Report 3: Real-time Object 2-D Recognition

> Edited by Hui Hu @hui hwoo and Yao Zhong @Zhong Yao

Description

The objective of the project is to develop an object recognition system capable of identifying a set of specified objects in real-time from a camera looking down onto a white surface. The system should differentiate objects based on their 2D shape and a uniform dark color, regardless of their position, scale, or rotation. By completing each of the five tasks, the project aims to create an efficient and effective object recognition system that could be used in a variety of applications.

Tasks

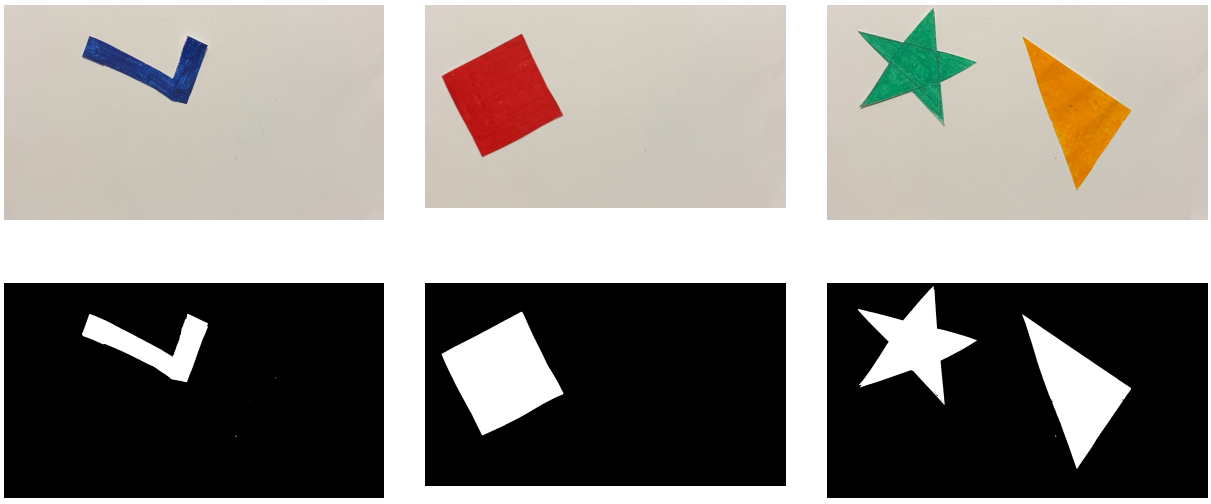
Task 1 : Threshold the input video

> Build an object recognition system using OpenCV. To implement the system, first create a thresholding algorithm to separate an object from the background. Pre-process the image by blurring it or adjusting the saturation levels. Display the thresholded video and test it on a complete set of objects to ensure functionality.

In this task, we used global thresholding. However, since each user has a different work environment, providing an optimal threshold value for all environments is not feasible. This means that it can be difficult to detect objects without resetting the threshold value when first used.

To address this problem, we proposed two solutions in this part. Firstly, we implemented a GUI for user, by draging the trackbar, the user will see the result of different threshold, and are able to deterimine the threshold to use. As an alternative we implemented an adaptive thresholding in the extension part. However, due to concerns about delay, we continue to use global thresholding in our object recognition system with GUI modifing.

Examples:(row1: original images, row2: thresholding images)



Task 2 : Clean up the binary image

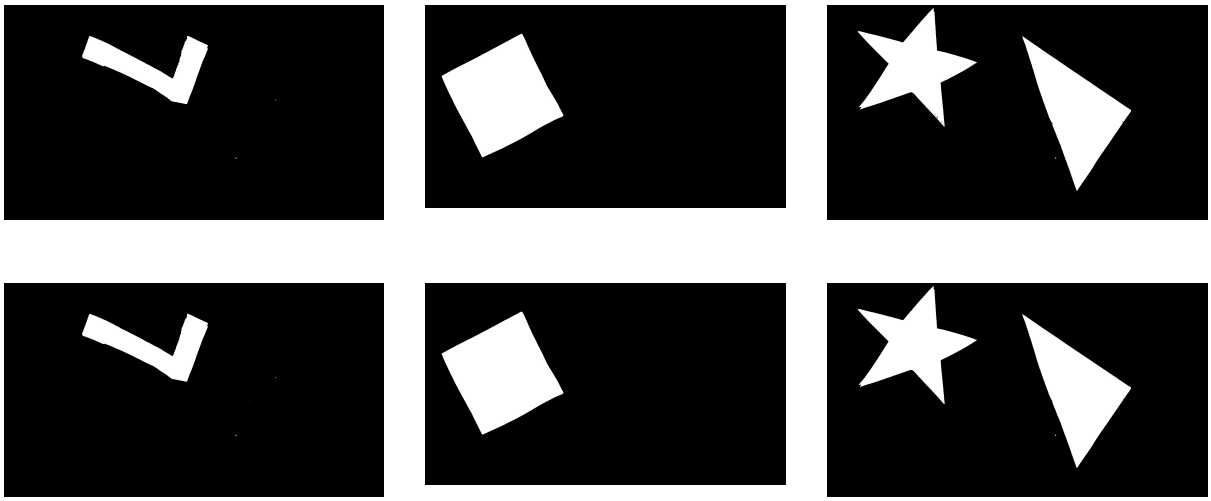
> In this task, the goal is to clean up a thresholded image using morphological filtering to address any issues such as noise or holes.

