

# CS 223b: Introduction to Computer Vision

## Assignment 1: Introduction to Image Processing and Matlab

Due date: **Monday, January 19th 23:59 PST**

You may work in **teams of up to 3 persons**

Submission via email with subject "Assignment 1: [NAMES]" to cs223b09@gmail.com, where [NAMES] is a comma separated list of the full names of everyone in your group.

### 1 Image Processing and Matlab (20 pts)

#### 1.1 Software Installation (0 pts)

Real-world implementation is essential in computer vision. Vision algorithms are often implemented in software using either MATLAB or Intels Open Computer Vision (OpenCV) library for C/C++. The purpose of this assignment is to give you an initial familiarity with MATLAB. If obtaining access to MATLAB is an issue for you, please email the course staff as soon as possible.

#### Why Matlab?

- Easy to use and learn, with a strong integrated development environment.
- Having both an interpreter and compiler enables rapid prototyping and relatively fast processing.
- Advanced graphics and visualization capabilities are invaluable during experimentation.
- Widely used, with many useful packages available.

#### Octave

Octave is a free software implementation of a language which is mostly compatible with Matlab. You may use Octave if you so desire, but there are a few points that should give you pause:

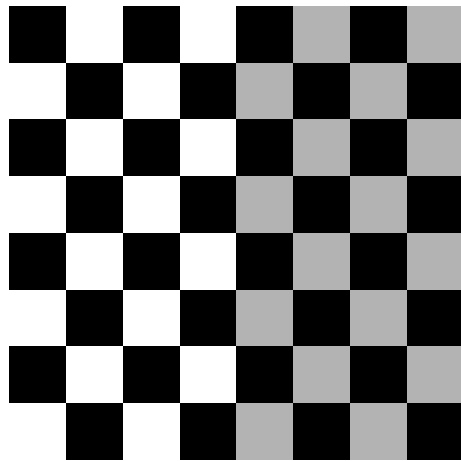
- Octave is not perfectly compatible with Matlab. We are only testing with Matlab, and so your code must run in Matlab unmodified. You must test your code thoroughly in Matlab before submitting.
- Octave is missing some important image processing features. Most notably it only supports loading PPM, PostScript, and an proprietary image format. Most of the images you will encounter in this course will be PNG, JPEG, or BMP.
- Octave itself does not include a graphical front-end. There are some efforts in this direction, but they are relatively immature.
- The TAs will **NOT** help you with Octave!

## 1.2 Basic Matlab (5 pts)

After you have obtained access to a Matlab install, familiarize yourself with the environment a bit. If you have never used Matlab before you may want to read the getting started section in the help document. You are also encouraged to read through some of the help material for the Image Processing Toolbox. Once you feel comfortable with the basics of Matlab run through the following steps to perform some simple image manipulations in Matlab.

### Create and display a checkerboard

- Create a checkerboard with the command `board = checkerboard(20,4);`. Note the `;` at the end of the line. If you leave this out matlab will display the return value of the statement, which in this case will be a 160x160 array of doubles. If you want to inspect a variable in this way you can simply type it into the console. Use the command `board` to inspect the array you just created.
- Display the image you just created with the command `imshow(board)`. You should see something like:



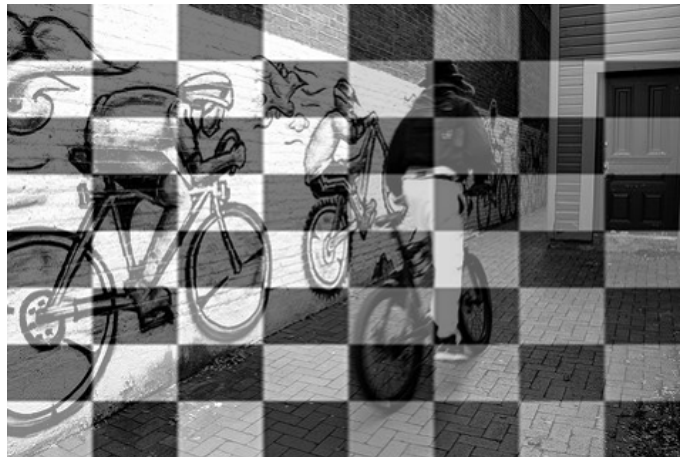
- Save the image to the current working directory as `checkerboard.png` with: `imwrite(board, 'checkerboard.png');`

### Load from a file and blend two images

- Now try loading an image from file. We have provided the image `bike.jpg`, so load it with `bike = imread('bike.jpg');`
- Display the bike image. You should see something like the following image.
- Now convert the image to a matrix of doubles with one double per pixel (ranging from 0.0 to 1.0). Call this new matrix `bikeMatrix`.
- Next resize the checkerboard you created earlier to be the same size as the bike image with `resizedCheckerboard = imresize(board, size(bikeMatrix));`



- Finally perform a blend between `bikeMatrix` and `board`, so that each pixel of the result is the average of the pixels at corresponding locations in the two input images. Display the resulting image, and save it as `blend.png`. Your image should look like this:



- The only things you will need to submit from this section are the two images you saved: `checkerboard.png` and `blend.png`.

