OpenCV – 비디오 처리

김성영교수
금오공과대학교
컴퓨터공학과

학습 내용

- Reading video sequences
- Seeking video sequences
- Writing video sequences
- Foreground extraction

Reading video sequences

`VideoCapture`: class for video capturing from video files or cameras

Constructors

```
VideoCapture::VideoCapture(const string& filename)
VideoCapture::VideoCapture(int device)
```

Checking if video capturing has been initialized already

```
bool VideoCapture::isOpened()
```

Grabs, decodes and returns the next video frame

```
bool VideoCapture::read(Mat& image)
VideoCapture& VideoCapture::operator>>(Mat& image)
```

```
bool VideoCapture::grab()
          +
bool VideoCapture::retrieve(Mat& image, int channel=0)
```

Returns and set the specified VideoCapture property

```
double VideoCapture::get(int propId)
bool VideoCapture::set(int propId, double value)
```

- **CV_CAP_PROP_POS_MSEC** Current position of the video file in milliseconds or video capture timestamp.
- **CV_CAP_PROP_POS_FRAMES** 0-based index of the frame to be decoded/captured next.
- **CV_CAP_PROP_POS_AVI_RATIO** Relative position of the video file: 0 - start of the film, 1 - end of the film.
- **CV_CAP_PROP_FRAME_WIDTH** Width of the frames in the video stream.
- **CV_CAP_PROP_FRAME_HEIGHT** Height of the frames in the video stream.
- **CV_CAP_PROP_FPS** Frame rate.
- **CV_CAP_PROP_FOURCC** 4-character code of codec.
- **CV_CAP_PROP_FRAME_COUNT** Number of frames in the video file.
- **CV_CAP_PROP_FORMAT** Format of the Mat objects returned by **retrieve()** .
- **CV_CAP_PROP_MODE** Backend-specific value indicating the current capture mode.
- **CV_CAP_PROP_BRIGHTNESS** Brightness of the image (only for cameras).
- **CV_CAP_PROP_CONTRAST** Contrast of the image (only for cameras).
- **CV_CAP_PROP_SATURATION** Saturation of the image (only for cameras).
- **CV_CAP_PROP_HUE** Hue of the image (only for cameras).
- **CV_CAP_PROP_GAIN** Gain of the image (only for cameras).
- **CV_CAP_PROP_EXPOSURE** Exposure (only for cameras).

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp“

using namespace cv;

int main()
{
    // Open the video file
    VideoCapture capture( "sample.avi" );

    // check if video successfully opened
    if (!capture.isOpened())
        return 1;

    // Get the frame rate
    double rate = capture.get( CV_CAP_PROP_FPS );

    bool stop(false);
    Mat frame; // current video frame
    namedWindow( "video" );

    int delay = 1000 / rate;
```

```
    // for all frames in video
    while( !stop )
    {
        // read next frame if any
        if( !capture.read(frame) )
            break;

        imshow( "video", frame );

        // wait by a delay
        // or press key to stop
        if( waitKey(delay) >=0 )
            stop = true;
    }

    waitKey();
}
```

Seeking video sequences

```
void play( int pos, void *capture );

char* trackbarName = "Current position:";
char* windowName = "video";

int main() {
    // Open the video file
    VideoCapture capture;
    capture.open( "sample.avi" );

    // check if video successfully opened
    if (!capture.isOpened())return 1;

    // Get the frame rate
    double rate = capture.get( CV_CAP_PROP_FPS );
    // Get the video length
    double length = capture.get( CV_CAP_PROP_FRAME_COUNT );

    // Create a window
    namedWindow( windowName, CV_WINDOW_AUTOSIZE );

    // Create a Trackbar
    int curPosition = 0;
    createTrackbar( trackbarName, windowName, &curPosition,
            (int)length, play, &capture );
```

```
    bool stop(false);
    Mat frame; // current video frame
    int delay = 1000 / rate;
    // for all frames in video
    while( !stop )
    {
        // read next frame if any
        if( !capture.grab() )
            break;
        if( !capture.retrieve(frame) )
            break;

        imshow( windowName, frame );
        setTrackbarPos( trackbarName, windowName, curPosition );
        curPosition++;

        if( waitKey(delay) >=0 )
            stop = true;
    }

    waitKey();
}
```

```
void play( int pos, void *capture )
{
    VideoCapture *nCapture = (VideoCapture *)capture;

