Objectives:

- Detect different same shapes like chips, cans and bottles from an color image.
- Implement traditional image processing for feature-based detection.
- Enhance preprocessing techniques to improve target object visibility.
- Achieve high accuracy in object segmentation without false positives or negatives.
- Create a user-friendly interface for system adjustments and feedback.

Introduction:

In real-world applications like retail inventory management and automated checkout systems, accurately detecting and classifying items such as bottles, cans, and chips from images is crucial. This task involves identifying and distinguishing between different items and localizing them accurately. The challenge increases when items are placed on shelves with similar backgrounds, complicating the isolation and recognition of individual objects. For this project, I use Spyder IDE and several libraries to handle image processing, GUI creation, and scientific computing. Below is the list of libraries: sys: Provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. cv2 (OpenCV): A powerful library used for image processing and computer vision tasks. numpy: A fundamental package for scientific computing with Python, providing support for arrays and matrices. PyQt5.QtWidgets: Qapplication, QWidgetQPushButton,Qlabel,QVBoxLayout,QHBoxLayout,QFileDialog,PyQt5.QtGui,Qpixmap,Qimage,PyQt5.QtCore,Qt,
Methodology:

Step-1:

Check if an Image is Loaded ,then read that Image and convert the Image to Grayscale.

Step-2:(Apply Thresholding)

Converts the grayscale image to a binary image, making it easier to identify and isolate objects of interest.

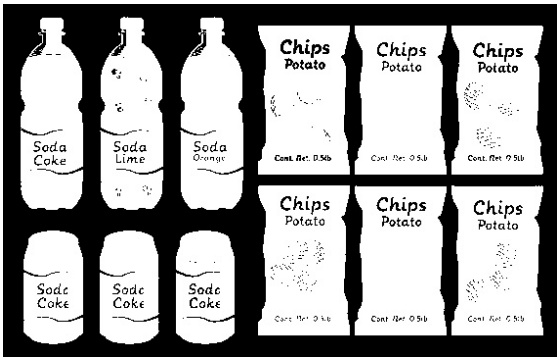
How It Works:

- At first a specific pixel intensity value (threshold) is chosen (e.g., 220). This value is used to differentiate between foreground (objects) and background.
- Each pixel in the grayscale image is compared to the threshold value.

- If the pixel value is greater than the threshold, it is set to 0 (black) in the binary image.

- If the pixel value is less than or equal to the threshold, it is set to 255 (white) in the binary image.

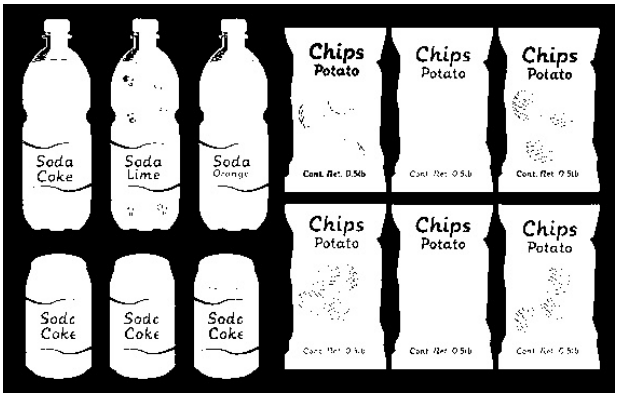
Input image



Binary image

Step-3:(Find Contours)

- The **cv2.findContours** function processes the binary image to detect the contours. It scans the image from top to bottom, left to right, to find the boundaries of white objects.
- A contour is a curve joining all the continuous points (along the boundary), having the same color or intensity. Essentially, it traces the edges of the objects in the binary image.



Binary image contoured image

Step-4: (Process Each Contour)

- Now iterate over each contour detected in the binary image. Contours are simply the boundaries of objects found in the image.
- Filter Small Contours. Now check if the contour area is greater than a minimum area (min_area). This step filters out noise or very small objects that are not relevant.
- Make Bounding Rectangle. For each valid contour, now calculate the bounding rectangle. This rectangle is the smallest rectangle that can contain the contour.
- Mathematically, the bounding rectangle is defined by the top-left corner (x, y) and the width w and height h.

