Echtzeitbildverarbeitung SoSe 2015
Prof. Dr.: Udo Frese

Annika Ofenloch - 2992807 - ofenloch@uni-bremen.de
Frank Ihle - 3010158 - fihle@uni-bremen.de
Simon Schirrmacher - 4000884 - simons@informatik.uni-bremen
Noshaba Cheema - ncheema@uni-bremen.de

Übungsblatt

Abgabe am: 05.06.15

Aufgabe 1: Ballspiele I (10 Punkte)

```
1
2  HoughCircle::Parameters::Parameters()
3   :rMin(10), rMax(15), sobelThreshold(7), houghThreshold(7), radiusHoughThreshold(0), localMaxRange
      (100)
4  {
5    // so that findBestRadius and other functions do not have a buffer overflow
6    if (rMin < 0) rMin = 0;
7    if (rMax < 0) rMax = 0;
8  }
9
10 void HoughCircle::create (int width, int height, const Parameters &param)
11 {
12     // implement (2P)
13   imgWidth = width;
14   imgHeight = height;
15   // + 2 * RMAX to not to exceed edges of image
16   houghImgWidth = width + (param.rMax << 1);
17   houghImgHeight = height + (param.rMax << 1);
18   // create dummy image to determine step size
19   Mat_<ushort> dummyImg(houghImgHeight, houghImgWidth);
20   houghImgWidthStep = dummyImg.step[0] / dummyImg.step[1];
21   //deallocate memory
22   dummyImg.release();
23   // init sobel angle LUT
24   sobelTab.clear();
25   float angle, ang;
26   for (int y = -128; y < 128; ++y) for (int x = -128; x < 128; ++x) {
27     ang = atan2(y, x);
28     angle = fabs(ang) * 255.0f / M_PI; // normalize and discretize angle
29     sobelTab.push_back(SobelEntry((int)round(sqrt(sq(x) + sq(y))), (int) round(angle), cos(ang), sin(
         ang)));
30   }
31   // init relative address LUT
32   relativeAddressForAngleAndR.clear();
33   int dX, dY;
34   for (int angle = 0; angle < Parameters::NR_OF_ORIENTATIONS; ++angle) for (int r = 0; r <= param.rMax
       ; ++r){
35     dX = r * cos(angle);
36     dY = r * sin(angle);
37     relativeAddressForAngleAndR.push_back(dX + houghImgWidthStep * dY);
38   }
39   // error handling
40   assertSobelTab();
41     assertRelativeAddressForAngleAndRTab();
42 }
43
44 void HoughCircle::addPointToAccumulator (ushort* houghImgOrigin, int x, int y, int sobelCoded) const
45 {
46     // implement (1P)
47   int relAddr;
48   for (int r = param.rMin; r <= param.rMax; ++r) {
49     relAddr = relativeAddressForAngleAndR[sobelTab[sobelCoded].angle*(param.rMax+1)+r];
50     // along normal
51     houghImgOrigin[y * houghImgWidthStep + x + relAddr]++;
52     // opposed normal
53     houghImgOrigin[y * houghImgWidthStep + x - relAddr]++;
54   }
55 }
```

```
1  void HoughCircle::hough (Mat_<ushort>& houghImg, const Mat_<ushort>& sobelImgPrev, const Mat_<ushort>&
       sobelImg) const
2  {
3      // implement (1P)
4    // error handling
5    assert("sobelPrev and sobel must have the same size" && sobelImgPrev.rows == sobelImg.rows &&
       sobelImgPrev.cols == sobelImg.cols);
6    int sobelLen, sobelLenPrev;
7    // calculate origin of hough image
8    ushort* origin = houghImg.ptr<ushort>(param.rMax)+param.rMax;
9    // for more performance
10   #pragma omp for
11   for (int y = 0; y < sobelImg.rows; ++y) {
12     for (int x = 0; x < sobelImg.cols; ++x) {
13       // calculate sobel length of previous sobelImg
14       sobelLenPrev = sobelTab[sobelImgPrev(y, x)].length;
15       // calculate sobel length of sobelImg
16       sobelLen = sobelTab[sobelImg(y, x)].length;
17       // wasn't sure if > or >= since in line 97 in the header "exceed" is written
18       // but in line 22 it says it ignore every value below sobelThreshold
19       if ((sobelLen - sobelLenPrev) > param.sobelThreshold)
20         addPointToAccumulator(origin, x, y, sobelImg(y,x));
21     }
22   }
23 }
24
25
26 bool HoughCircle::isLocalMaximum (const Mat_<ushort>& houghImg, int xC, int yC) const
27 {
28     // implement (2P)
