

Robotics: Principles and Practice

Module 5: Robot Vision

Lecture 5: Segmentation; boundary-based approaches; edge detection

David Vernon
Carnegie Mellon University Africa

www.vernon.eu

Boundary Detection

The usual approach to segmentation by boundary detection is to:

- Construct an **edge image** from the original grey-scale image
- Use this edge to construct the **boundary image** without reference to the original grey-scale data by edge linking to generate short curve segments

Boundary Detection

Boundary detection algorithms

- Use **domain-dependent information** or knowledge which they incorporate in associating or **linking the edges**
 - edge-thinning
 - gap-filling
 - curve segment linking
- Their effectiveness is dependent on the quality of the edge image

Edge Detection

- An approach to segmentation
- Based on the analysis of the discontinuities in an image

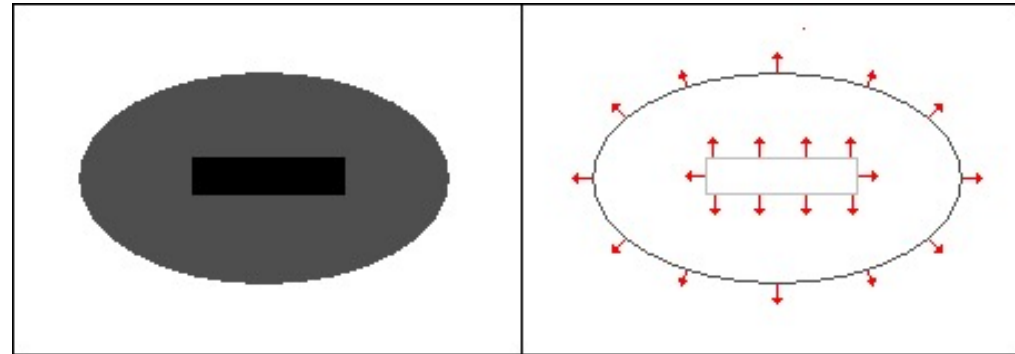


Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Edge Detection

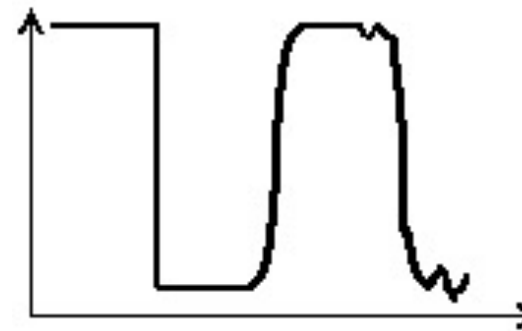
Edges have

- Magnitude
- Direction (Orientation)



