Homework. 7: Building a Codebook using $k-$means

Ignacio Vizzo, E-Mail ivizzo@uni-bonn.de

Handout : 31.05.2020
Handin: 19.06.2020 at 23:59:59 (CET)

General rules

1. You need to provide the build system for this homework. This means, you need to provide as many **CMakeLists.txt** files as you think is needed.
2. Your code will be checked using `clang-format`. If your code in not properly formatted, then the tests will not run.
3. Your code will be static-analyzed by `clang-tidy`, if your code does not comply with the analysis, the tests will not run. If you are using VSCode with the reccomended setup, most of the errors will be visible while you program.
4. **ALL** tasks should be solved within the `homework_7` folder, no need to create `task_x` folders.
5. The design of how to solve this exercise is up to you. The **ONLY** requirement to pass the tests are:
   - You should provide one header file (put it where you preffer) and call it `homework_7.h`.
   - All the functions of this exercise must be accessible form this header file
   - You should guarantee this header file can be included thorugh your build system.
6. Do not add binary files or images to your repository. The only images you are allowed to submit is the ones that comes with this homework. If any **binary** file or image file(**.png**) is found on your submission, then your homework will be completely discarded.

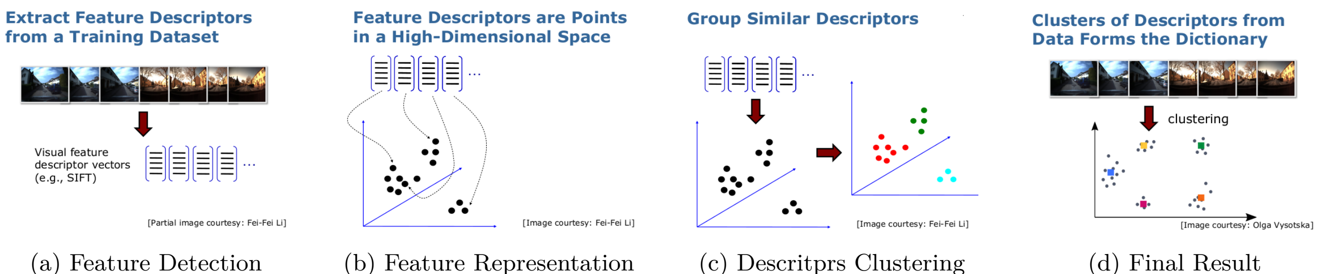
A Building a Bag Of Visual Words Dictionary using k-means`(10 points)`

In this homework you will be creating a **class** to build a Bag of Visual Words dictionary.

Your task is to create a "Dictionary", also called "Codebook", that represents a given dataset. The key idea behind this dictionary is to somehow "represent" or "model" the dataset. This dictionary will be used later on to query input sensor data and attempt to recognize if a robot has already seen the place or not. This is also called "Place Recognition".

To do this you will compute **SIFTS** descriptors for all input images of the dataset, and try to build a **Dictionary** from this. Using `ALL` the descriptors is not feasible, so, in order to reduce the number of features, or "words", you need reduce the dimensionality. One popular way of doing this, is **clustering** the features into a set of **k** most representative features, also called "centroids." In your case you will be implementing a variation of one of the most popular clustering algorithms: ***k*-means**.

An overview of the pipeline you will implement in this homework is shown in Figure 1:



(a) Feature Detection (b) Feature Representation (c) Descritprs Clustering (d) Final Result

Figure 1: Homework 7 pipeline overview: Build a Bag of Visual Words dictionary. First we need to extract the features from the dataset, then we cluster these features into k different centroids and lastly, each centroid represents a given visual word in the dictionary

A.1 k-means

In this task you need to implement the k**-means** algorithm. To do so you basically have two options:

- Use the k-means implementation from the *OpenCV* library. This algorithm is defined in the `opencv2/core.hpp` header file. This option is a good starting point, even if you consider moving to the next option.
- Write your own implementation of k-means. This will be more challenging and time-consuming, **BUT**, keep in mind that for your final project you **MUST** implement it anyways. You can basically save time now for your final project. This is the recomendded option if you have the time.

