Ryan Johnson cslogin:**rjohnson** **CS378 Pset3**

Section II – Image Mosaics

0. Implementation Overview

Getting correspondences:

```
function [ ccoords ] = getCoords( image1, image2 )
%GETCOORDS assists user in mapping related points in two images
%   returns an Nx2x2 matrix, where the last dimension specifies
%       the image and the second dimension specifies x,y (1,2)
```

Computing the homography parameters:

```
function [ H ] = computeHomography( coords )
%COMPUTEHOMOGRAPHY takes a set of related coordinates lists, computes H
%   coords is a 3-d matrix containing xy-coordinate tuples
%   coords(row,col,depth) format: 'row' indexes the coordinates tuples,
%       col=1 selects x coordinate, col=2 selects y coordinate,
%       depth indexes the tuple
%       ex: coords(3,1,2) gives the x value of the 2nd value from the 3rd
%           xy-coordinate tuple
%   returned H is the homography matrix for a transform from coords(:,:,1)
%       to coords(:,:,2);
%   a minimum of 4 xy-coordinate tuples are required
```

Warping between image planes:

```
function [ dstCoordGrid, vectXY ] = warpUsingInv( image, H )
%WARPUSINGINV warps image using H and its computed inverse
%   'image' is an RGB image of uint8 or double (converted to double here)
%   'H' is a 3x3 homography matrix
%   'dstCoordGrid' is the resulting warped image. note that its dimensions
%       will (probably) differ from the source image, and its coordinate
%       system is translated from the destination images coord-system
%   'vectXY' is a vector which translates dstCoordGrid onto the eventual
%       destination images coordinate system.
```

Create the output mosaic:

```
function [ finalImage ] = combineResults( image, imWarped, vectXY )
%COMBINERESULTS overlay two images at a given translation
%   vectXY is a vector of x and dy which gives the location of "imWarped"'s
%       origin pixel in relation to "image"'s origin pixel
```

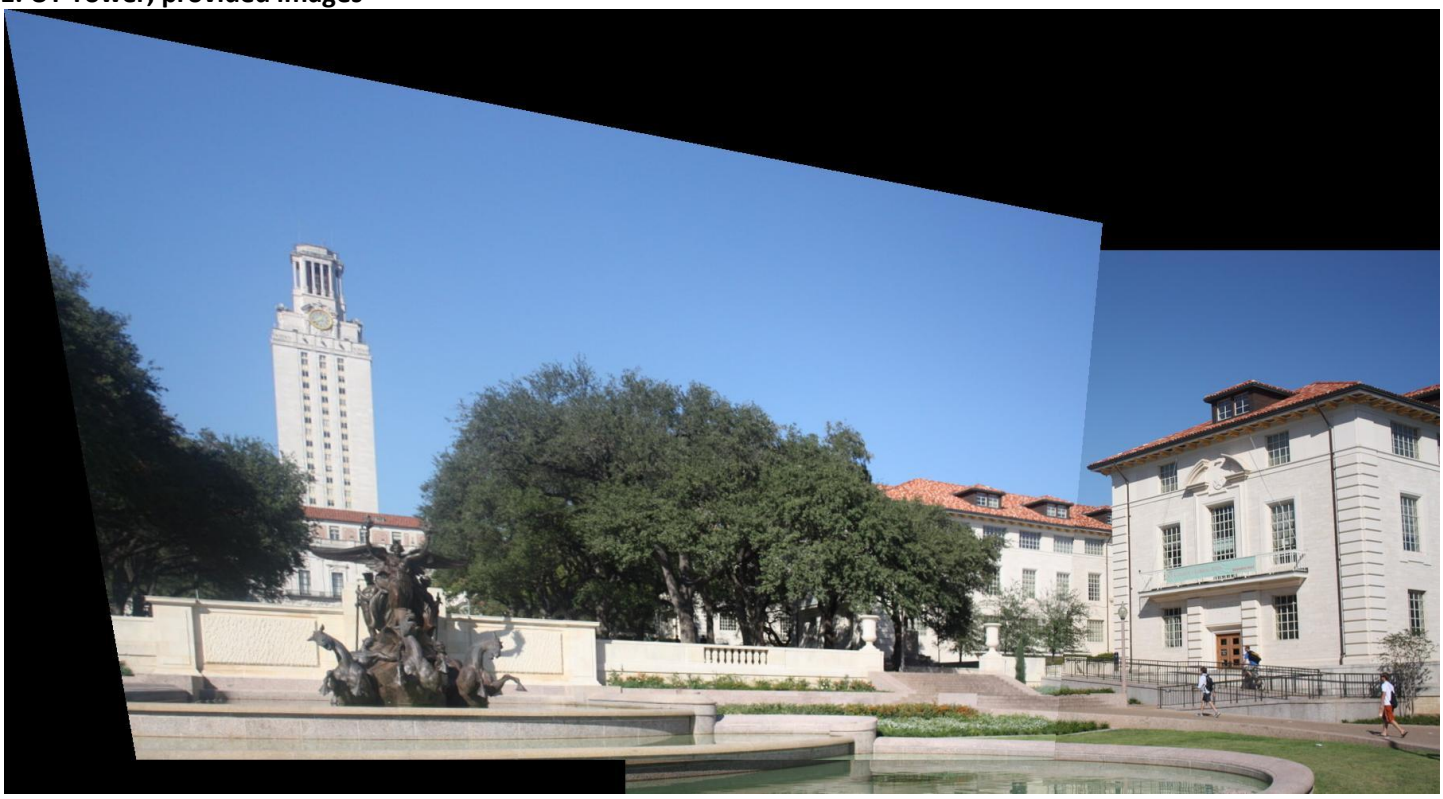
'Wrapper' function:

```
function [ final ] = pset3( image1f, image2f)
%PSET3 'wrapper' function for pset3
%   takes two image filenames as an argument and then calls getCoords,
%   computeHomography, warpUsingInv, and combineResults scripts in order
%   to create a mosaic. also uses hom2cart convenience function while
%   verifying transform
```

