```
// ************************************************************************
//
// Project:    Alignment and Gauging for industrial parts.
// Copyright:  Tilo Gockel (Author)
// Date:       February 25th 2007
// Filename:   main.cpp
// Author:     Tilo Gockel, Chair Prof. Dillmann (IAIM),
//             Institute for Computer Science and Engineering (ITEC/CSE),
//             University of Karlsruhe. All rights reserved.
//
// ************************************************************************
//
// Description:
// Program searches *.jpg-Files in the current directory. Then:
// calculation of center of gravity and principal axis for alignment,
// Then gauging (measurement) of a given distance,
//
//
// Algorithms:
// Spatial moments, central moments,
// calculation of direction of major axis,
// gauging (counting pixels to next b/w change), in [Pixels].
//
//
// Comments:
// OS: Windows 2000 or XP; Compiler: MS Visual C++ 6.0,
// Libs used: IVT, QT, OpenCV.
//
// ************************************************************************


#include "Image/ByteImage.h"
#include "Image/ImageAccessCV.h"
#include "Image/ImageProcessor.h"
#include "Image/ImageProcessorCV.h"
#include "Image/PrimitivesDrawer.h"
#include "Image/PrimitivesDrawerCV.h"
#include "Image/IplImageAdaptor.h"
#include "Math/Constants.h"
#include "Helpers/helpers.h"
#include "gui/QTWindow.h"
#include "gui/QTApplicationHandler.h"

#include <cv.h>

#include <qstring.h>
#include <qstringlist.h>
#include <qdir.h>

#include <iostream>
#include <iomanip>
#include <windows.h>
#include <string.h>
#include <math.h>



using namespace std;



// modified version of DrawLine(): returns sum of visited non-black pixels
```



```
// (but here also used for line-drawing)
int WalkTheLine(CByteImage *pImage, const Vec2d &p1, const Vec2d &p2,
   int r, int g, int b)
{
   int pixelcount = 0;
   const double dx = p1.x - p2.x;
   const double dy = p1.y - p2.y;

   if (fabs(dy) < fabs(dx))
   {
      const double slope = dy / dx;
      const int max_x = int(p2.x + 0.5);
      double y = p1.y + 0.5;

      if (p1.x < p2.x)
      {
         for (int x = int(p1.x + 0.5); x <= max_x; x++, y += slope)
            {
            if (pImage->pixels[int(y) * pImage->width + x] != 0)
               pixelcount++;
            PrimitivesDrawer::DrawPoint(pImage, x, int(y), r, g, b);
            }
      }
      else
      {
         for (int x = int(p1.x + 0.5); x >= max_x; x--, y -= slope)
         {
            if (pImage->pixels[int(y) * pImage->width + x] != 0)
               pixelcount++;
            PrimitivesDrawer::DrawPoint(pImage, x, int(y), r, g, b);
         }
      }
   }
   else
   {
      const double slope = dx / dy;
      const int step = (p1.y < p2.y) ? 1 : -1;
      const int max_y = int(p2.y + 0.5);
      double x = p1.x + 0.5;

      if (p1.y < p2.y)
      {
         for (int y = int(p1.y + 0.5); y <= max_y; y++, x += slope)
         {
            if (pImage->pixels[y * pImage->width + int(x)] != 0)
               pixelcount++;
            PrimitivesDrawer::DrawPoint(pImage, int(x), y, r, g, b);
         }
      }
      else
      {
         for (int y = int(p1.y + 0.5); y >= max_y; y--, x -= slope)
         {
            if (pImage->pixels[int(y) * pImage->width + int(x)] != 0)
               pixelcount++;
            PrimitivesDrawer::DrawPoint(pImage, int(x), y, r, g, b);
         }
      }
   }
   return pixelcount;
}
```

```
void MomentCalculations(CByteImage *pImage, Vec2d &center,
                  PointPair2d &orientation, double &theta)
{
   // calculate moments
   IplImage *pIplInputImage = IplImageAdaptor::Adapt(pImage);
   CvMoments moments;
   cvMoments(pIplInputImage, &moments, 1); //1: treat grayvalues != 0 as 1
   cvReleaseImageHeader(&pIplInputImage);

   // for center of gravity
   const double m00 = cvGetSpatialMoment(&moments, 0, 0);
   const double m01 = cvGetSpatialMoment(&moments, 0, 1);
   const double m10 = cvGetSpatialMoment(&moments, 1, 0);

   // for angle of major axis
   const double u11 = cvGetCentralMoment(&moments, 1, 1);
   const double u20 = cvGetCentralMoment(&moments, 2, 0);
   const double u02 = cvGetCentralMoment(&moments, 0, 2);

   theta = 0.0;

   // now: case differentiation:
   // cmp.: [Johannes Kilian 01], Simple Image Analysis by Moments]
   // online: http://serdis.dis.ulpgc.es/~itis-fia/FIA/doc/Moments/OpenCv/
   // but: STILL AMBIGUOUS in n * 180 Degrees !

