Write-up for Problem Set 3 of CMSC 733

Jason Filippou

October 26, 2014

Contents

1 Expectation - Maximization for line fitting 1
1.1 Pure line fitting . . . 1
1.2 Full E-M . . . 2
1.2.1 'E' step . . . 3
1.2.2 'M' step . . . 3
1.2.3 Results . . . 3

2 Mosaicing with RANSAC 6
2.1 SIFT feature extraction . . . 6
2.2 Three best matches . . . 6
2.3 Affine transformation . . . 7
2.4 RANSAC . . . 7
2.5 Stitching . . . 8
2.6 Mosaicing . . . 8
2.7 Results on custom images . . . 9
2.7.1 Dining room images . . . 9
2.7.2 Our room . . . 9

1 Expectation - Maximization for line fitting

In this section, we analyze our implementation of the first part of the problem set.

1.1 Pure line fitting

This is essentially the M-step of E-M, where all weights are 1. As such, it can be easily transformed to boil down to linear regression over a set of 2D variables. Specifically, let **D** be a 2 x n matrix of 2D data points, where the data points are in the columns:

$$\mathbf{D} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \\ y_1 & y_2 & \dots & y_n \end{bmatrix}$$

This is the input to our method. Then, we can form the vector $\mathbf{y} = [y_1\ y_2\ \ldots y_n]^T$ from the second row of D and the 2 x n matrix $\mathbf{X}$ from the first row and a vector of all 1's:

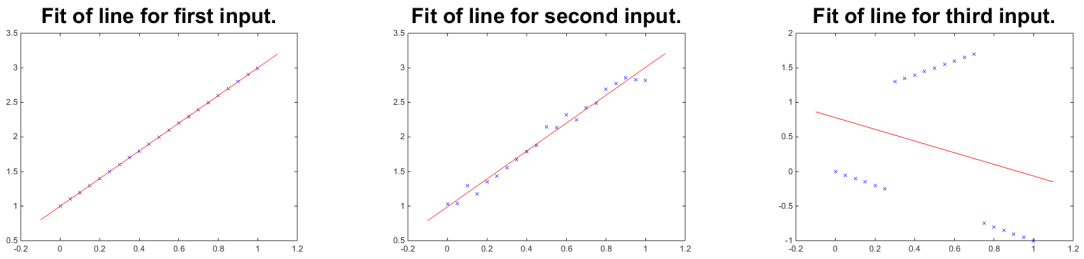
$$\mathbf{X} = \begin{bmatrix} x_1 & x_2 & \ldots & x_n \\ 1 & 1 & \ldots & 1 \end{bmatrix}$$

In this form, the desired parameter vector $\hat{v} = [a\ b]^T$ is the solution to the following linear regression problem:

$$\hat{v} = (XX^T)^{-1}Xy \tag{1}$$

which, if properly unpacked, will convince the reader that it has exactly the same form as the one presented in Yair Weiss' notes.

Our results for the data provided are shown in figure 1. As shown in figure 1(c), for even simple, artificial datasets, linear regression can often produce fits far away from the optimal fit (which cannot be linear, in this case).



(a) (b) (c)

Figure 1: Results for pure line fitting.

The code for this section can be found in the MATLAB function `fit_line.m`. Note that, in that function, we transform the matrix $\mathbf{X}$ such that the patterns are in the rows instead of the columns, and this changes equation 1 to

$$\hat{v} = (X^TX)^{-1}X^Ty$$

1.2 Full E-M

We now show how we can iteratively find a pair of good linear fits for a dataset of 2D points using the E-M algorithm, and examine certain cases where the algorithm has trouble finding good fits because of noisy data. We analyze the E and M steps in isolation and then we show our results when both steps are combined.

