Computer Vision Assignment 4

Luca Ebner (ebnerl@student.ethz.ch)

October 29, 2020

When trying to estimate parameters from a given dataset, least-squares approaches can lead to huge errors if the data is noisy. To cope with this problem, the RANSAC (RANdom SAmple Consensus) method can be used. To compare the errors of least-squares fitting and the RANSAC method, we start by implementing a line fitting algorithm from a given dataset of 2D points.

1. Line Fitting

The RANSAC pseudo code to find the best line fit to the dataset is as follows:

```
1 for i=1:N
2     points = random_choice(dataset,2)                %randomly select 2 points from dataset
3     line = fit_line(points)                          %fit line to these points
4     num_inliers = compute_inliers(dataset,line,tol) %number of points within tolerance
5     if num_inliers > best_num_inliers
6         best_num_inliers = num_inliers
7         best_line = line                             %update best line fit
8     end
9 end
```

The algorithm stops after N iterations and yields the best line fit it has found. Note that in Line 2 when randomly choosing points to fit a line one could also use more than two points. After N iterations one could optionally again do a least-squares fit of the inliers of the best line fit to get an even better result. Figure 1 shows a result comparison between ordinary least-squares fitting and a RANSAC approach.

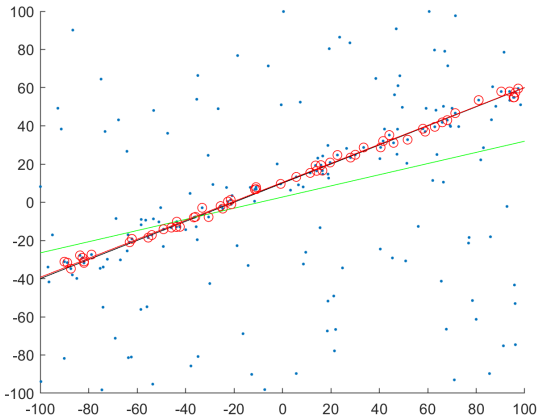

Figure 1: Comparison of fitting algorithms: Real Line (black), Least-Squares (green) and RANSAC (red).

If we analyze the error measurement outputs provided in the code framework, weg get:
`err_real = 35.3719, err_ls = 153.2209, err_ransac = 35.5657`
Note that the error resulting from least-squares is around 5 times bigger than with RANSAC!

2. Fundamental Matrix Estimation

Of course the RANSAC approach can not only be used to fit lines, but also to estimate the fundamental matrix $\mathbf{F}$ between two cameras from given point correspondences. The underlying equation here is

$$\mathbf{x_2^T F x_1} = 0 \tag{1}$$

$$= \begin{bmatrix} x_1x_2 & x_2y_1 & x_2 & x_1y_2 & y_1y_2 & y_2 & x_1 & y_1 & 1 \end{bmatrix} * vec(\mathbf{F}) = 0 \tag{2}$$

In order to solve this equation for $\mathbf{F}$, one needs at least 8 point correspondences. Note that $\mathbf{F}$ is a 3×3 matrix and has 9 entries, but since it is scale invariant we only need 8 equations.
Of course our point correspondences do not match *exactly*, which is why we stack the equations for each point correspondence in a matrix and use singular value decomposition to compute $\mathbf{F}$.
To increase numerical stability, the correspondence points are first standardized by subtracting the centroid and dividing by the standard deviation. Followingly, to compute the real fundamental matrix $\mathbf{F}$, we have to perform a similarity transformation ($\mathbf{F} = \mathbf{T_2^T F_n T_1}$), where $\mathbf{T_1}, \mathbf{T_2}$ are the transformation matrices to standardize the datasets ($\mathbf{\hat{x}_i = T_i x}$) and $\mathbf{F_n}$ is the fundamental matrix resulting from the singular value decomposition using the standardized point correspondences.

```
% standardized points
stdx = std(points(1,:)); % standard deviation in x
stdy = std(points(2,:)); % standard deviation in y
T = [1/stdx,     0,    -centroidx/stdx; % transformation matrix
        0,    1/stdy,  -centroidy/stdy;
        0,       0,           1       ];
```

```
A = [x1.*x2, x2.*y1, x2, xy.*y2, y2.*y1, y2, x1, y1, ones(size(nx1s,2),1)];

[U,S,V] = svd(A);
nFvec = V(:,end); % the most right column corresponds to the nullspace vector, which ...
    solves the equation A*nFvec = 0
nF = [nFvec(1:3)';
      nFvec(4:6)';
      nFvec(7:9)'];

F = T2'*nF*T1;
scale = F(3,3);
F = F/scale;
```

