```
/**
 * @file   auto_c_detect_node.cpp
 * @author Johanna Gleichauf
 * @date   Stand 04.05.2017
 *
 *
 */
#include <cstdlib>
#include <stdlib.h>
#include <stdio.h>
#include <iostream>
#include "opencv2/core/core.hpp"
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/calib3d/calib3d.hpp"
//#include "opencv2/nonfree/nonfree.hpp"
#include <opencv2/opencv.hpp>
#include "opencv2/imgproc/imgproc.hpp"
#include <ros/ros.h>
#include <sensor_msgs/Image.h>
#include <sensor_msgs/image_encodings.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <math.h>

#include <sensor_msgs/PointCloud.h>
#include <sensor_msgs/PointCloud2.h>
#include <sensor_msgs/point_field_conversion.h>
#include <sensor_msgs/point_cloud_conversion.h>
#include <sensor_msgs/PointField.h>
#include "tf/message_filter.h"
#include <message_filters/subscriber.h>
#include <laser_geometry/laser_geometry.h>
#include "tf/transform_datatypes.h"
#include "tf/transform_broadcaster.h"
#include "tf/transform_listener.h"

using namespace cv;
using namespace std;

Mat src_unten; Mat src_unten_gray;
Mat src_links; Mat src_links_gray;
Mat src_rechts; Mat src_rechts_gray;
Mat src_oben; Mat src_oben_gray;
Mat src_los; Mat src_los_gray;
Mat src_lus; Mat src_lus_gray;
Mat src_ros; Mat src_ros_gray;
Mat src_rus; Mat src_rus_gray;
Mat img; Mat img_gray;
int thresh = 100;
int max_thresh = 255;


bool u_min=false, o_min=false, r_min=false, l_min=false, ro_min=false, ru_min=false, lo_min=false, lu_min=false;
bool u_mid=false, o_mid=false, r_mid=false, l_mid=false, ro_mid=false, ru_mid=false, lo_mid=false, lu_mid=false;
double max_area=0;
double min_area=10000;
double mid_area;

vector<vector<Point> > g_detected_contours;


RNG rng(12345);

/// Function header
void thresh_callback(int, void* );

cv::Mat _Input;
cv::Mat _InputM;
cv::Mat thermo_bin;

void callCam(const sensor_msgs::Image camImage)
{
  std::cout <<"callback " << std::endl;
  cv_bridge::CvImagePtr frame;
  frame = cv_bridge::toCvCopy(camImage, sensor_msgs::image_encodings::BGR8);
  _Input=frame->image;
};

void callThermoCam(const sensor_msgs::Image thermal_image)
{
  cv_bridge::CvImagePtr thermo_frame;
  thermo_frame = cv_bridge::toCvCopy(thermal_image, sensor_msgs::image_encodings::MONO16);
  _InputM=thermo_frame->image;
}

/** @function main */
int main( int argc, char** argv )
{
  ros::init(argc, argv, "camera");
  ros::NodeHandle nh;

  image_transport::ImageTransport it(nh);
  sensor_msgs::ImagePtr msg;


  //Topic Realsense D415
  ros::Subscriber imgSub = nh.subscribe<sensor_msgs::Image>("/camera/color/image_raw",1, callCam);



  while((_Input.rows == 0)){//}&&(_InputM.rows==0)){
       std::cout << "No data" << std::endl;
       ros::spinOnce();
     }

  /// Load source image and convert it to gray - C unten
    src_unten = imread( "./auto_c_detect/C_unten.jpg", 1 );

    if(! src_unten.data )                              // Check for invalid input
        {
            cout <<  "Could not open or find the image" << std::endl ;
            return -1;
        }
    /// Load source image and convert it to gray - C links
    src_links = imread( "./auto_c_detect/C_links.jpg", 1 );

    if(! src_links.data )                              // Check for invalid input
    {
      cout <<  "Could not open or find the image" << std::endl ;
      return -1;
    }
   /// Load source image and convert it to gray - C rechts
    src_rechts = imread( "./auto_c_detect/C_rechts.jpg", 1 );

    if(! src_rechts.data )                              // Check for invalid input
```

```
{
  cout <<  "Could not open or find the image" << std::endl ;
  return -1;
}

  /// Load source image and convert it to gray - C oben
src_oben = imread( "./auto_c_detect/C_oben.jpg", 1 );

if(! src_oben.data )                              // Check for invalid input
{
  cout <<  "Could not open or find the image" << std::endl ;
  return -1;
}

  /// Load source image and convert it to gray - C oben links schraeg
src_los = imread( "./auto_c_detect/C_links_45.jpg", 1 );

if(! src_los.data )                              // Check for invalid input
{
  cout <<  "Could not open or find the image" << std::endl ;
  return -1;
}

/// Load source image and convert it to gray - C links unten schraeg
src_lus = imread( "./auto_c_detect/C_links_unten_45.jpg", 1 );

if(! src_lus.data )                              // Check for invalid input
{
  cout <<  "Could not open or find the image" << std::endl ;
  return -1;
}

/// Load source image and convert it to gray - C rechts oben schraeg
src_ros = imread( "./auto_c_detect/C_rechts_45.jpg", 1 );

if(! src_ros.data )                              // Check for invalid input
{
  cout <<  "Could not open or find the image" << std::endl ;
  return -1;
}

/// Load source image and convert it to gray - C rechts oben schraeg
```

```
  src_rus = imread( "./auto_c_detect/C_rechts_unten_45.jpg", 1 );

  if(! src_rus.data )                              // Check for invalid input
  {
    cout <<  "Could not open or find the image" << std::endl ;
    return -1;
  }