Coordinate conversion:

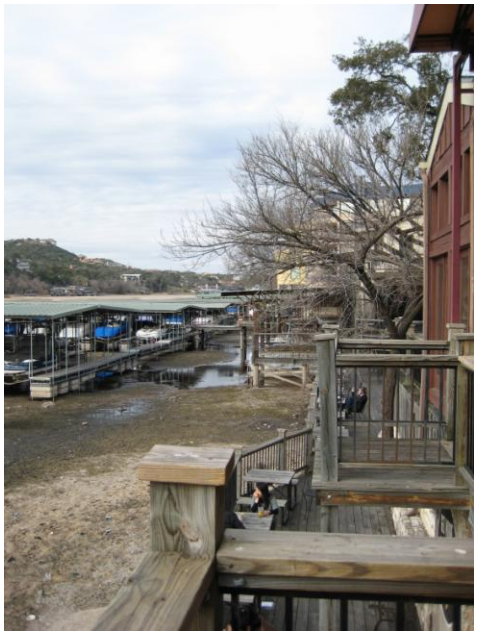
```
function [ result ] = hom2cart( M )
%HMULT converts a matrix of homographic coordinates into cartesian coords
%   divides first two rows by the third, discards the third
```

1. UT Tower, provided images



2. Mozart's Coffee minus a lake

source images:

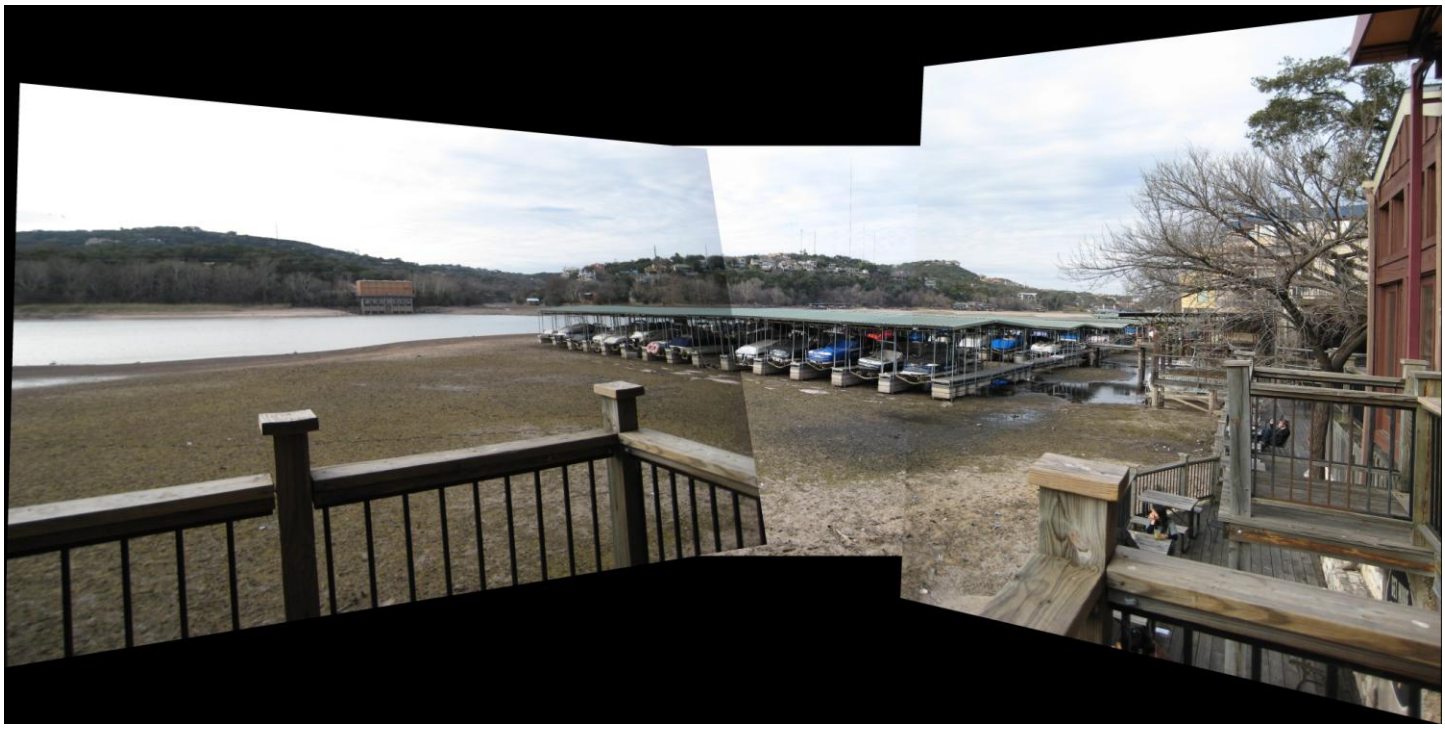


3rd image, from a shooting point a few feet back

2.

Mozart's coffee 2-image mosaic

2. Mozart's 3-image mosaic using an additional image from a slightly shifted camera location. Distant locations on the image merge well since the distance is >> than the shift in camera, but artifacts occur with nearfield objects in the left and middle source images since, in the nearfield, the shift and distance are of similar magnitude:

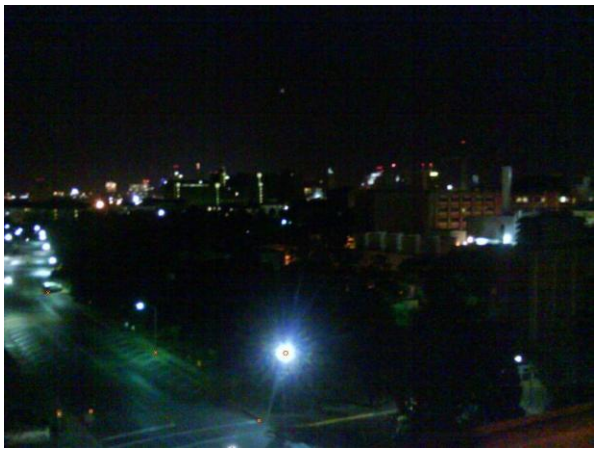


However, if we crop away the nearfield (and also black at the edges just for cleanliness' sake) it is difficult to tell that one of the source images is from a shifted location. This would not work well if the subject matter were close in, like a room indoors.

