Product recognition on Store shelves

Student: Hu Stefano Jin Cheng

Introduction

Overall task description

Develop a computer vision system that can identify cereal boxes of different brands from a single store shelf image, using a reference image for each product. For each type of product displayed on the shelf, the system should report:

1. Number of instances
2. Dimensions of each instance (width and height of the enclosing bounding box in pixels)
3. Position of each instance in the image reference system (center of the enclosing bounding box in pixels)

For example, as output for the above image, the system should print:

Product 0 - 2 instances found:

Instance 1 {position: (256,328), width: 57px, height: 80px}
Instance 2 {position: (311,328), width: 57px, height: 80px}

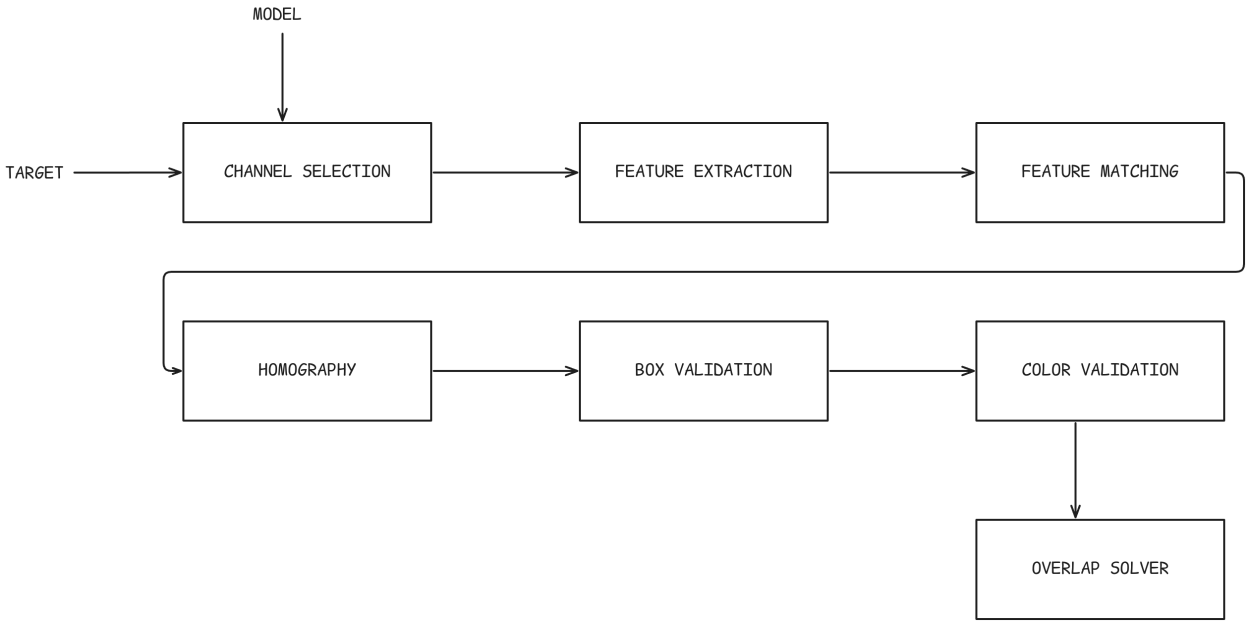
Product 1 - 1 instance found:...

Step A - Multi Product Detection

Task description

Develop an object detection system to identify single instance of products given: one reference image for each item and a scene image. The system should be able to correctly identify all the product in the shelves image.

One way to solve this task could be the use of local invariant feature as explained in lab session 5



Solution

Model

Image of single product

Target

Image of the scene that contains only one instance of ant product

Channel selection

Since some model images mainly differ from each other by color, it is challenging to distinguish them using only grayscale feature detection and matching.

The solution proposed is:

1. split the model image into rgb channels
2. determine the major intensity color channel
3. using that channel for detection and matching

Feacture extraction

SIFT (Scale-Invariant Feature Transform) extracts keypoints from both model and target images to detect and describe distinctive features

```
sift = cv2.SIFT_create(sigma=1.6)
kp_query = sift.detect(query)
kp_train = sift.detect(train)
kp_query, des_query = sift.compute(query, kp_query)
kp_train, des_train = sift.compute(train, kp_train)
```

Feature matching

Keypoints are matched using a KNN Flann Matcher that relies on KD-trees. If insufficient matches are found, the system determines that the current Model is not present in the Target (scene) image.

```
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees =
search_params = dict(checks = 50)

flann = cv2.FlannBasedMatcher(index_params, search_params)
for m,n in flann.knnMatch(des_query,des_train,k=2):
```

```
        if m.distance < knn_distance*n.distance:
            good.append(m)
```

Homography

When sufficient matches are found, they are used to estimate a homography using RANSAC. This allows the system to identify an initial bounding box in the target image.

```
Homography, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANS
```

Box Validation

Some calculated bounding boxes may deviate significantly from a rectangular shape or have unrealistic dimensions. These invalid bounding boxes must be discarded.

