CALIFORNIA INSTITUTE OF TECHNOLOGY

ME/CS 132a, Winter 2011

Lab #1 Due: February 15th, 2011

Note: You are required to finish both Part 1 and Part 2 before coming to the lab session. The only difference is that they require implementation in different programming languages.

Part 1 – C/C++ Component [25 points total]

The first part is a series of exercises to familiarize yourself with remote development in simulation, testing your software on the actual hardware, and basic sensor acquisition. Before you start, go to the Lab 1 webpage[1] to learn:

- How to use Player/Stage remotely on the course server;
- How to compile the client code (see the webpage for its definition);
- How to use Player on the actual robot.

We will also use the words "binary" and "executable" interchangeably.

I. Learning the Player/Stage environment [5 points]

- Source code location: `me132_tutorial_player/me132_tutorial_0.cc`

Here you will learn to control the robot using Player/Stage.

1. Remotely connect to the course server. Compile the test programs, and run `me132_tutorial_0`. You should see the robot moving in a circle in the Stage simulator.
2. (Lab session only) Once you have successfully tested your program in simulation, you will demonstrate to the TAs that the binary also works on the real robot. Before demonstrating, change the speed to be 0.1 m/s and the turnrate to be 0.1 rad/s.

II. Acquiring Laser Sensory Data [10 points]

- Source code location: `me132_tutorial_player/me132_tutorial_1.cc`

Here you will learn to acquire 2D laser scanner data using Player/Stage.

1. Run `me132_tutorial_1` to watch how the laser scanner behaves in Stage. Modify the source code `me132_tutorial_1.cc` to save one laser scan to a text file and plot it in MATLAB.
2. (Lab session only) Once you have successfully tested your program in simulation, you will demonstrate to the TAs that your binary works on the real laser scanner. Before demonstrating, change the speed to be 0.1 m/s and the turnrate to be 0 rad/s.

[1] `http://www.cds.caltech.edu/~murray/wiki/index.php/ME/CS_132a,_Winter_2010,_Lab_1`

III. Generating/Displaying SIFT Features [5 points]

- Source code location: `me132_tutorial_camera/me132_tutorial_2.cc`

Here you will learn to:

- Use the OpenCV library to load and display a saved reference image and a saved test image;
- Apply SIFT feature extraction using SIFT library on the reference image to generate a database of features and save these features to file;
- Apply SIFT feature extraction using SIFT library on the test image and match those features against the database;
- Use OpenCV to draw features and matches on the images.

Instructions:

1. Run `me132_tutorial_2` and report the results. Press any key to quit the problem.
2. This program does not particularly require any demonstration of functionality to the TAs during your lab session, but is necessary for the next task.

IV. Acquiring Stereo Image Data [5 points]

- Source code location: `me132_tutorial_camera/me132_tutorial_3.cc`

Here you will learn to:

- Use the provided Bumblebee camera driver to acquire images from the left and right camera in the stereo pair;
- Display both the left and right images using the OpenCV library;
- Draw extracted live SIFT features on the right camera image.

Instructions:

1. Note that you will only be able to compile this program remotely. There is no simulator in place to simulate acquired camera data so testing of it will have to be done during your lab time and demonstrated to the TAs.
2. (Lab session only) Demonstrate the program on a real Bumblebee stereo camera.

Part 2 – MATLAB Component [85 points total]

I. RANSAC/homography [45 points total]

Note: There is no lab session corresponding to this, but it will be used as a key component in the next task required during the lab session.

Design and implement a RANSAC algorithm to detect and remove outliers in a list of paired feature points generated from feature detection and tracking. You will test your algorithm on two lists from the *Mars Hill* image sequence. The first feature list (`image1_2.txt`) is from image 1 and 2 and the second list (`image5_6.txt`) is from image 5 and 6. Each row in the dataset includes a pair of corresponding points in the two images:

$$\left[\begin{array}{cccc} x & y & x' & y' \end{array}\right].$$

1. [10 points] Assume the desired probability of success is $p_{\text{suc}} = 99.9\%$, and the probability of a sample being an outlier is $p_{\text{out}} = 0.5$, independently from other samples. Suppose we need at least k samples for each trial in RANSAC. Compute the required minimum number of RANSAC trials for $k = 3$, 4, and 5.

2. [10 points] We know from the lecture that for any point on a given planar surface, its two corresponding image points, $p = (x, y)$ and $p' = (x', y')$, taken at different camera poses, are related by the same homography transformation $p' = H(p)$, defined as

$$x' = \frac{h_1 x + h_2 y + h_3}{h_7 x + h_8 y + 1}, \quad y' = \frac{h_4 x + h_5 y + h_6}{h_7 x + h_8 y + 1}.$$

   Determine the minimum number (call this k) of pairs of image points (x_i, y_i, x_i', y_i') $(i = 1, 2, \cdots, k)$ required to unambiguously define a homography transformation, and write the linear equations that solve the h_i's, in the form of:

$$Ah = b, \quad h = \left[\begin{array}{cccc} h_1 & h_2 & \cdots & h_8 \end{array}\right]^T.$$

   where the matrix A and/or vector b should depend on the image point coordinates.

