

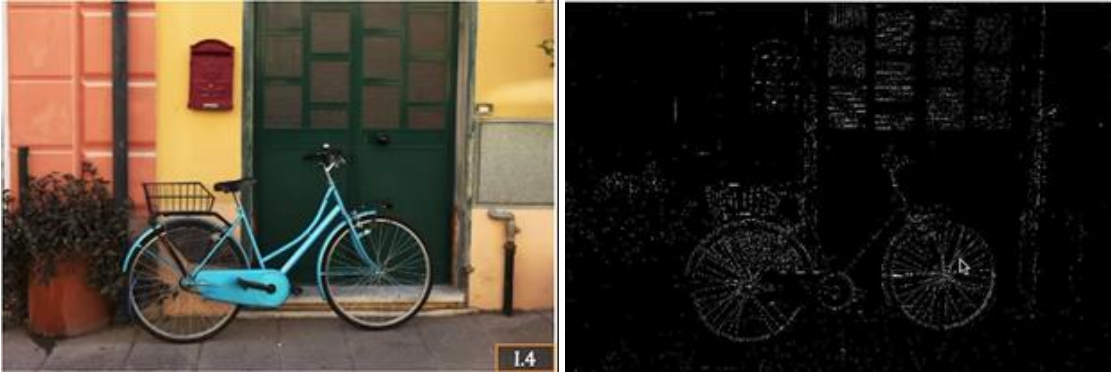
GÖRÜNTÜ İŞLEME-11.HAFTA

HOUGH DÖNÜŞÜM ALGORİTMASI (Doğru, Daire Elipslerin vb tespiti)

İçindekiler

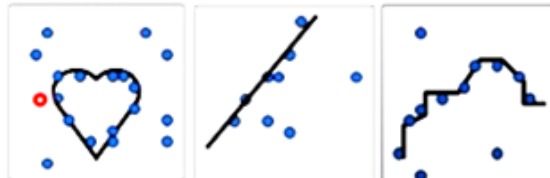
GÖRÜNTÜ İŞLEME-11.HAFTA	1
HOUGH DÖNÜŞÜM ALGORİTMASI (Doğru, Daire Elipslerin vb tespiti)	1
KARTEZYEN KOORDİNATLARDA BİR DOĞRUNUN TESPİTİ	2
Sayısal Örnek:	3
DOĞRUNUN DENKLEMİNİ POLAR KOORDİNATLARDA OLUŞTURMA	4
Doğrunun Tespiti Programı	11
HOUGH TRANSFORM: CIRCLE DETECTION	21
Using Gradient Information.....	24
Circle Detection: R Is Not Known.....	25
Referanslar.....	26

Aşağıdaki bir resim üzerinde bisikletin tekerini oluşturan daireyi oluşturan formülasyonu bulmak istiyoruz. Yani noktalardan oluşan bir resmin matematiğini elde etmek istiyoruz. Bunu başarabilirsek artık o bilgiyi bir çok yerde kullanabiliriz. Artık dairenin alanını, çapını, çevresini bulabiliriz. Üzerinde eksik noktalar varsa onları tamamlayıp yeniden çizebiliriz gibi onlarca amaç için kullanabiliriz.



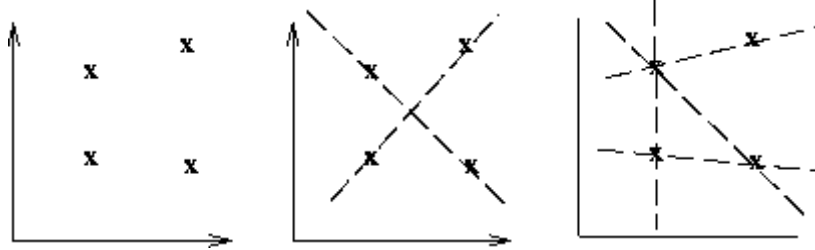
Klasik Hough dönüşüm (Classical Hough transform) algoritması en yaygın olarak parametrik ifade edilebilen doğru, daire ve elips gibi düzenli eğrileri tespit için kullanılır. Daha karmaşık bir hesaplama tekniği olan Genelleştirilmiş Hough Algoritması (Generalized Hough algorithm) burada ele alınmayacaktır. Bu algoritmanın en büyük avantajı resim üzerindeki boşluk ve gürültülere karşı toleranslı olmasıdır.

Bir resim içerisinde yerel olarak birbirinden bağımsız nokta (point) ve çizgiler (edge) bulunur. Bunlar içinden hangileri bir cismin kenarına (boundary) aittir bunun belirlenmesi bu algoritmada ana hedefimizdir.



Burada sınırları tespit edilen şeklin matematiksel modelinin (formülünün) çıkarılması esas amaçtır. Yani bitmap bir resmi (piksellerden oluşan bir resmi), vektörel (matematiksel formülle saklanan) bir resme dönüştürmek yapılmak istenen işlemdir. Bunun için hedeflenen cismin şekli önceden biliniyorsa bilginin çıkarılması daha kolaydır. Yani önceden resimde daire yada line olduğu biliniyorsa buna uygun arama yapmak işi kolaylaştırır. Eğer cismin ne olduğu bilinmiyorsa o zaman kenarları oluşturan parçalar bir seri olarak bulunup, ortaya çıkan eğri gösterilebilir.

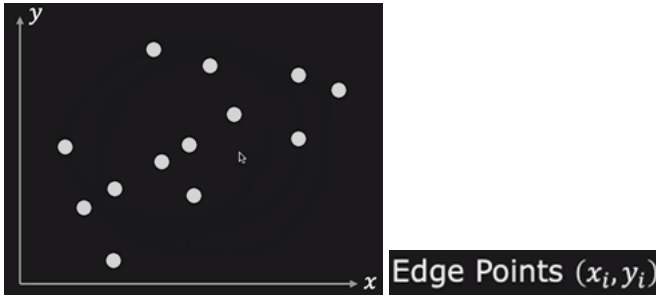
Aşağıdaki resmi incelediğimizde 4 noktanın hangi çizgilere ait olduğunu belirlemeye çalışsak, çok sayıda çizgi olasılığı ortaya çıkacaktır.



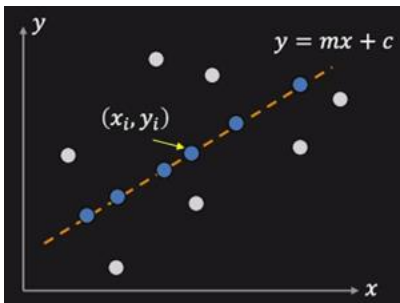
Herhangi bir noktanın muhtemel bir çizgiye ait olması durumunda bu çizgiyi iki şekilde resim üzerinde gösterebiliriz. (x,y) koordinatı çizgi üzerinde herhangi bir noktayı gösteriyorsa parametrik olarak kartezyen ve Polar koordinatlar olarak gösterebiliriz.

KARTEZYEN KOORDİNALARDA BİR DOĞRUNUN TESPİTİ

Aşağıdaki resimde verildiği gibi birbirinden bağımsız yerleştirilmiş noktalardan oluşan bir resim olsun. Bu noktaların içinde düz bir şekilde sıralanmış noktalardan bir doğruyu tespit etmeye çalışalım. Doğrunun denklemi kartezyen koordinatlarda $y=mx + c$ şeklinde tarif edilebilir. Burada m doğrunun eğimi, c ise bir sabittir.



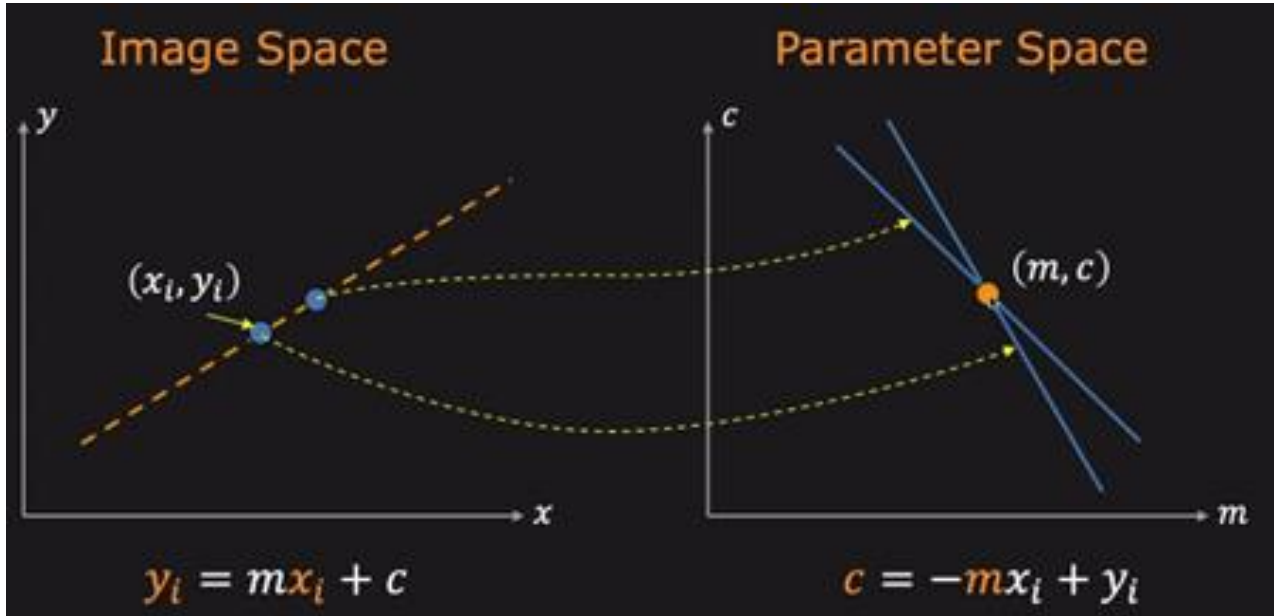
Bu noktaların bulunduğu konumlar resim üzerinde (uzayında) aşağıdaki gibi gösterilebilir. Her bir nokta sonsuza giden bir çizginin üzerindeki bir konumu temsil edecektir. Buradaki x ve y koodinatlari değiştirilerek çizgi sonsuza kadar çizilebilir.



