hw3
Brian Burns <bburns@cs.utexas.edu>
CS376 Computer Vision, Spring 2011
Dr. Kristen Grauman

I. Programming Problem: Image Mosaics

Image Mosaic I

The image below shows the result of using 8 manually selected corresponding points to calculate a homography matrix, then warping the second image into the camera plane of the first image, then merging the two.

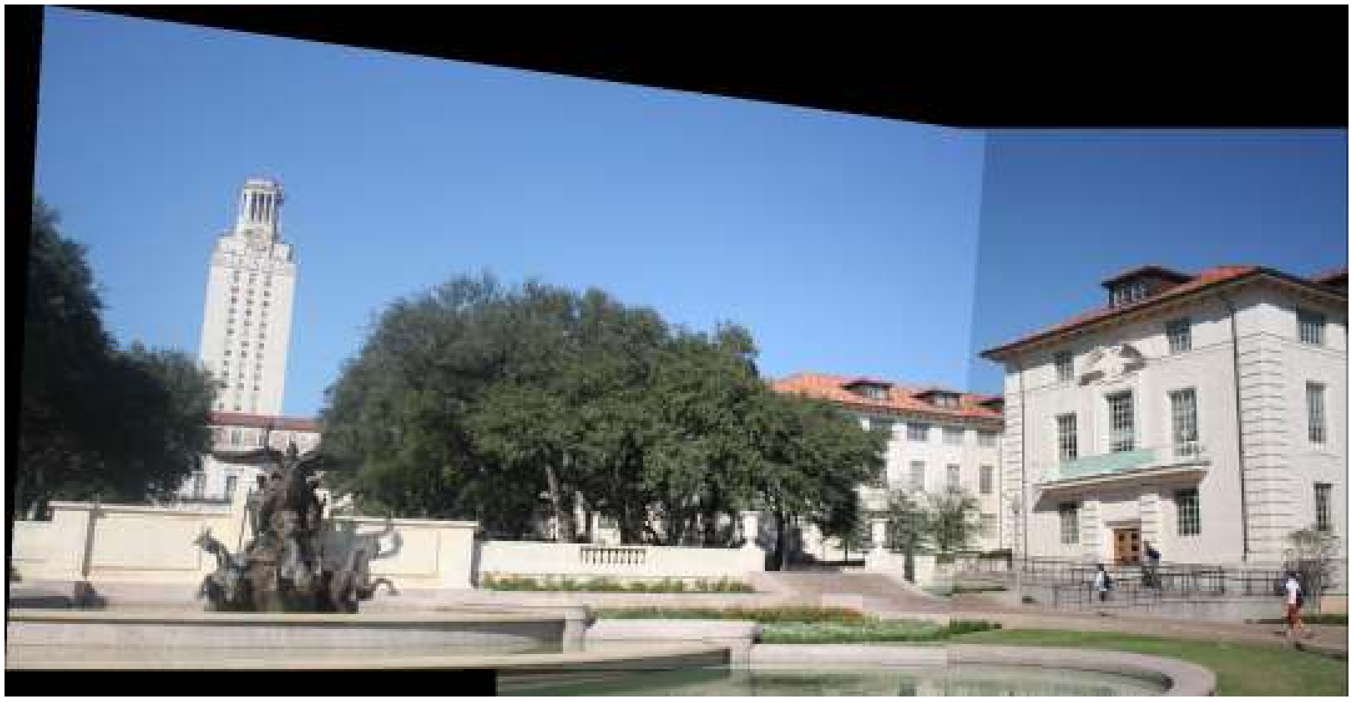
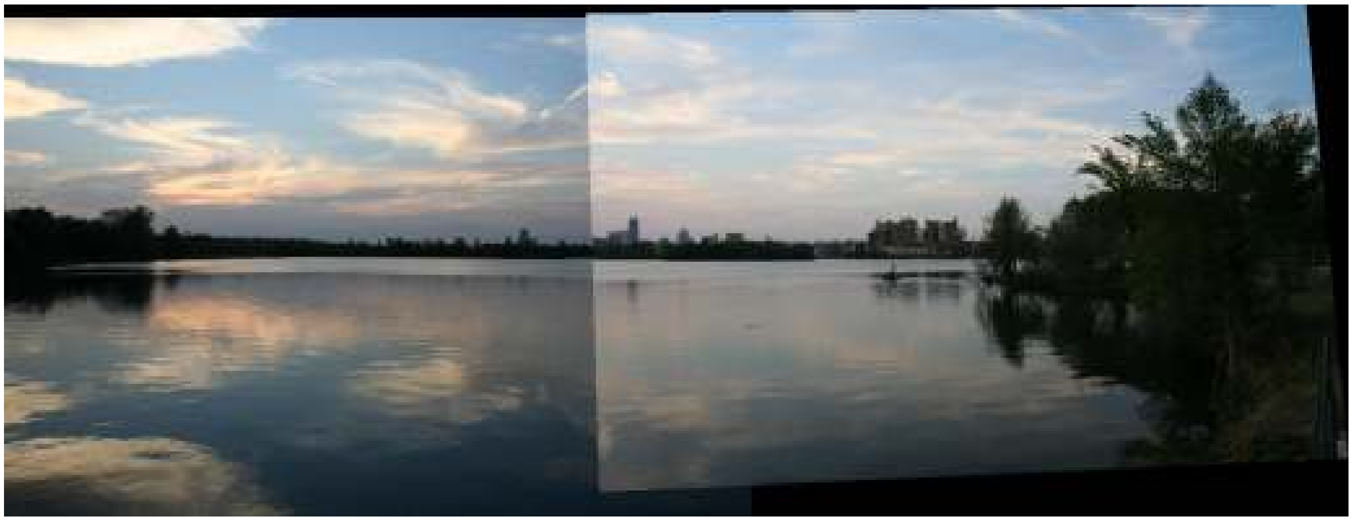