Color validation

After cropping the target image using the bounding box, we compare its color with the model image. Any candidates scoring below a set threshold are discarded.

```
    color_d = ((r_diff)**2 +(g_diff)**2+(b_diff)**2)**0.5
    if color_d < color_threshold:
        return True
    return False
```

Overlap solver

If there are some boxes that have its position overlapping into another bounding box in the same scene, then they must be detected. The resolution is to eliminate one of them by choosing to keep the one with bigger matching number.

```
if (instance1.position[0]>=instance2.minx and instance1.posit
ion[0]<=instance2.maxx and instance1.position[1]>=instance2.m
iny and instance1.position[1]<=instance2.maxy):
      if instance1.matches_number>instance2.matches_number:
```

```
            product_info[j].instances.remove(instance2)
        else:
            product_info[i].instances.remove(instance1)
```

Step B-Multiple Instance Detection

Task description

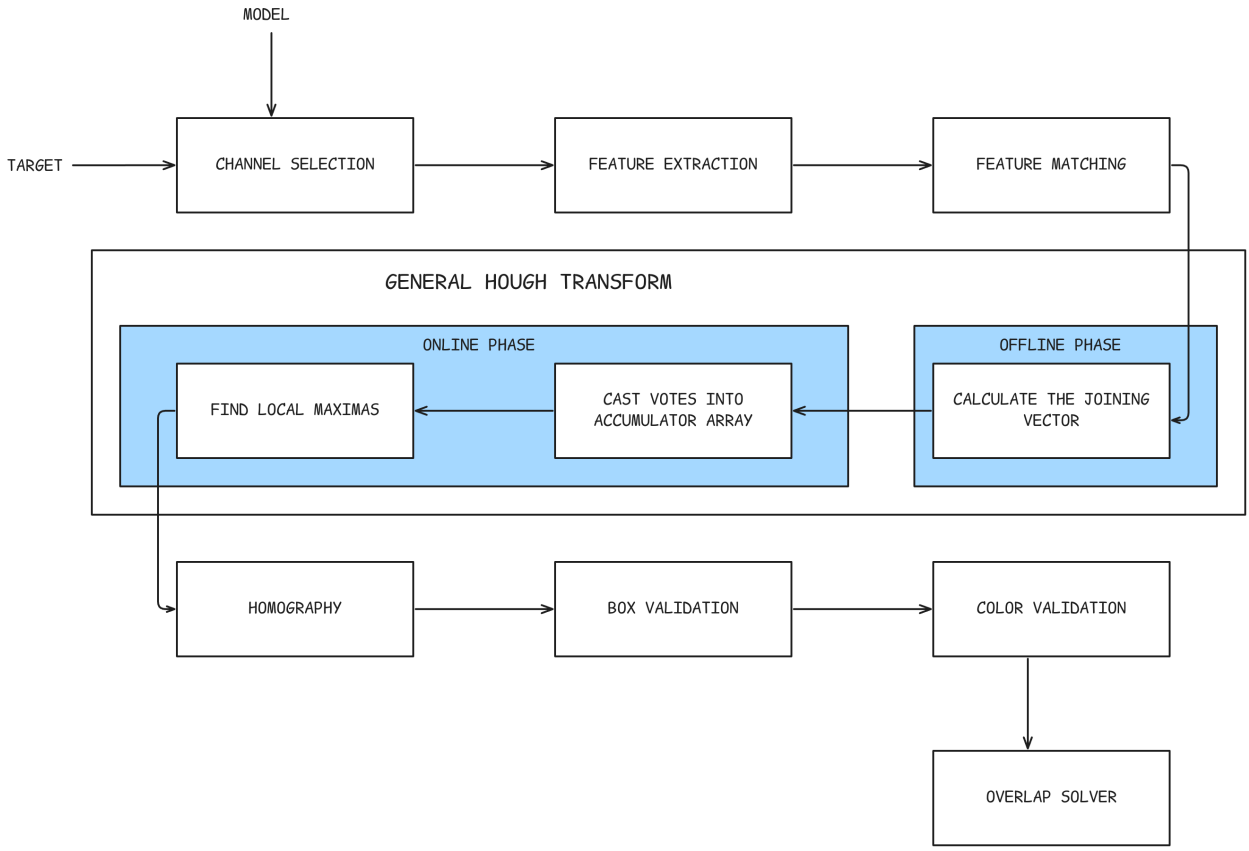
In addition to what was achieved in step A, the system should now be able to detect multiple instances of the same product.

Students may use local invariant features together with the GHT (Generalized Hough Transform). Instead of relying on the usual R-Table, the object model acquired during training should consist of vectors joining all features extracted in the model image to their barycenter.

At runtime, all image features matched with the model would cast votes for the barycenter's position by scaling the associated joining vectors appropriately (based on the size ratio between matching features).

Solution

In addition to the step A in order to detect multiple instances of the same model, a **General Hough transform** step is introduced to handle this task.



Finding an object in an image requires locating its model's position. The **generalized Hough transform** converts this task into finding the transformation parameters that map the model onto the image. Once these parameters are determined, we can locate the model's exact position in the image.

The implementation is structured as following:

- **Offline phase**:
    1. Computation the model’s joining vectors by using the barycenter as the reference point
- **Online phase**:
    1. The joining vector are rescaled and rotated
    2. Each new vectors cast a vote into a **quantized accumulator array** (AA)
    3. By finding the local maximas in AA to find the barycentres of the model instances in the target image


```
------------ scena m5 ------------
Product 25 - 2 found:
        Instance 1 { position: (233,286), width: 366px, height: 573px}
        Instance 2 { position: (567,287), width: 344px, height: 575px}
Product 1 - 2 found:
        Instance 1 { position: (496,711), width: 313px, height: 418px}
        Instance 2 { position: (836,716), width: 305px, height: 410px}
Product 11 - 1 found:
        Instance 1 { position: (162,713), width: 307px, height: 404px}
Product 19 - 1 found:
        Instance 1 { position: (908,191), width: 295px, height: 382px}
```

Github repository:

https://github.com/hjcSteve/cvProj