Assignment 1:

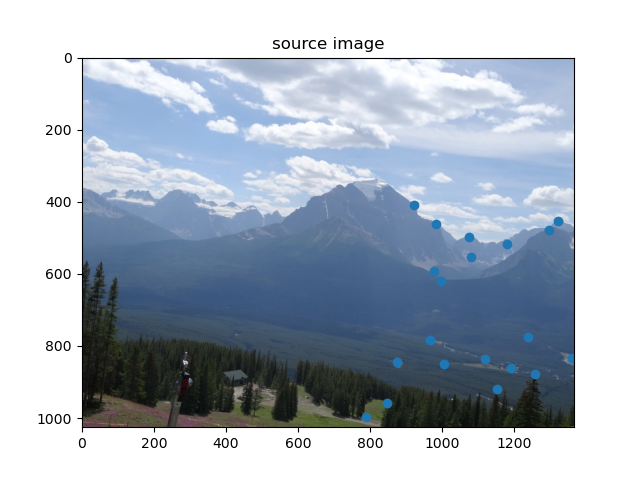
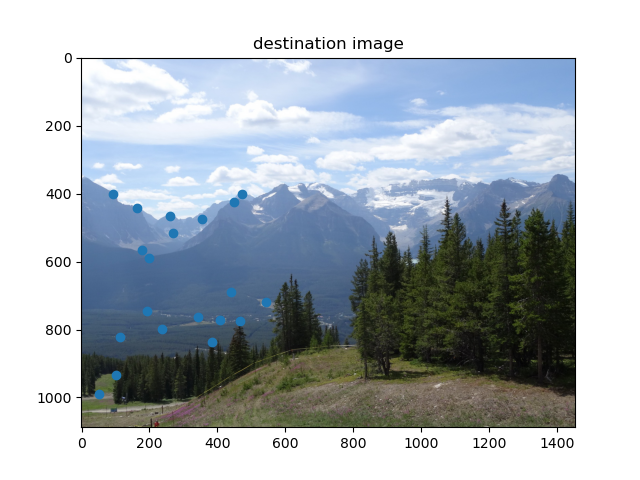
Homography & Panorama

Avner Lavi, 308202076

Lev Ayzenberg, 312201288

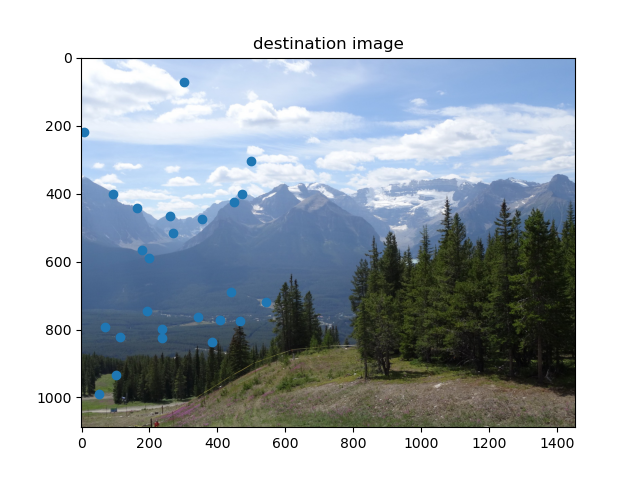
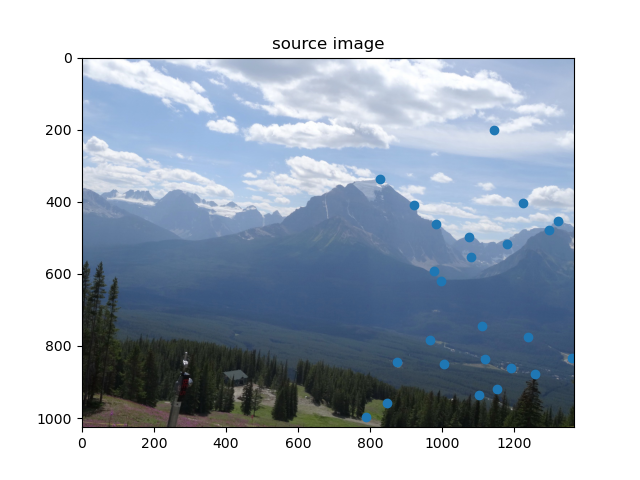
# Intro:

The source and destination images, with perfectly-matching points, are:



The points displayed above are indeed a perfect match in the two images.

The same pictures, with imperfectly-matching points, are:

And it can be seen that some point pairs are not a perfect match.

