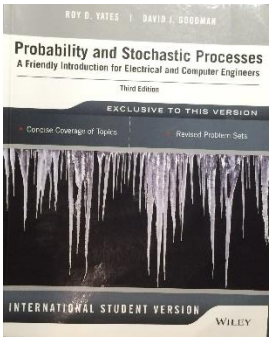
Computer Vision – Programming Project 3

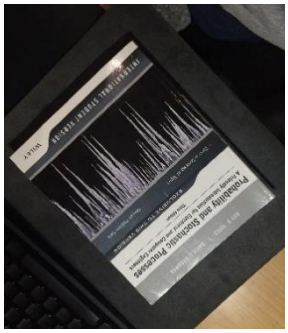
王順興　0210184

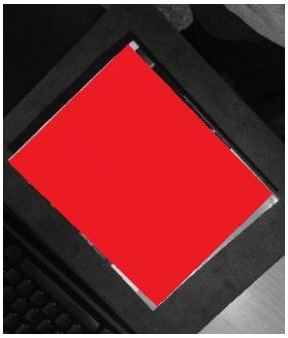
Introduction

In this project, I will be working on projection problem:



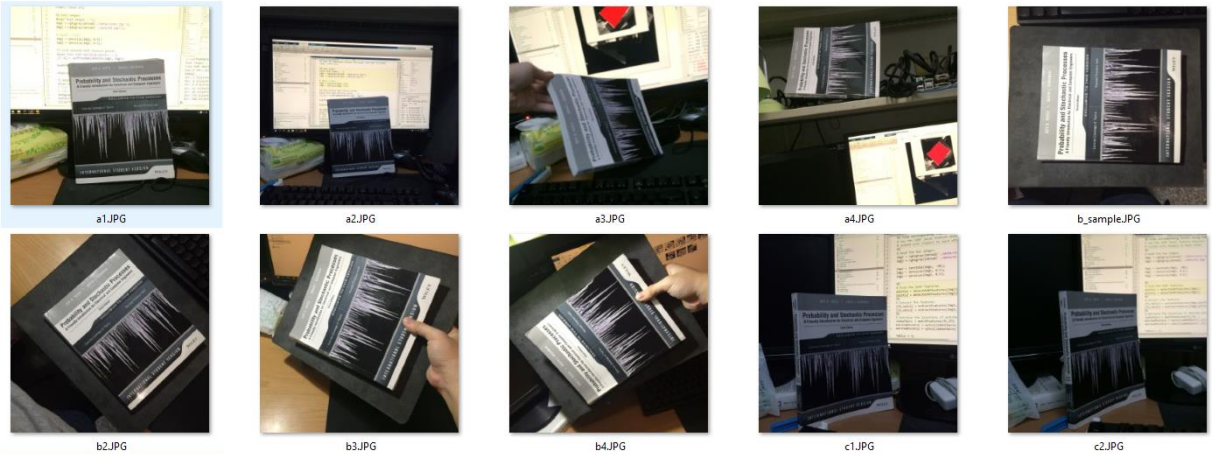
1. Given a **target** and a **pattern** to project onto it





2. Try to **project** the **pattern** onto the **target** in **another photo**

i. The Dataset



The dataset is a series of photos taken by me. The photos contain that target in various position/brightness/facing.

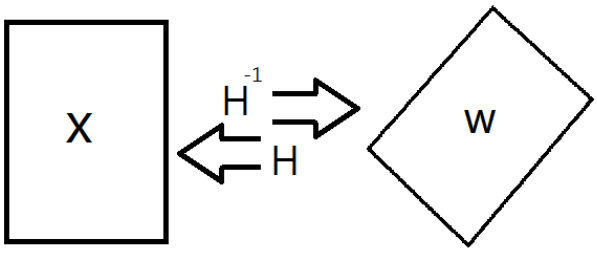
ii. Motivation

Projecting virtual planar objects onto physical surfaces is the basis of augmented reality. However, the technology is probably most used on sport broadcasting:

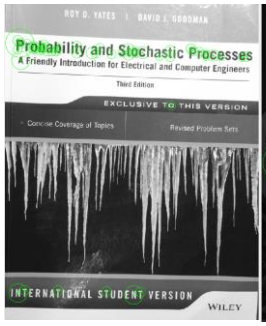
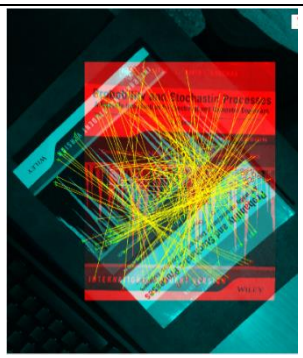
My goal is to see if I can have these kind of projections using the methods we've learnt. Though just by seeing these two images, I knew that they're probably not using methods similar to mine.

Method

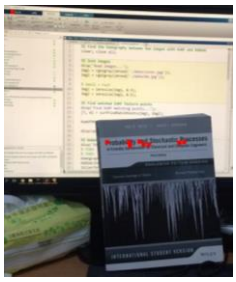
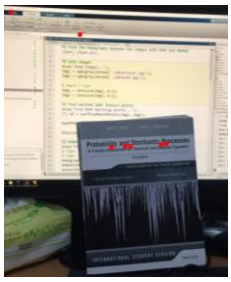
To project the pattern onto the target, we need to find the homography **H** between **X** (Fixed target coordinate, the cover) and **W** (Coordinate of target in photo):

