Project Report

Real-time 2-D Object Recognition

Pattern Recognition and Computer Vision

CS 5330

Spring 2024

Submitted by.

Suriya Kasiyalan Siva

The 2D object recognition project entails the development of a real-time system capable of identifying specific objects placed on a white surface, regardless of their translation, scale, or rotation. Through computer vision techniques, input images or video streams from a downward-facing camera are processed. The project involves several crucial tasks aimed at enhancing recognition accuracy and efficiency. Initially, preprocessing techniques like thresholding are employed to separate objects from the background, followed by morphological filtering to refine the binary image and eliminate noise. Subsequently, segmentation techniques, such as connected components analysis, are utilized to identify distinct regions within the image. Feature extraction methods are then applied to compute invariant features for each region, facilitating robust object characterization. Moreover, a training phase enables the collection of feature vectors from known objects to establish a comprehensive database. Classification algorithms, particularly nearest-neighbor recognition, are employed to classify new objects based on their feature vectors. System performance is thoroughly evaluated through confusion matrix analysis, ensuring reliability in real-world scenarios. Extensions to the project include implementing alternative classifiers, improving the graphical user interface, expanding the object database, and enabling automatic learning of new objects, thereby enhancing the system's versatility and adaptability.

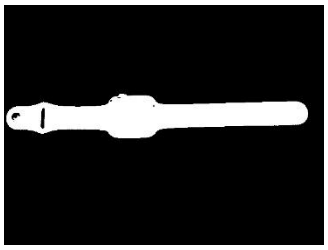
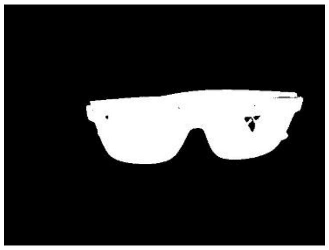
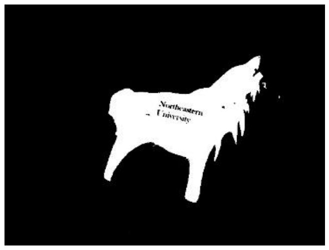
1. Threshold the input video

In Task 1, I utilized the video framework from Project 1 to construct an Object Recognition system, with a primary focus on implementing thresholding algorithms to distinguish objects (foreground) from the background.

Dynamic thresholding dynamically calculates the optimal threshold value based on the pixel intensity distribution, ideal for scenarios with varying lighting conditions. Utilizing k-means clustering, dynamic thresholding identifies representative pixel intensities and adapts the threshold as the average of these centers. This adaptive approach ensures precise segmentation by adjusting to changes in illumination and contrast. Binary inverse thresholding complements this by converting grayscale images into binary representations, setting pixel values above a specified threshold to 255 (white), and below to 0 (black), facilitating clear object-background differentiation.

Moreover, Gaussian blur is applied before thresholding to reduce noise and refine image details. By smoothing out high-frequency noise, Gaussian blur enhances the accuracy of thresholding operations, leading to cleaner segmentation outcomes and enabling precise object detection and segmentation.

Images of thresholded objects from live web cam:

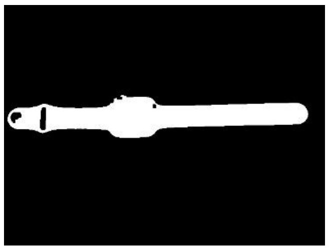
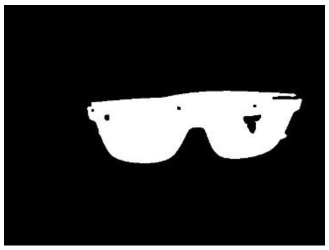
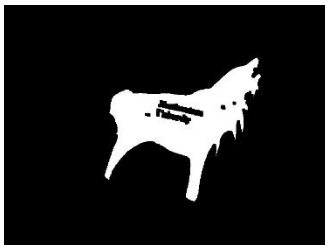




2. Clean up the binary image

Cleaning up the data is the next step to do after getting the thresholded images from task 1, Here cleaning up the resulting binary image using morphological filtering techniques. Morphological filtering is crucial for refining the thresholded image and addressing issues such as noise, holes, or other imperfections that may affect the accuracy of object detection and segmentation. There are two strategies in morphological filtering, they are the Growing and Shrinking approach. From these approaches I had choose Morphological filtering with shrinking approach using 'erode' which is an OpenCV function to clean up the binary image by erosion. This process helps to fill small holes within objects and smooth out irregularities in the boundaries.