# Part A: Homography computation

1. A system of equations of the form describing the projective transformation can be found by examining the projective transformation over a set of points (at least 4 are required for the conversion matrix to be found), which are mapped by the conversion matrix (with row vectors ) to .   
   Following the notation described in class:

Where is a zero vector.

Thus, we obtain a set of two linear equations in the elements of for each point. Using such points, we obtain the following system of linear equations in matrix form:

We have reached a system of equations of the desired form, where is as described above, is a column vector containing the elements of the conversion matrix , and is a vector of zeros.

As described in class, we wish to minimize the squared norm, that is , while satisfying the constraint . The solution of the optimization problem is given by the eigenvector corresponding to the smallest eigenvalue of , which when reshaped is a matrix yields the conversion matrix .

1. An implementation of compute\_homography\_naive is given.
2. Calculating the transformation coefficient using the matching point sets given in match\_perfect.mat yields the following conversion matrix:

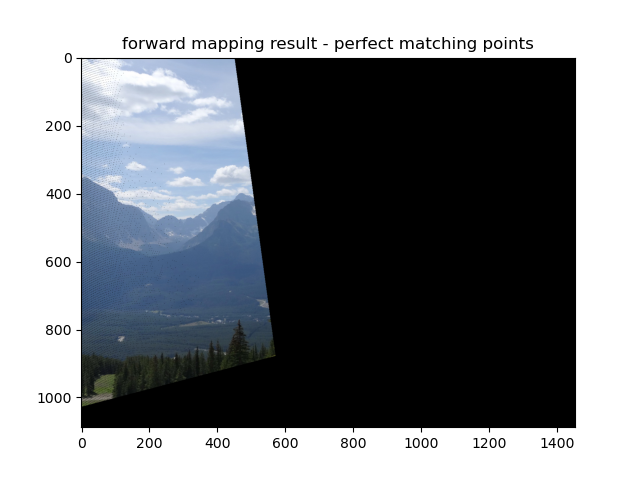
[[ 1.43457214e+00 2.10443232e-01 -1.27718679e+03]

[ 1.34265154e-02 1.34706123e+00 -1.60455874e+01]

[ 3.79279298e-04 5.56523147e-05 1.00000000e+00]]

# 

# Part A2: Forward mapping slow and fast

1. An implementation of compute\_forward\_mapping\_slow is given, and calling the function with the homography derived from the matching points in match\_perfect.mat gives out the following transformed image: 

One can see that the locations in the source image matching points which were displayed in the intro have been ‘moved’ to the locations in which they were depicted in the destination image.

1. An implementation of compute\_forward\_mapping\_fast is given.
2. The main problems with forward mapping are:

* Since the pixels in the source image are transformed one by one, there is no guarantee that each pixel in the destination image coordinate system will get a value from the source image- this leads to ‘holes’ in the mapped image. These regions can be seen in the above image in the leftmost columns, where they are vertical and diagonal black ‘streaks’.
* The destination image indices are integers, and there is no guarantee of the indices of the source image (integers as well) being mapped to integers. This is handled in the presented forward mapping implementation by rounding the mapped index values, and leads to inaccuracies at the pixel level.

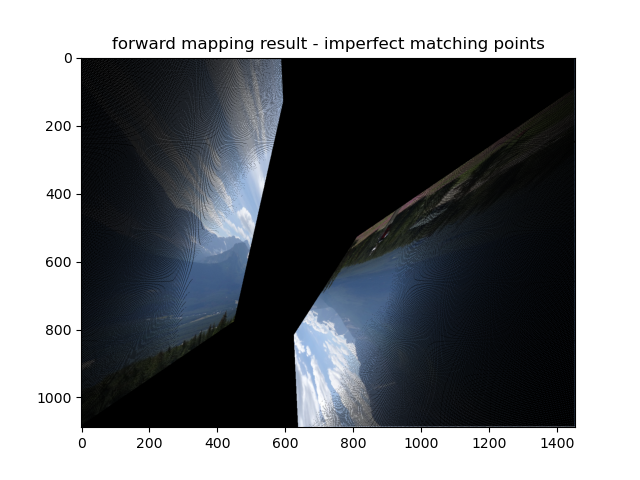
1. Using the matching point pairs from matches.mat, compute\_homography\_naive returns the following homography conversion matrix:

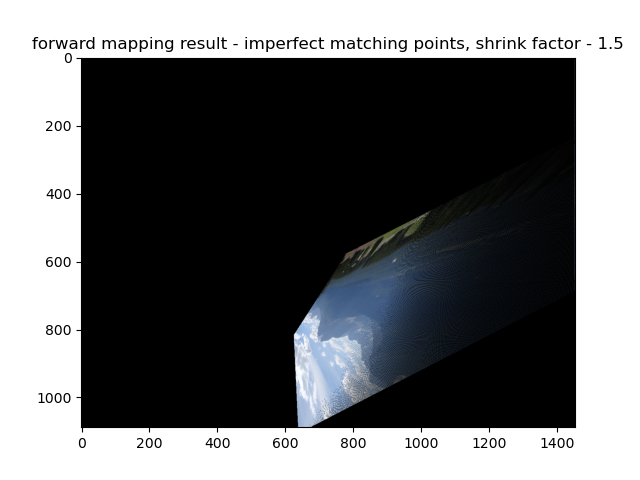
[[-5.86018376e-01 -1.31259752e-01 6.25479638e+02]

[-6.25851768e-01 -4.85276777e-01 8.17143199e+02]

[-9.48023409e-04 -3.85199768e-04 1.00000000e+00]]

Calling compute\_forward\_mapping\_fast using the above conversion matrix gives out the following mapped image:

As the mapped image is too large to be placed in the destination image coordinate system it folds back to the other end of the image. 

Shrinking the source image by a factor of 1.5 in both width and height yields:

The result obtained is different from the original result, as the different and imperfect set of matching points has resulted in a conversion matrix which incorrectly maps the source image to the destination image’s coordinate system.

# Part B: Dealing with outliers

1. An implementation of test\_homography is given.
2. An implementation of meet\_the\_model\_points is given.
3. If there are 30 match points and it is known that 80% of them are correct, the probability of failing to create a ‘correct’ model after randomizations is given as:

Where is the minimal number of match points needed to generate a model, which in this case is 4 (as we want to solve a -by-9 equation system to find the homography parameters, and the solution is scale-independant). This leads to:

For a confidence level of 90% we obtain , so the number of randomizations needed is 5.

For a confidence level of 99% we obtain , so the number of randomizations needed is 9.

In order to cover all options of match points, and assuming we might choose the same set of match points as we have chosen before, we would theoretically need to perform an infinite number of randomizations.

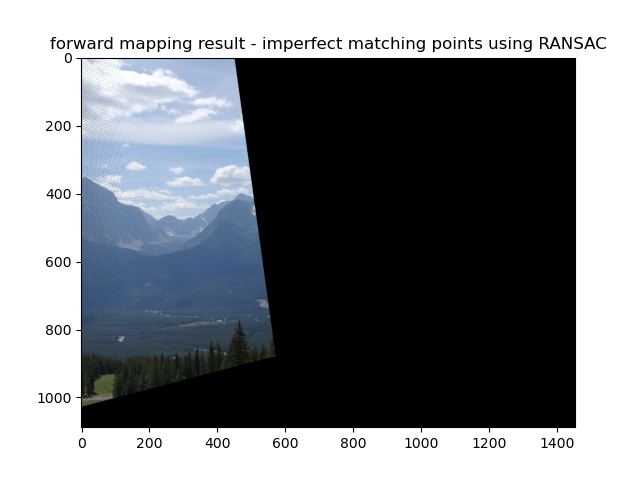
1. An implementation of compute\_homography is given.
2. Running cumpute\_homography with the matching points given in match.mat, with fit\_percent=0.8 and max\_err=25, gives out the following conversion matrix (unnormalized):

[[ 1.43457214e+00 2.10443232e-01 -1.27718679e+03]

[ 1.34265153e-02 1.34706123e+00 -1.60455872e+01]

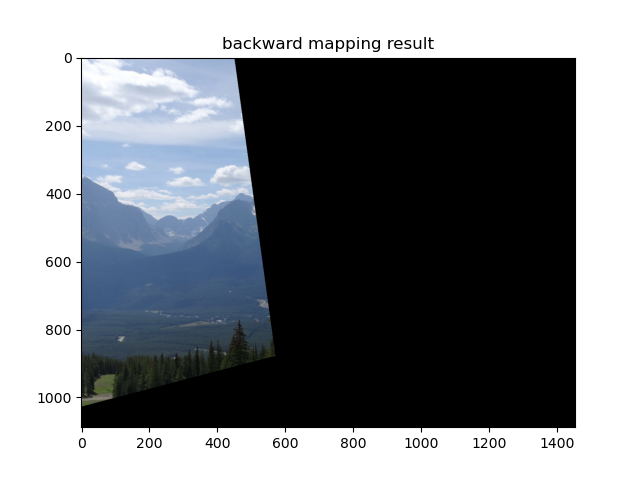
[ 3.79279298e-04 5.56523146e-05 1.00000000e+00]]

