Code

# Canny Edge Detector

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <stdlib.h>

#include <stdio.h>

**using** **namespace** cv;

*/// Global variables*

Mat src, src\_gray;

Mat dst, detected\_edges;

int edgeThresh = 1;

int lowThreshold;

int **const** max\_lowThreshold = 100;

int ratio = 3;

int kernel\_size = 3;

char\* window\_name = "Edge Map";

*/\*\**

*\* @function CannyThreshold*

*\* @brief Trackbar callback - Canny thresholds input with a ratio 1:3*

*\*/*

void CannyThreshold(int, void\*)

{

*/// Reduce noise with a kernel 3x3*

blur( src\_gray, detected\_edges, Size(3,3) );

*/// Canny detector*

Canny( detected\_edges, detected\_edges, lowThreshold, lowThreshold\*ratio, kernel\_size );

*/// Using Canny's output as a mask, we display our result*

dst = Scalar::all(0);

src.copyTo( dst, detected\_edges);

imshow( window\_name, dst );

}

*/\*\* @function main \*/*

int main( int argc, char\*\* argv )

{

*/// Load an image*

src = imread( argv[1] );

**if**( !src.data )

{ **return** -1; }

*/// Create a matrix of the same type and size as src (for dst)*

dst.create( src.size(), src.type() );

*/// Convert the image to grayscale*

cvtColor( src, src\_gray, CV\_BGR2GRAY );

*/// Create a window*

namedWindow( window\_name, CV\_WINDOW\_AUTOSIZE );

*/// Create a Trackbar for user to enter threshold*

createTrackbar( "Min Threshold:", window\_name, &lowThreshold, max\_lowThreshold, CannyThreshold );

*/// Show the image*

CannyThreshold(0, 0);

*/// Wait until user exit program by pressing a key*

waitKey(0);

**return** 0;

}

# contours

#include "opencv2/highgui/highgui.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

**using** **namespace** cv;

**using** **namespace** std;

Mat src; Mat src\_gray;

int thresh = 100;

int max\_thresh = 255;

RNG rng(12345);

*/// Function header*

void thresh\_callback(int, void\* );

*/\*\* @function main \*/*

int main( int argc, char\*\* argv )

{

*/// Load source image and convert it to gray*

src = imread( argv[1], 1 );

*/// Convert image to gray and blur it*

cvtColor( src, src\_gray, CV\_BGR2GRAY );

blur( src\_gray, src\_gray, Size(3,3) );

*/// Create Window*

char\* source\_window = "Source";

namedWindow( source\_window, CV\_WINDOW\_AUTOSIZE );

imshow( source\_window, src );

createTrackbar( " Canny thresh:", "Source", &thresh, max\_thresh, thresh\_callback );

thresh\_callback( 0, 0 );

waitKey(0);

**return**(0);

}

*/\*\* @function thresh\_callback \*/*

void thresh\_callback(int, void\* )

{

Mat canny\_output;

vector<vector<Point> > contours;

vector<Vec4i> hierarchy;

*/// Detect edges using canny*

Canny( src\_gray, canny\_output, thresh, thresh\*2, 3 );

*/// Find contours*

findContours( canny\_output, contours, hierarchy, CV\_RETR\_TREE, CV\_CHAIN\_APPROX\_SIMPLE, Point(0, 0) );

*/// Draw contours*

Mat drawing = Mat::zeros( canny\_output.size(), CV\_8UC3 );

**for**( int i = 0; i< contours.size(); i++ )

{

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() );

}

*/// Show in a window*

namedWindow( "Contours", CV\_WINDOW\_AUTOSIZE );

imshow( "Contours", drawing );

}