Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа программной инженерии

Отчет

Лабораторная работа № 6

Детекция движущихся объектов на видео с использованием библиотеки OpenCV 3.2

Выполнил

студент гр. B3530904/80021 И. С. Томилин

Руководитель С. А. Молодяков

«___»_____201__г.

Санкт-Петербург

2020

Текст программы

player.h

```
#ifndef __PLAYER_H__
#define __PLAYER_H__


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

#include <string>


int create_window( std::string window_name,
                   int         h_size,
                   int         v_size,
                   int         h_pos,
                   int         v_pos
                 );


#endif // __PLAYER_H__
```

player.cpp

```
#include "player.h"


int create_window( std::string window_name,
                   int         h_size,
                   int         v_size,
                   int         h_pos,
                   int         v_pos
                 )
{
    // Create the window
    cv::namedWindow( window_name, cv::WINDOW_NORMAL );

    //cv::resizeWindow( window_name, 600, 480 );
    cv::resizeWindow( window_name, h_size, v_size );

    //cv::moveWindow( window_name, 420, 210 );
    cv::moveWindow( window_name, h_pos, v_pos );

    return 0;
}
```

blob.h

```
#ifndef __MY_BLOB__
#define __MY_BLOB__

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


class Blob
{

public:
    std::vector< cv::Point > contour;


    cv::Rect boundingRect;


    cv::Point centerPosition;


    double dblDiagonalSize;


    double dblAspectRatio;


    Blob( std::vector< cv::Point > _contour );

};

#endif // __MY_BLOB__
```

blob.cpp

```
#include "blob.h"


Blob::Blob( std::vector< cv::Point > _contour )
{
    contour = _contour;

    boundingRect = cv::boundingRect( contour );

    centerPosition.x = ( boundingRect.x +
                         boundingRect.x +
                         boundingRect.width ) / 2;

    centerPosition.y = ( boundingRect.y +
                         boundingRect.y +
                         boundingRect.height ) / 2;

    dblDiagonalSize = sqrt( pow( boundingRect.width, 2 ) +
                            pow( boundingRect.height, 2 ) );

    dblAspectRatio = (float)boundingRect.width / (float)boundingRect.height;

}
```

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

#include<iostream>

#include "blob.h"
#include "player.h"

int third         = 0,
    sign          = 0,
    current_frame = 0;

cv::VideoCapture capture_video;

// Обработчик уровня бинаризации
void trackbar_handler_scroll( int current_position,
                      void (*ptr) )
{
}


void trackbar_handler_thresh( int current_position,
                      void (*ptr) )
{
    capture_video.set( cv::CAP_PROP_POS_FRAMES, current_position );
}


int main( int argc, char **argv )
{
    cv::Mat imgFrame1;
    cv::Mat imgFrame2;

    capture_video.open( argv[1] );

    if ( !capture_video.isOpened() )
    {
        std::cout << "Could not open the file." << std::endl << std::endl;
        return 1;
    }

    capture_video.read( imgFrame1 );
    capture_video.read( imgFrame2 );

    create_window( "binary",    480, 320, 50,  50 );
    create_window( "detection", 480, 320, 550, 50 );
    create_window( "original",  480, 320, 550, 450 );

    trackbar_handler_scroll( third, &capture_video );
    trackbar_handler_thresh( sign, &capture_video );


    cv::createTrackbar( "br",
                        "binary",
                        &third,
                        255,
                        trackbar_handler_scroll
                      );
    cv::createTrackbar( "pos",
                        "original",
                        &sign,
                        capture_video.get( cv::CAP_PROP_FRAME_COUNT ),
                        trackbar_handler_thresh
                      );

    cv::waitKey();
```

```
    while ( !imgFrame2.empty() && cv::waitKey( 33 ) != 27 )
    {
        std::vector< Blob > blobs;
        std::vector< std::vector< cv::Point > > contours;

        cv::Mat imgFrame1Copy = imgFrame1.clone();
        cv::Mat imgFrame2Copy = imgFrame2.clone();

        cv::Mat imgDifference;
        cv::Mat imgThresh;

        cv::cvtColor( imgFrame1Copy, imgFrame1Copy, CV_BGR2GRAY );
        cv::cvtColor( imgFrame2Copy, imgFrame2Copy, CV_BGR2GRAY );

        cv::imshow( "original", imgFrame2 );

        cv::absdiff( imgFrame1Copy, imgFrame2Copy, imgDifference );

        cv::threshold( imgDifference,
                       imgThresh,
                       third,
                       255,
                       CV_THRESH_BINARY );

