Exercise 3 – Features, Transformations, RANSAC

Eyal Waserman
305210189

Omer Sofer
201507340

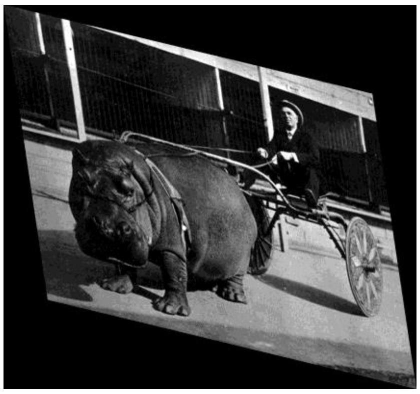
20.06.2017

1. We implemented the `ComputeProjective` function in the `ComputeProjective.m` file.

Example:

Original:

Transformed:



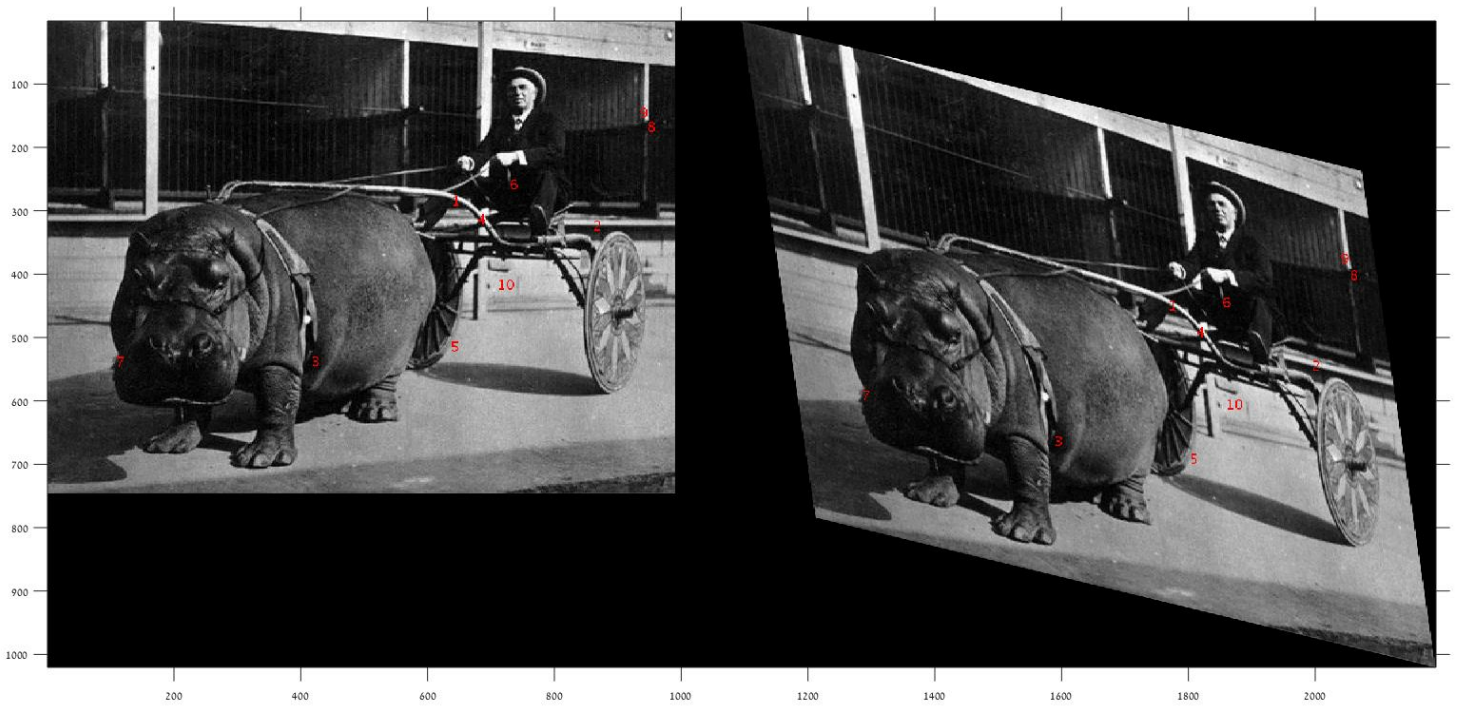
The transformation used is $H_{trans} = H_{rot} \cdot H_{skew}$, where:

$$\theta = \frac{5}{100}\pi, \qquad H_{rot} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad H_{skew} = \begin{bmatrix} 1 & 0.4 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

```
H_trans =

    0.9877    0.2386         0
    0.1564    1.0503         0
         0         0    1.0000
```

