#define \_ITERATOR\_DEBUG\_LEVEL 0

#include <opencv2\highgui\highgui.hpp>

#include <iostream>

#include <opencv2/features2d.hpp>

#include <opencv2/features2d/features2d.hpp>

#include <opencv2\opencv.hpp>

#include "opencv2/core/core.hpp"

#include "opencv2/imgproc/imgproc.hpp"

#include "opencv2/video/background\_segm.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <stdio.h>

#include <stdlib.h>

#include <vector>

#include "opencv2/opencv.hpp"

#include "opencv2/imgproc.hpp"

#include <opencv2/highgui.hpp>

#include <opencv2/video.hpp>

using namespace std;

using namespace cv;

#include <fstream>

#include "opencv2/opencv.hpp"

#include "opencv2/highgui/highgui.hpp"

#include <time.h>

#include <sstream>

#include <sys/types.h>

#include <sys/stat.h>

/// Global variables

Mat src, dst;

//Mat fgMaskMOG; //fg mask generated by MOG method

//Mat fgMaskMOG2; //fg mask fg mask generated by MOG2 method

//Ptr<BackgroundSubtractor> pMOG;

//Ptr<BackgroundSubtractor> pMOG2;

int morph\_elem = 0;

int morph\_size = 0;

int morph\_operator = 0;

int const max\_operator = 4;

int const max\_elem = 2;

int const max\_kernel\_size = 21;

char\* window\_name = "Morphology Transformations Demo";

void Morphology\_Operations(int, void\*)

{

// Since MORPH\_X : 2,3,4,5 and 6

int operation = morph\_operator + 2;

Mat element = getStructuringElement(morph\_elem, Size(2 \* morph\_size + 1, 2 \* morph\_size + 1), Point(morph\_size, morph\_size));

/// Apply the specified morphology operation

morphologyEx(src, dst, operation, element);

//imshow(window\_name, dst);

}

void Morphology\_OperationsVertical(int, void\*)

{

// Since MORPH\_X : 2,3,4,5 and 6

int operation = morph\_operator + 2;

Mat element = getStructuringElement(morph\_elem, Size(1, 2 \* morph\_size + 1), Point(0, morph\_size));

/// Apply the specified morphology operation

morphologyEx(src, dst, operation, element);

//imshow(window\_name, dst);

}

void Morphology\_OperationsHorizontal(int, void\*)

{

// Since MORPH\_X : 2,3,4,5 and 6

int operation = morph\_operator + 2;

Mat element = getStructuringElement(morph\_elem, Size(2 \* morph\_size + 1, 1), Point(morph\_size, 0));

/// Apply the specified morphology operation

morphologyEx(src, dst, operation, element);

//imshow(window\_name, dst);

}

// Check if there is motion in the result matrix

// count the number of changes and return.

inline int detectMotion(const Mat & motion, Mat & result, Mat & result\_cropped,

int x\_start, int x\_stop, int y\_start, int y\_stop,

int max\_deviation,

Scalar & color)

{

// calculate the standard deviation

Scalar mean, stddev;

meanStdDev(motion, mean, stddev);

// if not to much changes then the motion is real (neglect agressive snow, temporary sunlight)

if (stddev[0] < max\_deviation)

{

int number\_of\_changes = 0;

int min\_x = motion.cols, max\_x = 0;

int min\_y = motion.rows, max\_y = 0;

// loop over image and detect changes

for (int j = y\_start; j < y\_stop; j += 2) { // height

for (int i = x\_start; i < x\_stop; i += 2) { // width

// check if at pixel (j,i) intensity is equal to 255

// this means that the pixel is different in the sequence

// of images (prev\_frame, current\_frame, next\_frame)

if (motion.at<uchar>(j,i) == 255)

{

number\_of\_changes++;

if (min\_x>i) min\_x = i;

if (max\_x<i) max\_x = i;

if (min\_y>j) min\_y = j;

if (max\_y<j) max\_y = j;

}

}

}

if (number\_of\_changes) {

//check if not out of bounds

if (min\_x - 10 > 0) min\_x -= 10;

if (min\_y - 10 > 0) min\_y -= 10;

if (max\_x + 10 < result.cols - 1) max\_x += 10;

if (max\_y + 10 < result.rows - 1) max\_y += 10;

// draw rectangle round the changed pixel

Point x(min\_x, min\_y);

Point y(max\_x, max\_y);

Rect rect(x, y);

Mat cropped = result(rect);

cropped.copyTo(result\_cropped);

rectangle(result, rect, color, 1);

}

return number\_of\_changes;

}

return 0;

}

int main(int argc, char \* const argv[])

{

const int DELAY = 30; // in mseconds, take a picture every 1/2 second

// Set up camera

//CvCapture \* camera = cvCaptureFromCAM(CV\_CAP\_ANY);

//cvSetCaptureProperty(camera, CV\_CAP\_PROP\_FRAME\_WIDTH, 1280); // width of viewport of camera

//cvSetCaptureProperty(camera, CV\_CAP\_PROP\_FRAME\_HEIGHT, 720); // height of ...

