Introduction to Cognitive Robotics

Module 5: Robot Vision

Lecture 4: Segmentation; region-based approaches; feature-based thresholding; graph cuts

David Vernon
Carnegie Mellon University Africa

www.vernon.eu

Segmentation

- Partitioning the image into its constituent parts

- Constituent parts depend on the task
  - Detect object, object class, foreground/background

Segmentation

- Partitioning the image into its constituent parts

- Constituent parts depend on the task
  - Detect object, object class, foreground/background

Segmentation

A grouping process:

- the components of a group are similar with respect to some feature or set of features
- This grouping should identify regions in the image which correspond to unique and distinct objects

Segmentation

Two complementary approaches:

1. Region Growing
   - Grouping elemental areas (in simple cases, individual image pixels)
   - That share a common feature
   - Into connected two-dimensional areas called regions
   - e.g. pixel grey-level, hue, or some textural pattern

2. Boundary Detection
   - Detecting or enhancing the boundary pixels of objects within the image
   - Edge detection … discontinuities is some feature between regions
   - Typical feature: image intensity

Segmentation

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

Region Growing

Binary Thresholding

- Intensity or colour thresholding is a simple region based segmentation technique
- Works well where
  - an object exhibits a uniform grey-level or colour
  - and rests against a background of a different grey-level or colour

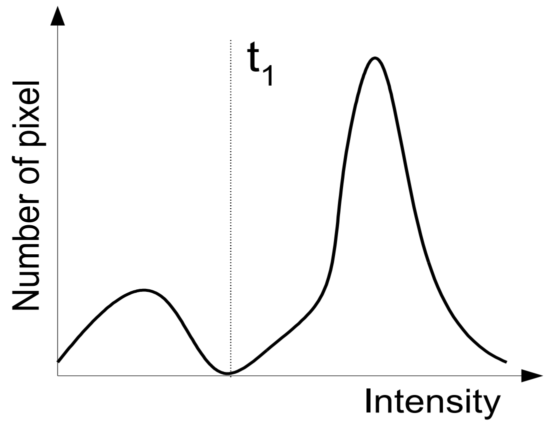
Binary Thresholding

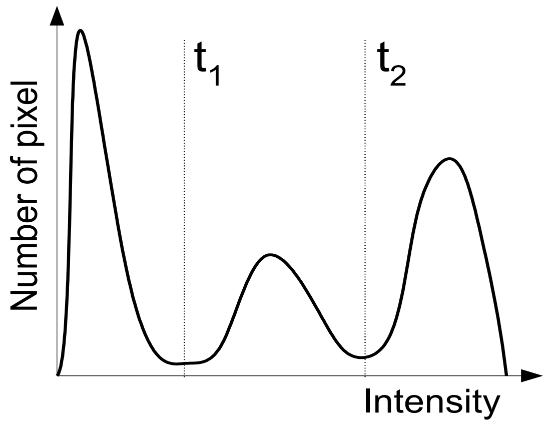
If $g(x, y)$ is a thresholded version of $f(x, y)$ for some global threshold T

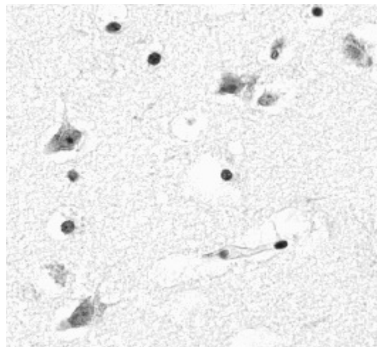
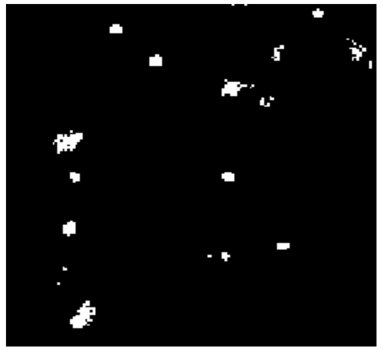
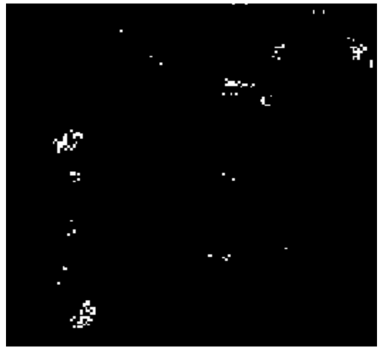
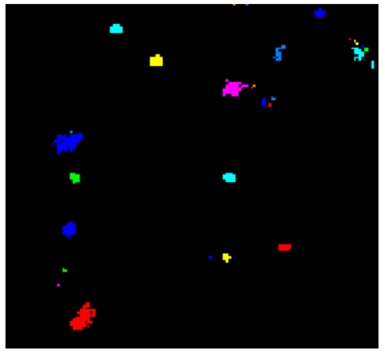
$$\begin{aligned} g(x, y) &= \overset{255}{1} \quad \text{if } f(x, y) \geq T \\ &= 0 \quad \text{otherwise} \end{aligned}$$

Binary Thresholding

Histogram



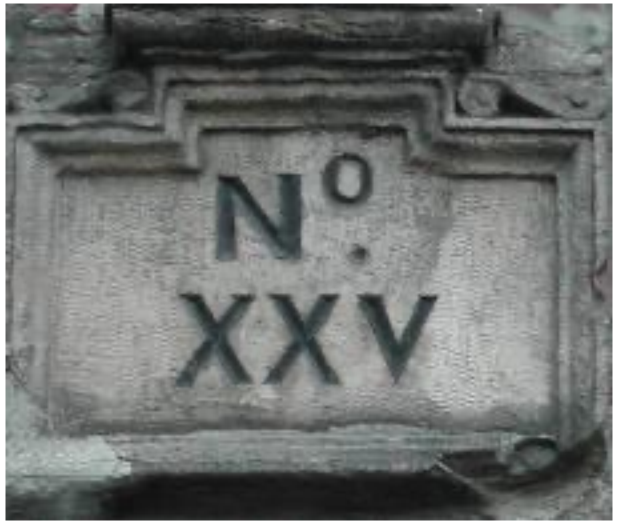


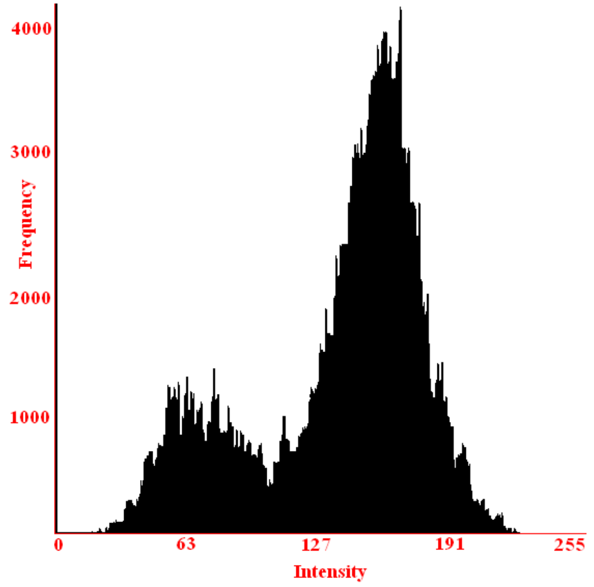





Original image t_1=150 t_1=130, t_2=150 Regions (blobs)