2. We implemented the `match` and `DisplayCorr` functions.

Here is an example, run on the results of sec.1, with `x=10` and `distRatio = 0.5`:



Explicitly describing top matches in ascending ratio:

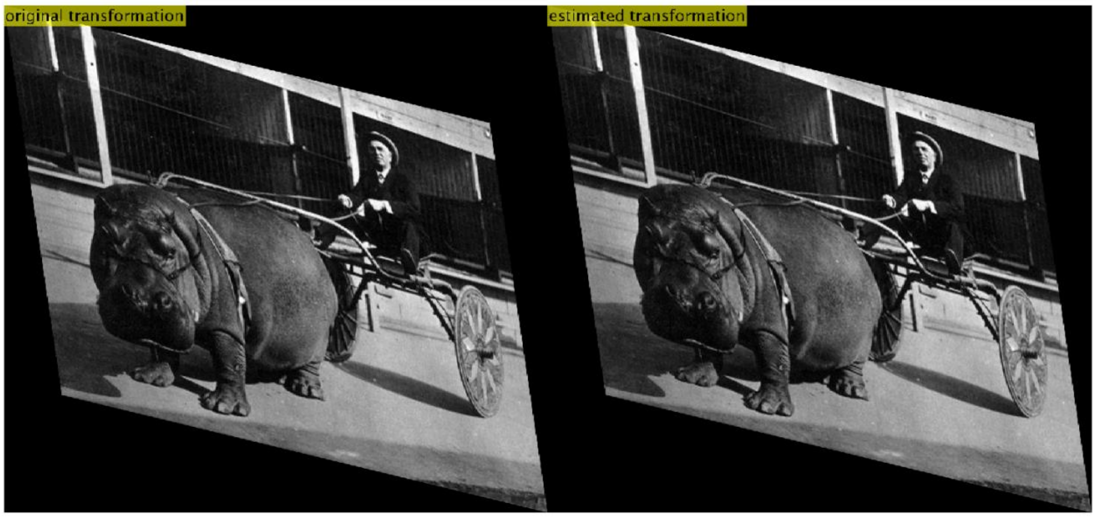
```
635.96,269.17 -> 670.58,434.48
859.84,307.92 -> 897.61,528.81
414.92,522.10 -> 491.83,647.72
676.75,298.34 -> 715.79,475.50
634.54,499.26 -> 704.92,676.44
727.97,242.89 -> 757.29,428.92
106.64,523.05 -> 187.56,575.75
945.10,152.48 -> 957.70,386.06
933.36,130.37 -> 942.50,359.81
708.26,400.57 -> 762.41,590.11
```

3. We implemented the DLT algorithm function.

For the matches received for the Hippo picture we received this following resulting H:

```
H_comp =

    0.9881    0.2389    0.0000
    0.1560    1.0496   -0.0000
    0.4058    0.4948    1.0000
```

We can qualitatively compare the result by transforming the original Hippo picture using the H_comp transformation, and checking whether the pictures look the same:



The above image shows that qualitatively the DLT estimation worked well, as to the eye, the transformations look similar.

Evidently, no optimization is required.

4. We implemented the `ComputeError` function and the `ComputeTestPoints` function as requested.

For example, the H_comp from previous DLT section produced an error of:

```
error_dlt =

    0.7933
```

5. We implemented the RANSAC_Wrapper as was requested.

```
function [H_ransac] = RANSAC_Wrapper(matches, fittingfn, distfn,...
                                    degenfn, s, t, feedback,...
                                    maxDataTrials, maxTrials)
    [H_ransac, ~] = ransac(matches',...
        fittingfn, distfn, degenfn, s, t, feedback, ...
                                maxDataTrials, maxTrials);
end
```

As seen above, the wrapper itself is not the interesting part of reconstructing the transformation.

- Since “matches” is a matrix which each of its rows is a data sample, and `ransac` expects such a matrix with columns as samples, a transpose was made.
- Furthermore, `ransac` returns the set of inliers which conform to the model `H_ransac`, but the wrapper has no use with it, hence the “~”.

