Computer and Robot Vision

Homework#9

R01944040 柳成蔭

這次的作業是對原圖做 Edge Detection。
我使用 VS2012 編寫程式

(a) Robert's Operator

Roberts operators: two 2X2 masks to calculate gradient

Figure 7.21 Masks used for the Roberts operators.

- gradient magnitude: $\sqrt{r_1^2 + r_2^2}$
- where r_1, r_2 are values from first, second masks respectively

```
double RoberGradient( const Mat src, const Kernel Mask1,
const Kernel Mask2, int sI, int sJ )
{
    float gradient=0;
    float r1=0;
    float r2=0;
    for (int MaskI = 0; MaskI <= Mask1.kRows-1; MaskI++)
    {
        for (int MaskJ = 0; MaskJ <= Mask1.kCols-1; MaskJ++)
```

```
        {
            int sX=sI+(MaskI-Mask1.anchorX);
            int sY=sJ+(MaskJ-Mask1.anchorY);
            if (sX>=0 && sX<=src.rows-1 &&
                sY>=0 && sY<=src.cols-1)
            {
                r1=r1+(int)src.at<uchar>(sX,sY) *
(float)Mask1.values.at<float>(MaskI, MaskJ);
                r2=r2+(int)src.at<uchar>(sX,sY) *
(float)Mask2.values.at<float>(MaskI, MaskJ);
            }
            else
                return 0;
        }
    }
    gradient=sqrt(r1*r1+r2*r2);
    return gradient;
}

void Robert(const Mat src, Mat res, int threshold)
{
    //kernal
    float m1[]={-1, 0,
                0, 1 };
    Mat M1=Mat(2,2,CV_32F,m1).clone();
    Kernel Mask1(2, 2, 0, 0, M1);

    float m2[]={0,-1,
                1,0  };
    Mat M2=Mat(2,2,CV_32F,m2).clone();
    Kernel Mask2(2, 2, 0, 0, M2);

    float gradient;
    for (int sI = 0; sI <= src.rows-1; sI++)
    {
        for (int sJ = 0; sJ <= src.cols-1; sJ++)
        {
```

```
            gradient=RoberGradient(src, Mask1, Mask2, sI,
sJ);
            if(gradient>=threshold)
                res.at<uchar>(sI,sJ)=0;
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
}
```

Threshold 取 12 的處理結果：

(b) Prewitt's Edge Detector

Prewitt edge detector: two 3X3 masks in row column direction

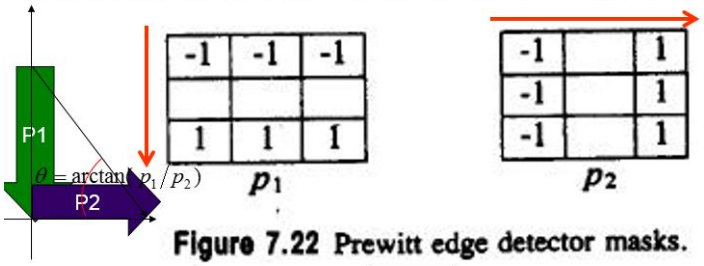


Figure 7.22 Prewitt edge detector masks.

