Bab 9

Kontur dan Representasinya

Pendeteksi tepi menghasilkan citra tepi yang berupa citra biner (*pixel* tepi berwarna putih, sedangkan *pixel* bukan-tepi berwarna hitam). Tetapi, tepi belum memberikan informasi yang berguna karena belum ada keterkaitan antara suatu tepi dengan tepi lainnya. Citra tepi ini harus diproses lebih lanjut untuk menghasilkan informasi yang lebih berguna yang dapat digunakan dalam mendeteksi bentuk-bentuk sederhana (misalnya garis lurus, lingkaran, elips, dan sebagainya) pada proses analisis citra.

Rangkaian *pixel-pixel* tepi yang membentuk batas daerah (*region boundary*) disebut **kontur** (*contour*) [JAI95]. Kontur dapat terbuka atau tertutup. Kontur tertutup berkoresponden dengan batas yang mengelililingi suatu daerah lihat Gambar 9.1(a). *Pixel-pixel* di dalam daerah dapat ditemukan dengan algoritma pengisian (*filling algorithm*). Batas daerah berguna untuk mendeskripsikan bentuk objek dalam tahap analisis citra (misalnya untuk mengenali objek).

Kontur terbuka dapat berupa fragmen garis atau bagian dari batas daerah yang tidak membentuk sirkuit (Gambar 9.1(b)).

(a) (b)

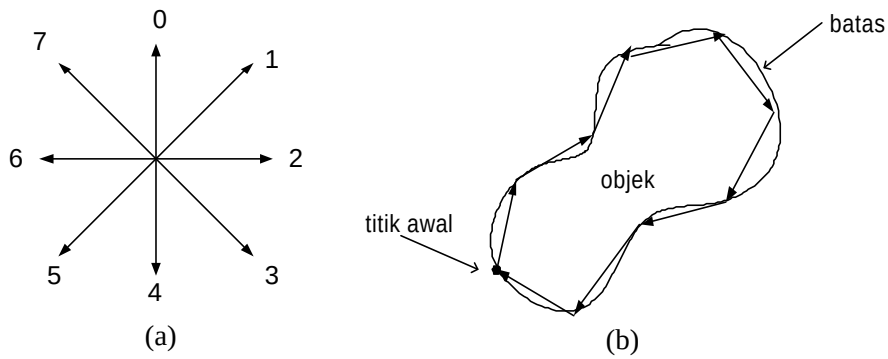
Gambar 9.1 *(a) kontur tertutup, (b) kontur terbuka*

9.1 Representasi Kontur

Representasi kontur dapat berupa senarai tepi (*edge list*) atau berupa kurva. Senarai tepi merupakan himpunan terurut *pixel-pixel* tepi. Representasi kontur ke dalam kurva merupakan representasi yang kompak dan mangkus untuk analisis citra. Misalnya, rangkaian *pixel* tepi yang membentuk garis dapat direpresentasikan hanya dengan sebuah persamaan garis lurus. Representasi semacam ini menyederhanakan perhitungan selanjutnya seperti arah dan panjang garis.