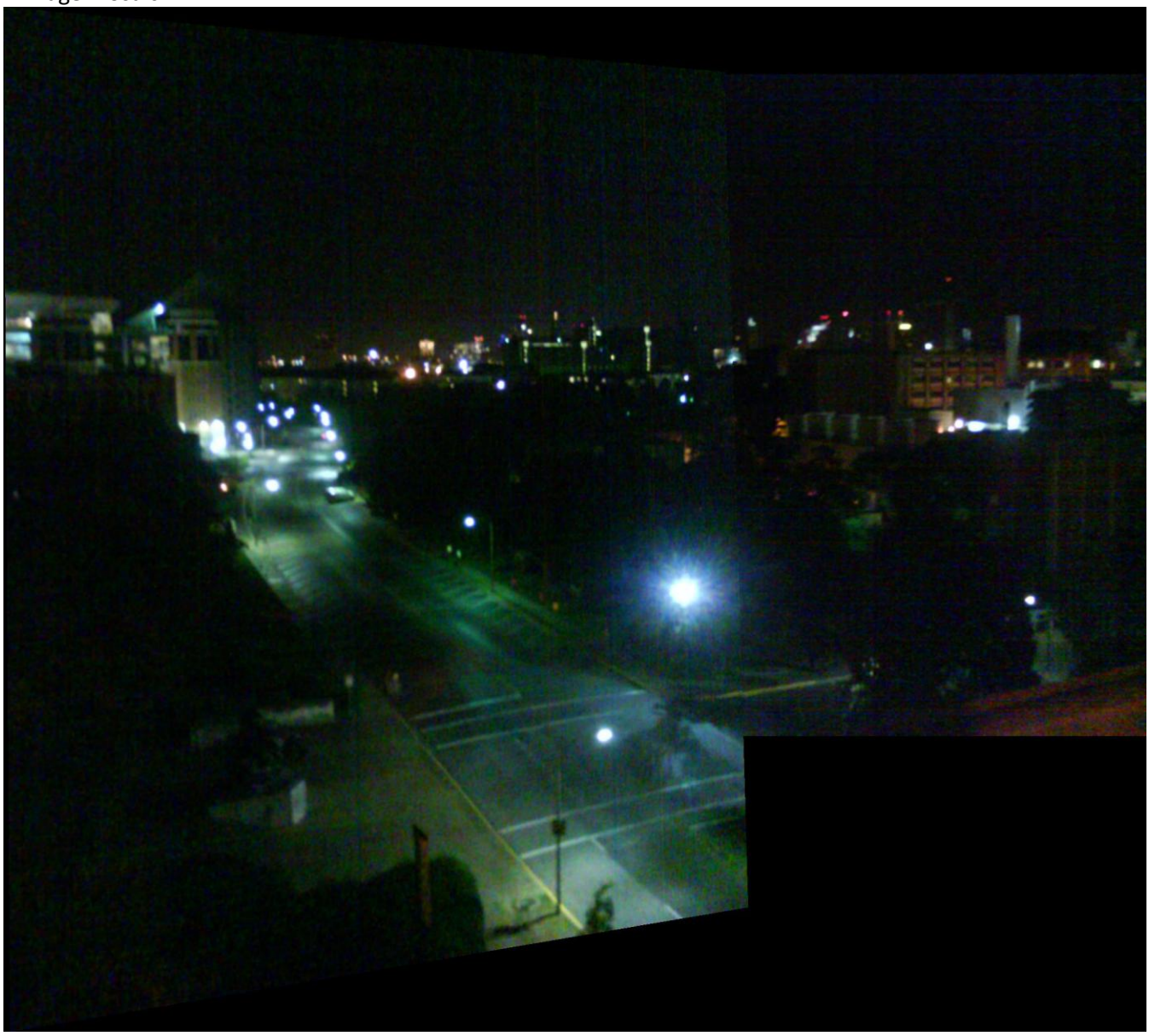
Cropped 3-image mosaic

2. UT skyline at night, from the top of the San Jacinto parking garage

source images:



2-image mosaic:

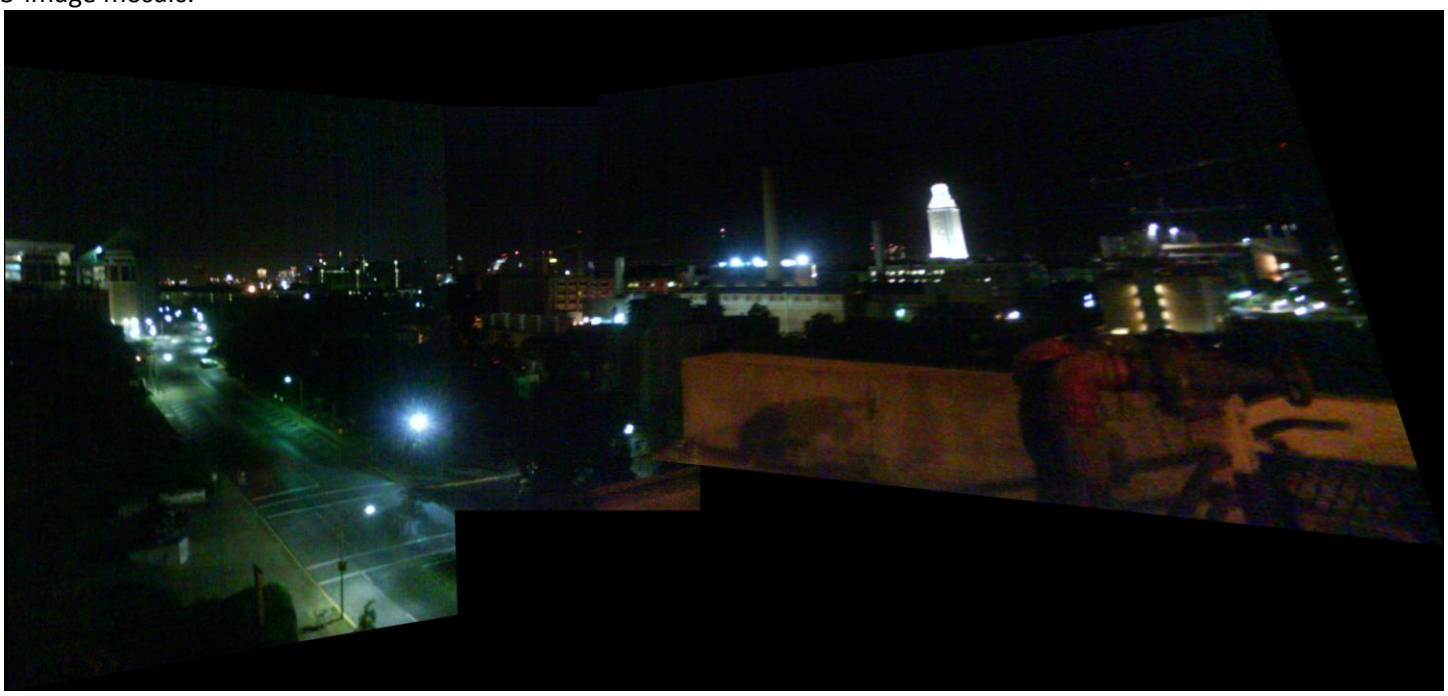


2. UT Night skyline continued

As before, adding a 3rd image from a slightly shifted camera location which works well at a distance but introduces artifacts with nearer objects

