Sum and difference histograms

Brandon Marquez Salazar

Universidad de Guanajuato - August 23rd, 2022

1 Introduction

1.1 Texture (visual) definition

Described by an article of Sonymage [1], a visual texture refers to the appearance of an object, and can be given different adjectives such as rough, smooth, fine, coarse, etc.

For Dan Scott, texture regards the "*surface quality*"[2]. Also, Dan distinguishes between the physical texture and the illusion of texture. Where the first one is actual surface properties being appreciable by touch and sight; and the second only can be appreciated by sight.

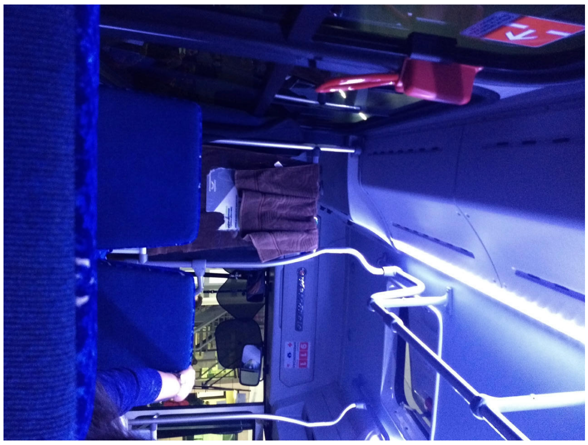
Following what Anil wrote in [3], "the term texture generally refers to repetition of basic texture elements called *texels*".

2 Methods and Materials

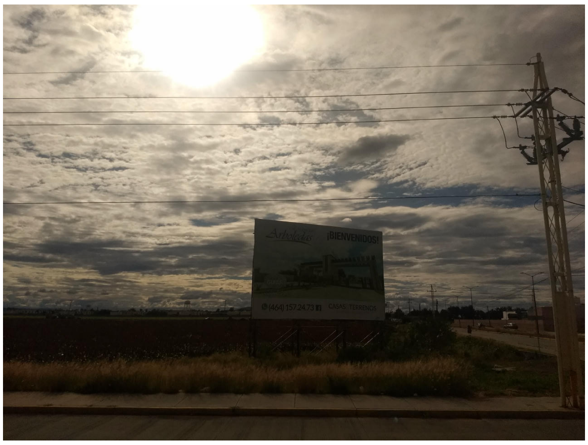
In this experiment, it'll be implemented the SDH algorithm explained in [4]. To afford this, the library **OpenCV** will be needed. The chosen language is **C++** due it's performance, explicitness and object oriented structure. On the other hand, for the experimentation, 4 images will be processed and documented in this text.

The images to be processed are the following:

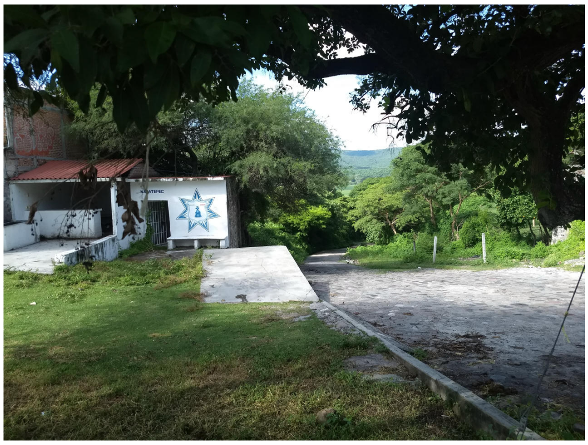
(a) Solid.png



(b) Bus.jpeg



(c) Sunset-cirro.jpeg



(d) Natur-police.jpeg

Figure 1: Images to be processed
Photos by Lang Lovdog Inu Oókami

3 Implementation of the SDH algorithm

For this experiment, it was implemented a class **SDH**, which uses some of the **OpenCV** library functions. The purpose is to compute texture properties from an image, for specific d and θ

First, of all, the library includes to assure all it needs to work properly.