1.2.1 'E' step

In this step, the parameters of the lines $[a_1\ b_1]^T,\ [a_2\ b_2]^T$ are assumed to be known, and we estimate the n-dimensional weight vectors $\mathbf{w_1}$ and $\mathbf{w_2}$. We follow Yarr Weiss' notes on this. First, for $i \in \{1, 2\}$, we compute the residual vectors:

$$\mathbf{r_i} = a_i \cdot \mathbf{x} + b_i - \mathbf{y} \tag{2}$$

and then pass them through a softmin function to compute the relevant weight vectors:

$$\mathbf{w_i} = \frac{e^{-\mathbf{r_i}^2/\sigma^2}}{e^{-\mathbf{r_1}^2/\sigma^2} + e^{-\mathbf{r_2}^2/\sigma^2}} \tag{3}$$

The code for the E-step is available in the MATLAB function `Estep.m`.

1.2.2 'M' step

The 'M' step consists of solving weighted linear least squares [1]. The form of the problem is very similar to equation 1. If we form the n x n diagonal matrix $\mathbf{W}$ with the weights of every point along its main diagonal:

$$W = \begin{pmatrix} w_1 & & & \\ & w_2 & & 0 \\ & & \dots & \\ & 0 & & \dots \\ & & & w_n \end{pmatrix}$$

Then we would need to solve the equation:

$$\hat{v} = (XWX^T)^{-1}WXy \tag{4}$$

The code is available in the function file `Mstep.m`.

1.2.3 Results

In this section, we show our results on applying E-M in both the data used in section 1.1 as well as data that has been altered by noise.