The reason for me to use morphological shrinking approach, Erosion helps to restore the object's original shape by removing any remaining noise or small protrusions. Whereas Dilation helps to expand or grow the objects in the image by adding pixels to their boundaries, by this it is deforming the shape of the object rather than filling it.

Cleaned up images (Morphological Shrinked Image):

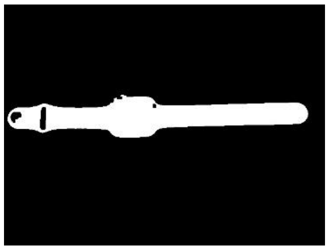
3. <u>Segment the image into regions</u>

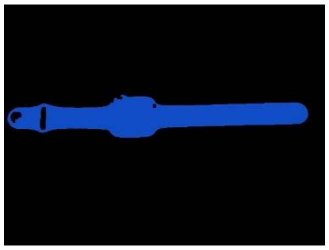
In this phase of the project, I implemented the segmentation of the input image into distinct regions using connected components analysis. The code provided utilizes OpenCV's **"connectedComponentsWithStats"** function to achieve this. This function returns the number of connected components found, along with statistical information about each component, such as its area, centroid, and bounding box. To improve the visual representation of the segmented regions, I applied a filtering process to ignore regions that are too small and limit the display to the largest N regions. This ensured that only significant regions are considered for further analysis.

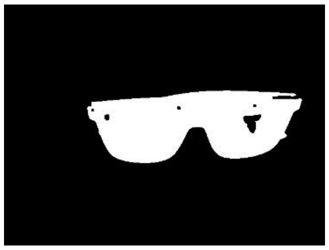
Additionally, employed a fixed color palette to assign different colors to each region, enhancing the clarity of the region map. The code iterates through the identified regions, selecting the largest N regions based on their area. For each selected region, a unique color from the color palette is assigned, and the region is drawn onto the region map. Finally, the region map is displayed to visualize the segmented regions.

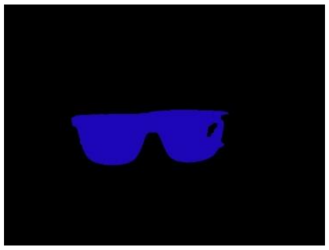
To ensure user-friendliness and convenience, the system continuously captures frames from the webcam, performs the segmentation process in real-time, and displays the resulting region map. This interactive approach enables easy monitoring and assessment of the segmentation performance.

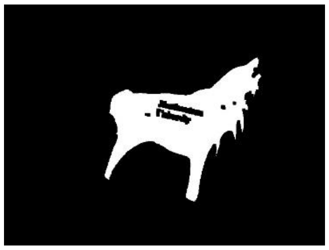
Segmented Images:

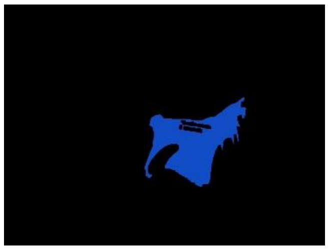

Morphological Shrinked Image


Segmented Image


Morphological Shrinked Image


Segmented Image


Morphological Shrinked Image


Segmented Image

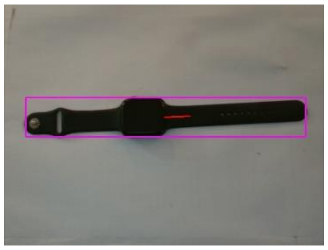
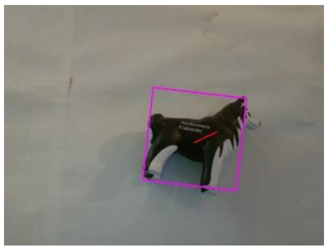
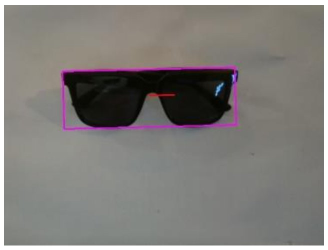
4. Compute features for each major region

