Loop Closure Guide

*Jacky Chen*

*Updated: 12/19/2010*

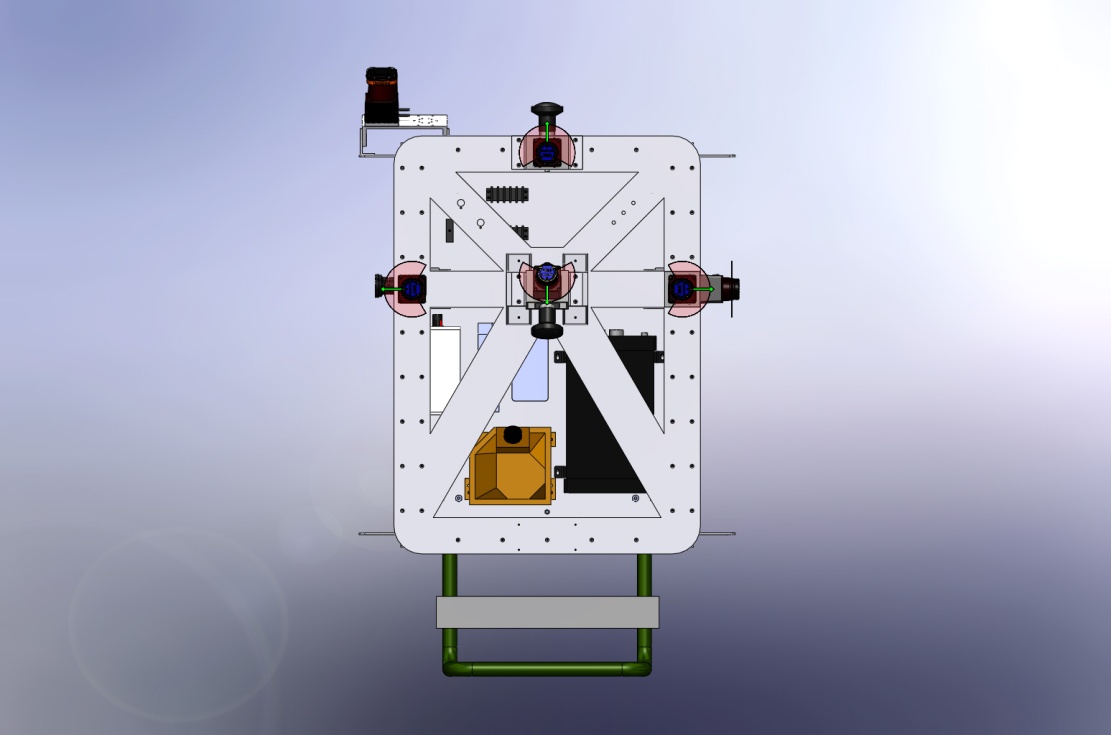
*Datasets supported: May and August, 2010*

**1. Introduction**

This document describes how to identify loop closure events as pairs of images, obtained by the indoor modeling backpack data acquisition system, which was designed by the Video and Image Processing Laboratory at the University of California, Berkeley. All of the code required for identifying loop closures can be found at \\bam\Code Transfer\Jacky Chen.

**2. Backpack Configuration for Modeling**

The loop closure algorithm can operate using just one out of four cameras. The left camera is the most helpful (up showing ceiling and down showing floor) and is used in all cases, with the right camera also used if we want to detect 180° loop closures. The configuration of the modeling sensors on the backpack is shown below, based on the May 4, 2010 dataset. Sensors are named based on the direction they face: left, right, up, or down.



right scanner & right camera

down camera & down scanner

left camera & left scanner

up camera & up scanner

Useful information about the cameras:

|  |  |  |
| --- | --- | --- |
| **Camera Name** | **Serial Number** | **Example Image Name** |
| left camera | 7240131 | Camera\_7240131\_Image000001.jpg |
| right camera | 9071192 | Camera\_9071192\_Image000001.jpg |

**3. File Formats**

Here’s a list of the different file formats used as input for identifying loop closures.

*.texture*: ASCII file that is used to specify exactly what images are to be used in the texture map of a 3d model. This file is important for performing IBR, but it is also handy in providing a subset of the entire image.

