Homework 4

Due Monday, March 4, 2013 (by 8:00 pm)

Notes: Please email me your solutions for these problems (in order) as a single Word or PDF document. If you do a problem on paper by hand, please scan it in and paste it into the document (although I would prefer it typed!).

1. (25 pts) A set of 2D points $\mathbf{P}_A = \{\mathbf{p}_A^{(1)},\ldots,\mathbf{p}_A^{(N)}\} = \{(x_A^{(1)}, y_A^{(1)}), \ldots, (x_A^{(N)}, y_A^{(N)})\}$ is extracted from image A. Corresponding points $\mathbf{P}_B = \{\mathbf{p}_B^{(1)},\ldots,\mathbf{p}_B^{(N)}\}$ are extracted from image B. You want to find a scaled similarity transform[1] that aligns the two, using a least squares fit. The transformation can be written as a nonlinear function $\mathbf{P}_B = \mathbf{f}(\mathbf{P}_A, \mathbf{x})$, where $\mathbf{x} = (s, \theta, t_x, t_y)^T$ is the vector of unknown transformation parameters.
   a. Write the symbolic form of the Jacobian matrix of **f**.
   b. Write code to solve for the transformation, using the point correspondences given below (for an initial guess, use *s*=1, and zeros for the other parameters). What is the final residual error (in terms of the sum of the squared errors)?

   Image points from image A (x;y):

   126 34 170 103
   60 107 77 163

   Corresponding points from image B (x;y):

   173 95 196 121
   49 47 77 107

2. (25 pts) Your task in this problem is to do some experiments with the SIFT code developed in class. This code extracted SIFT features from two images, matched the features, and found a set of correspondences that had the same scale, rotation, and location. The code can be used to recognize an object in a cluttered scene. Download the image dataset from the website http://www.computing.dundee.ac.uk/staff/jessehoey/teaching/vision/project1.html.

   a. Take the object images "book1.pgm" and "book2.pgm". Look at each of the images named "Img0[i].pgm", where [i]=1...10., and determine whether each object is present in the scene (note: each object is present in 5 of the images). Now use the SIFT code to match each object to each image, and determine the number of points that were matched.

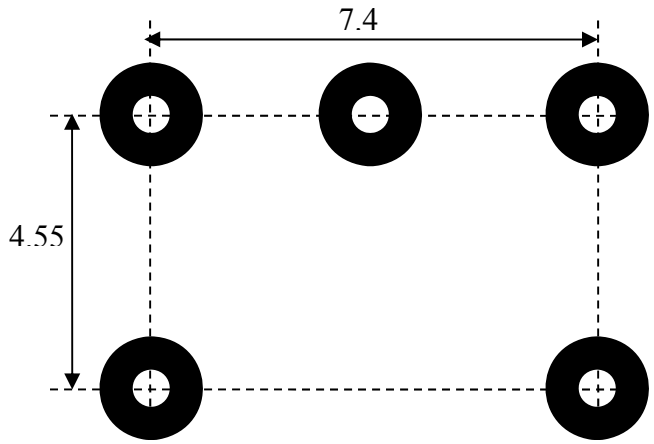
   b. Using your results from above, decide upon a threshold on the number of feature matches that can classify whether the object is present in an image. Ideally, if you get that number of matches or more, the object is actually present in the image;

[1] See lecture notes on 2D-2D image transforms, slide 6.

otherwise, if you get fewer matches, the object is actually not present. You may not be able to find a threshold that gives perfect classification results, but choose the value that the minimizes the number of misses (number of images that contained the object that were classified as not containing the object) and the number of false positives (number of images that do not contain the object that were classified as containing the object).

c. Now match the two objects to the test images named "TestImg0[i].pgm", where [i]=1...10, and use the same threshold to indicate the presence of the object. Compute the number of misses and the number of false positives.

3. (25 pts) The objective of this exercise is to compute the pose of a known object with respect to a camera, from a set of correspondences between object points and observed image points. In Lecture 16 on "pose estimation", I gave an example of how to compute the pose using nonlinear least squares. Using that method, find the pose of the 5-point concentric circle target with respect to the camera. Use the images "robot1_rect.jpg", "robot2_rect.jpg", and "robot3_rect.jpg" on the course website[2]. The measurements of the model (in inches) are indicated in the figure. The top middle target feature is midway between the top left and the top right feature. Use the top left feature as the origin of the target's coordinate system, with its x-axis pointing to the right, its y-axis pointing down, and its z-axis pointing into the page.



(a) For each image, give the pose of the target with respect to the camera (in terms of XYZ angles and XYZ translation) in each image.
(b) Use that pose to draw the XYZ coordinate axes of the target as an overlay on each image.

[2] Note – these images have been "rectified" to remove lens distortion. We will cover how to do this a little later in the course. We will also cover how to compute the camera intrinsic parameters

Note:

- Your program should automatically detect and identify the CCC targets in the images, using the methods developed in HW2.
- If you need to, it is ok to use a different initial guess for the pose in each image, in order to get your algorithm to converge to the correct solution (you will know that the solution is correct because the graphical overlays will look right). The model is about 95 inches from the camera in the first image.
- Use the following values for the camera intrinsic parameters: focal length = 2443 (in pixels), image center (x,y in pixels) = (1124, 831).

4. (25 pts) In the previous problem, you computed the Jacobian matrix numerically. It is also possible to compute the Jacobian symbolically (*i.e.*, by doing the derivatives by hand), although the expressions can be rather complicated. However, under certain conditions the Jacobian is easier to compute symbolically. In particular, if the rotation angles are close to zero[3], the rotation matrix can be simplified using the "small angle approximation", which is $\cos(\theta) \approx 1$, $\sin(\theta) \approx \theta$, and $\theta * \theta \approx 0$.

   (a) Write the 3x3 rotation matrix $\mathbf{R}(\theta_X, \theta_Y, \theta_Z)$, using the small angle approximation.

   (b) Write the equations for the function that projects a point $\mathbf{P} = (X, Y, Z)^T$ onto an image using both weak perspective and the small angle approximation.

   (c) Finally, write the equations for the Jacobian of that function with respect to the unknown parameters $\theta_X, \theta_Y, \theta_Z, t_x, t_y, Z_{avg}$; i.e., find the expressions in the matrix

$$\mathbf{J} = \begin{pmatrix} \partial x/\partial\theta_X & \partial x/\partial\theta_Y & \partial x/\partial\theta_Z & \partial x/\partial t_X & \partial x/\partial t_Y & \partial x/\partial Z_{avg} \\ \partial y/\partial\theta_X & \partial y/\partial\theta_Y & \partial y/\partial\theta_Z & \partial y/\partial t_X & \partial y/\partial t_Y & \partial y/\partial Z_{avg} \end{pmatrix}$$

[3] For example, if you are tracking an object and are just computing the correction to its estimated pose.