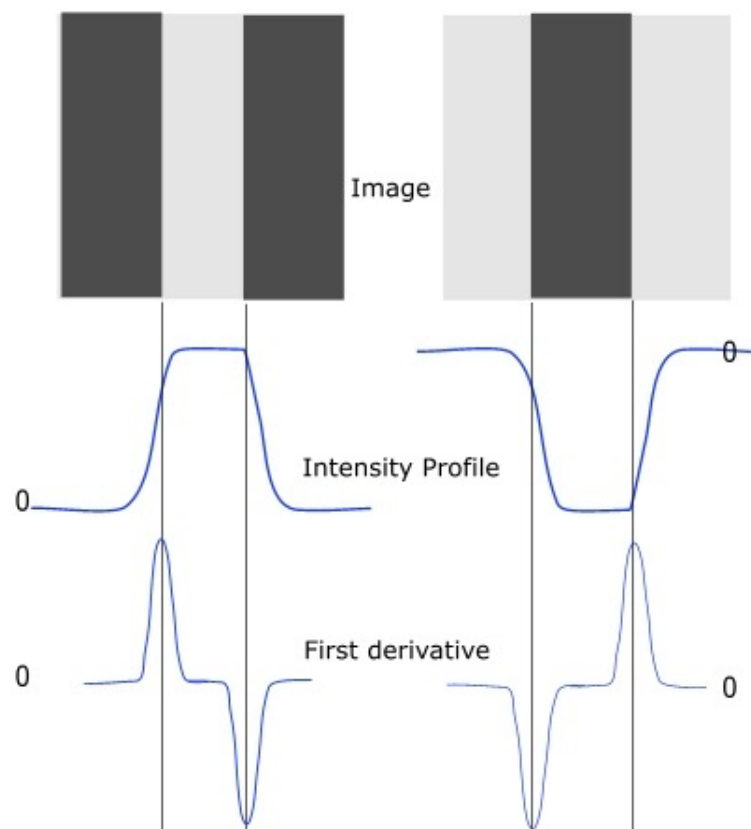
Edge Profiles

- Step
- Real
- Noisy



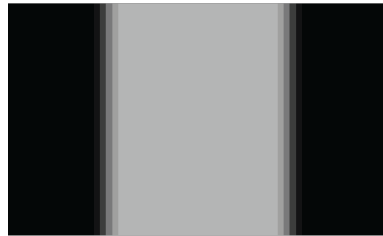
Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Edge Detection

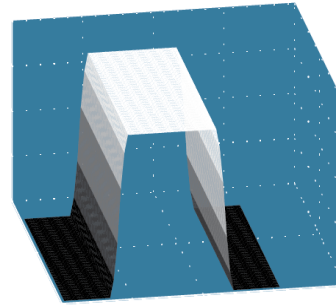


Source: https://mipav.cit.nih.gov/pubwiki/index.php/Edge_Detection:_Zero_X_Laplacian

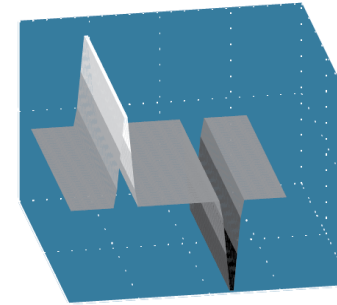
Edge Detection



Image



Intensity profile



First derivative

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Edge Detection

- Define a local edge in an image to be a **transition** between two regions of **significantly different intensities**
- The **gradient function** of the image, which measures the **rate of change**, will have large values in these transitional boundary areas
 - Enhance the image $f(x, y)$ by **estimating** its **gradient function** $g(x, y)$
 - An edge is present if the **gradient magnitude** is greater than some defined **threshold**

Edge Detection

- Gradient functions are easy to understand in the discrete domain of digital images

Derivatives become simple first differences ($h = 1$)

$$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

- Thus, the first difference of a 1D function $f(x)$ is simply

$$f(x+1) - f(x)$$

Edge Detection

Consider the following

1-D discrete (sampled & quantized) signal $f(x)$

its first derivative (i.e. 1st difference) $\frac{df(x)}{dx}$

$f(x)$	1	2	2	1	0	1	1	0	1	9	8	9	9	9	8
--------	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

$\frac{df(x)}{dx}$	1	0	-1	-1	1	0	-1	1	8	-1	1	0	0	-1	
--------------------	---	---	----	----	---	---	----	---	---	----	---	---	---	----	--

Edge Detection

In a 2D image $f(x, y)$ the gradient $g(x, y)$ is a **vector: it has magnitude and direction**

$\frac{\partial f(x, y)}{\partial x}$ and $\frac{\partial f(x, y)}{\partial y}$ represent the rates of change of a 2D function $f(x, y)$

in the x and y directions respectively:

$$\frac{\partial f(x, y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h, y) - f(x, y)}{h}$$

$$\frac{\partial f(x, y)}{\partial y} = \lim_{h \rightarrow 0} \frac{f(x, y + h) - f(x, y)}{h}$$

