```
#include "funtions.h"


//观察的位置
cv::Point vP;
string wName = "鼠标左键点击选择像素，选择后按任意键开始处理";
int sub_threshold = 0;
Mat bgMat;
Mat subMat;
Mat bny_subMat;

bool useCamera = USE_CAMERA;
string videoPath = VIDEO_PATH;

void threshold_track(int, void *)//这里就是定义的一个回调函数，里面是canny相关的操作
{

threshold(subMat, bny_subMat,sub_threshold , 255, CV_THRESH_BINARY);
imshow("Result", bny_subMat);
}


//该demo验证并演示，视频中的像素灰度值变换是否呈高斯分布
int verifyGaussian()
{
//----------------------读取视频文件-------------------------
VideoCapture capVideo = createInput(useCamera, videoPath);

//如果视频打开失败
if (!capVideo.isOpened()) {
std::cout << "Unable to open video!" << std::endl;
return -1;
}

int cnt = 0;
int bin_width = 3;
int bin_heght = 100;
float histgram[256] = {0};

cv::Mat histMat;

while (1) {

Mat frame;
Mat grayMat;
capVideo >> frame;

if (frame.empty()) {
std::cout << "Unable to read frame!" << std::endl;
return -1;
}

//第一帧选取像素
if (cnt == 0) {
Mat selectMat;
frame.copyTo(selectMat);
namedWindow(wName);
imshow(wName, selectMat);
setMouseCallback(wName, on_mouse, &selectMat);
waitKey(0);
destroyAllWindows();
}

cvtColor(frame,grayMat,COLOR_BGR2GRAY);

//获得像素灰度值
int index = grayMat.at<uchar>(vP.y,vP.x);
//直方图相应的bin加1
histgram[index]++;

//绘制直方图
drawHist(histMat, histgram, bin_width,bin_heght);

drawMarker(frame, vP, Scalar(255, 255, 255));
imshow("frame",frame);
imshow("histMat",histMat);

//显示图片，延时30ms，必须要加waitKey()，否则无法显示图像
//等待键盘相应，按下ESC键退出
if (waitKey(30) == 27) {
destroyAllWindows();
```

```
break;
}
cnt++;
}

return 0;
}


int bgSub_demo()
{
//----------------------读取视频文件--------------------------
VideoCapture capVideo = createInput(useCamera, videoPath);

//如果视频打开失败
if (!capVideo.isOpened()) {
std::cout << "Unable to open video!" << std::endl;
return -1;
}

//计数器
int cnt = 0;
Mat frame;
while (1) {

capVideo >> frame;
cvtColor(frame,frame,COLOR_BGR2GRAY);

if (frame.empty()) {
std::cout << "Unable to read frame!" << std::endl;
return -1;
}

if (cnt== 0) {
//第一帧，获得背景图像
frame.copyTo(bgMat);
}
else {
//第二帧开始背景差分
//背景图像和当前图像相减
absdiff(frame, bgMat, subMat);
//差分结果二值化
namedWindow("Result", WINDOW_AUTOSIZE);
//滑动条创建
cv::createTrackbar("threshold", "Result", &sub_threshold, 255, threshold_track);
threshold_track(0,0);

imshow("frame",frame);
}

//显示图片，延时30ms，必须要加waitKey()，否则无法显示图像
//等待键盘相应，按下ESC键退出
if (waitKey(30) == 27) {
destroyAllWindows();
break;
}

cnt++;
}

return 0;
}

int bgSubGaussian_demo()
{
//----------------------读取视频文件--------------------------
//----------------------读取视频文件--------------------------
VideoCapture capVideo = createInput(useCamera, videoPath);

//如果视频打开失败
if (!capVideo.isOpened()) {
std::cout << "Unable to open video!" << std::endl;
return -1;
}

//如果视频打开失败
if (!capVideo.isOpened()) {
std::cout << "Unable to open video!" << std::endl;
return -1;
}
```

```
//用来计算背景模型的图像
std::vector<cv::Mat> srcMats;


int nBg = FRAME_NUMBER; //用来建立背景模型的数量
float wVar = VAR_WEIGHT; //方差权重

int cnt = 0;
bool calcModel= true;
cv::Mat frame;
cv::Mat meanMat;
cv::Mat varMat;
cv::Mat dstMat;

while (true)
{
capVideo >> frame;
cvtColor(frame, frame, COLOR_BGR2GRAY);

if (frame.empty()) {
std::cout << "Unable to read frame!" << std::endl;
return -1;
}

//前面的nBg帧，计算背景
if (cnt <= nBg) {

srcMats.push_back(frame);

if (cnt == 0) {
std::cout << "--- reading frame --- " << std::endl;
}
else {
std::cout << "-";
if (cnt % 50 == 0)std::cout << std::endl;
}
}
else {
if (calcModel) {
std::cout << std::endl << "calculating background models" << std::endl;
//计算模型
meanMat.create(frame.size(), CV_8UC1);
varMat.create(frame.size(), CV_32FC1);
//调用计算模型函数
calcGaussianBackground(srcMats, meanMat, varMat);
}
calcModel = false;

//背景差分
dstMat.create(frame.size(), CV_8UC1);
//利用均值mat和方差mat，计算差分
gaussianThreshold(frame, meanMat, varMat, wVar, dstMat);
imshow("result", dstMat);
imshow("frame", frame);

}

//显示图片，延时30ms，必须要加waitKey()，否则无法显示图像
//等待键盘相应，按下ESC键退出
if (waitKey(30) == 27) {
destroyAllWindows();
break;
}
cnt++;
}

return 0;
}

int calcGaussianBackground(std::vector<cv::Mat> srcMats, cv::Mat & meanMat, cv::Mat &varMat)