Credit: Markus Vincze, Technische Universität Wien

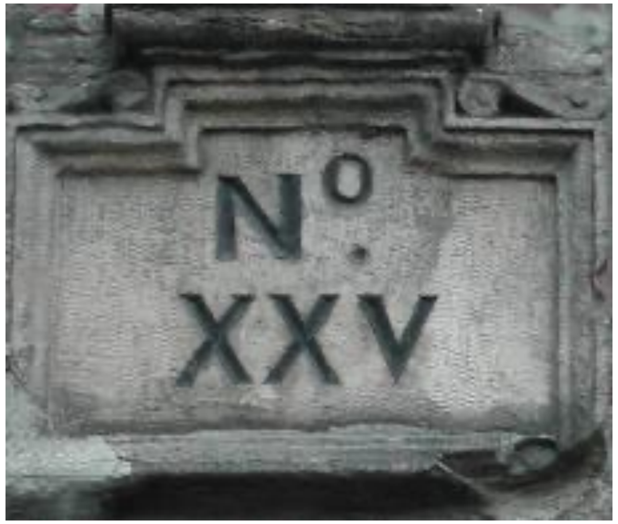
Binary Thresholding

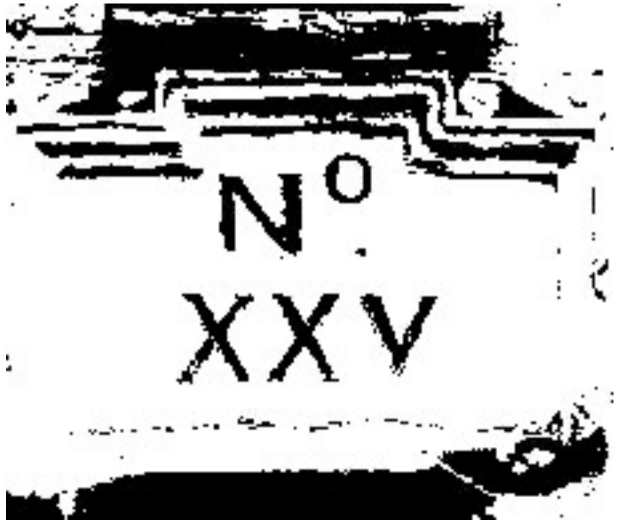




Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

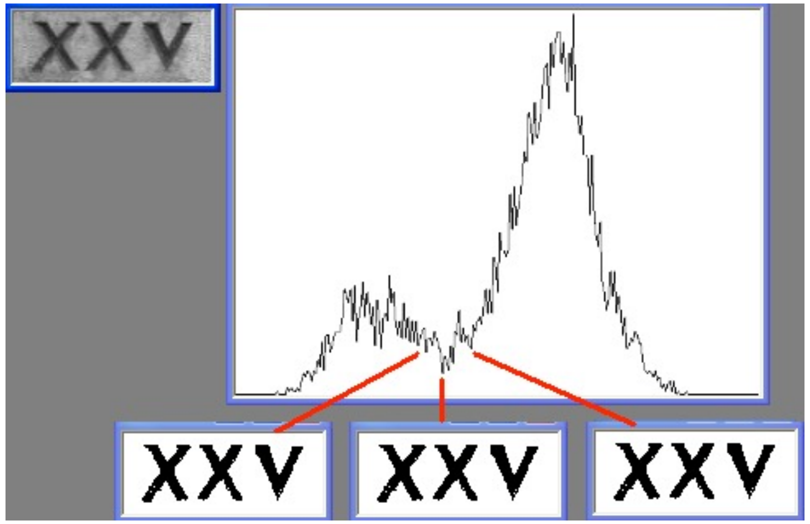
Binary Thresholding





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

Binary Thresholding

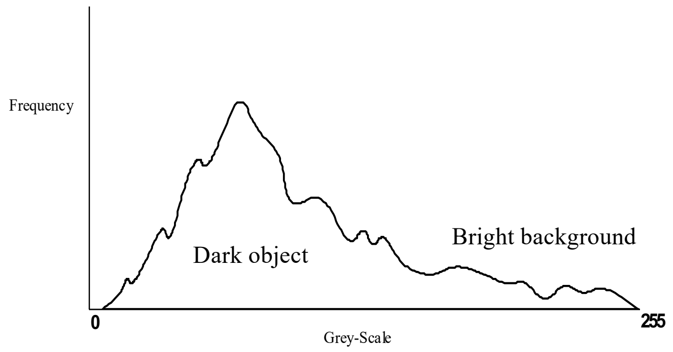


Credit: Kenneth Dawson-Howe, A Practical Introduction to Computer Vision with OpenCV, © Wiley & Sons Inc. 2014

Binary Thresholding

Threshold Selection

- Most techniques are based on histogram analysis
  - select thresholds which lie in the region between the modes
- Assumption of bi-modal histogram may not be valid
  - Histograms are noisy
  - Histograms may be uni-modal
  - Histogram smoothing is often required

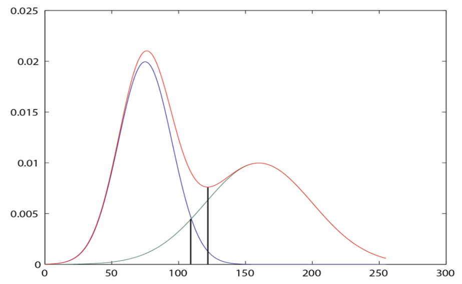


Binary Thresholding

Threshold Selection – Otsu Algorithm

A technique that finds the threshold separating the two classes (background and foreground) so that

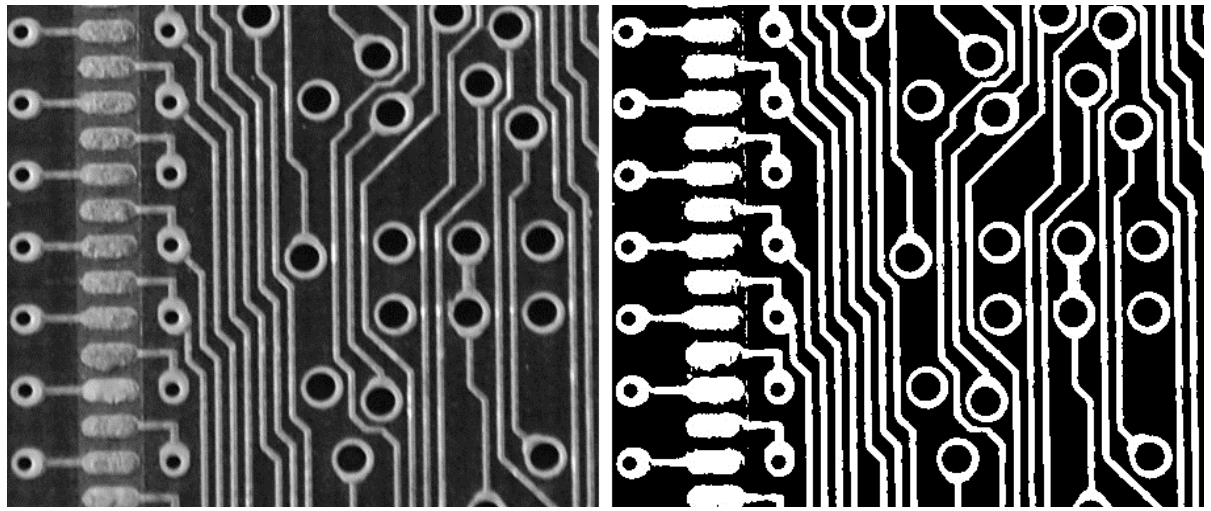
- their combined spread (intra-class variance) is minimal,
- or, equivalently, so that their inter-class variance is maximal



N. Otsu, "A threshold selection method from gray-level histograms”, IEEE Trans. Sys., Man., Cyber. 9 (1): 62–66, 1979.

Binary Thresholding

Threshold Selection – Otsu Algorithm



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

Colour Segmentation

- To segment images based on colour
  - Transform to HSI space
  - Discard I
  - Set minimum and maximum limits (thresholds) of acceptable H and S
  - Need to be careful where H wraps from 0 to 360 (or 180 in if using HLS in OpenCV)
- It may be necessary to smooth the image first
- It will be necessary to perform connected component analysis after the segmentation to label the segmented regions with distinct (mutually-exclusive) labels

Connected Component Analysis

Adjacency conventions

- This problem is one of defining exactly which are the neighbours of a given pixel
- Consider the 3*3 neighbourhood in an image where the pixels are labelled 0 through 8

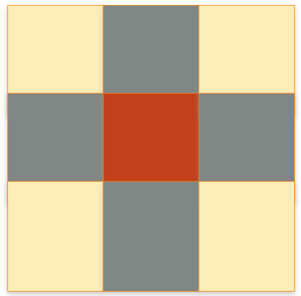
3	2	1
4	8	0
5	6	7

Which pixels does pixel 8 touch?