If we take the found $\mathbf{F}$ and draw the epipolar lines on the images, we notice that these lines do not properly intersect in singular points, namely the epipoles. This is due to the fact that our measurements are noisy. Since a *real* fundamental matrix should satisfy $det(\mathbf{F}) = 0$, we have to enforce that constraint by manually setting the smallest eigenvalue, which already is very close to zero, to zero. Figure 2 shows the difference between before and after enforcing the singularity constraint. The complete set of the epipolar line plots with enforced singularity constraint to the test image sets can be found in Figure 5 in the Appendix.

```
[U,S,V] = svd(F);
S(3,3) = 0; % enforce singularity
Fh= U*S*V'; % the 'real' fundamental matrix
```

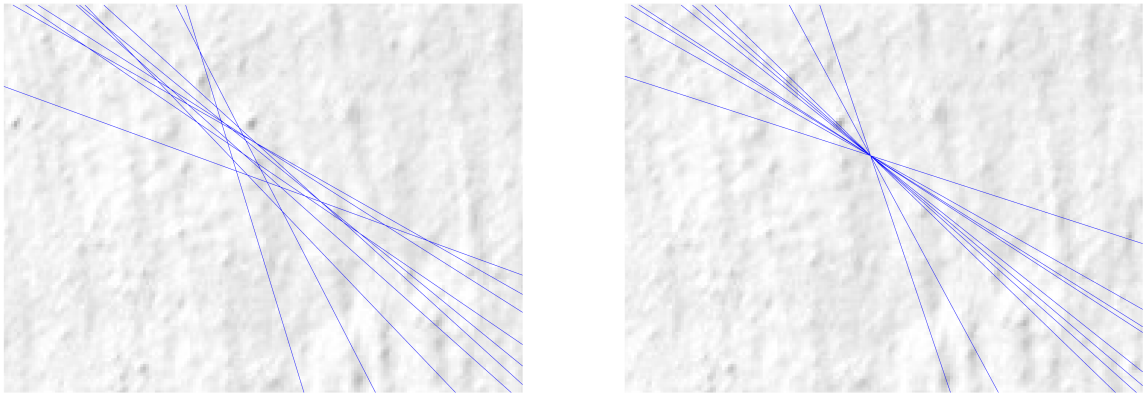

Figure 2: Comparison between fundamental Matrix F without (left) and with (right) enforced singularity contstraint. When we zoom in on the intersection area of the drawn epipolar lines in the "rect" test image set we can see that if we don't force singularity the epipolar lines (blue) do not properly intersect in the epipole, where with the forced singularity constraint they indeed do.

3. Feature Extraction and Matching

For the installation of VLFeat the instructions on `https://www.vlfeat.org/` were followed. The image SIFT feature points can then be extracted with

```
1 [fa, da] = vl_sift(img1);
2 [fb, db] = vl_sift(img2);
```

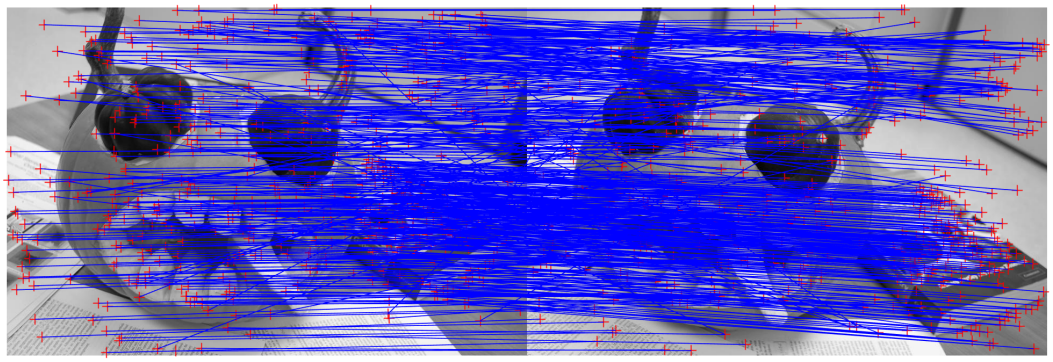

Figure 3: Raw extracted SIFT feature matches of the "pumpkin" test image set. Note that the raw matches contain several wrong matches.

4. Eight-Point RANSAC

Our goal now is to find the best fundamental matrix model for a given image set. This time, instead of manually clicking correspondence points we use the previously found SIFT feature matches as our dataset. As we saw, the SIFT feature matches contain a lot of errors, which is why we use the RANSAC algorithm.

