Chapter 7

Programming OpenCV

On successful completion of this course, students will be able to:

- Explain the morphological filtering.
- Explain MeanShift and CamShift algorithms.
- Develop a program to tracking an object using MeanShift() and CamShift().

Introduction

Morphological filtering is a theory developed in the 1960s for the analysis and processing of discrete images. It defines a series of operators which transform an image by probing it with a predefined shape element. The way this shape element intersects the neighborhood of a pixel determines the result of the operation. Tracking an object is important features for intelligent robotics. Meansift is an algorithm that finds an object in a backprjected histograma image. Camshift is a histogram backprojection-based object tracking algorithm that uses meanshift at its heart. It takes the detection window output by meanshift and figures out the best size and rotation of that window to track the object.

Morphological Filtering

As morphological filters usually work on binary images, we will use a binary image produced through thresholding. However, since in morphology, the convention is to have foreground objects represented by high (white) pixel values and background by low (black) pixel values, we have negated the image. Morphological operations are a set of operations that process images based on shapes. Morphological operations apply a structuring element to an input image and generate an output image. The most basic morphological operations are two: Erosion and Dilation. They have a wide array of uses, i.e.:

- Removing noise.
- Isolation of individual elements and joining disparate elements in an image.
- Finding of intensity bumps or holes in an image.

Erosion and dilation are implemented in OpenCV as simple functions which are cv::erode and cv::dilate. The opening and closing filters are simply defined

in terms of the basic erosion and dilation operations. Closing is defined as the erosion of the dilation of an image, and Opening is defined as the dilation of the erosion of an image. The code below show an example of erotion and dilation:

Morphology.cpp:

```
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "highgui.h"
#include <stdlib.h>
#include <stdio.h>

using namespace cv;

/// Global variables
Mat src, erosion_dst, dilation_dst;

int erosion_elem = 0;
int erosion_size = 0;
int dilation_elem = 0;
int dilation_size = 0;
int const max_elem = 2;
int const max_kernel_size = 21;

/** Function Headers */
void Erosion( int, void* );
void Dilation( int, void* );

int main( int argc, char** argv )
{
 /// Load an image
   // src = imread( argv[1] ); // if using command prompt
      src = imread("flower.jpg");

 if( !src.data )
 { return -1; }

 /// Create windows
 namedWindow( "Erosion Demo", CV_WINDOW_AUTOSIZE );
 namedWindow( "Dilation Demo", CV_WINDOW_AUTOSIZE );
 cvMoveWindow( "Dilation Demo", src.cols, 0 );
```

```
  /// Create Erosion Trackbar
  createTrackbar( "Element:\n 0: Rect \n 1: Cross \n 2: Ellipse",
"Erosion Demo",
        &erosion_elem, max_elem,
        Erosion );

  createTrackbar( "Kernel size:\n 2n +1", "Erosion Demo",
        &erosion_size, max_kernel_size,
        Erosion );

  /// Create Dilation Trackbar
  createTrackbar( "Element:\n 0: Rect \n 1: Cross \n 2: Ellipse",
"Dilation Demo",
        &dilation_elem, max_elem,
        Dilation );

  createTrackbar( "Kernel size:\n 2n +1", "Dilation Demo",
        &dilation_size, max_kernel_size,
        Dilation );

  /// Default start
  Erosion( 0, 0 );
  Dilation( 0, 0 );

  waitKey(0);
  return 0;
 }

 void Erosion( int, void* )
 {
  int erosion_type;
  if( erosion_elem == 0 ){ erosion_type = MORPH_RECT; }
  else if( erosion_elem == 1 ){ erosion_type = MORPH_CROSS; }
  else if( erosion_elem == 2) { erosion_type = MORPH_ELLIPSE; }

  Mat element = getStructuringElement( erosion_type,
      Size( 2*erosion_size + 1, 2*erosion_size+1 ),
      Point( erosion_size, erosion_size ) );

  /// Apply the erosion operation
  erode( src, erosion_dst, element );
```

```
 imshow( "Erosion Demo", erosion_dst );
}

void Dilation( int, void* )
{
 int dilation_type;
 if( dilation_elem == 0 ){ dilation_type = MORPH_RECT; }
 else if( dilation_elem == 1 ){ dilation_type = MORPH_CROSS; }
 else if( dilation_elem == 2) { dilation_type = MORPH_ELLIPSE; }

 Mat element = getStructuringElement( dilation_type,
       Size( 2*dilation_size + 1, 2*dilation_size+1 ),
       Point( dilation_size, dilation_size ) );
 /// Apply the dilation operation
 dilate( src, dilation_dst, element );
 imshow( "Dilation Demo", dilation_dst );
}
```