A.1.1 Requirements to pass the tests

- Define and implement a function that performs clustering on a set of descriptors, this function **must** be under the **ipb** namespace and have the following signature:

```
/**
* @brief
* 1. Given cluster centroids i initialized in some way,
* 2. For iteration t=1..T:
*   1. Compute the distance from each point x to each cluster centroid ,
*   2. Assign each point to the centroid it is closest to,
*   3. Recompute each centroid  as the mean of all points assigned to it,
*
* @param descriptors    The input SIFT descriptors to cluster.
* @param k              The size of the dictionary, ie, number of visual words.
* @param max_iterations Maximum number of iterations before convergence.
* @return cv::Mat       One unique Matrix representing all the $k$-means(stacked).
*/
cv::Mat kMeans(const std::vector<cv::Mat> &descriptors, int k, int max_iter);
```

- Provide thorugh the build system a library called **kmeans**, the bot will link against this library to test your implementation.

A.1.2 *Tips*

1. For this task you might want to reuse the serialization library (or a part of it) from **homework_5**. It's much faster to work with pre-computed SIFT descriptors saved in the filesystem than doing it online.
2. It's a good idea to inspect the **tests** that the `hw_bot` will run before implemting this exercise, this will give you a broad overview of what you need to achieve.
3. To test your code, make sure you start simple, don't try to cluster a really big amount of features or test your implementation with a huge dataset, it will most likely fail the very first time and you want to avoid some pain.
4. It would be great if you start writing your own unit-tests. This will guarantee you that your code is working beyond your imagination.
5. Before you even start coding, make sure to watch the Introduction on Bag of Visual words video: `https://www.youtube.com/watch?v=a4cFONdc6nc`

A.2 Bag of Visual Words Dictionary

Write a C++**class** to create the dictionary. This class will help you to take advantage of your k-means implementation and add high-level functionality to it. Writing this class, is what typically you **can't** do with languages like `C` where you must conform with justs plain functions.

Keep in mind that this class will be used in your final project, so you are free to add functionality to it if you feel it will help you in the future.

A.2.1 Requirements to pass the tests

- Reuse the serialization library from `homework_5`, for this you could copy paste all the code. You will need to provide:
  1. A library called **serialization** with all the functionality you implemented for homework 5.
  2. Access to all the methods from the `homework_7.h` header file.
- Define and implement a **class** called **BowDictionary** this class **must** be under the **ipb**.

- This class must be a **singleton** class. It makes no sense to create multiple copies of the dictionary for the same application. So you must ensure that nobody can copy/create new objects of this class. The tests will not even compile if you don't implement this pattern.
- To properly represent the data we want to model, the **BowDictionary** must containt the following **private** data members(you pick the name):
  1. The maximum number of iterations for the clustering algorithm `(int)`.
  2. The dictionary size, ie, the number of words `(int)`.
  3. The input descriptors `(vector)`.
  4. The computed dictionary/codebook `(Mat)`.
- You also need to provide **public** **`setters`** and **`getters`** methods for all the class data members. Keep in mind whatever these method should be marked as `const` or not. Here is a list of the names that the bot expect while testing your code. **NOTE** This functions declarations are stripped from the type, input parameters, and any other modifier(like const) It's just the naming convention:

```
// Getters methods
max_iterations();
size(); // number of centroids / codewords
descriptors();
vocabulary();
total_features(); // number of input features
empty();

// Setters methods
set_max_iterations();
set_size();
set_descriptors();
```

- Provide a class member function that set all the input parameters of the **BowDictionary**, and call it `set_params()`. This **public** member function should take as input parameters the the Maximum number of iterations, the size of the dictionary and the descriptors in that order.
- If **ANY** of the input parameters change, then you **MUST** guarantee that the **BowDictionary** will also change, this means that you will need to recompute some part of it if necessary. This step is particually important, because it will give you the robustness of knowing that no matter what you change in the dictionary, you will always have available the latest version(plus the fact that is a singleton class tells you that you can't have an "outdated" version of the dictionary).

NOTE 1: Remember to test your code with small examples, otherwise you will most likely suffer some pain. If the simple scenarios and cases doesn't work, then the big ones will never do.

NOTE 2: If you want to play with the complete dataset that you will be using for your final project, you can download it from : `https://uni-bonn.sciebo.de/s/c2d0a1ebbe575fdba2a35a8033f1e2ab`