TTIC 31040, Spring 2021
Project: instance-level object detection

May 21, 2021

1 Project goals

The goal of this project is to develop a system that can detect a given object in an image. "Object" here means a particular instance, rather than a category. I.e., a specific mug, or a book, and not any mug or any book.

This system will be organized as follows. A *reference* image of the object to search for is provided. You can assume that the object is not occluded in the reference image, and that no other objects are visible in it. The implication of this is that all (or at least a large majority of) interest point detected in the reference image can be assume to belong to the object.

The reference image containing the object defines a *bounding box* for it. Intuitively, you describe a detected instance of the object in the input image by specifying how you'd overlay the reference image over the input image so that the detected instance would be aligned with the reference instance. See Figure 1.

Broadly speaking, you will be implementing the system proposed in David Lowe's paper in 1999, that introduced the SIFT descriptors for the first time.

2 Steps to implement

1. Figure out how to extract interest point operators and compute the corresponding SIFT descriptors (you should already be reasonably comfortable with this after some of the problem sets)

2. Implement robust matching of SIFT descriptors (again, probably already done in problem sets).

3. Implement construction of a reference database $\mathcal{D}_I$ from the reference image I. This database should enable the following lookup operation: You can query it with a SIFT descriptor $\mathbf{f}$, along with the location (x, y), orientation θ and scale s of the keypoint in the input image for which this descriptor was computed. The output for this query

is either an indication that no reliable matches were found, or the parameters of the (rectangular) bounding box in the input image that is implied by the found reliable match. Refer to Fig. 1 for clarity. Note that you will need access to the location of both keypoints in a match (one in the reference image and one in the input image) for the following steps.

4. Implement Hough transform for (rectangular) bounding box detection, that uses the reference database lookup as a source of (potential) vote, and produces a set of hypothesized detections from peaks in the accumulator array. You should be able to "backtrack" from any such hypothesis to the set of keypoints in the input image (and the matching keypoints in the reference image) that voted for it; this will be the basis for fitting a more refined deformation model (affine or projective).

5. Implement robust fitting of a 2D affine transformation (and, for two-person projects, a projective transformation) to a set of correspondences between the reference image and the input image. The obvious choice here is to use RANSAC but you are welcome to consider other options. Make sure you include the final step of re-fitting of the model to inliers. You will be fitting the 2D transformation to a set of correspondences that "contributed" to the hypothesis that lead to the Hough detection. Note that in this step you can ignore all the information about the point correspondences besides their coordinates in the two images.

6. Finally, put it all together and implement a detector that takes as arguments the reference database, an input image, and any detection hyperparameters, e.g., indicating sensitivity (you are free to define these as you see fit). The detector should do two things: display the (affine transformed) bounding box[es] drawn on the input image (make sure to use bright colors and thick lines), and yield the detections as a tuple of parameterized (transformed) bounding boxes as its return value.

We leave all the details of the implementation to you.

3 Project submission

Please describe the pipeline you end up implementing in the PDF writeup. We recommend LaTeX, but other typesetting environments like Word or Google Docs are fine, as long as the submitted document is in PDF. Please submit this PDF (try to keep it shorter than 4 pages) along with the full notebook, and a `.zip` file containing any additional material we'd need to replicate your experiments. This may include your own images; Python code written in separate files imported into the notebooks (if you can avoid that, please do, as it makes our work easier); etc.

Your writeup should include:

- Precise description of the implementation, written in a way that would allow a fellow student in the course to replicate your design.

- Discussion of any choices you had to make, and an explanation of how and why you made a particular choice there. (E.g., things that can be parameterized in different way; settings for hyperparameters; etc.)

- Qualitative evaluation of the performance of your method, and any thoughts on potential improvements one could make with more time/effort. This includes improvements to the method per se, as well as improvements to implementation (such as speeding it up).

- Discussion of the fundamental limitations of the method as designed and implemented, including any specific failure modes (types of input/reference combinations in which the method is likely to fail).

Your results should demonstrate the detection performance on at least two objects. One is the stop sign included provided by us. The folder `stopsign` includes the reference image and five input image in which you should detect the reference stop sign. You can treat the entire extent of the `stop-reference.jpg` as the reference bounding object (so the image boundaries form the reference bounding box).

The second object is up to you. Take a picture of an object (a book, or if you want a challenge, something more 3-dimensional) on an uncluttered background, and crop it to make the reference image. Then take at least five substantially different other pictures of the reference object; try to vary illumination, pose of the object, distance to the camera, and include some occlusions. Make sure to show the reference and input images for your object(s) in the notebook.

For two-person projects: Your detector should be able to detect multiple instances of the reference object in an input image. Also, you should make an effort to tune the hyperparameters of the system (this includes the settings for the Hough transform, any hyperparameters you have for the robust affine fitting, and anything else you think might affect the results). Describe the procedure and settings for this step in the writeup, and include the code in the notebook. Finally, in addition to affine transformation, you should experiment with fitting a projective tranformation (i.e., accounting for perspective deformations), and discuss in the writeup the necessary modifications and the observations/conclusions from comparing the homography-based detector to the affine one.