After getting the threshold images, we find some small holes in the shapes. To address holes or gaps in the thresholded image, we used morphological closing, which involves first dilating the image to fill in gaps and then eroding it to restore the original

shape while smoothing the edges. This can help to fill in small gaps or holes in objects in the image. After implementing, we find out that 2 times of dilating and 2 times of erosion is the best fit for our shapes.

After cleaning we still have some “white dots” out side main regions. These small regions are dealt in the following segmentation step.

Examples:(row1: thresholding images, row2:cleaned up images)



Task 3 : Segment the image into regions

> In this task, the objective is to perform connected components analysis on the thresholded and cleaned image to obtain regions. The system should also be able to display the detected regions.

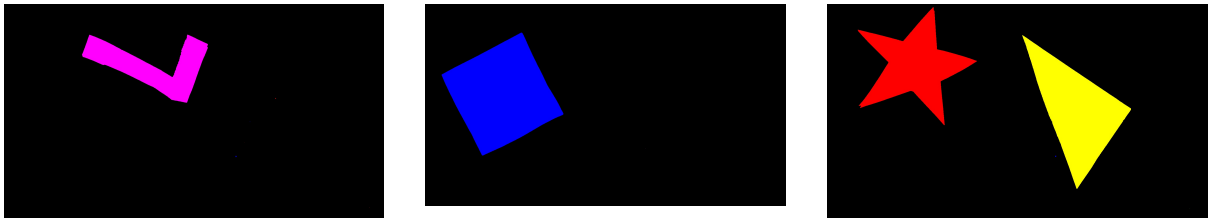
In this task, we use the `Union-Find (disjoint set)` data structure to extract the positions of the first N biggest regions from a cleaned binary image. This is a crucial step in the image processing pipeline, as it allows us to identify and isolate specific regions of interest. Once we have extracted these regions, we can calculate each region's feature vector to find its label.

The `Union-Find` algorithm works by grouping elements into sets based on some predefined criteria. In our case, we group together pixels that belong to the same region. By doing this, we can efficiently extract the positions of the largest regions in the image. However, the algorithm is not perfect and may sometimes group together pixels that do not belong to the same region. In such cases, we need to perform additional processing to remove these outliers.

Once we have the positions of these regions, we can calculate their feature vectors using a variety of techniques. These feature vectors are essentially a set of numerical values that describe the properties of each region. For example, we might calculate the average color of each region or the number of edges within the region. By comparing these feature vectors to a set of predefined labels, we can identify the contents of each region.

The followings are the segmented images, each region is colored by different shape.(We have total of 6 colors, and color is determined by regionID%6).

Examples:



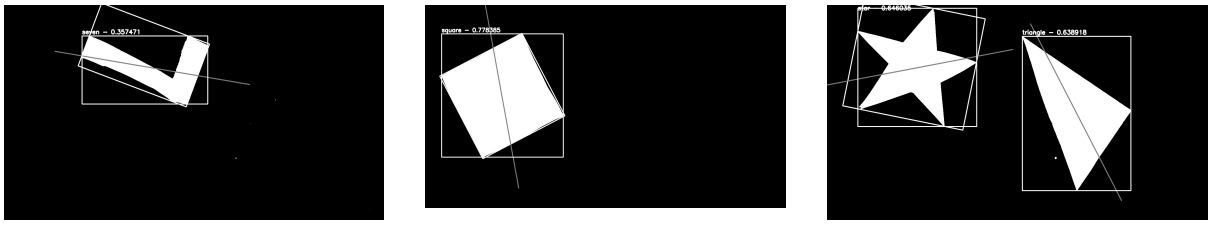
Task 4: Compute features for each major region

> The task requires creating a function that computes a set of features for a specified region using OpenCV's moments function. The function should calculate the axis of least central moment and the oriented bounding box. The features should be translation, scale, and rotation invariant. At least one feature should be displayed in real-time on the video output.

