CSCI 5561: Assignment #2

Registration

1 Submission

- Assignment due: Mar 8 (11:55pm)
- Individual assignment
- 1 page summary write-up with resulting visualization (more than 1 page assignment will be automatically returned.).
- Submission through Canvas.
- List of submission codes:
  - `FindMatch.m`
  - `AlignImageUsingFeature.m`
  - `WarpImage.m`
  - `AlignImage.m`
  - `TrackMultiFrames.m`
- The function that does not comply with its specification will not be graded.
- You are allowed to use MATLAB built-in functions except for the ones in the Computer Vision Toolbox. Please consult with TA if you are not sure about the list of allowed functions.

CSCI 5561: Assignment #2

Registration

2 VLFeat Installation

(a) Image (b) SIFT

Figure 1: Given an image (a), you will extract SIFT features using VLFeat.

One of key skills to learn in computer vision is the ability to use other open source code, which allow you not to re-invent the wheel. We will use `VLFeat` by A. Vedaldi and B. Fulkerson (2008) for SIFT extraction given your images. Install `VLFeat` from following:
`http://www.vlfeat.org/install-matlab.html`
Run `vl_demo_sift_basic` to double check the installation is completed.

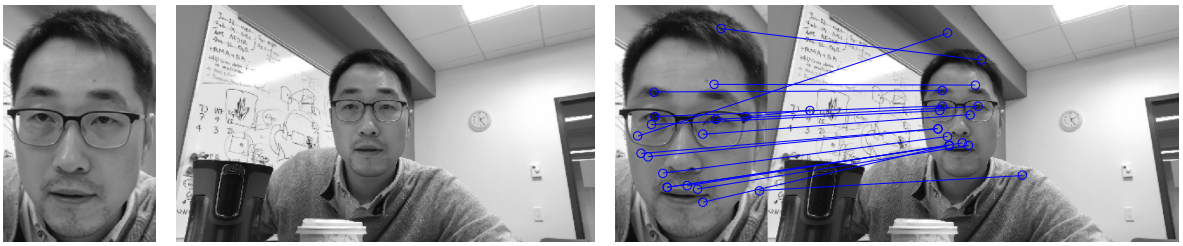
(Note) You will use this library only for SIFT feature extraction and its visualization. All following visualizations and algorithms must be done by your code. Using VLFeat, you can extract keypoints and associated descriptors as shown in Figure 1.

(SIFT visualization) Use `VLFeat` to visualize SIFT features with scale and orientation as shown in Figure 1. You may want to follow the following tutorial:
`http://www.vlfeat.org/overview/sift.html`

CSCI 5561: Assignment #2

Registration

3 SIFT Feature Matching



(a) Template (b) Target (c) SIFT matches with ratio test

Figure 2: You will match points between the template and target image using SIFT features.

The SIFT is composed of scale, orientation, and 128 dimensional local feature descriptor (integer), $\mathbf{f} \in \mathbb{Z}^{128}$. You will use the SIFT features to match between two images, I_1 and I_2. Use two sets of descriptors from the template and target, find the matches using nearest neighbor with the ratio test. You may use `knnsearch` built-in function in MATLAB.

`function [x1, x2] = FindMatch(I1, I2)`
Input: two input gray-scale images with `uint8` format.
Output: `x1` and `x2` are $n \times 2$ matrices that specify the correspondence.
Description: Each row of `x1` and `x2` contains the (x, y) coordinate of the point correspondence in I_1 ad I_2, respectively, i.e., `x1(i,:)` $\leftrightarrow$ `x2(i,:)`.

(Note) You can only use `VLFeat` for the SIFT descriptor extraction. Matching with the ratio test needs to be implemented by yourself.

CSCI 5561: Assignment #2

Registration

4 Feature-based Image Alignment

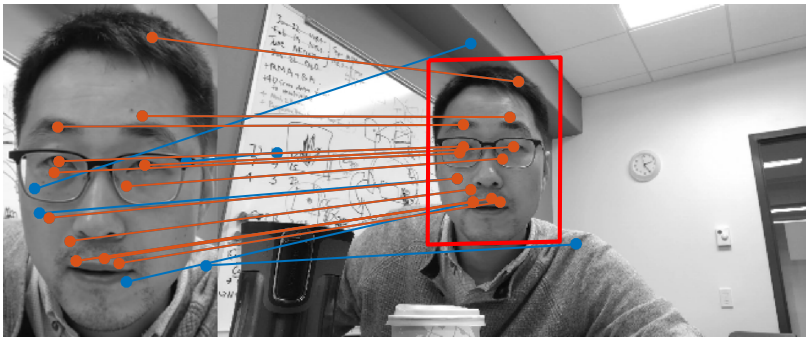


Figure 3: You will compute an affine transform using SIFT matches filtered by RANSAC. Blue: outliers; Orange: inliers; Red: the boundary of the transformed template.

