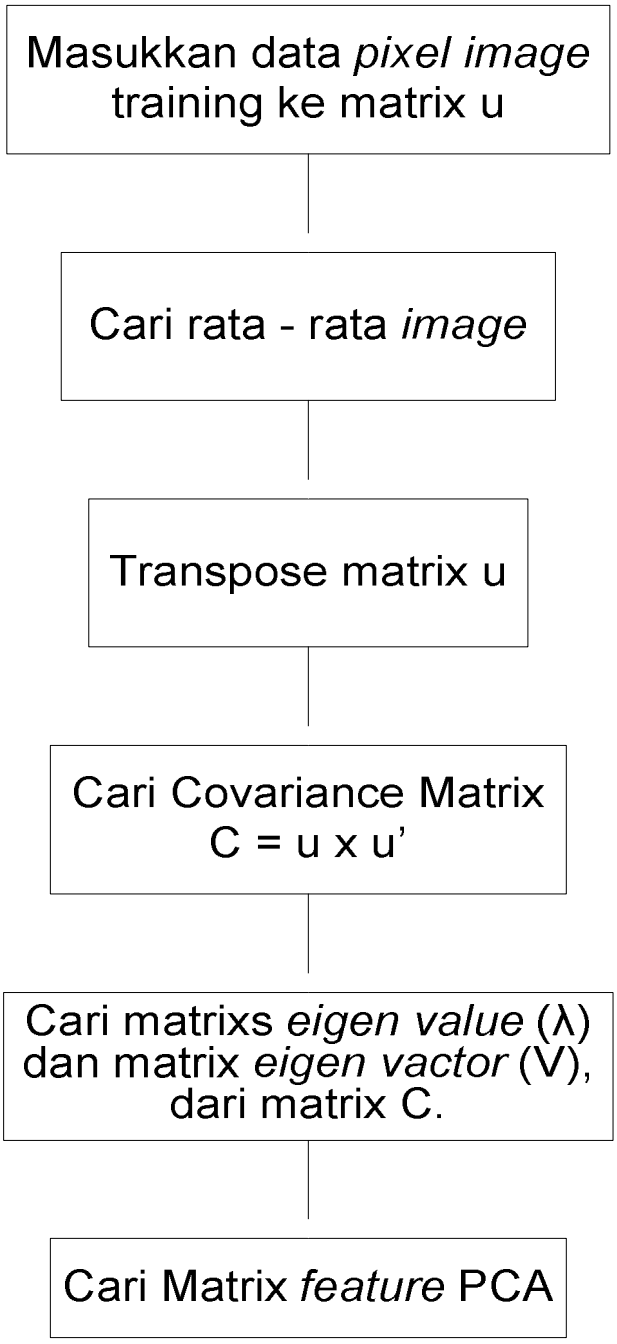
$$e_l^i \equiv \frac{1}{\sqrt{?_i}} \sum_{k=1}^{n} v_k^i . \left(u_l^k - \bar{u}_l \right) \qquad (2.3)$$

