REAL-TIME 2D OBJECT RECOGNITION

DONE BY: ADVAITH KANDIRAJU
002743436

TASK-1: Threshold the input video

To perform this task, we first plot a 2D histogram of the image and then split the histogram into two regions. The division is achieved by finding the mean value between the two highest intensities, which correspond to the object of interest and the background. This is based on the assumption that the background will have the highest intensity and appear white, while the object of interest will have a lower intensity. To improve thresholding, the image is first filtered using a Gaussian filter.

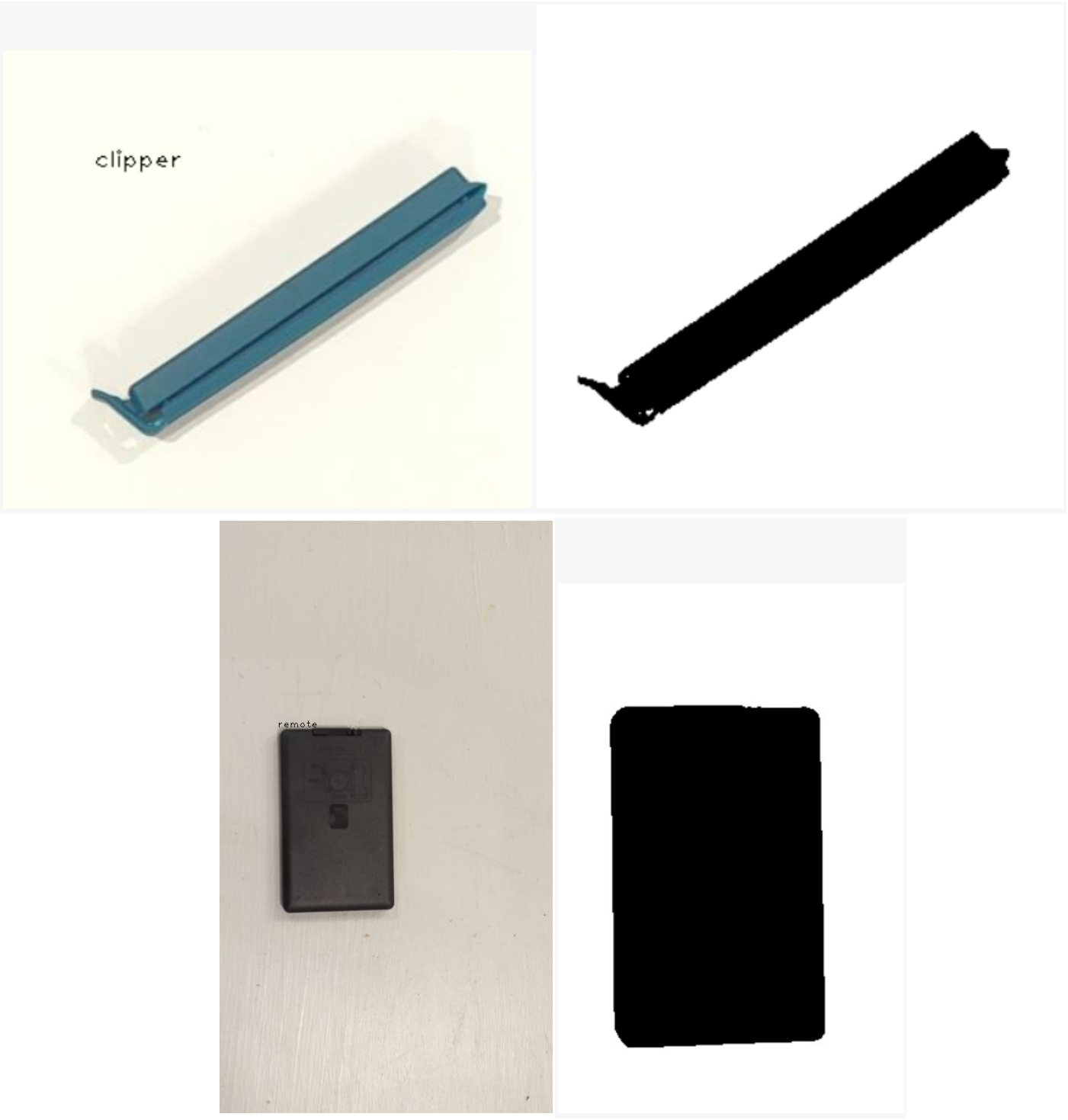

Figure-1: Thresholding of RGB to a binary image.

