Sara Spasojevic
CS5330, Spring 2025
Project 3
Feb 20, 2025

Project Report

Link to video demonstration of the project: https://drive.google.com/file/d/1Iyhhwd-Bn4Pm0D12C9UOj04UVogZ9gf4/view?usp=sharing

1. Project Description

The goal of this project is real-time 2D object recognition. Given a white surface as a background, and objects placed on top of it, with the camera positioned above, the program detects objects and does classification to assign labels to them. It does this by performing a series of steps - thresholding, morphological filtering and segmenting. The regions are then analyzed to compute a feature vector for each, and based on those, they are classified with either known labels or a unique unknown label. The program can perform either **nearest neighbor (NN)** or **k-NN (k-nearest neighbors)** classification, and this is specified when running the program in terminal. In addition, the program is limited to recognize only 3 closest objects to the center of the frame (based on their centroids). The program uses iPhone's back camera (continuity feature) for real-time video feed.

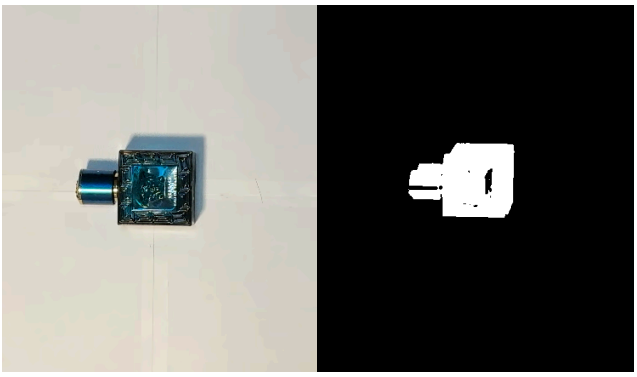
The program has two training (labeling) modes:
- **Normal** training mode - allows users to label all 3 objects in the frame.
- **Automatic** training mode (Extension 3) - allows labeling of only the unknown object in the frame.

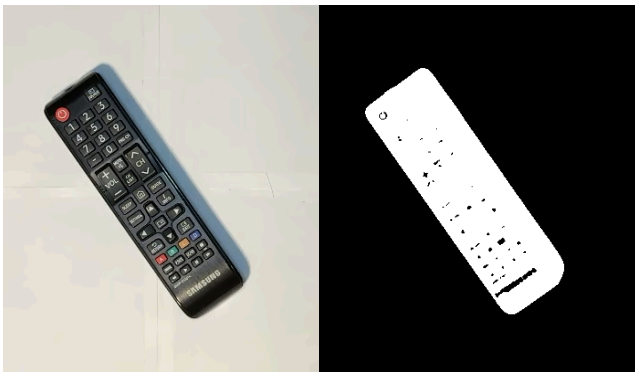
2. Required Images

Task 1. **Threshold the input video** (implemented from scratch)

- **Description**: Thresholds the input frame by **dynamically** setting the threshold by analyzing the pixel values. It uses the **ISODATA** algorithm (k-means on a 1/16 random sample of the pixels in the image) with K=2, to find the mean of the two dominant colors - the threshold is the average of those means. Then, the feed is thresholded based on this threshold. (Pre-processing steps - converted to grayscale, and blurred using 5x5 Gaussian filter).

See below: original (left) and thresholded (right).

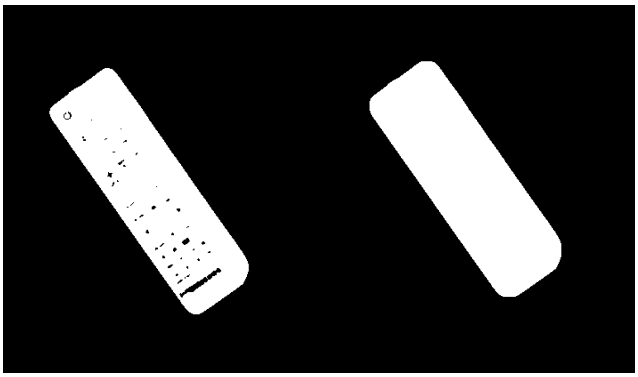




Task 2. **Clean up the binary image** (Extension 1. Implemented from scratch)

- **Description**: After thresholding the image, the regions have some “holes” and there is some noise in the background. Hence, **dilation** is applied to fill the holes in (if the pixel is in the foreground, everything under the kernel becomes foreground), and **erosion** is applied to refine the shapes and remove the noise (if the pixel is in the background, everything under the kernel becomes the background). Both erosion and dilation are 8-connected.