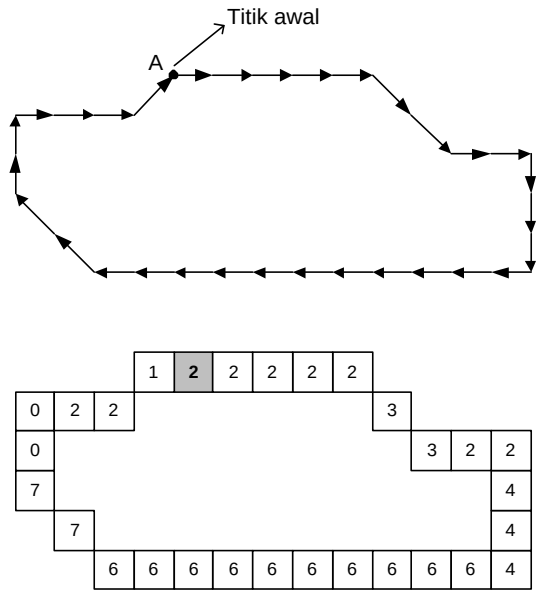
Kode Rantai

Kode rantai (*chain code*)adalah notasi untuk mengkodekan senarai tepi yang membentuk batas daerah. Kode rantai menspesifikasikan arah setiap *pixel* tepi di dalam senarai tepi. Arah yang digunakan adalah 8 arah mata angin seperti yang terlihat pada pada Gambar 9.2 (a).



Gambar 9.2 *(a) Kode rantai, (b) representasi batas objek dengan kode rantai.*

Dimulai dari sebuah *pixel* tepi dan searah jarum jam, arah setiap *pixel* tepi yang membentuk batas objek dikodekan dengan salah satu dari delapan kode rantai. Kode rantai merepresentasikan batas objek dengan koordinat *pixel* tepi pertama lalu diikuti dengan senarai kode rantai. Karena ada 8 arah, maka cukup 3 bit untuk mengkodekan setiap arah. Gambar 9.3 memperlihatkan contoh pengkodean batas objek dengan kode rantai.



Kode rantai: (A), 22222332244466666666667700221

Gambar 9.3 *Contoh pengkodean batas objek dengan kode rantai.*

Pencocokan Kurva

Kurva yang merepresentasikan kontur dicari dengan teknik pencocokan kurva (curve fitting). Ada dua macam teknik pencocakan kurva: **interpolasi** dan **penghampiran** (*approximation*). Interpolasi kurva adalah mencari kurva yang melalui semua *pixel* tepi, sedangkan penghampiran kurva adalah mencari kurva yang paling dekat melalui *pixel-pixel* tepi, tetapi tidak perlu melalui semua *pixel* tersebut.

Di dalam bab ini kita hanya membahas teknik pencocokan kurva dengan penghampiran. Salah satu metode penghampiran kurva yang populer dalam pengolahan citra adalah transformasi Hough. Transformasi Hough akan dibahas dalam upa-bab 9.3 di bawah ini.

9.2 Transformasi Hough

Transformasi Hough menspesifikasikan kurva dalam bentuk parametrik. Kurva dinyatakan sebagai bentuk parametrik

$$(x(u), y(u))$$

dari parameter u. Bentuk parametrik tersebut menspesifikasikan titik-titik sepanjang kurva dari titik awal kurva $p_1 = (x(u_1), y(u_1)$ ke titik akhir $p_2 = (x(u_2), y(u_1)$.

Panjang kurva adalah

$$L = \int_{u_1}^{u_2} \sqrt{\left(\frac{dx}{du}\right)^2 + \left(\frac{dy}{du}\right)^2}\, du \qquad (9.1)$$

Transformasi Hough menggunakan mekanisme *voting* untuk mengestimasi nilai parameter. Setiap titik di kurva menyumbang suara untuk beberapa kombinasi parameter. Parameter yang memperoleh suara terbanyak terpilih sebagai pemenang.

Pada awalnya, Transformasi Hough digunakan untuk mendeteksi garis lurus. Namun, ia juga dapat digunakan untuk mendeteksi kurva sederhana lainnya seperti lingkaran dana elips. Pembahasan dimulai dengan transformasi Hough untuk mendeteksi keberadaan garis lurus di dalam citra tepi.

Mendeteksi Garis Lurus

Misalkan citra tepi berukuran $n = N \times M$ *pixel*. Cara yang paling sederhana mendeteksi garis lurus adalah menemukan semua garis yang ditentukan oleh dua buah *pixel* dan memeriksa apakah sebagian dari *pixel* tepi termasuk ke dalam garis tersebut (cara *exhaustive search*).