[1] `http://en.wikipedia.org/wiki/Least_squares#Weighted_least_squares`

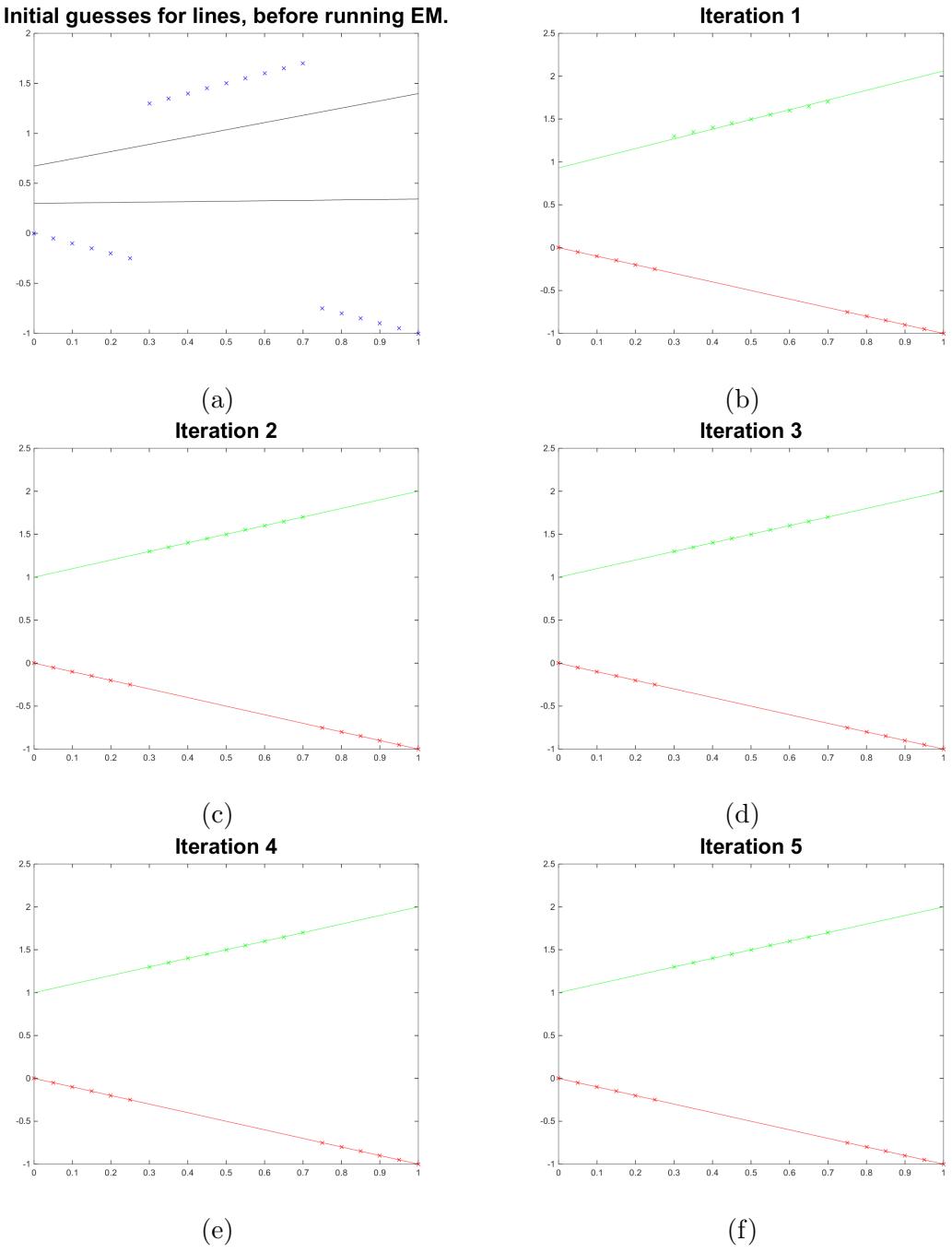


Figure 2: The convergence of E-M over 5 iterations on the data from part 1. Best viewed in color.

