Histogram of Shearlet Coefficients (HSC) for C++

version 0.0.1

William Robson Schwartz
http://www.liv.ic.unicamp.br/~wschwartz/softwares.html

August 31, 2011

Contents

Chapter 1

Introduction

This library provides a C++ class called *HSC* to perform feature extraction using the histogram of shearlet coefficients (HSC), method proposed in [1].

Shearlet transforms provide a general framework for analyzing and representing data with anisotropic information at multiple scales. As a consequence, signal singularities, such as edges, can be precisely detected and located in images. Based on the idea of employing histograms to estimate the distribution of edge orientations and on the accurate multi-scale analysis provided by shearlet transforms, we propose a feature descriptor called Histograms of Shearlet Coefficients (HSC).

This code works either on Windows or on Linux and requires OpenCV version 1.0 or superior (http://opencv.willowgarage.com/wiki/). In Windows, a project for Visual Studio 2005 is provided. A Makefile can be used to compile all files and generate an executable **example**, containing an example of usage. To incorporate this library in your project, copy every .cpp and .h file to your directory and compile them with your code. Then, call the methods provided by the class *HSC*.

If you find bugs or problems in this software or you have suggestions to improve or make it more user friendly, please send an e-mail to williamrobschwartz@gmail.com.

This implementation has been used as part of the paper written by Schwartz et al. [1]. We kindly ask you to cite that reference upon the use of this code with the following bibtex entry.

```
@inproceedings{Schwartz:ICIP:2011,
  author={W. R. Schwartz and R. D. da Silva and L. S. Davis and H. Pedrini},
  title={{A Novel Feature Descriptor Based on the Shearlet Transform}},
  booktitle = {IEEE International Conference on Image Processing},
  pages = {1053--1056},
  year = 2011,
}
```

Chapter 2

Code Interface

2.1 Main Class

This library implements a C++ class called *HSC* that provides a set of methods to extract HSC feature descriptors. Listing 2.1 displays available methods for this class.

The feature extraction for a set of images works as follows. First, the parameters need to be set with method *SetParameters*, then HSC has to be initialized calling *InitializeExtractionMethod*. To extract features from an image, method *AddNewImage* is called once and *ExtractFeatures* is called with the block setup. Then, when a new image is presented, method *AddNewImage* needs to be called first, then *ExtractFeatures* can be called multiple times.

Listing 2.1: class HSC

```
1  class HSC {
2    // add new image for this extraction method
3    void AddNewImage(IplImage *img);
4
5    // initilize extraction method
6    void InitializeExtractionMethod();
7
8    // get number of features per block
9    int GetNFeatures();
10
11   // set parameters
12   void SetParameter(string name, string value);
13
14   // function to extract feature vectors
15   Vector<float> *ExtractFeatures(int blockW, int blockH, int strideX, int strideY);
16
17   // function to extract feature vectors considering image as a whole (not dividing in blocks)
18   Vector<float> *ExtractFeatures();
19 };
```

AddNewImage Method to set a new image to be considered for feature extraction. Every time a different image is to be considered, it needs to be passed as parameter to this method.

InitializeExtractionMethod Method responsible to set all parameters prior to the execution. It has to be called after the parameters are set.

GetNFeatures Method returns the number of feature descriptors that will be extracted per block. This number depends on the number of orientations used by the shearlet transform and the number of decomposition levels considered.

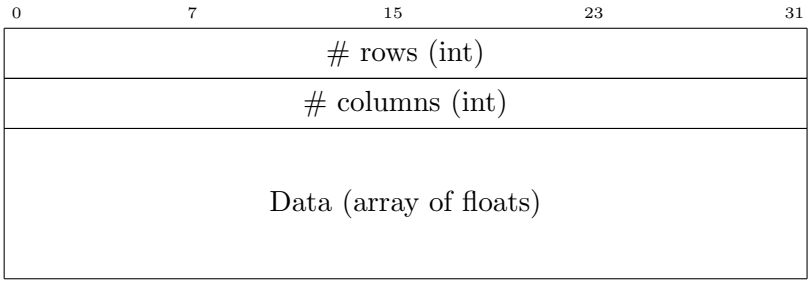
SetParameter Method to set parameters to be used by HSC. A list with descriptions and default values (in bold) is presented as follows.

Parameter	Description	Available values
LapFilter	filter for pyramid decomposition.	9/7, 5/3, **Burt**
norient	number of orientations considered.	1, 2, 4, **8**, 16, 32
maskSize	convolution mask to be used.	2, 4, 6, **8**, 10, 12, 14, 16, 18, 20, 22
nlevels	number of decomposition levels	1, **2**, ... 10
useCell	divide the block into cells (similar to HOG). If true, divide block into four cells and extract histograms for each.	false, **true**
normalization	perform histogram normalization (global: normalize after concatenation, local: normalize each histogram extracted from cells).	**global**, none, local

ExtractFeatures Method that performs feature extraction and returns a vector with float values. There are two options: (1) execute feature extraction considering blocks of blockW×blockH pixels with strides of strideX and strideY pixels. The resulting feature vector is the concatenation of the descriptors extracted from each block; (2) execute feature extraction considering the block size as the image size.

