International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 12, December-2016

Spinning Detection

Harita Chocha[#1], Chaitanyakumar Patel[#2]
[#]Department of Computer Science and Engineering,
Institute of Technology,
Nirma University, Ahmedabad,
Gujarat - 382481,
India

Abstract: **Image processing and pattern recognition is a wide area. Detection of a particular pattern or feature from an image or video. In this paper spinning detection is done on images using OpenCV opencv_haartraining and opencv_traincascade. It creates an XML file that enables the machine to recognize a spinning image. We have designed an application in Visual Studio using C# that uses OpenCV to detect the spinning objects in images.**

I. INTRODUCTION

Detecting any pattern or feature visually is easy for humans as they have their own way of visualizing things biologically. For the machine to recognize patterns we need to train it and enable recognizing in it. To enable this ability of a machine or a system we use OpenCV (CV – Computer Vision) to train it. OpenCV is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. OpenCV has a modular structure, which means that the package includes several shared or static libraries. Some of the modules available are: [1]

- core
- imgproc
- video
- calib3d
- features2d
- objdetect
- highgui
- gpu
- some other helper modules, such as FLANN and Google test wrappers, Python bindings, and others

In the following paper we use the *objdetect* module of OpenCV to create a classifier that has the ability to detect spinning objects in an image. We have also used the functionality of *imgproc*to to apply various image processing techniques such resize, grayscale conversion in order to detect the desired result.

II. APPICATIONS

Spinning detection application is used to detect the motion in an image. This can be used in astronomy, weather forecast and weather prediction. Seeing various patterns of cloud we can predict whether a storm or strong winds are approaching. In astronomy, we can detect the movement of galaxies and stars. The basic use of such an application is to find moving objects such a car, a giant wheel etc. Using satellite images we can detect particular locations, objects, rocks etc. by applying image processing techniques.

III. CASCADE CLASSIFIER

We have two major stages in the process: training and detection. This section describes cascade classifier training.

There are two applications in OpenCV to train cascade classifier: opencv_haartraining and opencv_traincascade. opencv_traincascade is a newer version, written in C++ in accordance to OpenCV 2.x API. But the main difference between this two applications is that opencv_traincascade supports both Haar [Viola2001] and LBP [Liao2007] (Local Binary Patterns) features. LBP features are integer in contrast to Haar features, so both training and detection with LBP are several times faster then with Haar features. Regarding the LBP and Haar detection quality, it depends on training: the quality of training dataset and training parameters too.[2]
Also there are some auxilary utilities related to the training:[2]

- opencv_createsamples is used to prepare a training dataset of positive and test samples. opencv_createsamples produces dataset of positive samples in a format that is supported by both opencv_haartraining and opencv_traincascade applications. The output is a file with *.vec extension, it is a binary format which contains images.
- opencv_performance may be used to evaluate the quality of classifiers, but for training we have to use opencv_haartraining only. It takes a collection of marked up images, runs the classifier and reports the performance, i.e. number of found objects, number of missed objects, number of false alarms and other information.

IV.TRAINING CLASSIFIER

Training data preparation[2]
Training data set consists of two samples: Positive and Negative.

- Positive: These images contain the object to be detected.
- Negative: Except object image, anything can be present.

Negative Samples: This images are taken arbitrary and they may contain anything except for the object to be detected. They are stored in a special file (.txt) where each line in the file indicates the storage location relative to the directory is being stored. Also negative samples and sample images are also called the background samples and are used interchangeably. This file must be created manually. This images can be of a different size but they are required to be of a size larger than the training window size because this images are used to subsample negative image to training size.

*Positive Samples:*This images are created using the opencv_createsamples utility. They may be created from a single object or from a previously used marked images. Here you need a large set of positive samples before giving it to the utility because it only applies the necessary transformation to a single image. For e.g. in case of some rigid body structure like logo of particular brand you may need only one sample image but in case of faces you may need thousands or even more samples and also they should cover each and every race, emotion, beard style etc.

We can control the intensity and other factors using the command line arguments of opencv_createsamples utility as given below:

- -vec <vec_file_name> - It contains name of the output file containing positive samples.
- -img <image_file_name> - Source object image (e.g., a company logo,pen)
- -bg <background_file_name> - Background description file containing a list of images which are used as a background for randomly distorted versions of the object.
- -num <number_of_samples> -Number of positive samples to generate.
- -bgcolor <background_color> -Background color (currently grayscale images are assumed); the background color denotes the transparent color. Since there might be compression artifacts, the amount of color tolerance can be specified by -bgthresh. All pixels withing bgcolor-bgthresh and bgcolor+bgthresh range are interpreted as transparent.
- -bgthresh <background_color_threshold>
- -inv-If specified, colors will be inverted.
- -randinv-If specified, colors will be inverted randomly.
- -maxidev <max_intensity_deviation>-Maximal intensity deviation of pixels in foreground samples.
- maxxangle <max_x_rotation_angle>
- -maxyangle <max_y_rotation_angle>
- -maxzangle <max_z_rotation_angle>-Maximum rotation angles must be given in radians.
- -show -Useful debugging option. If specified, each sample will be shown. Pressing Esc will continue the samples creation process without.
- -w <sample_width> -Width (in pixels) of the output samples.
- -h <sample_height> -Height (in pixels) of the output samples.