Edge Detection

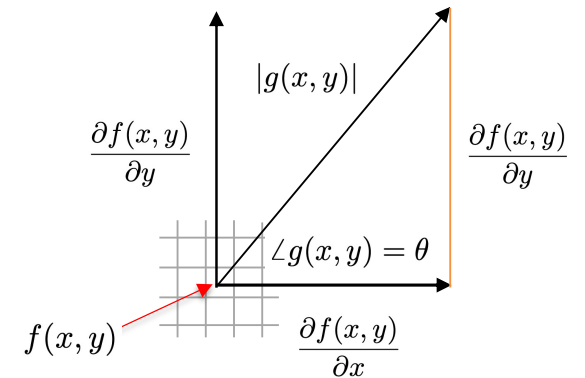
- The **direction** θ at which the rate of change has the **greatest magnitude** is given by

$$\angle g(x, y) = \theta = \arctan \left(\frac{\frac{\partial f(x, y)}{\partial y}}{\frac{\partial f(x, y)}{\partial x}} \right)$$

- The **magnitude** is given by

$$|g(x, y)| = \sqrt{\left(\frac{\partial f(x, y)}{\partial x} \right)^2 + \left(\frac{\partial f(x, y)}{\partial y} \right)^2}$$

- The **gradient of $f(x, y)$** is a **vector** at (x, y) with this magnitude and direction



Edge Detection

- Gradient functions are easy to understand in the discrete domain of digital images

Derivatives become simple first differences [$h = 1$]

$$\frac{\partial f(x, y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h, y) - f(x, y)}{h}$$

$$\frac{\partial f(x, y)}{\partial y} = \lim_{h \rightarrow 0} \frac{f(x, y + h) - f(x, y)}{h}$$

- So, the first difference of a 2D function in the x direction is simply:

$$f(x + 1, y) - f(x, y)$$

- Similarly, the first difference of a 2D function in the y direction is simply:

$$f(x, y + 1) - f(x, y)$$

Edge Detection

The essential differences between all gradient edge detectors are

- the **directions** which the operators use to estimate the **partial derivatives**
- the manner in which they **approximate the one-dimensional derivatives** of the image function in these directions
- the manner in which they **combine** these approximations to form the gradient magnitude

(a) Roberts

1	0
0	-1

0	1
-1	0

(a)

(b) Sobel

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

(b)

(c) Prewitt

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

(c)

