Logo Detection using Template Matching and Feature Matching

Thursday, August 14, 2025 12:02 PM

. Template Matching

What

- A **pixel-by-pixel comparison** method where you slide a smaller image (template) over a larger image and find the best match.
- Usually implemented with OpenCV's cv2.matchTemplate().
- Works by comparing pixel intensity patterns and returning a similarity score (e.g., correlation coefficient).

It returns a grayscale image, where each pixel denotes how much does the neighbourhood of that pixel match with template.

Template matching is one of **pattern recognition technique** where we provide a single part or object from image as
to mark multiple similar patterns from

Pixel level template matching:
where we don't consider any kind of rotation, translational invariance, shifts in degrees.
the template pattern is exactly similar to object patterns in test image.

images. https://medium.com/@BH_Chinmay/object-detection-lite-template-matching-c9af77517f6c

Template Matching

First Principles

1. **Pixel-wise similarity**
- The algorithm slides the template over the image, compares pixel intensities, and measures similarity.
- Works best when the **pixel values and their spatial arrangement remain unchanged** between template and target.
2. **Search space is limited**
- It has to check every possible position (and sometimes scale) → computationally expensive.
3. **Assumes rigid object**
- No deformation — only translation is reliably supported (unless you explicitly handle scale/rotation with multi-scale searches). [rotational]

Thumb Rules

- **Keep object appearance fixed** — no major rotation, scaling, or warping.
- **Normalize lighting** — changes in brightness can break matching unless you use a normalized method (TM_CCOEFF_NORMED).
- **Smaller template = faster search** — but too small loses context.
- **If scale changes are possible → use multi-scale search**.
- **If object can deform → use feature matching or deep learning instead**.
- Works best on **clean, uncluttered backgrounds**.

Pixel values

- This is **what** is at each pixel location.
- Example:
- Intensity: 0 (black), 255 (white), or any number i
- Color: (R, G, B) values for colored images.
- If you think of an image as a table of numbers, t

2. Spatial arrangement

- This is **where** those values are in relation to eac
- It's the geometry — the position of bright, dark,
- Even if pixel values are the same, **shuffling the arrangement**.

Example
Template (3×3 grayscale block):

```
100 255 100      x
255 255 255    x x x

100 255 100      x
```

- **Pixel values**: {100, 255}
- **Spatial arrangement**: bright pixels (255) form a

If we shuffle values:

```
255 100 255    x   x
100 255 100      x
255 100 255    x   x
```

- **Pixel values** are still {100, 255} — same numbe
- **Spatial arrangement** is now an X shape instea
- For template matching, this means **no match**, b

✅ **In short:**

- Pixel values = the "colors/brightness" of each sp
- Spatial arrangement = the "map" of where those

Feature Matching

First Principles

4. **Keypoint uniqueness**
- Identify parts of the image that are *repeatable and distinctive* (corners, blobs, edges).
- Each keypoint gets a descriptor — a numerical fingerprint of its local pattern.
5. **Descriptor comparison**
- Matching = finding descriptors in two images that are closest in descriptor space.
6. **Geometric consistency**
- True matches should follow a consistent geometric transform (translation, rotation, scaling, homography).
7. **Invariant features**
- Good detectors produce features that are stable under rotation, scaling, moderate lighting changes.

Thumb Rules

- **Choose the right detector**:
- ORB for speed (real-time),
- SIFT/SURF for robustness.
- **Filter matches** — raw matches always include false positives; use ratio test, RANSAC.
- **Grayscale images** are fine — most feature detectors ignore color.
- Works well even with **cluttered backgrounds** — as long as object has unique patterns.

- Avoid very low-texture objects (e.g., plain T-shirt) — no distinctive keypoints to match.
- If object can deform heavily → consider deformable models or dense optical flow instead.

- Template matching needs both to match exactly

Quick “When to Use” Compass

Scenario	Go for Template Matching if…	Go for Feature Matching if…
Object shape	Rigid, unchanged	May rotate, scale, partially hide
Background	Simple	Complex
Speed	Need very fast	Can afford more processing
Lighting change	Minimal	Moderate possible
Deformation	None	Mild, planar
Input	Exact template available	Just example images available

What ORB does

Detects keypoints → Finds important points in the image (like corners, edges, or texture-rich spots) that are good for matching.

Computes descriptors → For each keypoint, creates a small numeric “fingerprint” that describes how that area looks.

This is what lets it compare between two images.

What BFMatcher does

Takes descriptors from two images (desc1, desc2).

For each descriptor in image 1, it compares it to all descriptors in image 2 to find the best match.

“Best” is determined by a distance metric (how similar two descriptor vectors are).

For ORB (which gives binary descriptors), the best metric is Hamming distance (number of differing bits).

Sorts matches so closest matches (smallest distance) come first.