We use `Hu-moment` as our feature vector to classify different objects. `Hu-moment` is a mathematical concept that derives from the central moments of an image. It is a set of seven invariant moments that are used in image processing for shape analysis. We ensure that our feature vector only depends on one object's shape, no matter if it's rotated or not. To accomplish this, we draw the axis of least central moment on each detected object. This allows us to identify the object's major axis and thus, its orientation. Additionally, we draw the oriented bounding box, which is a rectangle that encloses the object and is aligned with the object's major axis. Finally, we calculate the first moment value for each detected object. By doing this, we can ensure that our algorithm is robust to changes in object orientation and shape, allowing us to classify objects accurately in different scenarios.

For example, in following example image, the bigger rectangle is the region we detected, the rotated smaller rectangle is its oriented bounding box, the number displayed is one of its hu-moment value.

Examples:



Task 5 : Collect training data

> In this task, the system needs to be implemented in training mode to collect feature vectors from known objects, label them, and store them in an object database. This can be done by implementing a keypress response system.

We created a training mode for our application which allowes the users to collect feature vectors from known objects. In this mode, the users are able to press a key ‘t’ to start the mode, input the label for this shape.

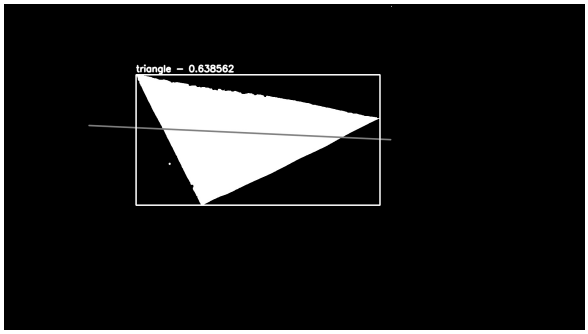
Our database consists of 10 different shapes we drawed by hand. For each shape, we saved at least 3 examples of it by moving it around and rotating them.

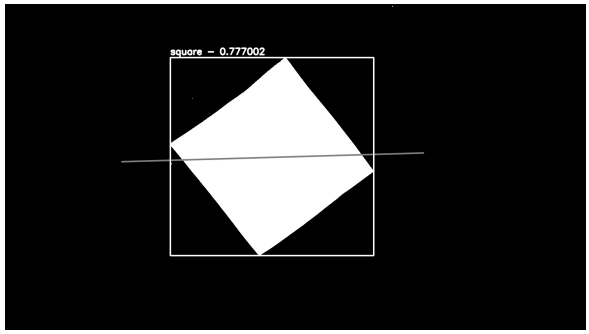
Task 6 : Classify new images

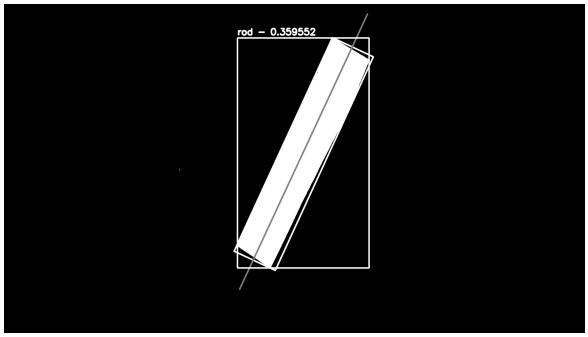
> The task requires the implementation of a classification system that can classify a new feature vector using a known object database and a scaled Euclidean distance metric. The system should label the unknown object based on the closest matching feature vector in the object DB using nearest-neighbor recognition. The system should indicate the label of the object on the output video stream.

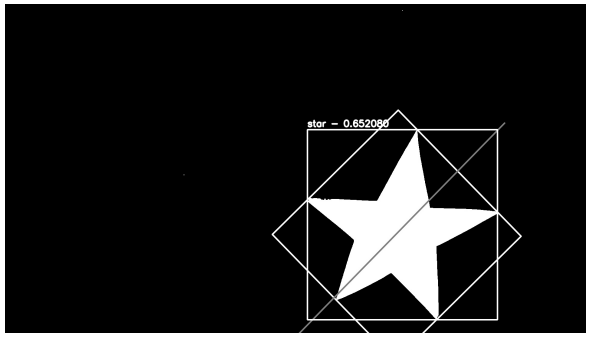
The metric we choosed for the distances is scaled Euclidean. First we calculated the standard deviation of each entry of the feature vectors in the database. Then, for the distance calculated through the following formula: $dis = \sum(\frac{|X_{1i} - X_{2i}|}{st\_dev_i})$.