TASK-2: Clean up the binary image

To perform the image cleaning process, I utilized OpenCV's in-built function called "floodfill." This algorithm is capable of filling a connected region in an image with a user-defined color. The process starts with a seed point and expands in all directions until it reaches the region's boundary, which is determined by a user-defined condition such as the seed point and boundary condition. This algorithm is preferred due to its speed and efficiency in filling up holes in the image without performing dilation or erosion. It fills each pixel that lies within the object's boundaries. The resulting images, obtained by applying the flood-fill algorithm on the thresholded objects, are shown below.

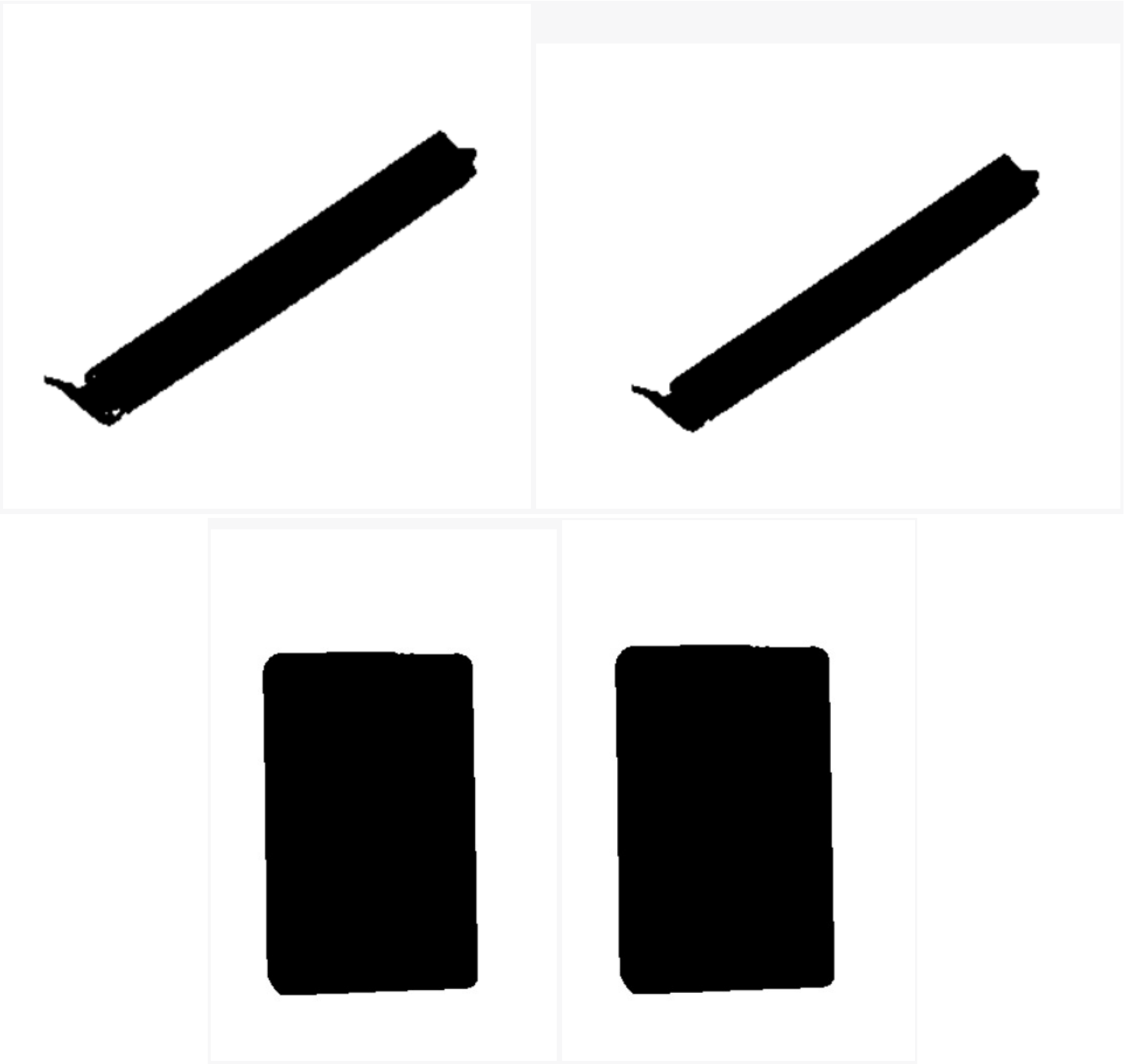

Figure-2: Cleaning-up process of the binary image

TASK-3: Segment the image into regions

We utilize the OpenCV function "connectedComponentsWithStats" to perform segmentation. This function performs 4 connected component analyses on the input image and outputs the labels for each component. Additionally, it provides statistics for each component, including its area, bounding box height, width, etc. The centroid output provides the pixel location of each component's centroid. The function returns the number of components identified in the image and has an integer return type. We demonstrate the implementation of this function on an image that is not well-denoised, with different segments coloured differently. The previous images used for this task were clean and had only one component. However, the current image is lighter and contains a blob on the top left, and random colours are generated for each component.