29   const ushort *hLine = nullptr;
30   const ushort center = houghImg.ptr<ushort>(yC)[xC];
31   // [+localMaxRange,-localMaxRange]
32   for (int dy = -param.localMaxRange; dy <= param.localMaxRange; ++dy) {
33     // check if no edge exceeding
34     if (yC + dy >= 0 && yC + dy < houghImgHeight){
35       hLine = houghImg.ptr<ushort>(yC);
36       hLine = houghImg.ptr<ushort>(yC + dy);
37       for (int dx = -param.localMaxRange; dx <= param.localMaxRange; ++dx) {
38         // check if no edge exceeding and return false if greater value was found
39         if (xC + dx >= 0 && xC + dx < houghImgWidth &&
40           center < hLine[xC + dx])
41           return false;
42       }
43     }
44   }
45   return true;
46 }
```

```
1  int HoughCircle::findBestRadius (const Mat_<ushort>& sobelImg, int xC, int yC, int& bestR) const
2  {
3      // implement (2P)
4
5    // init hough R
6    std::vector<int> houghR;
7    for (int r = 0; r <= param.rMax; ++r)
8      houghR.push_back(0);
9
10   // increase houghR at r if fitting value was found
11   SobelEntry s;
12   int r; float d;
13   for (int dY = -param.rMax; dY <= param.rMax; dY++)
14     for (int dX = -param.rMax; dX <= param.rMax; dX++) {
15       // check for edge exceeding
16       if (xC + dX >= 0 && xC + dX < sobelImg.cols &&
17         yC + dY >= 0 && yC + dY < sobelImg.rows) {
18         s = sobelTab[sobelImg(yC + dY, xC + dX)];
19         // calculate radius
20         r = fabs(s.cosAngle *dX + s.sinAngle*dY);
21         d = fabs(-s.sinAngle*dX + s.cosAngle*dY);
22         if (d <= 1 && s.length > param.sobelThreshold && r <= param.rMax)
23           houghR[r]++; // increase if good value
24       }
25     }
26
27   // find best radius and valueR
28   bestR = param.rMin;
29   int bestVal = 0;
30   for (int r = param.rMin; r <= param.rMax; r++) {
31     if (houghR[r]>houghR[bestR]) {
32       bestR = r;
33       bestVal = houghR[r];
34     }
35   }
36
37   return bestVal;
38 }
39
40
41 void HoughCircle::extractFromHoughImage (vector<Circle>& circles, const Mat_<ushort>& houghImg, const
     Mat_<ushort>& sobelImg) const
42 {
43     // implement (1P)
44   circles.clear();
45   const ushort* pLine = nullptr;
46   int bestR;
47   int valueR;
48   // starts at +RMAX and ends at end-RMAX because the hough room is 2*RMAX in both dimensions
49   // greater than the original image
50   for (int y = param.rMax; y < houghImgHeight - param.rMax; ++y) {
51     pLine = houghImg.ptr<ushort>(y);
52     for (int x = param.rMax; x < houghImgWidth - param.rMax; ++x) {
53       // find a high enough entry as center
54       if (pLine[x] >= param.houghThreshold && isLocalMaximum(houghImg, x, y)) {
55         valueR = findBestRadius(sobelImg, x, y, bestR);
56         // entry has to be at least radiusHoughThreshold (line 25 in header)
57         if (valueR >= param.radiusHoughThreshold)
58           // calculate hough img coordinated to original img coordinates
59           circles.push_back(Circle(x - param.rMax, y - param.rMax, bestR, pLine[x], valueR));
60       }
61     }
62   }
63 }
```

```
1
2  void HoughCircle::findCircles (vector<Circle> &circles, Mat_<ushort>& houghImg, const Mat_<ushort>&
     sobelPrev, const Mat_<ushort>& sobel) const
3  {
4      // implement (1P)
5    // error handling
6    assert("sobel images must be the same size" && sobel.cols == sobelPrev.cols && sobel.rows ==
       sobelPrev.rows);
7    circles.clear();
8    // create hough image if not allocated
9    if (houghImg.empty())
10     houghImg = createHoughImage();
11   #pragma omp parallel
12   {
13     // apply hough transform
14     hough(houghImg, sobelPrev, sobel);
15   }
16   // create circle out of transform
17   extractFromHoughImage(circles, houghImg, sobel);
18 }
```