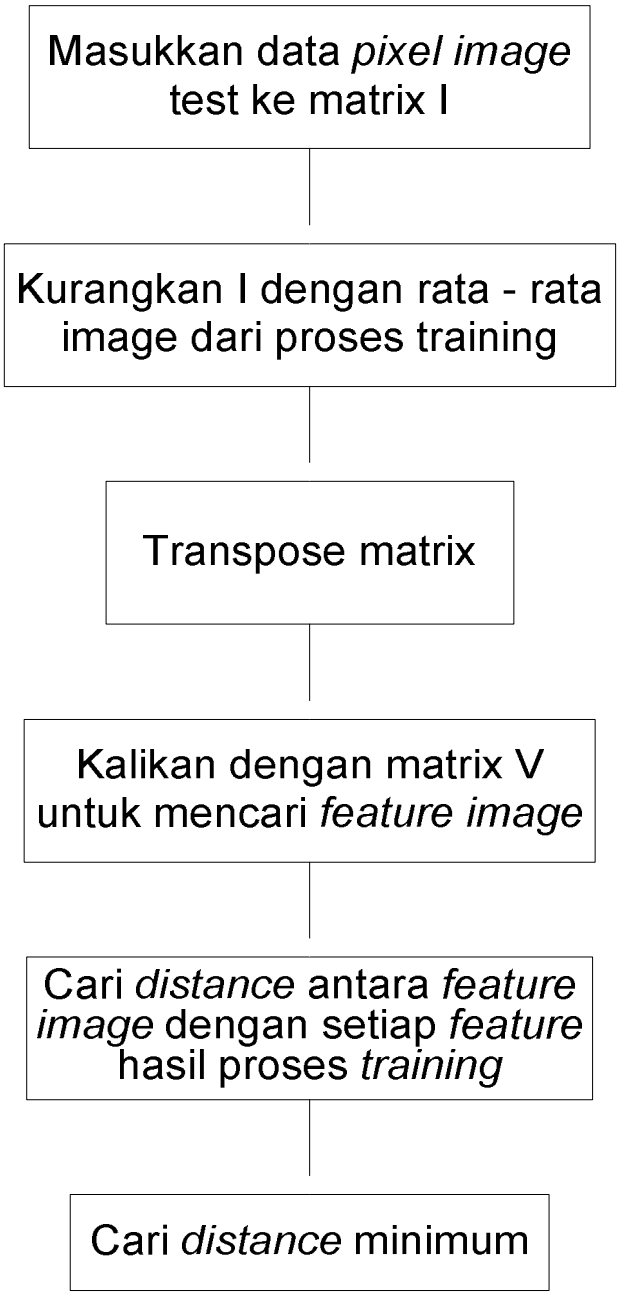
Feature adalah komponen-komponen penting dari image–image yang training yang didapatkan dari hasil proses training. Feature inilah yang akan digunakan untuk mengidentifikasikan suatu image yang akan dikenali.

Gambar 2.5
Eigen Space dari obyek U

Semua *input* obyek sebagaimana juga obyek lainnya dari u dapat dikomposisikan di dalam objek *eigen* m1-D.

$$w_i = \sum_{l=1}^{n} e_i^i . \left(u_l - \bar{u}_l \right) \qquad (2.4)$$


Masukkan data *pixel image* training ke matrix u
Cari rata - rata *image*
Transpose matrix u
Cari Covariance Matrix C = u x u'
Cari matrixs *eigen value* (λ) dan matrix *eigen vactor* (V), dari matrix C.
Cari Matrix *feature* PCA


Masukkan data *pixel image* test ke matrix I
Kurangkan I dengan rata - rata image dari proses training
Transpose matrix
Kalikan dengan matrix V untuk mencari *feature image*
Cari *distance* antara *feature image* dengan setiap *feature* hasil proses *training*
Cari *distance* minimum

Pengenalan suatu image menggunakan PCA memerlukan operasi dari 3 buah matrix yaitu Matrix rata–rata, Covariance matrix, Feature PCA yang sudah disimpan pada saat training. Kemudian image yang akan dikenali (selanjutnya disebut A), selanjutnya diproyeksikan juga kedalam bidang multidimensi seperti diatas dengan metode PCA.

Setelah A diproyeksikan kemudian dicari *Distance*-nya yang paling minimum. Untuk menghitung *distance* ini tidak perlu menghitung sebanyak pixel yang ada pada image yang akan dikenali tersebut tetapi cukup hanya sebanyak feature yang telah dihitung diatas tadi, misalnya 30, 40 dan yang lainnya. Berdasarkan jumlah feature yang diambil, *recognition presentage* dari PCA akan berubah–ubah. Jumlah *feature* yang optimal akan didapatkan dari hasil ekperiment beberapa kali.

$$Distance = |f_x - f_{database}|$$

$$Distance = \sqrt{(f_{x1} - f_{database1})^2 + (f_{x2} - f_{database2})^2 + \ldots\ldots\ldots + (f_{xi} - f_{databasei})^2} \quad (2.5)$$

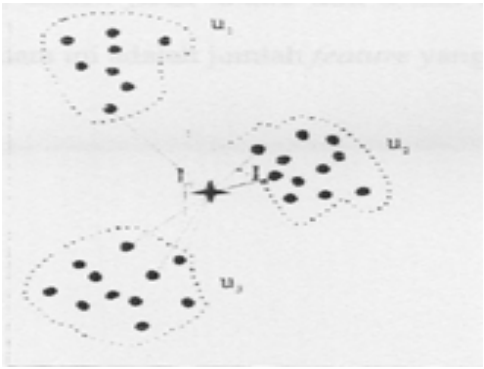
Setelah didapat semua distance, maka dicari yang paling minimum dan dilihat *distance* minimum tersebut bersesuaian dengan image ke berapa dan diambil nama orang dari image tersebut. Maka A akan dikenali dengan nama tersebut.

