CS534: Introduction to Computer Vision
Camera Calibration and Augmented Reality

Spring 2014: Assignment 3

Instructor: Ahmed Elgammal, Dept of Computer Science, Rutgers University

April 9, 2014

- Due April 20th, 2013, midnight
- Instructions: turn on matlab diary; or do your work in a cell m file and use Matlab publish functionality to show your steps and results.
- Files for this assignments can be downloaded from: ftp://ftp.cs.rutgers.edu/pub/elgammal/CS534/HW3/
- As usual, submit your work online using Sakai by the due date.
- Late Submission: 10% late penalty for every day after the due date.

Part I: Camera Calibration using 3D calibration object (50 points)

We want to calibrate the camera of a robot vehicle. We will use a linear method as described in the lectures. We place a large cubic frame of size 4 meters on the road several meters in front of the vehicle. The positions of the eight corners of the cubic frame are defined with respect to a world coordinate system with its axes parallel to the cube edges and with its origin at the center of the cube. The world coordinates of the cube vertices are:

2 2 2,
-2 2 2,
-2 2 -2,
2 2 -2,
2 -2 2,
-2 -2 2,
-2 -2 -2,
2 -2 -2

We detect the corresponding cube corners at the following pixel positions in the camera image:
422 323; 178 323; 118 483; 482 483; 438 73; 162 73; 78 117; 522 117

1. Draw the image points, using small circles for each image point.
2. Write a Matlab function that takes as argument the homogeneous coordinates of one cube corner and the homogeneous coordinates of its image, and returns 2 rows of the matrix $\mathcal{P}$ (slide 30 of the Camera Calibration pdf document). This matrix $\mathcal{P}$ will be used to compute the 12 elements of the projection matrix $\mathbf{M}$ such that $\lambda p_i = \mathbf{M} P_i$
3. Use this Matlab function to generate 2 rows of the matrix $\mathcal{P}$ for each cube corner and its image and obtain a matrix with 16 rows and 12 columns. Print matrix $\mathcal{P}$.

4. Now we need to solve the system $\mathcal{P}m = 0$. Find the singular value decomposition of matrix $\mathcal{P}$ using matlab svd function. The last column vector of V obtained by svd($\mathcal{P}$) should be the 12 elements in row order of the projection matrix that transformed the cube corner coordinates into their images. Print the matrix **M**.

5. Now we need to recover the translation vector which is a null vector of **M**. Find the singular value decomposition of matrix $\mathbf{M} = U\Sigma V^T$. The 4 elements of the last column of V are the homogeneous coordinates of the position of the camera center of projection in the frame of reference of the cube (as in slide 36). Print the corresponding 3 Euclidean coordinates of the camera center in the frame of reference of the cube.

6. Consider the 3x3 matrix **M**' composed of the first 3 columns of matrix **M**. Rescale the elements of this matrix so that its element m_{33} becomes equal to 1. Print matrix **M**' . Now let the rotation matrices be as defined in slide 38 where the axes e1, e2, e3 are the x, y, z axes respectively. Therefore **M**' can be written as $\mathbf{M}' = KR_z^T R_y^T R_x^T$

7. We will perform the RQ factorization of **M**' in several steps. First, find a rotation matrix R_x that sets the term at position (3,2) to zero when R_x is multiplied to **M**'. The cosine and sine used in this matrix are of the form

$$cos(\theta_x) = m_{33}/sqrt(m_{33}^2 + m_{32}^2)$$
$$sin(\theta_x) = -m_{32}/sqrt(m_{33}^2 + m_{32}^2)$$

   Note that the term at position (3, 2) would also be set to zero if the signs of $cos(\theta_x)$ and $sin(\theta_x)$ were reversed, but this would lead to finding a negative focal length for the camera. So we should choose the signs that leads to a positive focal length. Compute the angle θ_x of this rotation in degrees. Compute matrix $\mathbf{N} = \mathbf{M}' * R_x$. Print R_x, θ_x and **N**.

8. The element n_{31} of **N** is small enough so that there is no need for a rotation R_y. However, element n_{21} is large and a rotation matrix R_z is needed to set it to zero. Compute the rotation matrix R_z using cosine and sine of the form

$$cos(\theta_z) = n_{22}/sqrt(n_{21}^2 + n_{22}^2)$$
$$sin(\theta_z) = -n_{21}/sqrt(n_{21}^2 + n_{22}^2)$$

   Compute the rotation angle θ_z in degrees. This angle is actually very small.

9. Since we factorized out R_z we can directly compute the calibration matrix K, how? Compute K and rescale so that its element K_{33} is set to 1. Print K. What are the focal lengths of the camera in pixels? What are the pixel coordinates of the image center of the camera?

Part II: Camera Calibration using 2D calibration object (70 points)

In this part we are going to implement camera calibration from multiple images of 2D planes. Additionally we will learn how to augment images with virtual objects. We will follow the method described in the book chapter on camera calibration by Zhengyou Zhang which was proposed in his paper "Flexible Camera Calibration by Viewing a Plane from Unknown Orientations - Zhang, ICCV99". Start by carefully reading Section 2.4 of that Chapter.

Calibration Grid and images

We know the following about the grid. The grid is 9 squares in width and 7 in height. Each square is 30mm x 30mm. If we select the bottom left corner of the grid to be the origin of the world coordinate system, and the grid to be the plane corresponding to Z=0 , then we know the 3D coordinates of each corner in that grid.