- gradient magnitude: $g = \sqrt{p_1^2 + p_2^2}$
- gradient direction: $\theta = \arctan(p_1/p_2)$ clockwise w.r.t. column axis
- where p_1, p_2 are values from first, second masks respectively

```
double PrewittGradient( const Mat src, const Kernel Mask1,
```

```
const Kernel Mask2, int sI, int sJ )
{
    float gradient=0;
    float p1=0;
    float p2=0;
    for (int MaskI = 0; MaskI <= Mask1.kRows-1; MaskI++)
    {
        for (int MaskJ = 0; MaskJ <= Mask1.kCols-1; MaskJ++)
        {
            int sX=sI+(MaskI-Mask1.anchorX);
            int sY=sJ+(MaskJ-Mask1.anchorY);
            if (sX>=0 && sX<=src.rows-1 &&
                sY>=0 && sY<=src.cols-1)
            {
                p1=p1+(int)src.at<uchar>(sX,sY) *
(float)Mask1.values.at<float>(MaskI, MaskJ);
                p2=p2+(int)src.at<uchar>(sX,sY) *
(float)Mask2.values.at<float>(MaskI, MaskJ);
            }
            else
                return 0;
        }
    }
    gradient=sqrt(p1*p1+p2*p2);
    return gradient;
}

void Prewitt(const Mat src, Mat res, int threshold)
{
    //kernal
    float m1[]={-1,-1,-1,
                0, 0, 0,
                1, 1, 1 };
    Mat M1=Mat(3,3,CV_32F,m1).clone();
    Kernel Mask1(3, 3, 1, 1, M1);

    float m2[]={-1, 0, 1,
```

```
                -1, 0, 1,
                -1, 0, 1 };
    Mat M2=Mat(3,3,CV_32F,m2).clone();
    Kernel Mask2(3, 3, 1, 1, M2);

    float gradient;
    for (int sI = 0; sI <= src.rows-1; sI++)
    {
        for (int sJ = 0; sJ <= src.cols-1; sJ++)
        {
            gradient=PrewittGradient(src, Mask1, Mask2, sI,
sJ);
            if(gradient>=threshold)
                res.at<uchar>(sI,sJ)=0;
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
}
```

Threshold 取 24 的處理結果：

(c) Sobel's Edge Detector

Sobel edge detector: two 3X3 masks in row column direction

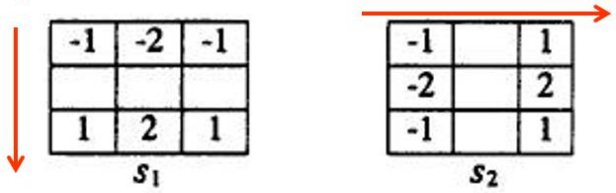


Figure 7.23 Sobel edge detector masks.

- gradient magnitude: $g = \sqrt{s_1^2 + s_2^2}$
- gradient direction: $\theta = \arctan(s_1/s_2)$ clockwise w.r.t. column axis
- where s_1, s_2 are values from first, second masks respectively

```
double SobelGradient( const Mat src, const Kernel Mask1,
const Kernel Mask2, int sI, int sJ )
{
    float gradient=0;
    float s1=0;
    float s2=0;
    for (int MaskI = 0; MaskI <= Mask1.kRows-1; MaskI++)
    {
        for (int MaskJ = 0; MaskJ <= Mask1.kCols-1;
MaskJ++)
        {
            int sX=sI+(MaskI-Mask1.anchorX);
            int sY=sJ+(MaskJ-Mask1.anchorY);
            if (sX>=0 && sX<=src.rows-1 &&
                sY>=0 && sY<=src.cols-1)
            {
                s1=s1+(int)src.at<uchar>(sX,sY) *
(float)Mask1.values.at<float>(MaskI, MaskJ);
                s2=s2+(int)src.at<uchar>(sX,sY) *
(float)Mask2.values.at<float>(MaskI, MaskJ);
            }
            else
                return 0;
        }
    }
    gradient=sqrt(s1*s1+s2*s2);
    return gradient;
}
```

```
void Sobel(const Mat src, Mat res, int threshold)
{
    //kernal
    float m1[]={-1,-2,-1,
                0, 0, 0,
                1, 2, 1 };
    Mat M1=Mat(3,3,CV_32F,m1).clone();
    Kernel Mask1(3, 3, 1, 1, M1);

    float m2[]={-1, 0, 1,
                -2, 0, 2,
                -1, 0, 1 };
    Mat M2=Mat(3,3,CV_32F,m2).clone();
    Kernel Mask2(3, 3, 1, 1, M2);

    float gradient;
    for (int sI = 0; sI <= src.rows-1; sI++)
    {
        for (int sJ = 0; sJ <= src.cols-1; sJ++)
        {
            gradient=SobelGradient(src, Mask1, Mask2, sI,
sJ);
            if(gradient>=threshold)
                res.at<uchar>(sI,sJ)=0;
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
}
```

Threshold 取 38 的處理結果：

(d) Frei and Chen's Gradient Operator

Frei and Chen edge detector: two in a set of nine orthogonal masks (3X3)

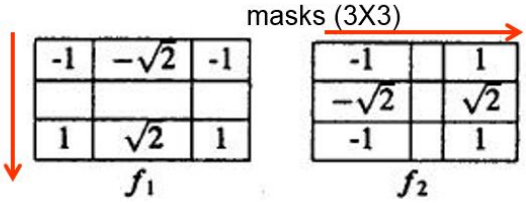


Figure 7.24 Frei and Chen gradient masks.