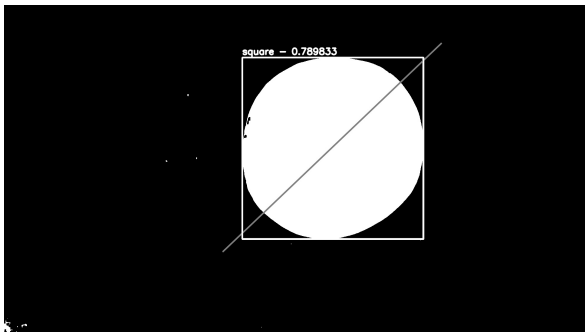
For the Nearest Neighbor match, we matched the object with the label that has the smalleest distance with the object feature.

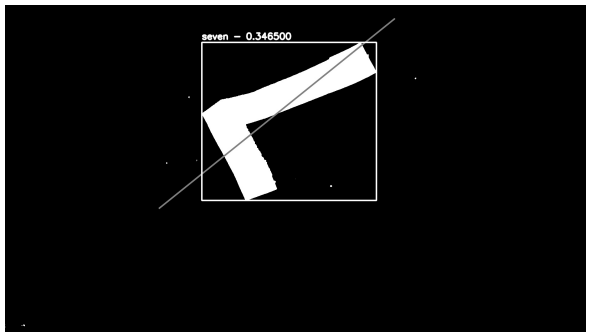

triangle – 0.638562

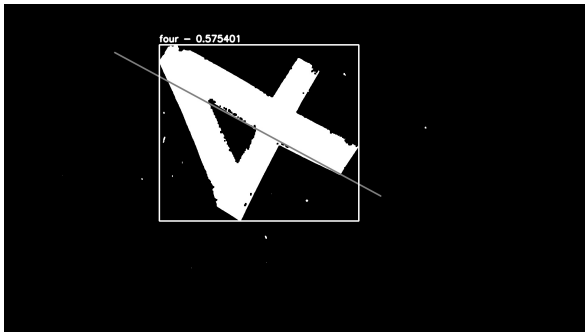

square – 0.777002

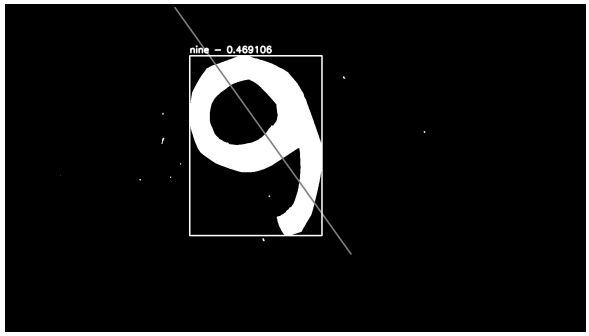

rod – 0.359552

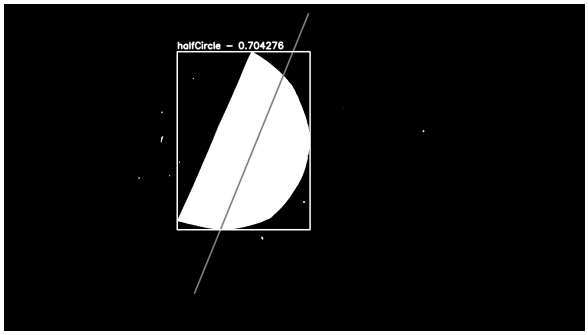

star – 0.652080

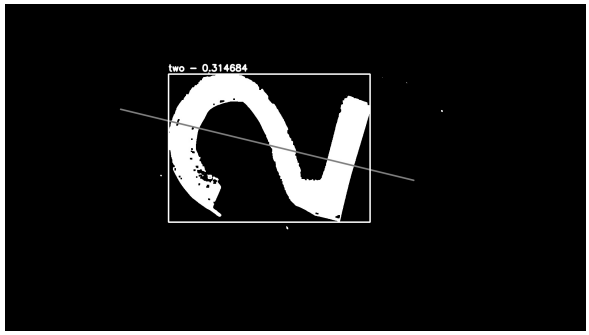

square – 0.789833


seven – 0.346500


four – 0.575401


nine – 0.469106

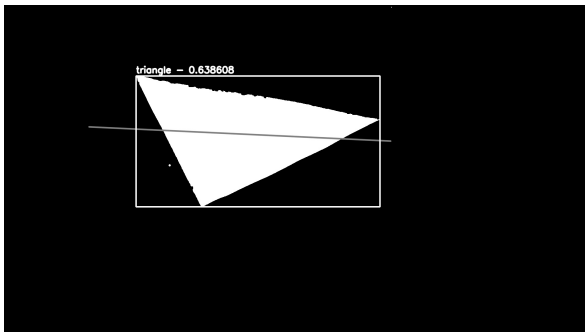

halfCircle – 0.704276

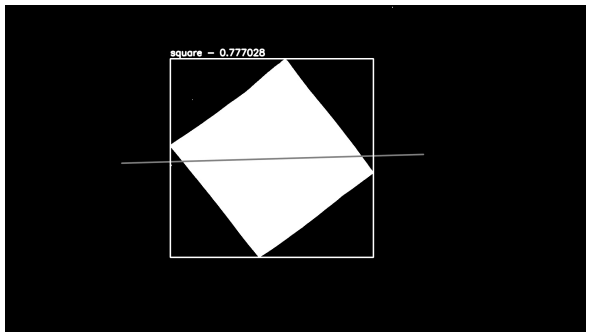

