Product Recognition on Store Shelves project report

Step A-B

Marco Scaramuzzi

2025-09-20

0.1 Introduction

This project implements a single-instance product detector that localizes cereal boxes in shelf images using SIFT features computed independently on the RGB channels, FLANN matching with Lowe's ratio test, and homography estimation with RANSAC. The notebook `step_A.ipynb` contains the runnable pipeline; this report highlights the workflow, key data structures, concise results (extracted from the notebook output), and a short conclusion.

0.2 Project overview

The project consists of a pipeline for detecting products on store shelves using computer vision techniques. The main steps include loading images, extracting features, matching them, and estimating the position of products in the scene.

The processing pipeline is compactly represented below.

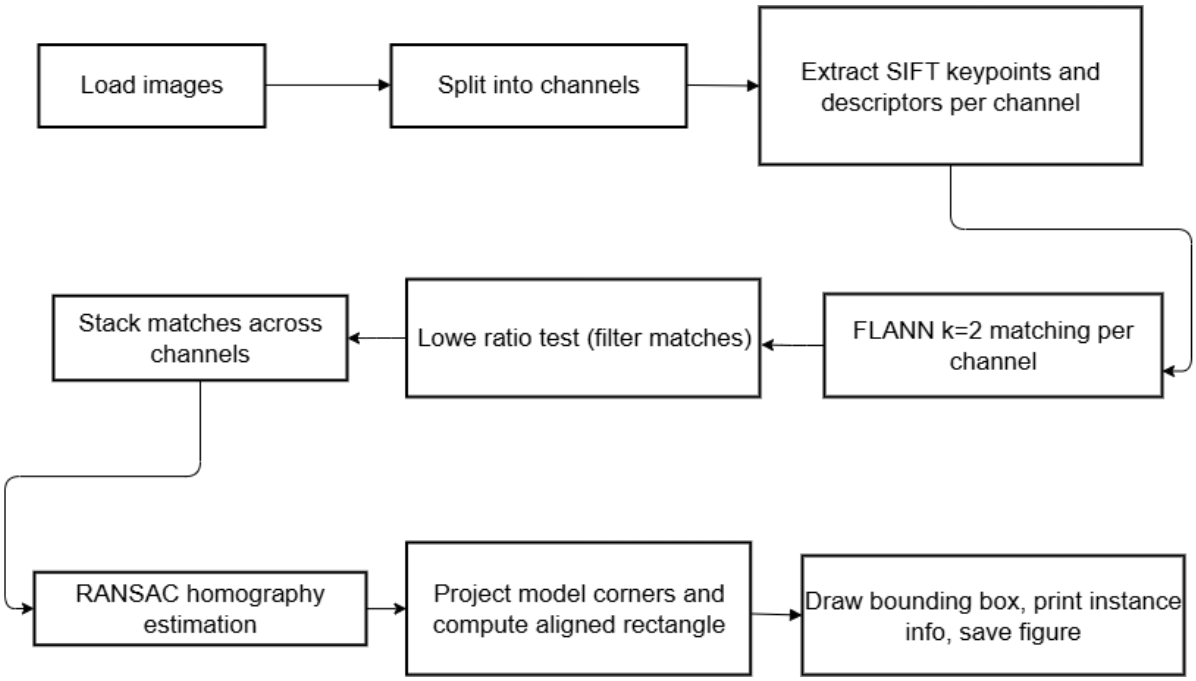

Figure 1: Workflow representation

0.3 Approach and key data structures

This section gives a deeper explanation of the pipeline (both theoretical motivations and implementation details) and the main in-memory data structures used by the notebook.