- gradient magnitude: $g = \sqrt{f_1^2 + f_2^2}$
- gradient direction: $\theta = \arctan(f_1 / f_2)$ clockwise w.r.t. column axis
- where f_1, f_2 are values from first, second masks respectively

```
double FreiChenGradient( const Mat src, const Kernel Mask1,
const Kernel Mask2, int sI, int sJ )
{
    float gradient=0;
    float f1=0;
    float f2=0;
    for (int MaskI = 0; MaskI <= Mask1.kRows-1; MaskI++)
    {
        for (int MaskJ = 0; MaskJ <= Mask1.kCols-1;
MaskJ++)
        {
            int sX=sI+(MaskI-Mask1.anchorX);
            int sY=sJ+(MaskJ-Mask1.anchorY);
            if (sX>=0 && sX<=src.rows-1 &&
```

```
                sY>=0 && sY<=src.cols-1)
            {
                f1=f1+(int)src.at<uchar>(sX,sY) *
(float)Mask1.values.at<float>(MaskI, MaskJ);
                f2=f2+(int)src.at<uchar>(sX,sY) *
(float)Mask2.values.at<float>(MaskI, MaskJ);
            }
            else
                return 0;
        }
    }
    gradient=sqrt(f1*f1+f2*f2);
    return gradient;
}

void FreiChen(const Mat src, Mat res, int threshold)
{
    //kernal
    float m1[]={-1,-sqrt(2),-1,
                0,       0, 0,
                1, sqrt(2), 1 };
    Mat M1=Mat(3,3,CV_32F,m1).clone();
    Kernel Mask1(3, 3, 1, 1, M1);

    float m2[]={     -1, 0,       1,
                -sqrt(2), 0, sqrt(2),
                     -1, 0,       1 };
    Mat M2=Mat(3,3,CV_32F,m2).clone();
    Kernel Mask2(3, 3, 1, 1, M2);

    float gradient;
    for (int sI = 0; sI <= src.rows-1; sI++)
    {
        for (int sJ = 0; sJ <= src.cols-1; sJ++)
        {
            gradient=FreiChenGradient(src, Mask1, Mask2, sI,
sJ);
            if(gradient>=threshold)
```

```
                res.at<uchar>(sI,sJ)=0;
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
}               res.at<uchar>(sI,sJ)=0;
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
}
```

Threshold 取 30 的處理結果：

(e) Kirsch's Compass Operator

Kirsch: set of eight compass template edge masks

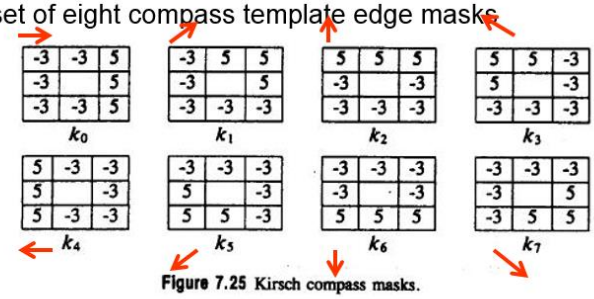


Figure 7.25 Kirsch compass masks.