two – 0.314684

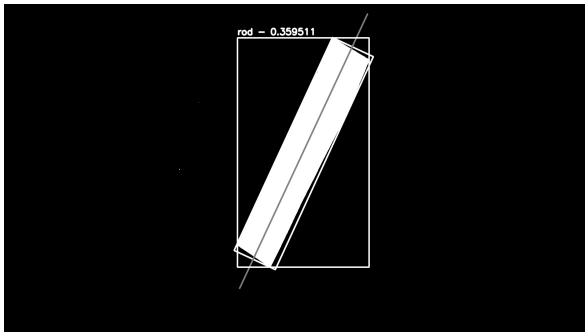
Task 7 : Implement a different classifier

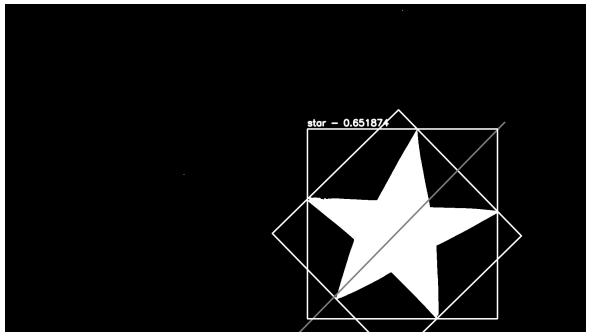
We use the `k-nearest neighbors (KNN)` (k = 3) algorithm as our second method for classifying objects and determining their labels.

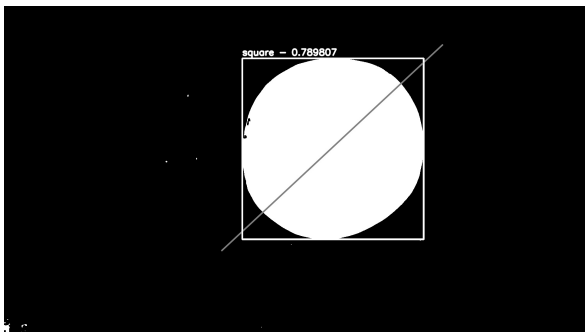
We used the same metric for each label. Differently from nearest neighbor, for each label, we consider the least 3 distance and take the average of them as the distance between the object with this label. And then, among all the labels, choose the minimum distance label as the matching result.