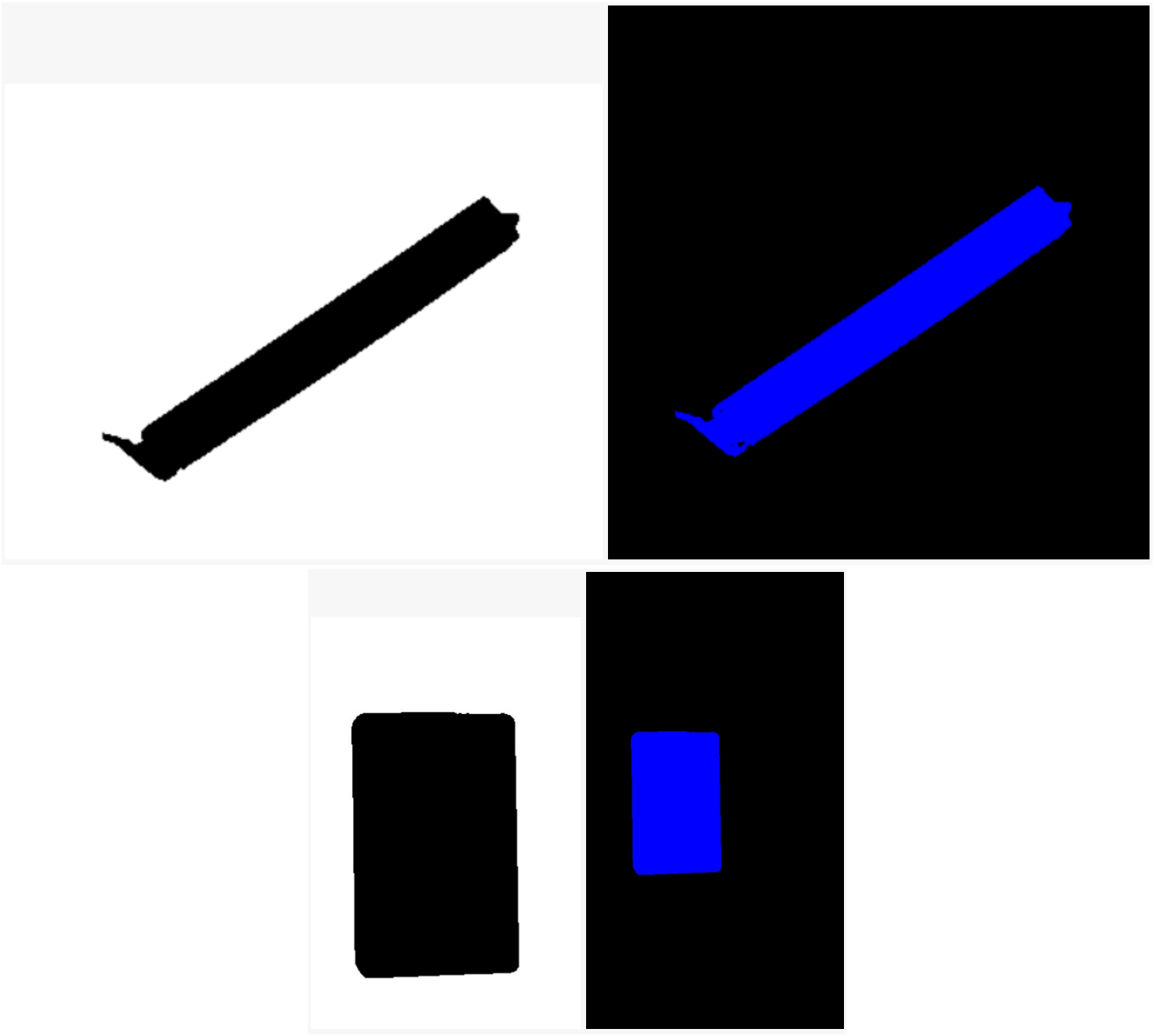

Figure-3: Highlighting various image components

TASK-4: Compute features for each major region

To extract features for the major regions, I considered several parameters, including the height, base of the bounding box, area of the region, and the percentage of the bounding box filled. For comparison purposes, I also used Hu-moments, which are a set of seven features that are scale, rotation, and translation invariant. These moments are derived from the central moments of an image, which are mathematical functions that describe the distribution of pixel intensities. The central moments are then normalized to eliminate any scaling and rotation effects, resulting in the computation of the seven Hu moments. These moments are calculated using a combination of mathematical functions such as the raw moments, central moments, and normalized central moments of an image. The resulting values provide a unique set of descriptors that can be used to differentiate between various shapes and objects present in an image.