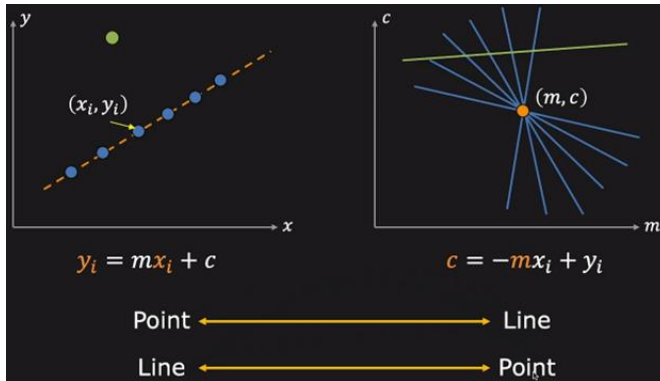
Bu sefer denklemimizi x-y ortamından (resim uzayından) parametrelerin olduğu c-m ortamına (parametre uzayına) dönüştürelim. Bu durumda denklemimiz $y=mx + c$ iken $c=-mx + y$ şekline dönüşmüş olacaktır.

$$y_i = mx_i + c \iff c = -mx_i + y_i$$

Resim üzerindeki iki noktayı alalım. Bu noktalara ait denklemleri x,y uzayından m,c uzayına dönüştürelim. Eğer A ve B noktası ayrık uzayda (resim uzayında) (spatial domain) iki tane birbirine bağlı nokta ise bu noktalara ait oluşturulan yeni m,c uzayındaki doğruların denklemleri m,c uzayında (parametre uzayı yada Hough uzayı denir) birbirini kesecektir.



Benzer şekilde çok sayıda sıralanmış noktalar (m,c) uzayında herbiri bir çizgiye karşılık gelir ve bunların $(c=-mx + y)$ denklemleri içindeki m ve c değerleri aynı olacağından bu nokta üzerinde kesişeceklerdir. Farklı bir yerde bulunan bir noktaya ait çizgi ise bu ortak noktadan geçmeyecektir.



(m,c) uzayında kesişen noktaların sayısı iki boyutlu bir dizi içinde tutulursa ortaya çıkan en yüksek değerli dizi elemanına ait m ve c değerleri resim uzayında (x,y) bir çizginin fonksiyonundaki m ve c katsayıları olacaktır. Böylece aranan doğrunun denklemleri $y=mx + c$ şeklinde bulunmuş olacaktır. Dizi üzerindeki elemanların büyüklükleri belli bir eşik değeri kullanılırsa az sayıda kesişimi gösteren değerler elenebilir. Böylece resim uzayında çok kısa çizgileri ele alınmamış olur.

		$A(m, c)$				
c	1	0	0	0	1	
	0	1	0	1	0	
	1	1	3	1	1	
	0	1	0	1	0	
	1	0	0	0	1	
		m				

Sayısal Örnek:

Q1. Using Hough transform, show that the following points are collinear. Also find the equation of the line. $(1,2)$, $(2,3)$ and $(3,4)$.

Equation of the line:
 $y = mx + c$

In order to perform Hough transform, we need to convert the line from (x,y) plane to (m,c) plane.

$$c = -mx + y$$

Noktalarımızı denklemlerde yerine korsak m,c uzayındaki denklemlerizi buluruz

(i) For $(x,y) = (1,2)$, $c = -m + 2$

if $c = 0$, $0 = -m + 2$
 $m = 2$

if $m = 0$, $c = 2$

Thus, $(m,c) = (2,2)$.

(ii) For $(x,y) = (2,3)$, $c = -2m + 3$

if $c = 0$, $0 = -2m + 3$
 $2m = 3$
 $m = 3/2 = 1.5$, $m = 1.5$

if $m = 0$, $c = 3$

Thus, $(m,c) = (1.5,3)$.

(iii) For $(x,y) = (3,4)$, $c = -3m + 4$

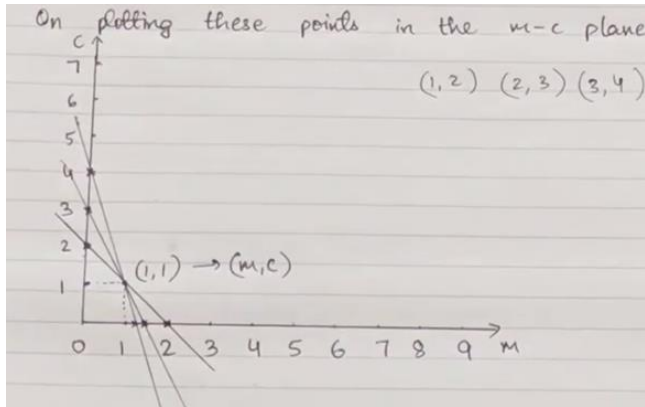
if $c = 0$, $0 = -3m + 4$
 $3m = 4$
 $m = 4/3 = 1.33$, $m = 1.33$

if $m = 0$, $c = 4$

Thus $(m,c) = (1.33,4)$.

$$(m,c) = (2,2), (1.5,3), (1.33,4)$$

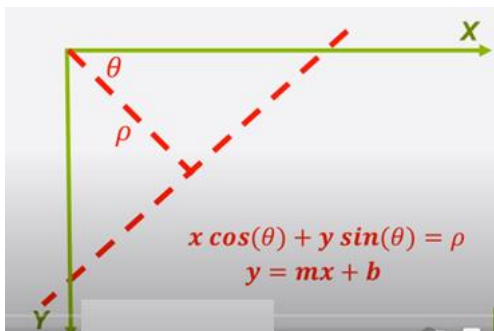
Bu durumda m,c uzayında doğrularımızı oluşturan noktalar olur. Bu noktalar birer m,c uzayında birer doğruyu gösterir. Bu üç tane doğrunun kesiştiği nokta $(m,c) = (1,1)$ noktasıdır. Bu nokta bize aradığımız resim uzayındaki doğrunun denklemini yani $y=mx+c$ den $y=x+1$ verecektir.



Original equation:
 $y = mx + c$

Substituting $(1,1)$:
 $y = x + 1 \rightarrow$ Final equation

DOĞRUNUN DENKLEMİNİ POLAR KOORDİNLARDA OLUŞTURMA



Denklemlerde bu formül kullanıldı..

Soru(GPT):Aşağıdaki gibi elimizde iki tane denklem var. Bu iki denklemin ortak parametresi olan θ ve ρ çözüm kümesini nasıl bulabilirim? $\rho = x_1 \cos(\theta) + y_1 \sin(\theta)$, $\rho = x_2 \cos(\theta) + y_2 \sin(\theta)$

Cevap (GPT):

Verilen denklemlerde θ ve ρ 'nin çözüm kümesini bulmak için denklemleri birleştirebiliriz:

$$x_1 \cos(\theta) + y_1 \sin(\theta) = x_2 \cos(\theta) + y_2 \sin(\theta)$$

Burada θ 'yi yalnız bırakıp denklemi çözebiliriz:

$$x_1 \cos(\theta) - x_2 \cos(\theta) = y_2 \sin(\theta) - y_1 \sin(\theta)$$

$$(x_1 - x_2) \cos(\theta) = (y_2 - y_1) \sin(\theta)$$

$$\sin(\theta) / \cos(\theta) = (x_1 - x_2) / (y_2 - y_1)$$

$$\tan(\theta) = (x_1 - x_2) / (y_2 - y_1)$$

$$\tan \theta \simeq \frac{(x_1 - x_2)}{(y_2 - y_1)}$$

$$\theta = \text{atan}((x_1 - x_2) / (y_2 - y_1))$$

Burada atan, arktanjanant fonksiyonunu temsil eder. θ 'nin değerini bu şekilde bulabiliriz.

Elde edilen θ değerini herhangi bir denklemde kullanarak ρ 'yı bulabiliriz. Örneğin, ilk denklemi kullanarak ρ 'yı bulmak için:

$$\rho = x_1 \cos(\theta) + y_1 \sin(\theta)$$

Bu şekilde θ ve ρ 'nin çözüm kümesini bulabilirsiniz.