3. [25 points] Using $p_{\text{suc}} = 99.9\%$, $p_{\text{out}} = 0.5$, run RANSAC to obtain the best homography transformation H for both datasets. For each RANSAC trial, use the linear equations obtained in part (b) to estimate H (i.e. the h_i's) from linear least-squares:

$$h = \arg\min_h \|Ah - b\|.$$

   Treat a feature point as an outlier if its reprojection error ϵ is greater than some threshold ϵ_{thr} (in pixels). The reprojection error between a pair of corresponding feature points, $p = (x, y)$ and $p' = (x', y')$, is defined by the Euclidean distance:

$$\epsilon \triangleq \|p' - H(p)\|.$$

   For each dataset:

   - Report the homography transformation H found and the percentage of inliers.
     - Compute the mean and the standard deviation of the reprojection errors of all the valid feature points (i.e. discard all the outliers detected by RANSAC). You should be able to obtain an error mean less than 1 pixel. Otherwise, try to increase the error rejection threshold ϵ_{thr} in your RANSAC implementation. Choose the same ϵ_{thr} for both datasets.
     - Plot the feature points from the two images in the same figure and draw lines between the corresponding feature points. Use different colors for valid correspondences (inliers) and invalid correspondences (outliers) as determined by RANSAC. Your plot should look like the one in Slide 21 from Lecture 6, but with the lines properly color coded.

For the ϵ_{thr} you choose, which dataset has a higher percentage of inliers?

4. [Extra credit] Implement another RANSAC outlier detection and removal algorithm that solves h directly from the nonlinear homography projection during each trial:

$$h = \arg\min_h \sum_{i=1}^{k} \|H(p_i; h) - p'_i\|^2.$$

Compare the results with those in part (c). You may use the Matlab function `lsqnonlin` for solving nonlinear least-squares. (Hint: use the linear solution from part (b) as an initial guess for h.)

II. Object recognition [40 points total]

Your task is to implement a program that finds known objects in a sensor image. You are given a *reference object database* containing the images of several objects. The *sensor data* comes from a stereo pair, and contains both RGB and XYZ data. Given the sensor data, you have to find which reference objects are present in the image, and their position. The required output is: the identity of the objects present in the sensor image; its position in the image as `(row,col)` pixel coordinates; the homography transformation between sensor and reference images; plus a series of plots describing intermediate results, as discussed below.

Input files

- **Reference object database.** A set of reference images, each containing a picture of an object.

```
<object_id1>.jpg,  <object_id2>.jpg, ...
```

These images constitute your object database. You can load them in Matlab like using:

```
ob1_rgb = imread(<object_id1>.jpg)
```

- **Sensor data.** A set of "sensor data" files, containing RGB and XYZ images.

```
<example1>.mat, <example2>.mat, ...
```

Load it in Matlab like using:

```
sensor_data = load(<example1>.mat)
```

The data has the following fields:

sensor_data.rgb RGB values (shape HxWx3)
sensor_data.XYZ points coordinates in the camera frame (shape HxWx3)

The example data also have a ground truth information which is the expected output of your algorithm:

sensor_data.object(i).id Names of the objects present in the image.
sensor_data.object(i).position image coordinates of upper left corner of detected object

This extra information will not be present in the data for the lab session.

Implementation guidelines.

- While you are given multiple sensor data files as examples for testing, the input to your program is only one of those at a time, plus the entire collection of reference object images.
- You are given a function *sift_ extractor()* that runs the SIFT keypoint detector on one image (see *sift_ extractor_ test.m* for an example use.) It extracts a list of SIFT features from an RGB image. For each feature, it reports: position, scale, orientation, and the SIFT descriptor (a vector of 128 numbers).

  Load each reference object image, and use *sift_ extractor()* to extract its features.

  Figure A: For each object, plot the position of the detected features on the reference image, as well as scale and orientation. You can represent orientation with an arrow, and the scale by changing the arrow size.
- Load the sensor data, and use *sift_ extractor()* to extract its features.

  Figure B: Plot position, scale, orientation of the extracted SIFT features on the image.
- Find the object in the database that has the most matches between its features and the features in the image.

  You can do this by matching each feature in the image with all features in the object database. You can define the similarity between two features using the sum of square differences between the SIFT descriptors vectors. As in the previous homework, find the best match in a robust way using the ratio threshold heuristic (comparing the ratio of the similarity of first and second best match). The number 0.6 is a reasonable value for the threshold ratio.

  If a feature in the image matches against a feature in the image of reference object x, it counts as a vote for object x.