*.feat*: Binary file that stores all of the SIFT features in a particular image. File format is the same as that of David Lowe: <http://www.cs.ubc.ca/~lowe/keypoints/>

There will be other input files (e.g. .mat, .txt), but they are intermediary as the output of first few programs.

**4. Loop Closure List Generation**

***4.1 Overview***

The following sections will guide you through all the programs and procedure calls needed to generate a list of genuine loop closures. Sections 4.2 to 4.4 do this for only the left camera, with Section 4.5 articulating any differences when dealing with loop closures formed from both cameras. If loop closures from different cameras are of interest, be sure to read Sections 4.2 to 4.4 anyway.

***4.2 Setting up a dataset folder***

Within the directory where the code resides, create a folder for each new dataset in which all input data (i.e. .feat files, .texture file) will reside. All results will be written to the parent directory (where the code is). The folder should be named using the date of acquisition and dataset number. For example, a good name of the folder for the third dataset acquired on May 4, 2010 might be "20100504\_3".

Image features along with the texture file for a particular dataset can be downloaded from [\\behshahr\data\Backpack](file:///\\behshahr\data\Backpack), with the corresponding images viewable in \\bam\Shared Folder\Data Collection\*Date*\*Dataset*. Download all features from the left camera to the directory. All features from the left camera are by default named features\_Camera\_7240131\_ImageX.feat. Do the same for the right camera if necessary, where all features from the right camera are by default named features\_Camera\_9071192\_ImageX.feat. The texture file is by default named imagesUsed\_X.texture, where X is the number corresponding to a camera, so download the one that matches with the left.

***4.3 FAB-MAP***

***4.3.1 Choosing a subset of images using .texture file from IBR***

All features from a dataset must reside in the corresponding dataset folder before this step.

Type the following at the Matlab Command Window so that what’s left after is the subset of images we’ll use.

1. dataset = ‘datasetDate\_datasetNumber’; //or whatever you named your folder to be
2. imageFilter2(dataset, ‘sift’); //2nd argument indicates it’s filtering features, not images

In the same folder, you should now see only the subset of features as suggested by the .texture file.

***4.3.2 Generating an observation matrix***

Here we use the bag-of-words approach to change images to observation vectors that indicate the presences of words/features. We first build a vocabulary using features of no more than 100 images from the subset via clustering. Then, we go through all images and convert them into vectors of binary variables that correspond to words. The output is a matrix with each row being an observation vector.

This step also throws out any images with a low count of features because they could skew the results of loop closure detection. A threshold needs to be supplied as input.

Type the following at the Matlab Command Window:

1. dataset = ‘datasetDate\_datassetNumber’; //or whatever the folder’s named

2. threshold = 300; //this is good for a set of ~100 images. The threshold should correspond proportionally to the size of the image set (e.g. 250 for fewer images, 350 for more). A low threshold would not throw away images of few features and a high threshold would throw away too many images.

3. convertImagesToObservations(dataset, threshold);

Upon completion, an obsX.mat (Matlab) file should appear in the parent directory, where X is the name of the folder.

***4.3.3 Computing an MST for dependence among words***

For FABMAP, we want to approximate better than a Naïve Bayes assumption for words, i.e. all words appear and disappear independently of one another. We form a tree out of the graph that characterizes all dependencies among words, so we can rely on the most dependent relationship shared by any two words. We compute the mutual information between any two words based on their co-occurrences, using the observation file generated from the last step. The output simply encodes the parent node of each word in the tree, which is also a word.

Use the C++ program *buildMST* and with the following command:

buildMST *obsX.mat*

The path should again be the directory where the code resides, and the obsX.mat file is generated from the previous step. Upon completion, an mstX.mat file should appear in the parent directory, where X is the name of the folder.

***4.3.4 FABMAP***

We are now ready to compute probability distributions over locations given all observation vectors, using recursive Bayes and iterating through all observations sequentially. By default, this program runs FABMAP 100 times and outputs a list of pairs of image numbers along with their counts. A count occurs when the probability of the two images forming a loop closure exceeds the input threshold.

Use the C++ program *fabmap* and with the following command:

fabmap *obsX.mat mstX.mat threshold*

The path should again be the directory where the code resides. The obsX.mat and mstX.mat files are generated from the previous step. A threshold of 0.999 is good for a set of ~100 images, but should be less if there are fewer images (e.g. 0.95 for ~50 images). Upon completion, a resultX.txt file should appear in the parent directory, where X is the name of the folder.