(Note) From this point, you cannot use any function provided by `VLFeat`.

The noisy SIFT matches can be filtered by RANSAC with an affine transformation as shown in Figure 3.

`function [A] = AlignImageUsingFeature(x1, x2, ransac_thr, ransac_iter)`
Input: `x1` and `x2` are the correspondence sets ($n \times 2$ matrices). `ransac_thr` and `ransac_iter` are the error threshold and the number of iterations for RANSAC.
Output: 3×3 affine transformation.
Description: The affine transform will transform x_1 to x_2, i.e., $x_2 = Ax_1$. You may visualize the inliers and the boundary of the transformed template to validate your implementation.

CSCI 5561: Assignment #2

Registration

5 Image Warping

(a) Image (b) Warped image (c) Template (d) Error map

Figure 4: You will use the affine transform to warp the target image to the template using the inverse mapping. Using the warped image, the error map $|I_{\text{tpl}} - I_{\text{wrp}}|$ can be computed to validate the correctness of the transformation where I_{tpl} and I_{wrp} are the template and warped images.

Given an affine transform A, you will write a code to warp an image $I(x) \rightarrow I(Ax)$.

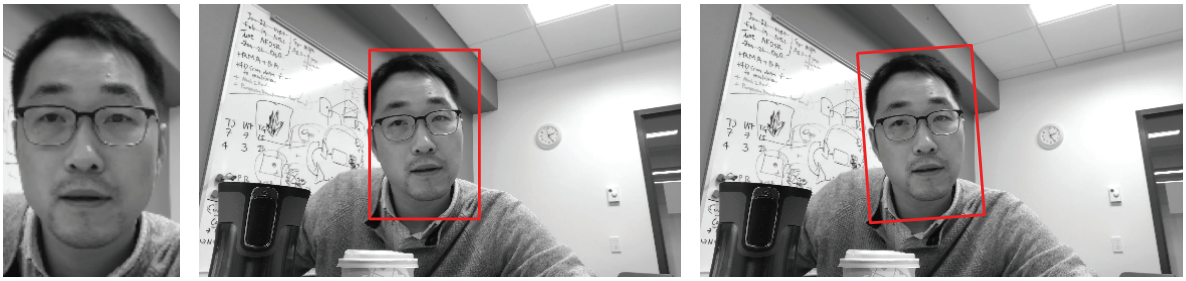
`function [I_warped] = WarpImage(I, A, output_size)`
Input: `I` is an image to warp, `A` is the affine transformation from the original coordinate to the warped coordinate, `output_size=[h,w]` is the size of the warped image where `w` and `h` are the width and height of the warped image.
Output: `I_warped` is the warped image with the size of `output_size`.
Description: The inverse mapping method needs to be applied to make sure the warped image does not produce empty pixel. You are allowed to use `interp2` build-in function in MATLAB for bilinear interpolation.

(Validation) Using the warped image, the error map $|I_{\text{tpl}} - I_{\text{wrp}}|$ can be computed to validate the correctness of the transformation where I_{tpl} and I_{wrp} are the template and warped images.

CSCI 5561: Assignment #2

Registration

6 Inverse Compositional Image Alignment



(a) Template (b) Initialization (c) Aligned image

Figure 5: You will use the initial estimate of the affine transform to align (i.e., track) next image. (a) Template image from the first frame image. (b) The second frame image with the initialization of the affine transform. (c) The second frame image with the optimized affine transform using the inverse compositional image alignment.

