ECEN 5623 Exercise 4 Code For Plagiarism Check

Dhiraj Bennadi
Maitreyee Rao

March 20, 2022

Code Repository

Github Link

Code for Problem 5: capture.cpp

```
/*
 *
 *  Example by Sam Siewert
 *  Adapted by: Dhiraj Bennadi & Maitreyee Rao
 *
 */
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sched.h>
#include <sys/sysinfo.h>
#include <sys/types.h>
#include <syslog.h>


#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>

#include <chrono>
#include <pthread.h>

using namespace cv;
using namespace std;
```

```
#define HRES 640
#define VRES 480
#define SCHED (SCHED_FIFO)
int dev=0;
char argumentValue[20];

#define NUM_OF_THREADS 2

pthread_attr_t rt_sched_attr[NUM_OF_THREADS];
int rt_max_priority;
int rt_min_priority;
struct sched_param rt_param[NUM_OF_THREADS];
struct sched_param main_param;
pthread_attr_t main_attr;
pid_t mainpid;


// Transform display window
char timg_window_name[] = "Edge Detector Transform";
int lowThreshold=0;
int const max_lowThreshold = 100;
int kernel_size = 3;
int edgeThresh = 1;
int ratioForImage = 3;
Mat canny_frame, cdst, timg_gray, timg_grad;

void CannyThreshold(int, void*);
void simpleHoughElliptical(void);
void simpleHough(void);

IplImage* frame;
int frames;
std::chrono::duration<double, std::milli> elapsed_seconds;

/*mutex*/
pthread_mutex_t mtx;

bool syncVariable = false;
bool threadExit = false;

void *loggingThread(void *args)
{
    cout << "Logging Thread created" << endl;
    while(1)
    {
```

```
            if(threadExit == true)
            {
                break;
            }
            if(syncVariable == true)
            {
                syncVariable = false;
                pthread_mutex_lock(&mtx);
                //cout << "End - Start " << elapsed_seconds.count() * 1000 << " ms"
                //cout << "End - Start " << elapsed_seconds.count() << " ms" << endl
                syslog(LOG_NOTICE, "Frame %d complete in %lf ms\n", frames, elapsed_

                pthread_mutex_unlock(&mtx);
            }
            else
            {
                ;
            }
        }

        pthread_exit(NULL);

    }

    void *captureThread(void *args)
    {
        cout << "**********" << endl;
        cout << "Transformation Invoked: " << (char*)args << endl;
        cout << "**********" << endl;

        char t1[] = "canny";
        char t2[] = "houghelliptical";
        char t3[] = "houghlines";

        CvCapture* capture;
        namedWindow( timg_window_name, CV_WINDOW_AUTOSIZE );
        // Create a Trackbar for user to enter threshold
        createTrackbar( "Min Threshold:", timg_window_name, &lowThreshold, max_lowT

        capture = (CvCapture *)cvCreateCameraCapture(dev);
        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, HRES);
        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, VRES);

        while(1)
        {
            auto startTime = std::chrono::system_clock::now();
```

```
            frame=cvQueryFrame(capture);
            if(!frame) break;

            if(!memcmp((string*)args, t1, sizeof(t1)))
            {
                CannyThreshold(0, 0);
            }
            else if(!memcmp((string*)args, t2, sizeof(t2)))
            {
                simpleHoughElliptical();
            }
            else if(!memcmp((string*)args, t3, sizeof(t3)))
            {
                simpleHough();
            }
            else
            {
                threadExit = true;
                break;
            }

          //  simpleHoughElliptical();
            //CannyThreshold(0, 0);
            //simpleHough();
            pthread_mutex_lock(&mtx);
            auto endTime = std::chrono::system_clock::now();
            elapsed_seconds = endTime - startTime;
            syncVariable = true;
            frames++;

            pthread_mutex_unlock(&mtx);
            //cout << "Frame size = " << sizeof(frame) << endl;
            //cout << "End - Start " << elapsed_seconds.count() << endl;
            //syslog(LOG_NOTICE, "Frame %d complete in %lf ms\n", frames, elapsed_se
            if(frames == 150){
                printf("frames done\n");
                cvReleaseCapture(&capture);
                threadExit = true;
                break;
            }

        }

        pthread_exit(NULL);
    }
```