Image Mosaic II

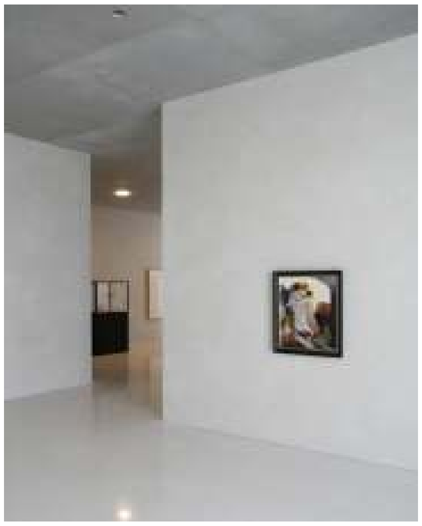
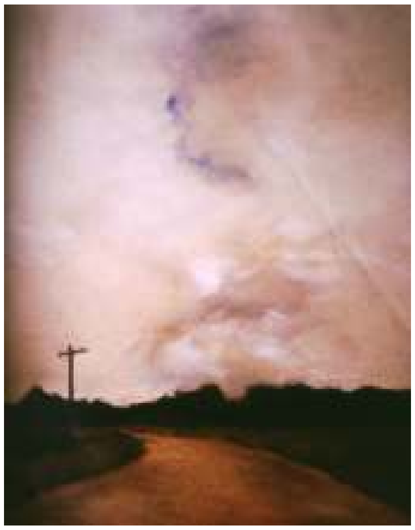
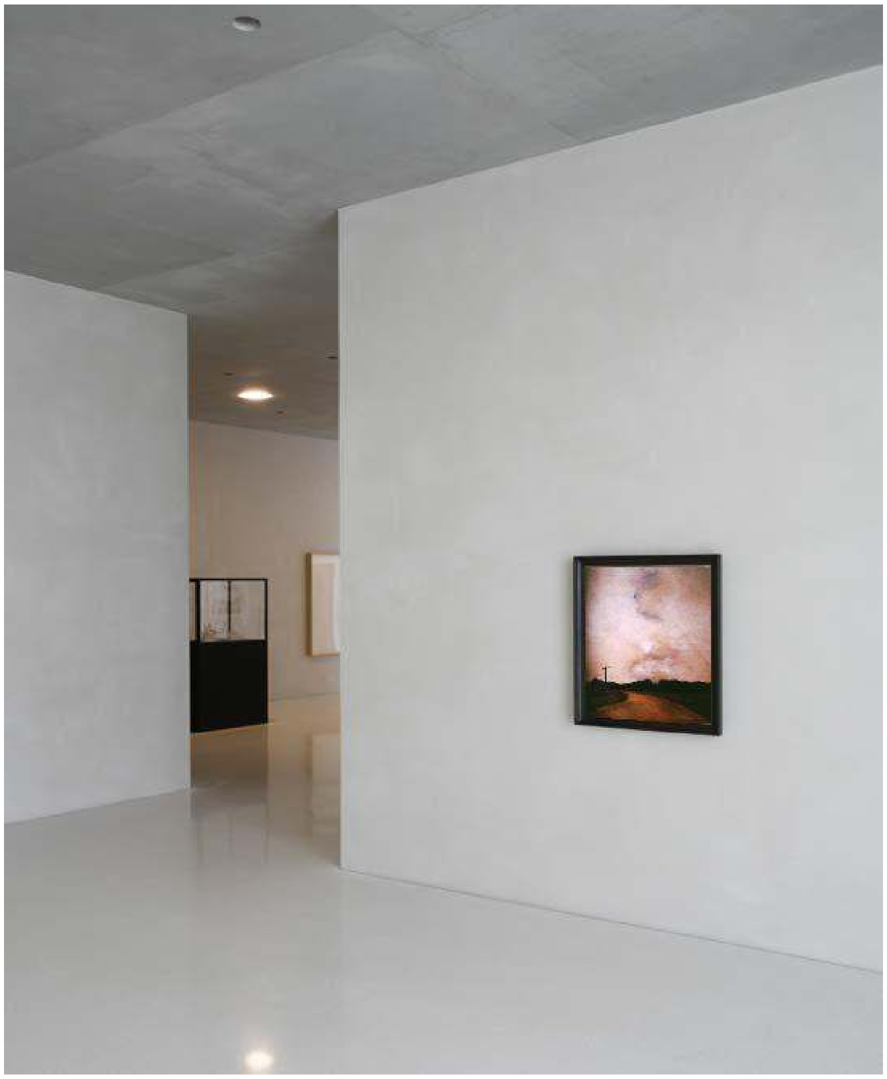
The same procedure produced the following mosaic of Lady Bird Lake. The exposures of the two images are different, unfortunately (I couldn't remember how to manually override the exposure time on my camera), and it was hard to get any corresponding points other than those on the skyline buildings. There was only one good correspondence point in the clouds.