Grafiği döngü kullanarak noktalar ile çizdirirsek şu formüllü kullanabiliriz.

$$\rho = x \cos(\theta) + y \sin(\theta) \Rightarrow y = \frac{\rho - x \cos(\theta)}{\sin(\theta)}$$

Üstteki formül programda x eksenini boyunca tarama yaparak çizilir. Fakat teta sıfır olursa hata verir. Bu durumda yani dikey çizgilerde formül değiştirilip tarama eksenini y boyunca olmalıdır. Bu durumda şu formülü kullanabiliriz.

$$\rho = x \cos(\theta) + y \sin(\theta) \Rightarrow x = \frac{\rho - y \sin(\theta)}{\cos(\theta)}$$

Better Parameterization

Issue: Slope of the line $-\infty \leq m \leq \infty$

- Large Accumulator
- More Memory and Computation

Solution: Use $x \sin \theta - y \cos \theta + \rho = 0$

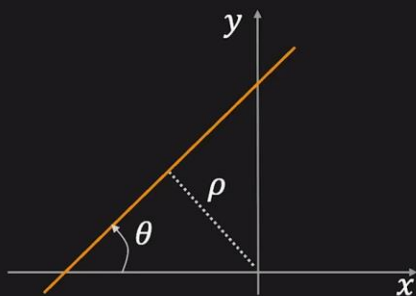
- Orientation θ is finite: $0 \leq \theta < \pi$
- Distance ρ is finite



© 2020 Shree K. Nayar

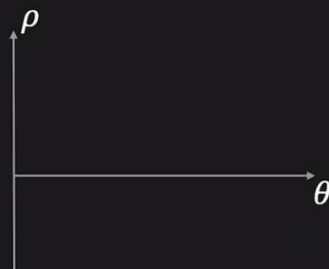
Better Parameterization

Image Space



$$x \sin \theta - y \cos \theta + \rho = 0$$

Parameter Space



$$x \sin \theta - y \cos \theta + \rho = 0$$

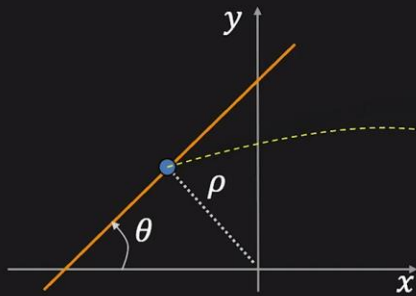


© 2020 Shree K. Nayar

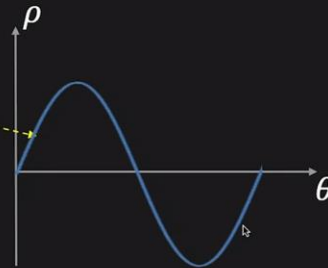
Better Parameterization

Image Space

Parameter Space



$$x \sin \theta - y \cos \theta + \rho = 0$$



$$x \sin \theta - y \cos \theta + \rho = 0$$

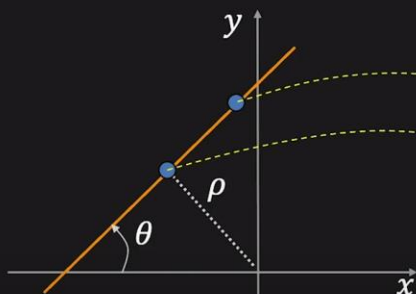
© 2020 Shree K. Nayar

Hough Transform | Boundary Detection

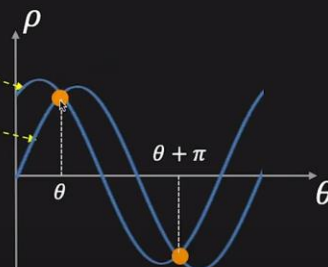
Better Parameterization

Image Space

Parameter Space



$$x \sin \theta - y \cos \theta + \rho = 0$$

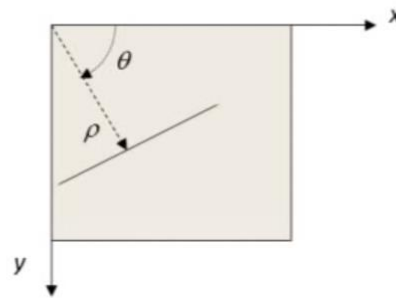


$$x \sin \theta - y \cos \theta + \rho = 0$$

10:24 / 21:39

Hough Transform

- Angle, axis conventions
 - Angle range is $-90..+89$
 - Rho range is $-D_{max}..+D_{max}$
 - D_{max} is the largest possible distance



- Example of a point at $(x,y) = (50,100)$



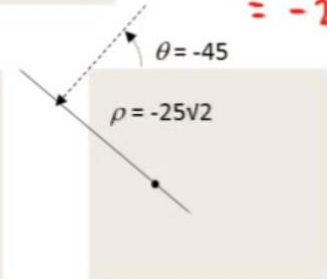
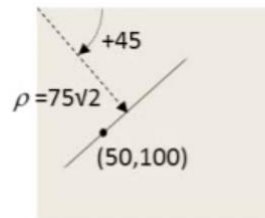
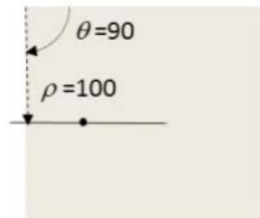
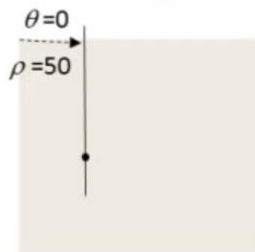
$$\theta = -45^\circ$$

$$\rho = 50 \frac{\sqrt{2}}{2} - 100 \frac{\sqrt{2}}{2}$$

$$= -25\sqrt{2}$$

$$\rho = x \cos \theta + y \sin \theta$$

$$\theta = 45^\circ: \rho = 50 \frac{\sqrt{2}}{2} + 100 \frac{\sqrt{2}}{2} = 75\sqrt{2}$$



Hough Transform Mechanics

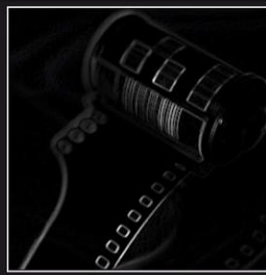
- How big should the accumulator cells be?
 - Too big, and different lines may be merged
 - Too small, and noise causes lines to be missed
- How many lines?
 - Count the peaks in the accumulator array
- Handling inaccurate edge locations:
 - Increment patch in accumulator rather than single point



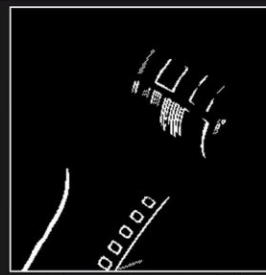
Line Detection Results



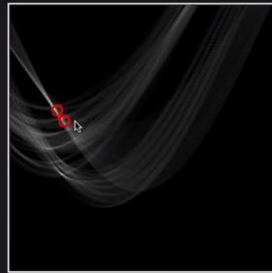
Original Image



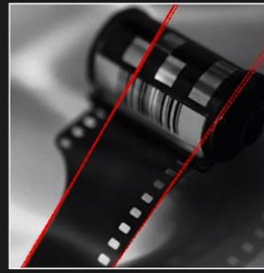
Gradient



Edge (Threshold)



Hough Transform $A(\rho, \theta)$



Detected Lines

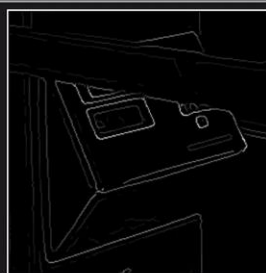
© 2020 Shree K. Nayar



Line Detection Results



Original Image



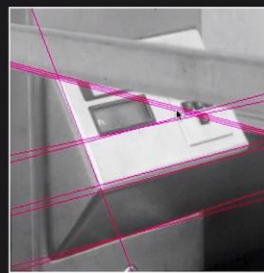
Gradient



Edge (Threshold)