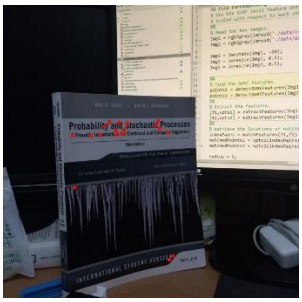
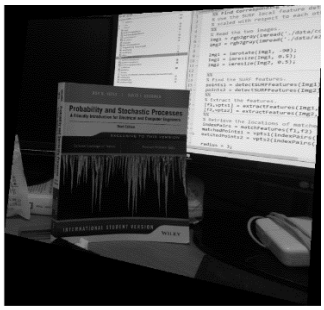


So the first step is to find **X** and **W**, this is divided into 2 steps:

1. Feature Extraction Find the feature points and their descriptor for **target** and **photo**	 
2. Feature Matching Find matching feature points, so **X** and **W** can be constructed.	 $\mathbf{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_N^T \end{bmatrix}$ $\boldsymbol{x}_n^T = [x_n \quad y_n \quad 1]$ $\boldsymbol{w}_n^T = [u_n \quad v_n \quad 1]$
I used SURF feature detecting/extraction/matching by Matlab	

Then, use RANSAC to find the best homography:

3. Come up with homographys Randomly pick 4 pairs matched feature points and find their homography.	$\mathbf{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \boldsymbol{x}_3^T \\ \boldsymbol{x}_4^T \end{bmatrix} \quad \mathbf{W} = \begin{bmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \boldsymbol{w}_3^T \\ \boldsymbol{w}_4^T \end{bmatrix}$ ➔ Find **H** through Gauss-Newton
4. Test the homographys Use the homography generated by 4 FPs onto all of the FPs, to see how many inliers can be found using this homography.	  **H1** **H2**

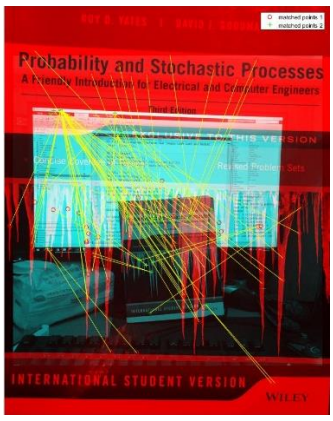
5. Find the best homography Use the largest set of inliers to find the best homography.	  **Inliers** **Projected by H**
Implemented by following the lecture slides	

Having found the homography **H** from **W** to **X**, the next step is to apply the inverse of **H** onto the pattern:

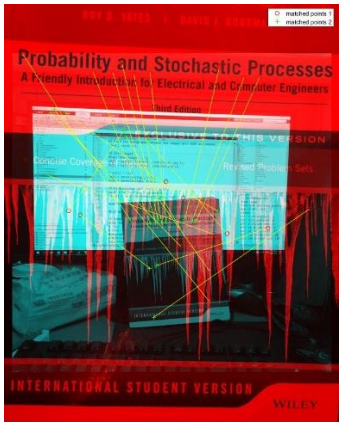
6. Apply H^{-1} onto the pattern	  **Original** **Projected by H^{-1}**
7. Append the projected pattern to original photo	
Projected image is produced by Matlab	

Technical Details

When using Matlab's SURF feature detection/matching:



Before

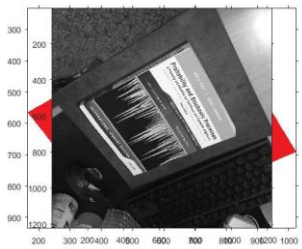


After

Sometimes one pixel position will be assigned to be multiple feature points (Probably because there's no dominant orientation), so a single pixel may have many other counterparts. So I decided to eliminate all duplicate FPs.

Failed to append projected pattern onto photo :

The plan is pretty straight forward and easy, so I was surprised when the projected pattern have the right orientation but not in scale and translation. Turns out, it's due to Matlab auto-adjusting the scale when showing images. So other procedure is required (view code) to put the projected pattern onto the same plain of the photo.



Results

b1.JPG

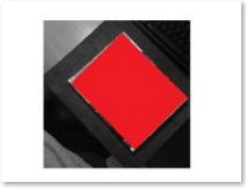
b2.JPG

b3.JPG

b4.JPG

c1.JPG

b1.jpg



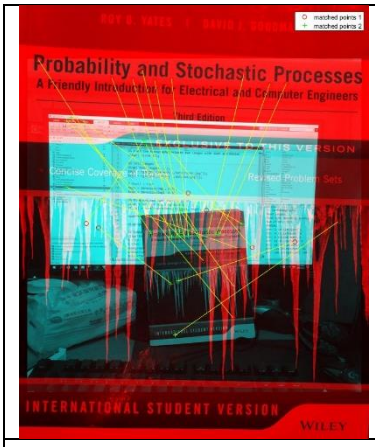
b2.jpg

b3.jpg

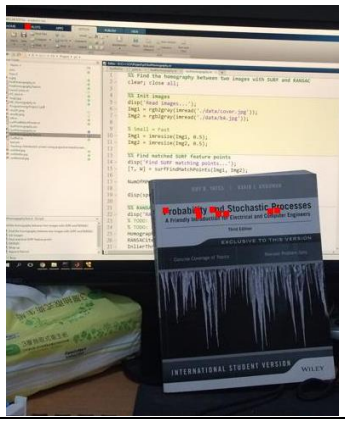
b4.jpg

c1.jpg

Most of the photo can be correctly projected, the ones that failed are mostly due to the lack of feature points and …



←The matched FP by SURF.
Notice that there is no correct FP at the bottom of the cover.



←In this project, inliers (red dot) should only appear on the book cover, but there is one located on the top

Using the inliers from the right, a wrong homography is produced, a single error inlier ruined the homography. However, even if I manually remove the outliers, because the points are forming a line, the correct homography still can't be found.

Using SURF + RANSAC to project planar objects works, but it requires stable FPs. Also, using Matlab makes the process extremely slow, I wonder how fast it can be when using OpenCV with C.