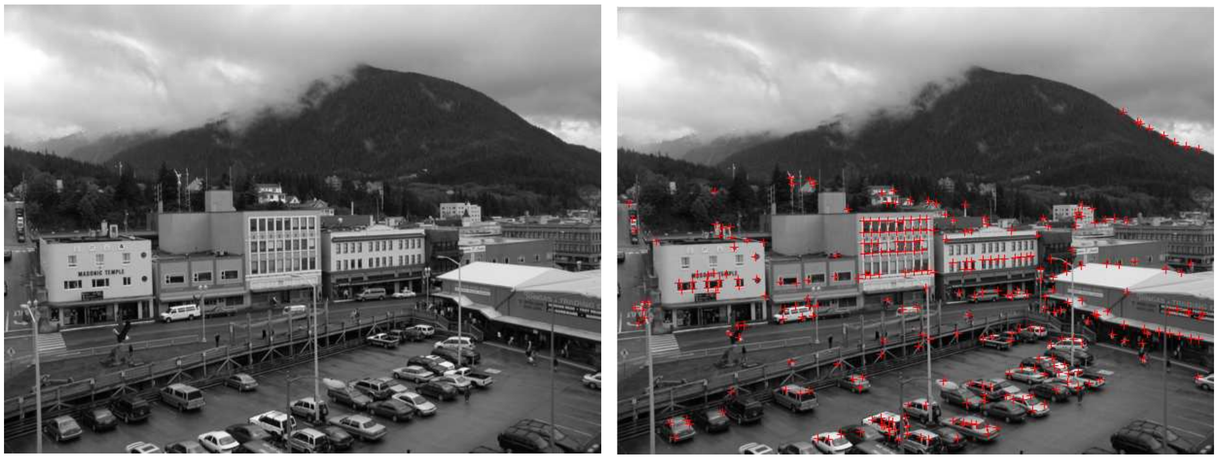
CS223B Introduction to Computer Vision, Winter 2007
Please email your answers to cs223b@gmail.com.

Assignment #2

1 Playing with the Harris Corner Detector

We have provided a Matlab implementation of the Harris corner detector which you can download from `http://cs223b.stanford.edu/homework/ass2/1/harris_corners.m`. This includes comments on the input parameters and how to run the detector. Your task is to run this corner detector on the images provided in `http://cs223b.stanford.edu/homework/ass2/1/`, visualize the results and answer some questions below. An example or the output of a corner detector is shown below.



(a) Input image (b) Detected corners indicated by red crosses

Figure 1: Example of Harris corner detection.

Here is a set of commands in Matlab demonstrating its use:

```
>> img = imread('lincoln.png');
>> corners = harris_corners(img, 7, 1.5);
>> imshow(img);
>> hold on
>> plot(corners(:,1), corners(:,2), 'r+')
```

This reads the image from file lincoln.png, runs the corner detector using a gaussian kernel of width 7 pixels and standard deviation 1.5. The last three commands display the image and the detected corners superimposed on it. Run the corner detector on the image `grid-1.png` and try to tweak the parameters so that all (or as many as you can manage) corners of the black squares are detected and the number of spurious corners detected is minimized.

Questions:

1. For fixed parameter values, run the detector on `grid-1.png` and `grid-rotated.png`. The latter image is a rotated copy of the former. Is this corner detector rotation-invariant ? That is, if we rotate the input image, do the detected corner positions rotate by the same amount ? Justify your answer based on your observations.

2. Is the corner detector scale invariant ? That is, if we scale down the input image, are all the detected corner positions scaled accordingly ? You can test this experimentally by comparing the corner detection on the images `grid-1.png, grid-2.png, grid-3.png`. Each image in this sequence is half the size of its predecessor. Justify your answer based on your observations.

Submit your answers to this question in a text file named Readme.txt. Include the parameters values you used.

Note: You are welcome to use any other implementation of the Harris corner detector you find online, in OpenCV, or modify the one we have provided if you think you can improve its performance. Just be sure to test it on the images we have provided and let us know in your submission file Readme.txt which detector you used.

2 Calibration of an Affine Camera

It was shown in class that a perspective camera can be modelled using a 3×4 matrix:

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

which means that the image of point (X, Y, Z) in the scene has pixel coordinates $(x/w,\ y/w)$. The 3×4 matrix can be factorized into intrinsic and extrinsic parameters.