Images: me

Framing an Image

Starting with an image of an art gallery, the corners of a picture frame were manually located, then another painting was warped and inserted into the frame.

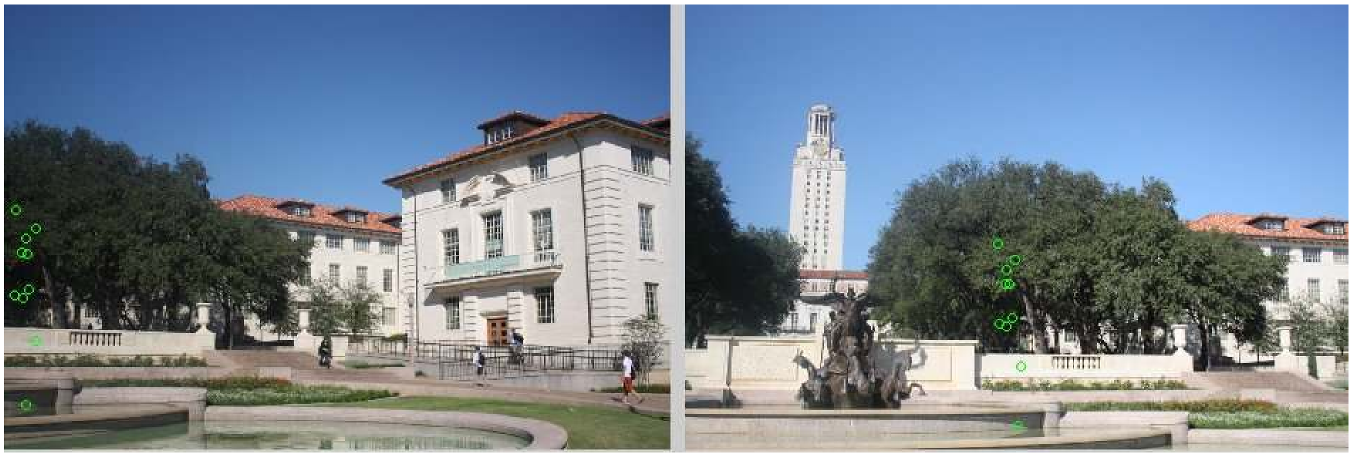




Gallery: http://www.architekturezt.com/kolumba-art-museum-by-peter-zumthor-in-cologne/kolumba-art-museum-interior-design/
Painting: me