Jumlah maksimum garis yang dideteksi adalah $n\ (n - 1)/2$. Karena setiap *pixel* harus diperiksa apakah ia termasuk ke dalam suatu garis, maka kompleksitas algoritma pendeteksian garis lurus untuk kasus terburuk adalah $O(n^3)$. Untuk aplikasi praktis, jelas metode pendeteksian dengan cara ini tidak mangkus.

Transformasi Hough mengurangi kompleksitas komputasi dengan menggunakan bentuk parametrik dan menggunakan mekanisme pemungutan suara terbanyak (*voting*) untuk menentukan nilai parameter yang tepat.

Tinjau persamaan garis lurus:

$$y = mx + c \qquad (9.2)$$

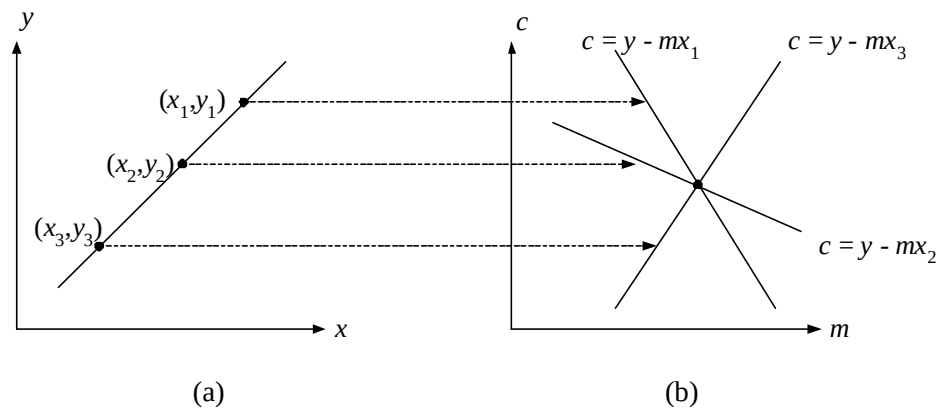
Dalam bentuk parametrik, setiap garis dinyatakan sebagai (m', c') di dalam ruang parameter m-c. Persamaan 9.2 dapat ditulis menjadi

$$c = y - mx \qquad (9.3)$$

Sembarang titik (x,y) pada bidang planar X-Y berkoresponden dengan sebuah garis lurus pada ruang parameter m–c.

Tinjau 3 buah titik pada sebuah garis lurus pada Gambar 9.4(a). Sembarang garis yang melalui titik (x_1,y_1) berkoresponden dengan garis $c = y_1 - mx_1$ pada ruang parameter *m-c*. Begitu juga, sembarang garis lurus yang melalui (x_2,y_2) berkoresponden dengan garis $c = y_2 - mx_2$ dan sembarang garis lurus yang melalui (x_3,y_3) berkoresponden dengan garis $c = y_3 - mx_3$ pada ruang *m-c*. Perpotongan (*m'*,*c'*) dari ketiga garis pada ruang *m-c* tersebut menentukan garis unik yang melalui (x_i,y_i), i = 1, 2, 3, di bidang *X-Y*.

Dengan cara ini, maka setiap *pixel* pada garis lurus di bidang citra berkoresponden dengan sejumlah garis lurus yang melalui **satu** ititik tertentu di ruang parameter *m-c*. Sifat ini dimanfaatkan untuk mendeteksi garis lurus. Jika setiap *pixel* tepi melakukan "pemungutan suara" pada ruang parameter, maka keberadaan garis lurus pada citra ditandai dengan penumpukan suara pada tempat-tempat tertentu di ruang parameter.



