ME 132 a: Lab 1

Shir Aharon
Tiffany Huang
Steven Okai

February 15, 2013

1 Part 1

1.1 Tutorial 0

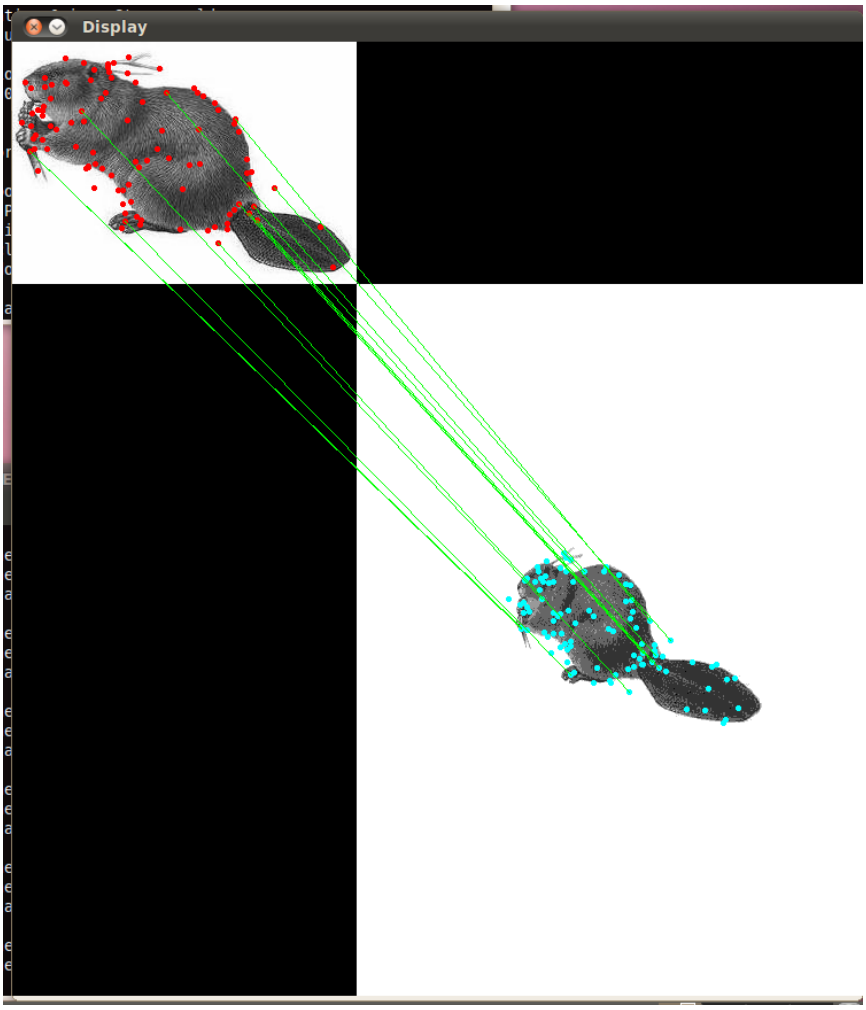
Code runs as expected. Speed and turnrate changed to 0.1.

1.2 Tutorial 1

The C++ code dumps the laser scan data into `data.txt`. This data is the robot pose and laser scan data which is in the form: `[x, y, yaw, range, bearing]` and can be plotted with the `me132_tutorial_1_plot.m` file by calling the function with the correct filename. Any points that have a range greater than or equal to 8 are ignored to minimize noise and false walls.

1.3 Tutorial 2

When we execute this program we obtain the below image. We can see that the features in both the test and reference image are matched with lines connecting the feature pairs.



1.4 Tutorial 3

We obtain results similar to the above, except that the two images update from a stereo camera and we can see features matched between them. Alternatively, a version named `me132_tutorial_3v2` is available which

fixes a reference image with a given file and compares it to the right image from the stereo camera. The output from these are saved in `right_stereo_3D.txt`.
The data is saved in the form `[x, y, z, r, g, b]` as .csv file to be plotted by `me132_tutorial_3_plot.m`.

2 Part 2

2.1 Run Time Notes

All driving code is in `ME132_Lab1.m`. The code is partitioned into cells so that later figures can be run independently once the earlier data is read in properly. The code run time is heavily dependent on the size of the input images. With the full size original example images the SIFT read in portion takes about 2.5 minutes while the histogram generation and feature matching takes approximately 45 minutes on all 3 example images. This is due to the cross image feature matching that requires very large arrays. Scaling the images down can significantly increase run time. Note: object image 2 and 12 had to be scaled down since `siftWin32.exe` runs out of memory if the images are too large.

2.2 Output files

All files are inside the `sift` folder. Images are labeled with the figure letter and a number to indicate the corresponding example or object image. The input object images have a copy in the `object_imgs - scale` folder which are reduced to 600 pixels across.
For the feature matching across two images, we plot the features in matching colors. For Figure F, we can see that the homography transform works by comparing the color regions for each feature.

2.3 Results

The output from a run on my computer. The homography matrix and the XY coordinates correspond to example images 1-3 in order.

```
Time to load images was 1.3258
Finding keypoints...
3814 keypoints found.
Finding keypoints...
8028 keypoints found.
Finding keypoints...
3463 keypoints found.
Finding keypoints...
679 keypoints found.
Finding keypoints...
6344 keypoints found.
Finding keypoints...
11258 keypoints found.
Finding keypoints...
18117 keypoints found.
Finding keypoints...
13159 keypoints found.
Finding keypoints...
8236 keypoints found.
Finding keypoints...
12894 keypoints found.
Finding keypoints...
6240 keypoints found.
Finding keypoints...
77262 keypoints found.
```

```
Finding keypoints...
916 keypoints found.
Finding keypoints...
1296 keypoints found.
Finding keypoints...
1101 keypoints found.
Time run sift was 154.6458
Time for figure A 5.7155
Time for figure B 1.2233
Time for figure C 2720.1311

H =  20.5     -6.7    -868.8
     10.1     12.6   -3931.7
      0.0     -0.0       1.0
rowcol = [114.8711, 219.6791]
xyz = [ 42.8741,  81.9976,   0.3733]

H =   6.7     -3.8     275.5
      5.1      3.1   -1033.0
      0.0      0.0       1.0
rowcol = [ 76.8476, 207.0157]
xyz = [ 39.1513, 105.5151,   0.5098]

H =  2.4     -5.5    1433.9
     6.3     -0.2    -795.6
     0.0     -0.0       1.0
rowcol = [134.6543, 321.5502]
xyz = [ 83.0928, 198.4456,   0.6172]
```