//Convert _Input to gray

ros::Rate loop_rate(30);
while(ros::ok())
{






std::cout << "Thermal 8" << std::endl;
//RGB Image
cvtColor(_Input,  img_gray, CV_BGR2GRAY);
blur( img_gray, img_gray, Size(3,3) );



/// Convert image to gray and blur it
cvtColor( src_unten, src_unten_gray, CV_BGR2GRAY );
blur( src_unten_gray, src_unten_gray, Size(3,3) );
/// Convert image to gray and blur it
cvtColor( src_links, src_links_gray, CV_BGR2GRAY );
blur( src_links_gray, src_links_gray, Size(3,3) );
/// Convert image to gray and blur it
cvtColor( src_rechts, src_rechts_gray, CV_BGR2GRAY );
blur( src_rechts_gray, src_rechts_gray, Size(3,3) );
/// Convert image to gray and blur it
cvtColor( src_oben, src_oben_gray, CV_BGR2GRAY );
blur( src_oben_gray, src_oben_gray, Size(3,3) );
/// Convert image to gray and blur it
cvtColor( src_los, src_los_gray, CV_BGR2GRAY );
blur( src_los_gray, src_los_gray, Size(3,3) );
```

```
  /// Convert image to gray and blur it
  cvtColor( src_lus, src_lus_gray, CV_BGR2GRAY );
  blur( src_lus_gray, src_lus_gray, Size(3,3) );
  /// Convert image to gray and blur it
  cvtColor( src_ros, src_ros_gray, CV_BGR2GRAY );
  blur( src_ros_gray, src_ros_gray, Size(3,3) );
  /// Convert image to gray and blur it
  cvtColor( src_rus, src_rus_gray, CV_BGR2GRAY );
  blur( src_rus_gray, src_rus_gray, Size(3,3) );



  /// Create Window
  char* source_window = "Source";
  namedWindow( source_window, CV_WINDOW_AUTOSIZE );
  imshow( source_window, img_gray);
//  imshow(source_window, thermo_bin);

  createTrackbar( " Canny thresh:", "Source", &thresh, max_thresh, thresh_callback );
  thresh_callback( 0, 0 );

//  waitKey(0); //auskommentieren damit dauerhaft checkt
  if (waitKey(30) >= 0)
  break;
  ros::spinOnce();
  loop_rate.sleep();

  }
}

/** @function thresh_callback */
void thresh_callback(int, void* )
{
  Mat canny_output;
//  Mat canny_thermal;
  Mat canny_unten_fixed;
  Mat canny_links_fixed;
  Mat canny_rechts_fixed;
  Mat canny_oben_fixed;
  Mat canny_los_fixed;
  Mat canny_lus_fixed;
  Mat canny_ros_fixed;
```

```
    Mat canny_rus_fixed;

    vector<vector<Point> > contours;
//  vector<vector<Point> > contours_thermal;
    vector<vector<Point> > contours_unten_fixed;
    vector<vector<Point> > contours_links_fixed;
    vector<vector<Point> > contours_rechts_fixed;
    vector<vector<Point> > contours_oben_fixed;
    vector<vector<Point> > contours_los_fixed;
    vector<vector<Point> > contours_lus_fixed;
    vector<vector<Point> > contours_ros_fixed;
    vector<vector<Point> > contours_rus_fixed;
    vector<Vec4i> hierarchy;


//Reference image

//  Detect edges using canny
    Canny( src_unten_gray, canny_unten_fixed, thresh, thresh*2, 3 );
    /// Find contours
    findContours( canny_unten_fixed, contours_unten_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

    Canny( src_links_gray, canny_links_fixed, thresh, thresh*2, 3 );
    //  std::cout << "before find contours " << std::endl;
      /// Find contours
    findContours( canny_links_fixed, contours_links_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

    Canny(src_rechts_gray, canny_rechts_fixed, thresh, thresh*2,3);
    findContours(canny_rechts_fixed, contours_rechts_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));

    Canny(src_oben_gray, canny_oben_fixed, thresh, thresh*2,3);
    findContours(canny_oben_fixed, contours_oben_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));

    Canny(src_los_gray, canny_los_fixed, thresh, thresh*2,3);
    findContours(canny_los_fixed, contours_los_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));

    Canny(src_lus_gray, canny_lus_fixed, thresh, thresh*2,3);
    findContours(canny_lus_fixed, contours_lus_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));

    Canny(src_ros_gray, canny_ros_fixed, thresh, thresh*2,3);
    findContours(canny_ros_fixed, contours_ros_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));
```

```
Canny(src_rus_gray, canny_rus_fixed, thresh, thresh*2,3);
findContours(canny_rus_fixed, contours_rus_fixed, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));


//RGB Image
Canny(img_gray, canny_output, thresh, thresh*2,3);
findContours(canny_output, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));



/// Get the moments
//Moments C unten
vector<Moments> mu_u(contours_unten_fixed.size() );
for( unsigned int i = 0; i < contours_unten_fixed.size(); i++ )
   { mu_u[i] = cv::moments(contours_unten_fixed[i], false);

   }

//Moments C links
vector<Moments> mu_l(contours_links_fixed.size() );
for( unsigned int i = 0; i < contours_links_fixed.size(); i++ )
   { mu_l[i] = cv::moments(contours_links_fixed[i], false);

   }

//Moments C links
vector<Moments> mu_r(contours_rechts_fixed.size() );
for( unsigned int i = 0; i < contours_rechts_fixed.size(); i++ )
{ mu_r[i] = cv::moments(contours_rechts_fixed[i], false);

}

//Moments C oben
vector<Moments> mu_o(contours_oben_fixed.size() );
for( unsigned int i = 0; i < contours_oben_fixed.size(); i++ )
{ mu_o[i] = cv::moments(contours_oben_fixed[i], false);