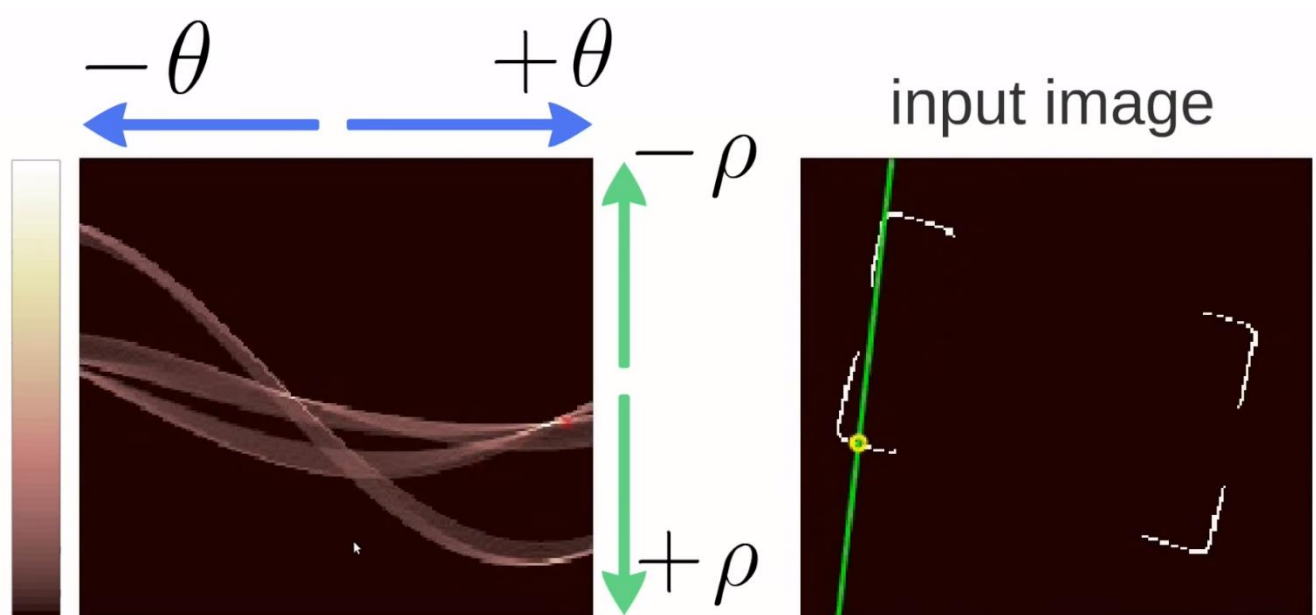
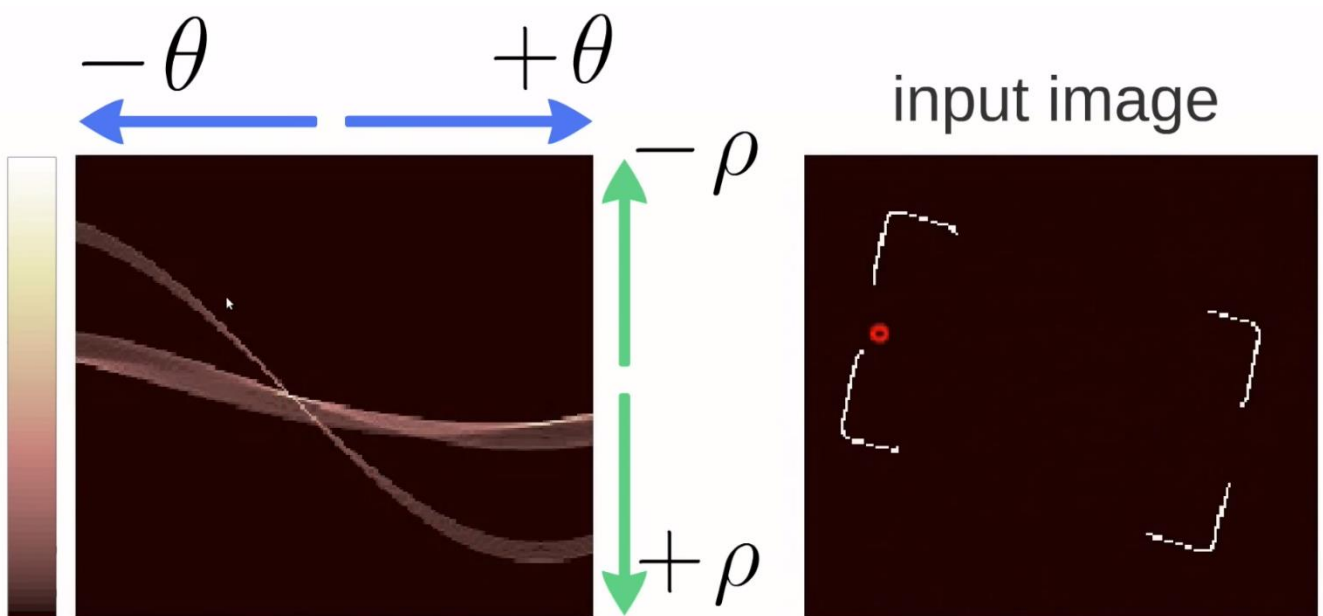
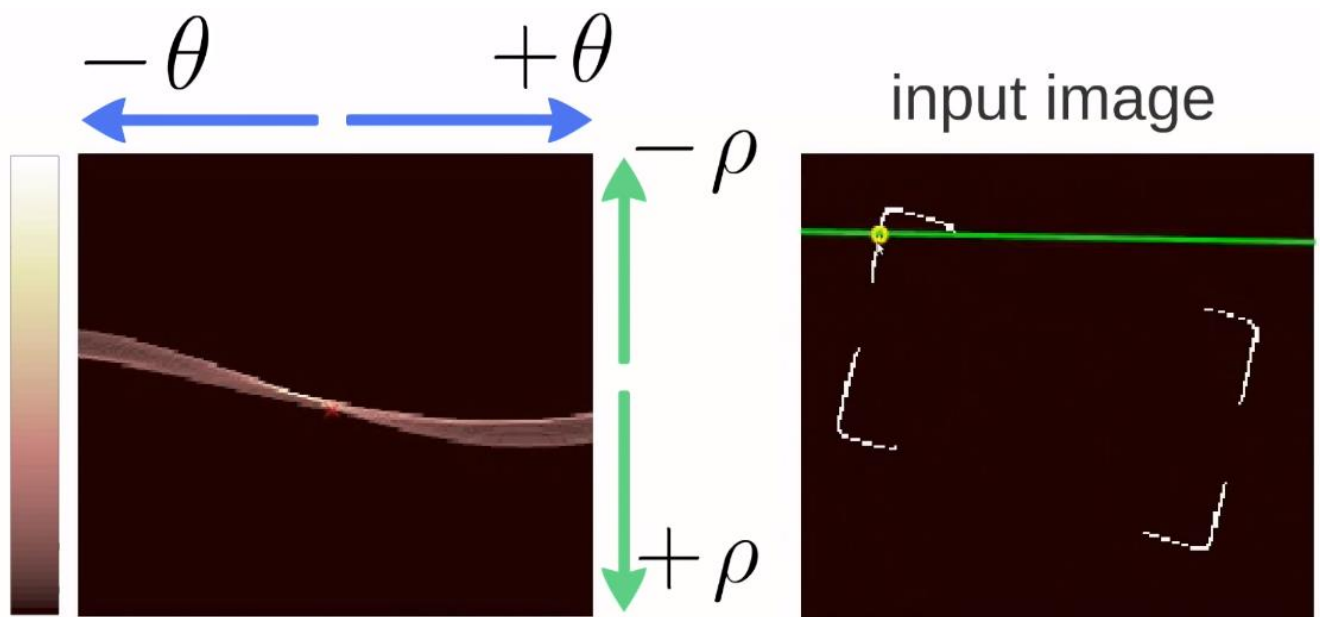
Hough Transform $A(\rho, \theta)$



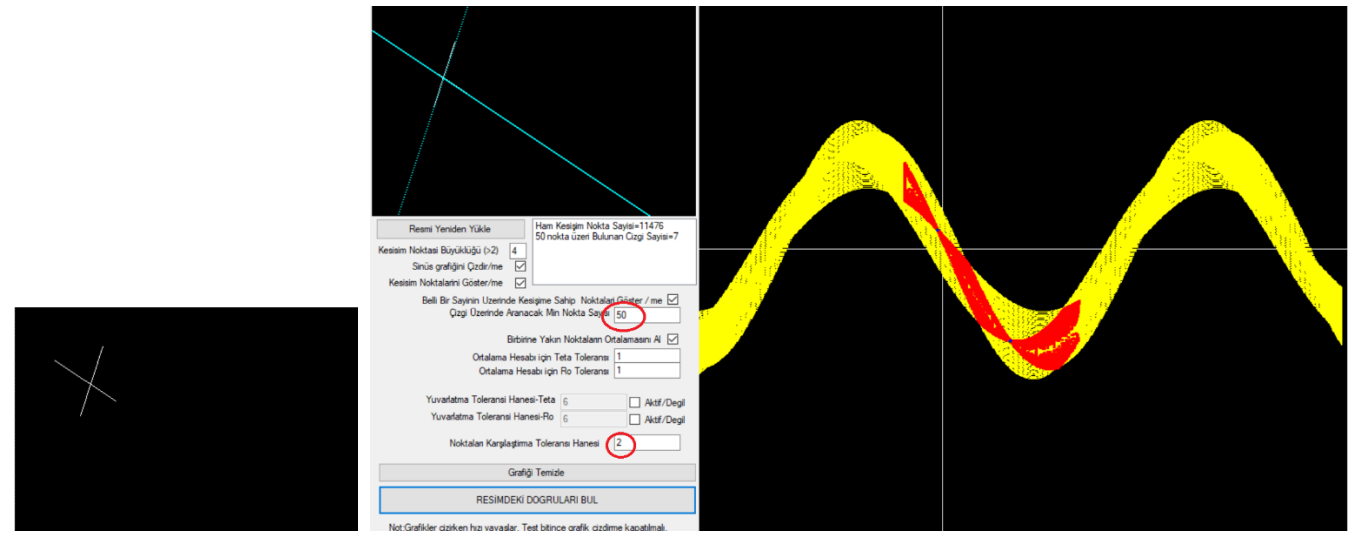
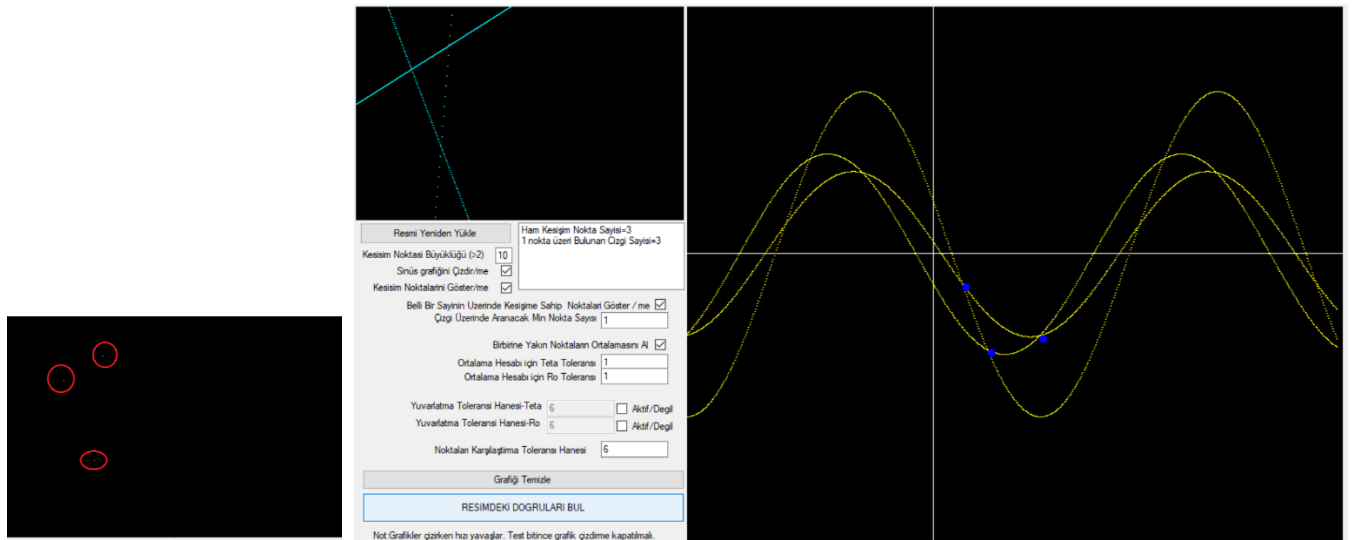
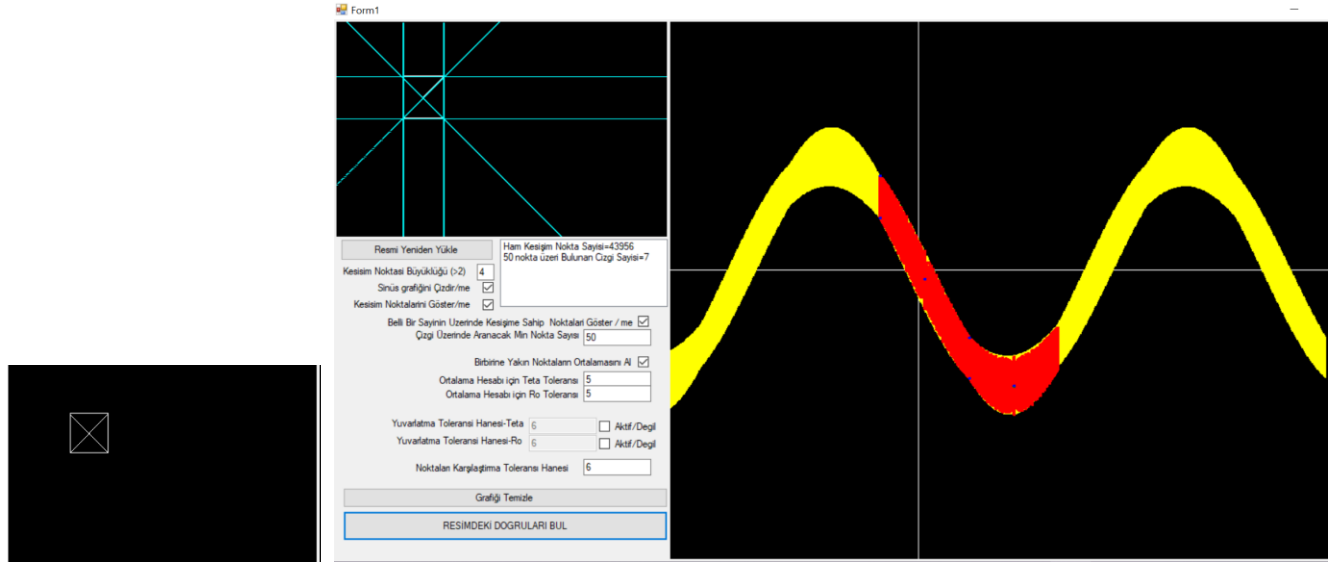
Detected Lines

