CSCI 5561: Project #2

Registration

1 Submission

- Assignment due: Tuesday, October 21, 2025 (11:59pm)
- Individual assignment
- Please submit your assignment to the Gradescope **Autograder**.
- You will complete `p2.py` that contains the following functions:
  - `find_match`
  - `align_image_using_feature`
  - `warp_image`
  - `align_image`
  - `track_multi_frames`

  `p2.py` is attached with the Canvas assignment for you to download.
- Any function that does not comply with its specification will not be graded (no credit).
- The code must be run with the Python 3 interpreter.
- You are not allowed to use computer vision related package functions unless explicitly mentioned here. Please consult with TA if you are not sure about the list of allowed functions.

CSCI 5561: Project #2

Registration

2 SIFT Feature Extraction

(a) Image (b) SIFT

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

One of the key skills to learn in computer vision (and software development in general) is the ability to use other, open-source code, which allows you to not reinvent the wheel. We will use OpenCV library for SIFT features extraction given your images.

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 OpenCV, you can extract keypoints and associated descriptors as shown in Figure 1. SIFT is now included with the *core* OpenCV library, as the patent for the algorithm expired in 2020. But older version of OpenCV will not have it in the core library, so please make sure you have a recent version of OpenCV.

Use `OpenCV` to visualize SIFT features with scale and orientation as shown in Figure 1 (OpenCV may use different colors to visualize). You may want to follow the following tutorial:
`https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`

CSCI 5561: Project #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.

SIFT is composed of scale, orientation, and a 128-dimensional local feature descriptor, $\mathbf{f} \in \mathbb{Z}^{128}$. 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 `NearestNeighbors` function imported from `sklearn.neighbors` (You can install `sklearn` package easily by "pip3 install -U scikit-learn").

You can only use the SIFT module of `OpenCV` for the SIFT descriptor extraction. **You need to implement the matching with the ratio test yourself**.

```
def find_match(img1, img2):
  ...
  return x1, x2
```

Input: two input grayscale 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 and I_2, respectively, i.e., `x1(i,:)` $\leftrightarrow$ `x2(i,:)`.

CSCI 5561: Project #2

Registration

4 Feature-based Image Alignment

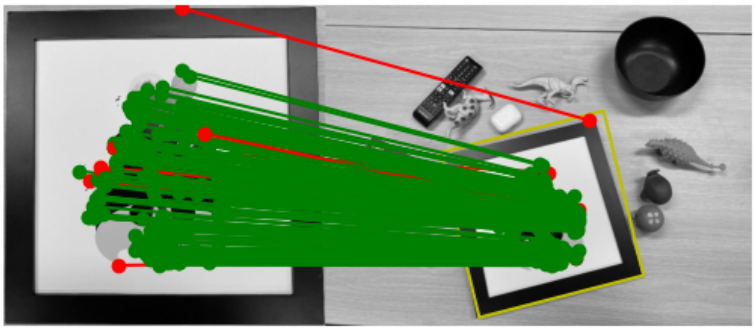


Figure 3: You will compute an affine transform using SIFT matches filtered by RANSAC. Red: outliers; Green: inliers; Yellow: the boundary of the transformed template.

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

From this point, you cannot use any function provided by `OpenCV`, except for purely visualization purposes.

```
def align_image_using_feature(x1, x2, ransac_thr, ransac_iter):
  ...
  return A
```

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: Project #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 code to warp an image $I(x) \rightarrow I(Ax)$.

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.

```
def warp_image(img, A, output_size):
  ...
  return img_warped
```

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: `img_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 pixels. You are allowed to use `interpn` function imported from `scipy.interpolate` for bilinear interpolation (`scipy` package can be easily installed through "pip3 install scipy" if you have not installed it yet).