   if (  ((u20 - u02) == 0) && (u11 == 0) )     // 1
      theta = 0.0;
   if (  ((u20 - u02) == 0) && (u11 >  0) )     // 2
      theta = PI / 4.0;
   if (  ((u20 - u02) == 0) && (u11 <  0) )     // 3
      theta = - (PI / 4.0);
   if (  ((u20 - u02) >  0) && (u11 == 0) )     // 4
      theta = 0.0;
   if (  ((u20 - u02) <  0) && (u11 == 0) )     // 5
      theta = - (PI /2);
   if (  ((u20 - u02) >  0) && (u11 >  0) )     // 6
      theta = 0.5 * atan(2 * u11 / (u20 - u02));
   if (  ((u20 - u02) >  0) && (u11 <  0) )     // 7
      theta = 0.5 * atan(2 * u11 / (u20 - u02));
   if (  ((u20 - u02) <  0) && (u11 >  0) )     // 8
      theta = (0.5 * atan(2 * u11 / (u20 - u02))) + PI / 2;
   if (  ((u20 - u02) <  0) && (u11 <  0) )     // 9
      theta = (0.5 * atan(2 * u11 / (u20 - u02))) - PI / 2;

   Math2d::SetVec(center, m10 / m00, m01 / m00);

   // now: determine direction of major axis
   // go cross-like, start from COG, go to borders
   // count pixels... (cmp. visualization)

   Vec2d v;

   v.x = cos(theta) * 250 + center.x;
   v.y = sin(theta) * 250 + center.y;
   int count1 = WalkTheLine(pImage, center, v, 255, 0, 0);

   v.x = cos(theta + PI) * 230 + center.x;
   v.y = sin(theta + PI) * 230 + center.y;
   int count2 = WalkTheLine(pImage, center, v, 255, 255, 0);
```



```
   v.x = cos(theta + PI/2) * 230 + center.x;
   v.y = sin(theta + PI/2) * 230 + center.y;
   int count3 = WalkTheLine(pImage, center, v, 128, 0, 0);

   v.x = cos(theta - PI/2) * 230 + center.x;
   v.y = sin(theta - PI/2) * 230 + center.y;
   int count4 = WalkTheLine(pImage, center, v, 64, 0, 0);

   if ((count1 > count2) && (count3 < count4))
      theta = theta + PI;

   // Optional / for debugging: Console output
   // cout << "Area: " << m00 << endl;
   // cout << "Center (x,y): " << center.x << " "  << center.y << endl;
   // cout << "Theta [DEG]: " << ((theta * 180.0) / PI)  << endl << endl;
}



int main(int argc, char *argv[])
{
   double theta = 0.0;

   QString path = QDir::currentDirPath();
   QDir dir(path);
   QStringList files = dir.entryList("*.jpg", QDir::Files);

   if (files.empty())
   {
      cout << "Error: could not find any *.jpg Files" << endl;
      return 1;
   }

   QStringList::Iterator it = files.begin();
   QString buf = QFileInfo(path, *it).baseName();
   buf += ".jpg";

   CQTApplicationHandler qtApplicationHandler(argc, argv);
   qtApplicationHandler.Reset();

   // width, height must be multiples of 4 (!)
   CByteImage colorimage;
   if (!ImageAccessCV::LoadFromFile(&colorimage, buf.ascii()))
   {
      printf("error: could not open input image file\n");
      return 1;
   }

   CByteImage grayimage(colorimage.width, colorimage.height,
      CByteImage::eGrayScale);
   CByteImage binaryimage(colorimage.width, colorimage.height,
      CByteImage::eGrayScale);

   ImageProcessor::ConvertImage(&colorimage, &grayimage);


   // calculations in grayimage and binaryimage
   // drawings and writings in colorimage for display

   CQTWindow imgwindow1(colorimage.width, colorimage.height);
   imgwindow1.DrawImage(&colorimage);
```

```
   imgwindow1.Show();

   CQTWindow imgwindow2(binaryimage.width, binaryimage.height);
   imgwindow2.DrawImage(&binaryimage);
   imgwindow2.Show();


   // main loop: cyclic loading all *.jpg in the directory and processing

   while (!qtApplicationHandler.ProcessEventsAndGetExit())
   {
      buf = QFileInfo(path, *it).baseName();
      buf += ".jpg";
      cout << buf.ascii() << endl;

      if (!ImageAccessCV::LoadFromFile(&colorimage, buf.ascii()))
      {
         printf("error: could not open input image file\n");
         return 1;
      }

      // Inversion: OpenCV calculates Moments for _white_ objects!
      ImageProcessor::ConvertImage(&colorimage, &grayimage);
      ImageProcessor::Invert(&grayimage, &grayimage); // (!)
      ImageProcessor::ThresholdBinarize(&grayimage, &binaryimage, 128);

      // Moments...
      Vec2d center;
      PointPair2d orientation;
      MomentCalculations(&binaryimage, center, orientation, theta);


      // Visualization / Output:
      // Center
      PrimitivesDrawerCV::DrawCircle(&colorimage, center, 3, 0, 255, 0, -1);


      // Two Lines to show coordinate system
      Vec2d v1, v2;
      v1.x = cos(theta) * 100 + center.x;
      v1.y = sin(theta) * 100 + center.y;
      WalkTheLine(&colorimage, center, v1, 255, 0, 0);

      v1.x = cos(theta + PI/2) * 100 + center.x;
      v1.y = sin(theta + PI/2) * 100 + center.y;
      WalkTheLine(&colorimage, center, v1, 255, 255, 0);


      ImageProcessor::Rotate(&binaryimage, &binaryimage, center.x, center.y,
         theta, true);


      // we gauge the cross section near the minor axis
      // (going parallel to the minor axis):
      v1.x = center.x+5;
      v1.y = center.y - 200;
      v2.x = center.x+5;
      v2.y = center.y + 200;
      int i = WalkTheLine(&binaryimage, v1, v2, 255, 255, 255);

      cout << "Gauging after alignment [pixel]: " << i << endl << endl;
```



```
      char text[512];
      sprintf(text, "Cross section in pixels: %d", i);

      PrimitivesDrawerCV::PutText(&colorimage, text, 20, 60, 0.8, 0.8,
         255, 0, 100, 1);

      imgwindow1.DrawImage(&colorimage);
      imgwindow2.DrawImage(&binaryimage);

      //Sleep(1200); // oops, too fast to see anything....

      ++it;
      if (it == files.end()) it = files.begin();  // until hell freezes over
   }

   return 0;
}
```