Sebelum mengenali wajah seseorang sebaiknya kita memanfaatkan suatu fasilitas yang bernama ROI (Region Of Intereset) untuk daerah muka saja. Dengan adanya ROI ini kita dapat memfokuskan pengenalan wajah seseorang dengan mengkalkulasikan daerah wajah saja dan tidak berpengaruh dengan daerah diluar wajah.

6. Metode Nearest K-Neighbor

Metode ini memiliki *recognition percentage* yang paling tinggi, namun perhitungannya cenderung rumit. Pada metode ini diambil k buah nilai

distance minimum. Dari k buah nilai itu dicari kelas mana yang paling banyak jumlahnya. Kelas inilah yang dipakai untuk mengindentifikasikan fiture yang akan dicari.



Gambar 2.8
Metode K-nearest Neighbor

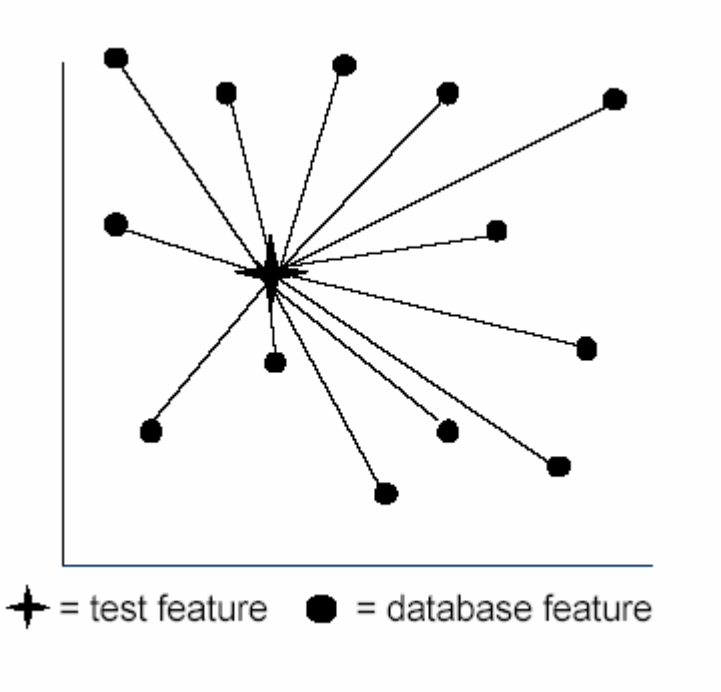
Pertimbangan dari metode ini, nilai *distance* minimum saja tidak cukup mengidentifikasi feature yang akan dicari, karena belum tentu nilai *distance* minimum itu merepresentasikan kelas dimana image dari fiture yang akan dicari tersebut sesungguhnya berada.

Pada gambar 2.8 dapat dilihat bahwa *distance* minimum dari l_x terletak pada ln. Tetapi setelah diambil sebanyak 6 buah *distance* minimum, ternyata kelas yang terbanyak dalam 6 nilai tersebut adalah u_3 bukan l_n.

Distance adalah jarak antara 2 titik dalam ruang multidimensi, persamaan untuk mengukur *distance* ini didapat dari penurunan teori *phytagoras* untuk n buah dimensi. Pada perhitungannya dengan mengkalkulasikan semua bobot nilai yang ada pada sekumpulan gambar training $W_d = \{W_d^1, W_d^2, ..., W_d^m)$ untuk masing masing data dengan menggunakan metode PCA pada tahap sebelumnya. Perhitungan juga dilakukan pada gambar yang akan ditest. $W_t = \{W_t^1, W_t^2, ..., W_t^m)$), tentunya dengan cara yang sama. Bobot jarak antara hasil *training* dengan tes kemudian dihitung dengan menggunakan persamaan *Euclidean Distance* (∂) pada m-dimensi, sehingga didapat:

$$\partial = \sqrt{\sum_{i=1}^{m} \left(w_d^i - w_t^i\right)^2} \qquad (2.6)$$

Dimana m adalah banyaknya jumlah feature bobot. Dari hasil ini kemudian dicari mana yang paling memiliki nilai terkecil diantara set jarak tersebut (gambar 2.9). Hasil ini dicari pada *database* yang mempunyai nilai pada posisi yang bersangkutan. Wajah pada posisi adalah hasil dari pengenalan tersebut.



Gambar 2.9

I – Nearest Neighbor Classifier