GPCV code

Multi-Objective Interest Point Detector

Determines interest points on an image.

```
C++: void detectorMOP(cv::Mat imageInput,
                      cv::Mat &operatorOuput,
                      vector<cv::KeyPoint> &keypoints_,
                      double W,
                      double qualityLevel,
                      double minDistance,
                      cv::Mat Mask,
                      int patchSize,
                      bool orientation = false)
```

Parameters:

- imageInput – Input floating-point 32-bit single-channel image.
- operatorOutput - Output image of the MOP operator function.
- keypoints – Output cv::Keypoint vector of detected corners.
- W – Weight parameter of the Multi-Objective Interest Point detector (MOP) operator , with a range from 0.0 to 1.0; indicating dispersion and repeatability of the interest points, respectively.
- qualityLevel – Parameter characterizing the minimal accepted quality of the image points, with a range from 0.0 to 1.0. The parameter value is a percentage threshold of the maximun intensity value of the image (see "method"). The points less than the parameter are rejected. For example, if the maximum intensity value = 100 , and the qualityLevel=0.10 , then all the points with the intensity value less than 10 are rejected.
- minDistance – Minimum possible Euclidean distance between the returned points [pixels].
- mask – Optional region of interest, it specifies the region in which the points are detected. If mask = null , then the region is mindistance smaller than the region of the image .

- patchSize - Side lenght of the patch from each keypoint, the value is recomended to be an odd number so the keypoint can be in the center of the patch.
- orientation - Parameter that indicates if the gradient angle of the patch is calculated and stored for each keypoint, the default value is false.

The function finds the most prominent points in the image or in the specified image region, as described in [5, 2, 3]:

- Function calculates the MOP operator of the image.
- Function performs a non-maximum suppression using the nonMaximaSuppression() implemented by Hilton Bristow and is based on [1].
- The points with higher value of intensity than the qualityLevel are rejected using the threshold().
- Function throws away each point for which there is a stronger corner at a distance less than maxDistance.
- If orientation = true , function takes a patch of each keypoint with a size of [patchSize x patchSize] and computes the image gradiente using the Sobel() and cartToPolar() from openCV, and then it creates a histogram of the gradient orientation weighted with the gradient magnitud; the maximun bin from the histogram is the orientation [rad] of the patch stored in vector keypoint. .

The function can be used to initialize a point-based tracker of an object.

Example:

The example shows an image with a trackbar for the parameters of "w","qualityLevel" and "minDistance".

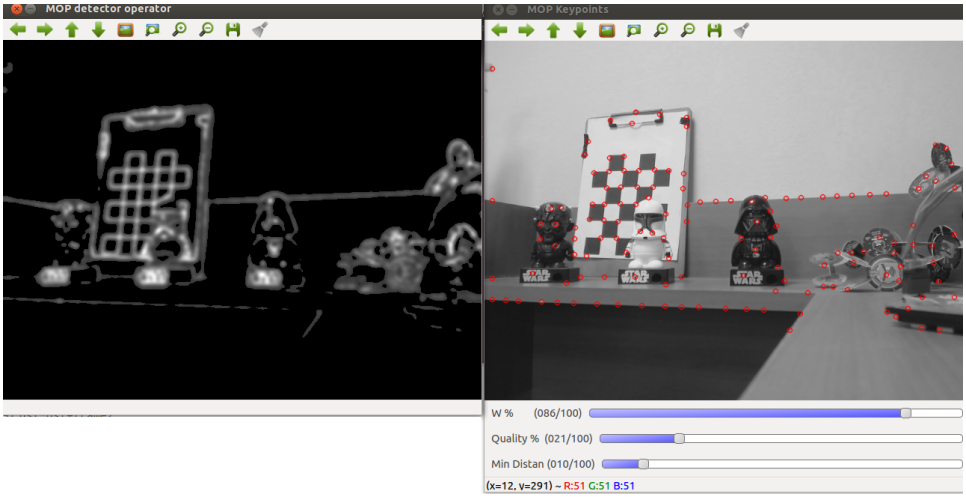
//code

```
/*
GPCVLibrary Copyright (C) 2014
Victor Raul Lopez Lopez,
Leonardo Trujillo Reyes,
Gustavo Olague Caballero,
Pierrick Legran.

GPCVLibrary is free software: you can redistribute it and/or
    modify it under the terms of the GNU General Public License
    as published by the Free Software Foundation, either version
     3 of the License, or (at your option) any later version.

GPCVLibrary is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty
    of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
    the GNU General Public License for more details.
```

```
You should have received a copy of the GNU General Public
    License along with GPCVLibrary. If not, see <http://www.gnu.
    org/licenses/>.
*/

#include "GPCV.h"
#include <stdio.h>
#include <opencv2/opencv.hpp>
#include <limits>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>

using namespace GPCV;
using namespace cv;
using namespace std;

int main()
{
cvNamedWindow("MOP Keypoints", 1); //Create window
CvCapture* capture = cvCaptureFromCAM(CV_CAP_ANY); //Capture
    using any camera connected to your system
Mat output;
char key;
int w_=50;
int QL=50;
int mD=10;
double iw_;
double iQL;
createTrackbar("W % ", "MOP Keypoints", &w_, 100);
createTrackbar("Quality % ","MOP Keypoints", &QL, 100);
createTrackbar("Min Distance [pixels] ", "MOP Keypoints", &mD,
    100);

while(true) {

//-- Step 1:Create image frames from capture and Convert images
    to CV_32FC1

iw_   = (w_)*0.01;
iQL   = (QL)*0.01;
vector<cv::KeyPoint> keypoints_1;
Mat img_1 = cvQueryFrame(capture);
img_1.convertTo(img_1,CV_32FC1);
cvtColor(img_1, img_1, CV_RGB2GRAY );

//Condition to avoid minDistance = 0
if(mD==0)
mD=1;

//-- Step 2: Detect the keypoints using detectorMOP()

detectorMOP(img_1, output, keypoints_1, iw_,iQL, mD, Mat(), 21 ,
     false);

//-- Step 3: Draw Keypoints and show frames on created window

img_1.convertTo(img_1,CV_8UC1);
```

```
drawKeypoints(img_1, keypoints_1, img_1, Scalar(0,0,255),
    DrawMatchesFlags::DEFAULT );
normalize(output,output, 0, 1, CV_MINMAX);

imshow("MOP detector operator", output);
imshow("MOP Keypoints", img_1);

key = cvWaitKey(2); //Capture Keyboard stroke

if (char(key) == 27)
{ break; //If you hit ESC key loop will break. }
}
return 0;
}
```