}

//Moments C links oben schraeg
 vector<Moments> mu_los(contours_los_fixed.size() );
 for( unsigned int i = 0; i < contours_los_fixed.size(); i++ )
```

```
   { mu_los[i] = cv::moments(contours_los_fixed[i], false);

   }

   //Moments C links unten schraeg
   vector<Moments> mu_lus(contours_lus_fixed.size() );
   for( unsigned int i = 0; i < contours_lus_fixed.size(); i++ )
   { mu_lus[i] = cv::moments(contours_lus_fixed[i], false);

   }

   //Moments C recht oben schraeg
   vector<Moments> mu_ros(contours_ros_fixed.size() );
   for( unsigned int i = 0; i < contours_ros_fixed.size(); i++ )
   { mu_ros[i] = cv::moments(contours_ros_fixed[i], false);

   }

   //Moments C recht oben schraeg
   vector<Moments> mu_rus(contours_rus_fixed.size() );
   for( unsigned int i = 0; i < contours_rus_fixed.size(); i++ )
   { mu_rus[i] = cv::moments(contours_rus_fixed[i], false);

   }


  //Moments incoming image camera
  vector<Moments> mu(contours.size() );
  for( unsigned int i = 0; i < contours.size(); i++ )
     { mu[i] = cv::moments(contours[i], false);

     }





//  for(unsigned int i=0; i<contours.size();i++)
//    {
//      std::cout << "Central moment 1 camera " << abs(mu[i].nu20) << std::endl;
//      std::cout << "Central moment 2 camera " << abs(mu[i].nu11) << std::endl;
//      std::cout << "Central moment 3 camera " << abs(mu[i].nu02) << std::endl;
//      std::cout << "Central moment 4 camera " << abs(mu[i].nu30) << std::endl;
//      std::cout << "Central moment 5 camera " << abs(mu[i].nu21) << std::endl;
//      std::cout << "Central moment 6 camera " << abs(mu[i].nu12) << std::endl;
//      std::cout << "Central moment 7 camera " << abs(mu[i].nu03) << std::endl;
//
//    }


  double area_u, area_l, area_r, area_o, area_lu, area_lo, area_ru, area_ro;
  int x=0.0;
  double num[3]= {0,0,0};
  double nam[3]= {0,0,0};
  bool u_max=false, o_max=false, r_max=false, l_max=false, ro_max=false, ru_max=false, lo_max=false, lu_max=false;

  //C unten Detektion

  g_detected_contours.clear();

  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_unten_fixed.size(); j++)
//      if((abs(mu[i].nu20-mu_u[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_u[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_u[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_u[j].nu30)<0.0123)&& (abs(mu[i].nu21-mu_u[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_u[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_u[j].nu03)<0.0123))
       if((abs(mu[i].nu20-mu_u[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_u[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_u[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_u[j].nu30)<0.0115)&& (abs(mu[i].nu21-mu_u[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_u[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_u[j].nu03)<0.0115))

      {
        std::cout << "C unten detected " << std::endl;
        if(!u_max)
        {
//          std::cout << "C unten detected " << std::endl;
          std::cout << "diff unten " << abs(mu[i].nu20-mu_u[j].nu20) << std::endl;
          area_u = contourArea(contours[i]);
          std::cout << "Area unten " << area_u << std::endl;
          num[x] = area_u;
          nam[x] = 1;

          x++;
          u_max = true;
          g_detected_contours.push_back(contours[i]);
        }
//        if(area_u > max_area)
//        {
//          max_area = area_u;
//          u_max = true;
//
//          o_max=false, r_max=false, l_max=false, ro_max=false, ru_max=false, lo_max=false, lu_max=false;
//
//
//        }
//        else
//          if(area_u < min_area)
//        {
//          min_area=area_u;
//          u_min = true;
//          o_min=false, r_min=false, l_min=false, ro_min=false, ru_min=false, lo_min=false, lu_min=false;
//
//        }
//        else
//         if((area_u > min_area) &&(area_u < max_area))
//         {
//           mid_area = area_u;
//           u_mid = true;
//           u_min = false;
//           u_max = false;
//           o_mid=false, r_mid=false, l_mid=false, ro_mid=false, ru_mid=false, lo_mid=false, lu_mid=false;
//
//         }
      }
  }




  //C links Detektion
  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_links_fixed.size();j++)
//      if((abs(mu[i].nu20-mu_l[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_l[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_l[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_l[j].nu30)<0.0123) && (abs(mu[i].nu21-mu_l[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_l[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_l[j].nu03)<0.0123))
        if((abs(mu[i].nu20-mu_l[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_l[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_l[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_l[j].nu30)<0.0115) && (abs(mu[i].nu21-mu_l[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_l[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_l[j].nu03)<0.0115))

      {