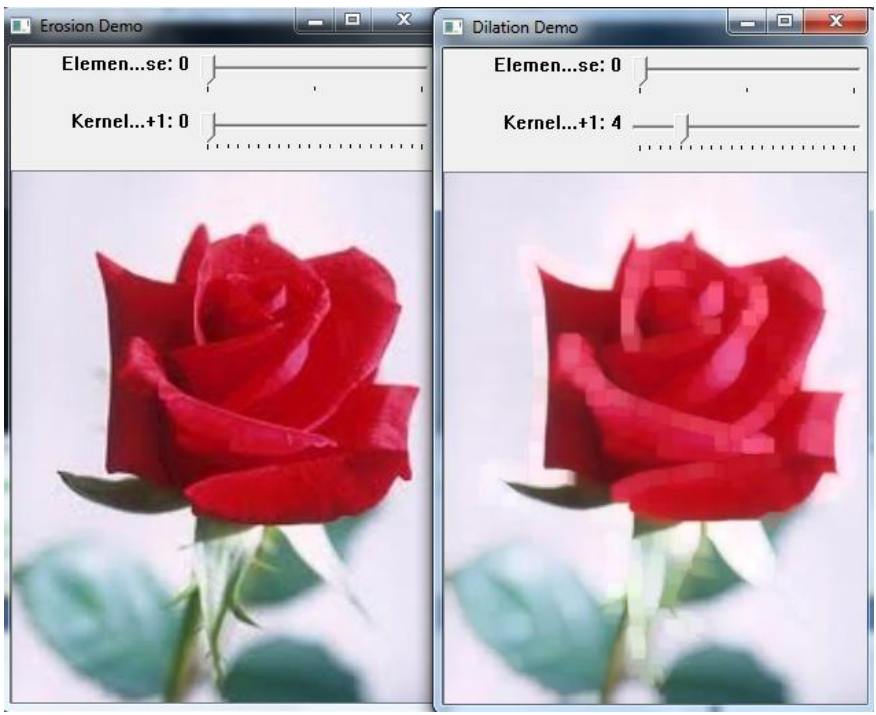

Figure 7.1 *result of erotion and dilation.*

Camshift for Tracking Object

Meansift is an algorithm that finds an object in a backprjected histogram image. Camshift is a histogram backprojection-based object tracking algorithm that uses meanshift at its heart. It takes the detection window output by meanshift and figures out the best size and rotation of that window to track the object. OpenCV fuctions meanShift() and CamShift() implement these algorithm.