***4.4 Post-processing***

***4.4.1 Keypoint matching***

The list generated before contains genuine loop closures as well as false positives, so we need a post processing step that filters false candidates via keypoint matching. It calculates, for every feature of an image in a pair, the ratio of the feature’s distance with the closest neighboring feature to that with the 2nd closest in the other image. This works because the distribution for the value of this ratio differs between correct and incorrect matches, so a default threshold of 0.09 separates genuine loop closures from false positives (i.e. a genuine loop closure would have more than 9% of its features satisfying the ratio property).

Type the following at the Matlab Command Window:

1. dataset = ‘datasetDate\_datasetNumber’; //or whatever you named your folder to be
2. binaryFilter3(dataset); //binary in the sense that it is decisively a loop closure or not

Upon completion, an unsortedMatchesX.txt file will appear in the parent directory. (Note: a misnomer on unsorted, but a previous working version sorted image pairs in temporally sequential order, which is not as correct as this current one sorted by counts) This file will refer to features by their entire name, so a timestamp is retrievable.

***4.4.2 Same Scene Alignment***

Not surprisingly, since the subset is from the .texture file for IBR, there is nothing to guarantee that images forming a loop closure are as identical as they can be. It could be that one image captures a slightly shifted view of the scene from the other image. To achieve exact alignment, we can go back to the entire image dataset and, fixing one image, locate a neighbor for the other image so that this neighbor is the most closely matched to the fixed image. This requires a folder with every feature again, so repeat Section 4.2 and, to avoid confusion, name the folder something else, e.g. ‘20100501\_1e’ (‘e’ for entire).

Three inputs are supplied, one of which is a usual reference to the directory of interest. The range specifies how many neighbors are examined from both the left and the right (i.e. a range of 20 will look at images at most + 20 away). A large range would make the finding of an aligned image more likely, while a small range allows faster computation. Again, the threshold is to avoid comparisons of images of few features. Any image with feature count lower the threshold will not be compared and hence chosen.

Type the following at the Matlab Command Window:

1. dataset = ‘*datasetDate*\_*datasetNumber*e’; //or whatever you named your folder to be
2. range = 20; //the approximate difference in timestamps of neighboring images supplied by .texture file for IBR, guaranteeing the find of an aligned image
3. threshold = 300; //greater for larger set; smaller otherwise
4. findBestMatches(dataset, range, threshold);

Upon completion, a bmX.txt file will appear in the parent directory. It lists image pairs that are genuine loop closures and are as aligned as possible from the entire image set.

***4.5 Left-Right Camera Loop Closures***

***4.5.1 Introduction***

The previous sections detail all the procedures for detecting loop closures formed by images captured by one camera, i.e. the left camera. But loop closures can also be formed when one traverses a previous path in the opposite direction, with the same scenes captured by the other camera, i.e. the right camera in this case.

Many procedures remain the same since there are also feature files provided for the right camera. The following sections denote any differences from the previous procedures.

***4.5.2 Setting up***

Unfortunately, another folder will have to be created just for the features for the right camera for the procedure of choosing a subset. Simply make a folder, e.g. ‘20100504\_1r’ (‘r’ for right), put the .texture file inside, and run imageFilter2 as in Section 4.3.1. Now, you can either make a new folder for the entire set of both left and right features (copying them over), or copy the right ones into the original folder with the left ones, **the latter of which however would mean the output files for same camera loop closures will be overwritten.** So if the results from the left cameras are to be kept, make a new folder, e.g. ‘20100504\_1b’ (‘b’ for both). (Note: Same camera loop closures will still appear in the results even if right features are included, so overwriting results may be okay depending on what you want to do.)

***4.5.3 Procedure calls***

Here are just a few things to notice.

1. At the Matlab Command Window, the input dataset argument should now be the name of whatever folder you’re working with.
2. Now that we’re working with around twice as many images as before, a higher threshold can reduce the size and allow faster computations (e.g. 350 as opposed to 300). However, it is possible that an image forming a genuine loop closure would be thrown away, though unlikely.
3. For aligning images, now the entire folder will contain the entire sets of both left and right features.