An example of description file:
Directory structure: /image

```
        img1.jpg
        img2.jpg
info.dat
```

File info.dat: image/img1.jpg 1 140 100 45 45

```
        image/img1.jpg 2 100 200 50 50     50 30 25 25
```

Cascade Training [2]

The next step is the training of the cascade classifier. As said earlier opencv_traincascade or opencv_haarcascade can be use dbut here opencv_traincascade is described: Command line arguments of opencv_traincascade application grouped by purposes:

1. *Common arguments:*

- -data<cascade_dir_name> -Where the trained classifier should be stored.
- -vec<vec_file_name> -vec-file with positive samples (created by opencv_createsamples utility).
- -bg<background_file_name> -Background description file.
- -numPos<number_of_positive_samples>
- numNeg<number_of_negative_samples> -Number of positive/negative samples used in training for every classifier stage.
- -numStages<number_of_stages> - Number of cascade stages to be trained.
- precalcValBufSize<precalculated_vals_buffer_size_in_Mb> -Size of buffer for precalculated feature values (in Mb).
- precalcIdxBufSize<precalculated_idxs_buffer_size_in_Mb> -Size of buffer for precalculated feature indices (in Mb). The more memory you have the faster the training process.
- -baseFormatSave -This argument is actual in case of Haar-like features. If it is specified, the cascade will be saved in the old format.

2. *Cascade parameters:*

- -stageType <BOOST(default)> -Type of stages. Only boosted classifier are supported as a stage type at the moment.
- -featureType<{HAAR(default),LBP}> -Type of features: HAAR - Haar-like features, LBP - local binary patterns.
- -w <sampleWidth>
- -h <sampleHeight> - Size of training samples (in pixels). Must have exactly the same values as used during training samples creation (opencv_createsamples utility).

3. *Boosted classifer parameters:*

- -bt <{DAB, RAB, LB, GAB(default)}> -Type of boosted classifiers: DAB - Discrete AdaBoost, RAB - Real AdaBoost, LB - LogitBoost, GAB - Gentle AdaBoost.
- -minHitRate <min_hit_rate> -Minimal desired hit rate for each stage of the classifier. Overall hit rate may be estimated as (min_hit_rate^number_of_stages).
- -maxFalseAlarmRate <max_false_alarm_rate> - Maximal desired false alarm rate for each stage of the classifier. Overall false alarm rate may be estimated as (max_false_alarm_rate^number_of_stages).

- -weightTrimRate <weight_trim_rate> -Specifies whether trimming should be used and its weight. A decent choice is 0.95.
- -maxDepth <max_depth_of_weak_tree> -Maximal depth of a weak tree. A decent choice is 1, that is case of stumps.
- -maxWeakCount <max_weak_tree_count> -Maximal count of weak trees for every cascade stage. The boosted classifier (stage) will have so many weak trees (<=maxWeakCount), as needed to achieve the given -maxFalseAlarmRate.

4. *Haar-like feature parameters:*
- -mode <BASIC (default) | CORE | ALL> -Selects the type of Haar features set used in training. BASIC use only upright features, while ALL uses the full set of upright and 45 degree rotated feature set. See [Rainer2002] for more details.

5. *Local Binary Patterns parameters:*
   Local Binary Patterns don't have parameters.

V. DETECTION

We have two main functions of *objdetection*module for detection. They are:

- load[3]: load a .xml classifier file. It can be either a Haar or a LBP classifer

  C++: boolCascadeClassifier::load(const string& filename)

  Python: cv2.CascadeClassifier.load(filename) → retval

  Parameters: filename – Name of the file from which the classifier is loaded. The file may contain an old HAAR classifier trained by the haartraining application or a new cascade classifier trained by the traincascade application.[]