Camshift.cpp:

```
#include "stdafx.h"
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <iostream>
#include <ctype.h>

using namespace cv;
using namespace std;

Mat image;

bool backprojMode = false;
bool selectObject = false;
int trackObject = 0;
bool showHist = true;
Point origin;
Rect selection;
int vmin = 10, vmax = 256, smin = 30;

static void onMouse( int event, int x, int y, int, void* )
{
 if( selectObject )
 {
  selection.x = MIN(x, origin.x);
  selection.y = MIN(y, origin.y);
  selection.width = std::abs(x - origin.x);
  selection.height = std::abs(y - origin.y);
```

```
    selection &= Rect(0, 0, image.cols, image.rows);
  }

  switch( event )
  {
  case CV_EVENT_LBUTTONDOWN:
   origin = Point(x,y);
   selection = Rect(x,y,0,0);
   selectObject = true;
   break;
  case CV_EVENT_LBUTTONUP:
   selectObject = false;
   if( selection.width > 0 && selection.height > 0 )
    trackObject = -1;
   break;
  }
}

const char* keys =
{
 "{1|  | 0 | camera number}"
};

int main( int argc, const char** argv )
{

 VideoCapture cap;
 Rect trackWindow;
 int hsize = 16;
 float hranges[] = {0,180};
 const float* phranges = hranges;
 CommandLineParser parser(argc, argv, keys);
 int camNum = parser.get<int>("1");

 cap.open(camNum);
      IplImage img = image;

 if( !cap.isOpened() )
 {
```

```
    cout << "***Could not initialize capturing...***\n";
    parser.printParams();
    return -1;
  }

  namedWindow( "Histogram", 0 );
  namedWindow( "VISION-BASED GRASPING FOR ARM ROBOT", 0 );
  setMouseCallback("VISION-BASED GRASPING FOR ARM ROBOT",
onMouse, 0 );
  createTrackbar( "Vmin", "VISION-BASED GRASPING FOR ARM ROBOT",
&vmin, 256, 0 );
  createTrackbar( "Vmax", "VISION-BASED GRASPING FOR ARM ROBOT",
&vmax, 256, 0 );
  createTrackbar( "Smin", "VISION-BASED GRASPING FOR ARM ROBOT",
&smin, 256, 0 );

  Mat frame, hsv, hue, mask, hist, histimg = Mat::zeros(200, 320,
CV_8UC3), backproj;
  bool paused = false;

  for(;;)
  {
    if( !paused )
    {
      cap >> frame;
      if( frame.empty() )
        break;
    }

    frame.copyTo(image);

    if( !paused )
    {
      cvtColor(image, hsv, CV_BGR2HSV);

      if( trackObject )
      {
        int _vmin = vmin, _vmax = vmax;

        inRange(hsv, Scalar(0, smin, MIN(_vmin,_vmax)),
          Scalar(180, 256, MAX(_vmin, _vmax)), mask);
        int ch[] = {0, 0};
```

```
        hue.create(hsv.size(), hsv.depth());
        mixChannels(&hsv, 1, &hue, 1, ch, 1);

        if( trackObject < 0 )
        {
         Mat roi(hue, selection), maskroi(mask, selection);
         calcHist(&roi, 1, 0, maskroi, hist, 1, &hsize, &phranges);
         normalize(hist, hist, 0, 255, CV_MINMAX);

         trackWindow = selection;
         trackObject = 1;

         histimg = Scalar::all(0);
         int binW = histimg.cols / hsize;
         Mat buf(1, hsize, CV_8UC3);
         for( int i = 0; i < hsize; i++ )
          buf.at<Vec3b>(i) =
Vec3b(saturate_cast<uchar>(i*180./hsize), 255, 255);
         cvtColor(buf, buf, CV_HSV2BGR);

         for( int i = 0; i < hsize; i++ )
         {
          int val =
saturate_cast<int>(hist.at<float>(i)*histimg.rows/255);
          rectangle( histimg, Point(i*binW,histimg.rows),
           Point((i+1)*binW,histimg.rows - val),
           Scalar(buf.at<Vec3b>(i)), -1, 8 );
         }
        }

        calcBackProject(&hue, 1, 0, hist, backproj, &phranges);
        backproj &= mask;
        RotatedRect trackBox = CamShift(backproj, trackWindow,
          TermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10,
1 ));
        if( trackWindow.area() <= 1 )
        {
         int cols = backproj.cols, rows = backproj.rows, r =
(MIN(cols, rows) + 5)/6;
         trackWindow = Rect(trackWindow.x - r, trackWindow.y - r,
```

```
          trackWindow.x + r, trackWindow.y + r) &
         Rect(0, 0, cols, rows);
       }

       if( backprojMode )
        cvtColor( backproj, image, CV_GRAY2BGR );
       ellipse( image, trackBox, Scalar(0,0,255), 3, CV_AA );
          // cvRectangle
(&image,cvPoint(trackBox.center),cvPoint(trackBox.center),CV_RGB(
255,0,0), 3);

          //cvRectangle(&frame,
cvPoint(trackWindow.x,trackWindow.y),cvPoint(trackWindow.x+20,tra
ckWindow.y+20), CV_RGB(255,0,0), 3 );

      }
    }
    else if( trackObject < 0 )
     paused = false;

    if( selectObject && selection.width > 0 && selection.height >
0 )
    {
     Mat roi(image, selection);
     bitwise_not(roi, roi);
    }

    imshow( "VISION-BASED GRASPING FOR ARM ROBOT", image );
    imshow( "Histogram", histimg );

    char c = (char)waitKey(10);
    if( c == 27 )
     break;
    switch(c)
    {
    case 'b':
     backprojMode = !backprojMode;
     break;
    case 'c':
     trackObject = 0;
     histimg = Scalar::all(0);
```

```
      break;
    case 'h':
      showHist = !showHist;
      if( !showHist )
        destroyWindow( "Histogram" );
      else
        namedWindow( "Histogram", 1 );
      break;
    case 'p':
      paused = !paused;
      break;
    default:
      ;
    }
  }

  return 0;
}
```