Gambar 9.4 *(a) Garis lurus pada ruang X-Y; (b) representasinya dalam ruang parameter m-c.*

Karena itu, prosedur mendeteksi garis lurus adalah sebagai berikut:

1. Ruang parameter didiskritkan sebagai matriks $P(m, c)$, yang dalam hal ini $m_1 \le m \le m_K$ dan $c_1 \le c \le c_L$.
2. Tiap elemen pada ruang parameter diasumsikan sebagai akumulator. Inisialisasi setiap elemen $P(m, c)$ dengan 0.
3. Untuk setiap *pixel* tepi (x_i, y_i) –*pixel* tepi dicirikan mempunyai nilai intensitas putih (1) dalam skala 0 - 1)– hitung nilai $c = y_i - mx_i$. Untuk setiap nilai parameter m, $m_1 \le m \le m_K$, yang berkoresponden dengan nilai c, maka elemen matriks $P(m, c)$ yang bersesesuaian dinaikkan satu:

   $$P(m, c) = P(m, c) + 1 \qquad (9.4)$$

   Dengan kata lain, tambahkan satu suara pada ruang parameter *m-c*.

4. Ulangi langkah 3 sampai seluruh *pixel* di dalam citra tepi ditelusuri.
5. Pada akhir prosedur, tiap elemen matriks $P(m, c)$ menyatakan jumlah *pixel* tepi yang memenuhi persamaan (1). Tentukan elemen matriks yang memiliki penumpukan suara cukup besar (yang nilainya di atas nilai ambang tertentu). Misalkan tempat-tempat itu adalah

   $\{(m_1, c_1), (m_2, c_2), \ldots, \{(m_k, c_k),$

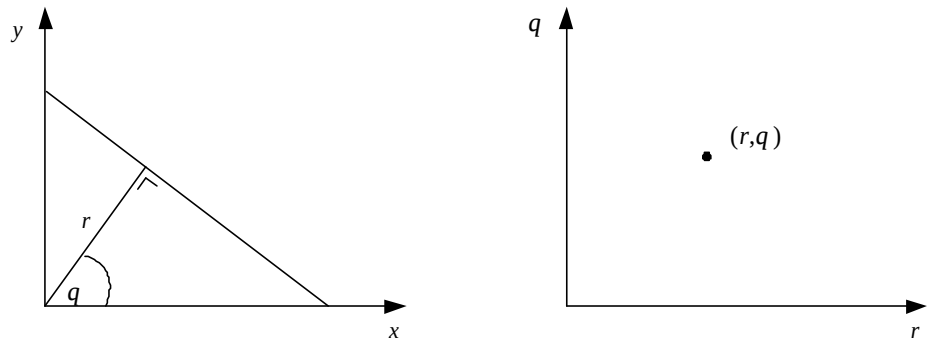
   Hal ini berarti terdapat k garis lurus yang terdeteksi pada citra.

Tingkat ketelitian dari Transformasi Hough bergantung pada ukuran matriks $P(m, c)$, yaitu $K \times L$. Kompleksitas komputasi Transformasi Hough pada kasus terburuk adalah $O(Kn)$, yang dalam hal ini K adalah jumlah pembagian parameter m, dan n adalah jumlah *pixel* di dalam citra tepi. Karena $O(Kn) < O(n^3)$, maka pendeteksian garis lurus dengan Transformasi Hough lebih cepat daripada metode *exhaustive search*.

Model parametrik pada persamaan 9.2 tidak dapat digunakan untuk mendeteksi garis vertikal atau hampir vertikal karena gradiennya (m) menuju nilai tak-berhingga. Karena itu, garis dinyatakan dalam representasi polar:

$$r = x \cos q + y \sin q \qquad (9.4)$$

yang dalam hal ini r adalah jarak garis ke titik asal (Gambar 9.5).



Gambar 9.5 *Representasi polar dari garis lurus*

Sembarang garis yang melalui (x_1,y_1) pada ruang x–y berkoresponden dengan kurva sinusoida $r = x_1 \cos q + y_1 \sin q$ pada ruang r–q. *Pixel-pixel* yang terletak segaris pada citra tepi berkoresponden dengan titik potong seluruh kurva sinusoidanya pada ruang parameter r–q.