An *affine* camera is a special case of this model in which rays joining a point in the scene to its projection on the image plane are parallel. Examples of affine cameras include orthographic projection and weakly perspective projection. An affine camera can be modelled as:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

which gives the relation between a scene point (X, Y, Z) and its image (x, y). The difference is that the bottom row of the matrix is $[0\ 0\ 0\ 1]$, so there are fewer parameters we need to calibrate. More importantly, there is no division required (the homogeneous coordinate is 1) which means this is a *linear model.* This makes the affine model much simpler to work with mathematically - at the cost of losing some accuracy. The affine model is used as an approximation of the perspective model when the loss of accuracy can be tolerated, or to reduce the number of parameters being modelled.

Calibration of an affine camera involves estimating the 8 unknown entries of the matrix in Eq. (2). (This can also be factorized into intrinsics and extrinsics, but that is outside the scope of this homework). This is done by having the camera observe a calibration pattern with easy-to-detect corners.

2.1 Coordinate system

The calibration pattern used is shown in Fig. 2, which has a 6×6 grid of squares printed on a flat base. Each square is $50mm \times 50mm$. The separation between adjacent squares is $30mm$, so the entire grid is $450mm \times 450mm$. For calibration, images of the pattern at two different positions were captured. These are shown in Fig. 2 and can be downloaded from `http://cs223b.stanford.edu/homework/ass2/2/`. For the second image, the calibration pattern was been moved back (along its normal) from the first position by 150mm.

We will choose the origin of our 3D coordinate system to be the top left corner of the calibration pattern in the first position. The X-axis runs left to right parallel to the rows of squares. The Y-axis runs top to bottom parallel to the columns of squares. We will work in units of millimeters. All the square corners from the first position have Z-coordinate 0. The second position of the calibration grid corresponds to Z=150.

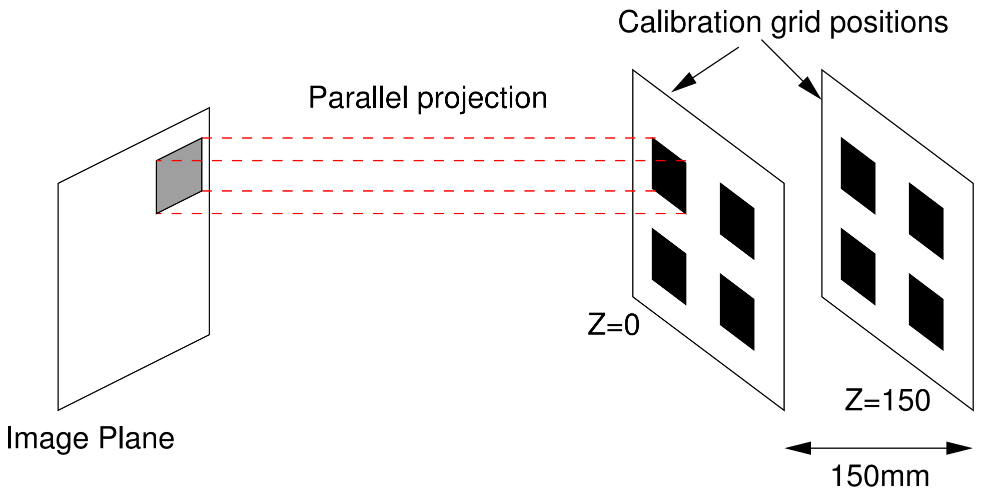
2.2 Corner Extraction

We begin by extracting the feature correspondences required for calibration. Find the pixel coordinates of the square corners in the calibration grid in wach of the two images. You may use the Harris corner detector we provided (plus manual input) or any 3rd-party software for extracting corners from the calibration grid images. For Linux, a calibration grid finding program is available from `http://graphics.stanford.edu/software/findgrid/`. This is known to run on the vine.stanford.edu cluster, but has not been tested on other machines or distributions.

