GDV Praktikum 5

Gruppe : Ziton / Neumann / Neumann
Unser vollständiger Code ist unter : https://github.com/panexe/GDV_P5 zu finden.

Aufgabe 1 : Liniendetektion mit Hough-Räumen

Zur Implementierung von der Hough-Transformation werden die Trigonometrischen Funktionen sinus und cosinus im Grad-Maß benötigt. Die cmath Bibliothek liefert allerdings nur Funktionen im Winkelmaß. Um dieses Problem zu lösen füllen wir 2 LUTs mit den Winkel von 0 bis 360 Grad. Die Umrechnung von Rad zu Grad funktioniert über eine Multiplikation mit `deg2grad`.

```
static double sinLUT[360], cosLUT[360];
const static double deg2grad = 0.0174533;

void fillLUT() {
    for (int i = 0; i < 360; i++) {
        sinLUT[i] = sin(deg2grad * i);
        cosLUT[i] = cos(deg2grad * i);
    }
}
```

Dank der eigenschaften von sinus und cosinus können wir für sin(-x) und cos(-x) jeweils -sin(x) und -cos(x) zurückgeben. Dazu folgende Helfer-Funktionen :

```
double sine(int angle) {
    if (abs(angle) < 360) angle = angle % 360;

    if (angle < 0) {
        angle = abs(angle);
        return -1*sinLUT[angle];
    }
    return sinLUT[angle];
}

double cosine(int angle) {
    if (abs(angle) < 360) angle = angle % 360;

    if (angle < 0) {
        angle = abs(angle);
        return -1*cosLUT[angle];
```

```
    }
    return cosLUT[angle];
}
```

Schritte :

1. Umwandlung in Grayscale:

```
cv::Mat img_gray;
cv::cvtColor(img, img_gray, cv::COLOR_BGR2GRAY);
```

2. Canny Edge Detection
   Hierbei haben wir durch ausprobieren einen Threshold-Wert von 150 und ein Verhältnis von 1:2.5 gewählt.

```
cv::Mat edges;
double threshold = 150;
cv::Canny(img_gray, edges, threshold, threshold * 2.5);
```

3. Berechnung der Diagonalen
   Die Diagonale kann man einfach mit dem Satz des Pytagoras berechnen.

```
int diag = round(cv::sqrt((edges.rows * edges.rows) + (edges.cols * edges.cols)));
```

4. Akkumulatorarray mit 0 initialisieren

```
Mat acc = cv::Mat::zeros(180, 2*diag+1, CV_32S);
```

5. Berechnung aller möglichen geraden durch einen Kantenpixel
   Hierbei iterieren wir erstmal über das Ergebnis aus Schritt 2 und schauen ob der Wert für den Pixel ungleich 0 ist. Falls dies der Fall ist tragen wir für jeden Winkel im Bereich von -90 bis 89 Grad die Parameter d und theta im Akkumulatorarray ein. Dadurch dass das Array den Wertebereich {-90, 89} in einer Dimension und {-diag, diag} in der anderen hat, man jedoch ein Array nicht mit negativen Index ansprechen kann, addieren wir an dieser Stelle 90 und diag zu den Werten und ziehen es an späterer Stelle wieder ab.

```
for (int y = 0; y < edges.rows; y++) {
    for (int x = 0; x < edges.cols; x++) {
        if ((int)edges.at<char>(y, x) != 0) {
            for (int a = -90; a < 90; a++) {
                int d = (x * cosine(a)) + (y * sine(a));
```

```
                acc.at<int>(a+90, d + diag) += 1;
            }
        }
    }
}
```

6. siehe Schritt 5
7. Ermitteln der 5 höchsten Einträge
   Zum Ermitteln der höchsten Werte erstellen wir ein Array mit 5 einträgen und schreiben beim Iterieren über den Parameterraum die höchsten Werte ein.

```
int max_vals[5] = { 0,0,0,0,0 };
std::pair<int, int> max_vals_content[5];

for (int i = 0; i < 180; i++) {
    for (int j = 0; j < 2*diag+1; j++) {
        for (int c = 0; c < 5; c++) {
            if (max_vals[c] < acc.at<int>(i, j)) {
                max_vals[c] = acc.at<int>(i, j);
                max_vals_content[c] = std::pair<int, int>(i-90, j-diag);
                break;
            }
        }
    }
}
```