//output picture



GP-Hölder Region Descriptor

Determines a descriptor of an image.

```
C++: holderDescriptor(cv::Mat imageInput1,
                      vector<cv::KeyPoint> keypoints,
                      vector<int> samplesInput,
                      cv::Mat &descriptorOuput)
```

Parameters:

- imageInput – Image input type floating-point 32-bit, single-channel image.
- keypoints - Input vector of the detected keypoints.
- samplesInput – Input vector where the number of elements indicates the number of concentric circles to sample around the center of the image, the

value of the elements indicates the number_of_samples in each concentric circle; respectively. The Total_number_of_samples is equal to the sum of all the values in the vector plus one.

- descriptorOuput - Output image containing the Hölder descriptors of a patch of each Keypoint. The size of the image is [Total_number_of_samples x vector_keypoint_size].

The function constructs an image descriptor of the Hölder operator image of each keypoint, as described in [6, 4]:

- Function calculates a patch of each keypoint.
- Function calculates the Hölder operator of the patch using holderOperator().
- Function performs a concentric circle sampling of the Hölder operator image of the patch starting at a angles specified by the detected keypoint vector.
- Function constructs a descriptor with the sampled values as mentioned in the previous point.
- Funciton constructs an image with the descriptor of each patch.

The function can be used to match points in object detection aplication.

Example:

The example shows an image with a trackbar for the parameters of "k","qualityLevel" and "thershold".