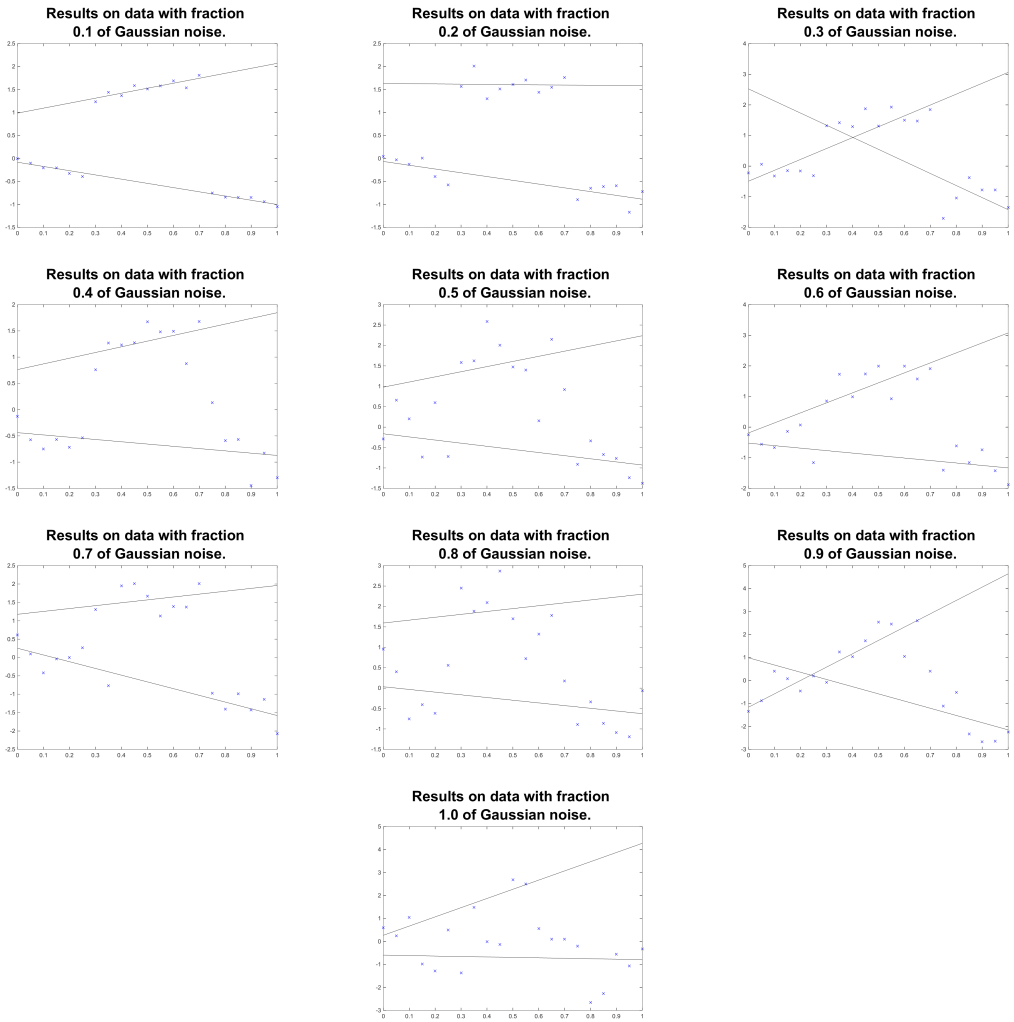


Figure 3: Estimating the impact of noise on the result of E-M.

Figure 2 shows the convergence of EM on the dataset from section 1.1 where straightforward line fitting produced a low fidelity result (figure 1(c)). Initially, the algorithm begins with a guess on what the lines should be and very quickly (in just two iterations, it seems) converges to a pair of lines which represents the data pretty well. The color-coding on the figures represents the points assigned to the two lines (latent assignment variables $\mathbf{z}$).

We also experimented with adding noise to the same dataset in order to estimate its impact on the accuracy of the algorithm. We experimented with fractions of Gaussian white noise generated with $\sigma = 1$. Figure 3 shows the results. It's not hard to see that a significant amount of noise causes EM to be unable to converge to a qualitatively acceptable solution.

2 Mosaicing with RANSAC

In this section, we outline our approach for dealing with part 2 of the problem set, mosaicing a pair of images with RANSAC.

2.1 SIFT feature extraction

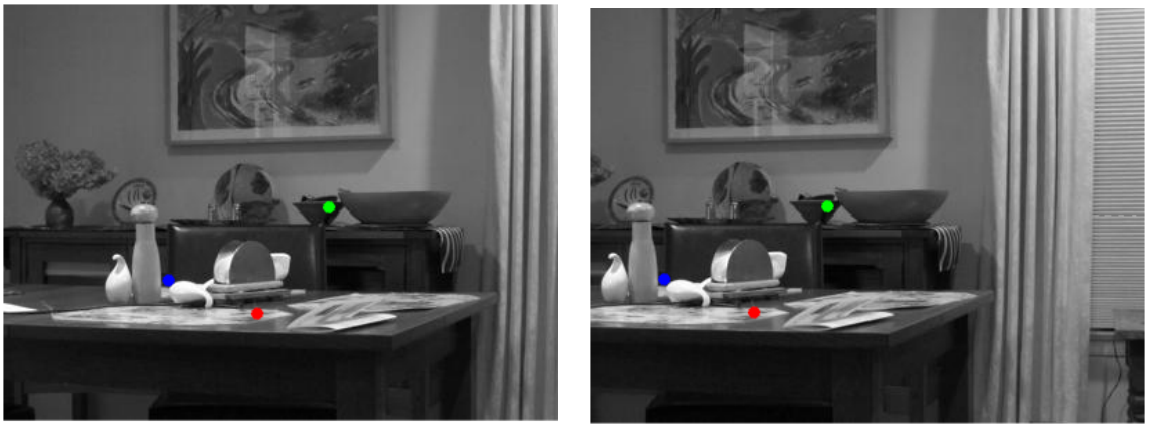
We first prove that `VLFeat` is running correctly. Figure 4 visualizes the SIFT features extracted by `VLFeat`. The `VLFeat` version used was 0.9.19, and it appears to be extracting denser features than previous versions of the library, perhaps because of less aggressive non-maximum suppression.

(a) (b) (c)

Figure 4: SIFT feature extraction on a simple intensity square(a) and the provided pictures (b, c).

To reproduce these results, run the function `part2` with parameter 1.

2.2 Three best matches



(a) (b)

Figure 5: Best three matches in terms of Euclidean Distance in SIFT space. Best viewed in color.