- detectMultiScale[4]: To perform the detection

  C++: void CascadeClassifier::detectMultiScale(const Mat& image, vector<Rect>& objects, double scaleFactor=1.1, intminNeighbors=3, int flags=0, Size minSize=Size(), Size maxSize=Size())
  Python: cv2.CascadeClassifier.detectMultiScale(image[, scaleFactor[, minNeighbors[, flags[, minSize[, maxSize]]]]]) → objects
  The parameters of detectMultiScale are:
- cascade – Haar classifier cascade (OpenCV 1.x API only). It can be loaded from XML or YAML file using Load(). When the cascade is not needed anymore, release it using cvReleaseHaarClassifierCascade(&cascade).
- image – Matrix of the type CV_8U containing an image where objects are detected.
- objects – Vector of rectangles where each rectangle contains the detected object.
- scaleFactor – Parameter specifying how much the image size is reduced at each image scale.
- minNeighbors – Parameter specifying how many neighbors each candidate rectangle should have to retain it.
- flags – Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
- minSize – Minimum possible object size. Objects smaller than that are ignored.
- maxSize – Maximum possible object size. Objects larger than that are ignored.

VI. CREATING APPLICATION

We have used Visual Studio to create an application that helps us in spinning detection in an image. We have used C# language and Emgu, a wrapper that allows to use the functionality of OpenCV to be used with C#.

Initialization:
Create four folders neg, pos, samples, cascade.
Gathering data:
229 images that had a spinning pattern were taken as positive images. 177 images that does not represent spin were taken as negative images.

Store negative images in neg folder, positive images in positive folder.

Creating the .txt file:
Using commands given below a .dat file for negative and positive images was created.

```
find ./neg -iname '*.ppm' >negatives.txt

find ./pos -iname '*.png' >positives.txt
```

Create Samples:
Using a pearl script we store training samples in samples folder.

```
perl createtrainsamples.pl positives.dat negatives.dat samples 250 "./opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1 -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 200 -h 200"
```

The next two commands use the data in samples folder to create a unified training data, that is used for training

```
find samples/ -name '*.vec' > samples.dat

./mergevec samples.dat samples.vec
```

NOTE: ./mergecevis a executable C++ file that can be created using the following command.

Copy the following files from your installed opencv folder and save them in cascadeTraining folder

cvboost.cpp, cvclassifier.h, cvcommon.cpp, _cvcommon.h, cvhaarclassifier.cpp, cvhaartraining.cpp, cvhaartraining.h, _cvhaartraining.h, cvsamples.cpp

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 5 Issue 12, December-2016

Open a terminal in the haartraining folder and type:

```
1. chmod +x mergevec.cpp
```

```
2. g++ `pkg-config --cflagsopencv` -o mergevec mergevec.cpp cvboost.cpp cvcommon.cpp cvsamples.cpp cvhaarclassifier.cpp cvhaartraining.cpp `pkg-config --libs opencv`
```

After all the *.vec files are created finally we need to start the train the cascade.

```
$ opencv_traincascade -data cascade -vecpos-samples.vec -bg neg-filepaths.txt -precalcValBufSize 2048 -precalcIdxBufSize 2048 -numPos 200 -numNeg 2000 -nstages 20 -minhitrate 0.999 -maxfalsealarm 0.5 -w 200 -h 200 -nonsym–baseFormatSave
```

After the command is completed we get an .xml file in the cascade folder named cascasde.xml that can be used with the detection functions to detect the spinning images.