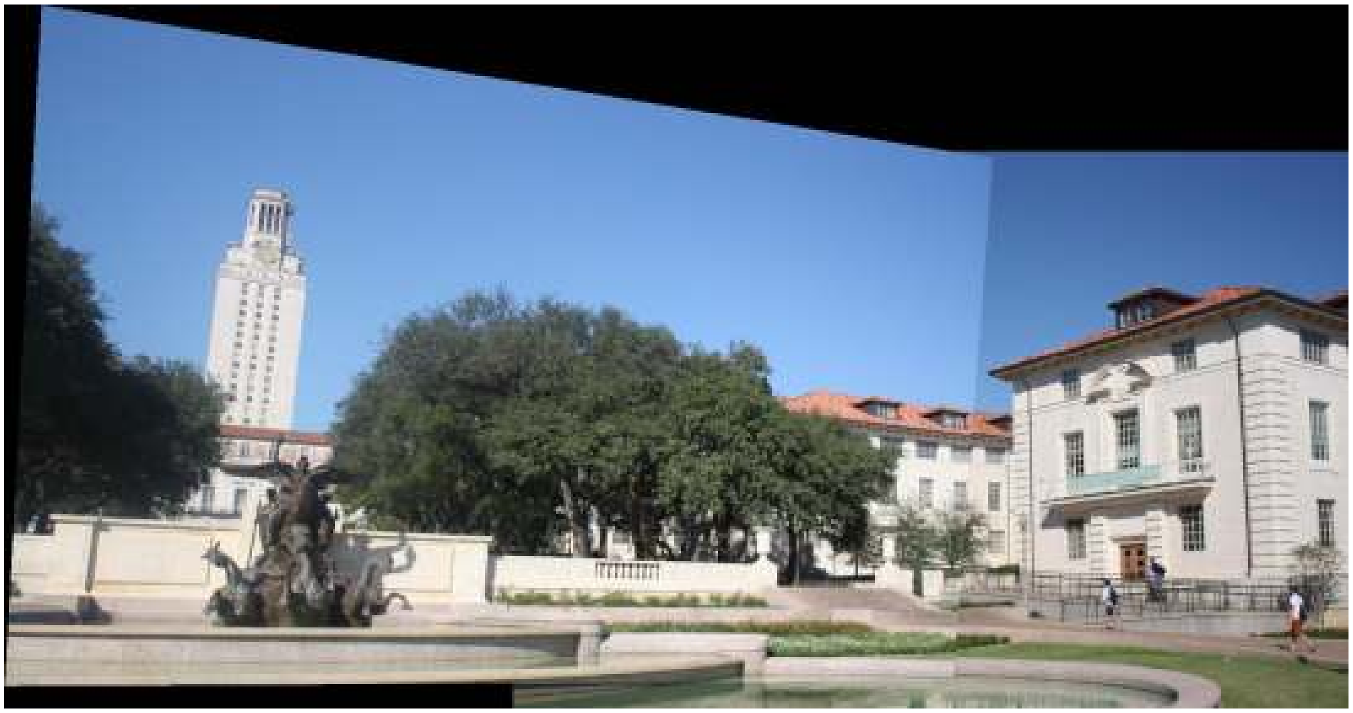
II. [OPTIONAL] Extra credit

1. Replace the manual correspondence stage with automatic interest point detection and local feature matching.

Did this using the suggested library from vlfeat.org. SIFT feature points are found in each image using the vl_sift function, then correspondences are found with the vl_ubcmatch function. The image below shows the first 10 correspondence points (unsorted by score) used to calculate the homography matrix.