Figure 5 contains the color-coded matches between the three best matches that we computed based on SSD between the SIFT histograms. Function `find_best_match.m` implements the matching. To reproduce these results, run the function `part2` with parameter 2.

2.3 Affine transformation

Function `affine_transformation.m` uses three 2D point correspondences to solve for the affine transformation that relates them. The output of `part2(3)` is:

```
>> part2(3)

ans =

1.0e-15 *

   -0.0555    0.1110   -0.2220
   -0.4441   -0.1110   -0.2220
```

So the difference between the hard-coded affine transformation and the one found by our MATLAB routine is vanishingly small, and in the order of our system's numerical accuracy about zero.

2.4 RANSAC

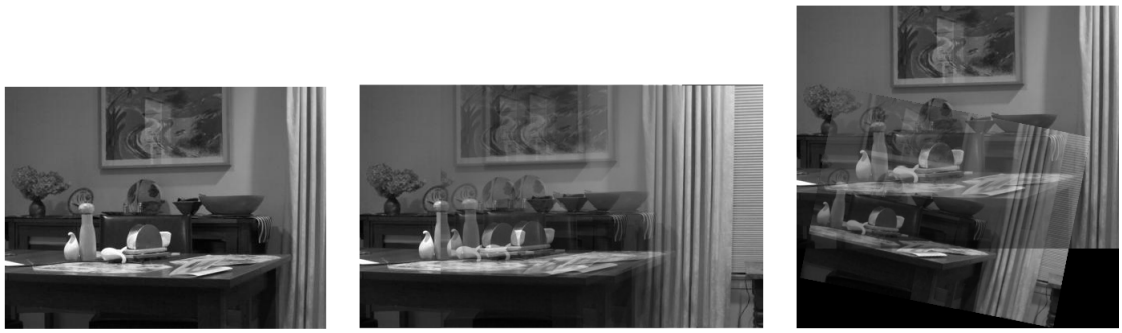
Our implementation of RANSAC is contained in MATLAB function `ransac.m`. In our experiments with the images provided, we ran RANSAC for 1000 iterations. Running `part2(4)` yields the same matrix for every call, for instance:

```
>> part2(4)

A =

   54.6484   69.4537   94.4210
   55.0828   92.7494   90.3263


Ares =

   54.6484   69.4537   94.4210
   55.0828   92.7494   90.3263
```

2.5 Stitching

For this part, we adapted stitching code available online [2] to our needs. The function `stitch.m` calculates both a stitched image as well as the enveloping mask, such that future stitchings of the resulting images can be made possible (an already existing mask is among the optional arguments to the method). The method uses the MATLAB built-ins `maketform` and `imtransform` instead of the recommended `affine2d` and `imwarp` since it needs to make use of the second and third return values of `imtransform` (`XData` and `YData`, respectively) to calculate the position of the transformed images in the common co-ordinate system of both images.

The entire code is available in `stitch.m`. Figure 6 contains the results of calling `part2(5)`, which stitches the original "left" part of the dining room image (LR1bw) and two affine transformations which we also had to implement (`translate100` and `trs`).



(a) (b) (c)

Figure 6: The result of stitching between (a) the original image, (b) Its translation at x = +100 and (c) A rotation by $\pi/16$, scaling by 0.8, and a translation of (50, 100). Note that the results are not aesthetically pleasing because the transformations in this section of the problem set have not been computed by RANSAC, but are statically provided to the stitching algorithm.

2.6 Mosaicing

Calling `part2(6)` performs our full image mosaicing algorithm. The result can be found in figure 7. We can see that the transformation recovered by RANSAC is a translation almost entirely along the x axis.

[2] Panorama Stitching in MATLAB.

(a) (b) (c)

Figure 7: Mosaicing images (a) and (b) yields (c).

2.7 Results on custom images

In this section, we perform mosaicing on custom images. Our results can be reproduced by running `part2(7)`.