For which forward mapping yields the following mapped source image:

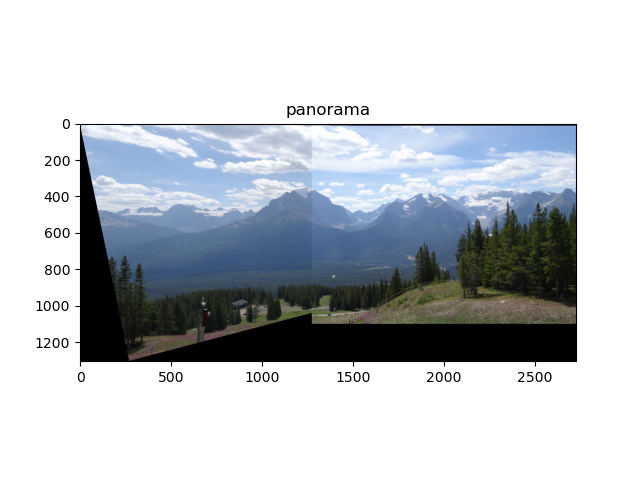
The resulting image is very similar to that obtained in question 4, as opposed to that obtained in question 7. Therefore, we can easily see the effects of RANSAC in eliminating the effects of outlier matching points which could distort the homography.

# Part C: Panorama Creation

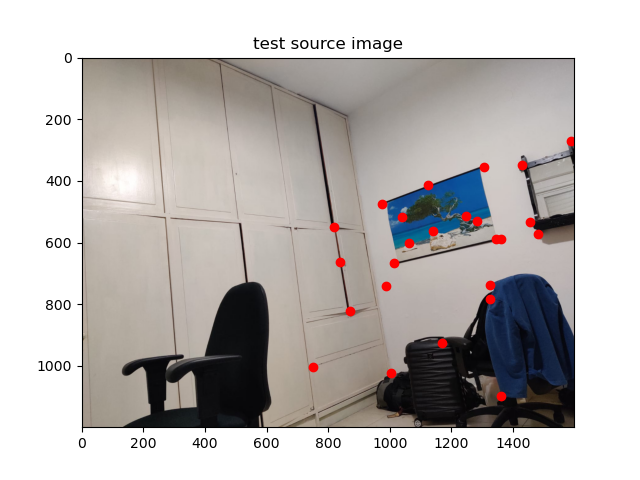
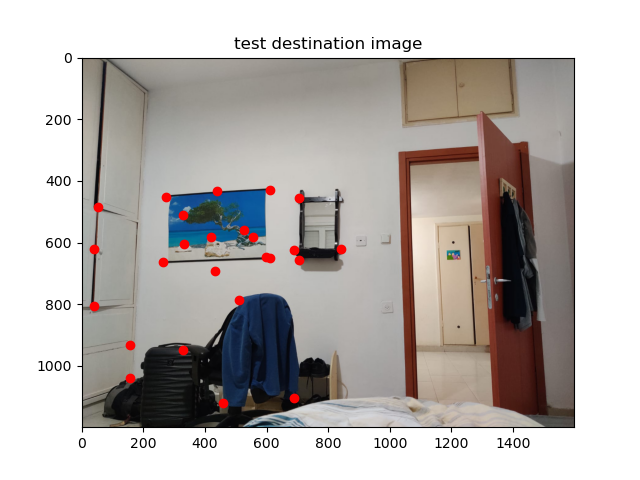
1. An implementation of compute\_backward\_mapping is given, and calling the function with the inverse homography to that found in question 12 gives out the following transformed image:

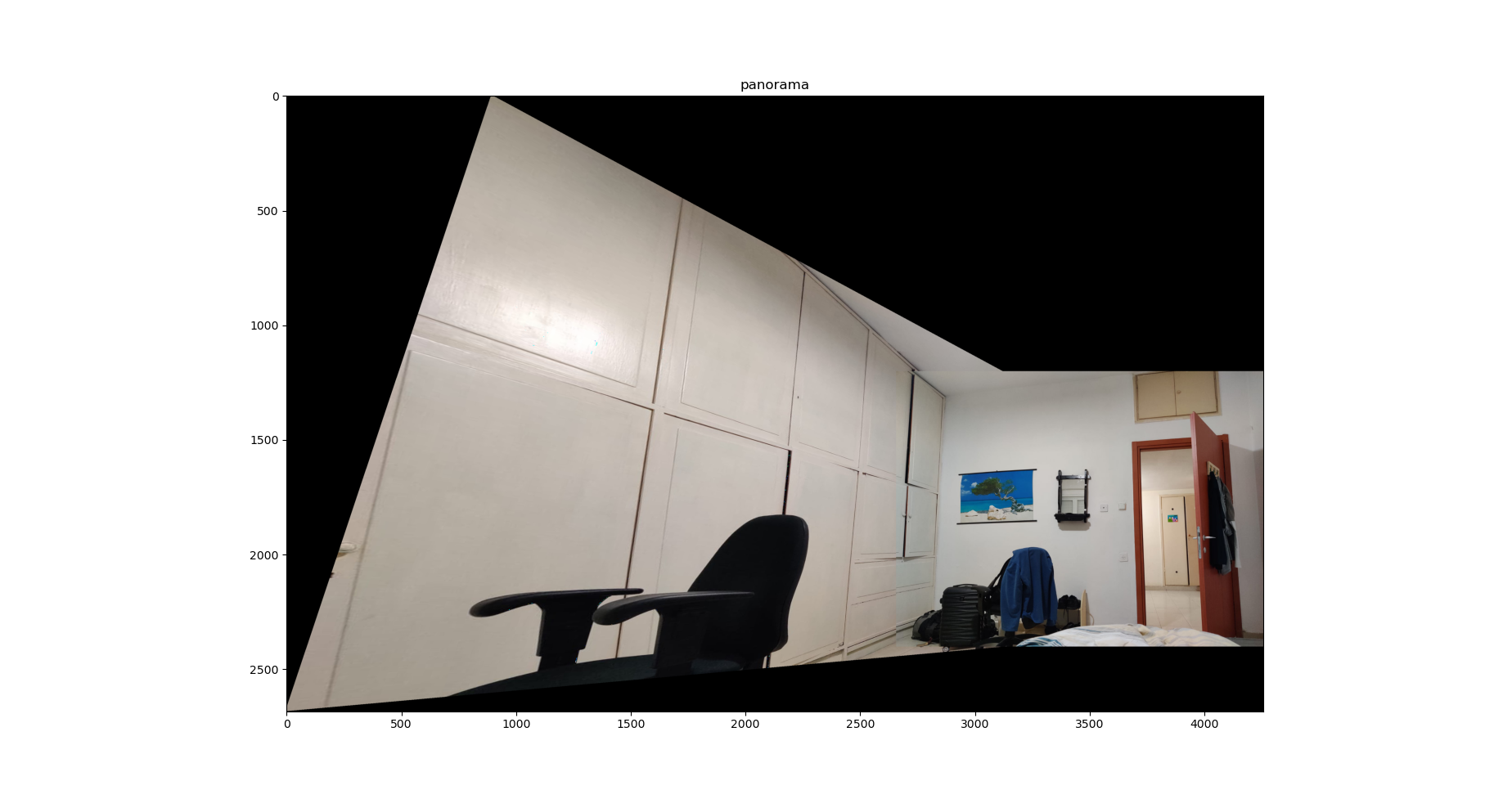
It can be seen that the overall result is very similar to that found in question 12, with the addition of the vertical black lines in the leftmost side of the image now being gone- this is expected, as backward mapping solves the problem of ‘holes’ appearing in the mapped image by going over each pixel in the destination coordinate system and assigns it a value, rather than values from the source image being mapped to points in the destination coordinate system as in forward mapping.

1. An implementation of add\_translation\_to\_backward\_homography is given.
2. An implementation of panorama is given.
3. Calling the panorama function with the matching points from the matches.mat file, while setting the inlier percent to 80% and the maximum error to 25 pixels, gives the following result:

The panorama image obtained presents a good fitting of the key points. We note that the result can be further improved, for example by smoothing the area around the stitching line to create a more ‘natural’ look.

1. We use the following source and destination images:

Note that most points are inliers, with a few intentional outliers picked (for example on the cloud edges).

We obtain the following panorama image:

As we can see, the panorama is relatively good- the source image is transformed such that the match points are well-aligned with those in the destination images.

The stitching is not perfect, as on closer examination one can notice the change in brightness between the images, as well as a slight mismatch in the top and bottom of the destination image (where the closet connects to its continuation in the source image), but it demonstrates the principles of image stitching well.