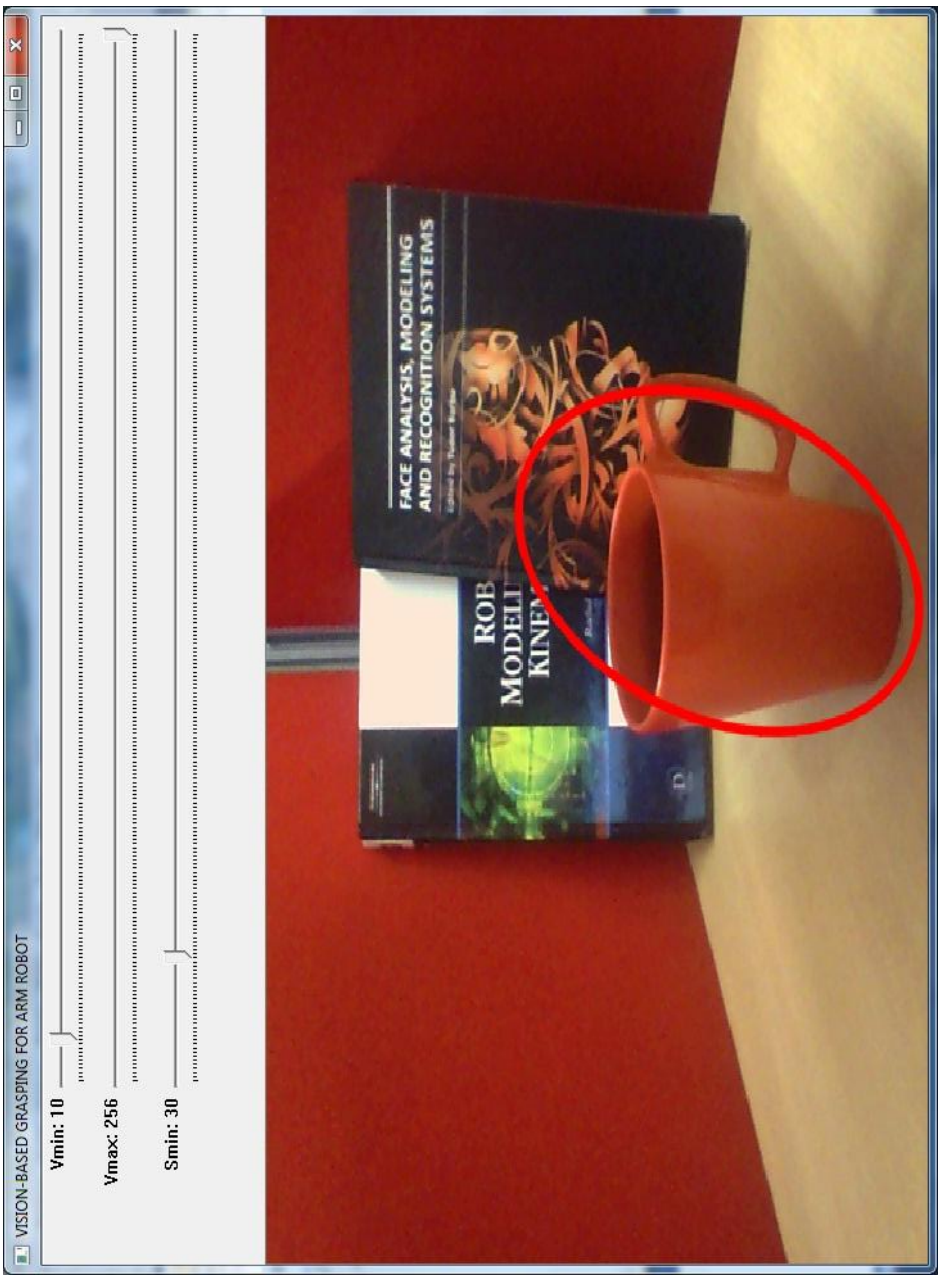

Figure 7.2 *Result of object detection.*

References

[1] Adrian Kaehler & Garry Bradksy, Learning OpenCV: Computer Vision in C++ with the OpenCV Library, O'Reilly PUblisher, 2014.

[2] Samarth Brahmbatt, Practical OpenCV, Technology in Action Publisher, 2013.

[3] Daniel bagio et al., Mastering OpenCV with Practical Computer Vision Project, Packt Publisher, 2012.