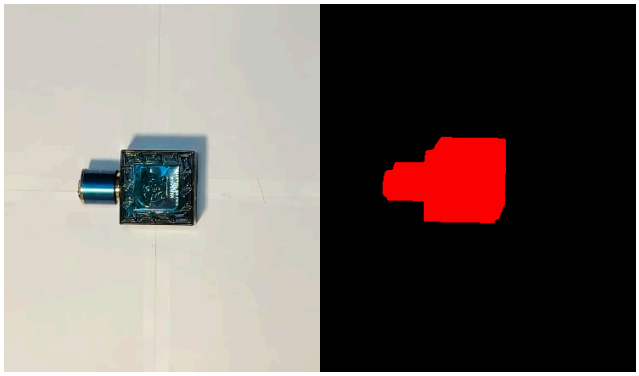
See below: thresholded (left) and cleaned up (right).



Task 3. **Segment the image into regions** (Extension 2. Implemented from scratch)

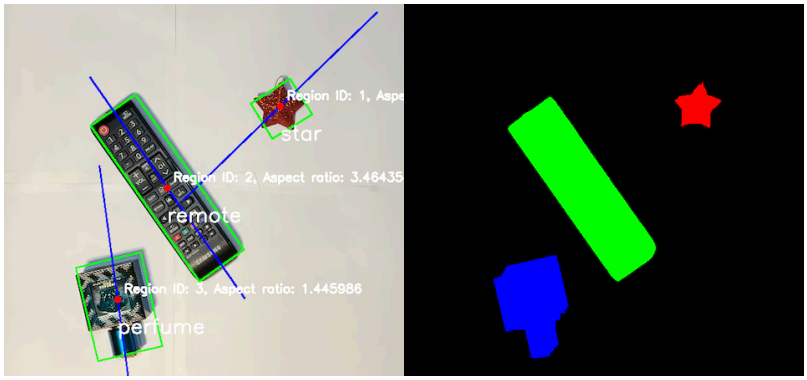
- **Description**: After the image is refined, it is now segmented. This was implemented from scratch using the **two-pass algorithm** that utilizes **Union-Find** data structure to resolve labels. The function populates the region map with labels, but it also colors the regions found. It filters the objects based on size, and also ignores the ones that enter within the margins area. In addition, it sorts the objects by their proximity to the center of the frame (based on the centroids), and returns the top 3 closest objects by their region ID for use in the rest of the program.[1]

See below: original (left) and segmented up (right).



[1] I have also implemented a similar function using OpenCV’s “connectedComponentsWithStats” to compare the lag in live video.

Another example of objects segmented is when there are multiple objects in the feed.

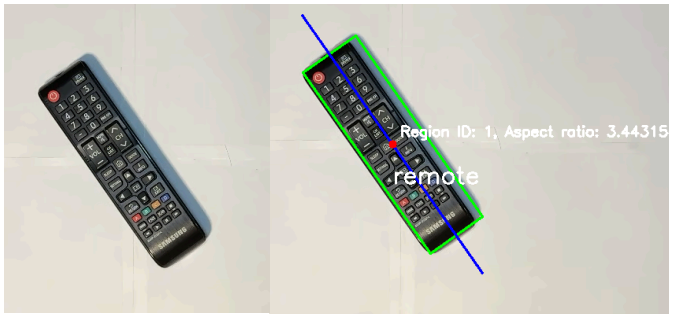


Task 4. **Compute features for each major region**

- **Description**: For each of the top 3 region IDs, this function computes the features (translation, scale, and rotation invariant) and draws the bounding box, primary axis and centroid; and prints the region ID and aspect ratio. These are then used for training and classification purposes.

- I used the OpenCV's calculation for moments, and using those, I retrieved the **area**, **centroids** and calculated the angle of the axis of least central moment, which I then used to get the unit direction vector of the **principal axis**. I retrieved the contour by checking for regionID in the region map, which I then used with minAreaRect function to get the **rotated bounding box**. With the help of the rotated bounding box and area, I retrieved the **percentage filled** and **aspect ratio**. And finally, I used the **first 3 Hu moments** as additional feature vectors.

See below: original (left) and computed features: bounding box, central axis of rotation, aspect ratio, centroid (right).