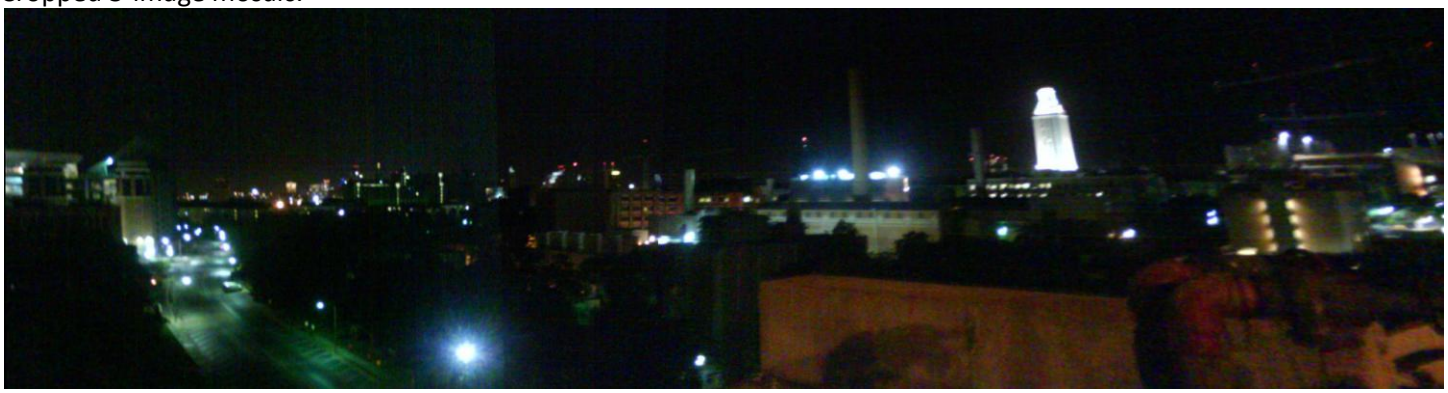
(slightly) shifted location source image:

3-image mosaic:



If we crop away the shared near field, the shift in camera location for the right 1/3rd of the image is essentially unnoticeable. In fact, the articles from lighting differences between the left and middle 1/3rds of the image are for more noticeable, even though these two sources were taken at the same origin point.

Cropped 3-image mosaic:



3. Combining a photo of my room with an old photo of New York's Rockefeller center to pretend that I live in a high rise

Rockefeller Center from 515 Madison Avenue by Samuel Gottscho, 1933

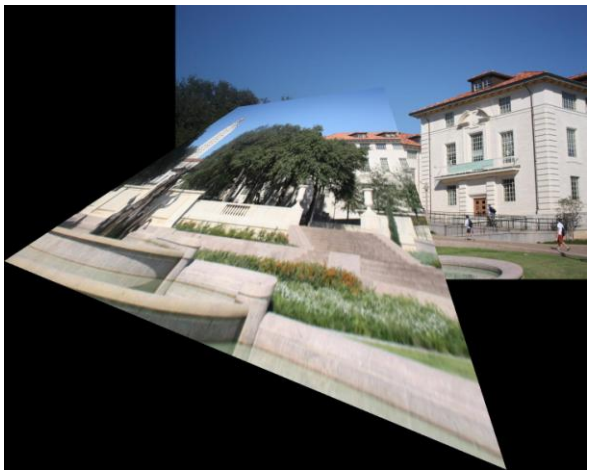
Section III

1. RANSAC

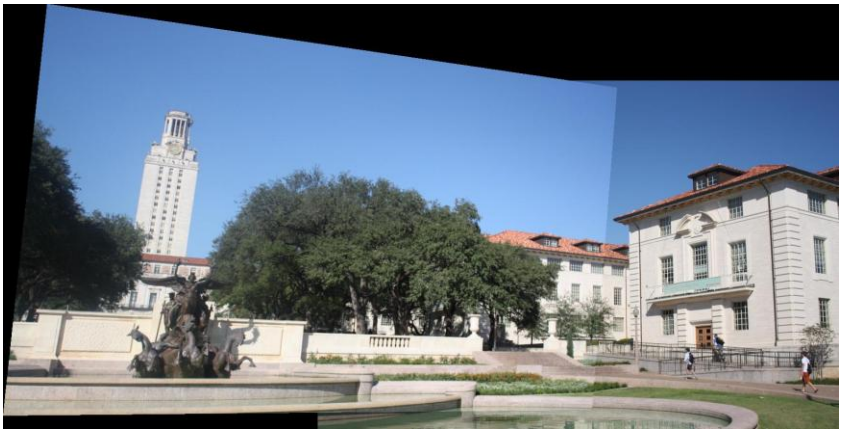
```
function [ H ] = ransac( ccoords )
%RANSAC robustly estimate the homography matrix from noisy correspondences
%   takes a matrix of xy-coordinate tuples generated by getCoords
%   uses RANSAC algorithm to find an appropriate homography matrix for
%   a set of coordinate transforms, even if some of them are 'bad'
%   makes calls to computeHomography and hom2cart
```

A comparison using the same sets of coordinates with RANSAC enabled & disabled.

RANSAC **OFF**



RANSAC **ON**



Differences between sample points and their calculated projections:

RANSAC **OFF**

RANSAC **ON**

The coordinates at the top of the urns were selected on the left side in the first image, and the right side in the second.

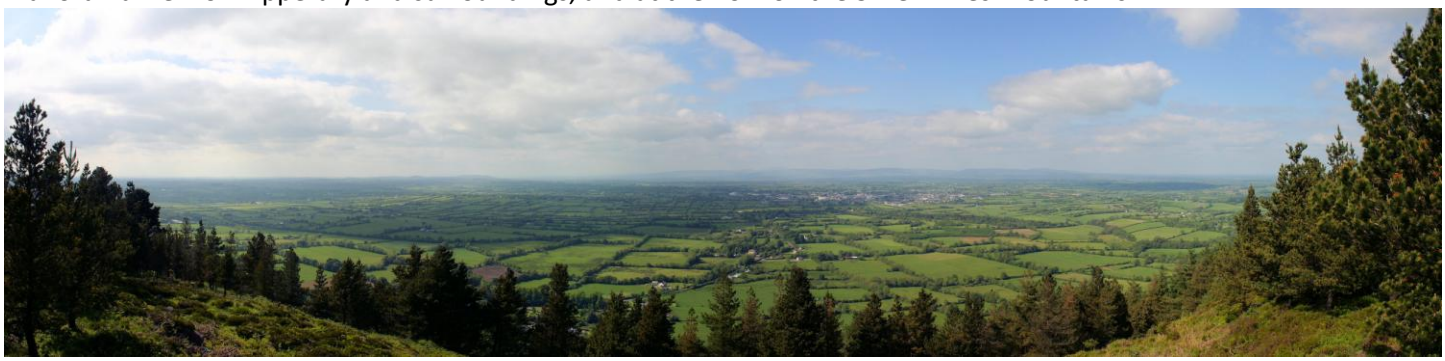
In the first image (RANSAC off) it can be seen that this discrepancy has thrown all of the coordinate calculations off.

In the second image, RANSAC is enabled and has ignored the two outliers on the urns when calculating H. The correct pairs are verified correctly, and we can see the yellow crosses on the urns marking where the two outlier samples should actually have been placed (instead of at the empty red circles)

3. 'square rectification' – bird's eye view of hedgerows & fields

```
function [ fimage ] = rectifySquare( image, N )
%RECTIFYSQUARE rectifies an image so a selected area is a square
%   get coordinates from an image, then maps them to a square of size N*N
%   warps the full image using the Homography matrix thus derived.
%   When selecting points: start at top left, proceed clockwise.
```

Original image – http://en.wikipedia.org/wiki/File:Panorama_tipperary_silvermines_mountains.jpg
"Panorama view on Tipperary and surroundings, and at the horizon the Silvermines Mountains"



Cropped to what I'm interested in – the fields & hedgerows

By picking a field that looks like it should be a square, the result is:

Which worked pretty well. The selected field to rectify is towards the lower middle of the image, and the of the upper left corner now have right-angleish corners as they should.