Returned values adaptation:

Model	The model which the ransac algorithm should recover in this scenario is a 3x3 matrix representing the affine2d transform (`H_ransac`).

Arguments adaptation:

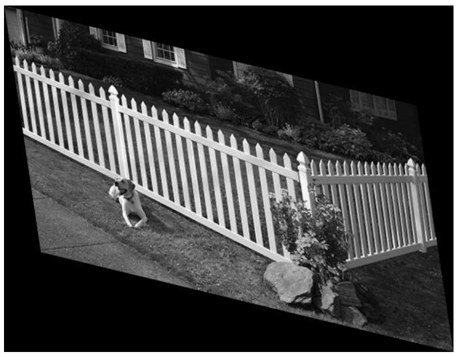
x	The data set. In our case the data set x is the set of matches. Each match is a 4-dimensional data point, which can conform to a model (a transformation) or not to.
fittingfn	Use with the handle `@ransac_fitting` for estimating a model out of a set of data points. The `ransac_fitting` function we implemented uses the built-in `estimateGeometricTransform` to extract an affine transform.
distfn	Use with the handle `@affine_trans_dist` which classifies each match as an inlier or an outlier. The way to do so is to calculate the Euclidean distance between the originally transformed point (RHS of the match), and the point calculated by transforming the original point (LHS of the match) with the proposed model. A point is considered an inlier when this Euclidean distance is smaller than the threshold and an outlier otherwise.
degenfn	This function should check for degeneracy of the dataset. Since we assume degeneracy will not happen, we provide the anonymous function `@(x) 0`.
s	The number of data samples. Although 6 is the minimal amount of samples to reconstruct an affine transform, we provide 10 in order to get better results.
t	Threshold. Although can vary, we often chose 10 as the threshold. Remember that 1 is the distance between neighboring pixels in the same row or column. We imagine that if a pixel is 10 pixels’ distance away from its intended location, than it should be classified as an inlier.
Feedback	Default 0.
Max Data Trials	Default 100.
Max trials	Default 1000.

5.a. Example when RANSAC does not give an improvement compared to the DLT:

To re-compute the results please run `Example1.m` in the handout directory. The script includes all parameters used for running DLT and RANSAC and will display them too.

Original:

Transformed:



The transformation used is $H_{trans} = H_{rot} \cdot H_{skew}$, same as question 1.

$$\theta = \frac{5}{100}\pi, \qquad H_{rot} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad H_{skew} = \begin{bmatrix} 1 & 0.4 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

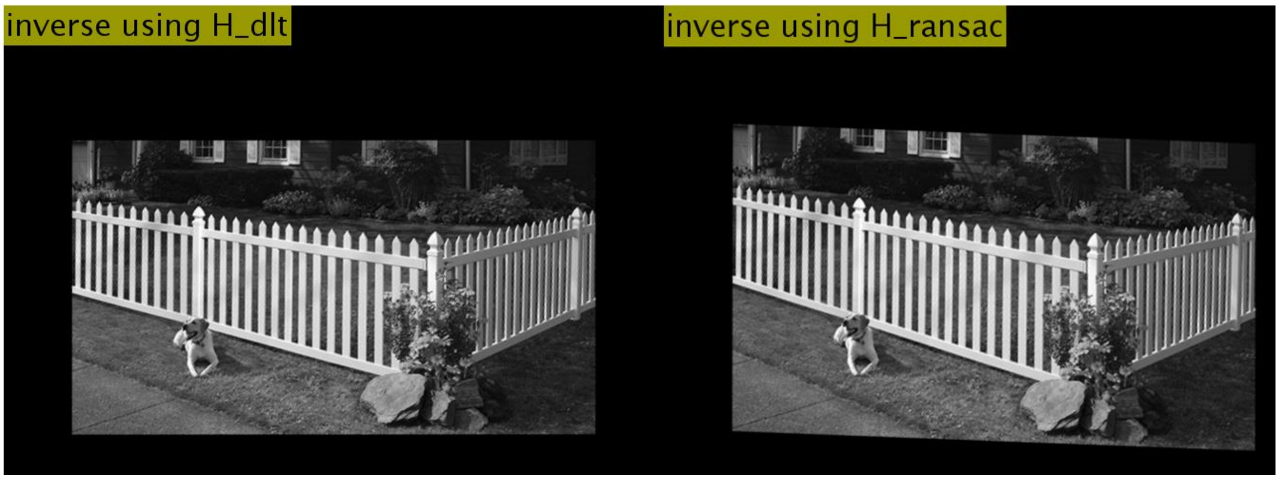
```
H_trans =

    0.9877    0.2386         0
    0.1564    1.0503         0
         0         0    1.0000
```

Since this image has many repetitions we assume that the overall matching will contain multiple incorrect matches. Incorrect matches are essentially outlier data points. The more contaminated the data is, the more iterations it is expected for the RANSAC to produce a good result. Given that the RANSAC also has an upper bound on the number of tries; it might finish the run with a non optimal result.