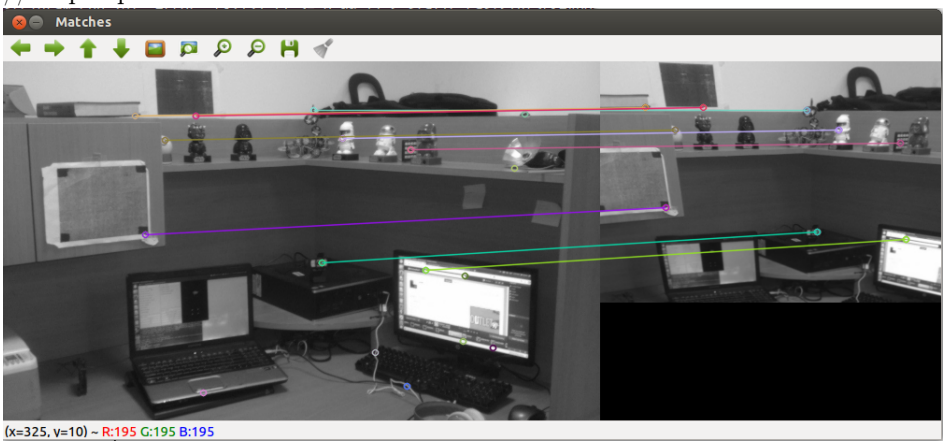
```
/*
GPCVLibrary Copyright (C) 2014
Victor Raul Lopez Lopez,
Leonardo Trujillo Reyes,
Gustavo Olague Caballero,
Pierrick Legran.

GPCVLibrary is free software: you can redistribute it and/or
    modify it under the terms of the GNU General Public License
    as published by the Free Software Foundation, either version
     3 of the License, or (at your option) any later version.

GPCVLibrary is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty
    of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
    the GNU General Public License for more details.

You should have received a copy of the GNU General Public
    License along with GPCVLibrary. If not, see <http://www.gnu.
    org/licenses/>.
*/

#include "GPCV.h"
```

```
#include <stdio.h>
#include <opencv2/opencv.hpp>
#include <limits>
#include <opencv2/core/core.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <numeric>
#include <iostream>
#include "opencv2/features2d/features2d.hpp"
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/nonfree/features2d.hpp"

using namespace GPCV;
using namespace cv;
using namespace std;

int main( int argc, char** argv ) {

  if( argc != 3 )
{ std::cout << " Usage: ./GPCVexample3 <img1> <img2>" << std::
    endl;
return -1; }

Mat img_1 = imread( argv[1], CV_LOAD_IMAGE_GRAYSCALE );
Mat img_2 = imread( argv[2], CV_LOAD_IMAGE_GRAYSCALE );

if( !img_1.data || !img_2.data )
{ return -1; }

  //-- Step 1: Convert images to CV_32FC1

img_1.convertTo(img_1,CV_32FC1);
img_2.convertTo(img_2,CV_32FC1);

  //-- Step 2: Detect the keypoints using detectorMOP()

// Detector Variables {
Mat output1, output2;
vector<cv::KeyPoint> keypoints_1, keypoints_2;
double w1 = 0.80; // Weight parameter of the MOP operator from
    the eq.(14) ref ().
double QL1 = 0.55; // Threshold parameter

double w2 = 0.80;
double QL2 = 0.40;
double s = 55;
double md= ((s/2));
bool ori = true;    // }

//*******************  Detector MOP  *******************//

detectorMOP(img_1,output1,keypoints_1,w1,QL1,md,Mat(),s,ori);
detectorMOP(img_2,output2,keypoints_2,w2,QL2,md,Mat(),s,ori);

//*****************************************************//

  if (keypoints_1.empty())
cerr << " Keypoint from image 1 is empty " << endl;
```

```
  if (keypoints_2.empty())
cerr << " Keypoint from image 2 is empty " << endl;

  //-- Step 3: Calculate descriptors (feature vectors) using
      holderDescriptor()

  // Descriptor Variables {

vector<int> samples;
samples.push_back(20); // # of samples of the first circle
samples.push_back(60); // # of samples of the second circle
samples.push_back(80); // # of samples of the third circle

  int sum = accumulate(samples.begin(), samples.end(),0) + 1; //
       Total number of samples, + 1 is the center point

cv::Mat descriptors_1(keypoints_1.size(),sum,CV_32F);
cv::Mat descriptors_2(keypoints_2.size(),sum,CV_32F);

  // }

//****************  Descriptor Holder  ********************//

  holderDescriptor(img_1, keypoints_1, samples, descriptors_1);
  holderDescriptor(img_2, keypoints_2, samples, descriptors_2);

//*********************************************************//

  //-- Step 4: Matching descriptor vectors using Brute Force
      matcher

BFMatcher matcher(NORM_L2,true);
std::vector< DMatch > matches;
matcher.match( descriptors_1, descriptors_2, matches );

img_1.convertTo(img_1,CV_8UC1);
img_2.convertTo(img_2,CV_8UC1);

drawKeypoints(img_1, keypoints_1, img_1, Scalar::all(-1),
    DrawMatchesFlags::DEFAULT );
drawKeypoints(img_2, keypoints_2, img_2, Scalar::all(-1),
    DrawMatchesFlags::DEFAULT );

  //-- Draw matches

Mat img_matches;
drawMatches( img_1, keypoints_1, img_2, keypoints_2, matches,
    img_matches );

  //-- Show detected matches

imshow("Matches", img_matches );

waitKey(0);
return 0;
```

```
}
```

```
//output picture
```



References

[1] Alexander Neubeck and Luc Van Gool. Efficient non-maximum suppression. In *Proceedings of the 18th International Conference on Pattern Recognition - Volume 03*, ICPR '06, pages 850–855, Washington, DC, USA, 2006. IEEE Computer Society.

[2] Gustavo Olague and Leonardo Trujillo. Evolutionary-computer-assisted design of image operators that detect interest points using genetic programming. *Image Vision Comput.*, 29(7):484–498, June 2011.

[3] Gustavo Olague and Leonardo Trujillo. Interest point detection through multiobjective genetic programming. *Appl. Soft Comput.*, 12(8):2566–2582, 2012.

[4] Leonardo Trujillo, Pierrick Legrand, Gustavo Olague, and Jacques LéVy-VéHel. Evolving estimators of the pointwise hölder exponent with genetic programming. *Inf. Sci.*, 209:61–79, November 2012.

[5] Leonardo Trujillo and Gustavo Olague. Automated design of image operators that detect interest points. *Evol. Comput.*, 16(4):483–507, 2008.

[6] Leonardo Trujillo, Gustavo Olague, Pierrick Legrand, and Evelyne Lutton. A new regularity based descriptor computed from local image oscillations. *Optics Express*, 15(10):6140–6145, 2007.