- gradient magnitude: $g = \max_{n,n=0,\ldots,7} k_n$
- gradient direction: $\theta = 45^\circ \arg\max k_n$

```
double KirschGradient( const Mat src, const vector<Mat> M,
int sI, int sJ )
{
    float gradient=0;
    float *k;
    k=new float(M.size());
    for (int i = 0; i <= M.size()-1; i++)
    {
        Kernel Mask(M[i].rows, M[i].cols, (M[i].rows+1)/2-
1, (M[i].rows+1)/2-1, M[i]);
        k[i]=0;
        for (int MaskI = 0; MaskI <= Mask.kRows-1; MaskI++)
        {
            for (int MaskJ = 0; MaskJ <= Mask.kCols-1;
MaskJ++)
            {
                int sX=sI+(MaskI-Mask.anchorX);
                int sY=sJ+(MaskJ-Mask.anchorY);
                if (sX>=0 && sX<=src.rows-1 &&
                    sY>=0 && sY<=src.cols-1)
                {
                    k[i]=k[i]+(int)src.at<uchar>(sX,sY) *
(float)Mask.values.at<float>(MaskI, MaskJ);
                }
                else
                    return 0;
            }
        }
    }

    gradient=ArrayMax(k, M.size());
    //cout<<gradient<<endl;
    //waitKey();
    return gradient;
}

void Kirsch(const Mat src, Mat res, int threshold)
{
```

```
    //kernal
    vector<Mat> M;
    float m0[]={-3,-3, 5, -3, 0, 5, -3,-3, 5 };
    Mat M0(3,3,CV_32F, m0); //Mat
M0=Mat(3,3,CV_32F,m0).clone();
    M.push_back(M0);
    //Kernel Mask0(3, 3, 1, 1, M0);

    float m1[]={-3, 5, 5, -3, 0, 5, -3,-3,-3 };
    Mat M1=Mat(3,3,CV_32F,m1).clone();
    M.push_back(M1);
    float m2[]={5,5, 5, -3, 0, -3, -3,-3,-3 };
    Mat M2=Mat(3,3,CV_32F,m2).clone();
    M.push_back(M2);
    float m3[]={5,5, -3, 5, 0, -3, -3,-3, -3 };
    Mat M3=Mat(3,3,CV_32F,m3).clone();
    M.push_back(M3);
    float m4[]={5,-3, -3, 5, 0, -3, 5,-3, -3 };
    Mat M4=Mat(3,3,CV_32F,m4).clone();
    M.push_back(M4);
    float m5[]={-3,-3, -3, 5, 0, -3, 5,5, -3 };
    Mat M5=Mat(3,3,CV_32F,m5).clone();
    M.push_back(M5);
    float m6[]={-3,-3, -3, -3, 0, -3, 5, 5, 5 };
    Mat M6=Mat(3,3,CV_32F,m6).clone();
    M.push_back(M6);
    float m7[]={-3,-3, -3, -3, 0, 5, -3, 5, 5 };
    Mat M7=Mat(3,3,CV_32F,m7).clone();
    M.push_back(M7);

    float gradient;
    for (int sI = 0; sI <= src.rows-1; sI++)
    {
        for (int sJ = 0; sJ <= src.cols-1; sJ++)
        {
            gradient=KirschGradient(src, M, sI, sJ);
            if(gradient>=threshold)
                res.at<uchar>(sI,sJ)=0;
```

```
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
}
```

Threshold 取 135 的處理結果：

(f) Robinson's Compass Operator

Robinson: compass template mask set with only $0, \pm 1, \pm 2$

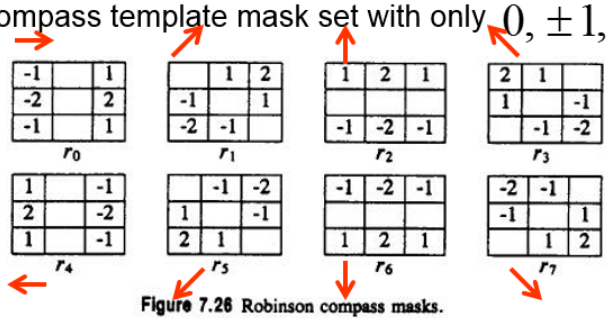


Figure 7.26 Robinson compass masks.