Edge Detection

Edge strength: $G = \sqrt{G_x^2 + G_y^2}$
 Angle: $\theta = \arctan(G_y / G_x)$

Sobel

-1	0	+1
-2	0	+2
-1	0	+1

G_x

+1	+2	+1
0	0	0
-1	-2	-1

G_y

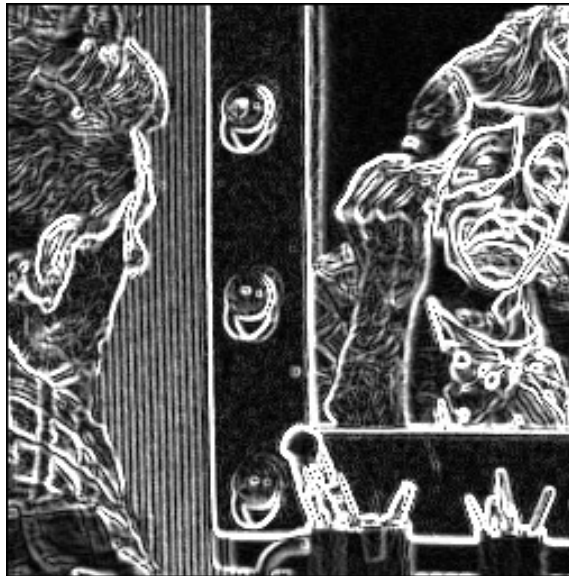
Prewitt

-1	0	+1
-1	0	+1
-1	0	+1

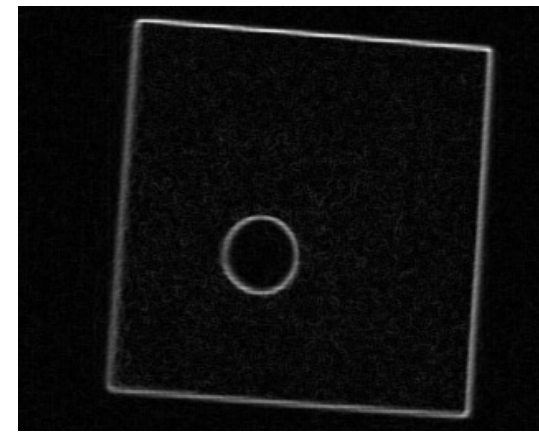
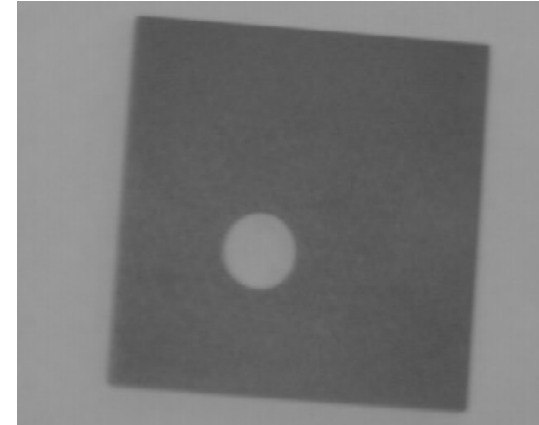
G_x

+1	+1	+1
0	0	0
-1	-1	-1

G_y



Credit: Markus Vincze, Technische Universität Wien

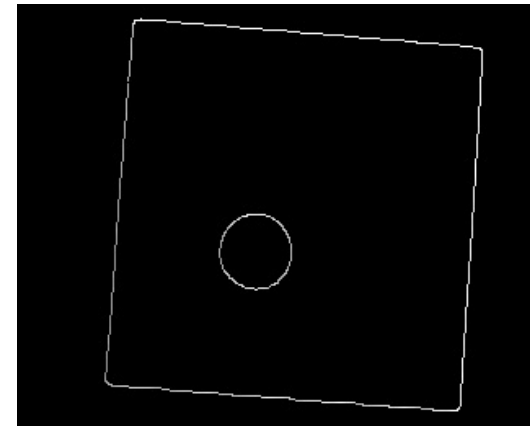
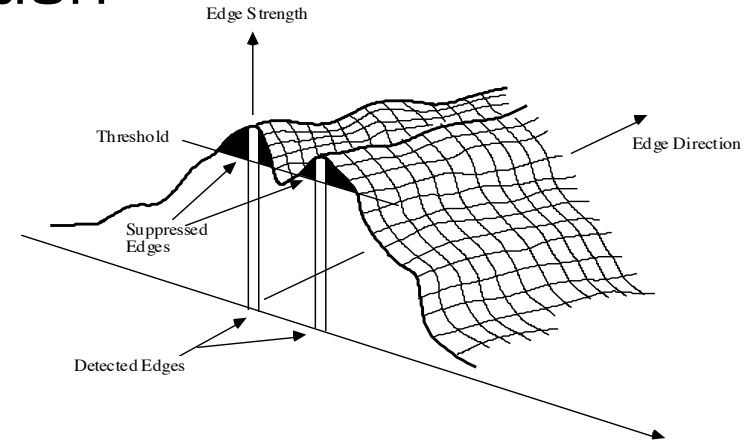