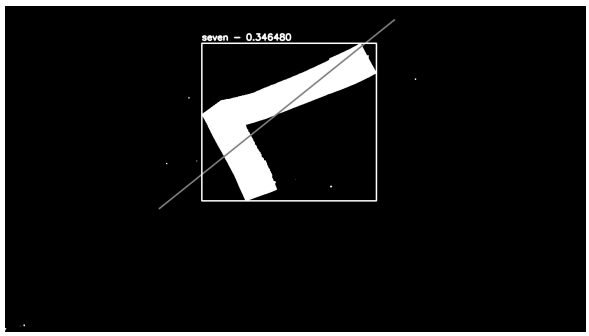


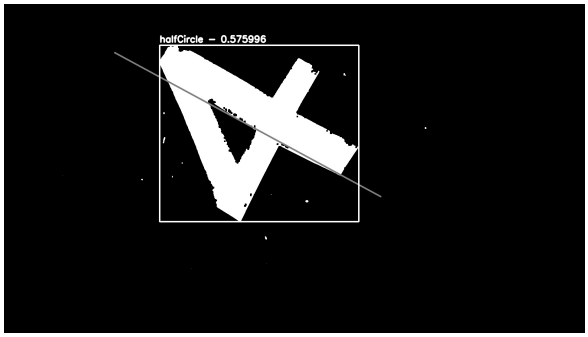


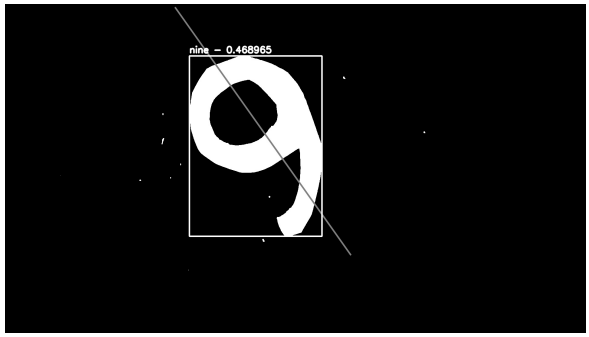


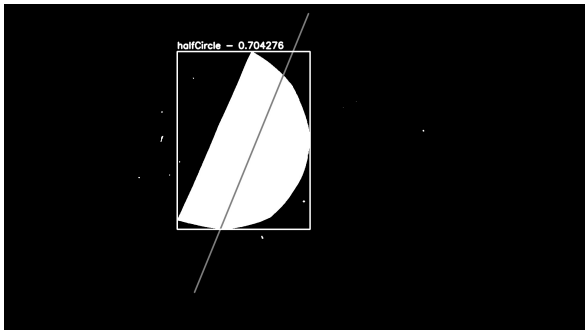


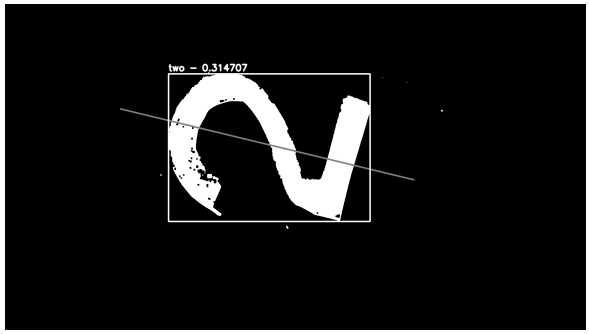












Task 8 : Evaluate the performance of your system

Evalueate Nearest Neighbor(cols are expected label(object), rows are the result label)

	triangle	square	rod	star	circle	seven	four
triangle	2						
square		2			2		
rod			2				
star				1			
circle							
seven						2	
four							2
nine							
half-circle				1			
two							

Generally, the nearest neighbor classifier is accurate. But for some of the shapes, it cannot differenciate correctly, the “star” will be falsely recognized as “half-circle”, and “circle”(actually is is a round shape) will be falsely recognized as “square”. Accuracy is 17/20 = 0.85

Evalueate Nearest K Neighbor(K=3, cols are expected label(object), rows are the result label)

Shape	triangle	square	rod	star	circle	seven	four
triangle	2						
square		2			1		
rod							
star				2			
circle					1		
seven						2	
four							2
nine							
half-circle							
two			2				

Generally, the nearest 3 neighbor classifier is accurate. But for some of the shapes, it cannot differenciate correctly, the “rod” will be falsely recognized as number “two”, and “circle”(actually is is a round shape) will be falsely recognized as “square”. And the number “nine” will be falsely recognized as “triangle”. The accuracy is 16/20 = 0.8.

Task 9 : Capture a demo of your system working

The demo video is in this link on YouTube: https://youtu.be/O6WMgxuzKnc

Extensions

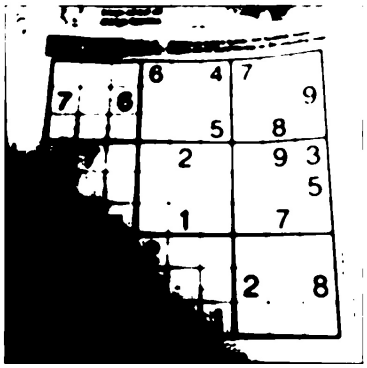
Extension 1 : Adapative Threshold

> Since the result of global threshold is not ideal, we use Adapative Threshold to get a more accurate result.

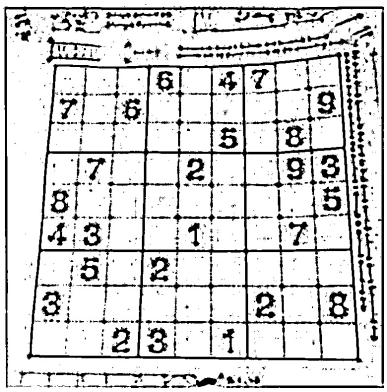
Since Global thresholding does not take into account variations in contrast and illumination across different parts of the image, which can lead to inaccuracies in the segmentation results. This can be particularly problematic in images with complex backgrounds or where the object of interest has low contrast or irregular shape.

To address these limitations, we used more advanced thresholding techniques, adaptive thresholding, which take into account local image statistics to determine the threshold value, resulting in more accurate segmentation results even in cases where the background and foreground intensities are not clearly separated.