        std::cout << "C links detected " << std::endl;
        if(!l_max)
        {

//          std::cout << "C links detected " << std::endl;
          std::cout << "diff links " << abs(mu[i].nu20-mu_l[j].nu20) << std::endl;
          area_l = contourArea(contours[i]);
          std::cout << "Area links " << area_l << std::endl;
          num[x] = area_l;
          nam[x] = 2;

          x++;
          l_max = true;
          g_detected_contours.push_back(contours[i]);
        }
//        if(area_l > max_area)
//        {
//          max_area = area_l;
//          l_max = true;
//          u_max=false, o_max=false, r_max=false, ro_max=false, ru_max=false, lo_max=false, lu_max=false;
//        }
//        else
//          if(area_l < min_area)
//        {
//          min_area = area_l;
//          l_min = true;
//          u_min = false, o_min=false, r_min=false, ro_min=false, ru_min=false, lo_min=false, lu_min=false;
//        }
//        else
//          if((area_l > min_area) && (area_l < max_area))
//          {
//            mid_area = area_l;
//            l_mid = true;
//            l_min = false;
//            l_max = false;
//            u_mid=false, o_mid=false, r_mid=false, ro_mid=false, ru_mid=false, lo_mid=false, lu_mid=false;
//          }
      }
```

```
  }



  //C rechts Detektion
  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_links_fixed.size();j++)
//      if((abs(mu[i].nu20-mu_r[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_r[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_r[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_r[j].nu30)<0.0123) && (abs(mu[i].nu21-mu_r[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_r[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_r[j].nu03)<0.0123))
        if((abs(mu[i].nu20-mu_r[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_r[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_r[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_r[j].nu30)<0.0115) && (abs(mu[i].nu21-mu_r[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_r[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_r[j].nu03)<0.0115))

      {
        std::cout << "C rechts detected " << std::endl;
        if(!r_max)
        {
//          std::cout << "C rechts detected " << std::endl;
          std::cout << "diff rechts " << abs(mu[i].nu20-mu_r[j].nu20) << std::endl;
          area_r = contourArea(contours[i]);
          std::cout << "Area rechts " << area_r << std::endl;
          num[x] = area_r;
          nam[x] = 3;

          x++;
          r_max = true;
          g_detected_contours.push_back(contours[i]);
        }
//        if(area_r > max_area)
//        {
//          max_area = area_r;
//          r_max = true;
//          u_max=false, o_max=false, l_max=false, ro_max=false, ru_max=false, lo_max=false, lu_max=false;
//        }
//        else
//          if(area_r < min_area)
//        {
//          min_area = area_r;
//          r_min = true;
//          u_min = false, o_min=false, l_min=false, ro_min=false, ru_min=false, lo_min=false, lu_min=false;
//        }
//        else
//         if((area_r > min_area) && (area_r < max_area))
//         {
//           mid_area = area_r;
//           r_mid = true;
//           r_min = false;
//           r_max = false;
//           u_mid=false, o_mid=false, l_mid=false, ro_mid=false, ru_mid=false, lo_mid=false, lu_mid=false;
//         }
      }
  }



  //C oben Detektion
  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_oben_fixed.size();j++)
//      if((abs(mu[i].nu20-mu_o[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_o[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_o[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_o[j].nu30)<0.0123) && (abs(mu[i].nu21-mu_o[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_o[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_o[j].nu03)<0.0123))
        if((abs(mu[i].nu20-mu_o[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_o[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_o[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_o[j].nu30)<0.0115) && (abs(mu[i].nu21-mu_o[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_o[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_o[j].nu03)<0.0115))

      {
        std::cout << "C oben detected " << std::endl;
        if(!o_max)
        {
//          std::cout << "C oben detected " << std::endl;
          std::cout << "diff oben " << abs(mu[i].nu20-mu_o[j].nu20) << std::endl;
          area_o = contourArea(contours[i]);
          std::cout << "Area oben " << area_o << std::endl;
          num[x] = area_o;
          nam[x] = 4;

          x++;
          o_max = true;
          g_detected_contours.push_back(contours[i]);
        }
//        if(area_o > max_area)
//        {
//          max_area = area_o;
//          o_max = true;
//          u_max=false, r_max=false, l_max=false, ro_max=false, ru_max=false, lo_max=false, lu_max=false;
//        }
//        else
//          if(area_o < min_area)
//        {
//          min_area = area_o;