Prosedur yang sama untuk mendeteksi garis lurus dapat digunakan kembali dengan mengganti ruang parameter *m–c* menjadi ruang parameter *r–q*, yang dalam hal ini,

$$-\sqrt{N^2+M^2} \le r \le \sqrt{N^2+M^2}$$
$$-p/2 \le \theta \le p/2$$

Algoritma Transformasi Hough diperlihatkan pada Algoritma 9.1 [PIT93]. Algoritma tersebut mengasumsikan citra tepi (citra hasil pendeteksian tepi) disimpan di dalam matriks `Edge[0..N-1,0..M-1]`. Ruang parameter *r–q* dinyatakan sebagai matriks `P` yang berukuran $n \times m$. Nilai *cosinus* dan *sinus* disimpan di dalam *lookup table* `COS[0..p-1]` dan `SIN[0..p-1]` yang dibentuk dengan Algoritma 9.2.

```
void Hough(citra Edge, int N, int M, imatriks P, int n, int m,
           float *COS, float *SIN)
/* prosedur yang melakukan Transformasi Hough.
   Masukan: citra tepi Edge yang berukuran  N x M.
   Keluaran: matriks parameter P yang berukuran n x m
*/
{
  int k, l, i, j, kk, ll;
  float r, b;
  float SQRTD =sqrt((float)N*(float)N + (float)M*(float)M);

  /* inisialisasi P[0..p-1, 0,,q-1] dengan 0 */
  for(kk=0;kk<=p-1;kk++)
    for(ll=0;ll<=q-1;ll++)
       P[kk][ll]=0;

  /*telusuri citra tepi. Jika pixel merupakan tepi, lakukan pemungutan
    suara pada elemen matriks P yang bersesuaian.
    tetha dari –pi/2 sampai pi/2.
    r dari –sqrt(N*N+M*M) sampai sqrt(N*N+M*M).
  */

  for (k=0;k<=N-1;k++)
    for (l=0;l<=M-1;l++)
    {
      if (Edge[k][l]==1)
      {
        for (i=0;i<=p-1;i++)
         {
           r = k*COS[i] + l*SIN[i];
           b = SQRTD;
           r+=b; r/=(SQRTD*2.0); r*=(m-1); r+=0.5;
           j=floor(r);
           P[i][j]++;
         }
      }
    }
}
```

Algoritma 9.1 *Transformasi Hough*

```
void LookUpTable(float *COS, *SIN, int m)
/* Membuat tabel cosinus dan sinus untuk fungsi COS dan SIN.
   Masukan: m adalah jumlah baris tabel
   Keluaran: tabel COS dan tabel SIN
*/
{
 int i;
 float th, R_TO_D = 0.017453

 for(i=0;i<=p-1;i++)
 {
   th = (float)i * 180.0/(m-1)-90.0;
   th = th * R_TO_D;
   COS[i] = (double) cos((double)th);
   SIN[i] = (double) sin((double)th);
 }
```

Algoritma 9.2 *Transformasi Hough*

Setelah Transformasi Hough selesai dilakukan, langkah berikutnya adalah melakukan operasi pengambangan (*thresholding*) untuk menentukan tempat-tempat pada ruang paramater *r*–*q* yang mempunyai penumpukan suara lebih besar dari nilai ambang *T*. Elemen matriks P yang nilainya di atas nilai ambang tersebut menyatakan parameter garis lurus. Misalkan tempat-tempat itu adalah {(m_1, c_1), (m_2, c_2), …, {(m_k, c_k), hal ini berarti terdapat *k* garis lurus yang terdeteksi pada citra. Algoritma pengambangan ditunjukkan pada Algoritma 9.3.

```
void threshold(imatriks P, int n, in m, int T)
/* Melakukan pengambangan pada matriks parameter P.
   Setiap elemen matriks P yang nilainya di atas T menyatakan parameter
   garis lurus.
   Masukan: matriks parameter P yang berukuran n x m.
   Keluaran: matriks parameter P yang sudah di-threshold.
*/
{ int i, j;

 for(i=0;i<n;i++)
   for(j=0;j<m;j++)
      if (P[i][j]>T)
        P[i][j]=1;
      else
        P[i][j=0;
}
```

