SUN YAT-SEN UNIVERSITY, SYSU-CMU JOINT INSTITUTION OF ENGINEERING

Final Project Code Manual

Shitao Weng(sweng@andrew.cmu.edu)
Shushan Chen(shushanc@andrew.cmu.edu)

November 29, 2014

Contents

1 Introduction

2 SIFT

3 SIFT MATCH & KDTREE

3.1 SIFT MATCH

Relevant source files:

1. src/include/SiftMatcher.h

3.1.1 Load Training Datas

```
void loadDir(const char *dirName);
void loadFile(const char *fileName);
void loadFeatures(std::vector<Feature> & inputFeat);
```

@ input: [directory name] / [file name] / [a set of feature].

@ description: These functions are called to add feature points into training database. It is easily to be understood that function **loadDir()** will call **loadFile()**.

3.1.2 Build KD-TREE

```
void setup();
```

@ description: This function should be called after you load all the training image into this class object.

It will build a **KD-TREE** on existed template feature points.

3.1.3 Match

```
std::pair<Feature *, Feature *> match(Feature & input);
unsigned long match(vector<Feature> &inputFeats);
```

@ input: [Feature] / [A set of Features]

@ description: These function are called to match input feature points. It is easily understood that function **match(vector<>)** will call function **match(Feature &)**.

Function **match(vector<>)** will return a unique Tag, which can be used to find an object(Typically a name of an matched object in template database).

Function **matchFeature &** input a feature, and search the nearest and the second nearest feature point in the **KD-TREE**.

These two nearest feature points are from different objects(To be more clear, if using on face recognition, these two points should from two different **people**).

These two nearest features will be tested using the following function:

```
bool isGoodMatch(std::pair<Feature *, Feature *> matchs, Feature &inputFeat) {
    ...
    ...
```

```
    return (bestVal / secBestVal < matchRatio);
}
```

@ input: The nearest and second nearest matched features

@ output: Good Match or Not.

@ description: What it does is simply check if the ratio between the distances from the input feature is lower then a ceil value(By default, 0.8).

3.2 KD TREE

In this section, we give an function-level introduction of our implementation for **KD-TREE**. You don't need to read it if you only want to use the front-end functions.

3.2.1 BUILD KD TREE

```
void buildTree(std::vector<Feature> & features);
```

@ input: A set of features

@ description: Build a kd-tree on the input features.

```
void split( KDNode * parent );
```

@ description: This function is called by function **buildTree()**, it will recursive split the nodes. At every split process, it will call:

```
int selectDimension( KDNode * node );
```

@ input: A node that is being splited.

@ output: Dimension(among 128 dimensions) with the largest variance.

@ description: It return the dimension with the largest variance. It is easily undetstood that spliting on this dimension will seperate the features into two sets with similar size.

```
double findMedian( KDNode * node, int k );
```

@ input: A node that is being splited, selected dimension.

@ output: Median in kth dimension.

@ description: After selecting the best dimension, function **split** will split its feature points into two sets comparing with the median of them.

3.2.2 BBF SEARCH

After generating a balanced kd-tree, the remaining of this class is searching process.

```
std::pair<Feature *,Feature *> bbfNearest( Feature & input );
```

@ input: A feature. **@ output**: The nearest and the second nearest features on the kd-tree for the input feature.

@ description: This is basically a **dfs** search process with support of prioriry quque, this search strategy is introduced by Dr. Lowe[1]. The basical idea is searching the closer branch firstly, and drop the very bad branches.

3.2.3 BUILD KD TREE

4 DEMO

References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.