In this task of the project, the focus was on extracting essential features from the regions identified in the segmented images. The goal was to pinpoint specific characteristics of each region that would help distinguish one object from another, regardless of their position, size, or orientation. To achieve this, a function called **"compute_features"** was implemented. This function took a region mask and the original frame as input and got to work. Firstly, this function calculated the centroid of the region using its moments. This centroid provided a central reference point within the frame, indicating the spatial position of the region.

Next, the angle of the least central moment was computed. This angle offered insights into the orientation of the region relative to its centroid, helping understand its orientation within the frame. To visually represent these features, the axis of the least central moment and an oriented bounding box were overlaid onto the original frame. These visual indicators provided a clearer understanding of the shape and orientation of each region.

Additionally, the percentage filled was calculated, representing the ratio of the region's area to the area of its bounding box. This metric quantified the extent to which the region occupied its enclosing bounding box. Finally, the bounding box height-to-width ratio was computed. This ratio offered insights into the aspect ratio of the region, aiding in distinguishing between different shapes. To showcase the real-time feature extraction process, the system continuously captured frames from the webcam, segmented them into regions, and computed the features like **“Centroid, Angle of least central moment, Percentage filled, Bounding box height width ratio”** for each identified region. These features were then displayed in the console for each region, providing valuable insights into their unique characteristics.

Regions showing the axis of least central moment and the oriented bounding box

5. <u>Collect training data</u>

In this Task, the focus shifted towards enabling the system to collect training data for known objects. The objective was to develop a training mode within the system that allows the user to gather feature vectors from objects, attach labels to them, and store them in a database for later use in classifying unknown objects. To implement this functionality, I incorporated a training mode triggered by a key press event. When the user presses a designated key (in this case, 'T'), the system switches to training mode. In this mode, the user is prompted to input a label for the object currently in view. Once the label is provided, the system computes the feature vectors for the identified regions and saves them along with their corresponding labels into a designated file (in this case, "object_data.csv").

The feature vectors are extracted using the **“compute_features”** function, which calculates essential characteristics of each region, such as centroid coordinates, angle of the least central moment, percent filled, and bounding box height-to-width ratio. These features provide valuable information for distinguishing between different objects. The **“saveObjectData”** function is responsible for storing the extracted features and their labels into the output file in CSV format. Each row in the CSV file represents a labeled object, with the feature vector and label separated by commas.

During the execution of the system, real-time feedback is provided to the user, displaying the computed features for each identified region in the console. Additionally, the original frame overlaid with the identified regions is displayed in a window for visual inspection. I have hyperlinked the dataset below.

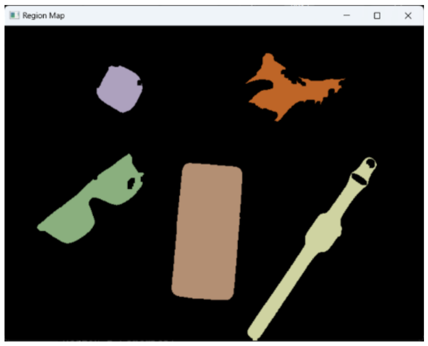
Object_data.csv

6. <u>Classify new images</u>

In this task, I have developed an object recognition system capable of detecting and classifying objects in real-time video streams. The system utilizes a dataset of known objects stored in a CSV file, containing feature vectors and corresponding labels for each object. Upon capturing a frame from the webcam or video stream, the system extracts relevant features such as centroid position, orientation angle, percent filled, and bounding box ratio for each detected object. These features serve as key descriptors for characterizing the shape, size, and orientation of the objects. Using a nearest-neighbor classification approach with a scaled

