Performance analysis of a Bunch of Lines Descriptor-based KNN classifier

Marc Groefsema (S2055236)
Marc de Groot (S283445)

June 2015

1 Introduction

We live in an era of robots. In the years to come, robots will take a significant role in our lives. Preparing our food, cleaning our house or fetching our remotes. For this, robots will need to be able to distinguish objects. Mistaking one object for another might result into giving the wrong medication to a patient, with dramatic results.

It all started when we decided to do something with computer vision for ALICE[1]. We spoke with the team and they came up with the idea for recognition of objects.

In order to recognize objects, and textureless objects, we researched some common computer vision algorithms and decided to implement both SIFT[3] and BOLD[2]. For SIFT, there was already an implementation, provided by Rik Timmers. For BOLD, we created our own representation.

In an ideal environment, ALICE should get a small cropped picture of an object as input, recognize it using BOLD and/or SIFT and return the correct label. For this, we needed a way of separating the object from the environment. There was code which did precisely this, but using PointClouds. To this day, we have not succeeded in rewriting the code to return images suitable for LSD[4].

We did succeed in building a *Bunch of Lines Descriptor* (BOLD) with a K-Nearest-Neighbors classifier. With this, ALICE should be able to recognize objects based on their form, rather than just their texture.

2 Problem description

We want to make a part of an object recognition program, where a segmented texture of an object is given as input and the respective label as output. The classification will be done using a Bunch of Lines Descriptor (BOLD) as a feature extractor whilst using a Line Segment Detector (LSD) to preprocess the image for the BOLD feature extraction.

In order to train the program, a data-set was made using various objects found in your average robot lab cupboard. The data set exists of 34 items, 10 of those being textureless. Pictures were taken from different angles on a black background, as shown in figure 1. The performance, by using cross-validations,

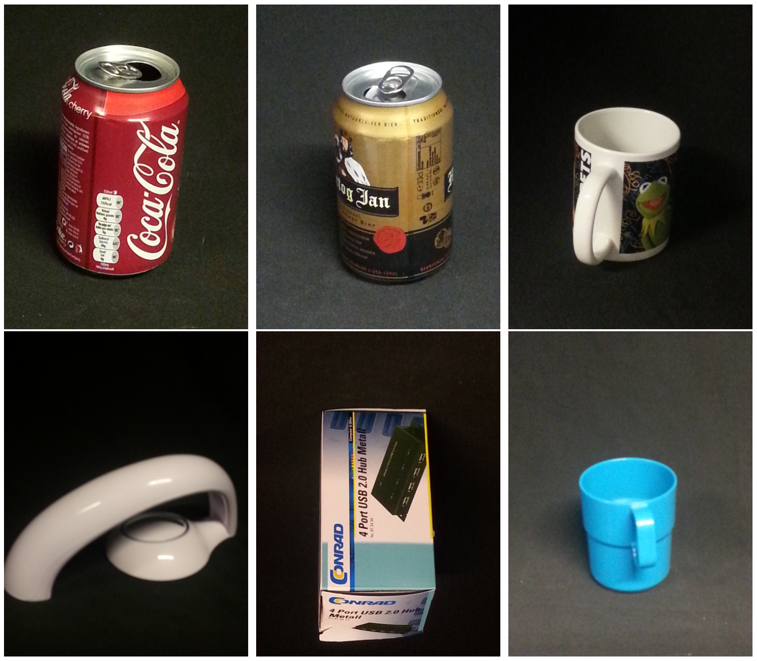
Figure 1: The custom BVD_M01

of the BOLD classification will be compared to SIFT and a method we thought of, combining BOLD with SIFT, called the *BOLDSIFTER*.

3 Problem analysis

3.1 Bunch of Lines Descriptor

For the BOLD Classifier, we need a line representation of an image as input. For this we chose to use the code provided with the LSD paper [4]. Once the lines are obtained, they are processed as described in the BOLD paper[2]. First line-pairs are set-up. Each line forms line-pairs with its K-Nearest lines. We chose for K=10, just as Tombari e.a. described[2]. Then the lines are processed. We will shortly repeat the geometric processing steps described by Tombari e.a.[2].

The geometric primitives deployed by BOLD are shown in figure 2. A line s_i is defined to have a begin-point e_{i1},an endpoint e_{i2} and a midpoint m_i. Besides the coordinates, the gradient at the mid-point of the line is a property as well, called $g(m_i)$.