Computed feature vectors for each object

```
Label,AspectRatio,PercentFilled, HuMoment1, HuMoment2, HuMoment3
remote,4.05882,0.983449,0.358923,0.101313,2.4458e-06
ball,1.05646,0.811091,0.159811,0.000100672,5.46544e-06
perfume,1.66824,0.778499,0.192544,0.00751593,0.00113015
star,1.07996,0.547671,0.176503,0.000504954,1.62264e-05
pen,12.0407,0.586394,1.53673,2.32296,0.323025
```

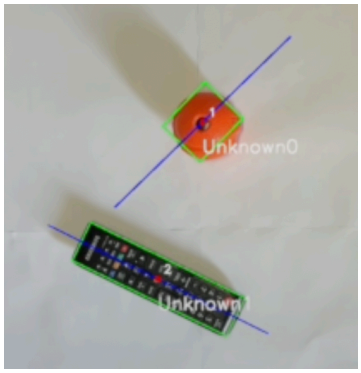
Task 5. **Collect training data**

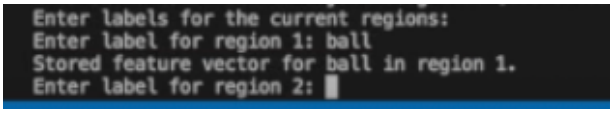
Description: The program saves each entry in a CSV file, the label, followed by the features, comma separated. There are two training modes, **normal** and **automatic** (as part of Extension 3.)

- Keypress 'n' - **normal labeling mode** - lets the user label all (up to 3 as per design choice) objects in the frame, regardless of them being known or unknown
- Keypress 'a' - **automatic labeling mode** - lets the user label only the objects that are not recognized by the program (labeled as "unknown").

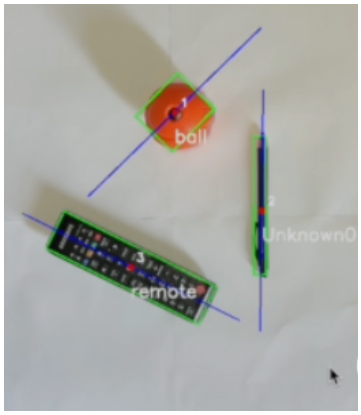
Depending on which classifying method was used, NN or k-NN, the program saves/uses either nn.csv or knn.csv file. NN file is envisioned to have only one entry per label, but it will also work if there are more, it will just use the nearest 1 neighbor. k-NN CSV file will typically have at least k labels per class.

When training mode is active, the frame is frozen and the user is prompted for the label name in terminal. After all required objects are named, the user needs to press 'b' to resume the program. Update is instantaneous, and right after resuming, the label will update (e.g. from 'unknown') to the label specified during training.