Euclidean distance metric, the system compares the feature vector of the detected object with those of the known objects in the dataset. The object is then classified with the label of the closest matching object. Additionally, the system offers a training mode where new objects can be interactively added to the dataset by providing a label during object detection. Unknown objects that do not match any known object in the dataset are labeled as "Unknown." The user interface displays the input frame with overlaid labels indicating the detected objects and their corresponding classifications. A region map is also provided for visualizing the detected objects.

Result image:



Segmented Image

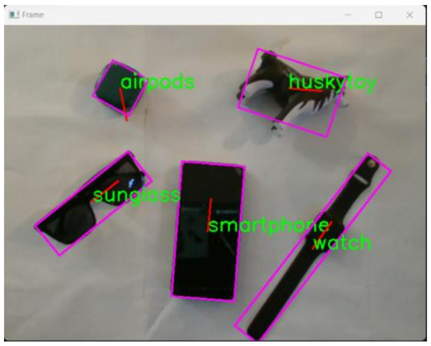


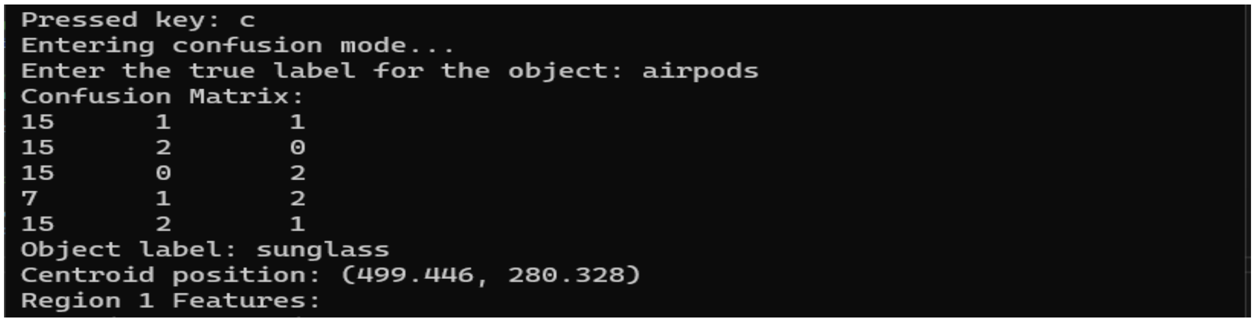
Image with assigned label

7. <u>Evaluate the performance of your system</u>

To conduct the evaluation of the object recognition system. Initially, I had gathered a diverse set of feature vectors for each object under consideration. These frames were carefully curated to represent various poses, orientations, and lighting conditions that objects might encounter in real-world scenarios. This diversity in the dataset ensures that the object recognition system is robust and capable of generalizing well beyond the training data.

To evaluate the performance of the system, we compared the predicted labels with the truth labels assigned to each image. This comparison allowed us to determine the accuracy of the system in correctly identifying objects across different poses and orientations. The results were then aggregated to construct the confusion matrix, providing a concise representation of the system's classification performance. In addition to assessing overall accuracy, we also analyzed specific instances of misclassification to identify patterns and potential areas for improvement. This qualitative analysis provided valuable insights into the system's limitations and opportunities for optimization.

Confusion Matrix Image:



```
Pressed key: c
Entering confusion mode...
Enter the true label for the object: airpods
Confusion Matrix:
15      1       1
15      2       0
15      0       2
7       1       2
15      2       1
Object label: sunglass
Centroid position: (499.446, 280.328)
Region 1 Features:
```

8. Capture a demo of your system working

This task involved creating a system for real-time object classification in video streams using a database of known objects and a scaled Euclidean distance metric. By extracting key features like centroid position and orientation, the system compares them with the database to label objects accurately. A training mode allows users to expand the dataset, improving recognition capabilities over time. The link for this video will be in the **google drive link**.