High-level pipeline

- Per-channel feature extraction: each model and scene image is split into R/G/B channels (function `create_channel_dict`) and SIFT keypoints/descriptors are computed independently on each channel using `cv2.SIFT_create()` inside `extract_features_dict`. The result is two nested dictionaries (`keypoints_dict` and `descriptors_dict`) keyed by image id and channel.
- Channel-wise matching: for each model-vs-scene pair, descriptors from channel `c` of the model are matched against descriptors from channel `c` of the scene using a FLANN approximate k-NN search (k=2). The reason for k=2 is to enable Lowe's ratio test which requires the nearest and second-nearest neighbors.
- Ratio-based filtering: the `filter_matches` function applies Lowe's ratio test (accept match if dist1 < 0.7 * dist2). This removes ambiguous matches where the nearest neighbor is not sufficiently better than the second nearest.
- Aggregation and geometric verification: the filtered 'good' matches from the three channels are stacked into arrays of source and destination points. These stacked correspondences feed `cv2.findHomography(..., cv2.RANSAC, ransacReprojThreshold=3)`. RANSAC rejects outlier matches and returns a robust 3×3 homography matrix that maps model coordinates to the scene.
- Projection and bounding rectangle: model corners are projected into the scene with `cv2.perspectiveTransform`. To produce a stable, axis-aligned bounding box the notebook calls `compute_aligned_rectangle` (in `utils/bounding_box_utils.py`), which centers and constrains the rectangle to image bounds and computes width/height/center.

Why SIFT across multiple channels

- Color-preserving detection: converting to grayscale discards chromatic edges and contrast that can be crucial for brand logos or colored patterns. Computing SIFT independently on R/G/B preserves such color-specific details.
- Complementary matches: features that are weak in one channel may be strong in another; stacking channel-wise good matches increases the chance of obtaining enough inliers for RANSAC while still allowing geometric pruning of false matches.
- Practical trade-offs: processing three channels multiplies descriptor storage and matching work by $\approx \times 3$. The notebook mitigates false positives by applying the Lowe ratio test per-channel and then applying RANSAC on stacked matches.

Key algorithmic components (practical notes referencing code)

- SIFT: instantiated via `sift = cv2.SIFT_create()` and applied per-channel in `extract_features_dict`.
- FLANN: the matcher is built in `initialize_flann()` with KD-tree (`trees=5`) and search `checks=50` (these parameters are tuned for a balance between speed and accuracy). FLANN's `knnMatch(..., k=2)` returns pairs used by Lowe's ratio.
- Lowe ratio test: implemented in `filter_matches` with threshold `0.7` (a standard value; higher values are stricter). The function returns only the best matches per channel.
- RANSAC/Homography: `compute_homography` stacks matched keypoint coordinates and calls `cv2.findHomography(..., cv2.RANSAC, ransacReprojThreshold=3)` to estimate a robust transformation. The reprojection threshold (3 px) controls inlier acceptance.

Representative in-memory structures

```
keypoints_dict = { image_id: {'R': [cv2.KeyPoint], 'G': [cv2.KeyPoint],
    'B': [cv2.KeyPoint]} }

descriptors_dict = { image_id: {'R': np.ndarray((N_R,128), np.float32),
    'G': ..., 'B': ...} }

matches_per_channel = { 'R': [cv2.DMatch], 'G': [...], 'B': [...] }

# homography: 3x3 numpy array
# projected corners: 4x1x2 float32 array
```

Parameters extracted from the code (explicit values)

- Model resize dimensions: `target_model_dims=(180, 240)` in `utils/image_utils.load_images` when `is_resized=True`.
- FLANN: KD-tree `trees=5`, search `checks=50` (`initialize_flann`).
- k in knnMatch: `k=2` (`compute_matches_dict_single`).
- Lowe ratio threshold: `0.7` (`filter_matches`).
- Minimum per-channel matches (threshold used in `main`): `min_count = 75` (the code compares total matches to `min_count * n_channels`).
- RANSAC reprojection threshold: 3 pixels (`compute_homography`).

> model images were resized to a common shape (`180x240`) before feature extraction to reduce inter-model scale variability. Although SIFT is scale-invariant theoretically, in practical template matching scenarios large differences in template sizes lead to inconsistent keypoint counts, differing keypoint scale distributions, and mismatched descriptor sampling densities. Normalizing templates to a common scale stabilizes the number and distribution of keypoints per model, simplifies the choice of numeric thresholds (e.g. `min_count`), and reduces false negatives caused by extreme template-to-scene scale mismatches. The trade-off is that very large scale variations in the scene may require multi-scale search or pyramid matching, but for a controlled, single-instance template setup this normalization improves consistency and reproducibility.