4.1 Simple RANSAC

In the simple RANSAC algorithm we do a fixed number of iterations denoted with `iter`. A visualization of the correct feature matches using the best found fit from the RANSAC algorithm can be seen in Figure 4. The algorithm looks as follows:

```
for i=1:iter
    % Randomly select 8 points and estimate the fundamental matrix using these.
    rand_indices = randsample(num_pts, sampleSize);
    x1s = x1(:, rand_indices);
    x2s = x2(:, rand_indices);
    [Fh, F] = fundamentalMatrix(x1s, x2s);

    % Compute the error.
    distances1 = distPointsLines(x2, Fh*x1);
    distances2 = distPointsLines(x1, Fh'*x2);
    error = (distances1 + distances2) / 2;

    % Compute the inliers with errors smaller than the threshold.
    inlier_indices = [];
    inlier_indices = find(error < threshold);
    num_inliers = numel(inlier_indices);

    % Update the number of inliers and fitting model if the current model is better.
    if num_inliers > best_num_inliers
        best_num_inliers = num_inliers;
        best_F = Fh;
        best_inliers = [];
        best_inliers = inlier_indices';
    end
end
```

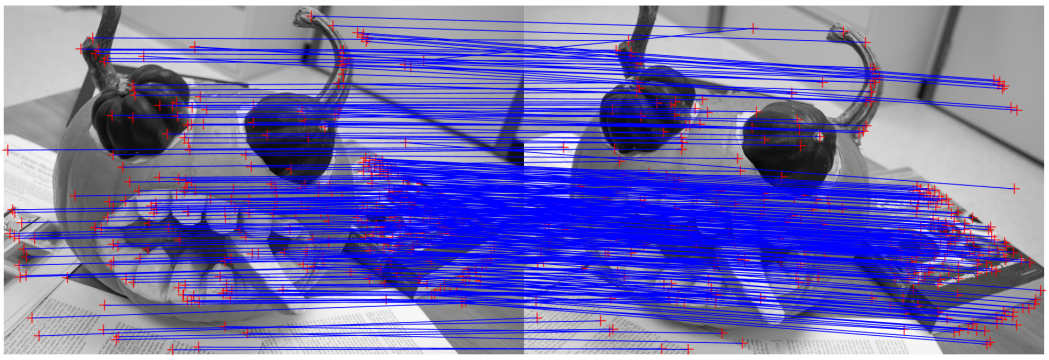

Figure 4: Visualisation of the inlier feature matches in the best found model of the RANSAC algorithm for the test image set "pumpkin". Note that it contains way less errors than its original in Figure 3.

The criterion whether a point correspondence is counted as an in- or outlier is set by saying that the error

$$\mathcal{E}(x_1, x_2) = \frac{d(x_2, Fx) + d(x_1, Fx_2)}{2} \tag{3}$$

has to be smaller than a certain `threshold` (see also in code snippet above, Lines 8-11). The table below shows a comparison of the results when varying the threshold in the test image set "pumpkin".

threshold	total inliers	mean error of inliers
2	333	0.7753
5	396	1.7912
10	424	3.9248

For the following exercise, we set the threshold equal to 2 pixels.

4.2 Adaptive RANSAC

Theoretically, one can only be sure to have found the very best solution if one has tested every single hypothesis combination, in this case every single combination of 8 point subsets. Of course this would computationally be very expensive, which is why in practice it is not done.
Instead, one either uses a fixed number of iterations like in the subsection 4.1 or one terminates the iteration if it is known with probability p that at least one of the random samples of N points (8 in this case) out of M trials is free from outliers. The equation for p is then given as

$$p = 1 - (1 - r^N)^M, \tag{4}$$

what leads to

$$M = \frac{\log(1-p)}{\log(1-r^N)}, \tag{5}$$

where r is the largest inlier ratio found so far at every RANSAC iteration.
This means that we only have to do M iterations to get the desired result. Followingly, in code we can update `r, M` after every iteration and then check if the current iteration index `m` exceeds `M`. If yes, the iteration loop can be stopped.

```
%for i=1:iter % simple RANSAC
while(m<M) % adaptive RANSAC

    ...

     if num_inliers > best_num_inliers

         ...

         r = best_num_inliers/num_pts; % adaptive RANSAC
         M = abs(round(log(1-p)/log(1-r^sampleSize)));
     end

     m = m+1;  % adaptive RANSAC
end
```

In the table below an overview is provided about which test image set needed how many iterations to satisfy the condition $p = 0.99$. Note that M changes each time the algorithm is run since the samples are random.

image pair	M
ladybug	7607
rect	301802
pumpkin	851

A reasonable question to ask is why the number of iterations differs so much between the image sets? This can be explained by looking at the raw SIFT feature matches. A closer inspection shows that the image sets with a higher ratio between wrong and total matches need more iterations, which actually makes sense. In other words: If your dataset contains more errors, the probability to draw a good sample is lower and you need to draw much more samples in order to find a good model.