        cv::Mat imgThreshCopy = imgThresh.clone();

        cv::imshow( "binary", imgThreshCopy );

        cv::findContours( imgThreshCopy,
                          contours,
                          cv::RETR_EXTERNAL,
                          cv::CHAIN_APPROX_SIMPLE
                        );

        for( auto  &contours : contours )
        {
            Blob possibleBlob( contours );
// Параметры объекта, для его захвата, нужно подгонять значения
          if ( possibleBlob.boundingRect.area() >  10 &&
               possibleBlob.dblAspectRatio      >= 0.2 &&
               possibleBlob.dblAspectRatio      <= 1.2 &&
               possibleBlob.boundingRect.width  >  15 &&
               possibleBlob.boundingRect.height >  20 &&
               possibleBlob.dblDiagonalSize     >  30.0)
          {
               blobs.push_back( possibleBlob );
          }
        }

        imgFrame2Copy = imgFrame2.clone();

        for ( auto &blob : blobs )
            cv::rectangle( imgFrame2Copy,
                           blob.boundingRect,
                           cv::Scalar( 0, 255, 0 ),
                           2
                         );

        cv::imshow( "detection", imgFrame2Copy );
        imgFrame1 = imgFrame2.clone();
        current_frame = capture_video.get( cv::CAP_PROP_POS_FRAMES );
        cv::setTrackbarPos( "pos",
                            "original",
                            current_frame );

        capture_video.read( imgFrame2 );
    }
    return 0;
}
```

Результаты

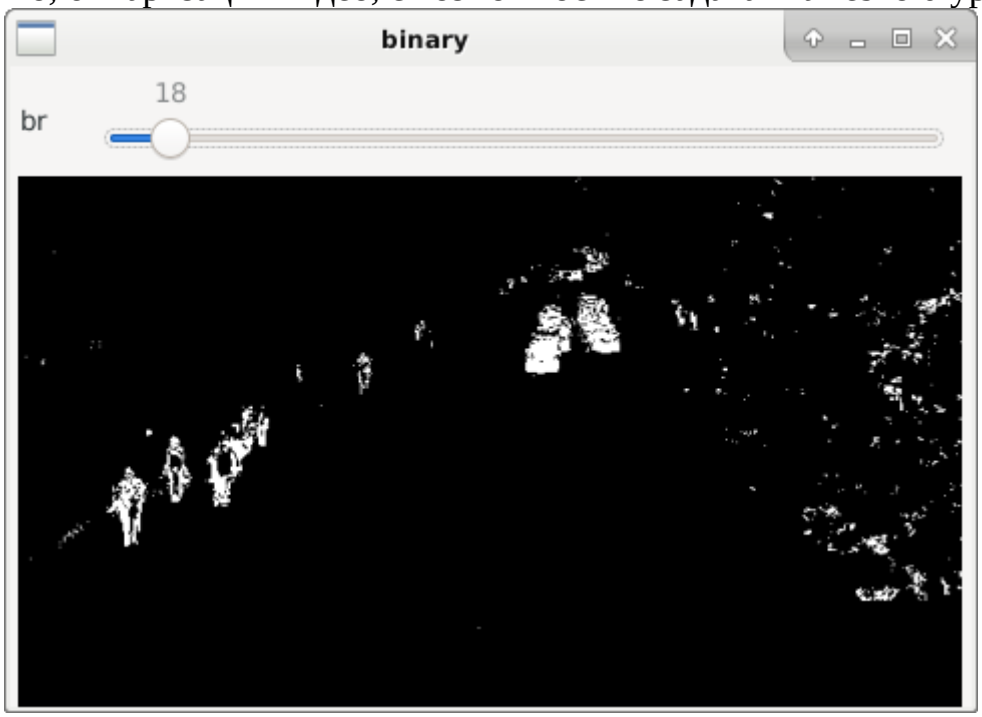
Например нам нужно детектить движущиеся объекты.

Берем два кадра, после сравниваем их, тем самым определяем двигается ли объект. На **статичные** объекты не обращаем внимание, они нам не интересны в данной реализации алгоритма.

Выполняем бинаризацию кадров, выделяются контура движущихся объектов, забрасываем набор этих контуров в контейнер и после по заданным параметрам производится выборка (по размерам) нужных объектов из набора контуров. Это выполняется для того, чтобы максимально отсеять мусорные частицы.

Для удобства создается 3 окна, чтобы увидеть весь процесс.

Первое окно, бинаризация видео, с возможностью задавать threshold уровень.



Второе окно, детекция **движущихся** объектов на кадрах.

Третье окно, оригинальное видео с возможностью перемотки.