2.2 Auxiliary Classes

This library also implements some auxiliary classes: *Vector* and *Matrix*. Matrices and vectors can be either loaded or saved in files with format *feat*. This format only supports 32-bits float data type and is defined as follows (matrices are stored column-wise). To facilitate the use, two MATLAB functions (load_matrix.m and write_matrix.m) are provided to read and write matrices and vectors in format *feat*.



Listing 2.2: class Vector

```
1  template <class T>
2  class Vector {
3
4      // create a vector with elements
5      Vector(int n);
6
7      // load a vector from a file
8      Vector(string filename);
9
10     // retrieve the element at position x
11     T GetElement(int x);
12
13     // set element value at position x
14     void SetElement(int x, T value);
15
16     // retrieve number of elements in the vector
17     int GetNElements();
18
19     // write vector to file
20     void Write(string filename);
21
22     // copy vector
23     Vector<T> *Copy();
24 }
```

Vector Constructor *Vector* either creates a vector with n elements or loads a stored *feat* file defined by string filename.

GetElement Method *GetElement* access the element at position x. **Note**: the first element of the vector is at position 0.

SetElement Method *SetElement* attributes value to position x of the vector.

GetNElements Method *GetNElements* retrieves the number of elements contained in the vector.

Write Method *Write* saves the vector in a file defined by string filename in the format *feat*. **Note**: to use this function, the type of class Vector must be float.

Copy Method *Copy* duplicates vector and returns a pointer to the new vector.

Listing 2.3: class Matrix

```
template <class T>
class Vector {

    // create a matrix with r rows and c columns
    Matrix(int r, int c);

    // load a matrix from a file
    Matrix(string filename);

    // retrieve the element at row y and column x
    T GetElement(int x, int y);

    // set element value at row y and column x
    void SetValue(int x, int y, T value);

    // get number of rows
    int GetNRows();

    // get number of columns
    int GetNCols();

    // contatenate rows of matrices m1 and m2, return a new matrix
    Matrix<T> *ConcatenateMatricesRows(Matrix<T> *m1, Matrix<T> *m2);

    // copy matrix
    Matrix<T> *Copy();

    // retrieve row i of the matrix, return a new vector
    Vector<T> *GetRow(int i);

    // set row i of the matrix with the vector InVector
    void SetRow(Vector<T> *InVector, int r);

    // write matrix to a file
    void Write(string filename);
}
```

Matrix Constructor *Matrix* either loads a matrix with format *feat* from file defined by string filename or creates a matrix with r rows and c columns.

GetElement Method *GetElement* access matrix element at column x and row y. **Note**: the matrix indexation starts at position $(0, 0)$.

SetValue Method *SetValue* set value to element at column x and row y.

GetNRows Method *GetNRows* retrieves the number of rows in the matrix.

GetNCols Method *GetNCols* retrieves the number of columns in the matrix.

ConcatenateMatricesRows Method *ConcatenateMatricesRows* concatenates two matrices: m1 and m2, returning a third matrix with the same number of columns and m1.GetNRows() + m2.GetNRows() rows.

Copy Method *Copy* duplicates the matrix returning a pointer to the newly created matrix.

GetRow Method *GetRow* retrieves the row r of the matrix and return a pointer to the created vector.

SetRow Method *SetRow* sets the r row of the matrix with the vector InVector, which has the same number of elements as number of columns of the matrix.

Write Method *Write* saves the matrix to a file defined by string filename in format *feat*.

Chapter 3

Example

File example.cpp contains an example of feature extraction performed using HSC, shown in Listing 3.1. This example performs feature extraction considering blocks of 256×256 pixels and strides of 256 pixels. At the end, the resulting feature vector is saved in a file called *features.feat*.

Listing 3.1: Example of feature extraction using HSC.

```
#include "HSC.h"

void Example() {
Vector<float> *feat;
IplImage *img;
HSC hsc;

  /* set parameters */
  hsc.SetParameter("LapFilter", "Burt");   // filter for pyramid decomposition
  hsc.SetParameter("norient", "8");   // number of orientations
  hsc.SetParameter("maskSize", "4");   // convolution mask size
  hsc.SetParameter("nlevels", "1");   // number of decomposition levels
  hsc.SetParameter("useCell", "true");   // split block into four cells
  hsc.SetParameter("normalization", "global");  // normalization type

  /* initialize feature extraction method */
  hsc.InitializeExtractionMethod();

  /* feature extraction */
  img = cvLoadImage("peppers.pgm");    // load image
  hsc.AddNewImage(img);       // add image to extraction
  feat = hsc.ExtractFeatures(256, 256, 256, 256); // extract feature descriptors considering block
       sizes of 256x256 pixels

  /* save results */
  feat->Write("features.feat");
  return 0;
}
```

Bibliography

[1] W. R. Schwartz, R. D. da Silva, L. S. Davis, and H. Pedrini, "A Novel Feature Descriptor Based on the Shearlet Transform," in *IEEE International Conference on Image Processing*, 2011, pp. 1053–1056.