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

    //Mat mat_frame(frame);
    Mat mat_frame = cv::cvarrToMat(frame);

    cvtColor(mat_frame, timg_gray, CV_RGB2GRAY);

    /// Reduce noise with a kernel 3x3
    blur( timg_gray, canny_frame, Size(3,3) );

    /// Canny detector
    Canny( canny_frame, canny_frame, lowThreshold, lowThreshold*ratioForImage, k

    /// Using Canny's output as a mask, we display our result
    timg_grad = Scalar::all(0);

    //cout << "Size = " << sizeof(frame) << endl;

    mat_frame.copyTo( timg_grad, canny_frame);

    imshow( timg_window_name, timg_grad );


}


/*Printing Scheduler Details*/
void print_scheduler(void)
{
    int schedType;
    int scope;

    schedType = sched_getscheduler(getpid());

    switch(schedType)
    {
        case SCHED_FIFO:
            cout << "Pthread Policy is SCHED_FIFO" << endl;
        break;
        case SCHED_OTHER:
            cout << "Pthread Policy is SCHED_OTHER" << endl;
        break;
        case SCHED_RR:
```

```
            cout << "Pthread Policy is SCHED_RR" << endl;
        break;

        default:
            cout << "Pthread Policy is unknown" << endl;
        break;

    }

    pthread_attr_getscope(&main_attr, &scope);

    if(scope == PTHREAD_SCOPE_SYSTEM)
    {
        cout << "Pthread Scope System" << endl;
    }
    else if(scope == PTHREAD_SCOPE_PROCESS)
    {
        cout << "Pthread Scope Process" << endl;
    }
    else
    {
        cout << "Pthread Scope Unknown" << endl;
    }
}
```

```
int main( int argc, char** argv )
{
    int rc;
    // CvCapture* capture;
    // int dev=0;
    /*This code to to change the default scheduler to SCHED_FIFO*/
    openlog("ECEN REAL TIME:", 0, 0);
    syslog(LOG_NOTICE, "\nTEST START\n");
    cout << "This system has " << get_nprocs_conf() << " cores with " << get_npr

    /*Obtain the process ID*/
    mainpid = getpid();

    rt_max_priority = sched_get_priority_max(SCHED);
    rt_min_priority = sched_get_priority_min(SCHED);
```

```
print_scheduler();

rc = sched_getparam(mainpid, &main_param);

main_param.sched_priority = rt_max_priority;

if(rc = sched_setscheduler(getpid(), SCHED_FIFO , &main_param) < 0)
{
    perror("Warning: sched_setscheduler");
}
print_scheduler();

cout << "rt_max_priority : " << rt_max_priority << endl;
cout << "rt_min_priority : " << rt_min_priority << endl;



if(argc > 1)
{
    sscanf(argv[1], "%d", &dev);
    printf("using %s\n", argv[1]);
    sscanf(argv[2], "%s", argumentValue);
    //strncpy(argumentValue , argv[2], s)
    printf("Transformation = %s\n", argumentValue);
}
else if(argc == 1)
    printf("using default\n");

else
{
    printf("usage: capture [dev]\n");
    exit(-1);
}

// namedWindow( timg_window_name, CV_WINDOW_AUTOSIZE );
// // Create a Trackbar for user to enter threshold
// createTrackbar( "Min Threshold:", timg_window_name, &lowThreshold, max_lo

// capture = (CvCapture *)cvCreateCameraCapture(dev);
// cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, HRES);
// cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, VRES);

rc = pthread_attr_init(&rt_sched_attr[0]);
rc = pthread_attr_setinheritsched(&rt_sched_attr[0], PTHREAD_EXPLICIT_SCHED)
rc = pthread_attr_setschedpolicy(&rt_sched_attr[0] , SCHED_FIFO);
```