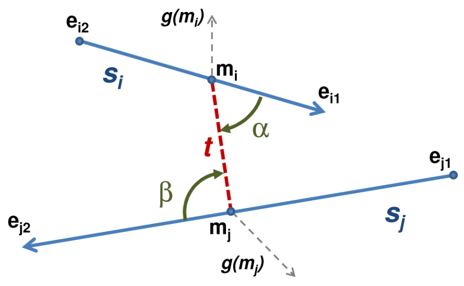

Figure 2: The geometric primitives deployed by BOLD (copied from [2])

When these properties are known, a new line is defined as the *midpoint segment*. This line has a begin point t_{ij} and an endpoint t_{ji}. These are defined as:

$$t_{ij} = m_j - m_i$$

$$t_{ji} = m_i - m_j$$

Next the lines have to be transformed to be canonical oriented lines, such that the line direction is independent of the texture orientation. The lines are transformed as follows:

$$sign(s_i) = \frac{(e_{i2} - e_{i1}) \times g(m_i)}{||(e_{i2} - e_{i1}) \times g(m_i)||} \cdot n$$

$$s_i = sign(s_i) \cdot (e_{i2} - e_{i1})$$

Where $\cdot$ is the dot product, $\times$ the cross product and n the unit vector normal to the image, pointing towards the observer.

$Sign(s_i)$ indicates whether the begin- and endpoints of the line must be swapped, in order to be oriented based on the gradient.

Since the lines are typical close to image contours, the gradients are expected to be high, causing robust orientation.

Next the angles α and β are calculated as shown in figure 2, representing the clockwise rotation of a line towards the midline segment. To do this, first the smaller angle between the vectors are calculated:

$$\alpha^* = arccos(\frac{s_i \cdot t_{ij}}{||s_i|| \cdot ||t_{ij}||})$$

$$\beta^* = arccos(\frac{s_j \cdot t_{ji}}{||s_j|| \cdot ||t_{ji}||})$$

Next, either the larger or the smaller angle is chosen:

$$\alpha = \begin{cases} \alpha^*, \frac{s_i \times t_{ij}}{||s_i \times t_{ij}||} \cdot n = 1 \\ 2\pi - \alpha^*, otherwise \end{cases}$$

$$\beta = \begin{cases} \beta^*, \frac{s_j \times t_{ji}}{||s_j \times t_{ji}||} \cdot n = 1 \\ 2\pi - \beta^*, otherwise \end{cases}$$

Now we can describe each line combination with an α and a β. Based on these angles a 2D-histogram is built, to represent all of the line combinations withing the K-nearest lines method. After adding all angle combinations, the histogram is normalized by dividing each element by the amount of combo-entries. This normalized 2D-histogram is the BOLD feature vector used in further classification and training.

3.2 Classification

New images are classified based on K-Nearest-Neighbor classification using Manhattan distances between the BOLD features. Trained features are stored in a dynamic array, with the respective label embedded in the feature class. We use the 3 nearest neighbors to classify new images.

3.3 BOLDSIFTER

After we implemented BOLD and added SIFT to the pipeline, we wanted to combine the two methods. We called it the BOLDSIFTER, as it uses both BOLD and SIFT. At classification an image is classified using BOLD. Next to the normal 3-nearest neighbors, BOLD stores a larger list of neighbors that are not used during BOLD classification. Of these nearest neighbors we sort the acquired neighbor label on frequency in the nearest neighbor list. This list is passed to SIFT. As the image is classified using SIFT, SIFT will only compare the features to the trained labels from the nearest neighbor list from BOLD.

If SIFT doesn't return an answer, the by BOLD acquired classification label is returned. Otherwise the answer from the directed SIFT is returned as answer.
This way SIFT needs to compare a lot less features, causing a speedup. We hope that less false positives are generated by SIFT this way, and more robust true positives.

4 Program Description

The code of our program is available on Github (https://github.com/MarcGroef/ros-vision) or clone-able via (https://github.com/MarcGroef/ros-vision.git). The general layout of the program is shown in figure 3. The following sections will guide you through the diagram.