Zuletzt bleibt noch das Zeichnen der resultierenden Linien auf das Ursprungsbild. Hierzu verwenden wir einfach die gegebene Formel.

```
int x0 = 0, x1 = edges.rows - 1;
for (int i = 0; i < 5; i++) {
    int d = max_vals_content[i].second;
    int angle = max_vals_content[i].first;
    int y0, y1;
    if (angle == 0) {
        y0 = 0;
        y1 = img.rows - 1;
    }
    else {
        y0 = (d - x0 * cosine(angle)) / sine(angle);
        y1 = (d - x1 * cosine(angle)) / sine(angle);
    }
```

```
        cv::line(img, cv::Point(x0, y0), cv::Point(x1, y1), cv::Scalar(255, 255, 0));
    }
```

Der Akkumulator als Bild zeigt folgendes :

Das letzendliche Resultat :

Aufgabe 2 : Erosion und Dilatation

Dilation : 1 Iteration

Erosion : 1 Iteration

Dilation : 10 Iterationen

Erosion : 10 Iterationen

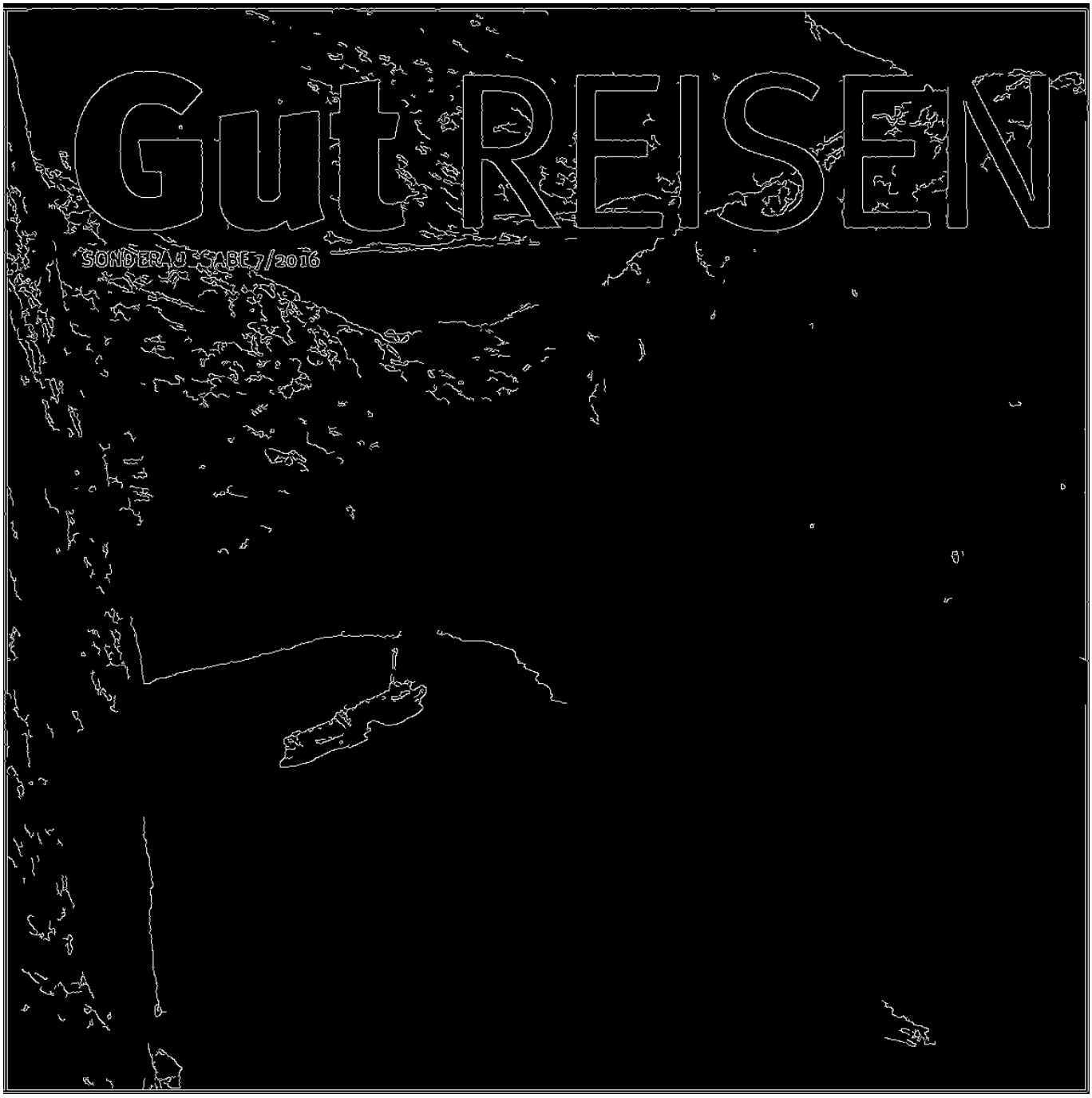
1. Was passiert, wenn Sie die Operationen mehrfach hintereinander ausführen?
   Der Effekt wird immer stärker dargestellt und nach einer gewissen Zeit werden wahrscheinlich alle Werte weiß oder schwarz.
2. Wann lässt sich das Ursprungsobjekt nicht mehr korrekt wiederherstellen? Warum?
   Das Ursprungsobjekt lässt sich nicht mehr korrekt wiederherstellen, sobald eine Information wie z.B. das Loch in der Mitte des 'O' verschwindet. Solange es noch vorhanden ist können die Pixel durch eine Errosion oder Dilatation wieder geweitet oder verengt werden. Wenn diese Infomation jedoch grundlegend verloren geht, kann sie nicht wieder zurück geholt werden.