```
rt_param[0].sched_priority = rt_max_priority - 1;
pthread_attr_setschedparam(&rt_sched_attr[0] , &rt_param[0]);


pthread_t capThread;
pthread_create(&capThread, NULL , captureThread, (void*)argumentValue);


rc = pthread_attr_init(&rt_sched_attr[1]);
rc = pthread_attr_setinheritsched(&rt_sched_attr[1], PTHREAD_EXPLICIT_SCHED)
rc = pthread_attr_setschedpolicy(&rt_sched_attr[1] , SCHED_FIFO);

rt_param[1].sched_priority = rt_max_priority - 1;
pthread_attr_setschedparam(&rt_sched_attr[1] , &rt_param[1]);
pthread_t logThread;
pthread_create(&logThread, NULL , loggingThread, NULL);


pthread_join(capThread, NULL);
pthread_join(logThread, NULL);

while(1)
{
    if(threadExit)
    {
        break;
    }
}



// while(1)
// {
//     pthread_mutex_lock(&mtx);
//     auto startTime = std::chrono::system_clock::now();
//     frame=cvQueryFrame(capture);
//     if(!frame) break;


//     CannyThreshold(0, 0);
//     auto endTime = std::chrono::system_clock::now();
//     elapsed_seconds = endTime - startTime;
//     syncVariable = true;
//     pthread_mutex_unlock(&mtx);
//     //cout << "Frame size = " << sizeof(frame) << endl;
```

```
    //      //cout << "End - Start " << elapsed_seconds.count() << endl;
    //      char q = cvWaitKey(33);
    //      if( q == 'q' )
    //      {
    //          printf("got quit\n");
    //          break;
    //      }
    // }

    // cvReleaseCapture(&capture);

    cvDestroyWindow(timg_window_name);
    syslog(LOG_NOTICE, "\nThread Exit\n");


};



/*Transformations*/

void simpleHoughElliptical(void)
{
    Mat gray;
    vector<Vec3f> circles;
    Mat mat_frame = cv::cvarrToMat(frame);
    cvtColor(mat_frame, gray, CV_BGR2GRAY);
    GaussianBlur(gray, gray, Size(9,9), 2, 2);

    HoughCircles(gray, circles, CV_HOUGH_GRADIENT, 1, gray.rows/8, 100, 50, 0, 0

    printf("circles.size = %d\n", circles.size());

    for( size_t i = 0; i < circles.size(); i++ )
    {
        Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
        int radius = cvRound(circles[i][2]);
        // circle center
        circle( mat_frame, center, 3, Scalar(0,255,0), -1, 8, 0 );
        // circle outline
        circle( mat_frame, center, radius, Scalar(0,0,255), 3, 8, 0 );
    }


    if(!frame)
    {
```

```
        return;
    }

    //cvShowImage(timg_window_name, frame);
    imshow("Capture Example", mat_frame);

}

void simpleHough(void)
{
    vector<Vec4i> lines;
    Mat canny_frame;
    Mat mat_frame = cv::cvarrToMat(frame);
    Canny(mat_frame, canny_frame, 50, 200, 3);

    //cvtColor(canny_frame, cdst, CV_GRAY2BGR);
    //cvtColor(mat_frame, gray, CV_BGR2GRAY);

    HoughLinesP(canny_frame, lines, 1, CV_PI/180, 50, 50, 10);

    for( size_t i = 0; i < lines.size(); i++ )
    {
        Vec4i l = lines[i];
        line(mat_frame, Point(l[0], l[1]), Point(l[2], l[3]), Scalar(0,0,255), 3
    }


    if(!frame)
    {
        return;
    }
    imshow("Capture Example", mat_frame);
    //cvShowImage(timg_window_name, frame);
}
```