Algoritma 9.3 *Pengambangan hasil transformasi Hough*

Pixel-pixel tepi yang termasuk di dalam garis lurus hasil deteksi Transformasi Hough dapat dihasilkan dengan algoritma **Transformasi Hough Balikan** (*inverse Hough Transform*). Untuk setiap elemen matriks Par yang bernilai 1,

garis lurus yang bersesuaian (yang intensitasnya 1) digambarkan pada matriks keluaran `Out`. Operasi `and` dengan citra tepi dilakukan untuk mengklarifikasi keberadaan *pixel* tepi.

Algoritma Transformasi Hough Balikan diperlihatkan pada Algoritma 9.4 [PIT93]. Algoritma tersebut mengasumsikan citra tepi (citra hasil pendeteksian tepi) disimpan di dalam matriks `Edge[0..N-1,0..M-1]`. Ruang parameter *r–q* dinyatakan sebagai matriks `P` yang berukuran $n \times n$. Citra keluaran yang berisi *pixel-pixel* tepi pembentuk garis lurus disimpan di dalam matriks `Out[0..N-1,0..M-1]`.

```
void InverseHough(citra Edge, citra Out, int N, int M, imatriks P,
                  int n, int m, float *COS, float *SIN)
/* prosedur yang melakukan Transformasi Hough Balikan
   Masukan: 1. citra tepi Edge yang berukuran  N x M.
            2. matriks parameter P yang berukuran n x m
   Keluaran: citra Out yang berisi pixel-pixel pembentuk garis lurus.
*/
{
  int k, l, i, j;
  float r, y;
  float SQRTD =sqrt((float)N*(float)N + (float)M*(float)M);
 /* inisialisasi citra keluaran dengan 0 */
  for(kk=0;kk<=p-1;kk++)
    for(ll=0;ll<=q-1;ll++)
       Out[p][q]=0;
  /* Matriks parameter P telah dilakukan operasi thresholding.
     Untuk setiap elemen P yang bernilai 1, garis lurus yang
     bersesuaian digambarkan ke dalam matriks Out.
      Operasi and dengan citra tepi dikerjakan.

     for (k=0;k<=p-1;k++)
       for (l=0;l<=q-1;l++)
        {
         y=(float)0.0;
         if (P[k][l]==1)  /* atau P[k][l]== 255 */
          {
           for (i=0;i<=N-1;i++)
            {
              r = (float)l * 2.0 * SQRTD/(m-1) – SQRTD;
              if (SIN[k]==(float)0.0)
                y++;
              else
                y=(r – (float)i * COS[k])/SIN[k];

              y+=0.5; j=floor(y);
              if (j >=0 && j < M)
                if (Edge[i][j]==1) Out[i][j]++;
            }
          }
        }
}
```

Algoritma 9.4 *Transformasi Hough Balikan*

Mendeteksi Lingkaran

Transformasi Hough dapat juga digunakan untuk mendeteksi bentuk lingkaran di dalam citra tepi. Persamaan lingkaran yang berpusat di titik (a, b) dengan jari-jari r adalah

$$(x-a)^2 + (y-b)^2 = r^2 \qquad (9.5)$$

Jadi, ruang parameter untuk lingkaran adalah r–a–b , sehingga matriks trimatra $P(r, a, b)$ dibutuhkan untuk menyimpan perhitungan suara.