Edge Detection

Canny Edge Detector

1. Gaussian smoothing
2. Gradient estimation
3. Ridge following with non-maxima suppression and hysteresis ($t_2 > t_1$)

- Optimised, standard method
- Good compromise
- Thin, one pixel edge (ridge)
- Smoothing eliminates detail



$$\sigma = 1, t_2 = 255, t_1 = 1$$

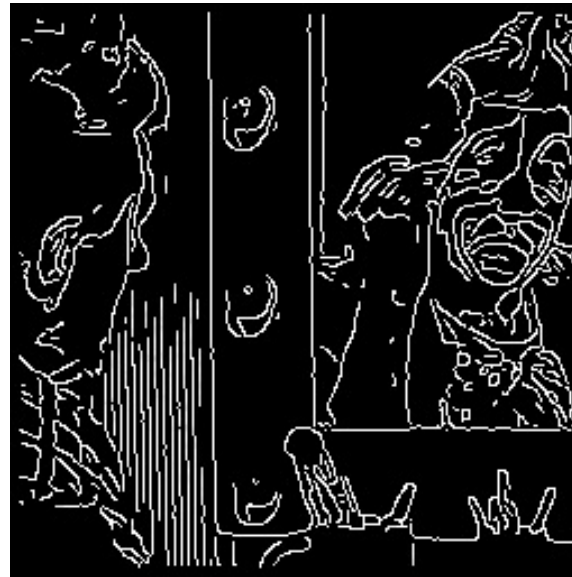
Edge Detection

Canny Edge Detector

- Y-Effect: 3 edges meeting in a point are not connected
- Adaptive: detail and edge elements, but image dependent



$$\sigma = 1, t_2 = 255, t_1 = 1$$



$$\sigma = 1, t_2 = 255, t_2 = 220$$



$$\sigma = 2, t_2 = 255, t_1 = 1$$

Credit: Markus Vincze, Vienna University of Technology

Edge Detection

Why do we convolve the image with a Gaussian function?

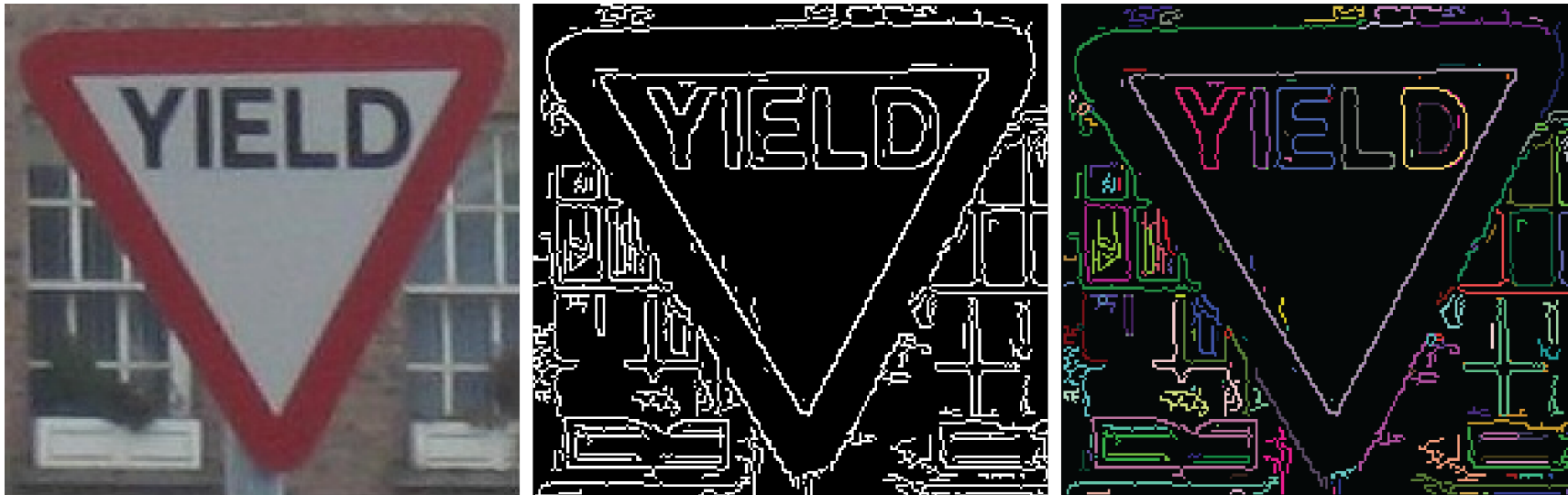
- The Gaussian blurs the image
- This wipes out all structure at scales much smaller than the space constant σ (standard deviation) of the Gaussian
- Thus, we can select the spatial scale at which we process the image