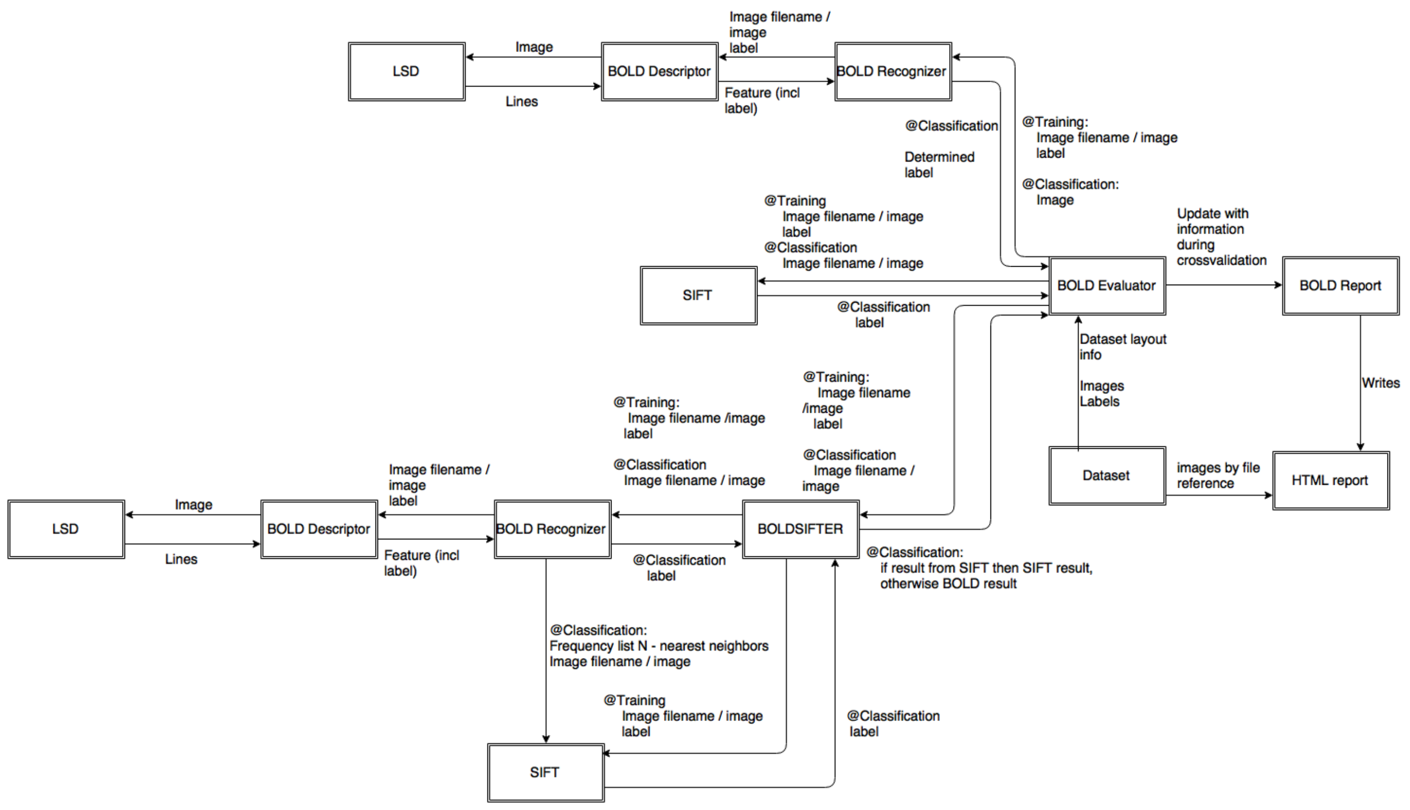
LSD
Image
Lines
BOLD Descriptor
Image filename / image label
Feature (incl label)
BOLD Recognizer
@Classification
Determined label
@Training: Image filename / image label
@Classification: Image
@Training Image filename / image label
@Classification Image filename / image
SIFT
@Classification label
BOLD Evaluator
Update with information during crossvalidation
BOLD Report
Writes
HTML report
Dataset layout info
Images Labels
Dataset
images by file reference
@Training: Image filename /image label
@Classification Image filename / image
@Training: Image filename /image label
@Classification Image filename / image
LSD
Image
Lines
BOLD Descriptor
Image filename / image label
Feature (incl label)
BOLD Recognizer
@Classification label
BOLDSIFTER
@Classification: if result from SIFT then SIFT result, otherwise BOLD result
@Classification: Frequency list N - nearest neighbors Image filename / image
@Training Image filename / image label
@Classification label
SIFT

Figure 3: Pipeline diagram of the program

4.1 BOLD Evaluator (in diagram)

This is the base class of the program. The *BOLD Evaluator* reads the data set, and embeds the object recognition techniques for training and classification during crossvalidations. During the crossvalidations the results of each classification are passed to the *BOLD Report*(described later) to export the classification results.