Given the initial estimate of the affine transform `A` from the feature based image alignment (Section 4) as shown in Figure 5(b), you will track the next frame image using the inverse compositional method (Figure 5(c)). You will parametrize the affine transform with 6 parameters $p = (p_1, p_2, p_3, p_4, p_5, p_6)$, i.e.,

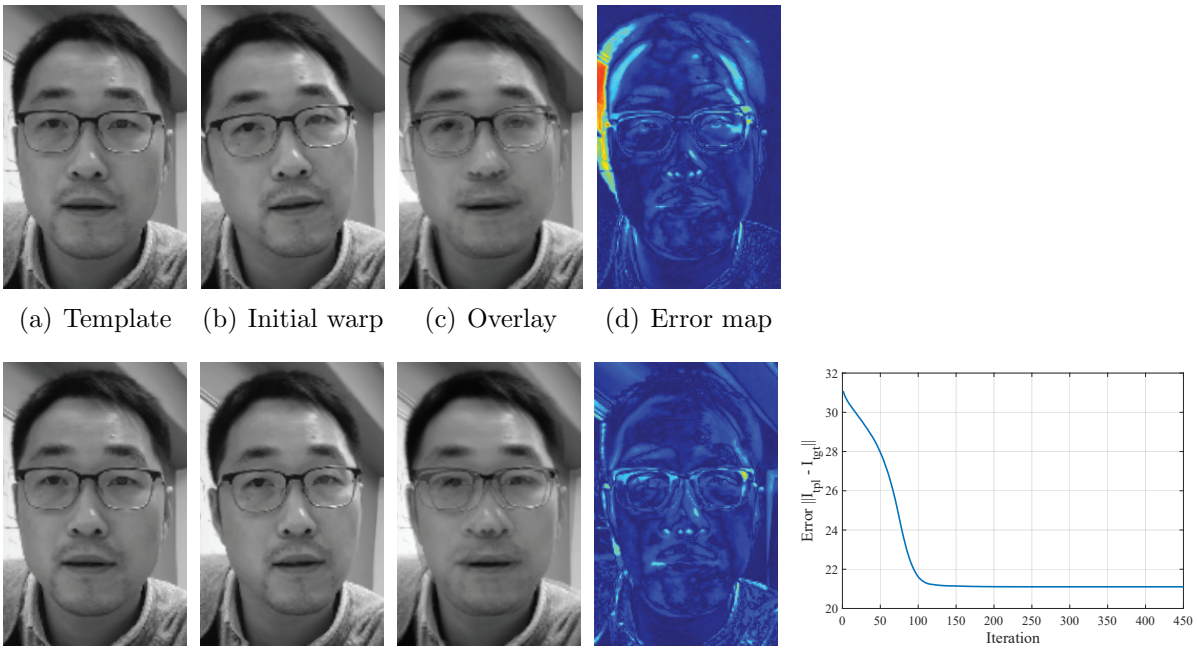
$$W(x;p) = \begin{bmatrix} p_1 + 1 & p_2 & p_3 \\ p_4 & p_5 + 1 & p_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = A(p)x \tag{1}$$

where $W(x;p)$ is the warping function from the template patch to the target image. $x = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$ is the coordinate of the point before warping, and $A(p)$ is the affine transform parametrized by p.

`function [A_refined] = AlignImage(template, target, A)`
Input: gray-scale template `template` and target image `target`; the initialization of 3×3 affine transform `A`, i.e., $x_{\text{tgt}} =$`A`x_{tpl} where x_{tgt} and x_{tpl} are points in the target and template images, respectively.
Output: `A_refined` is the refined affine transform based on inverse compositional image alignment
Description: You will refine the affine transform using inverse compositional image alignment, i.e., `A`→`A_refined`. The pseudo-code can be found in Algorithm 1.
Tip: You can validate your algorithm by visualizing their error map as shown in Figure 6(d) and 6(h). Also you can visualize the error plot over iterations, i.e., the error must decrease as shown in Figure 6(i).