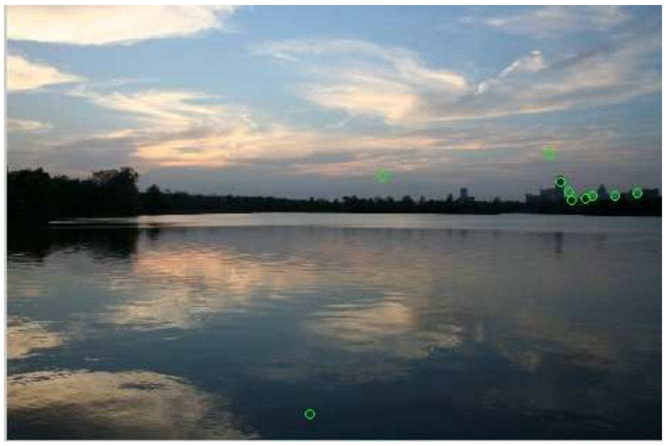
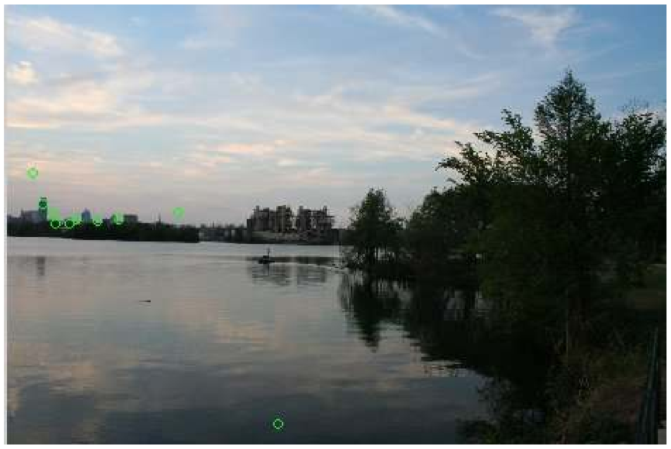


Here is the merged image result - it's nearly identical to the manual correspondence image.



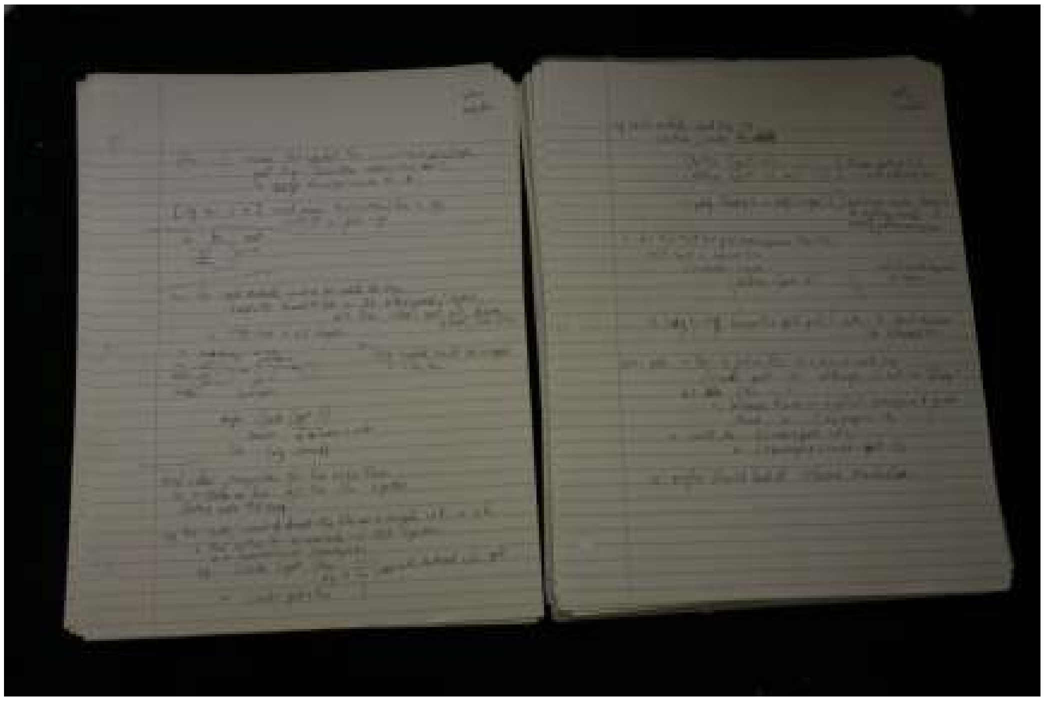
One problem was that when using more than 10 of the detected correspondences the homography matrix would be completely wrong. The routine finds hundreds of matches, but they're unsorted by similarity score.

So I sorted the matches by the similarity score, thinking that the top 20+ matches would then all be good ones. But when I tried this with the lake images and displayed the top 10 correspondences, that were still two outliers that were affecting the results. I tried it on the UT images also and some of the best matches were in random patches of the sky that weren't actual correspondence points. So using RANSAC on the matches would help, because it would eliminate such outliers.