SDH_feat.hxx

```
#ifndef __LOVDOG_SDH_FEAT_HXX__
#define __LOVDOG_SDH_FEAT_HXX__
#include <opencv2/highgui.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/core.hpp>
#include <vector>
#include <string>
```

Now, the class **SDH** was declared inside a namespace (**lovdog**) with the following attributes:

3.1 Public attributes

As a class, it's needed to define some public attributes and methods to be accesible to the user.

3.1.1 Static constants

Static constants for angle and histogram index.

SDH_feat.hxx

```
      static const uint
        ANGLE_0=0,
        ANGLE_45=45,
        ANGLE_90=90,
        ANGLE_135=135,
        ANGLE_180=180
      ;
      static constexpr uint ANGLE[5] = {
        ANGLE_0,
        ANGLE_45,
        ANGLE_90,
        ANGLE_135,
        ANGLE_180
      };
      static const int
        DIFF=0,
        SUM=1
      ;
```

3.1.2 Control variables and macros

Within the class, is also available some header macros for visualization and exports. And two control variables for log and verbosity.

SDH_feat.hxx

```
bool logOn;
char verbose;
static const uint
  ALL            = ~0,
  _MEAN_DIFF     = 0b000000000001,
  _MEAN_SUM      = 0b000000000010,
  _VARIANCE_DIFF = 0b000000000100,
  _VARIANCE_SUM  = 0b000000001000,
  CORRELATION    = 0b000000010000,
  CONTRAST       = 0b000000100000,
  HOMOGENEITY    = 0b000001000000,
  SHADOWNESS     = 0b000010000000,
  PROMINENCE     = 0b000100000000,
  ENERGY         = 0b001000000000,
  ENTROPY        = 0b010000000000,
  MEAN           = 0b100000000000
;
```

Some attributes for features and histograms.

SDH_feat.hxx

```
cv::Mat src;
uint
  d,
  angle
;
int
  dx,
  dy
;
double
  _mean[2],
  _variance[2],
  correlation,
  contrast,
  homogeneity,
  shadowness,
  prominence,
  energy,
  entropy,
  mean
;
```

3.2 Private attributes

There are few private attributes for the class, which are the histograms array and two pointers to each histogram.

SDH_feat.hxx

```
double
 *sumHist,
 *diffHist,
  Hist[2][511]
;
```

3.3 Public methods

The class **SDH** has public methods for access, (d, θ) definition and some other utilities most for the internal use of the class, but still public. Most of them are overloaded, but the most important ones are the following:

SDH_feat.hxx

```
SDH(const cv::Mat& src);
void set(int d, uint angle);
double  set(const int which_one, int index,
   double value);
double* at(const int which_one, int index);
double* atRel(const int which_one, size_t
   index);
void toCSV(std::string filename, unsigned
   int HEADER=0, bool append=false,
   std::string name="SDH");
static int getSDH(const cv::Mat& src, SDH&
   sdh);
static void computeFeatures(SDH& sdh);
static void toCSV(std::vector<SDH>& sdh,
   std::string filename, unsigned int
   HEADER=SDH::ALL);
static void toCSV_WriteHeader(std::string
   filename, unsigned int HEADER=SDH::ALL);
void printFeatures(unsigned int
   HEADER=SDH::ALL);
```

3.3.1 The constructor

There are other constructors, but here, the used in this experiment initializes the class with a given image. By default, the (d, θ) pair is set to (1,0).

3.3.2 The set method

The **set** method is intended to change the value of a given bin of any histogram.

3.3.3 The at method

The **at** method is intended to return a pointer to the element at the $index$ position of any histogram. Index ranges are from 0 to 510 for `SDH::SUM`, and from -255 to 255 for `SDH::DIFF`.

3.3.4 The atRel method

The **atRel** method is intended to return a pointer to the element at the $index$ position of any histogram, but relative to the 0 index used in CC. In other words, both `SDH::DIFF` and `SDH::SUM` start at 0.