This can be shown quantitatively by comparing the error ratio:

```
DLT error: 0.405, RANSAC error: 44.4
RANSAC error is 10963.363% of DLT error
```

This can be also shown qualitatively by comparing the inverse transforms (i.e. the reconstructed image):



It is noticeable how in the DLT inverse the picture becomes horizontal again and resembles the original. For the RANSAC inverse, it is not horizontal, and features a higher error.

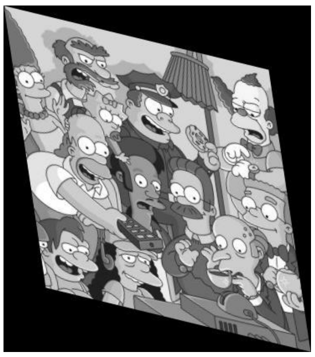
Note that this example is rare and hand-picked, most of the times RANSAC gives better results than this.

5.b. Example when RANSAC does give an improvement compared to the DLT:

To re-compute the results please run `Example2.m` in the handout directory. The script includes all parameters used for running DLT and RANSAC and will display them too.

Original:

Transformed:



The transformation used is $H_{trans} = H_{rot} \cdot H_{skew}$, same as question 1.

$$\theta = \frac{5}{100}\pi, \qquad H_{rot} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \qquad H_{skew} = \begin{bmatrix} 1 & 0.4 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

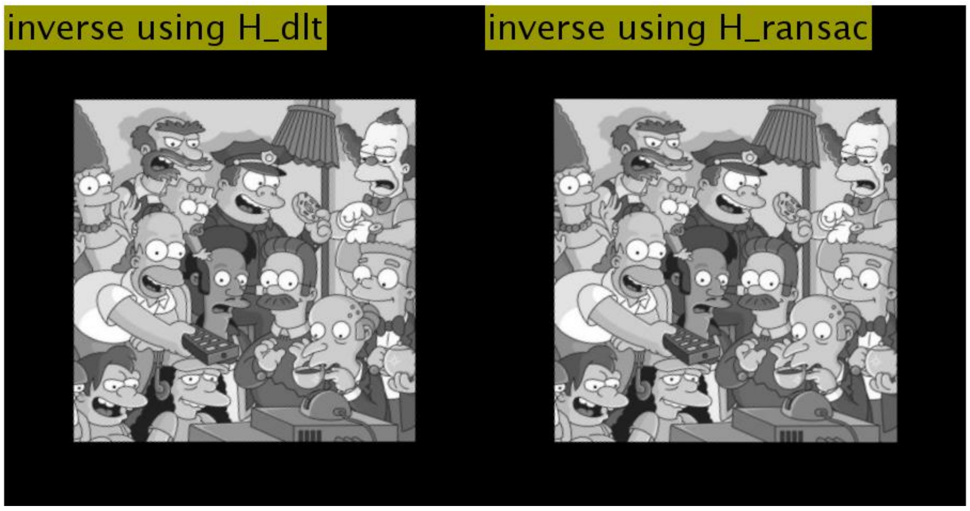
```
H_trans =

    0.9877    0.2386         0
    0.1564    1.0503         0
         0         0    1.0000
```

Since this image has many unique descriptors we assume that the overall matching will contain few incorrect matches, we assume that the RANSAC will produce a good result before it reaches its upper bound of the number of tries.

This can be shown quantitatively by comparing the error ratio:

```
DLT error: 15.3, RANSAC error: 2.93
RANSAC error is 19.166% of DLT error
```

This time, qualitative comparison doesn't work. For the naked eye, both reconstructions seem the same.



Note that because of the random nature of the RANSAC algorithm, Example1.m might show cases where RANSAC improves upon DLT, although many times it is worse. The same for Example2.m, sometimes the DLT improves upon the RANSAC, although most of the times it doesn’t.