HW1&2 Report

Computer Vision
Section 02
21800181 Kim Jisu

- **HW 1**
  - Purpose

This is a program that reads a video “background.mp4” and displays video for the first 3 seconds and prints the current frame number and the total frame number. The key point is to find proper input parameter value for waitKey().

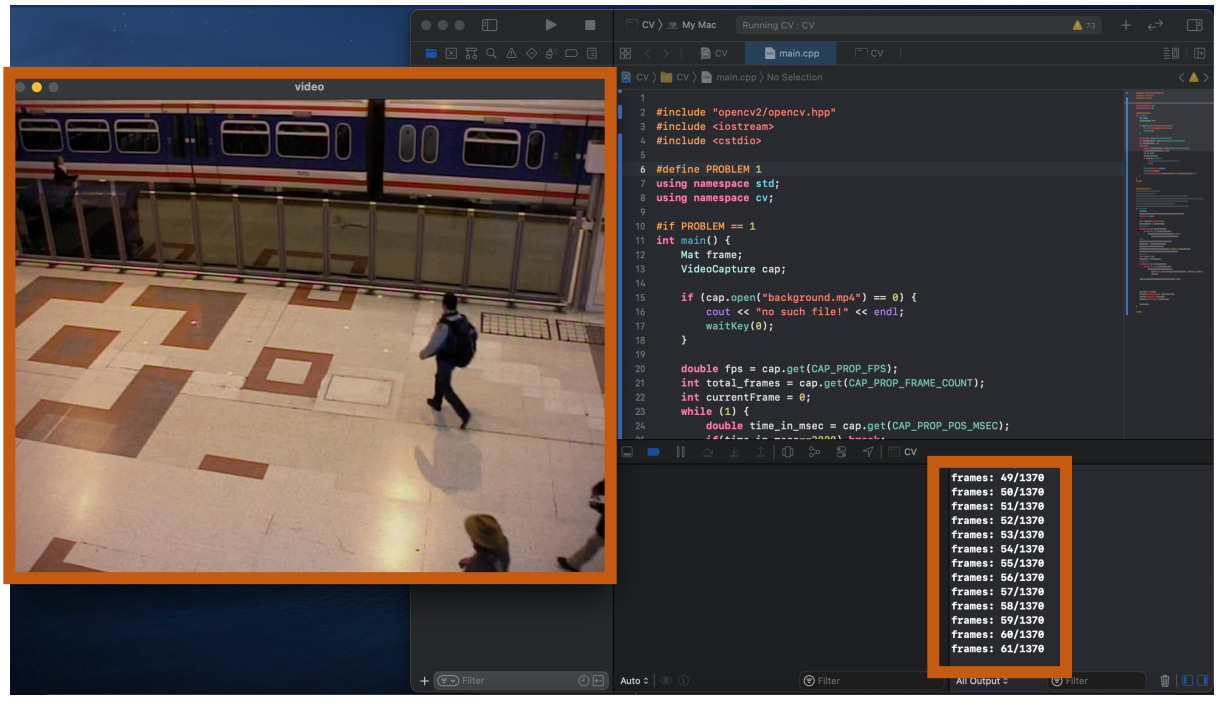
  - Principle

fps is frames per second.
CAP_PROP_FPS is frame rate.
CAP_PROP_FRAME_COUNT is number of frames in the video file.
CAP_PROP_POS_MSEC is current position of the video file in milliseconds or video capture timestamp.

  - Process

virtual bool cv::VideoCapture::open (const String & filename, int apiPreference = CAP_ANY)
This function opens a video file or a capturing device for video capturing. It returns true when the file has been successfully opened.
Virtual double cv::get(int propId) const
It returns the specified VideoCapture property.
bool cv::Mat::empty()const
Returns true if the array has no elements.
int cv::waitKey(int delay = 0)
Waits for a pressed key.

  - Result



The proper input parameter for waitKey() is 1000/fps. fps is frames per second. Therefore, the proper input parameter is one second divided by fps.