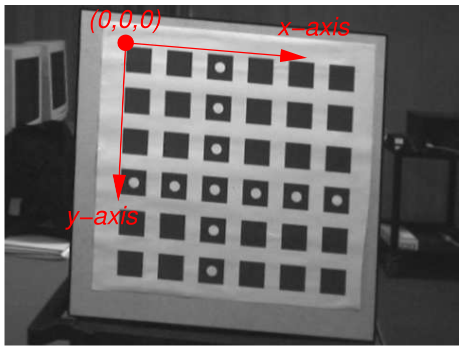
Each corner position (x_i, y_i) is to be associated with the corresponding 3D point coordinates (X_i, Y_i, Z_i). Thus, you should generate an $N \times 5$ matrix M where each row contains the 2D pixel coordinates of a detected corner and the 3D coordinates of the corresponding point:

$$M = \begin{bmatrix} x_1 & y_1 & X_1 & Y_1 & Z_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_i & y_i & X_i & Y_i & Z_i \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_N & y_N & X_N & Y_N & Z_N \end{bmatrix}$$

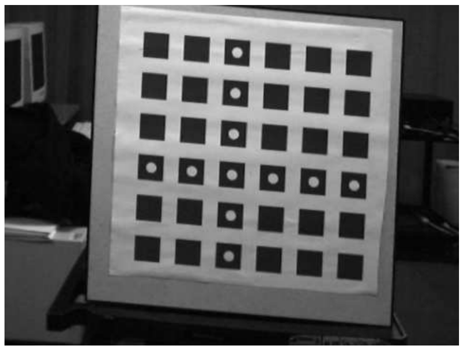
You should compute the 3D coordinates of the calibration grid corners using the grid geometry and coordinate system specified in section 2.1. For example, the top left corner in the front image has 3D coordinates (0,0,0) and the bottom right corner in the second image has 3D coordinates (450,450,150).



(a) Image formation in an affine camera. Points are projected via parallel rays on to the image plane.



(b) Image of calibration grid placed at Z=0 (front.png)



(c) Image of calibration grid placed at Z=150 (back.png)

Figure 2: Affine camera: image formation and calibration images. The coordinate system to be used is superimposed in (b).

If you wish to select the corners interactively in MATLAB, you may find the function `ginput` useful. It is strongly recommended you save this matrix of measurements in a file for later use. You do not need to find *all* the corners, but the more corners you use the more accurate the calibration is likely to be. Be sure to include corners from *both* images. We recommend using at least 12 corners from each image.

2.3 Solving for the camera matrix

Having measured enough features, we can now solve for the camera paramters using Eq. (2). Note that each measurement $(x_i, y_i) - (X_i, Y_i, Z_i)$ yields two linear equations in the 8 unknown camera parameters. Given N corner measurements, we have $2N$ equations in 8 unknowns. Using the measurements of section 2.2 as input, write code which constructs the linear system of equations and solves for the camera parameters which minimze the least-squares error.

Submit:

1. The code you wrote for finding the camera parameters.
2. The RMS error (in pixels) between the corner positions you detected in section 2.1 and the positions predicted by the camera parameters. Include this in the Readme.txt file.
3. The 3x4 camera matrix. Include this in the Readme.txt file.

3 A Cube in Perspective

Which of the following images can be obtained from a perspective projection (or weakly perspective projection) of a cube ? Justify your answers (include these in the Readme.txt file you submit).

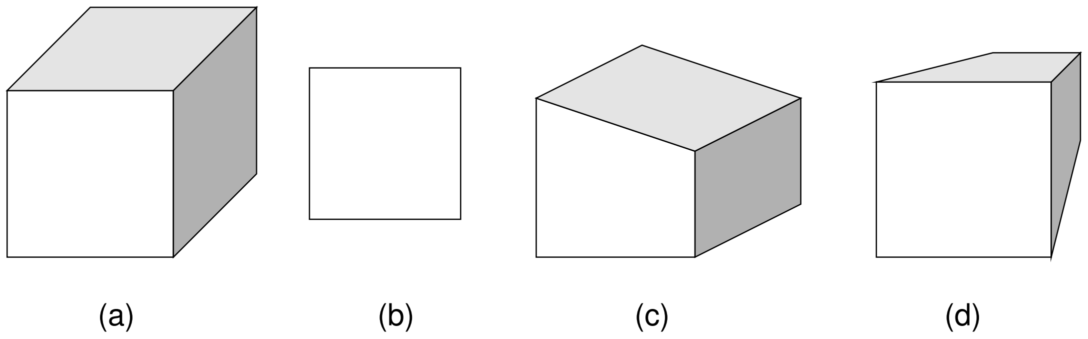


Figure 3: Images for Question 3: A Cube in Perspective