© 2020 Shree K. Nayar





Doğrunun Tespiti Programı



```
using System;
using System.Collections;
using System.Drawing;
using System.Windows.Forms;
```

```

namespace Uygulama1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();

            private void Form1_Load(object sender, EventArgs e)
            {
                this.WindowState = FormWindowState.Maximized;

                CizimAlani1 = pictureBox1.CreateGraphics();
                CizimAlani2 = pictureBox2.CreateGraphics();

                pictureBox1.Image = Image.FromFile("C:/Users/pc1/Pictures/1.bmp"); //varsayılan
                resim olarak siyah zeminli bir resim yükleniyor.

            }

            //HOUGH ALGORİTASI .***** x*cos(o) + y*sin(o) =r FONKSİYONU kullanılıyor.
            private void btnRESMIYENIDENYUKLE_Click(object sender, EventArgs e)
            {
                // Grafik Çizimi yapmadan önce çizim alanını temizle
                pictureBox2.Invalidate();

                pictureBox1.Image = Image.FromFile("C:/Users/pc1/Pictures/1.bmp"); //varsayılan
                resim olarak yükleniyor.

            }

            //AAA-----PICTUREBOX TIKLANAN NOKTALARI RESİM İÇİNE EKLİYOR-----
            private void pictureBox1_MouseUp(object sender, MouseEventArgs e)
            {
                int FareX = e.X;
                int FareY = e.Y;

                Color OkunanRenk, DonusenRenk;
                int R = 0; //, G = 0, B = 0; //tek renk çalışılacak.
                Bitmap GirisResmi = new Bitmap(pictureBox1.Image);

                int ResimGenisligi = GirisResmi.Width;
                int ResimYuksekligi = GirisResmi.Height;

                Bitmap CikisResmi = new Bitmap(ResimGenisligi, ResimYuksekligi); //Cikis resmini
                oluşturun. Boyutları giriş resmi ile aynı olur.

                for (int x = 0; x < ResimGenisligi; x++)
                {
                    for (int y = 0; y < ResimYuksekligi; y++)
                    {
                        OkunanRenk = GirisResmi.GetPixel(x, y);

                        R = OkunanRenk.R;

                        if (FareX == x && FareY == y)
                        {
                            DonusenRenk = Color.FromArgb(255, 255, 255);
                        }
                        else
                        {
                            DonusenRenk = Color.FromArgb(R, R, R);
                        }

                        CikisResmi.SetPixel(x, y, DonusenRenk);
                    }
                }
            }
        }
    }
}

```

```

    }

    pictureBox1.Image = CikisResmi;

}

Graphics CizimAlani1, CizimAlani2;

Pen KalemBeyaz = new Pen(System.Drawing.Color.White, 1);
Pen KalemSari = new Pen(System.Drawing.Color.Yellow, 1);
Pen KalemKirmizi = new Pen(System.Drawing.Color.Red, 1);
Pen KalemMavi = new Pen(System.Drawing.Color.Blue, 1);
Pen KalemYesil = new Pen(System.Drawing.Color.Green, 1);
Pen KalemTurkuaz = new Pen(System.Drawing.Color.Cyan, 1);

Brush FircaSari = new SolidBrush(System.Drawing.Color.Yellow);
Brush FircaKirmizi = new SolidBrush(System.Drawing.Color.Red);
Brush FircaMavi = new SolidBrush(System.Drawing.Color.Blue);
Brush FircaYesil = new SolidBrush(System.Drawing.Color.Green);
Brush FircaTurkuaz = new SolidBrush(System.Drawing.Color.Cyan);

private void btnRESIMDEKI_DOGRULARI_BUL_Click(object sender, EventArgs e)
{
    //Butona tekrar tıklanıldığında dizileri sıfırlama yapıyor.
    DiziXnoktalari.Clear();
    DiziYnoktalari.Clear();

    DiziKesisim_Teta.Clear();
    DiziKesisim_Ro.Clear();
    DiziKesisim_AynisiBulundu.Clear();

    DiziRafine_Teta.Clear();
    DiziRafine_Ro.Clear();
    DiziRafine_TekrarSayisi.Clear();

    Dizi_Teta.Clear();
    Dizi_Ro.Clear();

    Bitmap GirisResmi = new Bitmap(pictureBox1.Image);

    //***** ALT İŞLEMLER *****
    //A-----
    ResimIcindekiBeyazNoktalariDiziyeYukle(GirisResmi); //DiziXnoktalari.Clear();
    DiziYnoktalari.Clear();

    //B-----Grafik Çizdirme
    if (ckbSinGrafiginiCizdir.Checked==true)
    {
        DizidekiNoktalarinSinGrafiginiCiz(); //Sinüs Grafiği--Çalışırsa hesaplamayı
        yavaşlatır
    }

    //C-----
    KesisimNoktalariniDiziyeAt(); //(teta, ro)

    //D-----Grafik Çizdirme
    if (ckbKesisimNoktalariniCizdir.Checked==true)
    {
        DizidekiKesisimNoktalariniGrafikteGoster(); //Kesişim noktalarını Grafikte çiz--
        -DİREK HESAPLAMADA BURAYI KAPAT--HIZI YAVAŞLATIR
    }
}