//          o_min = true;
//          u_min = false, r_min=false, l_min=false, ro_min=false, ru_min=false, lo_min=false, lu_min=false;
//        }
//        else
//         if((area_o > min_area)&&(area_o < max_area))
//         {
//           mid_area = area_o;
//           o_mid = true;
//           o_min = false;
//           o_max = false;
//           u_mid=false, l_mid=false, r_mid=false, ro_mid=false, ru_mid=false, lo_mid=false, lu_mid=false;
//         }
      }
  }




  //C links oben schraeg Detektion
  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_los_fixed.size();j++)
//      if((abs(mu[i].nu20-mu_los[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_los[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_los[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_los[j].nu30)<0.0123) && (abs(mu[i].nu21-mu_los[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_los[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_los[j].nu03)<0.0123))
        if((abs(mu[i].nu20-mu_los[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_los[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_los[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_los[j].nu30)<0.0115) && (abs(mu[i].nu21-mu_los[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_los[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_los[j].nu03)<0.0115))

      {
        std::cout << "C links oben schraeg detected " << std::endl;
        if(!lo_max)
        {
```

```
//          std::cout << "C links oben schraeg detected " << std::endl;
          std::cout << "diff links oben schraeg " << abs(mu[i].nu20-mu_los[j].nu20) << std::endl;
          area_lo = contourArea(contours[i]);
          std::cout << "Area links oben " << area_lo << std::endl;
          num[x] = area_lo;
          nam[x] = 5;

          x++;
          lo_max = true;
          g_detected_contours.push_back(contours[i]);
        }
//        if(area_lo > max_area)
//        {
//          max_area = area_lo;
//          lo_max = true;
//          u_max=false, r_max=false, l_max=false, ro_max=false, ru_max=false, o_max=false, lu_max=false;
//        }
//        else
//          if(area_lo < min_area)
//        {
//          min_area = area_lo;
//          lo_min = true;
//          u_min = false, r_min=false, l_min=false, ro_min=false, ru_min=false, o_min=false, lu_min=false;
//        }
//        else
//         if((area_lo > min_area)&&(area_lo < max_area))
//        {
//          mid_area = area_lo;
//          lo_mid = true;
//          lo_min = false;
//          lo_max = false;
//          u_mid=false, l_mid=false, r_mid=false, ro_mid=false, ru_mid=false, o_mid=false, lu_mid=false;
//        }
      }
  }




  //C links unten schraeg Detektion
  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_lus_fixed.size();j++)
//      if((abs(mu[i].nu20-mu_lus[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_lus[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_lus[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_lus[j].nu30)<0.0123) && (abs(mu[i].nu21-mu_lus[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_lus[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_lus[j].nu03)<0.0123))
        if((abs(mu[i].nu20-mu_lus[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_lus[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_lus[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_lus[j].nu30)<0.0115) && (abs(mu[i].nu21-mu_lus[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_lus[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_lus[j].nu03)<0.0115))

      {
        std::cout << "C links unten schraeg detected " << std::endl;
        if(!lu_max)
        {
//          std::cout << "C links unten schraeg detected " << std::endl;
          std::cout << "diff links unten schraeg " << abs(mu[i].nu20-mu_lus[j].nu20) << std::endl;
          area_lu = contourArea(contours[i]);
          std::cout << "Area links unten " << area_lu << std::endl;
          num[x] = area_lu;
          nam[x] = 6;

          x++;
          lu_max = true;
          g_detected_contours.push_back(contours[i]);
        }
//        if(area_lu > max_area)
//        {
//          max_area = area_lu;
//          lu_max = true;
//          u_max=false, r_max=false, l_max=false, ro_max=false, ru_max=false, o_max=false, lo_max=false;
//        }
//        else
//          if(area_lu < min_area)
//        {
//          min_area = area_lu;
//          lu_min = true;
//          u_min = false, r_min=false, l_min=false, ro_min=false, ru_min=false, o_min=false, lo_min=false;
//        }
//        else
//         if((area_lu > min_area) && (area_lu < max_area))
//         {
//           mid_area = area_lu;
//           lu_mid = true;
//           lu_min = false;
//           lu_max = false;
//           u_mid=false, l_mid=false, r_mid=false, ro_mid=false, ru_mid=false, o_mid=false, lo_mid=false;
//         }
      }
  }





  //C rechts oben schraeg Detektion
  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_ros_fixed.size();j++)