### 1.3 Image Transforms (15 pts)

In this section we ask you to play around with some image transformations. For all transformations, begin with the image produced from the command `imread('channel.png');`. We intentionally give you substantially less guidance on this section, so expect to have to spend some time with the help document.

#### Perspective Transformations

- A 2D projective transform is a projection defined by a  $3 \times 3$  matrix  $M$  which transforms a point  $[x y]$  into  $[u v]$  where  $[p q r]' = M [x y 1]'$ , and  $u = p/r$  and  $v = q/r$ . (Look up homogeneous coordinates for more information on why you add the extra 1.) For example

the transform given by

$$\begin{bmatrix} 0.3757 & 0 & 0 \\ -0.2073 & 0.3744 & -0.0013 \\ 99.7018 & 0.6243 & 1.0000 \end{bmatrix}$$

should look like:



- Perform the 2D projective transforms defined by the following 4 matrices, and save the results as *projected1.png* through *projected4.png*. You will want to use built in Matlab functionality. With the functions available in the image processing toolkit you should only need a few lines of code.

$$\begin{bmatrix} 1.6322 & 0 & 0 \\ 0.2120 & 1.6336 & 0.0013 \\ -101.9757 & -0.6322 & 1.0000 \end{bmatrix}$$

$$\begin{bmatrix} 1.4219 & 0.3183 & 0.0013 \\ 0 & 1.4206 & 0.0000 \\ -0.4206 & -101.8704 & 1.0000 \end{bmatrix}$$

$$\begin{bmatrix} 0.7033 & -0.2239 & -0.0009 \\ 0 & 0.9991 & 0 \\ 0.2958 & 0.2239 & 1.0000 \end{bmatrix}$$

$$\begin{bmatrix} 1.1044 & -0.3493 & 0.0003 \\ 0.0011 & 1.5066 & 0.0011 \\ -0.1041 & -0.1560 & 1.0000 \end{bmatrix}$$

## Orthorectification

Orthorectification is the process of transforming an image such that distance in the image is proportional to distance on the object being imaged. It is frequently used with aerial photography so that distances can be easily measured. In this image we might be curious which of the graffiti bikes on the wall is taller. To answer this question, we can use a perspective transformation to orthorectify the wall and then look at the height in pixels of each bike. Write a piece of code to compute the 2D projection matrix needed to perform this orthorectification.

## Custom Image Transform

Create an 'M file' named *CustomTransform.m* and use it to define a custom function. This function should take a single argument, the input image, and return the transformed version of that image. Include a brief description of your transform in your writeup, as well as a few example images and their transformations.

### What to Submit

For this section you need to submit *projected1.png* through *projected4.png*, the orthonormalization matrix (along with the code you used to find it), the code for your custom image transformation, and two or more before and after examples.

## 1.4 The Relationship between Perspective and Affine Transformations (5 Points)

In the previous question, you learned about orthorectification. Can you orthorectify the graffiti in *bike.jpg* via an affine projection? An affine projection will be simply a projection via a 2-by-2 matrix  $A$ , such that image coordinates  $[x y]$  are mapped to  $[u v]' = A[x y]'$ . Prove your conjecture.

### What to Submit

The answer to the question and your proof.

## 1.5 Submitting

### Submission Checklist

- *checkerboard.png*
- *blend.png*
- *projected1.png* through *projected4.png*
- Orthonormalization matrix (in writeup)
- Code used to determine orthonormalization matrix (either as an M file or in writeup)
- *CustomTransform.m*
- Brief description of your custom transform (in writeup)
- Two or more before and after examples of your custom transform in action (in writeup)
- The answer to the question about orthorectification with an affine transformation and your proof (in writeup)

When finished, submit all sourcecode, all images produced, and a .doc or .pdf writeup. Submission should be via an e-mail to [cs223b09@gmail.com](mailto:cs223b09@gmail.com) with the subject "Assignment 1: [NAMES]".