Aufgabe 3 : Gut reisen

Um die Schrift aus dem Bild 'Gut reisen' zu extrahieren haben wir zuerst HoughLines mit Hilfe von Canny Edge Detection angewendet.

```
cv::Mat img_gray, dst;
cv::cvtColor(img, img_gray, cv::COLOR_BGR2GRAY);
cv::Mat edges;
double threshold = 150;
cv::Canny(img_gray, edges, threshold, threshold * 2.5);

vector<Vec2f> lines;
HoughLines(edges, lines, 1, CV_PI / 180, 200, 0, 0);
```



Danach filtern wir alle Linien, die für uns nicht relevant sind. Diese sind hierbei leicht zu filtern, weils wir uns erstmal nur für horizontale Linien interessieren. Dass heißt wir können die Differenz der y-Werte der zwei Linienpunkte betrachten und wenn dieser unter einem threshold ist, ist die Linie mehr oder weniger horizontal. Des weiteren können wir alle Linien die zu nah an einem der Bildränder sind filtern. Von den resultierenden Punkten speichern wir lediglich den tiefsten und höchsten Punkt im Bild um die obere und untere Kante des Schriftzuges zu erhalten.

```
Point bottom0, bottom1, top0, top1;
bottom0.x = bottom1.x = bottom0.y = bottom1.y = 10000000;
top0.x = top1.x = top0.y = top1.y = 0;
```

```
for (size_t i = 0; i < lines.size(); i++)
{
    float rho = lines[i][0], theta = lines[i][1];
    Point pt1, pt2;
    double a = cos(theta), b = sin(theta);
    double x0 = a * rho, y0 = b * rho;
    pt1.x = cvRound(x0 + 1000 * (-b));
    pt1.y = cvRound(y0 + 1000 * (a));
    pt2.x = cvRound(x0 - 1000 * (-b));
    pt2.y = cvRound(y0 - 1000 * (a));

    if (abs(pt1.x - pt2.x) < 100) {
        continue;
    }
    if ((pt1.y < 30) || (abs(pt1.y - img.cols) < 50)) {
        continue;
    }
    if ((pt2.y < 30) || (abs(pt2.y - img.cols) < 50)) {
        continue;
    }
    if (pt1.y <= bottom0.y) {
        bottom0 = pt1;
        bottom1 = pt2;
    }
    if (pt1.y >= top0.y) {
        top0 = pt1;
        top1 = pt2;
    }
}
```

An diesem Punkt können wir den Bildbereich mit dem Schriftzug nehmen und den Rest des Bildes ignorieren.

```
cv::Rect region(Point(0, bottom0.y), Point(img.cols, top0.y));
Mat subregion = img(region).clone();
```

Als letzte Berechungsschritte konvertieren wir das Bild zu Grayscale und wenden eine Threshold-Operation darauf an. Damit setzten wir alle Werte über dem Threshold auf weiß und unter dem Threshold auf schwarz. Dies funktioniert, weil der Schrittzug ganz weiß ist, im Gegensatz zu dem restlichen Bild.

Die überbleibenden Artefakte reduzieren wir über eine anreihung von Dilatation, Erosion, Erosion und wieder eine Dilatatition. Zuletzt wird noch einmal ein Gaußscher-Weichzeichner

verwendet um die Kanten zu glätten.

```
cvtColor(subregion, subregion, COLOR_BGR2GRAY);
cv::threshold(subregion, subregion, 230, 255, THRESH_BINARY);
cv::dilate(subregion, subregion, cv::Mat());
cv::erode(subregion, subregion, cv::Mat());
cv::erode(subregion, subregion, cv::Mat());
cv::dilate(subregion, subregion, cv::Mat());
cv::GaussianBlur(subregion, subregion, cv::Size(3, 3), 1);
```