VideoCapture camera("PlaneSim.mp4"); //open the capture for video file

int totalframe = camera.get(CV\_CAP\_PROP\_FRAME\_COUNT); // get total number of frames in video

Mat result, result\_cropped, current\_frame, next\_frame;

camera >> result;

totalframe--;

Mat prev\_frame = result;

src = result;

camera >> current\_frame;

totalframe--;

camera >> next\_frame;

totalframe--;

cvtColor(current\_frame, current\_frame, CV\_RGB2GRAY);

cvtColor(prev\_frame, prev\_frame, CV\_RGB2GRAY);

cvtColor(next\_frame, next\_frame, CV\_RGB2GRAY);

// d1 and d2 for calculating the differences

// result, the result of and operation, calculated on d1 and d2

// number\_of\_changes, the amount of changes in the result matrix.

// color, the color for drawing the rectangle when something has changed.

Mat d1, d2, motion;

int number\_of\_changes, number\_of\_sequence = 0;

Scalar mean\_, color(0, 255, 255); // yellow

// Detect motion in window

int x\_start = 0, x\_stop = current\_frame.cols;

int y\_start = 0, y\_stop = current\_frame.rows;

// If more than 'there\_is\_motion' pixels are changed, we say there is motion

// and store an image on disk

int there\_is\_motion = 5;

// Maximum deviation of the image, the higher the value, the more motion is allowed

int max\_deviation = 20;

// Erode kernel

Mat kernel\_ero = getStructuringElement(MORPH\_RECT, Size(2, 2));

// All settings have been set, now go in endless loop and

// take as many pictures you want..

/\*

namedWindow("Control", CV\_WINDOW\_AUTOSIZE); //create a window called "Control"

int iLowH = 0;

int iHighH = 83;

int iLowS = 0;

int iHighS = 255;

int iLowV = 0;

int iHighV = 255;

//Create trackbars in "Control" window

cvCreateTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)

cvCreateTrackbar("HighH", "Control", &iHighH, 179);

cvCreateTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 - 255)

cvCreateTrackbar("HighS", "Control", &iHighS, 255);

cvCreateTrackbar("LowV", "Control", &iLowV, 255); //Value (0 - 255)

cvCreateTrackbar("HighV", "Control", &iHighV, 255);

\*/

//pMOG = new BackgroundSubtractorMOG(); //MOG approach

//pMOG2 = new BackgroundSubtractorMOG2(); //MOG2 approach

// Setup SimpleBlobDetector parameters.

SimpleBlobDetector::Params params;

// Change thresholds

params.minThreshold = 120;

params.maxThreshold = 240;

/\*

//Filter by Color

params.filterByColor = true;

params.blobColor = 200; \*/

// Filter by Area.

params.filterByArea = true;

params.minArea = 700;

// Filter by Circularity

params.filterByCircularity = false;

params.minCircularity = 0.1;

// Filter by Convexity

params.filterByConvexity = false;

params.minConvexity = 0.87;

// Filter by Inertia

params.filterByInertia = false;

params.minInertiaRatio = 0.01;

// Storage for blobs

vector<KeyPoint> keypoints;

/// Create window

namedWindow(window\_name, CV\_WINDOW\_AUTOSIZE);

/// Create Trackbar to select Morphology operation

createTrackbar("Operator:\n 0: Opening - 1: Closing \n 2: Gradient - 3: Top Hat \n 4: Black Hat", window\_name, &morph\_operator, max\_operator, Morphology\_Operations);

/// Create Trackbar to select kernel type

createTrackbar("Element:\n 0: Rect - 1: Cross - 2: Ellipse", window\_name,

&morph\_elem, max\_elem,

Morphology\_Operations);

/// Create Trackbar to choose kernel size

createTrackbar("Kernel size:\n 2n +1", window\_name,

&morph\_size, max\_kernel\_size,

Morphology\_Operations);

while (waitKey(30) != 27 && totalframe>0)

//wait 30 milliseconds and check for esc key

{

// Take a new image

prev\_frame = current\_frame;

current\_frame = next\_frame;

camera >> next\_frame;

result = next\_frame;

src = next\_frame;

Mat extra = src;

cvtColor(next\_frame, next\_frame, CV\_RGB2GRAY);

// Calc differences between the images and do AND-operation

// threshold image, low differences are ignored (ex. contrast change due to sunlight)

absdiff(prev\_frame, next\_frame, d1);

absdiff(next\_frame, current\_frame, d2);

bitwise\_and(d1, d2, motion);

threshold(motion, motion, 55, 255, CV\_THRESH\_BINARY);

erode(motion, motion, kernel\_ero);

//Do edge detection stuff

Mat edges;

Canny(current\_frame, edges, 180, 180, 3);

bitwise\_or(motion, edges, motion);

//MOG(extra, fgMaskMOG2);

//pMOG->operator()(extra, fgMaskMOG2);

//pMOG2->operator()(extra, fgMaskMOG2);

//imshow("FG Mask MOG", fgMaskMOG);

//imshow("FG Mask MOG 2", fgMaskMOG2);

//imshow("Edges", edges);

//Morphology\_OperationsHorizontal(0, 0);

//Morphology\_OperationsVertical(0, 0);

#if CV\_MAJOR\_VERSION < 3 // If you are using OpenCV 2

// Set up detector with params

SimpleBlobDetector detector(params);

// Detect blobs

detector.detect(src, keypoints);

#else

// Set up detector with params

Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);

// Detect blobs

detector->detect(im, keypoints);

#endif

// Draw detected blobs as red circles.