3.3.5 The getSDH method

Here, the process of computing the SDH occurs, with a window-focused approach. This function computes two submatrices which will be Op1 and Op2, and then computes it's sum and difference matrices.

This method is overloaded allowing it to be called from the object with no arguments or with the facility to set the (d, θ) pair before processing the image.

With logOn=true, it exports to csv the SDH histograms, normalized and unnormalized.

3.3.6 The computeFeatures method

This method is where all the features are computed from the SDH histograms, there's an overload for it to be called from the object with no arguments.

The most important, is that it's needed to be called to compute the features after the SDH has been computed.

It's separate from the `getSDH` method because it's better having step by step each process.

3.3.7 The printFeatures method

The **printFeatures** method is intended to print the features of a `SDH` object, can be used with the macros described in 3.1.2

3.3.8 The toCSV and toCSV_WriteHeader methods

The **toCSV** method is intended to write the features of a `SDH` object to a CSV file. Can be used with the macros described in 3.1.2

toCSV_WriteHeaderC method is intended to write the header of a CSV file with the features of a `SDH` object, then the user may just append the features to the file if there are many `SDH` objects or they're within a loop.

3.4 Private methods

SDH_feat.hxx

```
        void incrementHist(int which_one, int index,
            double step=1);
        void decrementHist(int which_one, int index,
            double step=1);
```

3.4.1 The incrementHist and decrementHist methods

The **incrementHist** and **decrementHist** methods are intended to increment or decrement a bin of any histogram. The user can specify the step (by default, 1). The indexes here are the same as the `at` method, from 0 to 510 for `SDH::SUM`, and from -255 to 255 for `SDH::DIFF`.

3.5 Features extraction using the SDH algorithm

The program was made to receive an image and compute each of the eight SDH histograms from it.

In this section, the program as simple as it can be due to OOP paradigm, it's presented (only main function):

SDHmain.cpp

```
int main(int argc, char** argv){
  cv::Mat src; lovdogGetImage(argv[1], src);
  cv::cvtColor(src, src, cv::COLOR_BGR2GRAY);

  lovdog::SDH sdh(src);

  sdh.verbose = 1;
  uint ds[2] = {1,2} ;
  uint the_d, the_angle,
       mask =
          lovdog::SDH::HOMOGENEITY |
          lovdog::SDH::ENERGY      |
          lovdog::SDH::ENTROPY     |
          lovdog::SDH::MEAN        |
          lovdog::SDH::CONTRAST
        ;

  int the_err;

  std::string csvout =
    std::string("out/Features")+
    // Get only basename of file
    std::string(argv[1])
      .substr(std::string(argv[1])
      .find_last_of("/\\") + 1) +
    std::string(".csv");
  std::cout << "csvout: " << csvout << std::endl;

  lovdog::SDH::toCSV_WriteHeader(csvout, mask);

  the_d = the_angle = 0;
  while(the_d < 2){
    while(the_angle < 4){
      if((the_err=sdh.getSDH(ds[the_d],
          lovdog::SDH::ANGLE[the_angle]))){
        std::cout << "Error in getSDH, err: " <<
            the_err << std::endl;
        ++the_angle;
        continue;
      }
      sdh.computeFeatures();
      // Print all features
      //sdh.printFeatures(mask); std::cout <<
          std::endl;
      sdh.toCSV(csvout, mask, true, argv[1]);
      the_angle++;
    }
    the_angle = 0;
    the_d++;
  }

  return 0;
}
```

3.6 Results

3.6.1 SDH features with a pairs with d=1

This program will receive each image and will export a different csv for each one. And the results are show in the next tables.

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5493	57.8704	0.3818	0.0005	3.7026
Sunset-cirro.jpeg	102.3420	20.4588	0.5373	0.0007	3.5669
Natur-police.jpeg	79.0921	504.7280	0.2564	0.0002	4.2844

Table 1: SDH features with a pair of $(d, \theta) = (1, 0°)$.

