Least square optimization in 2D image alignment

Zili Wang

March 2016

1 Introduction

The purpose of this work is to get hand dirty on various transform, feature points detect and using optimization to calculate parameters of transform based on correspondence of point pairs.
The basic method is using photoshop to generate transform a image. Then using various type of feature point method to get the correspondence. And then I will use linear close form method to get the rotation, translation and scaling parameters. Also I will use nonlinear iterative method to get the rotation and translation, this time rotation is the function of rotation angle θ , so this is a nonlinear problem.

2 Feature Detection and Match

I tried various type of feature point: SURF, FAST, BRISK, Harris. In the end I choose SURF for all my works. The detection and match work in Matlab is very easy. But it takes the most of computational time in my pipeline.

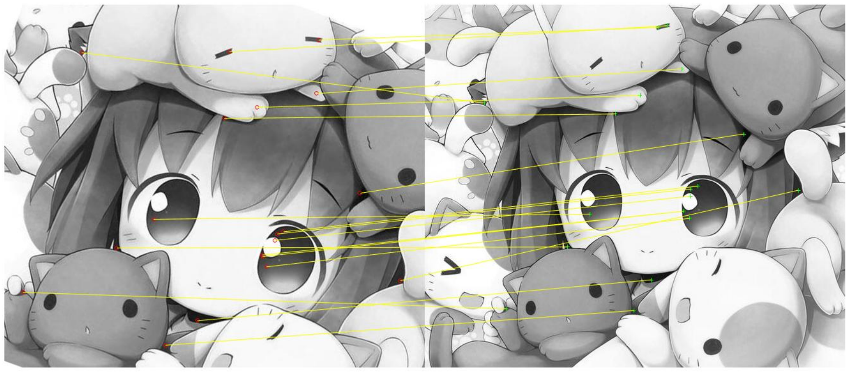
```
clear all;
oriImg =rgb2gray(imread('loli.jpg'));
modImg =rgb2gray(imread('loli_d.jpg'));
width = size(oriImg, 2);
height = size(oriImg, 1);
points1 = detectSURFFeatures(modImg,'MetricThreshold',10000);
[features1,validPoints1] = extractFeatures(modImg,points1);

points2 = detectSURFFeatures(oriImg,'MetricThreshold',10000);
[features2,validPoints2] = extractFeatures(oriImg,points2);

index_pairs = matchFeatures(features1,features2);

matched_pts1 = validPoints1(index_pairs(:, 1));
matched_pts2 = validPoints2(index_pairs(:, 2));
```

```
showMatchedFeatures(modImg,oriImg,matched_pts1,matched_pts2,'montage');
matchPtNumTotal=size(index_pairs,1);
%showMatchedFeatures(modImg,oriImg,matched_pts1,matched_pts2);
```

3 Linear Least Square

3.1 Algorithm

I apply rotation, translation and scaling to my image.



The transform model I use is as below:

$$x^{'} = A * x$$

$$A = \left[\begin{array}{ccc} \text{1+a} & \text{-b} & t_x \\ \text{b} & \text{1+a} & t_y \end{array} \right]$$

x is the feature position in original image, and $x^{'}$ is the corresponding feature position in transformed images. The cost function I want to minimise is:

$$E_{LS}(p) = \sum_i ||x_i^{'} - x_i||^2$$

, which sum over all feature point pairs. p is the vector contains unknown variables in A, which are a, b, t_x, t_y. So, the correct transform matrix should be:

$$\underset{p}{\operatorname{argmin}}(E_{LS}(p))$$

Here we need to express the function in linear form in term of p instead of x. With the fact that: $y = f(x) \implies y = f_x * x$, if f(x) is a linear function.
Also, if x is a vector, f_x should be the Jacobean of f(x).
So, we can express $x^{'} = A * x$ as $x^{'} = J_p * p$

$$J_p = \begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$$

Now, we have rewritten the problem into normal form of least square program:

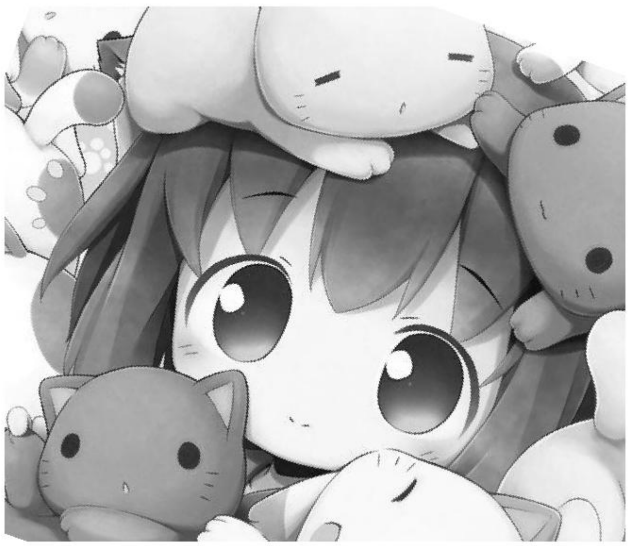
$$\underset{p}{\operatorname{argmin}}\left(\sum_i ||J_p(x_i) * p - x_i||^2\right)$$