5. APPENDIX

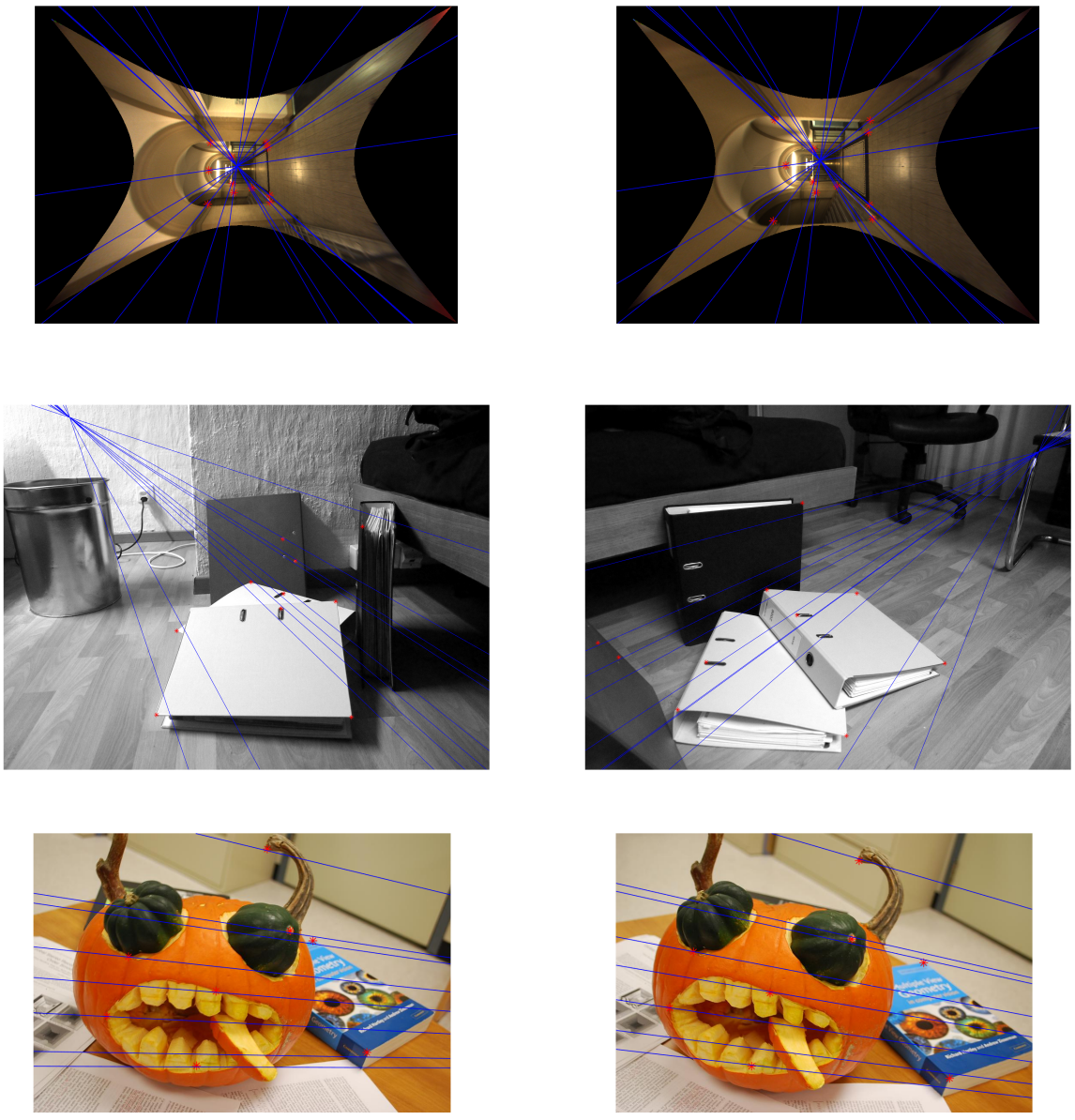

Figure 5: Complete set of epipolar line plots to the test images (with enforced singularity constraint).