For single-person projects: You are ecnouraged to include the results listed for the two-person projects, but can skip them. I.e., it's OK to only handle a single object detection per input image, and it's OK to only evaluate your method with a single set of hyperparameter values (but then you should explain why you decided on those values).

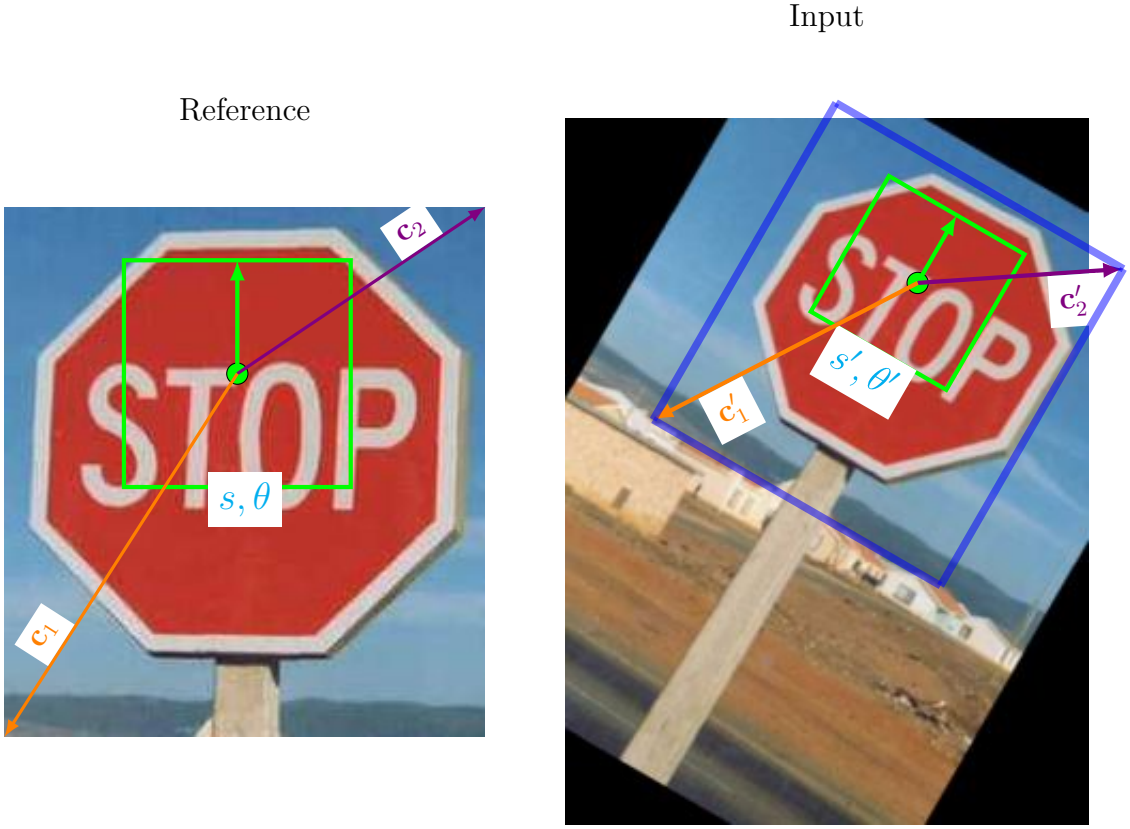

Figure 1: The basic idea of instance-level detection by matching features. Here we show a single feature match, yielding a detection shown by the blue bounding box in the input image. You will need to assemble evidence from multiple matches, as described. The bounding box here is parameterized by the locations of two opposite corners (four numbers total); you may want to use other parameterizations. The location of the matching point in the reference image yields vote for the relative location (offset) of the two corners, $\mathbf{c}_1$ and $\mathbf{c}_2$, respectively. By adjusting for the relative scale and orientation of the match, i.e., s/s' and $\theta - \theta'$, we get the estimated offsets $\mathbf{c}'_1, \mathbf{c}'_2$ that specify the corresponding bounding box in the input image. Notes: (1) This shows a single "vote" based on matching descriptors; your detector will accumulate votes from many descriptors. (2) Your detector will go beyond similarity transform (rotation/scaling/translation, leaving the box rectangular) and fit an affine/projective transformation from a set of correspondences. (3) The detected bounding box may "spill" over the boundaries of the input image; that's OK, and you should display it as such if it does.

Figure 2: A possible detection shown as an affine (left; parallelogram) and projective (right; general quadriliteral) bounding box. These are just to illustrate the kind of displayed output you should aim at; they are not results of an actual detector.