3.2 RANSAC

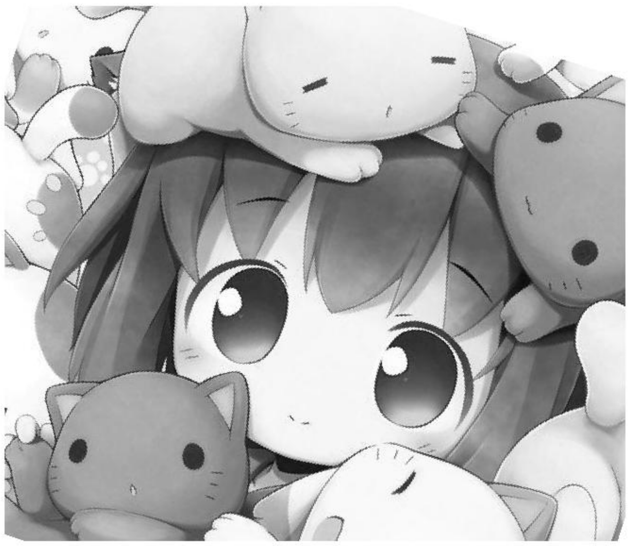
From the image with feature points correspondence, we can see there are outliers. To get rid of them, I apply the typical RANSAC. For the details of implementation, please refer to matlab codes. We can see the outlier (green points) are picked out from below:

3.3 Result

Below is the recovered image using RANSAC:



Below is the recovered image without using RANSAC:



4 Nonlinear Least Square

The rotation and translation is linear transform, but if we characterise the rotation in term of rotation angle, this actually is nonlinear problem.

4.1 Algorithm

I apply rotation, translation to my image.

The transform model I use is as below:

$$x^{'} = A * x$$

$$A = \begin{bmatrix} cos(\theta) & -sin(\theta) & t_x \\ sin(\theta) & cos(\theta) & t_y \end{bmatrix}$$

There is no way to reform the expression like in the linear case using Jacobean time parameter vector. The idea to solve this problem using Tyler extension to approximate the function, but we can only get good approximation nearby. Using first order Tyler extension, we convert the transform function to linear in term of incremental Δp. And change the cost function to:

$$E_{NLS}(\Delta p) = \sum_i ||J_p(x_i : p) * \Delta p - (x_i^{'} - x_i)||^2$$

This means that at this moment, I want a Δp that can drive the expected x to the real transformed x as near as possible. So, at each iteration, we offset the p by our best modification of p, and we hope in the end we can approach the best p. The most important thing is that we have a standard least square problem in term of Δp.

4.2 Levenberg-Marquardt Method

Based on the algorithm above, we solve this equation to get Δp:

$$A * \Delta p = b$$

And using $p = p + \Delta p$ to get new p for future iteration. But we want to control the size of step between each iteration, there are two way to do this:

1. $p = p + \lambda * \Delta p$
2. $(A + \lambda diag(A)) * \Delta p = b$

By adjusting λ, we can control the converging speed. And the second way is called Levenberg-Marquardt Method. The advantage of doing in this way, I still didn't get.

4.3 Result

The nonlinear iterative approach is really much trickier than the linear case. If we just simply casually choose a arbitrary initial value for p. Things are more like I will not get the p I want. So, we need some other method to get a rough result, and using nonlinear method to refine the result. And there two weird things I still don't quite understand:

1. In the case of we getting other local optimal in the end. Actually this isn't even the local optimal in term of $E_{NLS}(\Delta p)$. Basically, I can see the $E_{NLS}(\Delta p)$ decrease with the iteration, but at a certain point, it will increase again and the whole iteration will converge to larger E_{NLS}. Only when I set the proper initial value so the iteration can converge to the global optimal, the converge point and the optimal in term of E_{NLS} are consist. This makes sense if we consider we doesn't directly optimise E_{NLS}. But there should be some more clear explanation.
2. Even we I set the initial value properly, if I have outliers, it will still lead to other local optimal.

4.4 Summary on Various Nonlinear Optimisation Method

1. Gradient Descent: Using first order Tyler extension to approximate general nonlinear function.
2. Gaussian Method: Using second order Tyler extension to approximate general nonlinear function, the cost is we need calculate the Hessian.
3. Gauss-Newton Algorithm: First using first order Tyler to approximate the nonlinear part inside the L2-norm, hence converting this problem to linear Least Square problem. Avoiding the calculation of Hessian under the special case of L2-norm.
4. Levenberg-Marquardt Method: Based on Gauss?Newton Algorithm, add a way of controlling iteration speed.

Because the calculation of derivative of nonlinear function embedded inside a L2-norm is usually very nasty, so for the specially case we better use Gauss?Newton Algorithm, instead of directly applying general method on it.

5 Codes

All the related codes are on github: `https://github.com/ziliwangmoe/opti_least_square`

6 References

[1] Richard Szeliski, Computer Vision: Algorithms and Applications