{

int rows = srcMats[0].rows;
int cols = srcMats[0].cols;


for (int h = 0; h < rows; h++)
{
for (int w = 0; w < cols; w++)
{
```

```
int sum=0;
float var=0;
//求均值
for (int i = 0; i < srcMats.size(); i++) {
sum += srcMats[i].at<uchar>(h, w);
}
meanMat.at<uchar>(h, w) =(uchar)(sum / srcMats.size());
//求方差
for (int i = 0; i < srcMats.size(); i++) {
var += (float)pow((srcMats[i].at<uchar>(h, w) - meanMat.at<uchar>(h, w)), 2);
}
varMat.at<float>(h, w) = var / srcMats.size();
}
}

return 0;
}

int gaussianThreshold(cv::Mat srcMat, cv::Mat meanMat, cv::Mat varMat, float weight, cv::Mat & dstMat)
{
int rows = srcMat.rows;
int cols = srcMat.cols;

for (int h = 0; h < rows; h++)
{
for (int w = 0; w < cols; w++)
{
int dif = abs(srcMat.at<uchar>(h, w) - meanMat.at<uchar>(h, w));
int th = (int)(weight*varMat.at<float>(h, w));

if (dif > th) {

dstMat.at<uchar>(h, w) = 255;
}
else {
dstMat.at<uchar>(h, w)=0;
}
}
}

return 0;
}

//调用opencv的背景差分函数方法
int opencvBgSubtrator()
{
//----------------------读取视频文件--------------------------
VideoCapture capVideo = createInput(useCamera, videoPath);

//如果视频打开失败
if (!capVideo.isOpened()) {
std::cout << "Unable to open video!" << std::endl;
return -1;
}

//如果视频打开失败
if (!capVideo.isOpened()) {
std::cout << "Unable to open video!" << std::endl;
return -1;
}

Mat inputFrame, frame, foregroundMask, foreground, background;

int method = BG_METHOD;
Ptr<BackgroundSubtractor> model;
if (method == 0) {
model = createBackgroundSubtractorKNN();
}
else if (method == 1) {
model = createBackgroundSubtractorMOG2();
}
else {
cout << "Can not create background model using provided method: '" << method << "'" << endl;
}

bool doUpdateModel = true;
bool doSmoothMask = false;

while (1) {
capVideo >> frame;
```

```
if (frame.empty()) {
std::cout << "Unable to read frame!" << std::endl;
return -1;
}

// pass the frame to background model
model->apply(frame, foregroundMask, doUpdateModel ? -1 : 0);

// show processed frame
imshow("image", frame);

// show foreground image and mask (with optional smoothing)
if (doSmoothMask)
{
GaussianBlur(foregroundMask, foregroundMask, Size(11, 11), 3.5, 3.5);
threshold(foregroundMask, foregroundMask, 10, 255, THRESH_BINARY);
}
if (foreground.empty())
foreground.create(frame.size(), frame.type());
foreground = Scalar::all(0);
frame.copyTo(foreground, foregroundMask);
imshow("foreground mask", foregroundMask);
imshow("foreground image", foreground);

// show background image
model->getBackgroundImage(background);
if (!background.empty())
imshow("mean background image", background);

// interact with user
const char key = (char)waitKey(30);
if (key == 27 || key == 'q') // ESC
{
cout << "Exit requested" << endl;
break;
}
else if (key == ' ')
{
doUpdateModel = !doUpdateModel;
cout << "Toggle background update: " << (doUpdateModel ? "ON" : "OFF") << endl;
}
else if (key == 's')
{
doSmoothMask = !doSmoothMask;
cout << "Toggle foreground mask smoothing: " << (doSmoothMask ? "ON" : "OFF") << endl;
}

}




return 0;
}


//鼠标响应函数
void on_mouse(int EVENT, int x, int y, int flags, void* userdata)
{

Mat hh;
hh = *(Mat*)userdata;
switch (EVENT)
{
case EVENT_LBUTTONDOWN:
{
vP.x = x;
vP.y = y;
drawMarker(hh,vP,Scalar(255,255,255));
//circle(hh, vP, 4, cvScalar(255, 255, 255), -1);
imshow(wName, hh);
return;
}
break;
}

}

//绘制直方图
int drawHist(cv::Mat & histMat, float * srcHist, int bin_width, int bin_heght)
{
```

```
histMat.create(bin_heght, 256 * bin_width, CV_8UC3);

histMat = Scalar(255, 255, 255);

float maxVal = *std::max_element(srcHist, srcHist + 256);

for (int i = 0; i < 256; i++) {
Rect binRect;
binRect.x = i*bin_width;
float height_i = (float)bin_heght*srcHist[i] / maxVal;
binRect.height = (int)height_i;
binRect.y = bin_heght - binRect.height;
binRect.width = bin_width;
rectangle(histMat, binRect, CV_RGB(255, 0, 0), -1);
}

return 0;
}

VideoCapture createInput(bool useCamera,std::string videoPath)
{
//选择输入
VideoCapture capVideo;
if (useCamera) {
capVideo.open(0);
}
else {
capVideo.open(videoPath);
}
return capVideo;
}
```