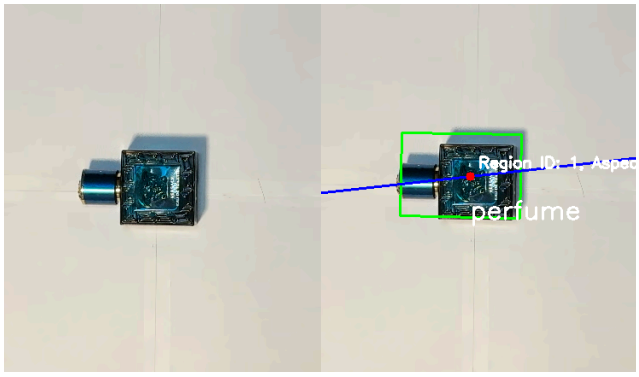
Normal Labeling Mode



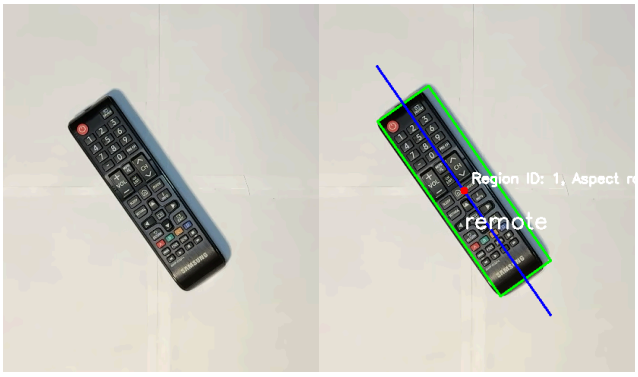
Automatic Labeling Mode

Task 6. **Classify new images**

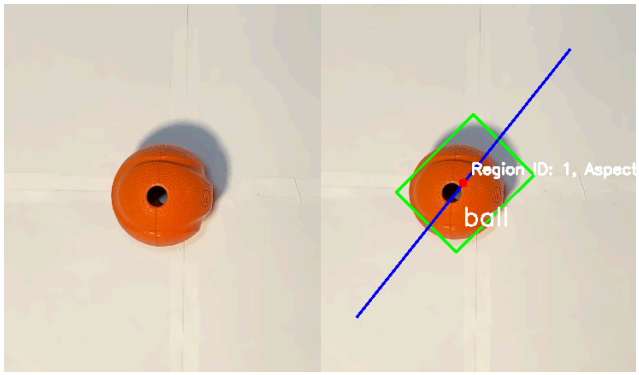
Description: The program automatically classifies all (up to 3) objects in the frame. There is no keypress needed. This function is called whenever the features are computed for each object. There is also a threshold, and if we are above this threshold, it will show the object as “unknown”. The **nearest neighbor** classification uses Scaled Euclidean Distance for the distance metric. Every time a we label an object in the training mode, **standard deviation** for each feature is re-computed, to allow for real-time accurate classifying.



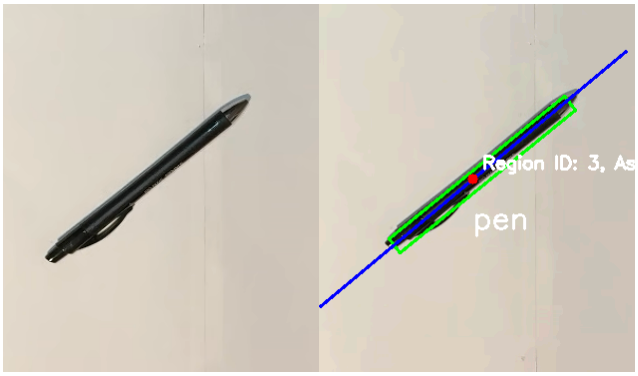
Perfume



Remote



Ball



Pen

Star

Task 7. **Evaluate the performance of your system**

- **Confusion Matrix for baseline (NN) classifier**

Actual/Predicted	Remote	Ball	Star	Perfume	Pen	Unknown
Remote	5	0	0	0	0	0
Ball	0	4	0	1	0	0
Star	0	0	5	0	0	0
Perfume	0	0	0	5	0	0
Pen	0	0	0	0	4	1

One thing I noticed with the program is that depending on the lighting and the way shadows drop, these can affect the size of the area, aspect ratio etc. because the shadows would become the part of the object. This would make the ball seem more like a perfume, since it would have a more elongated shape. In those cases, the basic classifier had more issues than the k-NN which I will discuss in task 9.

Accuracy based on above example: 92%

Task 8. **Capture a demo of your system working**

Link: https://drive.google.com/file/d/1Iyhhwd-Bn4Pm0D12C9UOj04UVogZ9gf4/view?usp=sharing

Task 9. **Implement a second classification method**

Description: The second classification method I implemented is **k-NN**. I set **k=3**, and I also used **Scaled Euclidean Distance**. I implemented version 2 of the k-NN discussed in class.

For each class, I calculated the distance of each entry to the unknown object, and summed up **k (3)** closest distances to get the distance per class. The class with the smallest sum distance is the assigned class and the new label of the item in the frame.

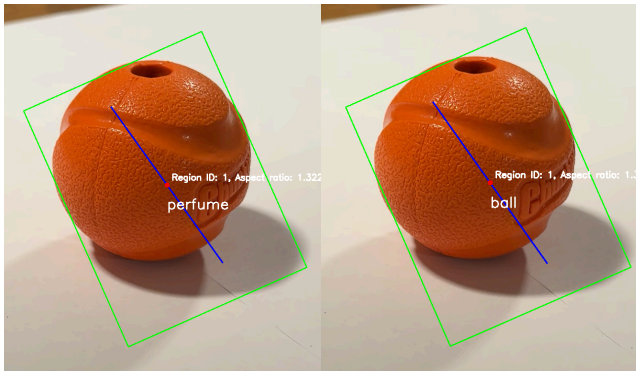
There is also a threshold that was fine-tuned, above which the object would be labeled as “unknown”.

The choice between **nn** and **k-nn** is specified when the program is started in terminal. The classification happens automatically, without a key-press needed, and the program uses the appropriate CSV file.