4.2 BOLD Datum

A BOLD Datum contains a label and a filename + directory. This class exists to pass label and filename information between other objects. The filename is passed since an image file is only read at the last possible moment it is needed. It is also useful for the *BOLD Report*, such that in case of wrong classification, the report can show which particular image an classified image is incorrectly compared to.

4.3 BOLD Recognizer (in diagram)

This class is able to perform K-Nearest Neighbor (KNN) classification. It stores trained features gathered from training and commands the *BOLD Recognizer*(described next) to acquire new features given an image. The *BOLD Recognizer* is able to train and classify on Mat images from openCV and on filenames including their directory. In the second case the *BOLD Descriptor* will load the images itself. This is used during classification, but for live robotics purposes also a Mat image can be classified and trained on.

BOLD Vector, a vector class we made to use the algebraic expressions from the BOLD paper[2] exactly the way described in the paper.

4.4 BOLD Descriptor (in diagram)

This class extracts a BOLD feature from image, after it is preprocessed by *LSD*(described next). LSD provides the BOLD Descriptor a bunch of lines, for which the following is done, as described by Tombari e.a.[2]

The lines, provided by LSD, are transformed into canonically oriented line segments. Only the gradient at the mid-point of the lines had still to be known, for which we wrote a custom function, which calculates the gradient given a line and an image. This gradient must be non-zero, otherwise it would result in a division-by-zero. Hence each line that has a gradient of zero is omitted and forgotten during feature extraction. In practice this happens once or twice every ten images.

After the feature is normalized, it is passed back to the BOLD Recognizer, which will store the feature-label combination as a trained feature during training, or classify the image using KNN and the trained features.

4.5 Line Segment Detector (LSD) (in diagram)

The LSD was provided by the paper in which it was introduced. [4]. We changed nothing in this code. Images where transformed by the BOLD Descriptor in such a way it could be used by LSD.

4.6 SIFT (in diagram)

The SIFT class is copied from the existing code from the BORG software, named *object_detection*. The code is based on the *openCV* implementation and is modified to handle the *BOLD Datum* class. This way the SIFT class can be nicely integrated in the *BOLD Evaluator*. SIFT is introduced by David Lowe in his paper Object Recognition From Local Scale-Invariant Features [3].

4.7 BOLDSIFTER (in diagram)

This class combines the BOLD and SIFT techniques. Each time the BOLDSIFTER is trained, it trains a BOLD Recognizer object and a SIFT object. When an image is classified, the image is classified first by the BOLD Descriptor. After this SIFT is used for classification, but SIFT does not check all the items it is trained on. The BOLD Recognizer stores a larger set of neighbors than used for KNN classification. It uses the three nearest neighbors for classification, but it stores the ten nearest neighbors. The labels are sorted based on the number of occurrences. These labels are passed to SIFT. When SIFT matches the features of the new image against trained features, SIFT only considers the trained features that carry the labels from the passed frequency list.

If SIFT does not return a matched label, the result of the BOLD classification is returned, otherwise the result from SIFT.

4.8 BOLD Report (in diagram)

The BOLD Report is a class that stores all the results and timings during crossvalidations. When the crossvalidation has finished, it makes a report in HTML. This report will be stored as *NewestReport* in the *Reports* folder. (If the Reports folder does not exist, it will be created.) This report shows the general results of the crossvalidation, and more detailed results of each label and even of each tested image. For all report depths, the images to which the testimage is incorrectly compared, are shown within the reports for convenience.

5 Usage

Our program can be called by using *rosrun* bold bold [parameters...]

- *'dia'* summons a dialogue in which the user can present training and test data.

- *'train' <int n> <float f>* trains on the first n items from our BVD_M01 training set. A portion of f will be used for testing purposes.
- *'crossfold' <int nFold> <int n> <float fracTestSet>*performs a nFold-crossfold on the first n items of our *BVD_M01* training set. A portion of *fracTestSet* will be used for testing purposes.

Training data can be saved, and loaded afterwards in the dialogue.

At times, it may be desired to use a subset of the *BVD_M01* data set. In order to do this, the *info.txt* can be edited. The first number lists the amount of items used, the next entries should be the names of the folders the items are in. If data is added, it is advised to add it in a similar way - create a folder with a title describing the object and place all the pictures of the object in this folder, together with an *info.txt* listing the amount of pictures in the folder, and the names of the respective pictures.