For calibration we will use four images: images2.png, images9.png, images12.png, images20.png

Corner Extraction and Homography computation (10 points)

First we want to extract the four corners of the calibration grid form each image. We will use the grid corners to estimate the homographies relating two images. We can manually get the four grid corners from each image. One way to let a user manually select points in matlab is using `ginput` function.

Once the 4 corners are extracted, compute the homography H that relates the grid 3d coordinates to the corners. Use the function `homography2d` which is provided. Repeat this for the four images provided. Report the computed H for each image [**deliverable**].

Computing the Intrinsic and Extrinsic parameters (30 points)

Now given the four homographies, follow the instructions in section 2.4.4 to compute the intrinsic parameters and extrinsic parameters. We need to linearize the two constraints in Eq 2.19 and 2.20 into two equations in a homogeneous system as in Eq 2.25. Then solve for b and estimate the intrinsic parameters as described in page 21. Print the computed matrix $\mathbf{B}$ and the intrinsic parameters. Then compute and print $\mathbf{R}, t$ for each image [**deliverable**].

Verify that your rotation matrix is in fact a rotation matrix, print $R^T R$, is it an identity as it should be? We can enforce $\mathbf{R}$ to be a rotation matrix by SVD decomposition of $\mathbf{R}$ and setting the singular values to ones, i.e., set the rotation matrix to UV^T where $R = U\Sigma V^T$.

Print the new R and $R^T R$ after enforcing the rotation matrix constraints. [**deliverable**]

Improving accuracy (30 points)

Since we used four manually entered points to compute the homography. A small error in one of the points will directly effect the computed homography. To fix this we are going to estimate the homography from all grid points.

- First given the computed homographies from Section 2, compute the approximate location of each grid corner in the image. (Hint : This can be done since we know the 3d locations of the grid corners and the approximate homography. Call these points p_approx. Create a figure with the image and approximate grid locations. Call this "Figure 1 : Projected grid corners " [**deliverable**]
- Second, using the provided Harris function detect Harris corners in the image and display them. Use the following parameter values for the Harris detection : sigma = 2, thresh = 500, radius = 2.
  $[\texttt{cim}, \texttt{r}, \texttt{c}, \texttt{rsubp}, \texttt{csubp}] = \texttt{harris}(\texttt{rgb2gray}(\texttt{im}), \texttt{sigma}, \texttt{thresh}, \texttt{radius}, \texttt{disp})$;
  Here r is the y-coordinate of the Harris corner, c is the x-coordinate of the Harris corner, rsubp is the y coordinate with subpixel accuracy, csubp is the x coordinate with subpixel accuracy. Use rsubp, csubp. Create a figure with image and overlayed Harris corners. Call this "Figure 2 : Harris corners ". [**deliverable**]
- Third, compute the closest Harris corner to each approximate grid corner. (You may find it useful to use the dist2.m function provided). Let these closest Harris corners be p_correct. Create a figure with the image and p_correct overlayed. Call this "Figure 3 : grid points ". [**deliverable**]
- Finally, compute a new homography from p_correct, print H [**deliverable**]
- Repeat this for the other three images. Then use the homographies to estimate $\mathbf{K}$ and $\mathbf{R}$, t for each image. Report your $\mathbf{K}$, $\mathbf{R}$'s, and t's [**deliverable**]. Save your results, you will need to use them in Part III
- Using the new computed H, compute the errors between points in p_correct and points you get by projecting grid corners to the image (Hint there is no need to use R, t for projecting) . Call this err_reprojection. Report your result. [**deliverable**]
- Now repeat the process using 4 images. Compare your results to your previous results and those of part 2 [**deliverable**].
- Can you suggest a way this can be done automatically (i.e without first letting the user manually select the 4 corners) ?

Part III: Augmented Reality 101 (50 points)

Augmenting an Image (30 points)

Now we would like to use our computed homographies from part II to map a clip art image onto the grid such that it seems to be part of the grid. The image should be synthesized such that the clip art bottom left corners is the same as the grids (0,0) corner. When fitting the clip art you should rescale it to fit the grid while keeping the clip art aspect ratio. Using your computed homography find a way to map your image on the grid such that you image will have the same projective distortion as the grid. If the clip art have any white pixels you should make these pixels appears transparent when overlayed over the grid. For each image of the four images in Part II, create a figure with the original image and your virtual clip art overlayed over the grid. Your image should be one of the images provided in the clipart directory. To find which clip art you are supposed to map, take the last 4 digits of your ruid id add them up and use the clip art file corresponding to the first digit in the sum, i.e., if your the first 4 digits in your RUID are 5243, use the clip art 4.xxx .

Augmenting an Object (20 points)

Now we would like to augment our images with 3D objects. For our purposes we are going to use a cube as a virtual object. We will only render the cube as a wire frame and we would like its base to be locate on the 3x3 grid of squares in the bottom left corner of our grid. The cube should be standing up from the grid. First print the 3D coordinates of the cube. Then, find a way to use your computed $\mathbf{H}$, $\mathbf{K}$, $\mathbf{R}$, t to synthesize new images with the virtual cube inserted.

Extra credit (20 points)

Do one of the following :

1. Instead of augmenting a cube, augment a general mesh from a 3D file of your choice.
2. Can you find a way to estimate the intrinsic and extrinsic parameters from only two images of the grid. What assumptions on the intrinsic parameters are needed to achieve this. (Hint the answer can be found in Sec 2.4)