    // set current position for play header
    nCapture->set( CV_CAP_PROP_POS_FRAMES, pos );
}
```

Writing video sequences

VideoWriter: class for video writering

Constructors

```
VideoWriter::VideoWriter(const string& filename,
                         int fourcc, double fps,
                         size frameSize, bool isColor=true)
```

fourcc:

CV_FOURCC('P','I','M,'1'): MPEG-1

CV_FOURCC('M','J','P','G') : motion-jpeg

CV_FOURCC('X', 'V', 'I', 'D'): XVID

-1: pop up a window

Checking if video writer has been successfully initialized

```
bool VideoWriter::isOpened()
```

Write the next video frame

```
void VideoWriter::write(const Mat& image)
VideoWriter& VideoWriter::operator<<(const Mat& image)
```

```
#include "opencv2/core/core.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <iostream>
using namespace cv;

void play( int pos, void *capture );
int getCodec( VideoCapture &capture, char codec[4] );
Size getFrameSize( VideoCapture &capture );

char* trackbarName = "Current position:";
char* windowName = "video";

int main() {
    // Open the video file
    VideoCapture capture( "sample.avi" );
    // check if video successfully opened
    if( !capture.isOpened() ) return 1;

    // Get the frame rate
    double rate = capture.get( CV_CAP_PROP_FPS );
    // Get the video length
    double length = capture.get( CV_CAP_PROP_FRAME_COUNT );

    // Create a window
    namedWindow( windowName, CV_WINDOW_AUTOSIZE );
```

```
// Create a Trackbar
int curPosition = 0;
createTrackbar( trackbarName, windowName, &curPosition,
            int)length, play, &capture );

bool stop(false);
Mat frame; // current video frame
int delay = 1000 / rate;

// open output video
char c[4];
VideoWriter writer( "output.avi", // filename
            getCodec(capture, c), // codec to be used
            rate,      // frame rate of the video
            getFrameSize(capture), // frame size
            1 );

// check if video successfully opened
if( !writer.isOpened() )
    return 1;
std::cout << "Codec: " << c[0] << c[1] <<
        c[2] << c[3] << std::endl;
```

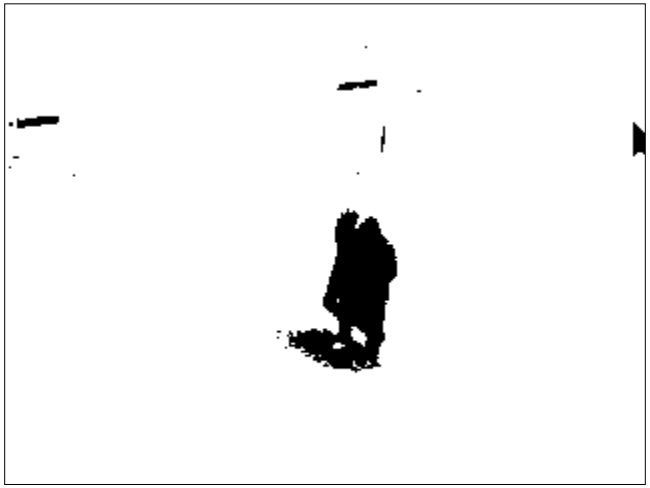
```
    // for all frames in video
    while( !stop ){
        // read next frame if any
        if( !capture.read(frame) )
            break;
        imshow( windowName, frame );
        setTrackbarPos( trackbarName, windowName, curPosition );
        curPosition++;

        writer.write( frame );

        if( waitKey(delay) >=0 )
            stop = true;
    }
    waitKey();
}

void play( int pos, void *capture )
{
    VideoCapture *nCapture = (VideoCapture *)capture;

    // set current position for play header
    nCapture->set( CV_CAP_PROP_POS_FRAMES, pos );
}
```

```
// get the codec of input video
int getCodec( VideoCapture &capture, char codec[4] ){
    union {
        int value;
        char code[4];
    } returned;
    returned.value = static_cast<int>(
                capture.get(CV_CAP_PROP_FOURCC) );
    codec[0] = returned.code[0];
    codec[1] = returned.code[1];
    codec[2] = returned.code[2];
    codec[3] = returned.code[3];
    return returned.value;
}
// return the size of the video frame
Size getFrameSize( VideoCapture &capture ){
    int w = static_cast<int>(
                capture.get(CV_CAP_PROP_FRAME_WIDTH) );
    int h = static_cast<int>(
                capture.get(CV_CAP_PROP_FRAME_HEIGHT) );
    return Size( w, h );
}
```

Foreground Extraction

Background

Current frame



Creating dynamic background model

running average (moving average)

$$\mu_t = (1-\alpha) \cdot \mu_{t-1} + \alpha \cdot p_t$$

α: learning rate