CSCI 5561: Assignment #2

Registration



(a) Template (b) Initial warp (c) Overlay (d) Error map

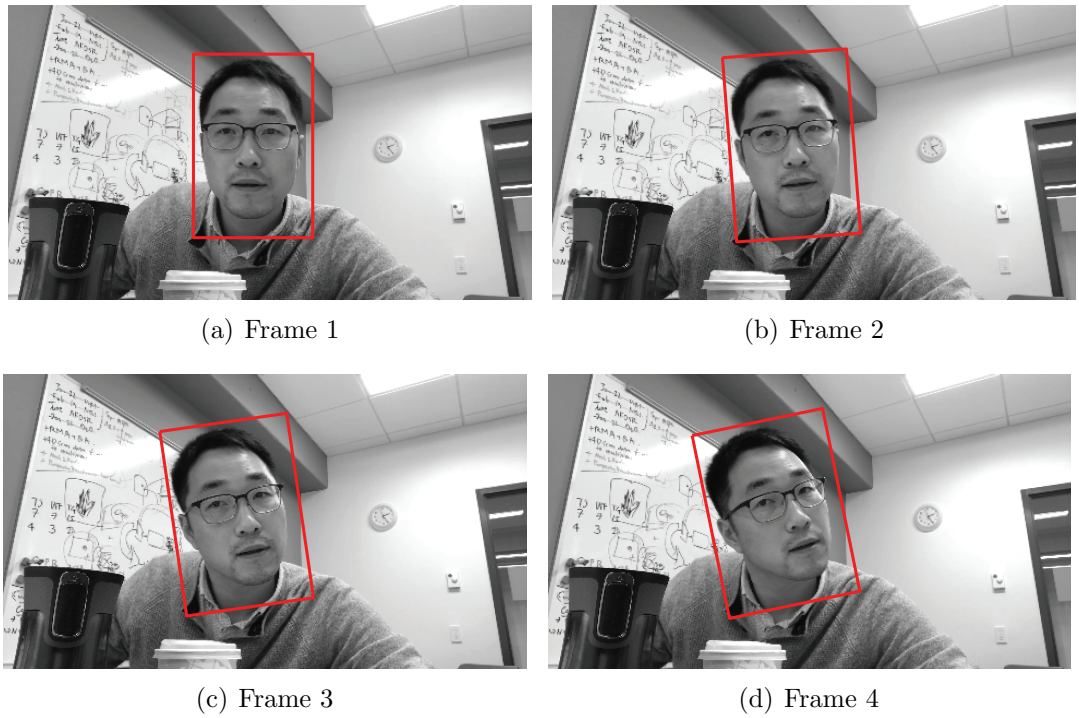
(e) Template (f) Opt. warp (g) Overlay (h) Error map (i) Error map

Figure 6: (a,e) Template images of the first frame. (b) Warped image based on the initialization of the affine parameters. (c) Template image is overlaid by the initialization. (d) Error map of the initialization. (f) Optimized warped image using the inverse compositional image alignment. (g) Template image is overlaid by the optimized warped image. (h) Error map of the optimization. (i) An error plot over iterations.

Algorithm 1 Inverse Compositional Image Alignment

1: Initialize $p = p_0$ from input `A`.
2: Compute the gradient of template image, ∇I_{tpl}
3: Compute the Jacobian $\frac{\partial W}{\partial p}$ at $(x; 0)$.
4: Compute the steepest decent images $\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}$
5: Compute the 6×6 Hessian $H = \sum_x \left[\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}\right]^\top \left[\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}\right]$
6: **while** $\|p\| > \epsilon$ **do**
7: Warp the target to the template domain $I_{\text{tgt}}(W(x; p))$.
8: Compute the error image $I_{\text{err}} = I_{\text{tgt}}(W(x; p)) - I_{\text{tpl}}$.
9: Compute $F = \sum_x \left[\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}\right]^\top I_{\text{err}}$.
10: Compute $\Delta p = H^{-1} F$.
11: Update $W(x; p) \leftarrow W(x; p) \circ W(x; \Delta p) = W(W(x; \Delta p); p)$.
12: **end while**
13: Return `A_refined` made of p.

7 Putting Things Together: Multiframe Tracking



(a) Frame 1 (b) Frame 2

(c) Frame 3 (d) Frame 4

Figure 7: You will use the inverse compositional image alignment to track 4 frames of images.

Given a template and a set of consecutive images, you will (1) initialize the affine transform using the feature based alignment and then (2) track over frames using the inverse compositional image alignment.

`function [A_cell] = TrackMultiFrames(template, image_cell)`
Input: `template` is gray-scale template. `image_cell` is a cell structure that contains a set of consecutive image frames, i.e., `image_cell{i}` is the i^{th} frame.
Output: `A_cell` is the set of affine transforms from the template to each frame of image, i.e., `A_cell{i}` is the affine transform from the template to the i^{th} image.
Description: You will apply the inverse compositional image alignment sequentially to track over frames as shown in Figure 7. Note that the template image needs to be updated at every frame, i.e., `template`←`WarpImage(I, inv(A), size(template))`.