Boundary Detection

- Edge detection is just the first stage of the boundary-based segmentation process
- Also to aggregate these local edge elements
 - into structures better suited to the process of interpretation
- Normally achieved using processes such as
 - edge thinning (gradient-based edge operators produce thick edges)
 - edge linking
 - gap filling
 - curve-segment linking

Boundary Detection

Representation of Boundaries



In OpenCV, each individual contour is stored as a vector of points and all the contours are stored as a vector of contours (i.e. a vector of vector of points)

Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Reading

R. Szeliski, *Computer Vision: Algorithms and Applications*, Springer, 2010.

Section 3.3 More neighborhood operations

Section 3.3.4 Connected components

Section 4.2 Edges

D. Vernon, *Machine Vision*, 1991.

Section 5.1 Introduction: region- and boundary-based approaches

Section 5.3.1 Gradient- and difference-based operators

Summary of min-cut approaches in computer vision

Boykov, Y. and Veksler, O. 2006. "Graph Cuts in Vision and Graphics: Theories and Applications", in Handbook of Mathematical Models of Computer Vision, Paragios, N., Chen, Y., Faugeras, O. D. (eds.), Springer, pp. 79-96.

GrabCut

Rother, C., Kolmogorov, V., and Blake, A. 2004. "GrabCut – Interactive Foreground Extraction using Iterated Graph Cuts, ACM Transactions on Graphics.

Demo

The following code is taken from the `sobelEdgeDetection` example application

See:

```
sobelEdgeDetection.h  
sobelEdgeDetectionImplementation.cpp  
sobelEdgeDetectionApplication.cpp
```

To run the example:

```
Ubuntu 16.04: rosrun module5 sobelEdgeDetection
```

```

/*
 * function sobelEdgeDetection
 * Trackbar callback - threshold user input
 */

void sobelEdgeDetection(int, void*) {

    extern Mat inputImage;
    extern int thresholdValue;
    extern char* magnitude_window_name;
    extern char* direction_window_name;
    extern char* edge_window_name;

    Mat greyscaleImage;
    Mat edgeImage;
    Mat horizontal_partial_derivative;
    Mat vertical_partial_derivative;
    Mat l2norm_gradient;
    Mat orientation;

    if (inputImage.type() == CV_8UC3) { // colour image
        cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
    }
    else {
        greyscaleImage = inputImage.clone();
    }
    /*****
    /*
     * This code is provided as part of "A Practical Introduction to Computer Vision with OpenCV"
     * by Kenneth Dawson-Howe © Wiley & Sons Inc. 2014. All rights reserved.
     */

    Sobel(greyscaleImage, horizontal_partial_derivative, CV_32F, 1, 0);
    Sobel(greyscaleImage, vertical_partial_derivative, CV_32F, 0, 1);
    cartToPolar(horizontal_partial_derivative, vertical_partial_derivative, l2norm_gradient, orientation);
    Mat l2norm_gradient_gray = convert_32bit_image_for_display( l2norm_gradient );
    Mat l2norm_gradient_mask, display_orientation;
    l2norm_gradient.convertTo(l2norm_gradient_mask, CV_8U);
    threshold(l2norm_gradient_mask, edgeImage, thresholdValue, 255, THRESH_BINARY); // DV thresholdValue edgeImage
    orientation.copyTo(display_orientation, edgeImage);
    Mat orientation_gray = convert_32bit_image_for_display(display_orientation, 0.0, 255.0/(2.0*PI) );

    imshow(magnitude_window_name, l2norm_gradient_gray); // DV
    imshow(direction_window_name, orientation_gray); // DV
    imshow(edge_window_name, edgeImage); // DV
}