//      if((abs(mu[i].nu20-mu_ros[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_ros[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_ros[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_ros[j].nu30)<0.0123) && (abs(mu[i].nu21-mu_ros[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_ros[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_ros[j].nu03)<0.0123))
        if((abs(mu[i].nu20-mu_ros[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_ros[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_ros[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_ros[j].nu30)<0.0115) && (abs(mu[i].nu21-mu_ros[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_ros[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_ros[j].nu03)<0.0115))

      {
        std::cout << "C rechts oben schraeg detected " << std::endl;
        if(!ro_max)
        {
//          std::cout << "C rechts oben schraeg detected " << std::endl;
          std::cout << "diff rechts oben schraeg " << abs(mu[i].nu20-mu_ros[j].nu20) << std::endl;
          area_ro = contourArea(contours[i]);
          std::cout << "Area rechts oben " << area_ro << std::endl;
          num[x] = area_ro;
          nam[x] = 7;

          x++;
          ro_max = true;
          g_detected_contours.push_back(contours[i]);
        }
//        if(area_ro > max_area)
//        {
//          max_area = area_ro;
//          ro_max = true;
//          u_max=false, r_max=false, l_max=false, lu_max=false, ru_max=false, o_max=false, lo_max=false;
//        }
//        else
//          if(area_ro < min_area)
//        {
//          min_area = area_ro;
//          ro_min = true;
//          u_min = false, r_min=false, l_min=false, lu_min=false, ru_min=false, o_min=false, lo_min=false;
//        }
//        else
//          if((area_ro > min_area) && (area_ro < min_area) )
//          {
//            mid_area = area_ro;
//            ro_mid = true;
//            ro_min = false;
//            ro_max = false;
//            u_mid=false, l_mid=false, r_mid=false, lu_mid=false, ru_mid=false, o_mid=false, lo_mid=false;
//          }
      }
  }




  //C rechts unten schraeg Detektion
  for(unsigned int i = 0; i<contours.size();i++)
  {
    for(unsigned int j=0; j<contours_rus_fixed.size();j++)
//      if((abs(mu[i].nu20-mu_rus[j].nu20)<0.0123) && (abs(mu[i].nu11-mu_rus[j].nu11)<0.0123) && (abs(mu[i].nu02-
mu_rus[j].nu02)<0.0123) && (abs(mu[i].nu30-mu_rus[j].nu30)<0.0123) && (abs(mu[i].nu21-mu_rus[j].nu21)<0.0123) &&
(abs(mu[i].nu12-mu_rus[j].nu12)<0.0123) && (abs(mu[i].nu03-mu_rus[j].nu03)<0.0123))
        if((abs(mu[i].nu20-mu_rus[j].nu20)<0.0115) && (abs(mu[i].nu11-mu_rus[j].nu11)<0.0115) && (abs(mu[i].nu02-
mu_rus[j].nu02)<0.0115) && (abs(mu[i].nu30-mu_rus[j].nu30)<0.0115) && (abs(mu[i].nu21-mu_rus[j].nu21)<0.0115) &&
(abs(mu[i].nu12-mu_rus[j].nu12)<0.0115) && (abs(mu[i].nu03-mu_rus[j].nu03)<0.0115))

      {

        std::cout << "C rechts unten schraeg detected " << std::endl;
        std::cout << "diff rechts unten schraeg " << abs(mu[i].nu20-mu_rus[j].nu20) << std::endl;
        area_ru = contourArea(contours[i]);
        std::cout << "Area rechts unten " << area_ru << std::endl;
        num[x] = area_ru;
        nam[x] = 8;

        g_detected_contours.push_back(contours[i]);
```

```
//        if(area_ru > max_area)
//        {
//          max_area  = area_ru;
//          ru_max = true;
//          u_max=false, r_max=false, l_max=false, lu_max=false, ro_max=false, o_max=false, lo_max=false;
//        }
//        else
//          if(area_ru < min_area)
//        {
//          min_area = area_ru;
//          ru_min = true;
//          u_min = false, r_min=false, l_min=false, lu_min=false, ro_min=false, o_min=false, lo_min=false;
//        }
//        else
//          if((area_ru > min_area) && (area_ru < max_area) )
//          {
//            mid_area = area_ru;
//            ru_mid = true;
//            ru_min = false;
//            ru_max = false;
//            u_mid=false, l_mid=false, r_mid=false, lu_mid=false, ro_mid=false, o_mid=false, lo_mid=false;
//          }
      }
  }




  std::cout << " Before bubblesort " << std::endl;
  std::cout << "Num " << num[0] << " Nam " << nam[0] << std::endl;
  std::cout << "Num " << num[1] << " Nam " << nam[1] << std::endl;
  std::cout << "Num " << num[2] << " Nam " << nam[2] << std::endl;




//  Bubblesort - find maximum area, minimum area and middle sized area
  int flag = 1;    // set flag to 1 to start first pass
  double temp;            // holding variable
  int temp_nam;

  for(unsigned int i = 1; (i <= 3) && flag; i++)
  {
    flag = 0;
    for (unsigned int j=0; j < (3 -1); j++)
    {
      if (num[j+1] > num[j])      // ascending order simply changes to <
      {
        temp = num[j];             // swap elements
        num[j] = num[j+1];
        num[j+1] = temp;
        temp_nam = nam[j];
        nam[j] = nam[j+1];
        nam[j+1] = temp_nam;
        flag = 1;               // indicates that a swap occurred.
