AutoVision X Hackathon

Project: Home Part Assist(HPA)

Team name: Techolics

Member 1: Pratyush Kurhe

Member 2: Ankit Saha

Approach:

Since it is familiar for the team members to code the Computer Vision project in python, so we first tried to execute the company expectations using python and then we converted it into C++. (Both the source codes of python and C++ are attached. Also the o/p video is attached.)

Explanation of the source code:

1)Importing libraries:

#include <iostream>

#include <opencv2/opencv.hpp>

#include <opencv2/imgproc.hpp>

#include <opencv2/highgui.hpp>

#include <opencv2/video.hpp>

using namespace std;

using namespace cv;

First we imported all the required libraried to execute the code.

2) Taking i/p from command line argument:

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

In main function we have taken argc and argv as parameters. ‘argc’ stores total number of arguments where as ‘argv’ will store the arguments viz. video file and base address.

3)Capturing the Video:

VideoCapture cap(argv[1]);

Using VideoCapture class the code reads the video file which is passed as second(first being name of the code file) argument in terminal and stores it in the object ‘cap’.

4)No. of frames per second is stored in the pointer variable ‘fps’.

double\* fps=new double;

\*fps = cap.get(CAP\_PROP\_FPS);

5)We made the o/p video file which will store the desired results. It is made using VideoWriter class and ‘writer’ object. I/p to the VideoWriter class are

a)file name,

b)codec to be used. ‘fourcc’ stands for four character code so we i/p four characters which stands for Motion JPEG, a video compression format.

c)no. of frames per second

d) Size of frames in the video. Width and Height repectively.

VideoWriter writer("Hackathon\_output2.mp4", VideoWriter::fourcc('M', 'J', 'P', 'G'), \*fps, Size(cap.get(CAP\_PROP\_FRAME\_WIDTH), cap.get(CAP\_PROP\_FRAME\_HEIGHT)));

6) We then made a variable to store total no. of frames we worked upon and total no. of frames in the video using ‘get’ method of VideoCapture class. This can be used to terminate the while loop when the video ends.

int \*frame\_number = new int(0);

double frame\_count = cap.get(CAP\_PROP\_FRAME\_COUNT);

7)The frame with frame no. ‘frame\_number’ is then captured from the video and stored in image1 matrix pointer variable. Set() function will set the properties of the ‘cap’ object makes it to capture the frame with frame no. ‘frame\_number’.

Mat \*image1=new Mat;

Mat \*image2=new Mat;

cap.set(CAP\_PROP\_POS\_FRAMES, \*frame\_number);

cap >> \*image1;

8) While loop will run the ‘frame\_number’ reach the last frame no. , ‘endframe’ will store the next corresponding frame and will store it in matrix pointer variable ‘image2’.

 while (\*frame\_number <= frame\_count) {

        int endFrame = \*frame\_number + 1;

        cap.set(CAP\_PROP\_POS\_FRAMES, endFrame);

        cap >> \*image2;

9)If the ‘image2’ gets captured successfully, we created some matrix pointer variables for further computations.

if (!(\*image2).empty()) {

            Mat \*gotFrame=new Mat;

            Mat \*frame1=new Mat;

            Mat \*frame2=new Mat;

10) The main part of the dynamic object detection is to find the difference between the two frames, which can efficiently done on gray images, so we converted the colored images into gray images using cvtColor function. The parameters to the function are:

a) colored image,

b)variable to store gray images,

c)predefined conversion code.

Then to find difference between images we took absolute difference between the two gray images using ‘absdiff’ method taking 1st frame, 2nd frame and the resulting frame as parameters.

cvtColor(\*image1, \*frame1, COLOR\_BGR2GRAY);

cvtColor(\*image2, \*frame2, COLOR\_BGR2GRAY);

absdiff(\*frame1, \*frame2, \*gotFrame);

11)We used threshold function to separate regions of the image based on pixel intensity values. The parameters of the function are respectively:

a)Image on which the thresholding to be applied. Here we applied the thresholding to the image difference between two consecutive image frames of the i/p video.

b)variable to store the thresholded image.

c)lower threshold,

d)upper threshold,

e)Two thresholding flags, first flag will indicate that the thresholding operation will produce binary image. Second flag calculates optimal threshold value using image histogram.

Mat \*thresh=new Mat;

threshold(\*gotFrame, \*thresh, 0, 255, THRESH\_BINARY | THRESH\_OTSU);

12) After that we created the structuring element ‘kernel’ which can be used for morphological operations like dilation, erosion, closing , opening. Here we created it for dilation. Parameters are:

a)First argument specifies the shape of the structuring element i.e. rectangular shape,

b) Second argument specifies the size of the structuring element. Here 3x3.

Mat \*kernel=new Mat;

\*kernel = getStructuringElement(MORPH\_RECT, Size(3, 3));

13)We then performed dilation operation on the binary image ‘\*thresh’ using rectangular structuring element ‘kernel’. Parameters are:

a)binary image

b)variable in which dilated image to be stored.

c)structuring element for dilation operation.

d) default anchor point.

e)no. of times dilation operation to be applied. Here we used 2, since it gives the optimum results.

Mat \*dilate=new Mat;

dilate(\*thresh, \*dilate, kernel, Point(-1, -1), 2);

14)2D Vector is then created to store no. of contours.

It is stored in the ‘contours’ matrix using findContours method. Parameters are:

a)dilated image

b) vector in which the contours are stored.

c)contour retrieval mode. ‘RETR\_EXTERNAL’ retrieves only external contours.

d)contour approximation method. Here CHAIN\_APPROX\_SIMPLE compresses horizontal, vertical, and diagonal segments and leaves only their end points.

vector<vector<Point>> contours;

findContours((\*dilate).clone(), contours, RETR\_EXTERNAL, CHAIN\_APPROX\_SIMPLE);

15) Finally drawing contours on the image using drawContours method. Parameters are:

a) the image onto which the contours will be drawn.

b) vector containing the contours to be drawn.

c) Passing -1 draws all contours.

d) parameter specifies the color of the contours to be drawn.

e) parameter specifies the thickness of the contour lines to be drawn.

Mat \*contour\_img= new Mat;

\*contour\_img = (\*image1).clone();

drawContours(\*contour\_img, contours, -1, Scalar(0, 255, 0), 3);

16) Appending the desired contour image to the video file using ‘write’ method.

writer.write(\*contour\_img);

17)Count of ‘frame\_number’ is increased and since ‘image2’ is the next corresponding frame we copied that frame into ‘image1’.

\*frame\_number++;

\*image1 = (\*image2).clone();

18)As a part of efficient memory management we then freed all the pointer variables.

free(image2);

free(gotFrame);

free(frame1);

free(frame2);

free(kernel);

free(dilate);

free(thresh);

free(contour\_img);

}

19)if the next corresponding frame to ‘image1’ is not found then following o/p will be shown.

else {

        cout << "Image not found after frame number " << \*frame\_number <<endl;

        break;

        }

    }

20)Finally ‘writer’ and ‘cap’ objects are released and pointer variables ‘fps’ and ‘image1’ are deleted.

writer.release();

cap.release();

delete fps;

delete image1;

return 0;

}

After successfully executing the code we got output video in the present directory.