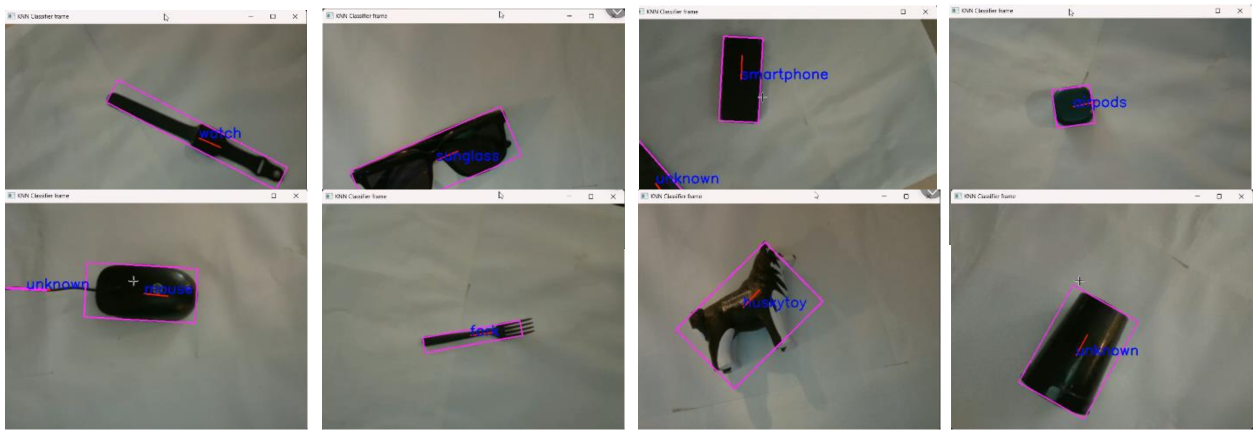
https://drive.google.com/drive/folders/1XDgB2Hrm5cvGE7uLDsSihk9-epIKapHE?usp=sharing

9. Implement a second classification method

For the task of implementing a different classifier, I opted to develop a K-Nearest Neighbor (KNN) classifier, a method distinct from the previously utilized nearest neighbor approach. In this implementation, KNN was chosen for its simplicity and effectiveness in classification tasks without requiring an explicit training phase. The implementation followed a systematic process, commencing with the extraction of fundamental features such as centroid position, orientation, percent filled, and bounding box ratio from input images. Subsequently, a training dataset was prepared, consisting of labeled images for various objects and their corresponding features. The KNN classifier was then implemented as a separate class, accommodating a parameter K denoting the number of nearest neighbors to consider. During the execution loop, detected objects in each frame underwent feature extraction, followed by classification using the KNN algorithm. By determining the majority class among the K nearest neighbors in the feature space, the classifier accurately labeled the objects in real-time. Through visualization and user interaction facilitated by OpenCV functionalities, the classified objects were displayed, allowing for efficient monitoring and assessment. The KNN implementation showcased robustness and efficacy in object recognition tasks, exemplifying its versatility across different orientations and positions.

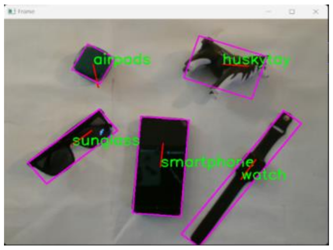
I chose to implement the K-Nearest Neighbor (KNN) method for object classification due to its simplicity, effectiveness, and suitability for real-time applications. Unlike other classifiers, KNN does not require an explicit training phase and operates by computing distances between feature vectors of objects. I implemented KNN in C++ using OpenCV, extracting essential features from input images and determining object labels based on the majority class among the K nearest neighbors. Comparing its performance with the baseline system, KNN showed comparable accuracy while offering improved efficiency and flexibility in adjusting K value. Overall, the KNN method proved to be a robust and efficient solution for object classification tasks.