- done by only four masks since negation of each mask is also a mask
- gradient magnitude and direction same as Kirsch operator

$$g = \max_{n,n=0,\ldots,7} r_n \qquad \theta = 45^\circ \arg\max r_n$$

```
double RobinsonGradient( const Mat src, const vector<Mat>
M, int sI, int sJ )
{
    float gradient=0;
```

```
    float *r;
    r=new float(M.size());
    for (int i = 0; i <= M.size()-1; i++)
    {
        Kernel Mask(M[i].rows, M[i].cols, (M[i].rows+1)/2-
1, (M[i].rows+1)/2-1, M[i]);
        r[i]=0;
        for (int MaskI = 0; MaskI <= Mask.kRows-1; MaskI++)
        {
            for (int MaskJ = 0; MaskJ <= Mask.kCols-1;
MaskJ++)
            {
                int sX=sI+(MaskI-Mask.anchorX);
                int sY=sJ+(MaskJ-Mask.anchorY);
                if (sX>=0 && sX<=src.rows-1 &&
                    sY>=0 && sY<=src.cols-1)
                {
                    r[i]=r[i]+(int)src.at<uchar>(sX,sY) *
(float)Mask.values.at<float>(MaskI, MaskJ);
                }
                else
                    return 0;
            }
        }
    }

    gradient=ArrayMax(r, M.size());
    //cout<<gradient<<endl;
    //waitKey();
    return gradient;
}

void Robinson(const Mat src, Mat res, int threshold)
{
    //kernal
    vector<Mat> M;
    float m0[]={-1,0,1,-2,0,2,-1,0,1};
    Mat M0(3,3,CV_32F, m0); //Mat
```

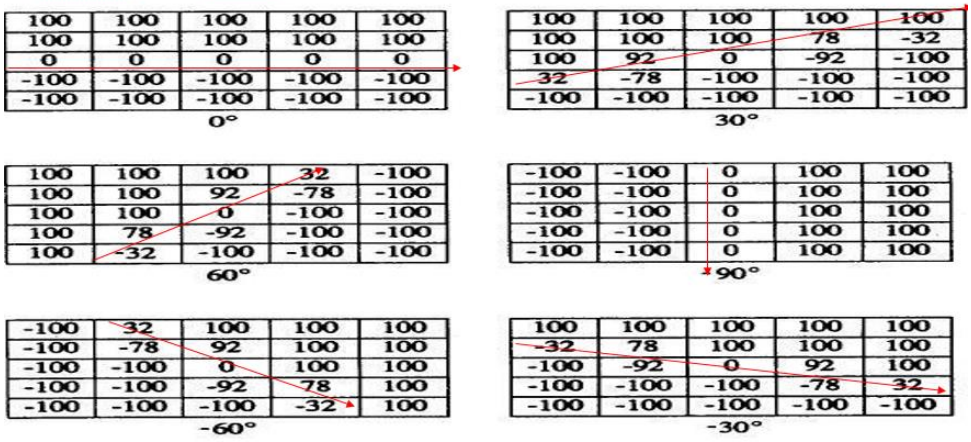
```
M0=Mat(3,3,CV_32F,m0).clone();
    M.push_back(M0);
    //Kernel Mask0(3, 3, 1, 1, M0);

    float m1[]={0,1,2,-1,0,1,-2,-1,0};
    Mat M1=Mat(3,3,CV_32F,m1).clone();
    M.push_back(M1);
    float m2[]={1,2,1,0,0,0,-1,-2,-1};
    Mat M2=Mat(3,3,CV_32F,m2).clone();
    M.push_back(M2);
    float m3[]={2,1,0,1,0,-1,0,-1,-2};
    Mat M3=Mat(3,3,CV_32F,m3).clone();
    M.push_back(M3);
    float m4[]={1,0,-1,2,0,-2,1,0,-1};
    Mat M4=Mat(3,3,CV_32F,m4).clone();
    M.push_back(M4);
    float m5[]={0,-1,-2,1,0,-1,2,1,0};
    Mat M5=Mat(3,3,CV_32F,m5).clone();
    M.push_back(M5);
    float m6[]={-1,-2,-1,0,0,0,1,2,1};
    Mat M6=Mat(3,3,CV_32F,m6).clone();
    M.push_back(M6);
    float m7[]={-2,-1,0,-1,0,1,0,1,2};
    Mat M7=Mat(3,3,CV_32F,m7).clone();
    M.push_back(M7);

    float gradient;
    for (int sI = 0; sI <= src.rows-1; sI++)
    {
        for (int sJ = 0; sJ <= src.cols-1; sJ++)
        {
            gradient=RobinsonGradient(src, M, sI, sJ);
            if(gradient>=threshold)
                res.at<uchar>(sI,sJ)=0;
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
```

```
}
```