```


Demo

The following code is taken from the `cannyEdgeDetection` example application

See:

```
cannyEdgeDetection.h  
cannyEdgeDetectionImplementation.cpp  
cannyEdgeDetectionApplication.cpp
```

To run the example:

```
Ubuntu 16.04: rosrun module5 cannyEdgeDetection
```

```

/*
 * CannyThreshold
 * Trackbar callback - Canny thresholds input with a ratio 1:3
 */

void CannyThreshold(int, void*)
{
    extern Mat src;
    extern Mat src_gray;
    extern Mat src_blur;
    extern Mat detected_edges;
    extern int cannyThreshold;
    extern char* canny_window_name;
    extern int gaussian_std_dev;

    int ratio = 3;
    int kernel_size = 3;
    int filter_size;

    filter_size = gaussian_std_dev * 4 + 1; // multiplier must be even to ensure an odd filter size as required by OpenCV
                                           // this places an upper limit on gaussian_std_dev of 7 to ensure the filter size < 31
                                           // which is the maximum size for the Laplacian operator

    cvtColor(src, src_gray, CV_BGR2GRAY);

    GaussianBlur(src_gray, src_blur, Size(filter_size,filter_size), gaussian_std_dev);

    Canny( src_blur, detected_edges, cannyThreshold, cannyThreshold*ratio, kernel_size );

    imshow( canny_window_name, detected_edges );
}

```

Demo

The following code is taken from the `contourExtraction` example application
See:

```
contourExtraction.h  
contourExtractionImplementation.cpp  
contourExtractionApplication.cpp
```

To run the example:

```
Ubuntu 16.04: rosrun module5 contourExtraction
```

```

/*
 * ContourExtraction
 * Tracker callback - Canny hysteresis thresholds input with a ratio 1:3 and Gaussian standard deviation
 */

void ContourExtraction(int, void*) {
    extern Mat src;
    extern Mat src_gray;
    extern Mat src_blur;
    extern Mat detected_edges;
    extern int cannyThreshold;
    extern char* canny_window_name;
    extern char* contour_window_name;
    extern int gaussian_std_dev;

    bool debug = true;
    int ratio = 3;
    int kernel_size = 3;
    int filter_size;
    vector<vector<Point>> contours;
    vector<Vec4i> hierarchy;
    Mat thresholdedImage;

    filter_size = gaussian_std_dev * 4 + 1; // multiplier must be even to ensure an odd filter size as required by OpenCV
                                         // this places an upper limit on gaussian_std_dev of 7 to ensure the filter size < 31
                                         // which is the maximum size for the Laplacian operator

    cvtColor(src, src_gray, CV_BGR2GRAY);

    GaussianBlur(src_gray, src_blur, Size(filter_size,filter_size), gaussian_std_dev);

    Canny( src_blur, detected_edges, cannyThreshold, cannyThreshold*ratio, kernel_size );
}

```

```

Mat canny_edge_image_copy = detected_edges.clone(); // clone the edge image because findContours overwrites it

/* see http://docs.opencv.org/2.4/modules/imgproc/doc/structural\_analysis\_and\_shape\_descriptors.html#findcontours */
/* and http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find\_contours/find\_contours.html */
findContours(canny_edge_image_copy, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_NONE);

Mat contours_image = Mat::zeros(src.size(), CV_8UC3); // draw the contours on a black background

for (int contour_number=0; (contour_number<(int)contours.size()); contour_number++) {
    Scalar colour( rand()&0xFF, rand()&0xFF, rand()&0xFF ); // use a random colour for each contour
    drawContours( contours_image, contours, contour_number, colour, 1, 8, hierarchy );
}

if (debug) printf("Number of contours %d: \n", contours.size());

imshow( canny_window_name, detected_edges );
imshow( contour_window_name, contours_image );
}

```