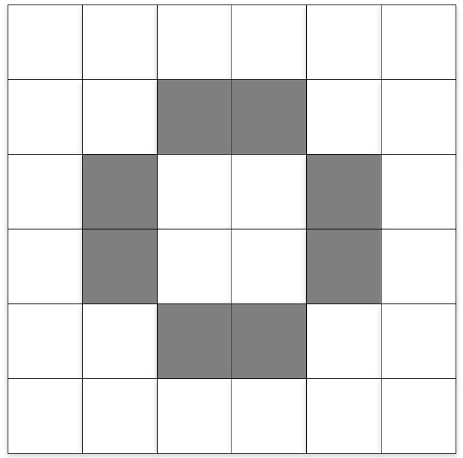
Connected Component Analysis

Adjacency conventions

- A pixel p at coordinates (i, j) has four horizontal and vertical neighbors at coordinates
  $(i\text{-}1, j)$ $(i\text{+}1, \text{j})$ $(i, j\text{-}1)$ $(i, j\text{+}1)$
- This set is called 4-neighborhood $N_4(p)$
- The pixel also has four diagonal neighbors:
  $(i\text{-}1, j\text{-}1)$ $(i\text{+}1, j\text{-}1)$ $(i\text{+}1, j\ \text{-}1)$ $(i\text{+}1, j\text{+}1)$
- The 8 points together form a 8-neighborhood $N_8(p)$



Connected Component Analysis

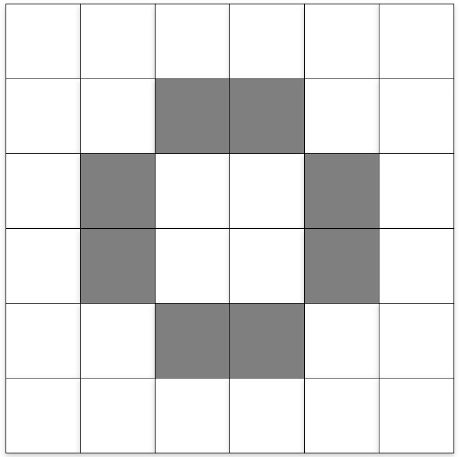


If figure and ground are both 8-connected it means the hole in the ‘ring’ is connected to the region surrounding the ‘ring’

Connected Component Analysis

It is normal practice to use both conventions

- one for an object
- one for the background on which it rests



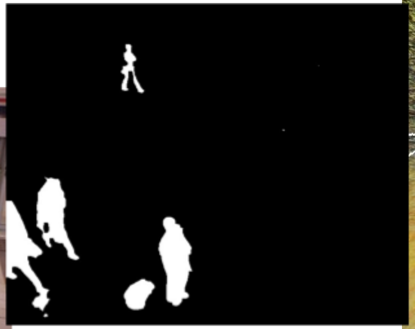
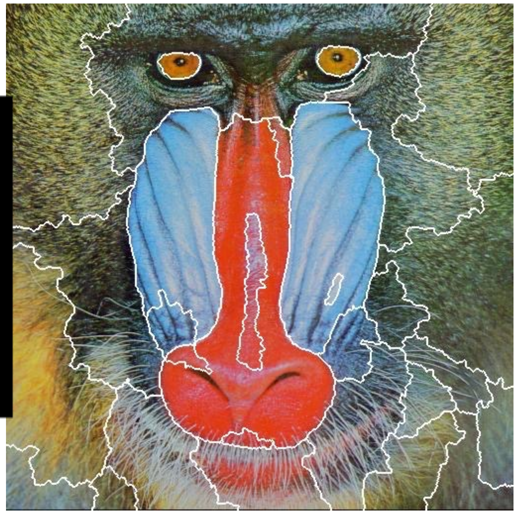
This can be extended quite generally

- adjacency conventions are applied alternatively to image regions which are recursively nested (or embedded) within other regions as one goes from level to level in the nesting

Connected Component Analysis

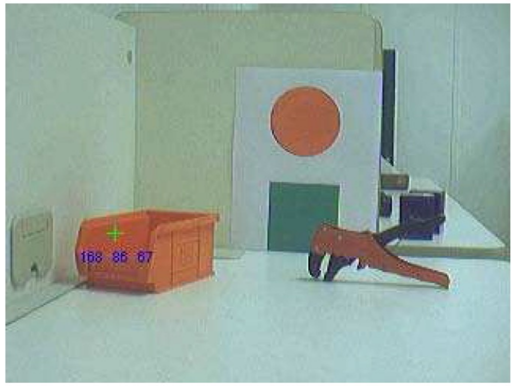
Connected components

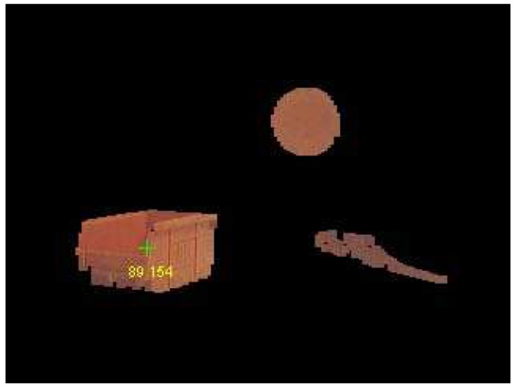
- groups of connected pixels with common properties
- the properties could be a similar color, texture, or motion pattern, ...

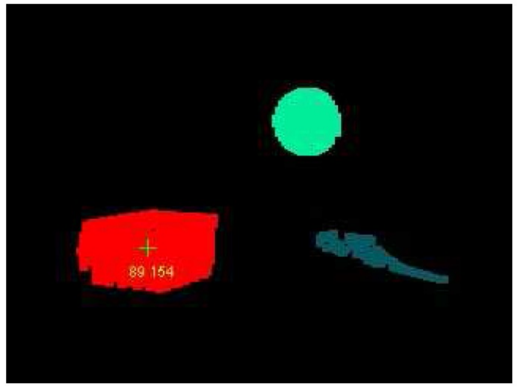



Credit: Francesca Odone, University of Genova

Connected Component Analysis

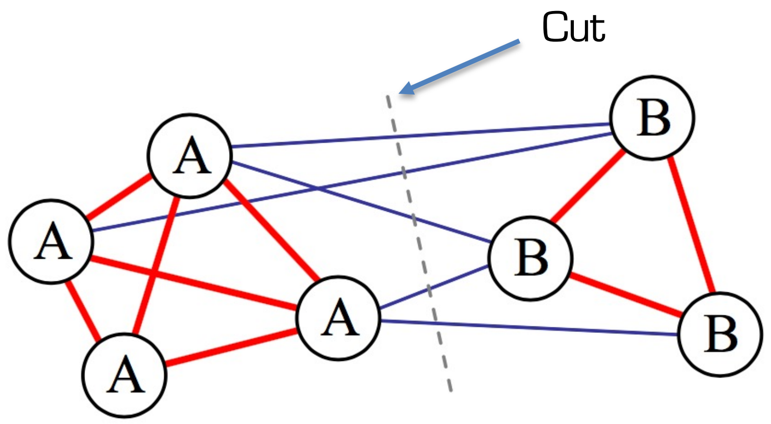






Graph Cuts

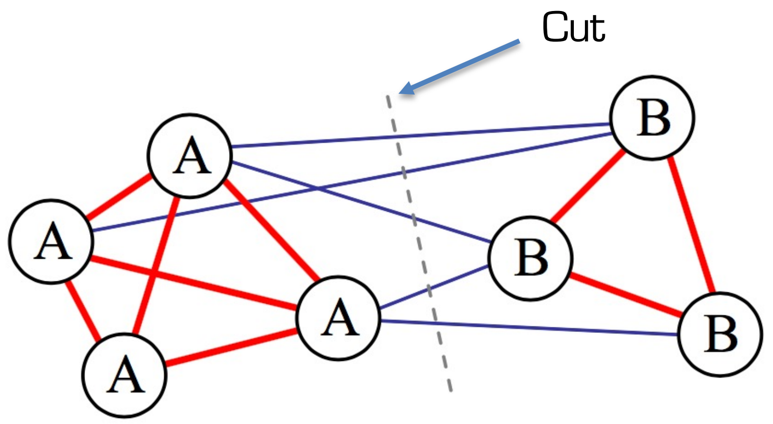
- Image as graph
  - Initially with one connected component
    (there is a path from any pixel to any other pixel)
- Segmentation as a process of finding a cut to separate the graph into two or more components