Step-5:(Extract ROI (Region of Interest))

The region of interest (ROI) is extracted from the grayscale image using the coordinates of the bounding rectangle. Extracting the ROI involves isolating a specific rectangular area from an image. Here's a step-by-step mathematical explanation:

- To extract the ROI, we use the coordinates of the bounding rectangle. The top-left corner of the ROI in the image is (x, y).

- The bottom-right corner of the ROI in the image is (x + w, y + h).
- To extract the ROI from the image, we slice this array.

- The ROI is defined as image[y:y+h, x:x+w].

- y:y+h selects the rows from y to y + h (not inclusive of y + h).

- x:x+w selects the columns from x to x + w (not inclusive of x + w).

- Now separate that region in another window.

Contoured image

Segmented image

Step-6: (Template Matching:)

Template matching is a technique in computer vision used to identify parts of an image that match a template image. Here's an explanation of how it works, specifically using SIFT (Scale-Invariant Feature Transform) and FLANN (Fast Library for Approximate Nearest Neighbors):

Feature Detection and Description:

- **SIFT (Scale-Invariant Feature Transform)**: SIFT is a feature detection algorithm that identifies key points in an image and extracts distinctive descriptors for those key points. These descriptors are invariant to scale and rotation, making them robust for matching objects in images taken from different viewpoints or under different lighting conditions.
- For both the template and the target image, SIFT detects key points and computes their descriptors.

- **Key Points and Descriptors**:
- These are specific, highly distinctive points in the image (e.g., corners, edges) identified by SIFT. Each key point is characterized by its location, scale, and orientation.
- For each key point, SIFT generates a descriptor, which is a vector that encapsulates the local image gradients around the key point. This vector is used to match key points between different images.
- **Descriptor Matching**:
- **FLANN (Fast Library for Approximate Nearest Neighbors)**: FLANN is an efficient library for finding approximate nearest neighbors in high-dimensional spaces. It is used to find matches between the descriptors of the template and the target image.

k-NN Matching: FLANN performs k-nearest neighbors matching to find the closest descriptors between the template and the target image. For each descriptor in the template, it finds the two closest descriptors in the target image (k=2).

Filtering Matches:

- **Distance Ratio Test**: To filter out poor matches, a distance ratio test is applied. For each pair of matches found, if the distance of the closest match is significantly less than the distance of the second-closest match (typically less than 0.7 times the second-closest distance), it is considered a good match.
- This ratio test helps to eliminate ambiguous matches and retain only the most reliable correspondences between the template and the target image.

Counting Matches:

- The final step is to count the number of good matches that pass the distance ratio test. This count is used as a measure of similarity between the template and the target image.
- A higher number of good matches indicates a higher similarity and suggests that the template is likely present in the target image.

Input image

detected image

Obstacles:

1. **Using region descriptor rather feature description:**

- using region descriptors every items are detected as same items,

- I use normalize process for train images to train…but iste not properly working. That's why I use feature descriptors.

Input image

detected image

2. **Connected Components Not Detected:**

- Items placed next to each other or touching make it difficult for the algorithm to separate them as individual objects.

- Leads to inaccurate detection and classification since multiple items may be grouped as a single entity.

Input image detected image

3. Disoriented Components:

- Items not aligned consistently (e.g., rotated, tilted, or partially occluded) are harder to match with predefined templates.
- Reduced accuracy in template matching and item identification.

Input image detected image

4. Using Clustering algorithm:

- We utilized image processing and clustering to detect and classify items like bottles, cans, and chips. Key steps included converting the image to grayscale, applying Gaussian blur, edge detection, and extracting features from contours.

KMeans clustering grouped these features, allowing us to label and visualize detected items with bounding boxes.

Input image

detected image

Conclusion:

- The project successfully developed an image processing application to detect and classify different items (bottles, cans, chips) using feature descriptors.
- Though image thresholding, contour detection, and shape-based descriptors, the application effectively isolated and but can't recognized items properly.
- While the system demonstrated accurate detection in controlled environments, challenges such as distinguishing items connected by shelf lines and variability in item orientation and size were noted.
- Future enhancements will focus on improving robustness against varying backgrounds and lighting conditions, integrating machine learning models for advanced recognition, and refining preprocessing techniques to handle connected items more effectively.