      }
    }
  }

//  Num[0] = maxArea, num[1] = midArea, num[2] = minArea -> nur nam[] ist interessant um herauszufinden welches C
  for(unsigned int x = 0; x<sizeof(nam);x++)
  {
   if(x==0)
   {
     std::cout << "Outer " << std::endl;
     if(nam[x]==1)
       std::cout << "C Bottom " << std::endl;
     else if(nam[x]==2)
       std::cout << "C Left " << std::endl;
     else if(nam[x]==3)
       std::cout << "C Right " << std::endl;
     else if(nam[x]==4)
       std::cout << "C Top " << std::endl;
     else if(nam[x]==5)
       std::cout << "C Left Top " << std::endl;
     else if(nam[x]==6)
       std::cout << "C Left Bottom " << std::endl;
     else if(nam[x]==7)
       std::cout << "C Right Top " << std::endl;
     else if(nam[x]==8)
       std::cout << "C Right Bottom " << std::endl;
   }
```

```
else
 if(x==1)
 {
   std::cout << "Mid " << std::endl;
   if(nam[x]==1)
     std::cout << "C Bottom " << std::endl;
   else if(nam[x]==2)
     std::cout << "C Left " << std::endl;
   else if(nam[x]==3)
     std::cout << "C Right " << std::endl;
   else if(nam[x]==4)
     std::cout << "C Top " << std::endl;
   else if(nam[x]==5)
     std::cout << "C Left Top " << std::endl;
   else if(nam[x]==6)
     std::cout << "C Left Bottom " << std::endl;
   else if(nam[x]==7)
     std::cout << "C Right Top " << std::endl;
   else if(nam[x]==8)
     std::cout << "C Right Bottom " << std::endl;
 }
else
  if(x==2)
  {
    std::cout << "Inner " << std::endl;
    if(nam[x]==1)
      std::cout << "C Bottom " << std::endl;
    else if(nam[x]==2)
      std::cout << "C Left " << std::endl;
    else if(nam[x]==3)
      std::cout << "C Right " << std::endl;
    else if(nam[x]==4)
      std::cout << "C Top " << std::endl;
    else if(nam[x]==5)
      std::cout << "C Left Top " << std::endl;
    else if(nam[x]==6)
      std::cout << "C Left Bottom " << std::endl;
    else if(nam[x]==7)
      std::cout << "C Right Top " << std::endl;
    else if(nam[x]==8)
      std::cout << "C Right Bottom " << std::endl;
      }


   }






//
//  next:
//  if(ru_max)
//    std::cout << "Outer C right bottom " << std::endl;
//  if(ro_max)
//    std::cout << "Outer C right top " << std::endl;
//  if(lu_max)
//    std::cout << "Outer C left bottom " << std::endl;
//  if(lo_max)
//    std::cout << "Outer C left top " << std::endl;
//  if(l_max)
//    std::cout << "Outer C left " << std::endl;
//  if(r_max)
//    std::cout << "Outer C right " << std::endl;
//  if(o_max)
//    std::cout << "Outer C top " << std::endl;
//  if(u_max)
//    std::cout << "Outer C bottom " << std::endl;
//
//  if(ru_min)
//    std::cout << "Inner C right bottom " << std::endl;
//  if(ro_min)
//    std::cout << "Inner C right top " << std::endl;
//  if(lu_min)
//    std::cout << "Inner C left bottom " << std::endl;
//  if(lo_min)
//    std::cout << "Inner C left top " << std::endl;
//  if(l_min)
//    std::cout << "Inner C left " << std::endl;
//  if(r_min)
//    std::cout << "Inner C right " << std::endl;
//  if(o_min)
//    std::cout << "Inner C top " << std::endl;
//  if(u_min)
//    std::cout << "Inner C bottom " << std::endl;
//
//  if(ru_mid)
//    std::cout << "Mid C right bottom " << std::endl;
//  if(ro_mid)
//    std::cout << "Mid C right top " << std::endl;
//  if(lu_mid)
//    std::cout << "Mid C left bottom " << std::endl;
//  if(lo_mid)
//    std::cout << "Mid C left top " << std::endl;
//  if(l_mid)
//    std::cout << "Mid C left " << std::endl;
//  if(r_mid)
//    std::cout << "Mid C right " << std::endl;
//  if(o_mid)
//    std::cout << "Mid C top " << std::endl;
//  if(u_mid)
//    std::cout << "Mid C bottom " << std::endl;