CSCI 5561: Project #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. (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. You may use `filter2D` to compute the gradient of template image.

```
def align_image(template, target, A):
  ...
  return A_refined, errors
```

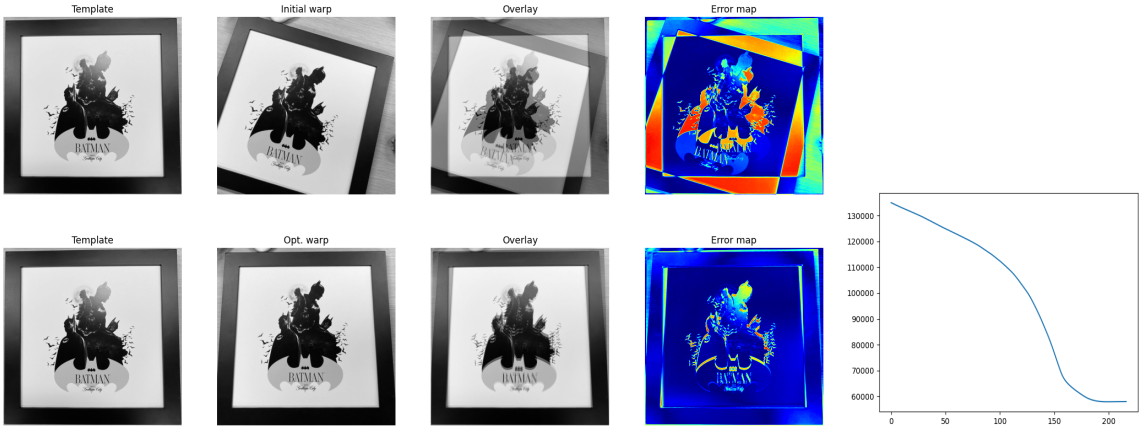
Input: grayscale template `template` and target image `target`; the initialization of 3×3 affine transform `A`, i.e., $x_{\text{tgt}} =$`A`x_{tpl} where x_{tgt} and x_{tpl} are points in the target and template images, respectively.
Output: `A_refined` is the refined affine transform based on inverse compositional image alignment; `errors` records the alignment error (L2 norm) at each iteration.
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(a). Also you can visualize the error plot over iterations, i.e., the error must decrease as shown in Figure 6(b).

CSCI 5561: Project #2

Registration

Algorithm 1 Inverse Compositional Image Alignment

Initialize $p = p_0$ from input `A`.
Compute the gradient of template image, ∇I_{tpl}
Compute the Jacobian $\frac{\partial W}{\partial p}$ at $(x; 0)$.
Compute the steepest descent images $\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}$
Compute the 6×6 Hessian $H = \sum_x \left[\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}\right]^{\mathsf{T}} \left[\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}\right]$
while True **do**
    Warp the target to the template domain $I_{\text{tgt}}(W(x; p))$.
    Compute the error image $I_{\text{err}} = I_{\text{tgt}}(W(x; p)) - I_{\text{tpl}}$.
    Compute $F = \sum_x \left[\nabla I_{\text{tpl}} \frac{\partial W}{\partial p}\right]^{\mathsf{T}} I_{\text{err}}$.
    Compute $\Delta p = H^{-1} F$.
    Update $W(x; p) \leftarrow W(x; p) \circ W^{-1}(x; \Delta p) = W(W^{-1}(x; \Delta p); p)$.
    if $\|\Delta p\| < \epsilon$ **then**
        break
    end if
end while
Return `A_refined` made of p.



(a) Image Alignment Steps and Error Map (b) Error map

Figure 6: Left to right, top to bottom: Template images of the first frame, warped image based on the initialization of the affine parameters, template image is overlaid by the initialization, error map of the initialization, repeated template images of the first frame, optimized warped image using the inverse compositional image alignment, template image is overlaid by the optimized warped image, error map of the optimization. (b) An error plot over iterations.

CSCI 5561: Project #2

Registration

7 Putting Things Together: Multiframe Tracking

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.

```
def track_multi_frames(template, img_list):
  ...
  return A_list, errors_list
```

Input: `template` is grayscale template. `image_list` is a list of consecutive image frames, i.e., `img_list[i]` is the i^{th} frame.
Output: `A_list` is the set of affine transforms from the template to each frame of image, i.e., `A_list[i]` is the affine transform from the template to the i^{th} image; `errors_list` stores the alignment errors per frame across iterations.
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`←`warp_image(img, A, template.shape)`.