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5503	136.8360	0.2921	0.0003	3.8736
Sunset-cirro.jpeg	102.3370	94.5174	0.3715	0.0004	3.8852
Natur-police.jpeg	79.1390	731.6960	0.2086	0.0001	4.4068

Table 2: SDH features with a pair of $(d, \theta) = (1, 45°)$.

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5524	82.5273	0.3741	0.0005	3.7241
Sunset-cirro.jpeg	102.3220	82.0923	0.4180	0.0004	3.8148
Natur-police.jpeg	79.1420	552.9490	0.2457	0.0002	4.3244

Table 3: SDH features with a pair of $(d, \theta) = (1, 90°)$.

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5503	112.7550	0.2980	0.0003	3.8535
Sunset-cirro.jpeg	102.3370	95.7842	0.3672	0.0003	3.8897
Natur-police.jpeg	79.1390	779.7240	0.2083	0.0001	4.4171

Table 4: SDH features with a pair of $(d, \theta) = (1, 135°)$.

3.6.2 SDH features with a pair of d=2

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5473	149.4110	0.2793	0.0003	3.8871
Sunset-cirro.jpeg	102.3570	42.9679	0.3993	0.0004	3.7914
Natur-police.jpeg	79.0884	834.9240	0.1998	0.0001	4.4174

Table 5: SDH features with a pair of $(d, \theta) = (2, 0°)$.

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5494	358.2190	0.2170	0.0002	4.0445
Sunset-cirro.jpeg	102.3460	166.8680	0.2779	0.0002	4.0828
Natur-police.jpeg	79.1831	1070.4500	0.1666	0.0001	4.5035

Table 6: SDH features with a pair of $(d, \theta) = (2, 45°)$.

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5536	235.5320	0.2593	0.0003	3.9400
Sunset-cirro.jpeg	102.3160	148.7230	0.3068	0.0003	4.0359
Natur-police.jpeg	79.1898	918.4460	0.1945	0.0001	4.4575

Table 7: SDH features with a pair of $(d, \theta) = (2, 90°)$.

Image	Mean	Contrast	Homogeneity	Energy	Entropy
Solid.png	153.0000	0.0000	1.0000	1.0000	-0.0000
Bus.jpeg	66.5495	292.1720	0.2243	0.0002	4.0146
Sunset-cirro.jpeg	102.3460	167.1190	0.2732	0.0002	4.0880
Natur-police.jpeg	79.1831	1111.9100	0.1675	0.0001	4.5102

Table 8: SDH features with a pair of $(d, \theta) = (2, 135°)$.

4 Final thoughts

Here the results for *Solid.png*, show an interesting behaviour compared to the *Sunset-cirro.jpeg* and *Natur-police.jpeg*.

As can be seen into every table, for *Solid.png* almost every value is the same no matter which paoir of (d, θ) is used.

Something similar happens with the **mean** of each image, which, no mather the pair of (d, θ) is used, the mean is almost the same per image.

Most of the metrics, slightly change depending on the pair of (d, θ) used.

On the other hand for the image *Bus.jpeg*, with $d = 1$, the **contrast** changes abruptly. This exaggerate changes occur also with *Natur-police.jpeg* specially when looking at the pair of (d, θ) = (2, 135°).

It seems redundant (in this experiment), having different pairs of (d, θ) for each image, except for the **contrast**.

Now, about how the numbers can be felt looking at the images, it's fair to point that *Solid.png* and *Sunset-cirro.jpeg* have the higher **mean** and **homogeneity**, and the smallest **contrast**; numbers which can be interpreted as less variation in the texture, which is what can be seen.

Finally, it's good to point the oposite case, which indicates different behaviour from *Natur-police.jpeg* and *Bus.jpeg* that have more variations in the texture.

4.1 Some future experiments

It would be interesting to make more experiments with other images with different textures, structures and illuminations, applying image enhancement techniques, variations on light, reducing the noise, etc.

Also, it would be interesting to see how classifiers can behave with those images, how dimensionality reduction algorithms can confirm if there are actual redundancies within the obtained features.