```
void accumulateWeighted(
        InputArray src,
        InputOutputArray dst,
        double alpha,
        InputArray mask=noArray()
)
```

calculates the weighted sum of the input image **src** and the accumulator **dst** so that **dst** becomes a running average of a frame sequence

$$\texttt{dst}(x,y) \leftarrow (1-\texttt{alpha}) \cdot \texttt{dst}(x,y) + \texttt{alpha} \cdot \texttt{src}(x,y) \quad \text{if} \quad \texttt{mask}(x,y) \neq 0$$

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
using namespace cv;

int main() {
    // Open the video file
    VideoCapture capture( "sample.avi" );
    // check if video successfully opened
    if( !capture.isOpened() )
        return 1;

    // Get the frame rate
    double rate = capture.get( CV_CAP_PROP_FPS );

    bool stop(false);

    namedWindow( "video" );

    // Delay between each frame
    // corresponds to video frame rate
    int delay = 1000 / rate;
    Mat frame, gray; // current video frame
    Mat background, backImage, foreground;
    Mat output;
    double thres=25, learningRate=0.01;
```

```
// for all frames in video
while( !stop )
{
    // read next frame if any
    if( !capture.read(frame) )
        break;
    cvtColor( frame, gray, CV_BGR2GRAY );

    // initialize background to 1st frame
    if( background.empty() )
        gray.convertTo( background, CV_32F );

    // convert background to 8U
    background.convertTo( backImage, CV_8U );

    // difference between current image and background
    absdiff( backImage, gray, foreground );

    // apply threshold to foreground image
    threshold( foreground, output, thres,
            255, THRESH_BINARY_INV );
```

```
        // accumulate background
        accumulateWeighted( gray, background,
                learningRate, output );

        imshow( "video", output );

        // introduce a delay
        // or press key to stop
        if( waitKey(delay) >=0 )
            stop = true;
    }

    waitKey( );
}
```

The Mixture of Gaussian method

$$\mu_t = (1-\alpha)\cdot\mu_{t-1} + \alpha\cdot p_t$$

$$\sigma_t^2 = (1-\alpha)\cdot\sigma_{t-1}^2 + \alpha\cdot(p_t-\mu_t)^2$$

BackgroundSubtractorMOG:

class for Gaussian Mixture-based Background/Foreground Segmentation Algorithm

```
BackgroundSubtractorMOG::BackgroundSubtractorMOG(
)

BackgroundSubtractorMOG::BackgroundSubtractorMOG(
        int history,
        int nmixtures,
        double backgroundRatio,
        double noiseSigma=0
)
```

```
void BackgroundSubtractorMOG::operator()(
      InputArray image,
      OutputArray fgmask,
      double learningRate=0
)
```

Updates the background model and returns the foreground mask

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include <opencv2/video/background_segm.hpp>
using namespace cv;

int main(){
    // Open the video file
    VideoCapture capture( "sample.avi" );
    // check if video successfully opened
    if( !capture.isOpened() )
        return 1;

    // Get the frame rate
    double rate = capture.get( CV_CAP_PROP_FPS );
    bool stop(false);

    namedWindow( "video" );

    int delay = 1000 / rate;
    BackgroundSubtractorMOG mog;
    Mat frame; // current video frame
    Mat foreground;
    double learningRate=0.01;
```

```
    // for all frames in video
    while( !stop )
    {
        // read next frame if any
        if( !capture.read(frame) )
            break;

        // update the background
        // and return the foreground
        mog( frame, foreground, learningRate );

        // apply threshold to foreground image
        threshold( foreground, foreground, 128, 255,
                THRESH_BINARY_INV );
        imshow( "video", foreground );

        if( waitKey(delay) >=0 )
            stop = true;
    }
    waitKey( );
}
```

Reference

- R. Laganière, **OpenCV2 Computer Vision: Application Programming Cookbook,** PACKT Publishing, 2011
- G. Bradski and A. Kaebler, **Learning OpenCV: Computer Vision with the OpenCV Library**, O'REILLY, 2008
- http://docs.opencv.org