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

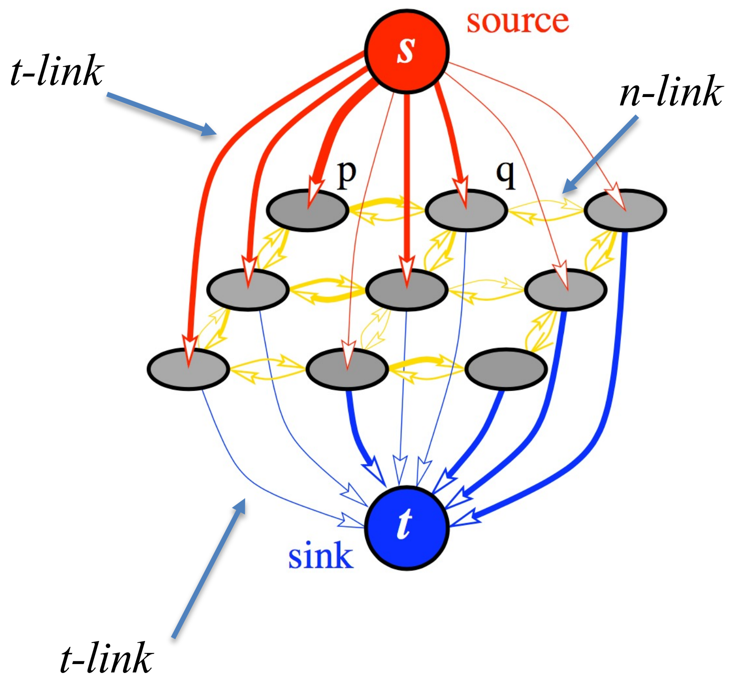
Graph Cuts

- Identify affinities (similarity) between nearby pixels
- Separate groups that are connected by weak affinities



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

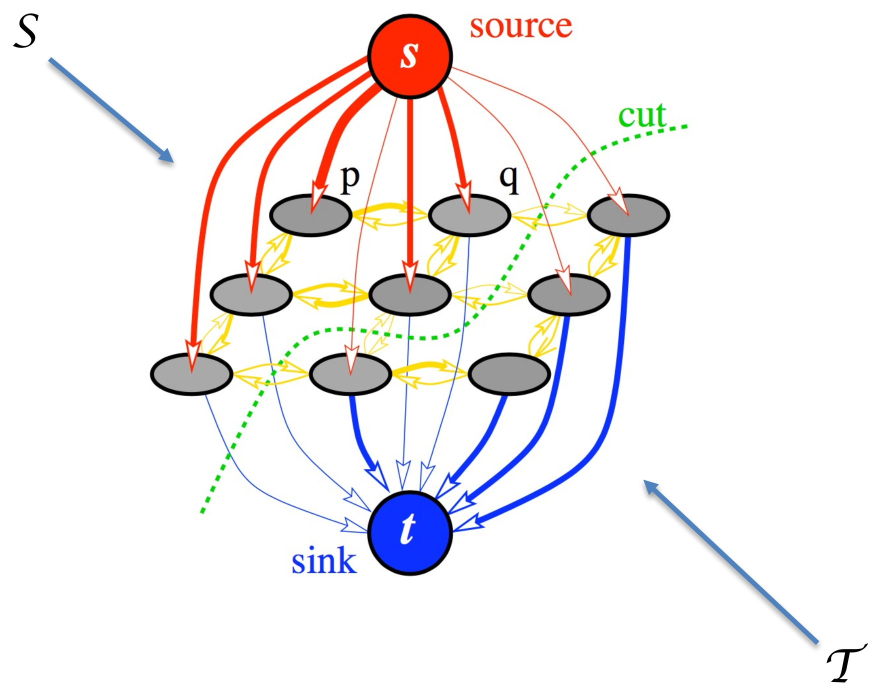
Graph Cuts



Edge weights / costs are reflected by thickness

Source: Boykov and Veksler 2006

Graph Cuts



Source: Boykov and Veksler 2006

Graph Cuts

The Min-Cut and Max-Flow Problem

- The cost of a cut $C = \{\mathcal{S}, \mathcal{T}\}$ is the sum of the costs/weights of “boundary” edges (p, q) such that $p \in \mathcal{S}$ and $q \in \mathcal{T}$

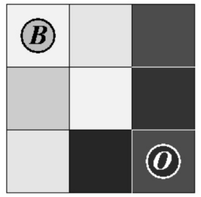
- If (p, q) is a boundary edge, we say cut C severs edge (p, q)

- The minimum cut problem is to find a cut that has the minimum cost among all cuts

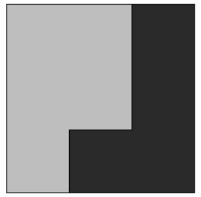
Source: Boykov and Veksler 2006

Graph Cuts

Using seeds



(a) Image with seeds.



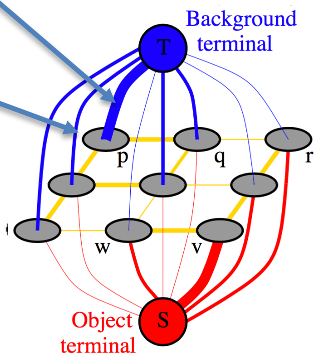
(d) Segmentation results.

Infinity cost t-link

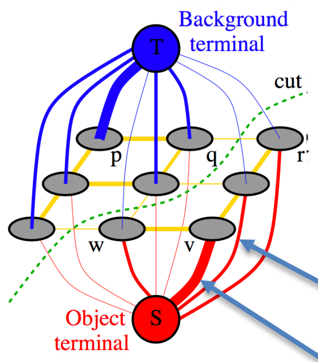
⇓

⇑

Background
Seed



⇒



Object
Seed

Infinity cost t-link

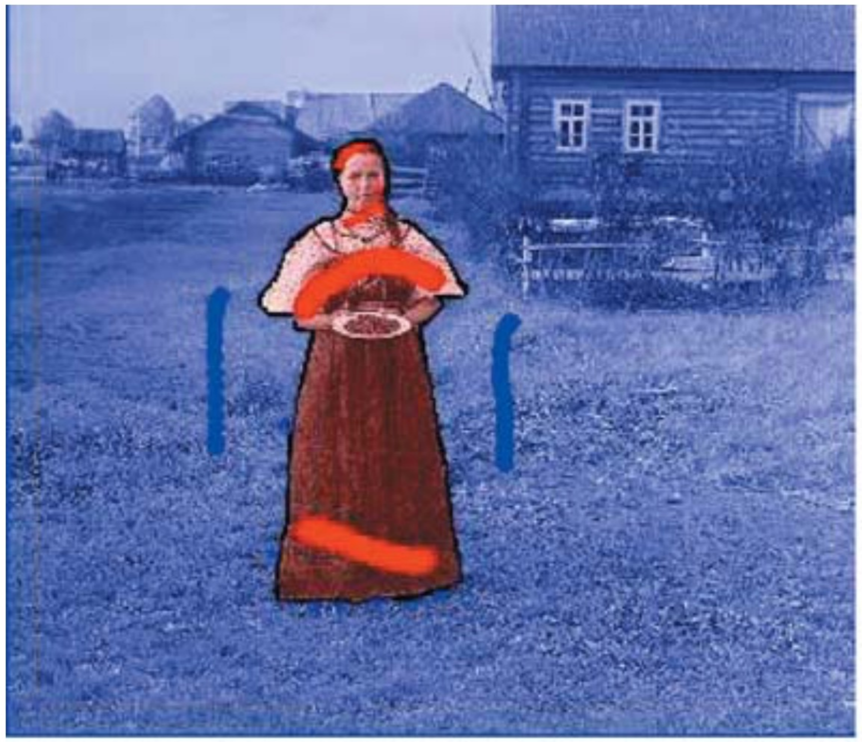
(b) Graph.