Hu moments are a set of 7 scales, rotation, and translation invariant features that are widely used in image processing and computer vision. They are derived from the central moments of an image, which are mathematical functions that describe the distribution of pixel intensities in an image. The central moments are then normalized to eliminate any scale and rotation effects, resulting in a set of seven Hu moments.

The formula for the i-th Hu moment (where i=1,2,3,...,7) is given as:

```
hu_moments[i] = sign(mu[i]) * log10(abs(mu[i]))
```

where mu[i] is the i-th normalized central moment of an image, and sign(x) denotes the sign of x.

The formula for normalized central moments is given as:

```
mu[i,j] = (1 / (M00 ^ (1 + (i+j)/2))) * sum(x^i * y^j * I(x,y))
```

where I(x,y) is the intensity of the pixel at location (x,y) in the image, and M00 is the zeroth order moment of the image, which is defined as:

```
M00 = sum(I(x,y))
```

The values of the seven Hu moments provide a unique set of descriptors that can be used to distinguish between different shapes and objects in an image.

To obtain the oriented bounding box for our object, we calculate the vertices of a rotated rectangle with minimum area. This rectangle is aligned with the minimum moment axis of the object. The resulting images show the minimum moment axis and the corresponding oriented bounding box for each object.

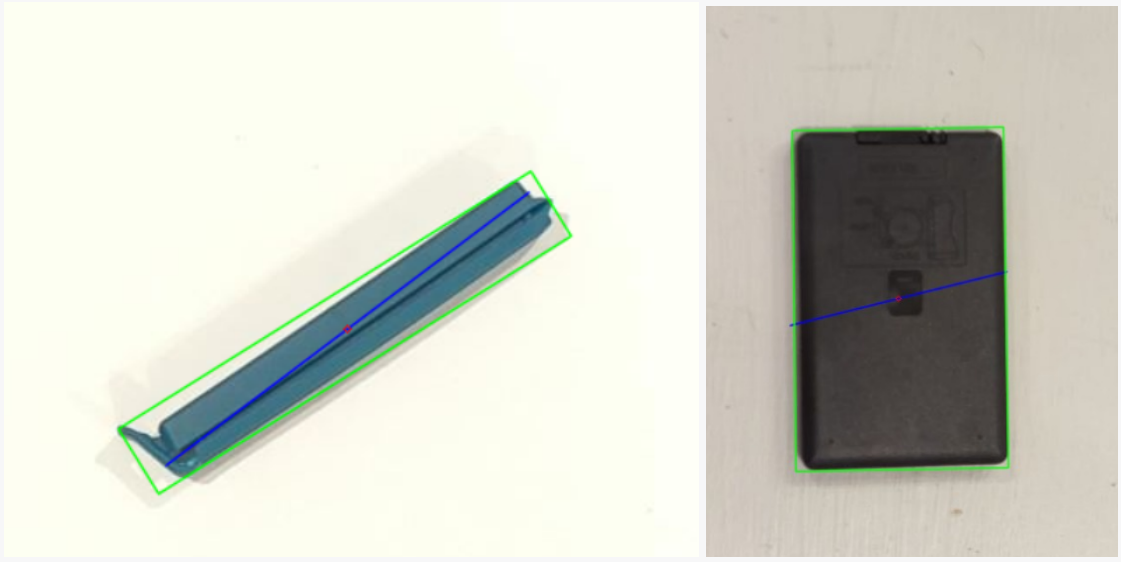

Figure-4: The axis with the least moment is highlighted in red. We can also see the oriented bounding box for our objects.

TASK-5: Collect training data

To create a dataset, I utilized images as they provided a more straightforward way to assess if the converted image was appropriately segmented. I extracted Hu-moments, which consist of seven invariant moments, as features. These moments are computed using the following equations:

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q I(x, y)$$

double huMoment1 *= invariantMoments[0] + invariantMoments[2];*

double huMoment2 *= pow(invariantMoments[0] - invariantMoments[2], 2) + 4 * pow(invariantMoments[1], 2);*

double huMoment3 *= pow(invariantMoments[0] - 3 * invariantMoments[2], 2) + pow(3 * invariantMoments[1] - invariantMoments[3], 2);*

double huMoment4 *= pow(invariantMoments[0] + invariantMoments[2], 2) + pow(invariantMoments[1] + invariantMoments[3], 2);*

double huMoment5 *= (invariantMoments[0] - 3 * invariantMoments[2]) * (invariantMoments[0] + invariantMoments[2]) * (pow(invariantMoments[0] + invariantMoments[2], 2) - 3 * pow(invariantMoments[1] + invariantMoments[3], 2)) + (3 * invariantMoments[1] - invariantMoments[3]) * (invariantMoments[1] + invariantMoments[3]) * (3 * pow(invariantMoments[0] + invariantMoments[2], 2) - pow(invariantMoments[1] + invariantMoments[3], 2));*

double huMoment6 *= (invariantMoments[0] - 3 * invariantMoments[2]) * (pow(invariantMoments[0] + invariantMoments[2], 2) - 3 * pow(invariantMoments[1] + invariantMoments[3], 2)) + (3 * invariantMoments[1] - invariantMoments[3]) * (invariantMoments[1] + invariantMoments[3]) * (3 * pow(invariantMoments[0] + invariantMoments[2], 2) - pow(invariantMoments[1] + invariantMoments[3], 2));*

double huMoment7 *= (invariantMoments[1] - invariantMoments[3]) * (pow(invariantMoments[0] + invariantMoments[2], 2) - pow(invariantMoments[1] + invariantMoments[3], 2)) + 4 * invariantMoments[0] * invariantMoments[2];*

The magnitude of these moments decreases with increasing index, with the seventh moment having a low value due to the multiplication of powers of numbers between (-1,1). Therefore, I converted the Hu-moments to log base 10 values for better comparison and understanding. Additionally, I calculated the percentage filled area as an extra parameter, but since it was calculated using the vertical bounding box, it was not rotation invariant and thus, not used for final comparison. To keep track of the dataset, I updated the standard deviation value for the features in a CSV file after adding a new label. For recognition, I utilized the least Euclidian distance between the features and the standard deviation (normalized) values.

TASK-6: Classify new images

The objects were classified using the method described above, and the resulting frames are presented below. It is important to note that the frames are different from the ones saved in the database, which highlights the scale, rotation, and translation invariant properties of the features used in the classification process.

Figure-5: Objects labelled and classified based on extracted features and classification algorithm.

TASK-7: Implement a different classifier

I implemented a K-nearest neighbours (KNN) classifier to classify objects based on the least distance from a set of similar labels. The algorithm calculates the Euclidean features for each label and normalizes it by the number of similar labels in the database. An example of the database for KNN filtering is shown below.

	A	B	C	D	E	F	G	H	I	J	K	L
1	case	1	58685	483	271	-2.97271	-6.18855	-10.2847	-11.165	0	0.448343	
2	case	1	61708	182	431	-3.0101	-6.26271	-10.9108	-11.9431	0	0.78667	
3	case	4	62690	430	184	-3.0149	-6.27927	-10.9244	-11.9497	0	0.792341	
4	case	1	58030	391	283	-3.01209	-6.2709	-10.9417	-11.9727	0	0.524432	
5	glove	1	163369	613	416	-3.10105	-6.75742	-10.5168	-11.8094	-22.9901	0.640643	
6	glove	2	159221	604	386	-3.10528	-6.7665	-10.5686	-11.8688	-23.1588	0.68293	
7	glove	2	161183	609	479	-3.10123	-6.75429	-10.5514	-11.8825	-23.1968	0.552543	
8	remote	1	25594	218	232	-3.13538	-6.95438	-13.7648	-15.1124	-30.3314	0.50605	
9	remote	1	53551	339	289	-3.13374	-6.94048	-12.7192	-14.0985	-29.2404	0.546601	
10	remote	1	35599	261	183	-3.1359	-6.96037	-14.6436	-15.6855	-31.3952	0.745326	
11	remote	1	27347	134	211	-3.13916	-6.98946	-13.6555	-14.7551	0	0.967214	
12												
13												

Figure-6: Classified objects indicated labels

TASK-8: Evaluate the performance of your system

	Image-1	Image-2		
case	Yes	No		
remote	Yes	Yes		
harness	Yes	No		Yes - Correctly Detected
cross	Yes	Yes		
clipper	Yes	Yes		No- Incorrectly Detected
powerbank	No	No		
scissors	Yes	Yes		
screw	Yes	Yes		
wallet	Yes	Yes		
gloves	No	No		
coin	No	No		

Accuracy = Approximately 75%

TASK-9: Capture a demo of your system working

https://drive.google.com/file/d/1G2pDq4NS4uH34BoZX7ekPknTUjxNUcIp/view?usp=share_link

Reflections: It was a great learning experience.

References: N/A