// DrawMatchesFlags::DRAW\_RICH\_KEYPOINTS flag ensures

// the size of the circle corresponds to the size of blob

Mat im\_with\_keypoints;

drawKeypoints(src, keypoints, im\_with\_keypoints, Scalar(0, 0, 255), DrawMatchesFlags::DRAW\_RICH\_KEYPOINTS);

// Show blobs

//imshow("blobs", im\_with\_keypoints);

for (std::vector<cv::KeyPoint>::iterator blobIterator = keypoints.begin(); blobIterator != keypoints.end(); blobIterator++) {

std::cout << "size of blob is: " << blobIterator->size << std::endl;

std::cout << "point is at: " << blobIterator->pt.x << " " << blobIterator->pt.y << std::endl;

}

/\*

//Lets look at color detection

Mat imgHSV;

cvtColor(extraNext\_frame, imgHSV, CV\_BGR2HSV); //Convert the captured frame from BGR to HSV

Mat imgThresholded;

inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH, iHighS, iHighV), imgThresholded); //Threshold the image

//morphological opening (remove small objects from the foreground)

erode(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

//morphological closing (fill small holes in the foreground)

dilate(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

erode(imgThresholded, imgThresholded, getStructuringElement(MORPH\_ELLIPSE, Size(5, 5)));

imshow("Thresholded Image", imgThresholded); //show the thresholded image

Mat hsvAndEdges;

bitwise\_or(imgThresholded, motion, hsvAndEdges);

imshow("hsvAndEdges", hsvAndEdges); //show the original image

\*/

//imshow("edges", edges);

//imshow("both", motion); //show image on window named Camera

//number\_of\_changes = detectMotion(motion, result, result\_cropped, x\_start, x\_stop, y\_start, y\_stop, max\_deviation, color);

// If a lot of changes happened, we assume something changed.

if (1/\*number\_of\_changes\*/ >= there\_is\_motion)

{

number\_of\_sequence++;

}

else

{

number\_of\_sequence = 0;

// Delay, wait given time (in ms)

cvWaitKey(DELAY);

}

totalframe--;

}

return 0;

}

/\*

void MyLine(Mat img, Point start, Point end)

{

int thickness = 3;

int lineType = 8;

line(img,

start,

end,

CV\_RGB(250,250,250),

thickness,

lineType);

}

int main(void)

{

Mat frame;

Mat edges;

VideoCapture cap("PlaneSim.mp4"); //open the capture for video file

int totalframe = cap.get(CV\_CAP\_PROP\_FRAME\_COUNT); // get total number of frames in video

namedWindow("Canvas", CV\_WINDOW\_AUTOSIZE);

cv::moveWindow("Canvas", 10, 20);

while (waitKey(30) != 27 && totalframe>0)

//wait 30 milliseconds and check for esc key

{

cap >> edges; //save captured image to frame variable

//cvtColor(frame, edges, CV\_BGR2GRAY);

//GaussianBlur(edges, edges, Size(1, 1), 0.5, 0.5, BORDER\_REFLECT);

//blur(edges, edges, Size(7, 7));

Canny(edges, edges, 180, 180, 3);

cv::Size s = edges.size();

int rows = s.height; cout << rows << endl;

int cols = s.width; cout << " " << cols << endl;

//cout << Point(rows / 2 + 40, cols / 2 + 40) << endl;

MyLine(edges, Point(cols/2 + 30, rows/2), Point(cols/2 - 30, rows/2 ));

MyLine(edges, Point(cols / 2, rows / 2 + 30), Point(cols / 2, rows / 2 - 30));

imshow("Canvas", edges); //show image on window named Camera

totalframe--;

}

}

\*/

/\*

#include <opencv2\opencv.hpp>

using namespace cv;

int main(int, char\*\*)

{

VideoCapture cap(0); // open the default camera

if (!cap.isOpened()) // check if we succeeded

return -1;

Mat edges;

namedWindow("edges", 1);

for (;;)

{

Mat frame;

cap >> frame; // get a new frame from camera

cvtColor(frame, edges, CV\_BGR2GRAY);

GaussianBlur(edges, edges, Size(7, 7), 1.5, 1.5);

Canny(edges, edges, 0, 30, 3);

imshow("edges", edges);

if (waitKey(30) >= 0) break;

}

// the camera will be deinitialized automatically in VideoCapture destructor

return 0;

}

\*/