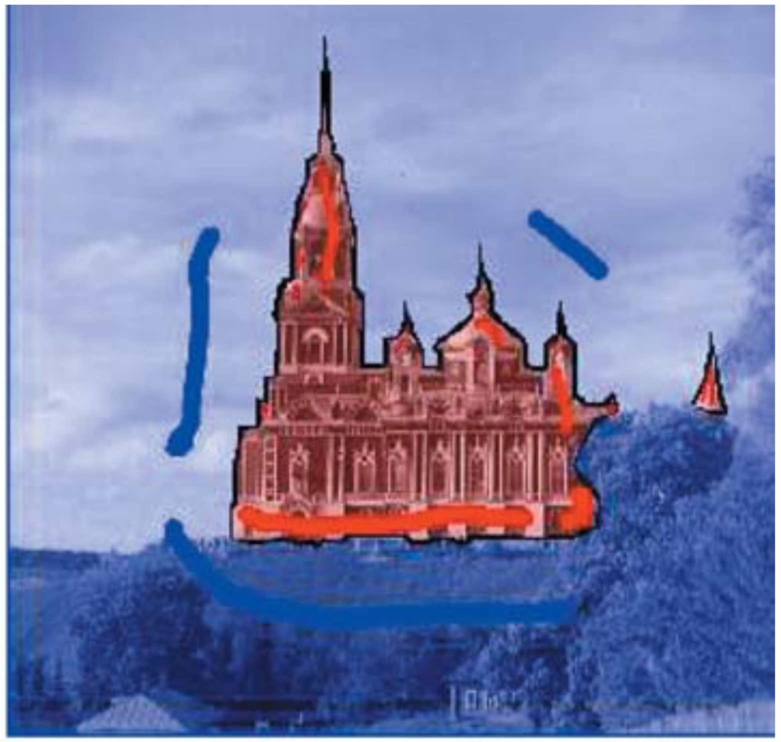
(c) Cut.

Edge weights / costs are reflected by thickness

Source: Boykov and Funka-Lea 2006

Graph Cuts


(a) A woman from a village


(b) A church in Mozhaisk (near Moscow)

Source: Boykov and Funka-Lea 2006

Graph Cuts

GrabCut

- Extension of the basic Boykov & Jolly (2001) technique by Rother, Komolgorov, and Blake (2004)
- Iteratively re-estimates the region statistics
- Region statistics are modelled as a mixture of Gaussians in colour space

Graph Cuts

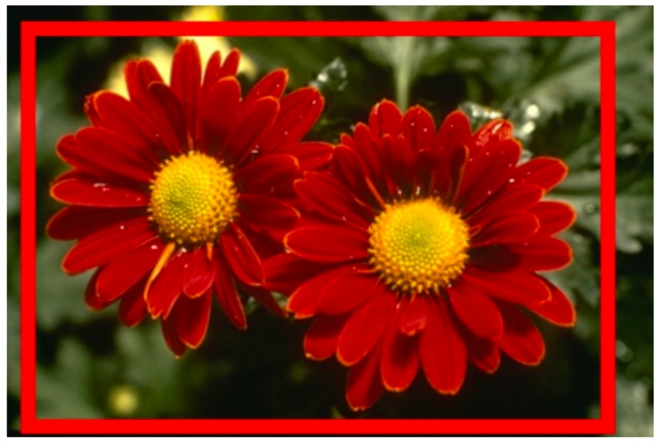
GrabCut

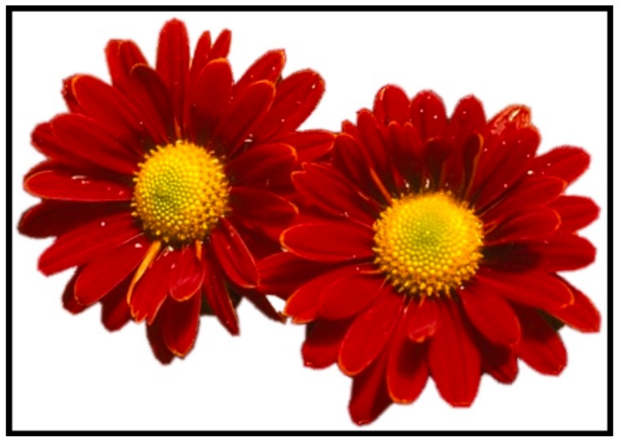
This allows the approach to operate with minimal user input

- Single bounding box
- The background colour model
    - initialized from a strip of pixels around the box outline
- The foreground colour model
    - initialized from the interior pixels
    - quickly converges to a better estimate of the object