2.7.1 Dining room images

The first image pair that we test on consists of the left part of the original dining room image, as well as its transformation induced by `trs`, which was already used in section 2.5. That is, we applied the transformation to the original image (using `affine2d` and `imwarp`), saved the resulting image under the subdirectory "figures", and then loaded both images to perform mosaicing on them. Figure 8 contains the results. It is evident that the results are quite aesthetically pleasing, up to some intensity blurring.

(a) (b) (c)

Figure 8: Mosaicing images (a) and (b) yields (c).

2.7.2 Our room

We also took a pair of pictures from a wall in our room in College Park. Figure 9 shows the original image in B & W. We divide this image into two portions which are overlapping by approximately 70% and then mosaic them. The results are shown in figure 10 and are quite pleasing, but for a small vertical white line approximately a third of the way from the left.

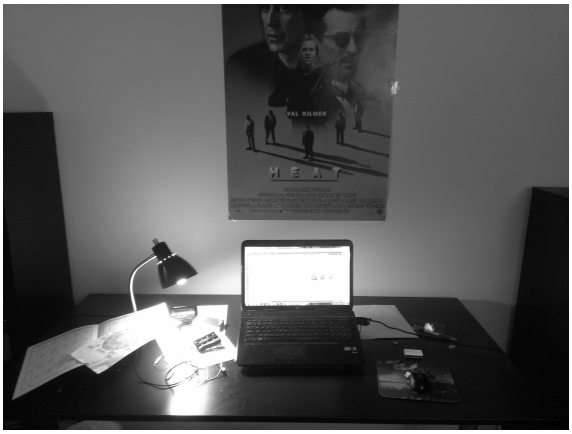
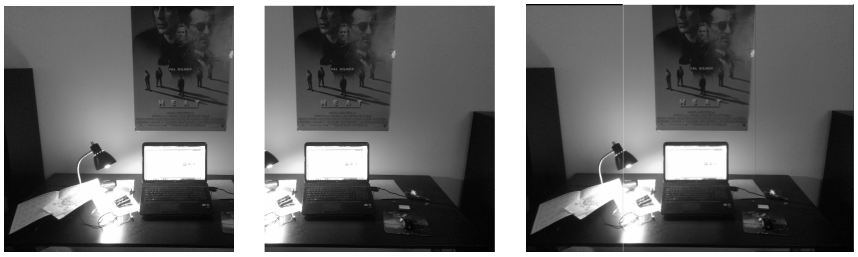


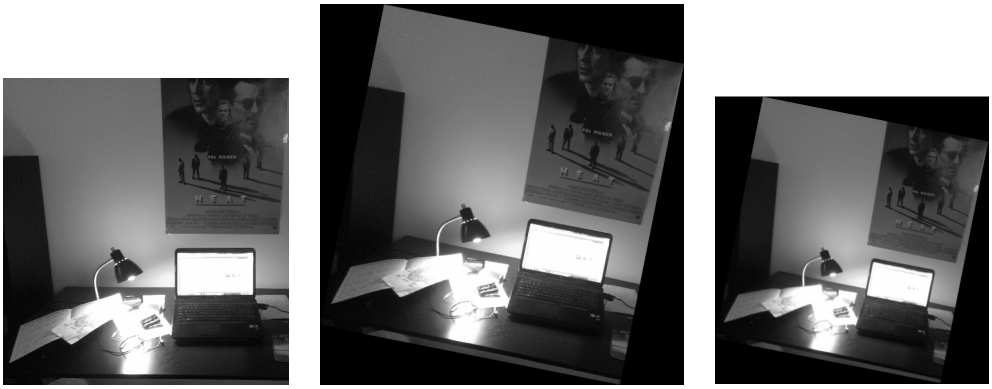
Figure 9: A picture of our bedroom wall.



(a) (b) (c)

Figure 10: Mosaicing images (a) and (b) yields (c).

We also experiment by applying the transformation `trs` to the image in figure 10(a) and mosaicing the two images. The results can be viewed in figure 11 and are also of high fidelity.



(a) (b) (c)

Figure 11: Mosaicing images (a) and (b) yields (c).