Aufgabe 2: Tag des Modellbaus (4 Punkte)

Unter Verwendung der Bildverarbeitung soll ein Zeppelinmodell, autonom eine vordefinierte Route über einer Wiese fliegen.

Um die Orientierung des Zeppelins auf der Wiese zu ermöglichen, werden Markierungen auf der Wiese ausgelegt, welche als Wegpunkte zu interpretieren sind. Die Markierungen haben die Form eines Pfeils und geben somit Aufschluss über die zu fliegende Richtung.Um diese erfassen zu können, wird unter dem Zeppelin eine Kamera angebracht, die senkrecht nach unten filmt (siehe Abb. 1).

Abbildung 1: Beispiel eines Markierungspfeils für die Bewegungsvorgabe des Zeppelins.

Für den Start wird eine besondere runde Markierung eingesetzt. Zu Beginn des Rundflugs befindet sich der Zeppelin direkt auf dem Kreis und schwebt beim Start senkrecht in die Höhe.
Anhand der Größe des Kreises, welche mit zunehmender Höhe immer weiter abnimmt, kann der Zeppelin feststellen, ob bereits die gewünschte Höhe erreicht wurde.

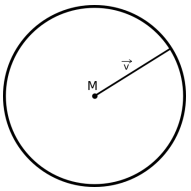

Abbildung 2: Start-Markierung

Ist diese Höhe erreicht, orientiert sich der Zeppelin an einem Strich, welcher vom Kreisrand bis zur Mitte des Kreises gezeichnet ist. Dieser Strich gibt die initiale Richtung an (siehe Abb. 2). Jede durch die Markierungen definierte Richtung zeigt auf den Mittelpunkt (Schwerpunkt) der nächsten Markierung. Die Markierungen, welche die Wegpunkte beschreiben, sind einfarbig gefärbt und es ist eine bestimmte, sich wiederholende, Farbfolge definiert, wodurch bei mehreren erkannten Markierungen die nächst folgende ausgewählt werden kann (siehe Abb. 3).

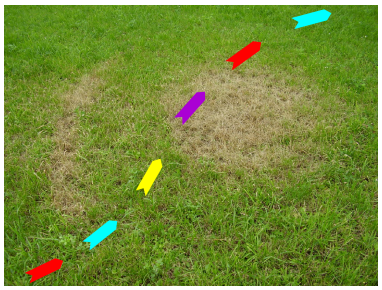

Abbildung 3: Beispiel einer Farbfolge

Zudem sind sie so geformt, dass nicht die gesamte Markierung im Bild erfasst sein muss, um die vorgegebene Richtung zu erfassen. Um dies zu erreichen, sind Anfang und Ende des Pfeils so geformt, dass diese eindeutig erkennbar sind (siehe Abb. 1).

Wird eine Markierung erkannt, richtet sich der Zeppelin entsprechend seine Vorgabe auf dem Boden aus. Von diesem Markierungspfeil werden die Trägheitsachsen gebildet und anhand der Pfeilspitzen die Bewegungsrichtung vorgegeben. Mit Hilfe dieser Trägheitsachse und der Geraden, auf der sich das Flugobjekt vorwärts bewegt, lässt sich über trigonometrische Funktionen den Winkel ermitteln, den es einzustellen gilt, um die Bewegungsrichtung zu ändern. Ist bereits eine neue Markierung im Blickfeld (neu bedeutet: in der oberen Bildhälfte), dann soll deren Mittelpunkt angesteuert werden. Wenn nicht wird entsprechend der Vorgaben der zuletzt gefundenen Markierung geradeaus geflogen. Erst wenn die angesteuerte Markierung in der unteren Bildhälfte liegt (also hinter dem Zeppelin), wird auf die nächst verfügbare navigiert.

Beim Einbau der Kamera ist davon auszugehen, dass deren vertikalen und horizontalen Achsen nicht senkrecht bzw. parallel zur Bewegungsrichtung verlaufen, wofür die Kamera vor Inbetriebnahme zunächst kalibriert werden muss.

Da nach diesen Vorgaben oft nachgesteuert wird, empfiehlt es sich die Bewegungsvorgabe mit einem Regler anzusteuern, um eventuellen Schwingzuständen entegegen zu wirken.

Für die Landung wird ein weiterer Kreis als Markierung eingesetzt, in welchen jedoch kein Strich eingezeichnet sein muss. Es kann sich bei diesem Kreis auch um den selben Kreis handeln, von dem aus gestartet wurde. Wird dieser Kreis erkannt, sinkt der Zeppelin senkrecht.

Aufgabe 3: Spargel, Bildverarbeitung und soziale Realität (1 Bonuspunkt)

Qualitätsmerkmale, wie beispielsweise die Farbe (bzw. Verfärbung), Vollständigkeit als auch die Form, lassen sich durch Angestellte schnell kontrollieren. Es kann demnach überprüft werden, ob der Spargel von Schädlingen befallen, verfault, verschmutzt, abgebrochen, krumm oder hohl ist.

Bei den Merkmalen wie, Länge und Durchmesser reicht das Augenmaß nicht immer aus. Insbesondere beim Durchmesser ist es schwierig die präzise Unterscheidung in die einzelnen Güteklassen durchzuführen, da es sich hier oft um Abweichungen von Millimetern handelt. Der Angestellte hat demnach Schwierigkeiten zu beurteilen, ob der Spargel einen Durchmesser von 8 oder 10 mm aufweist (Mindestdurchmesser bei Klasse I: 10 mm, bei Klasse II: 8 mm).

Auch die Länge des Spargels kann durch Angestellte nicht immer mit bloßen Augenmaß richtig eingeschätzt werden. Sobald sich die Länge in der Nähe von Grenzwerten befindet, ist es besonders wichtig, die exakte Länge zu bestimmen. Wenn der Spargel zum Beispiel 17,3 cm lang ist, zählt er zum langen und nicht mehr zum kurzen Spargel. Ein Angestellter würde dies gegebenenfalls nicht sehen und den Spargel falsch zuordnen. Es ist in solchen Fällen also wichtig, wenn sich der Angestellte nicht sicher ist, den Spargel an ein Bildverarbeitungssystem weiter zu geben. Dort kann der genaue Durchmesser als auch die Länge des Spargels ermittelt werden. Ansonsten ist der Angestellte in der Lage mit bloßem Auge erkennen zu können, ob es sich beispielsweise, um ein sehr kleines Exemplar handelt.