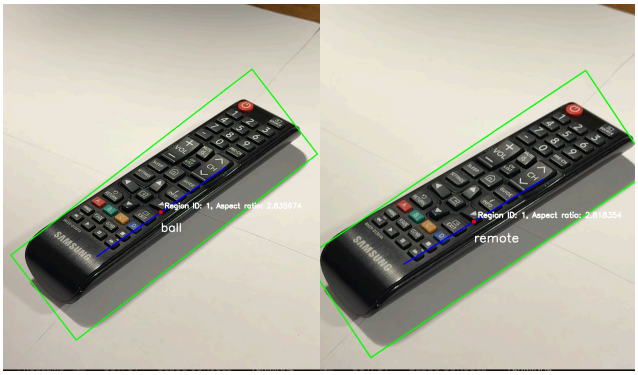
I wouldn't say the baseline performed bad, but this is probably because all of my objects were pretty distinct in shape. However, k-NN performs better than the NN regardless, since it has more data to compare to. Having only one option to match with can lead to a lot of misclassifications, but when having more neighbors, the system can be more "sure" in the choice made.

The main difference I noticed between the baseline system (NN) and k-NN classification method is that it allowed for more angles of the object to be captured. With **NN**, when I had only one example, any shadow that would "elongate" the bounding box and change the percentage filled would cause the ball, for example, to be misclassified as perfume. But when I captured an additional feature vector for the ball from a different angle, one that would show this shadow too, and saved it into the knn.csv database, my **k-NN** managed to take this into account as well, and correctly classifed the ball, even though the shadow was present. The same thing happened with the remote.

Following is the example of NN classifier (left) VS k-NN classifier (right) in case of different angles and shadows.



Ball



Remote

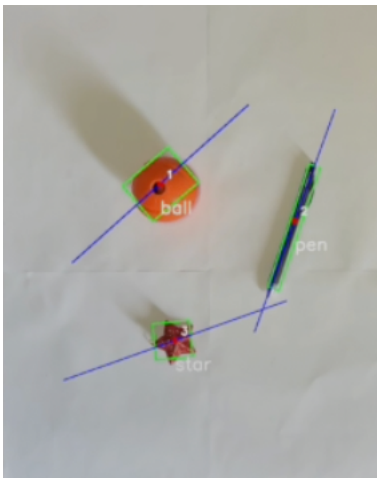
3. Extensions

Task 10. **Automatic training mode** (*Extension 3.*)

- **Description**: As mentioned above, I implemented a way for the program to recognize if the objects in the frame are known or unknown. If they are **unknown**, and if the **automatic** training mode is activated, the program will only let the user label these unknown objects, and will not ask for the known ones. This is particularly helpful when developing the training set for NN, since we do need one entry of every object in the database. This functionality is demonstrated in the video link above. The photo of this mode can be seen in task 5 above.

Task 11. **Multiple objects recognized** (*Extension 4.*)

- **Description**: The project is enabled to recognize up to 3 objects at the same time. The number could be easily changed to recognize more, but I limited it for the sake of processing.



4. Reflection and Conclusions

Overall this was a very fun project. It was interesting to see how different thresholding techniques affect the segmentation, how larger dilating kernel can better fill the object. Big takeaway from this project is that working with shadows can be a big problem, really showing the limitations of your thresholding processing (this can be seen in the images of task 1 above). If these shadows pass through, they can easily be misinterpreted as part of the object, which causes all of the features of the object to change. This can further lead to misclassifications. I also had an issue debugging Standard Deviation, since I didn't take into account having only one item in the database, and hence needed to include that edge case. In addition, working with a threshold to classify the object as either known or unknown was a process of its own, requiring me to fine-tune that parameter so it works properly with the calculation used. In case of k-NN, the higher the k, the higher the sum, and this number needs to either be increased, or normalized. Final observation is that k-NN had more flexibility in recognizing the objects from different angles, since it was able to "reference" examples of those objects from different angles in the database, whereas NN only simply relied on one example, and would easily misclassify things, if the camera was moved, or the shadow was casted.

5. Acknowledgments

- OpenCV documentation and tutorials
- Introduction to Disjoint Set (Union-Find Algorithm) - https://www.geeksforgeeks.org/introduction-to-disjoint-set-data-structure-or-union-find-algorithm/
- Unbiased estimation of standard deviation - https://en.wikipedia.org/wiki/Unbiased_estimation_of_standard_deviation
- Lecture notes for all algorithms implemented from scratch; and k-NN v2.