//





  ///  Get the mass centers:
  vector<Point2f> mc( contours.size() );
  for( int i = 0; i < contours.size(); i++ )
     { mc[i] = Point2f( mu[i].m10/mu[i].m00 , mu[i].m01/mu[i].m00 ); }

  /// Draw contours
  Mat drawing = Mat::zeros( canny_output.size(), CV_8UC3 );
  Scalar color = Scalar( 0, 0,255);
  for( int i = 0; i< contours.size(); i++ )
     {

       drawContours( drawing, contours, i, color, 2, 8, hierarchy, 0, Point() );
       circle( drawing, mc[i], 4, color, -1, 8, 0 );
     }
  /// Show in a window
  namedWindow( "Contours", CV_WINDOW_AUTOSIZE );
  imshow( "Contours", drawing );


  drawContours(_Input, g_detected_contours,-1, color, 3);
//  imshow("DetectedContours", _Input);

  std::cout << "Check 1 " << std::endl;

  //Find cirle around contour
  double cx;
  double cy;
  float radius;
  Point2f center;
//  Mat circle;
  Mat canny_circle;
  Mat canny_diff;
  Mat canny_roi;
  vector<vector<Point> > contour_circle;
  vector<vector<Point> > contour_diff;
  vector<vector<Point> > contour_roi;
  Scalar color_diff = Scalar( 0, 255,0);
  Mat C_area;
  Mat gap;

  std::cout << "Check 2 " << std::endl;

  for(int i=0; i< g_detected_contours.size(); i++)
  {
        minEnclosingCircle(g_detected_contours.at(i), center, radius);
        std::cout << "Center " << center << std::endl;
        std::cout << "Radius " << radius << std::endl;
        Rect r(center.x-radius, center.y-radius, radius*2, radius*2);
        circle(_Input,center,radius,(0,255,0),2);
//      Mat roi(_Input, r);
//      Canny(roi, canny_roi, thresh, thresh*2, 3);
//      findContours(canny_roi, contour_roi, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));
//      Mat mask(roi.size(), roi.type(), Scalar::all(0));
//      std::cout << "Check 3 " << std::endl;
//      circle(mask, Point(radius,radius), radius, Scalar::all(255), -1);
//      std::cout << "Check 3 a" << std::endl;
//      Mat diff = roi & mask;
//      std::cout << "Check 3 b " << std::endl;
//      Mat C_area = Mat(g_detected_contours.at(i));
//      std::cout << "Check 3 c" << std::endl;
//      subtract(diff, C_area, gap);
//      std::cout << "Check 4 " << std::endl;



//      Mat C_roi(_Input, g_detected_contours.at(i));
//      subtract(roi, C_roi, gap);
        //Find contours of enclosing circle
//      Canny(circle, canny_circle, thresh, thresh*2, 3);
//      findContours(canny_circle, contour_circle, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));
//      //Subtract C contour from circle contour
//      subtract(contour_circle.at(i), g_detected_contours.at(i), diff);
//      //Find contours of difference
//      Canny(gap, canny_diff, thresh, thresh*2,3);
//      findContours(canny_diff, contour_diff, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0,0));
//      drawContours(_Input, contour_diff, -1, color_diff, 3);
//      std::cout << "Check 5 " << std::endl;

  }
//  drawContours(_Input, contour_roi, -1, Scalar(0,0,255), CV_FILLED);
  drawContours(_Input, g_detected_contours,-1, Scalar(255), CV_FILLED);
  imshow("DetectedContours", _Input);



  /// Calculate the area with the moments 00 and compare with the result of the OpenCV function
//  printf("\t Info: Area and Contour Length \n");
  for( int i = 0; i< contours.size(); i++ )
     {
//       printf(" * Contour[%d] - Area (M_00) = %.2f - Area OpenCV: %.2f - Length: %.2f \n", i, mu[i].m00,
contourArea(contours[i]), arcLength( contours[i], true ) );
       Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
       drawContours( drawing, contours, i, color, 2, 8, hierarchy, 0, Point() );
       circle( drawing, mc[i], 4, color, -1, 8, 0 );
     }
}
```