Graph Cuts

GrabCut

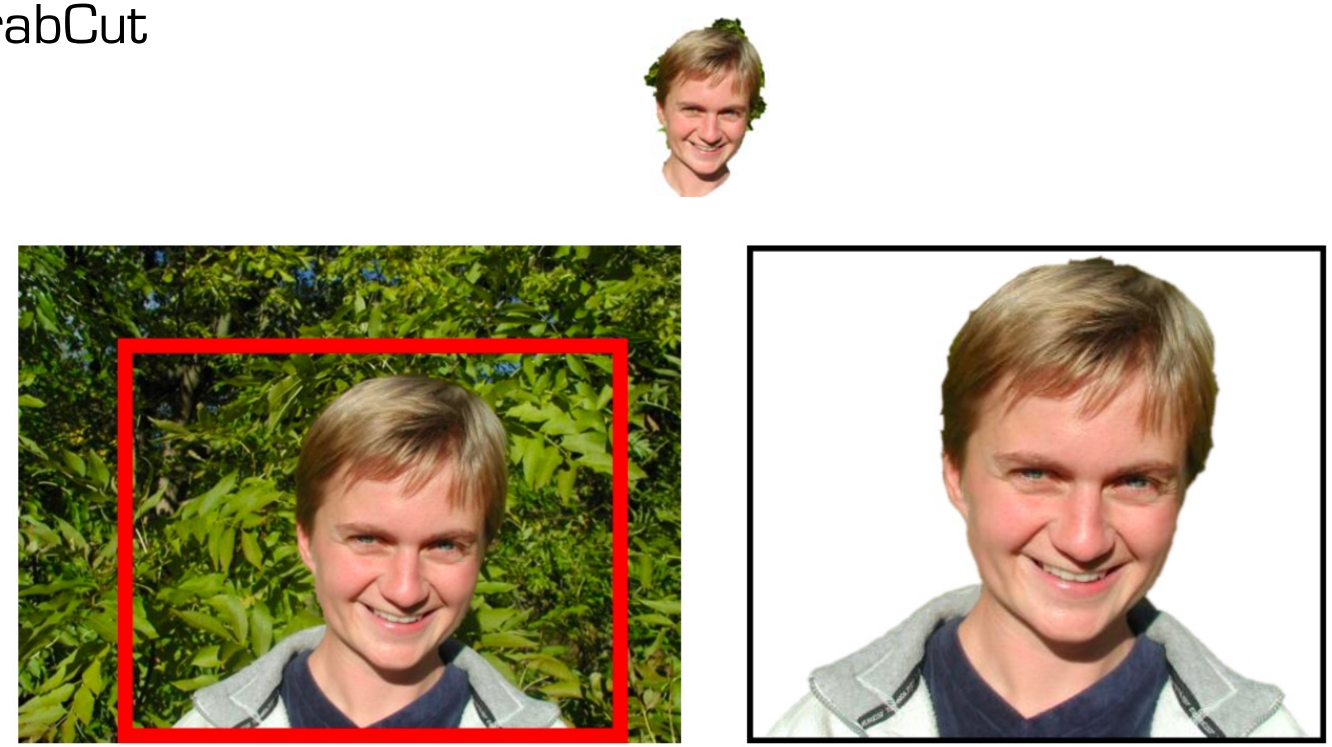




Source: Rother, Komolgorov, Blake 2004

Graph Cuts

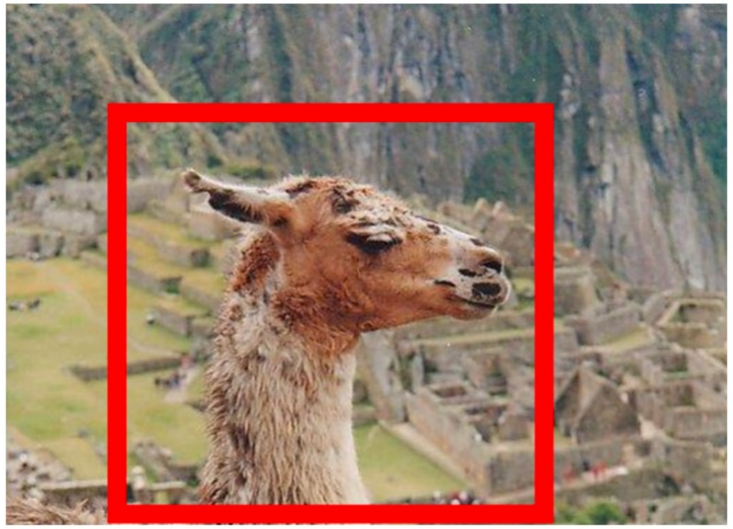
GrabCut

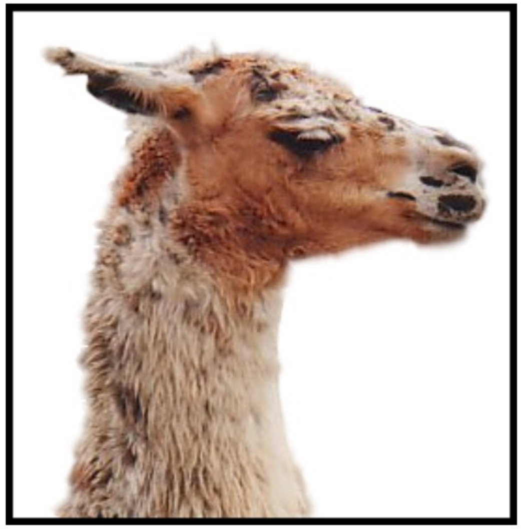


Source: Rother, Komolgorov, Blake 2004

Graph Cuts

GrabCut

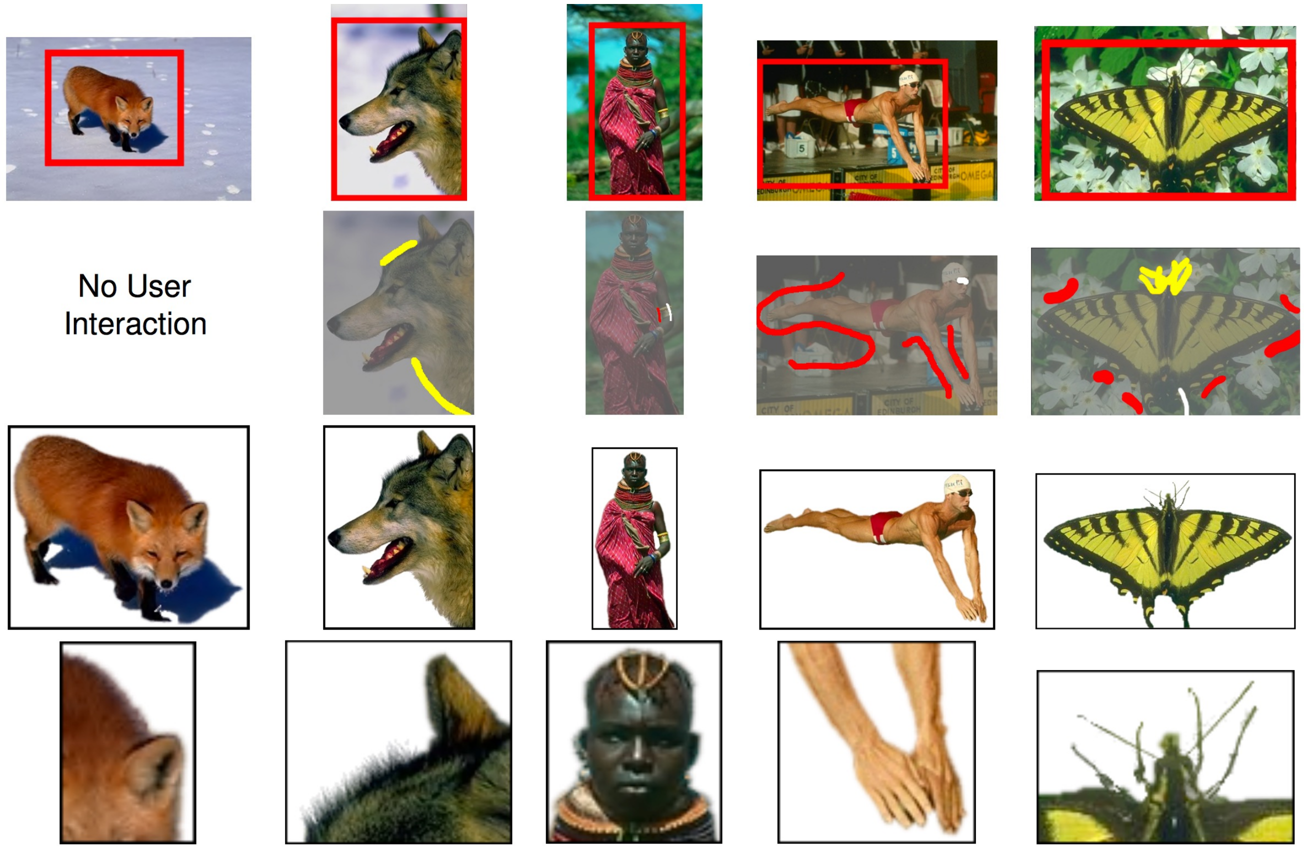




Source: Rother, Komolgorov, Blake 2004

Graph Cuts

GrabCut



Source: Rother, Komolgorov, Blake 2004

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 binaryThresholding example application

See:

```
binaryThresholding.h
binaryThresholdingImplementation.cpp
binaryThresholdingApplication.cpp
```

To run the example:

Ubuntu 16.04: `rosrun module5 binaryThresholding`
Windows 10: double-click `C:\CORO\lectures\bin\binaryThresholding`

```
void binaryThresholding(int, void*) {

   extern Mat inputImage;
   extern int thresholdValue;
   extern char* thresholded_window_name;
   Mat greyscaleImage;
   Mat thresholdedImage;
   int row, col;

   if (thresholdValue < 1)  // the trackbar has a lower value of 0 which is invalid
      thresholdValue = 1;

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }

   thresholdedImage.create(greyscaleImage.size(), CV_8UC1);

   for (row=0; row < greyscaleImage.rows; row++) {
      for (col=0; col < greyscaleImage.cols; col++) {
         if(greyscaleImage.at<uchar>(row,col) < thresholdValue) {
            thresholdedImage.at<uchar>(row,col) = (uchar) 0;
         }
         else {
            thresholdedImage.at<uchar>(row,col) = (uchar) 255;
         }
      }
   }

   /* alternatively, use OpenCV */

   // threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY);
   // threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY  | THRESH_OTSU); // automatic threshold selection

   imshow(thresholded_window_name, thresholdedImage);
}
```

Demo

The following code is taken from the binaryThresholdingOtsu example application

See:

```
binaryThresholdingOtsu.h
binaryThresholdingOtsuImplementation.cpp
binaryThresholdingOtsuApplication.cpp
```

To run the example:

Ubuntu 16.04: `rosrun module5 binaryThresholdingOtsu`
Windows 10: double-click `C:\CORO\lectures\bin\binaryThresholdingOtsu`

```
void binaryThresholdingOtsu(char *filename) {

   Mat inputImage;
   Mat greyscaleImage;
   Mat thresholdedImage;

   int thresholdValue            = 128; // default threshold

   char* input_window_name       = "Input Image";
   char* thresholded_window_name = "Thresholded Image";

   inputImage = imread(filename, CV_LOAD_IMAGE_UNCHANGED);
   if (inputImage.empty()) {
      cout << "can not open " << filename << endl;
      prompt_and_exit(-1);
   }

   printf("Press any key to continue ...\n");

   // Create a window for input and display it
   namedWindow(input_window_name, CV_WINDOW_AUTOSIZE );
   imshow(input_window_name, inputImage);

   // Create a window for thresholded image
   namedWindow(thresholded_window_name, CV_WINDOW_AUTOSIZE );

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }
```

```
    //thresholdedImage.create(greyscaleImage.size(), CV_8UC1);

    threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY  | THRESH_OTSU); // automatic threshold selection

    imshow(thresholded_window_name, thresholdedImage);

    do {
       waitKey(30);                                  // Must call this to allow openCV to display the images
    } while (!_kbhit());                             // We call it repeatedly to allow the user to move the windows
                                                    // (if we don't the window process hangs when you try to click and drag

    getchar(); // flush the buffer from the keyboard hit

    destroyWindow(input_window_name);
    destroyWindow(thresholded_window_name);
}
```

Demo

The following code is taken from the colourSegmentation example application

See:

```
colourSegmentation.h
colourSegmentationImplementation.cpp
colourSegmentationApplication.cpp
```

To run the example:

Ubuntu 16.04: `rosrun module5 colourSegmentation`
Windows 10: double-click `C:\CORO\lectures\bin\colourSegmentation`

```
/*
  Example use of openCV to perform colour segmentation

  The user must interactively select the colour sample that will form the basis of the segmentation.
  The user can also adjust the hue and saturation tolerances on that sample.
  -------------------------------------------------------------------------------------------------
  Application file

  David Vernon
  1 June 2017
*/

#include "colourSegmentation.h"

// Global variables to allow access by the display window callback functions

Mat inputBGRImage;
Mat inputHLSImage;
int hueRange            = 10; // default range
int saturationRange     = 10; // default range
Point2f sample_point;
int number_of_sample_points;

char* input_window_name       = "Input Image";
char* segmented_window_name   = "Segmented Image";
```

```
int main() {

   int end_of_file;
   bool debug = false;
   char filename[MAX_FILENAME_LENGTH];
   int max_hue_range = 180;
   int max_saturation_range = 128;
   Mat outputImage;

   FILE *fp_in;

   if ((fp_in = fopen("../data/colourSegmentationInput.txt","r")) == 0) {
     printf("Error can't open input colourSegmentationInput.txt\n");
     prompt_and_exit(1);
   }

   printf("Example of how to use openCV to perform colour segmentation.\n\n");

   do {
      end_of_file = fscanf(fp_in, "%s", filename);

      if (end_of_file != EOF) {

         inputBGRImage = imread(filename, CV_LOAD_IMAGE_UNCHANGED);
         if(inputBGRImage.empty()) {
            cout << "can not open " << filename << endl;
            prompt_and_exit(-1);
         }

         CV_Assert(inputBGRImage.type() == CV_8UC3 ); // make sure we are dealing with a colour image
         printf("Click on a sample point in the input image.\n");
         printf("When finished with this image, press any key to continue ...\n");

         /* Create a window for input and display it */
         namedWindow(input_window_name, CV_WINDOW_AUTOSIZE );
         setMouseCallback(input_window_name, getSamplePoint);    // use this callback to get the colour components of the sample point
         imshow(input_window_name, inputBGRImage);

         /* convert the BGR image to HLS to facilitate hue-saturation segmentation */
         cvtColor(inputBGRImage, inputHLSImage, CV_BGR2HLS);

         /* Create a window for segmentation based on hue and saturation thresholding */
         namedWindow(segmented_window_name, CV_WINDOW_AUTOSIZE );
         resizeWindow(segmented_window_name,0,0); // this forces the trackbar to be as small as possible (and to fit in the window)
         createTrackbar( "Hue Range", segmented_window_name, &hueRange,        max_hue_range,        colourSegmentation);
         createTrackbar( "Sat Range", segmented_window_name, &saturationRange, max_saturation_range, colourSegmentation);

         /* display a zero output */
         outputImage = Mat::zeros(inputBGRImage.rows, inputBGRImage.cols, inputBGRImage.type());
         imshow(segmented_window_name, outputImage);

         /* now wait for user interaction - mouse click to change the colour sample or trackbar adjustment to change the thresholds */
         number_of_sample_points = 0;
         do {
            waitKey(30);
         } while (!_kbhit());

         getchar(); // flush the buffer from the keyboard hit

         destroyWindow(input_window_name);
         destroyWindow(segmented window name);
      }
   } while (end_of_file != EOF);

   fclose(fp_in);

   return 0;
}
```

```
/*
  Example use of openCV to perform colour segmentation

  The user must interactively select the colour sample that will form the basis of the segmentation.
  The user can also adjust the hue and saturation tolerances on that sample.
  -------------------------------------------------------------------------------------------------
  Implementation file

  David Vernon
  1 June 2017
*/
#include "colourSegmentation.h"

void colourSegmentation(int, void*) {

   extern Mat inputBGRImage;
   extern Mat inputHLSImage;
   extern int hueRange;
   extern int saturationRange;
   extern Point2f sample_point;
   extern char* segmented_window_name;
   extern int number_of_sample_points;

   Mat segmentedImage;
   int row, col;

   int hue;
   int saturation;
   int h;
   int s;

   bool debug = false;
   /* now get the sample point */
   if (number_of_sample_points == 1) {

      segmentedImage = inputBGRImage.clone();

      hue        = inputHLSImage.at<Vec3b>((int)sample_point.y,(int)sample_point.x)[0]; // note order of indices
      saturation = inputHLSImage.at<Vec3b>((int)sample_point.y,(int)sample_point.x)[2]; // note order of indices

      if (debug) {
         printf("Sample point (%f, %f) Hue: %d  Saturation: %d\n", sample_point.y, sample_point.x, hue, saturation); // note order of indices
         printf("Hue range: %d  Saturation range: %d\n", hueRange, saturationRange);                                 // note order of indices
      }

      /* now perform segmentation */
      for (row=0; row < inputBGRImage.rows; row++) {
         for (col=0; col < inputBGRImage.cols; col++) {

            h = inputHLSImage.at<Vec3b>(row,col)[0];
            s = inputHLSImage.at<Vec3b>(row,col)[2];

            /* Note: 0 <= h <= 180 ... NOT as you'd expect: 0 <= h <= 360  */
            if ((((h >= hue     - hueRange) && (h <= hue     + hueRange)) ||
                 ((h >= hue+180 - hueRange) && (h <= hue+180 + hueRange)) ||
                 ((h >= hue-180 - hueRange) && (h <= hue-180 + hueRange)))
                 &&
                 ((s >= (saturation - saturationRange)) && (s <= (saturation + saturationRange)))) {
               segmentedImage.at<Vec3b>(row,col)[0] = inputBGRImage.at<Vec3b>(row,col)[0];
               segmentedImage.at<Vec3b>(row,col)[1] = inputBGRImage.at<Vec3b>(row,col)[1];
               segmentedImage.at<Vec3b>(row,col)[2] = inputBGRImage.at<Vec3b>(row,col)[2];
            }
            else {
               segmentedImage.at<Vec3b>(row,col)[0] = 0;
               segmentedImage.at<Vec3b>(row,col)[1] = 0;
               segmentedImage.at<Vec3b>(row,col)[2] = 0;
            }
         }
      }
      imshow(segmented_window_name, segmentedImage);
   }

   if (debug) printf("Leaving colourSegmentation() \n");

}
```

```
void getSamplePoint( int event, int x, int y, int, void* ) {

   extern char*   input_window_name;
   extern Mat     inputBGRImage;
   extern Point2f sample_point;
   extern int     number_of_sample_points;
   Mat            inputImageCopy;
   int crossHairSize = 10;

   if (event != EVENT_LBUTTONDOWN) {
      return;
   }
   else {
      number_of_sample_points = 1;
      sample_point.x = (float) x;
      sample_point.y = (float) y;

      inputImageCopy = inputBGRImage.clone();

      line(inputImageCopy,Point(x-crossHairSize/2,y),Point(x+crossHairSize/2,y),Scalar(0, 255, 0),1, CV_AA); // Green
      line(inputImageCopy,Point(x,y-crossHairSize/2),Point(x,y+crossHairSize/2),Scalar(0, 255, 0),1, CV_AA);

      imshow(input_window_name, inputImageCopy); // show the image with the cross-hairs

      colourSegmentation(0, 0); // Show the segmented image for new colour sample and current thresholds
   }
}
```