  Figure C: Plot an histogram of the number of matches for each reference object.
- An object with "enough" votes is considered "detected".
- For each detected object:
  - **Figure D**: Show the correspondence between the features in the sensor image and the features of the detected object. You can do this by either creating a composite image of sensor image and reference image, and connecting the features with lines; or saving two distinct images but labeling the corresponding features with the same label.
  - Once the object identity has been found, your goal is to find the homography transformation between the reference image for the object and the sensor image. To find the homography, it is better to have as many matches as possible to feed to the RANSAC algorithm. Match again the features in the sensor data with the features for the object (note that in the previous step you matched the sensor data features with all features for all objects in the database).

    Figure E: Show again the correspondence between features. You should have at least the same features as the previous steps. (For such a small object database, it is likely you will not get much more features)
  - Given the correspondences, use the algorithm you developed for the previous exercise to find the homography transformation. Find the pixel coordinates in the sensor image of the upper left corner of the reference object image (i.e., pixel (0,0)) by applying the

homography transformation. Find the world XYZ coordinates of that point by using the XYZ in the sensor data. Return the (row, col) and(X, Y, Z) coordinates as the output of your algorithm.

Figure F: Plot the features in the reference object image reprojected in the sensor image through the homography transformation.

Expected output

- **Fig. A** for each reference object image
- For each of the K sensor data files, provide:
  - **Fig. B**, **Fig. C** ($5K$ plots total)
  - For each detected object:
    - **Fig. D**, **Fig. E**, **Fig. F** ($3DK$ plots total assuming that there are D detected objects in each image)
    - The parameters describing the homography transformation.
    - The (row, col) and (X, Y, Z) coordinates of the pixel (0,0) in the reference image.

Checklist of things to finish before coming to the lab session

- C/C++ binaries: `me132_tutorial_0` to `me132_tutorial_3`. Make sure they can be run on the remote server. Understand how to execute them on the robot.
- MATLAB implementation of the object recognition algorithm in Part 2 (II). We will evaluate your algorithm by giving you new data generated on-site during the lab session.

Special guidelines for labs

- People in the group share the same source code, and prepare one lab report as a whole.
- All people in the group should be present during the lab session. Due to logistic issues, we cannot accommodate variations with respect to the schedule. People can be excused with a note from the dean or the health center; in that case, we will work something out.
- Add the following statement at the end of your solution: "*All persons in this group participated equally in preparing this set, and share the same level of understanding for all exercises*"; or amend the sentence so that it reflects a fair assessment of your situation.
- The general guidelines below still apply, but the "group" as a whole is given juristic personality and counts as the same "person". Therefore: there can be discussion but no sharing of code between groups.

General guidelines for problem sets involving software

- Send all the software required via email to the TA. It is most helpful if you package code, data, and answers in a .zip or .tgz file.

- We expect your solutions to be in the spirit of “reproducible research”[2]. Include instructions/scripts that allow reproducing your experiments with relatively little effort. For example, include a script “main.m” that calls the other files.
- Your software is evaluated on clarity and elegance as well as correctness. Comment your code. Insert a comment for everything that is not immediately obvious; and do *not* comment what *is* obvious. Think whether you would still understand the code you write after locking it up in a drawer for a year.
- You will be given code examples in a few languages (Matlab, C++, Python), but you are free to use any language with which you are comfortable.
- You are encouraged to use professional libraries (such as OpenCV) for reading/writing files and analogous tasks. However, you cannot use functions which the homework implies you have to write yourself (e.g., you cannot use the Harris detector in OpenCV).
- You are not required to follow the implementation guidelines perfectly. If you know a better alternative for a certain step, it is OK to use it.
- You are responsible for the parameters you choose. If we give you a “reasonable” value for a parameter that does not appear to work, you should try other values.
- You cannot share code for homework or look at other people’s code. You are free to discuss general ideas about the problem. Reading aloud your code does not count as discussion.
- **(new)** The only way to get software right is to insert consistency checks that make sure that your assumptions during the computations are correct.

  [CS majors]: Numerical code is a perfect occasion to learn and start to use *unit tests*[3].
- **(new)**Read Dijkstra’s words from his 1972 ACM Turing Lecture[4]:

  > *The competent programmer is fully aware of the strictly limited size of his own skull; therefore he approaches the programming task in full humility, and among other things he avoids clever tricks like the plague.*

  Be aware of the limited size of your skull, and split the code in smaller functions that are easier to get right.

  [CS majors]: Print out a picture of Dijkstra and hang it in your room; meditate daily on one of his EWDs[5]; when in doubt, ask yourself: *what would Dijkstra do?*
- **(new)**Remember: in elegant code, bugs have no space to hide.

[2] `http://reproducibleresearch.net/`
[3] `http://en.wikipedia.org/wiki/Unit_testing`
[4] `http://www.cs.utexas.edu/~EWD/ewd03xx/EWD340.PDF`
[5] `http://www.cs.utexas.edu/~EWD/`