Persamaan polar untuk setiap titik (x, y) di lingkaran:

$$x = a + r\cos q \qquad (9.6)$$
$$y = b + r\sin q \qquad (9.7)$$

Persamaan (9.6) dan (9.7) dapat ditulis menjadi persamaan

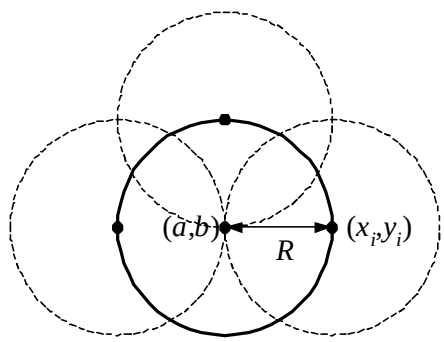
$$a = x - r\cos q \qquad (9.8)$$
$$b = y - r\sin q \qquad (9.9)$$

Pada operasi deteksi tepi, selain magnitudo *pixel* tepi, juga dihasilkan arah tepi, q. Karena itu, cos q dan sin q dapat dihitung.

Misalkan (x_i, y_i) adalah *pixel* tepi dan q adalah arah tepi . Ada dua kasus yang akan ditinjau:
(i) jika jari-jari lingkaran diketahui, (ii) jika jari-jari lingkaran tidak diketahui.

Kasus 1: Jari-jari lingkaran diketahui

Jika jari-jari jari-jari lingkaran diketahui $r = R$, maka ruang parametrik trimatra, $P(r,a,b)$, dapat direduksi menjadi ruang dwimatra, $P(a,b)$.



Gambar 9.5 *Proses penumpukan suara untuk mendeteksi lingkaran*

Titik pusat lingkaran, (a,b), yang mempunyai jari-jari $r = R$ dan melalui titik (x_i, y_i) dapat dihitung dengan persamaan

$$a = x_i - R\cos q \quad (9.10)$$
$$b = y_i - R\sin q \quad (9.11)$$

seperti yang ditunjukkan pada Gambar 9.5, lalu tambahkan elemen $P(a, b)$ yang bersesuaian dengan satu. Proses ini diulangi untuk *pixel-pixel* tepi yang lain. Elemen matriks $P(a, b)$ yang memiliki jumlah suara di atas nilai ambang tertentu menyatakan lingkaran yang terdapat di dalam citra tepi.

Kasus 2: Jari-jari lingkaran tidak diketahui

Jika jari-jari lingkaran tidak diketahui, maka penumpukan suara dilakukan untuk semua nilai r, $0 < r \le r_{max}$, nilai a dan b untuk *pixel* tepi (x_i, y_i) dihitung dengan persamaan

$$a = x_i - r\cos q \quad (9.12)$$
$$b = y_i - r\sin q \quad (9.13)$$

dan elemen $P(r, a, b)$ yang bersesuaian dinaikkan satu. Proses ini diulangi untuk *pixel-pixel* tepi yang lain. Elemen matriks $P(r, a, b)$ yang memiliki jumlah suara di atas nilai ambang tertentu menyatakan lingkaran yang terdapat di dalam citra tepi.

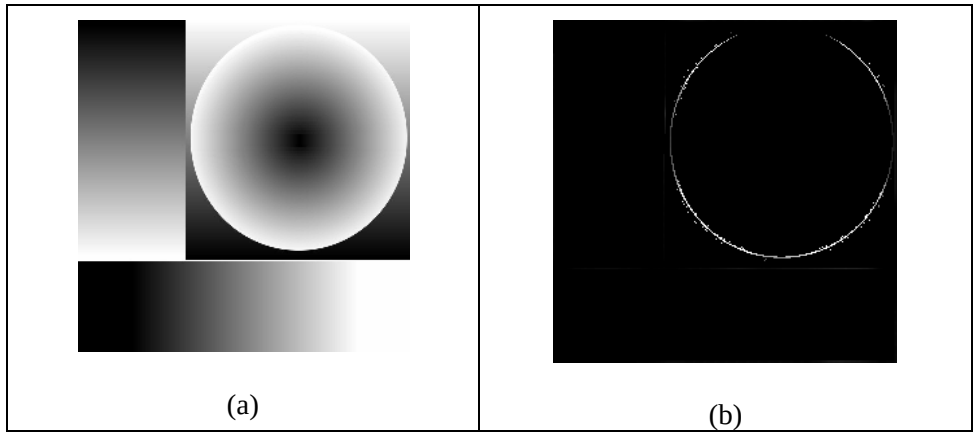
Persamaan (9.12) dan (9.13) dapat dimanipulasi dengan mengeliminasi r dari kedua persamaan:

$$a = x - r\cos q \;\rightarrow\; r = \frac{(x-a)}{\cos q}$$
$$b = y - r\sin q \;\rightarrow\; b = y - \frac{(x-a)}{\cos q}\sin q \;=\; y - (x-a)\tan q$$

$$b = a\tan q - x\tan q + y \quad (9.14)$$

Dengan demikian, maka ruang parametrik trimatra, $P(r,a,b)$, dapat direduksi menjadi ruang dwimatra, $P(a,b)$. Untuk untuk semua nilai r, yang dalam hal ini $a_1 < a \le a_K$, nilai ordinat b dari titik pusat lingkaran (a,b) yang melalui titik (x_i, y_i) dapat dihitung dengan persamaan (11), lalu tambahkan elemen $P(a, b)$ yang bersesuaian dengan satu. Proses ini diulangi untuk *pixel-pixel* tepi yang lain. Elemen matriks $P(a, b)$ yang memiliki jumlah suara di atas nilai ambang tertentu menyatakan lingkaran yang terdapat di dalam citra tepi.

Gambar 9.6 memperlihatkan hasil transformasi Hough untuk mendeteksi lingkaran dari citra *slope* dengan menggunakan nilai ambang $T = 30$.



Gambar 9.6 *(a) Citra slope, (b) hasil deteksi lingkaran dengan Transformasi Hough (Terima kasih kepada Danu Pranantha atas izin menggunakan output program tugasnya)*

Transformasi Hough untuk Mendeteksi Bentuk Sembarang

Transformasi Hough dapat dirampatkan untuk mendeteksi sembarang kurva yang berbentuk $f(\mathbf{x}, \mathbf{a}) = 0$, yang dalam hal ini **x** adalah vektor peubah dan **a** adalah vektor parameter. Memori yang dibutuhkan untuk matriks parametrik $P(\mathbf{a})$ meningkat menjadi K^q, yang dalam hal ini q adalah jumlah parameter. Tahapan yang dilakukan adalah [DUL97]:

1. tentukan lokasi pusat penumpukan suara;
2. tentukan fungsi jarak dari setiap *pixel* tepi ke pusat pemungutan suara.

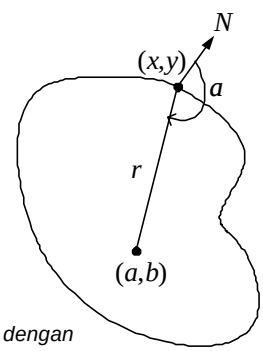
Jika kurva berbentuk lingkaran, maka lokasi pusat penumpukan suara adalah titik pusat lingkaran, sedangkan fungsi jarak dari setiap *pixel* tepi ke titik pusat lingkaran adalah fungsi konstan (yaitu akar pangkat dua dari persamaan (4)).

Sebagai contoh, pada Gambar 9.7 titik (a, b) adalah lokasi pusat penumpukan suara. Fungsi jarak r dari setiap titik (x, y) dan nilai α merupakan fungsi dari arah vektor normal N.

Untuk setiap *pixel* tepi (x,y) dengan sudut arah tepi q, lokasi pusat penumpukan suara dihitung dengan rumus

$$a = x - r(q)\cos(a(q)) \qquad (9.15)$$

$$b = y - r(q)\sin(a(q)) \qquad (9.16)$$



Gambar 9.7 *Pendeteksian bentuk kurva sembarang dengan Transformasi Hough rampatan*