Original image



Global threshold



Adapative Threshold

After careful observation of the images provided, it is evident that while the global thresholding technique can produce a satisfactory binary image, it tends to lose some of the fine details within the image. On the other hand, the adaptive thresholding method not only retains all the important details present in the image but also enhances the brightness and contrast, making it more informative and visually appealing.

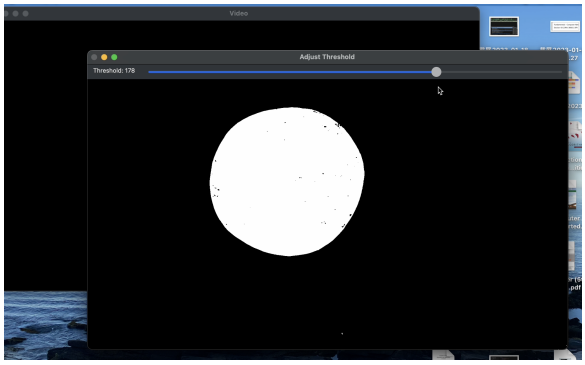
Furthermore, it is worth noting that the global thresholding technique is suitable for simple, uniform images that do not contain a lot of variations in lighting and texture. However, for more complex images that require accurate segmentation, adaptive thresholding is the preferred method, as it can adjust to changes in lighting and texture and produce a more precise binary image.

However, considering the latency to get a better binary image, we still use global thresholding technique in our program.

Extension 2 : Better GUI to modify threshold value

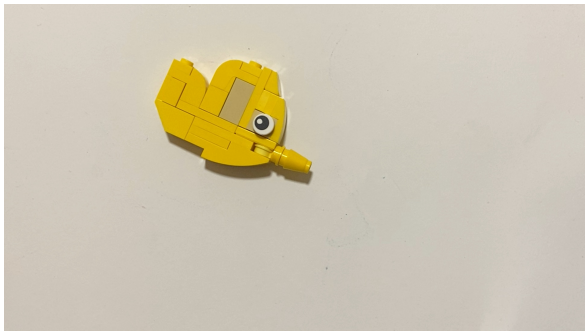
In our program, we used another way for users to adjust the threshold. From GUI we made, by draging the trackbar, the user will see the result of different threshold, and are able to deterimine the threshold to use.

Example:

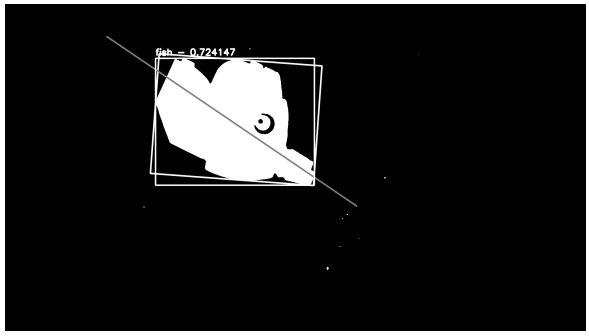


Extension 3 : More than 10 detectable object

I have a bunch of small lego fishes with diffrent shapes so I tried to save the features of them and then recogize each of them with our application. The following result is from nearest 3 neighbors. The accuracy is not bad!