```

```

        //E-----
        KesisimDizisindenTekrarliOlanlariCikarTekrarSayisiniKaydet(); //Tekrarlı noktaları
cıkarak

        //F-----
        BelliBirTekrarSayisiUzerindekiNoktalar_TetaRoDizisineAt();

        //G-----Grafik Çizdirme
        if (ckbBelliBirSayininUzerindekiKesisimleriGoster.Checked==true)
        {
            Belli_Bir_Sayinin_Uzerindeki_Kesisim_Noktalarini_Grafikte_Goster(); //TetaRo
dizisindeki noktaları gösteriyor.

        }

        BirbirineYakinNoktalarin_OrtalamaDegerleriniHesapla_BuDegerlerIleDiziyiYenile();

        Ortalama_DiziTetaRo_Noktalarini_Grafikte_Goster();

        TetaRo_Dizisindeki_Noktalar_XYdüzleminde_ForDongusuKullanarakCiz(); //Burada çizgi
döngü ile çizilecek. Böylece ekran dışına çıkan durumlardada çizim yapılmış olacak.

    }

    //*****RESİM NOKTALARINI DİZİYE YÜKLÜYOR *****
    ArrayList DiziXnoktalar = new ArrayList();
    ArrayList DiziYnoktalar = new ArrayList();
    public void ResimIcindekiBeyazNoktalarDiziyeyukle(Bitmap GirisResmi)
    {
        Color OkunanRenk;
        int R = 0;

        int ResimGenisligi = GirisResmi.Width;
        int ResimYuksekligi = GirisResmi.Height;

        for (int x = 0; x < ResimGenisligi; x++)
        {
            for (int y = 0; y < ResimYuksekligi; y++)
            {
                OkunanRenk = GirisResmi.GetPixel(x, y);

                R = OkunanRenk.R;

                if (R > 128) //BEYAZ NOKTA VAR İSE
                {
                    DiziXnoktalar.Add(x);
                    DiziYnoktalar.Add(y);
                }
            }
        }

        listBox1.Items.Add("Resim Nokta Sayisi=" + DiziXnoktalar.Count);
    }

    //ÖTELEME
    int OrijinOtelemeX = 300;
    int OrijinOtelemeY = 300;

    double OlcekX = 1.2;
    double OlcekY = 1;

    //***** SİNÜS GRAFİKLERİNİ ÇİZDİRİYOR

```



```

public void DizidekiNoktalarinSinGrafiginiCiz()
{
    //*****
    double x = 0, y = 0, TetaRadyan = 0, Ro = 0;

    //Eksen takımının yerini gösteriyor..
    CizimAlani2.DrawLine(KalemBeyaz, 0, OrijinOtelemeY, pictureBox2.Width,
OrijinOtelemeY);
    CizimAlani2.DrawLine(KalemBeyaz, OrijinOtelemeX, 0, OrijinOtelemeX,
pictureBox2.Height);

    for (int i = 0; i < DiziXnoktalari.Count; i++)
    {
        x = Convert.ToDouble(DiziXnoktalari[i]);
        y = Convert.ToDouble(DiziYnoktalari[i]);

        for (int TetaDerece = -360; TetaDerece <= 360; TetaDerece++)
        {
            TetaRadyan = (TetaDerece * 2 * Math.PI) / 360; //Dereceyi Radyan a çevir

            Ro = x * Math.Cos(TetaRadyan) + y * Math.Sin(TetaRadyan); //p = x cos(θ)+y
sin(θ)

            //Teta ve Ro ifadeleri x ve y gibi eksenleri gösteriyor
            CizimAlani2.FillEllipse(FircaSari, Convert.ToInt16((TetaDerece +
OrijinOtelemeX) * OlcekX), Convert.ToInt16((Ro + OrijinOtelemeY) * OlcekY), 2, 2);
        }
    }
}

ArrayList DiziKesisim_Teta = new ArrayList();
ArrayList DiziKesisim_Ro = new ArrayList();
ArrayList DiziKesisim_AynisiBulundu = new ArrayList(); //İki döngü iç içe arama yaparken
aynısı bulunanları her seferinde fazladan okumasın diye bir sonraki prosedürde kullanılıyor.

//***** KESİŞİM NOKTALARINI DİZİYE YÜKLÜYOR
public void KesisimNoktalariniDiziyeAt()
{
    int YuvarlatmToleransi_Teta = Convert.ToInt32(txtYuvarlatmaToleransi_Teta.Text);
    int YuvarlatmToleransi_Ro = Convert.ToInt32(txtYuvarlatmaToleransi_Ro.Text);

    listBox1.Items.Clear();
    double x1, y1, x2, y2;

    double TetaRadyan = 0;
    double TetaDerece = 0;
    double Ro = 0;

    for (int i = 0; i < DiziXnoktalari.Count; i++)
    {
        x1 = Convert.ToDouble(DiziXnoktalari[i]);
        y1 = Convert.ToDouble(DiziYnoktalari[i]);

        for (int j = i + 1; j < DiziXnoktalari.Count; j++)
        {
            x2 = Convert.ToDouble(DiziXnoktalari[j]);
            y2 = Convert.ToDouble(DiziYnoktalari[j]);

            //İKİ DENKLEMİN ORTAK ÇÖZÜM KÜMESİ BULUNUYOR.
            TetaRadyan = Math.Atan((x1 - x2) / (y2 - y1)); //Radyan cinsinden Teta

            Ro = x1 * Math.Cos(TetaRadyan) + y1 * Math.Sin(TetaRadyan);

```

```

TetaDerece = 360 * TetaRadyan / (2 * Math.PI); //Açıyı Dizide derece
cinsinden tutacak.

//Bulduğu kesişim sayılarını diziye atmadan önce belli bir toleransdan
geçirecek

if (ckbTetaYuvarlatmaToleransi.Checked == true)
{
    TetaDerece = Math.Round(TetaDerece, YuvarlatmToleransi_Teta);
}

if (ckbRoYuvarlatmaToleransi.Checked == true)
{
    Ro = Math.Round(Ro, YuvarlatmToleransi_Ro);
}

DiziKesisim_Teta.Add(TetaDerece);
DiziKesisim_Ro.Add(Ro);
DiziKesisim_AynisiBulundu.Add(false);

//listBox1.Items.Add("Kesişim Noktasi= Teta: " + TetaDerece + ", Ro:" + Ro);
}

listBox1.Items.Add("Ham Kesişim Nokta Sayisi=" + DiziKesisim_Teta.Count);
}

//***** KESİŞİM NOKTALARINI GRAFİKTE GÖSTERİYOR
public void DizidekiKesisimNoktalariniGrafikteGoster()
{
    double Teta = 0, Ro = 0;
    int NoktaBuyuklugu = Convert.ToInt32(txtKesisimNoktasiBuyuklugu.Text);

    for (int i = 0; i < DiziKesisim_Teta.Count; i++)
    {
        try
        {
            Teta = Convert.ToDouble(DiziKesisim_Teta[i]);
            Ro = Convert.ToDouble(DiziKesisim_Ro[i]);

            CizimAlani2.FillEllipse(FircaKirmizi, Convert.ToInt16((Teta +
OrijinOtelemeX) * OlcekX), Convert.ToInt16((Ro + OrijinOtelemeY) * OlcekY), NoktaBuyuklugu,
NoktaBuyuklugu);
        }
        catch { }
    }
}

ArrayList DiziRafine_Teta = new ArrayList();
ArrayList DiziRafine_Ro = new ArrayList();
ArrayList DiziRafine_TekrarSayisi = new ArrayList();

//***** TEKRARLI KESİŞİM NOKTALARINI AZALTIYOR-RAFİNE DİZİSİNİ
OLUŞTURUYOR
public void KesisimDizisindenTekrarliOlanlariCikarTekrarSayisiniKaydet()
{
    double Teta1 = 0, Ro1 = 0;
    double Teta2 = 0, Ro2 = 0;
    int YuvarlatmaToleransi = Convert.ToInt16(txtYuvarlatmaToleransi.Text);

    for (int i = 0; i < DiziKesisim_Teta.Count; i++)
    {
        Teta1 = Convert.ToDouble(DiziKesisim_Teta[i]);
        Ro1 = Convert.ToDouble(DiziKesisim_Ro[i]);
        bool AynisiBulundu1 = Convert.ToBoolean(DiziKesisim_AynisiBulundu[i]);

```

```

        if (AynisiBulundu1 == false) //Daha önce ilk aranan hiç kullanılmadıysa..
        {
            //listBox1.Items.Add("Tekrarsız Kesişim Noktası" + Teta1 + " , " + Ro1);

            DiziRafine_Teta.Add(Teta1);
            DiziRafine_Ro.Add(Ro1);

            int k = 1; //Rafine edilmiş noktalardan kaç tane bulunduğunu tutatacak. Bunlar
            çizginin kaç noktadan oluştuğu sayıdır.

            for (int j = i + 1; j < DiziKesisim_Teta.Count; j++)
            {
                Teta2 = Convert.ToDouble(DiziKesisim_Teta[j]);
                Ro2 = Convert.ToDouble(DiziKesisim_Ro[j]);
                bool AynisiBulundu2 = Convert.ToBoolean(DiziKesisim_AynisiBulundu[j]);

                if (Math.Round(Teta1, YuvarlatmaToleransi) == Math.Round(Teta2,
                YuvarlatmaToleransi) && Math.Round(Ro1, YuvarlatmaToleransi) == Math.Round(Ro2,
                YuvarlatmaToleransi) && AynisiBulundu2 == false) //İlk aranan ile ikinci aranan eşitse ve
                ikincisi daha önce hiç işlem görmediyse
                {
                    DiziKesisim_AynisiBulundu[i] = true; //Birincisi kullanıldı, bir
                    daha bakmamak için true yapıldı
                    DiziKesisim_AynisiBulundu[j] = true; //İkincisi kullanıldı, bir
                    dahakine pas geçilmesi için true yapıldı.

                    k++;
                }

                DiziRafine_TekrarSayisi.Add(k); //Aynı noktadan kaç tane bulduysa onun
                tekrar sayısı oluyor.
            }
        }

        ArrayList Dizi_Teta = new ArrayList();
        ArrayList Dizi_Ro = new ArrayList();
        //*****
        public void BelliBirTekrarSayisiUzerindekiNoktalari_TetaRoDizisineAt()
        {
            int CizgideBulunmasiGerekenMinimumNoktaSayisi =
            Convert.ToInt16(txtMinNoktaSayisi.Text);
            double Teta = 0, Ro = 0;

            for (int i = 0; i < DiziRafine_Teta.Count; i++)
            {
                int KesisimUzerindekiNoktaSayisi = Convert.ToInt32(DiziRafine_TekrarSayisi[i]);

                if (KesisimUzerindekiNoktaSayisi >= CizgideBulunmasiGerekenMinimumNoktaSayisi)
                //-5- adet kesişimden yüksek olan çizgileri ele alacak.
                {
                    Teta = Convert.ToDouble(DiziRafine_Teta[i]);
                    Ro = Convert.ToDouble(DiziRafine_Ro[i]);

                    Dizi_Teta.Add(Teta);
                    Dizi_Ro.Add(Ro);
                }
            }

            listBox1.Items.Add(CizgideBulunmasiGerekenMinimumNoktaSayisi + " nokta üzeri Bulunan
            Cizgi Sayisi=" + Dizi_Teta.Count);
        }