Threshold 取 43 的處理結果：

(g) Nevatia-Babu 5x5 Operator

- Nevatia and Babu: set of six 5X5 compass template masks




```
double NevatiaBabuGradient( const Mat src, const
vector<Mat> M, int sI, int sJ )
{
    float gradient=0;
    float *r;
    r=new float(M.size());
    for (int i = 0; i <= M.size()-1; i++)
```

```
    {
        Kernel Mask(M[i].rows, M[i].cols, (M[i].rows+1)/2-
1, (M[i].rows+1)/2-1, M[i]);
        r[i]=0;
        for (int MaskI = 0; MaskI <= Mask.kRows-1; MaskI++)
        {
            for (int MaskJ = 0; MaskJ <= Mask.kCols-1;
MaskJ++)
            {
                int sX=sI+(MaskI-Mask.anchorX);
                int sY=sJ+(MaskJ-Mask.anchorY);
                if (sX>=0 && sX<=src.rows-1 &&
                    sY>=0 && sY<=src.cols-1)
                {
                    r[i]=r[i]+(int)src.at<uchar>(sX,sY) *
(float)Mask.values.at<float>(MaskI, MaskJ);
                }
                else
                    return 0;
            }
        }
    }

    gradient=ArrayMax(r, M.size());
    //cout<<gradient<<endl;
    //waitKey();
    return gradient;
}

void NevatiaBabu(const Mat src, Mat res, int threshold)
{
    //kernal
    vector<Mat> M;
    float m0[]={100,100,100,100,100,
                100,100,100,100,100,
                0,0,0,0,0,
                -100,-100,-100,-100,-100,
                -100,-100,-100,-100,-100 };
```

```
    Mat M0(5,5,CV_32F, m0); //Mat
M0=Mat(3,3,CV_32F,m0).clone();
    M.push_back(M0);

    float m1[]={100,100,100,100,100,
                100,100,100,78,-32,
                100,92,0,-92,-100,
                32,-78,-100,-100,-100,
                -100,-100,-100,-100,-100 };
    Mat M1(5,5,CV_32F, m1);
    M.push_back(M1);

    float m2[]={100,100,100,32,-100,
                100,100,92,-78,-100,
                100,100,0,-100,-100,
                100,78,-92,-100,-100,
                100,-32,-100,-100,-100 };
    Mat M2(5,5,CV_32F, m2);
    M.push_back(M2);

    float m3[]={-100,-100,0,100,100,
                -100,-100,0,100,100,
                -100,-100,0,100,100,
                -100,-100,0,100,100,
                -100,-100,0,100,100 };
    Mat M3(5,5,CV_32F, m3);
    M.push_back(M3);

    float m4[]={-100,32,100,100,100,
                -100,-78,92,100,100,
                -100,-100,0,100,100,
                -100,-100,-92,78,100,
                -100,-100,-100,-32,100 };
    Mat M4(5,5,CV_32F, m4);
    M.push_back(M4);

    float m5[]={100,100,100,100,100,
                -32,78,100,100,100,
```

```
                -100,-92,0,92,100,
                -100,-100,-100,-78,32,
                -100,-100,-100,-100,-100 };
    Mat M5(5,5,CV_32F, m5);
    M.push_back(M5);

    float gradient;
    for (int sI = 0; sI <= src.rows-1; sI++)
    {
        for (int sJ = 0; sJ <= src.cols-1; sJ++)
        {
            gradient=NevatiaBabuGradient(src, M, sI, sJ);
            if(gradient>=threshold)
                res.at<uchar>(sI,sJ)=0;
            else
                res.at<uchar>(sI,sJ)=255;
        }
    }
}
```

Threshold 取 12500 的處理結果：