To optimize feedback, the testing mode can give feedback in a HTML-file, stating useful feedback data such as the images that are misclassified. In order to save the report, simply edit the name of the folder before running testing again.

6 Evaluation

6.1 Experiment

We evaluate the three different techniques using cross-validations with 10% as testset. We test the performance for the entire BVD_M01 data set, and a sub-set of 10 textureless objects. Both the mean correct/false ratio is printed and the mean processing time of 1 classification. All three techniques are tested each fold within the same testdata/trainingsdata split.

We are interested whether there is a difference between the techniques within the two experiments and between the two experiments. For this Chi-squared tests will be performed.

Our null-hypothesis is that there are no within-experiment and between-experiment differences between the recognition techniques. The null-hypothesis will be rejected when the p-value is smaller than 0.05.

6.2 Results

Table 1: Textureless experiment results

Textureless	% correct	% false	average time per classification (ms)
BOLD	65	34	75
SIFT	9	90	3923
BOLDSIFTER	64	35	1580

Table 2: Complete experiment results

Complete data set	% correct	% false	average time per classification (ms)
BOLD	50	50	84
SIFT	26	73	22580
BOLDSIFTER	55	44	4752

6.3 Analysis

6.3.1 Within-analysis

Within the experiment on the textureless data, SIFT performs significantly worse than BOLD and BOLDSIFTER ($\chi^2 = 1351.0$, $p << 0.05$ for BOLD vs. SIFT) and ($\chi^2 = 1310.4$, $p << 0.05$ for BOLDSIFTER vs. SIFT)
There is no significant difference between BOLD and BOLDSIFTER ($\chi^2 = 0.4$, $p = 0.52$.

Within the experiment on all the data, BOLD again out-performs SIFT ($\chi^2 = 236.5$, $p << 0.05$, just as BOLDSIFTER does ($\chi^2 = 350.2$, $p << 0.05$.)
There is also a significant difference between BOLD and BOLDSIFTER ($\chi^2 = 12.09$, $p = 0.0005$.

6.3.2 Between-analysis

Next all technique performances will be compared from the textureless-data crossvalidations and the complete-data crossvalidation.

BOLD does perform significantly better on textureless items than on a pool of both textured and textureless.($\chi^2 = 99.3$, $p << 0.05$)
SIFT does perform significantly better on a pool of both textured and textureless items than on only textureless items.($\chi^2 = 199.4$, $p << 0.05$)
BOLDSIFTER does perform significantly better on textureless items than on a pool of both textured and textureless.($\chi^2 = 33.7$, $p << 0.05$)

6.3.3 Time comparison

We cannot analytically compare the different average classification times, since we did not store enough information during the experiment to calculate variances. However, as a rough rule of thumb, it can be said that the variances in practice where small (about $\pm 10\%$), hence the mean timings in the table are a good indication of the required processing times.

7 Discussion

We have experimented with both BOLD, SIFT and a combination of the two, the BOLDSIFTER.

On a data set of purely textureless objects, BOLD is both faster and slightly outperforms the BOLDSIFTER by one percent. It is when there are both textureless and textured objects, that BOLDSIFT begins to outperform BOLD. This, however, comes at a cost - BOLDSIFT classifies at 0.2 items per second, where BOLD can classify almost 12 items per second. Pure SIFT is vastly outperformed by BOLD on both performance and speed.

If time matters not, BOLDSIFT should gain serious consideration. If time is at all a factor, however, BOLD performs nearly as well and at much greater speed. Using SIFT without prior knowledge or heuristics implemented (as given by BOLD in our BOLDSIFTer) is against our advice.

References

[1] Amirhosein Shantia, Anton Mulder, Ben Wolf, Rik Timmers, Rick van der Mark, Levente Sandor, Luis Knigge, Stef van der Struijk, Gereon Vienken, Francesco Bidoia, and Noel Luneburg. *BORG Team Description Paper* 2015

[2] F. Tombari, A. Franchi, L. Di Stefano, *“BOLD features to detect texture-less objects”* ICCV 2013

[3] Lowe, David G. (1999). *”Object recognition from local scale-invariant features”*. Proceedings of the International Conference on Computer Vision 2. pp. 1150–1157.

[4] Rafael Grompone von Gioi, Jeremie, J., Morel, J.-M., Gregory Randall. *“LSD: A fast line segment detector with a false detection control,”* IEEE Trans. Pattern Anal. Mach. Intell. , 32(4), 2010, pp. 722-732