```

```

//*****TETA_RO GRAFİK ÇİZİMİ
public void Belli_Bir_Sayinin_Uzerindeki_Kesisim_Noktalarini_Grafikte_Goster()
{
    double Teta = 0, Ro = 0;
    int NoktaBuyuklugu = Convert.ToInt32(txtKesisimNoktasiBuyuklugu.Text);

    for (int i = 0; i < Dizi_Teta.Count; i++)
    {
        Teta = Convert.ToDouble(Dizi_Teta[i]);
        Ro = Convert.ToDouble(Dizi_Ro[i]);

        CizimAlani2.FillEllipse(FircaYesil, Convert.ToInt16((Teta + OrijinOtelemeX) *
OlcekX), Convert.ToInt16((Ro + OrijinOtelemeY) * OlcekY), NoktaBuyuklugu, NoktaBuyuklugu);
    }
}

public void
BirbirineYakinNoktalarin_OrtalamaDegerleriniHesapla_BuDegerlerIleDiziyiYenile()
{
    double Teta1 = 0, Ro1 = 0;
    double Teta2 = 0, Ro2 = 0;
    double FarkTeta=0, FarkRo=0;
    double OrtalamaTeta = 0, OrtalamaRo = 0;

    int ToleransTeta =Convert.ToInt16(txtTetaToleransi.Text); //Ortalamasını almak için
Açılara uygulanacak Tolerans
    int ToleransRo = Convert.ToInt16(txtRoToleransi.Text); //Ortalamasını almak için
Boylara uygulanacak Tolerans

    if (ToleransTeta >= 1 && ToleransRo >= 1) //Tolerans verildi ise (1 den büyükse)
diziyi ortalama değerler ile yenileyecek..
    {
        for (int i = 0; i < Dizi_Teta.Count - 1; i++)
        {
            for (int j = i + 1; j < Dizi_Teta.Count; j++)
            {
                Teta1 = Convert.ToDouble(Dizi_Teta[i]);
                Ro1 = Convert.ToDouble(Dizi_Ro[i]);

                Teta2 = Convert.ToDouble(Dizi_Teta[j]);
                Ro2 = Convert.ToDouble(Dizi_Ro[j]);

                FarkTeta = Math.Abs(Teta1 - Teta2);
                FarkRo = Math.Abs(Ro1 - Ro2);

                if (FarkTeta <= ToleransTeta && FarkRo <= ToleransRo) //fark düşükse
noktalar yakın demektir.
                {
                    OrtalamaTeta = (Teta1 + Teta2) / 2;

                    Dizi_Teta[i] = OrtalamaTeta;
                    Dizi_Teta[j] = OrtalamaTeta;

                    OrtalamaRo = (Ro1 + Ro2) / 2;

                    Dizi_Ro[i] = OrtalamaRo;
                    Dizi_Ro[j] = OrtalamaRo;
                }
            }
        }
    }
}

//*****ORTALAMA TETA_RO GRAFİK ÇİZİMİ

```

```

public void Ortalama_DiziTetaRo_Noktalarini_Grafikte_Goster()
{
    double Teta = 0, Ro = 0;
    int NoktaBuyuklugu = Convert.ToInt32(txtKesisimNoktasiBuyuklugu.Text);

    for (int i = 0; i < Dizi_Teta.Count; i++)
    {
        Teta = Convert.ToDouble(Dizi_Teta[i]);
        Ro = Convert.ToDouble(Dizi_Ro[i]);

        CizimAlani2.FillEllipse(FircaMavi, Convert.ToInt16((Teta + OrijinOtelemeX) *
OlcekX), Convert.ToInt16((Ro + OrijinOtelemeY) * OlcekY), NoktaBuyuklugu, NoktaBuyuklugu);
    }
}

public void TetaRo_Dizisindeki_Noktalarini_XYdüzleminde_ForDongusuKullanarakCiz()
{
    double Ro = 0;
    double x = 0, y = 0;

    for (int i = 0; i < Dizi_Teta.Count; i++)
    {
        double TetaDerece = Convert.ToDouble(Dizi_Teta[i]); //Diziye teta derece olarak
kaydedilmişti.
        Ro = Convert.ToDouble(Dizi_Ro[i]);

        double TetaRadyan = TetaDerece * 2 * Math.PI / 360;

        if(TetaDerece==0) //Açı 0 ise çizgi DİKEY demektir. Dikey çizgileri formülde
kullanırken sıfıra bölme oluşuyor. 0 yüzden for döngüsü y ekseninde taranacak.
        {
            //MessageBox.Show("Açı 0=Çizgi Dikey");

            //Dikey çizgileri çizerken y ekseninde for döndürecek.
            //Her Teta-Ro çifti için bir çizgi oluşturacak.Bu çizgileri nokta kullanarak
çizecek.
            for (y = 0; y < 300; y++)
            {
                x = (Ro - y * Math.Sin(TetaRadyan)) / (Math.Cos(TetaRadyan));

                int X = Convert.ToInt32(x); //PictureBox integer ister
                int Y = Convert.ToInt32(y); //PictureBox integer ister

                //Grafiği nokta ile çizdirecek.
                CizimAlani1.FillEllipse(FircaTurkuaz, X, Y, 2, 2);
            }
        }
        else //Yatay çizgileri çizerken x ekseninde for döndürecek.
        {
            //Her Teta-Ro çifti için bir çizgi oluşturacak.Bu çizgileri nokta kullanarak
çizecek.
            for (x = 0; x < 400; x++)
            {
                y = (Ro - x * Math.Cos(TetaRadyan)) / (Math.Sin(TetaRadyan));

                int X = Convert.ToInt32(x); //PictureBox integer ister
                int Y = Convert.ToInt32(y); //PictureBox integer ister

                //Grafiği nokta ile çizdirecek.
                CizimAlani1.FillEllipse(FircaTurkuaz, X, Y, 2, 2);
            }
        }
    }
}

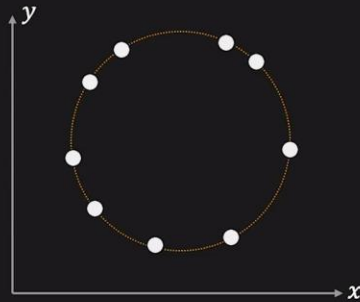
```

```
    }  
    }  
}  
  
    //***** EKRANLA İLGİLİ SEÇENEK VE AYARLAMA KODLARI ***** Algoritma ile  
ilgili değil  
    //***** EKRANLA İLGİLİ SEÇENEK VE AYARLAMA KODLARI ***** Algoritma ile  
ilgili değil  
    //***** EKRANLA İLGİLİ SEÇENEK VE AYARLAMA KODLARI ***** Algoritma ile  
ilgili değil  
    private void ckbTetaYuvarlatmaToleransi_CheckedChanged(object sender, EventArgs e)  
    {  
        if (ckbTetaYuvarlatmaToleransi.Checked == true)  
            txtYuvarlatmaToleransi_Teta.Enabled = true;  
        else  
            txtYuvarlatmaToleransi_Teta.Enabled = false;  
    }  
  
    private void ckbRoYuvarlatmaToleransi_CheckedChanged(object sender, EventArgs e)  
    {  
        if (ckbRoYuvarlatmaToleransi.Checked == true)  
            txtYuvarlatmaToleransi_Ro.Enabled = true;  
        else  
            txtYuvarlatmaToleransi_Ro.Enabled = false;  
    }  
    private void btnGrafigiTemizle_Click(object sender, EventArgs e)  
    {  
        // Grafik Çizimi yapmadan önce çizim alanını temizle  
        pictureBox1.Invalidate();  
        pictureBox2.Invalidate();  
    }  
}  
}
```

Kaynak Kodlar: www.IbrahimCayiroglu.com/Dokumanlar/GoruntuIsleme/HOUGH_ALGORITMASI.rar