  - Code

```
#include "opencv2/opencv.hpp"
#include <iostream>
#include <cstdio>

using namespace std;
using namespace cv;


int main() {
    Mat frame;
    VideoCapture cap;
    if (cap.open("background.mp4") == 0) {
        cout << "no such file!" << endl;
        waitKey(0);
    }

    double fps = cap.get(CAP_PROP_FPS);
    int total_frames = cap.get(CAP_PROP_FRAME_COUNT);
    int currentFrame = 0;
    while (1) {
       double time_in_msec = cap.get(CAP_PROP_POS_MSEC);
       if(time_in_msec==3000) break;
       cap >> frame;
       currentFrame++;
       if (frame.empty()) {
          break;
       }
       imshow("video", frame);
       waitKey(1000/fps);
       cout<<"frames: "<<currentFrame<<"/"<<total_frames<<endl;
       }
}
```

- **HW 2**
  - Purpose

This is a program that reads image “lena.png” and performs image transformation (negative, log, gamma).

- Principle

Mat cv::imread(const String & filename, int flags = IMREAD_COLOR)

Loads an image from a file.

Mat cv::Mat::clone ()const

Creates a full copy of the array and the underlying data.

Mat convertTo(OutputArray m, int rtype, double alpha=1, double beta=0)

Converts an array to another data type with optional scaling.

int abs (int n)

Returns the absolute value of parameter n.

Void convertScaleAbs(InputArray src, OutputArray dst, double alpha=1, double beta=0)

Scales, calculates absolute values, and converts the result to 8-bit. On each element of the input array, the function convertScaleAbs performs three operations sequentially: scaling, taking an absolute value, conversion to an unsigned 8-bit type

- Process

1. Negative transformation

When the image range of intensity is 0 to L-1, the negative transformation of the image is ‘output = L-1-input’. For negative transformation, each pixel value is changed according to the following equation ‘255-value’.

2. Log transformation

Output = C * log (1 + input), C = constant

To use log function, the image pixel type must be a floating point. For this reason, the image was converted to 32bit float before performing log transformation. For log transformation, convert pixel type to float type and add +1 to the absolute value. After that, perform log transformation and normalization to numbers ranging from 0 to 255. Finally, execute convertScaleAbs() to multiply the constant and convert unsigned 8bit.

3. Gamma transformation

Output = C * {input^(gamma)}, C = constant

To Calculate gamma transformation, normalize pixel values from 0 to 1.0 before calculating. Finally, perform convertScaleAbs() to multiply the constant and convert unsigned 8bit.

- Result

- Code

```
#include "opencv2/opencv.hpp"
#include <iostream>
```

```
#include <cstdio>
using namespace std;
using namespace cv;

int main(){
	Mat img;
	Mat negative_img, log_float,log_img,gamma_img;
	double c = 1.5f;
	img = imread("Lena.png",0);
	negative_img = img.clone();

	for(int i = 0;i <img.rows;i++)
		for(int j = 0; j<img.cols;j++)
			negative_img.at<uchar>(i,j) = 255 - negative_img.at<uchar>(i,j);

	img.convertTo(log_float, CV_32F);
	log_float = abs(log_float)+1;
	log(log_float,log_float);
	normalize(log_float, log_float, 0, 255, NORM_MINMAX);
	convertScaleAbs(log_float, log_img,c,0);

	float gamma = 0.5;
	gamma_img = img.clone();

	for(int i = 0;i <img.rows;i++)
		for(int j = 0; j<img.cols;j++)
			gamma_img.at<uchar>(i,j) = pow((float)(gamma_img.at<uchar>(i,j) / 255.0), gamma) * 255.0f;

	convertScaleAbs(gamma_img, gamma_img, 1,0);
	imshow("img",img);
	imshow("negative_img", negative_img);
	imshow("log_img", log_img);
	imshow("gamma_img", gamma_img);
	waitKey();
}
```

- References

https://docs.opencv.org/master/