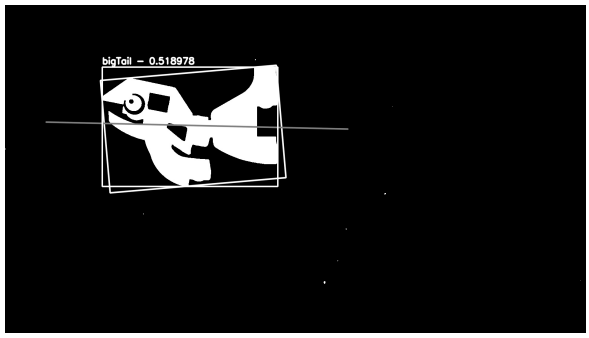


Labeled as: fish

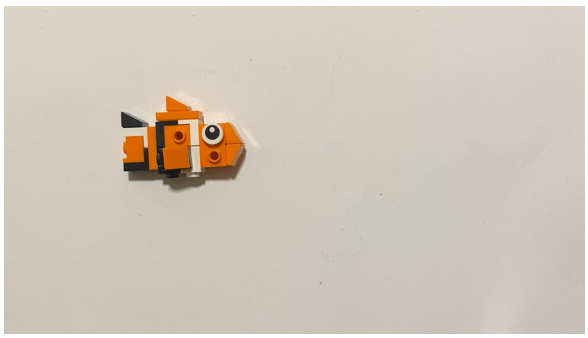


Recognized as: fish

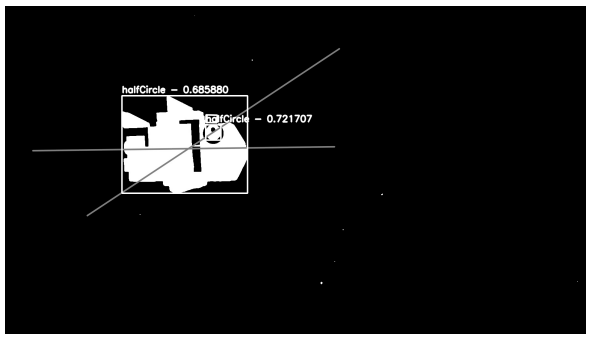
Labeled as: bigTail



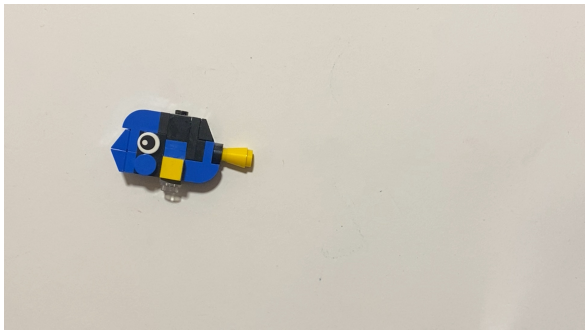
Recognied as: bigTail



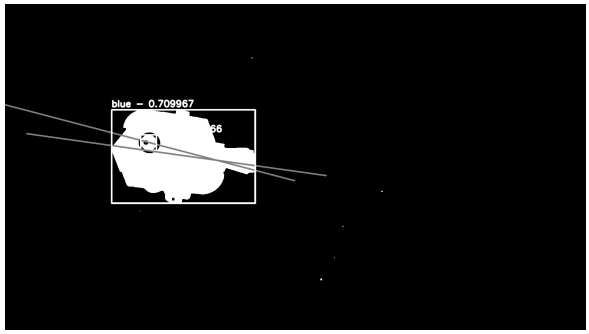
Labeled as: nemo



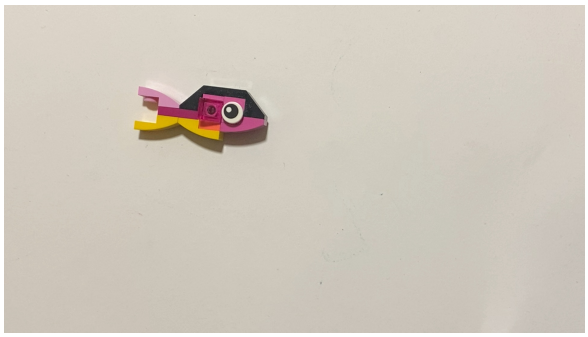
Recognied as:halfCircle



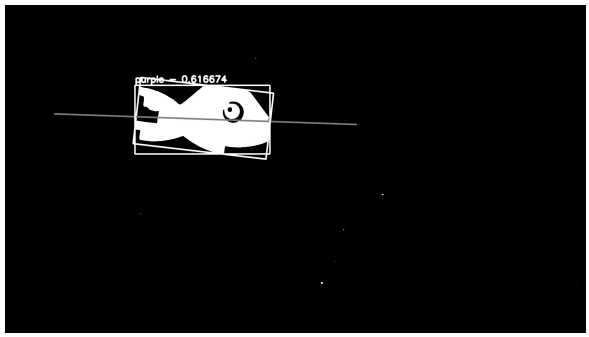
Labeled as: blue



Recognized as: blue



Labeled as: purple



Recognied as: purple

Reflection

Through this project, I learned how to use OpenCV functions to perform tasks such as thresholding, morphological filtering, connected components analysis, and moment calculations. I also learned how to use these tasks to create a real-time system that can recognize objects in a video sequence.

One of the most important takeaways from this project was the importance of pre-processing images before performing thresholding. For example, blurring an image can make the regions more uniform, and it can also help to remove noise. I also learned how to use morphological filtering to clean up thresholded images, which can be useful for removing holes or other issues.

Overall, this project was a great learning experience, and it gave me a deeper understanding of image processing and computer vision.

Acknowledgement

[1] *OpenCV modules*. OpenCV. (n.d.). Retrieved February 10, 2023, from https://docs.opencv.org/4.x/

[2] Kartikkukreja. (n.d.). *Blog-codes/union find (disjoint set) data Structure.cpp at master · Kartikkukreja/blog-codes*. GitHub. Retrieved February 24, 2023, from https://github.com/kartikkukreja/blog-codes/blob/master/src/Union%20Find%20(Disjoint%20Set)%20Data%20Structure.cpp