Suggested figures, tables and additional content to add for rigor

- Per-channel keypoint panels: for a representative model and scene, show three small images with SIFT keypoints plotted on `R`, `G`, and `B` channels respectively.
- Match-count table: CSV-style table with rows for (scene, model) and columns `matches_R`, `matches_G`, `matches_B`, `total_matches`. This empirically supports the `min_count` choice.
- Inlier evolution figure: for 2–3 pairs show (left) raw k-NN matches, (center) matches after Lowe filter, (right) RANSAC inliers overlayed on the scene. This visually demonstrates the filtering pipeline.
- Parameter summary: a small code block or table listing the numeric parameters (resize dims, FLANN trees/checks, Lowe ratio, min_count, RANSAC thresh).
- Optional metrics: if you can prepare a small ground-truth table (manually or from annotations), compute detection precision/recall or a per-scene Table of detected vs expected.

These additions make the report reproducible and easier to evaluate by a reader.

0.4 Results (parsed from notebook output)

Below are the detections produced by `main(min_count=75, ...)` for the five scenes. For each scene I include only products that had 1 instance (the single-instance constraint). The instance lines are verbatim from the notebook prints. For clarity the saved visualization is now shown before the textual detection output for each scene.

- Scene 1 — visualization

Figure 2: Scene 1 bounding boxes

  - Product 0 — 1 instance found:
    * Instance 1 {position: (162, 216), width: 309px, height: 432px}
  - Product 11 — 1 instance found:
    * Instance 1 {position: (444, 180), width: 299px, height: 359px}

- Scene 2 — visualization

Figure 3: Scene 2 bounding boxes

  - Product 24 — 1 instance found:
    * Instance 1 {position: (167, 232), width: 334px, height: 464px}
  - Product 25 — 1 instance found:
    * Instance 1 {position: (878, 232), width: 311px, height: 440px}
  - Product 26 — 1 instance found:
    * Instance 1 {position: (538, 230), width: 332px, height: 461px}

- Scene 3 — visualization

Figure 4: Scene 3 bounding boxes

  - Product `0` — 1 instance found:
    * Instance 1 {position: (172, 234), width: 323px, height: 436px}
  - Product `1` — 1 instance found:
    * Instance 1 {position: (816, 197), width: 303px, height: 394px}
  - Product `11` — 1 instance found:
    * Instance 1 {position: (476, 192), width: 303px, height: 385px}

- Scene 4 — visualization

Figure 5: Scene 4 bounding boxes

  - Product `0` — 1 instance found:
    * Instance 1 {position: (160, 738), width: 320px, height: 435px}
  - Product `11` — 1 instance found:
    * Instance 1 {position: (463, 690), width: 304px, height: 396px}
  - Product `25` — 1 instance found:
    * Instance 1 {position: (554, 216), width: 318px, height: 433px}
  - Product `26` — 1 instance found:
    * Instance 1 {position: (206, 221), width: 341px, height: 442px}

- Scene 5 — visualization

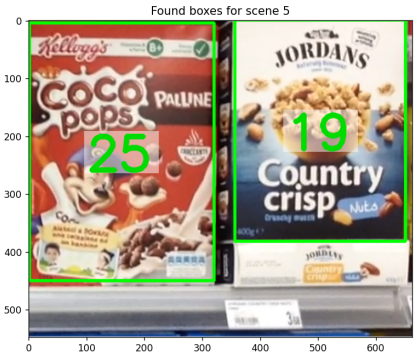

Figure 6: Scene 5 bounding boxes

  - Product `19` — 1 instance found:
    * Instance 1 {position: (504, 192), width: 295px, height: 383px}
  - Product `25` — 1 instance found:
    * Instance 1 {position: (161, 228), width: 320px, height: 445px}

0.5 Conclusions

Summary: the SIFT-per-channel + FLANN + RANSAC pipeline reliably localizes single instances of the provided product models in the five test scenes. The saved `figures/scene_*.png` images contain the annotated outputs used to verify detections.

Next steps: extend to multi-instance detection, add quantitative metrics (precision/recall), or move to a learned detector for greater robustness under heavy occlusion.