HOUGH TRANSFORM: CIRCLE DETECTION

Hough Transform: Circle Detection

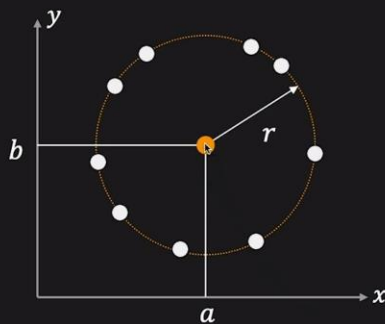


Equation of Circle: $(x_i - a)^2 + (y_i - b)^2 = r^2$

© 2020 Shree K. Nayar



Hough Transform: Circle Detection



Equation of Circle: $(x_i - a)^2 + (y_i - b)^2 = r^2$

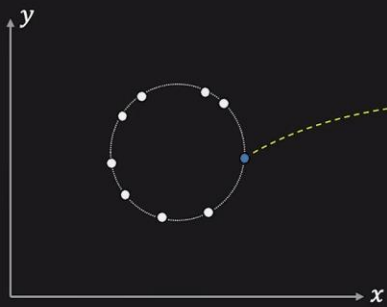
© 2020 Shree K. Nayar



Hough Transform: Circle Detection

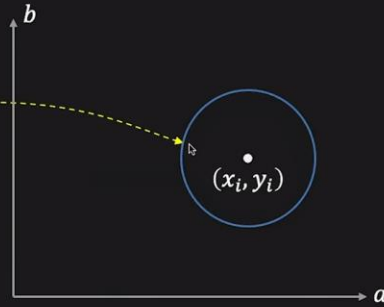
If radius r is known: Accumulator Array: $A(a, b)$

Image Space



$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space



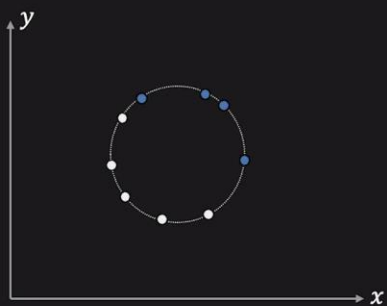
$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

© 2020 Shree K. Nayar

Hough Transform: Circle Detection

If radius r is known: Accumulator Array: $A(a, b)$

Image Space



$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space



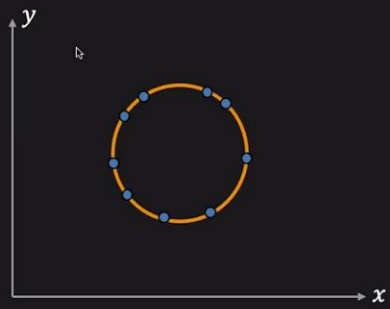
$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

© 2020 Shree K. Nayar

Hough Transform: Circle Detection

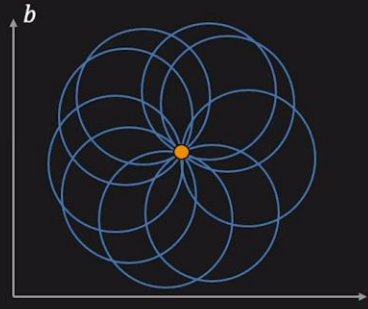
If radius r is known: Accumulator Array: $A(a, b)$

Image Space



$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

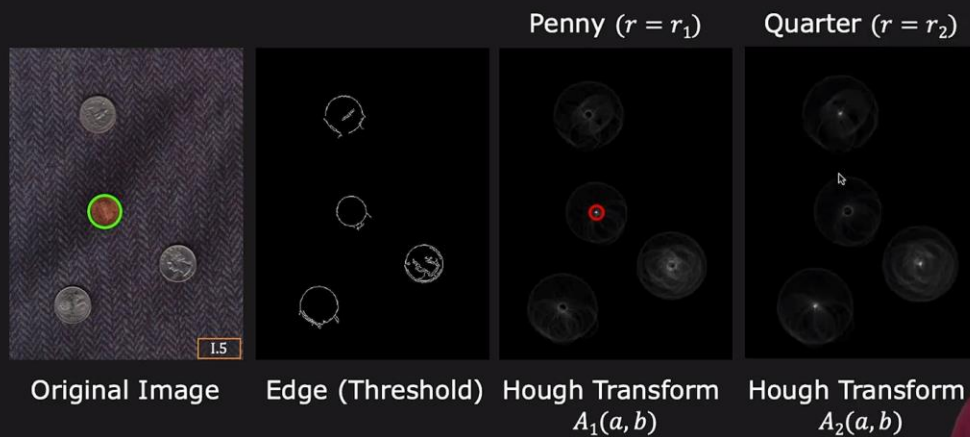
Parameter Space



$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

© 2020 Shree K. Nayar

Circle Detection Results

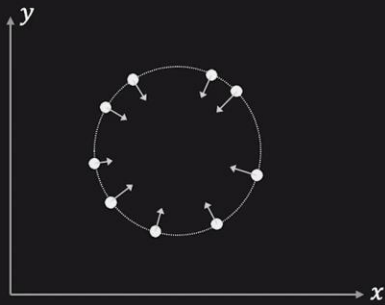


© 2020 Shree K. Nayar

Using Gradient Information

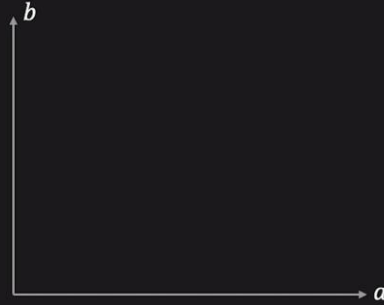
Given: Edge Location (x_i, y_i) , Edge Direction φ_i and Radius r

Image Space



$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space



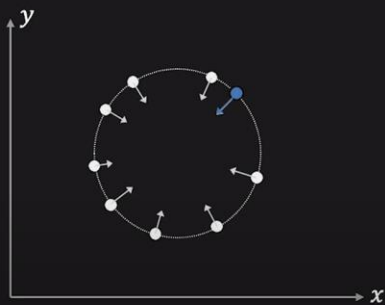
© 2020 Shree K. Nayar

Hough Transform | Boundary Detection

Using Gradient Information

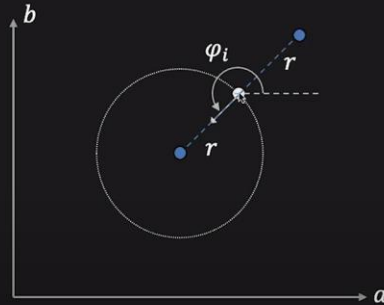
Given: Edge Location (x_i, y_i) , Edge Direction φ_i and Radius r

Image Space



$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space



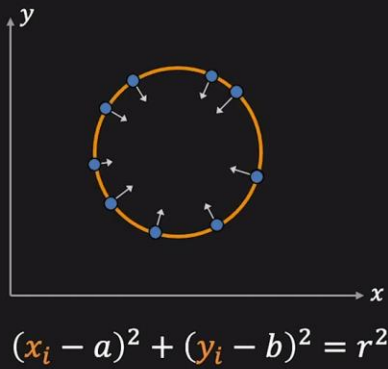
$$a = x_i \pm r \cos \varphi_i$$

$$b = y_i \pm r \sin \varphi_i$$

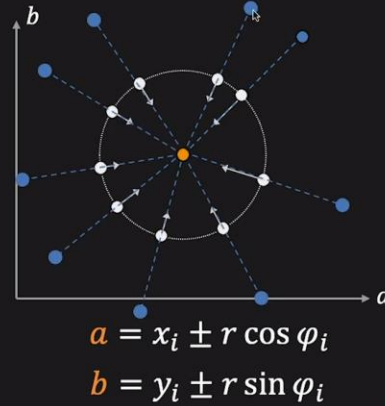
Using Gradient Information

Given: Edge Location (x_i, y_i) , Edge Direction φ_i and Radius r

Image Space



Parameter Space



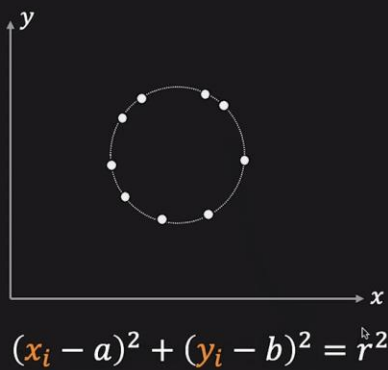
© 2020 Shree K. Nayar

Circle Detection: R Is Not Known

Hough Transform: Circle Detection

If radius r is NOT known: Accumulator Array: $A(a, b, r)$

Image Space

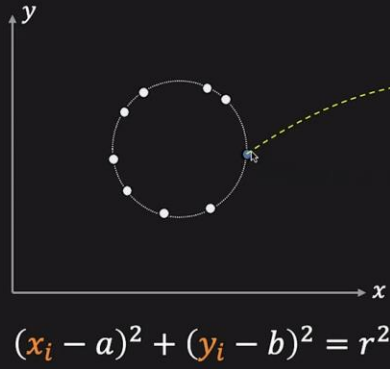


© 2020 Shree K. Nayar

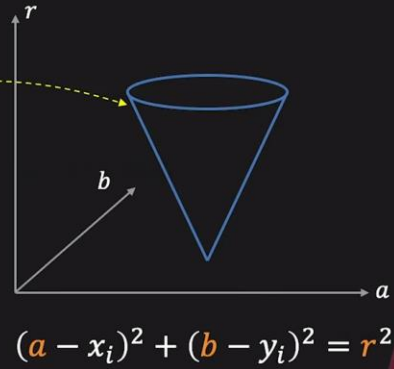
Hough Transform: Circle Detection

If radius r is NOT known: Accumulator Array: $A(a,b,r)$

Image Space



Parameter Space



© 2020 Shree K. Nayar

Referanslar

https://www.youtube.com/watch?v=XRBC_xkZREg

<https://www.youtube.com/watch?v=uDB2qGqnQ1g>

<https://www.youtube.com/watch?v=t1GXMvK9m84>

<https://www.youtube.com/watch?v=4zHbl-fFIII>