Project 3: Real-time 2D Object Recognition

by Yixiang Xie for CS5330 CVPR

Project Description

This is the third project for the class CS5300 Pattern Recognition & Computer Vision. The project is about real-time 2D object recognition, which is to enable the computer to use a camera looking straight down to capture live footage of a specified set of objects placed on a white surface, and identify the objects in a translation, scale and rotation invariant manner. The basic goal is to allow the computer to recognize single objects placed in the image. It is better if the computer can recognize multiple objects in the same image.

For easier development, the project can be set up using a set of static images. Once the system works, the goal is to allow real-time recognition, which can be achieved by setting up a downward facing camera capturing a white surface.

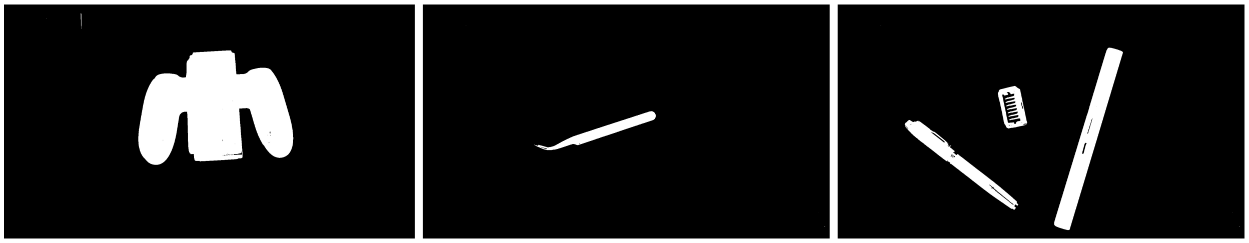
To recognize the objects, the project is carried through the following tasks: thresholding the input video to turn the frames into binary images, cleaning up the binary images by performing morphological filtering, segmenting the images into regions by performing connected components analysis, computing the features from moments for each region, collecting training data of the labels and features, classifying new images and evaluating the performance.

Required Images

1. Thresholding the input video

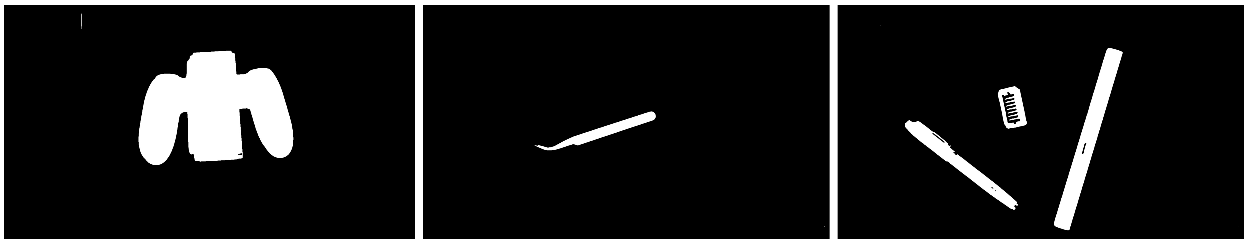
Before thresholding the input video, I need to pre-process the video footage. I first blur the frame with a 3x3 Gaussian blur filter to make the regions more uniform. Then I convert the color space of the image into HSV, and reduce the brightness of the pixel by 50%, if the saturation of the pixel is greater than 100. This is to allow the strongly colored pixels to move further away from the white background. The I convert the image back to BGR and then into grayscale. This turn the image into a 1-channel image. I want to turn the background into dark pixels and the foreground into bright pixels for connected component analysis, so I invert the image as well.
Then I develop my own fix thresholding code, which looks at each pixel and set the pixel value to 255 if the pixel value is greater than 155. Otherwise it is set to 0. This turns the background to black and the foreground to white, and the image is now a binary image.

Original video frames


Binary images

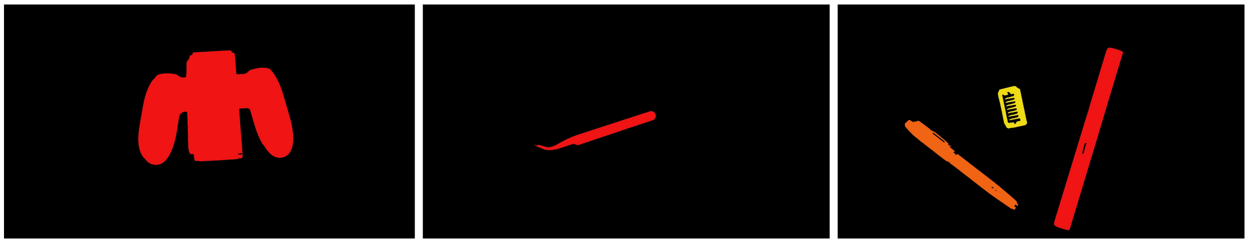
2. Cleaning up the binary images

On the binary images, we can see that there are some black spots on the foreground object, which is known as holes. This might be a result of the dust on the object surface, the intrinsic pattern on the object or the surface reflections created by the light, making the foreground too bright and thus treated as background. The morphological filter closing, which is growing followed by shrinking, is useful to close these small holes in the foreground object. I used the built-in morphologyEx with MORPH_CLOSE to achieve this. We can also see that there are some white spots on the background, which is known as noises. This might be a result of the dust on the background or the shadow created by the overlapping papers. Because these areas are small, we can deal with them later when segmenting the image.