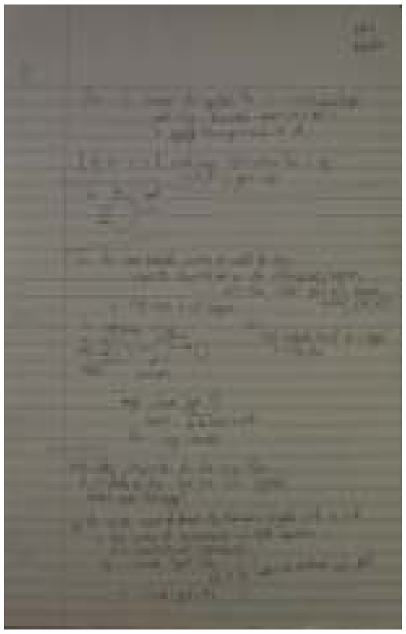



3. Rectify an image with some known planar surface (say, a square floor tile, or the rectangular face of a building façade) and show the virtual fronto-parallel view.

I've been scanning in a bunch of old documents and papers using a digital camera on a tripod, pointing down at a desk. There is some perspective distortion though, so being able to rectify the images will be helpful.

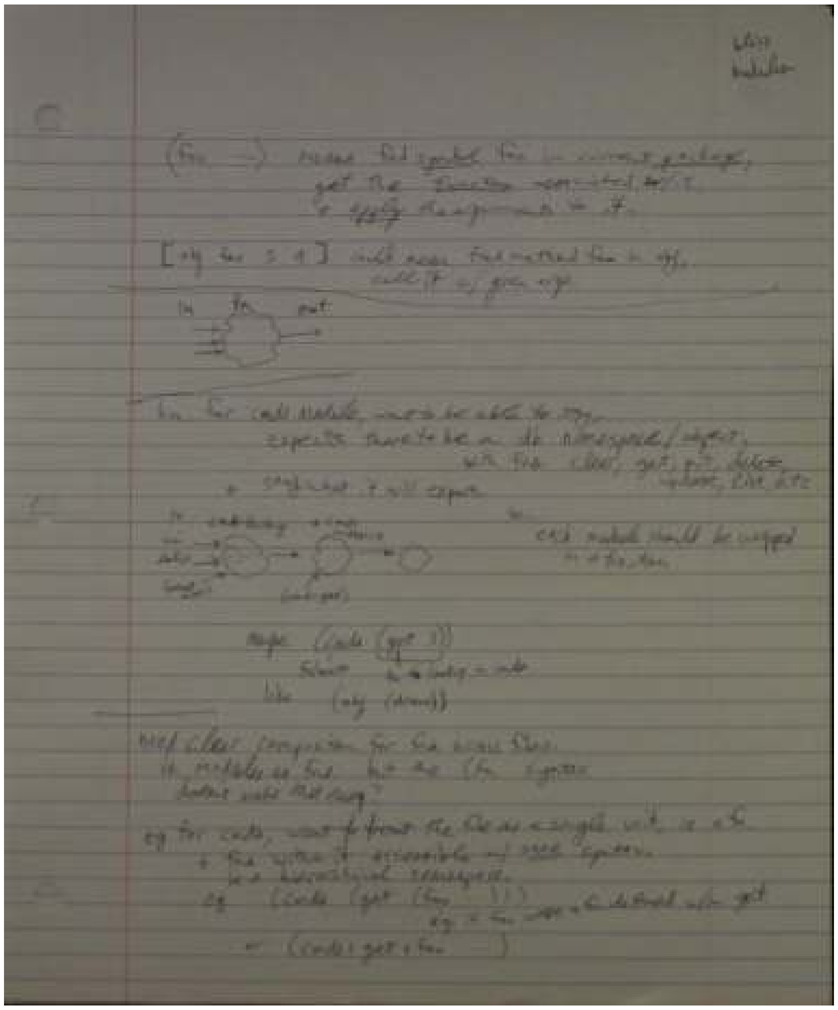


I selected four corner points and then warped the image into a new rectangular image, using the existing code. To calculate the dimensions of the new image, I used the maximum of the horizontal and vertical extents from the selected corner points on the original image, in order to use as much information as possible. This winds up distorting the actual dimensions of the paper though:



[illegible]

So apparently there's no way to get the correct aspect ratio without knowing the camera geometry, or having another view on the same object. So for now I just applied the known aspect ratio to get a correctly rectified image:



[illegible]