Demo

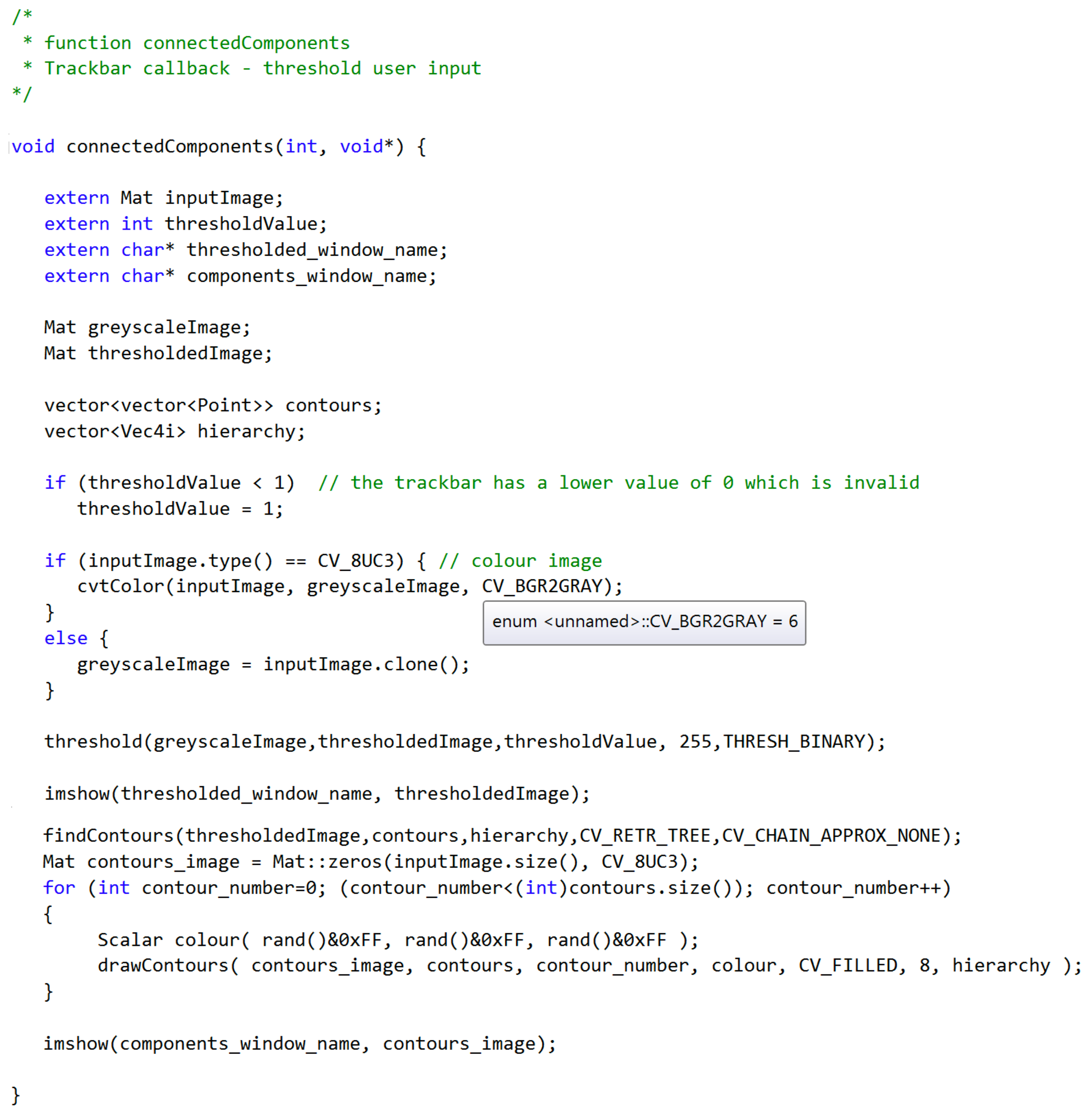
The following code is taken from the connectedComponents example application

See:

```
connectedComponents.h
connectedComponentsImplementation.cpp
connectedComponentsApplication.cpp
```

To run the example:

Ubuntu 16.04: `rosrun module5 connectedComponents`
Windows 10: double-click `C:\CORO\lectures\bin\connectedComponents`



```
/*
 * function connectedComponents
 * Trackbar callback - threshold user input
*/

void connectedComponents(int, void*) {

   extern Mat inputImage;
   extern int thresholdValue;
   extern char* thresholded_window_name;
   extern char* components_window_name;

   Mat greyscaleImage;
   Mat thresholdedImage;

   vector<vector<Point>> contours;
   vector<Vec4i> hierarchy;

   if (thresholdValue < 1)  // the trackbar has a lower value of 0 which is invalid
      thresholdValue = 1;

   if (inputImage.type() == CV_8UC3) { // colour image
      cvtColor(inputImage, greyscaleImage, CV_BGR2GRAY);
   }
   else {
      greyscaleImage = inputImage.clone();
   }

   threshold(greyscaleImage,thresholdedImage,thresholdValue, 255,THRESH_BINARY);

   imshow(thresholded_window_name, thresholdedImage);

   findContours(thresholdedImage,contours,hierarchy,CV_RETR_TREE,CV_CHAIN_APPROX_NONE);
   Mat contours_image = Mat::zeros(inputImage.size(), CV_8UC3);
   for (int contour_number=0; (contour_number<(int)contours.size()); contour_number++)
   {
        Scalar colour( rand()&0xFF, rand()&0xFF, rand()&0xFF );
        drawContours( contours_image, contours, contour_number, colour, CV_FILLED, 8, hierarchy );
   }

   imshow(components_window_name, contours_image);

}
```

enum <unnamed>::CV_BGR2GRAY = 6

Demo

The following code is taken from the grabCut example application

See:

```
grabCut.h
grabCutImplementation.cpp
grabCutApplication.cpp
```

To run the example:

Ubuntu 16.04: `rosrun module5 grabCut`
Windows 10: double-click `C:\CORO\lectures\bin\grabCut`

```
/*
 * function grabCut
 * Trackbar callback - number of iterations user input
*/
void performGrabCut(int, void*) {

   extern Mat   inputImage;
   extern int   numberOfIterations;
   extern int   number_of_control_points;
   extern char* grabcut_window_name;
   Mat result;           // segmentation result
   Mat bgModel,fgModel; // the models (hard constraints)

   if (numberOfIterations < 1)  // the trackbar has a lower value of 0 which is invalid
      numberOfIterations = 1;

   /* get two control points (top left and bottom right) and rectangle */
   do {
      waitKey(30);
   } while (number_of_control_points < 2);

    /* GrabCut segmentation                                                            */
    /* see: http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html#grabcut */
    grabCut(inputImage,         // input image
            result,             // segmentation result (4 values); can also be used as an input mask providing constraints
            rect,               // rectangle containing foreground
            bgModel,fgModel,    // for internal use ... allows continuation of iterative solution on subsequent calls
            numberOfIterations, // number of iterations
            GC_INIT_WITH_RECT); // use rectangle

    /* Get the pixels marked as likely foreground */
    compare(result,GC_PR_FGD,result,CMP_EQ);

    /* Generate output image */
    Mat foreground(inputImage.size(),CV_8UC3,cv::Scalar(255,255,255));
    inputImage.copyTo(foreground,result); // use result to mask out the background pixels

    imshow(grabcut_window_name, foreground);
}
```