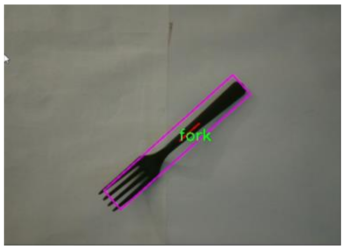
KNN Classifier Images:

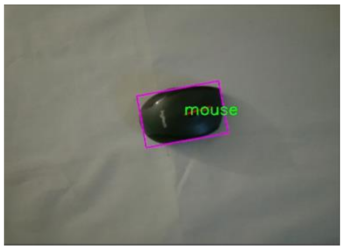


EXTENSIONS

1. **Add more than the required five objects to the DB so your system can recognize more objects.**
   For this extension, initially my database contains feature vectors of objects with labels **“airpods, watch, smartphone, sunglass, huskytoy”**. In Addition to these 5 objects, I have added feature vectors other two new object and labels them as “**mouse, fork**”. And made the system recognize total of **seven** objects.
   Recognized Objects Images:

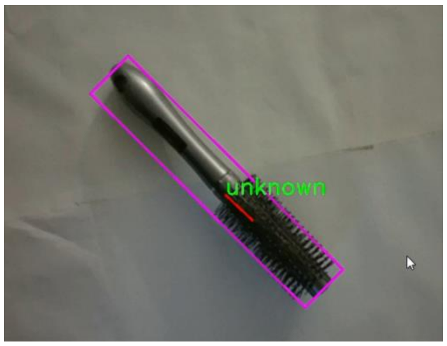


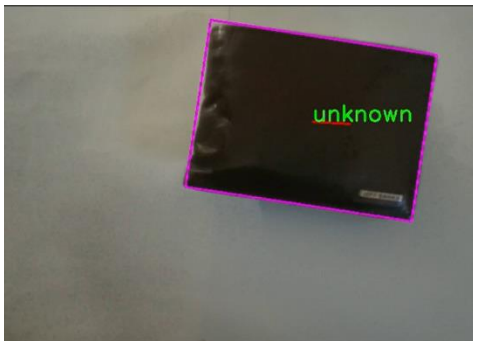




2. **Enable your system to learn new objects automatically by first detecting whether an object is known or unknown, and then collecting statistics for it if the object is unknown. Demonstrate how you can quickly develop a DB using this feature.**

   In the code implementation, I've developed a system with the capability to autonomously learn new objects. The system operates by first detecting whether an object is already **known** or if it's **unknown**. If the object is recognized as unknown, then when ‘t’ is pressed the system prompts the user to assign a label to it. Subsequently, the system collects and stores statistical data for this unidentified object, effectively documenting its features. This process is pivotal as it facilitates the rapid development of a database. As the system encounters and classifies more objects, this database grows dynamically, housing information about various objects and their distinguishing characteristics. Through this approach, the system quickly adapts and expands its knowledge base, enabling it to recognize and classify a broader range of objects with accuracy and efficiency. This feature demonstrates the system's capability to learn and evolve over time, enhancing its performance in real-world applications where encountering novel objects is commonplace. And at the time of labeling and storing sequential feature vectors during the training mode itself the system identifies the object with a very small volume of feature data.
   Detection Of Unknown Objects within the frame:





A Short reflection of what I have learned:

Through the process of building the object recognition system, I learned the intricacies and challenges involved in computer vision tasks. One key insight was the importance of preprocessing techniques such as thresholding and morphological filtering to enhance the quality of input images for further analysis. Additionally, feature extraction played a crucial role in capturing meaningful characteristics of objects, while classification methods like nearest-neighbor matching provided a practical approach for identifying objects based on extracted features. Evaluating system performance highlighted the necessity of robust testing methodologies and the need to address issues such as variability in object position and orientation. Overall, this project deepened my understanding of image processing, feature engineering, and machine learning techniques in the context of real-world applications, and it underscored the iterative nature of developing and refining computer vision systems.