VII. OBSERVATIONS AND SNAPSHOTS:

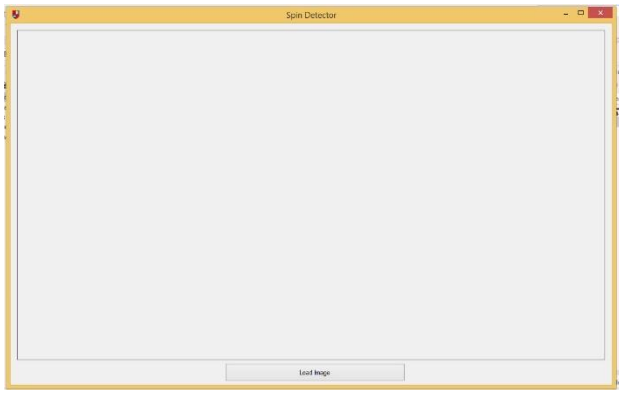

Figure 1. The Structure of the application

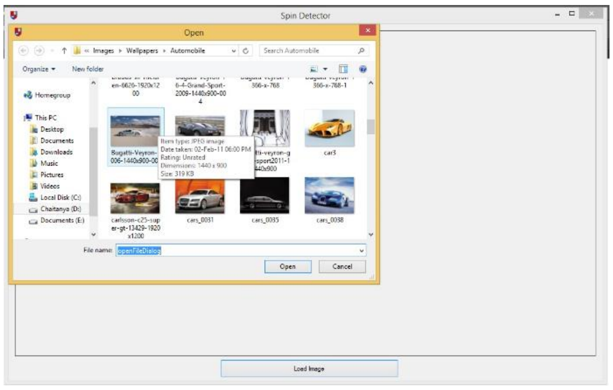

Figure 2. Loading an image in the application

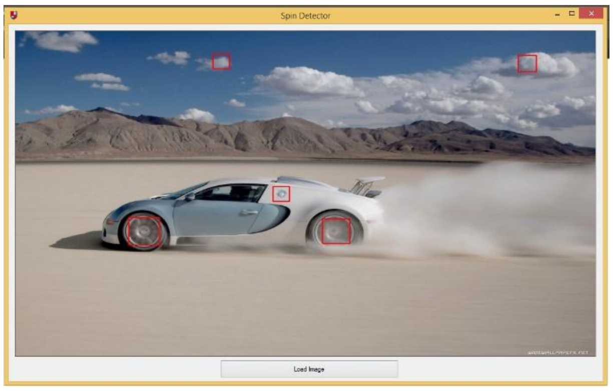

Figure 3. Output of an Image, red box shows spinning portions in image

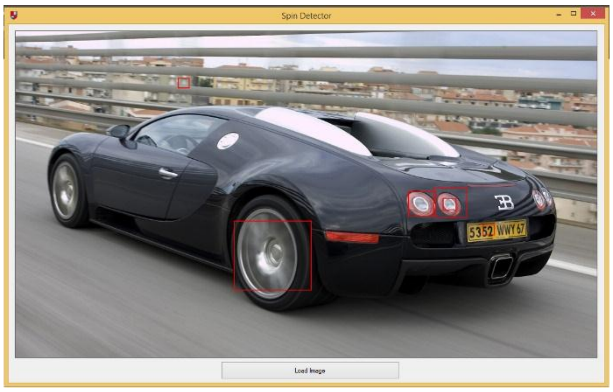

Figure 4. The red box indicates the spin portion in image, misses some portions

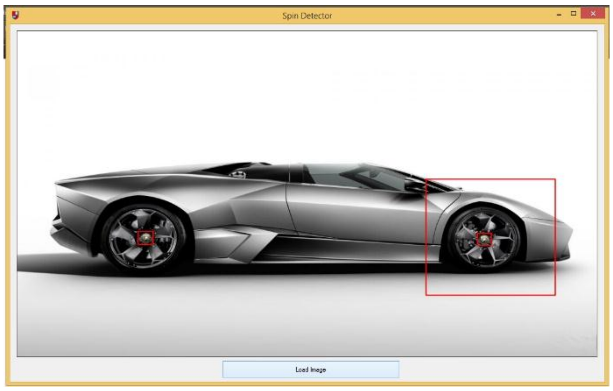

Figure 5. Detects incorrect portion

From the above snapshots we see that spin detection works fine with moving wheels that are completely visible without any tilt. But stationary wheels are also detected as they have a spinning pattern either in the center. Some motion images are also detected and small circular shapes are treated as spinning images.

VIII. RESULTS AND IMPROVEMENTS

Results:Train cascade is a good method to detect spinning images in an image but fails to recognize pattern a few times and a few times detect incorrect pattern. This problem is caused to due low resolution images, small dataset.

*Improvements and future work:*Improvements can be made by increasing the database size and training the images with different resolution images for removing the drawback of high resolution images. For future work we can train the classifier with more than 5000 positive images and 5000 negative images of different resolution and at different angles to train an efficient classifier.

IX. CONCLUSION

OpenCV has very efficient algorithms that enable us to apply various image transformation and pattern recognition functionalities. The classifier train functionality enables us to train a machine to detect particular patterns in an image. Using this application we can do various important task of weather forecasting, astronomy, movement of vehicles.

Improving data set and using proper resolution images we can easily improve the output of this application.

X. REFERENCES

[1] http://docs.opencv.org/index.html
[2] http://docs.opencv.org/doc/user_guide/ug_traincascade.html
[3] http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html?highlight=load#cascadeclassifier-load
[4] http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html?highlight=detectmultiscale#cascadeclassifier-detectmultiscale
[5] [Viola2001]:Paul Viola, Michael Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. Conference on Computer Vision and Pattern Recognition (CVPR), 2001, pp. 511-518.
[6] [Liao2007]: Shengcai Liao, Xiangxin Zhu, Zhen Lei, Lun Zhang and Stan Z. Li. Learning Multi-scale Block Local Binary Patterns for Face Recognition. International Conference on Biometrics (ICB), 2007, pp. 828-837.