Cleaned up images

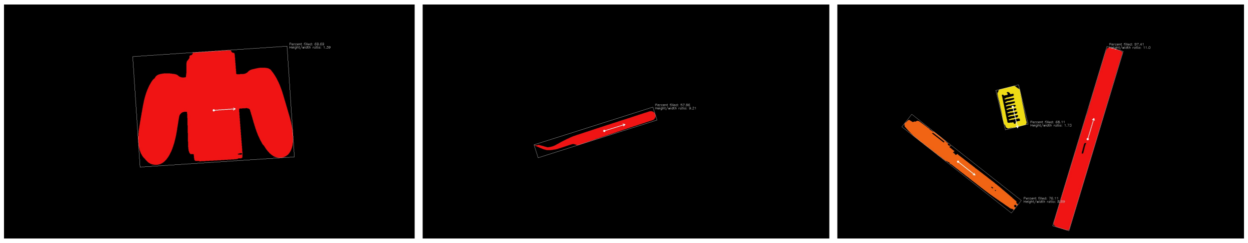
3. Segmenting the images into regions

After cleaning up the images, we need to run connected components analysis and segment the images into regions. In my implementation, I limit the recognition to the largest 7 regions (excluding the background) and ignore the regions that are smaller than 1000 pixels. I use the built-in connectedComponentsWithStats and 8-connectivity to achieve this.
To better display the regions returned, I sorted the regions by area size and color the 7 biggest regions in descending with 7 predefined colors (red, orange, yellow, green, blue, indigo and purple). This allows the colors to stay the same between frames.


Region maps

4. Computing features for each major region

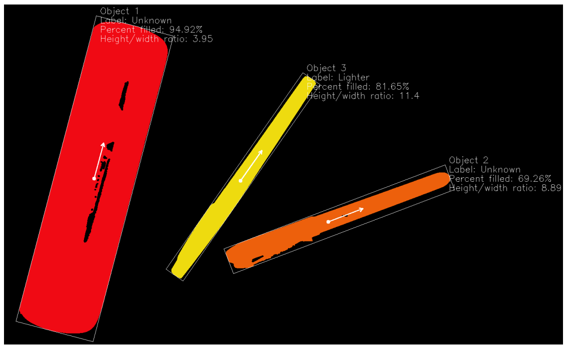
Similar to segmenting the images into regions, I only compute the features for the largest 7 foreground regions and ignore the regions that are smaller than 1000 pixels. For each region, I create a mask for it and compute its moments. I use the built-in moments function to achieve this. I used mu11, mu20 and mu02, which separately calculates the second order central moment, and the second order control moments along the x and y axes. With this, I can calculate the central axis angle, which allows me to draw the axis of least central moment. Then I calculate the corners and size of the oriented bounding box. This allows me the draw the oriented bounding box and to calculate other features, such as percent filled and height/width ratio. I also plot these features for display. It's been verified that percent filled and height/width ratio are translation, scale and rotation invariant.



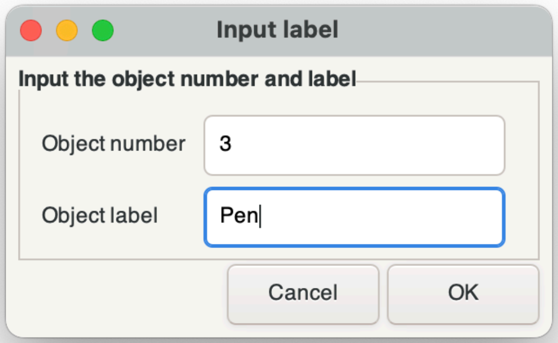
Region maps with axis or least central moment and oriented bounding box

5. Collecting training data

To enable the system to allow users to attach labels to objects and store the labels with their features to a database, I modified the system in the following ways.
First, in the region maps with features, I also attach object number and object label to each object. The object number starts from 1, and is in descending order according to the object area size. Brand new objects are labeled as "Unknown" (Pic 1). Sometimes, objects can be mislabeled (Pic 1).
When users want to assign a label to an object, either to a new object or to correct a wrong label, they can press "n" on the keyboard and the system will use Zenity to display a pop up window, prompting users to input the object number and the object label (Pic 2).
The system verifies the input, uses the number to find the object, and stores the label with the feature values (percent filled and height/width ratio) in a csv file. This file acts as the database of the system, and is read into memory at the starting of the system. The data is also added to memory so users can see the labels right away. With the same method, we can also collect new feature data for one object. This allows users to efficiently collect training data.



Pic 1, region map with numbers and labels


Input label

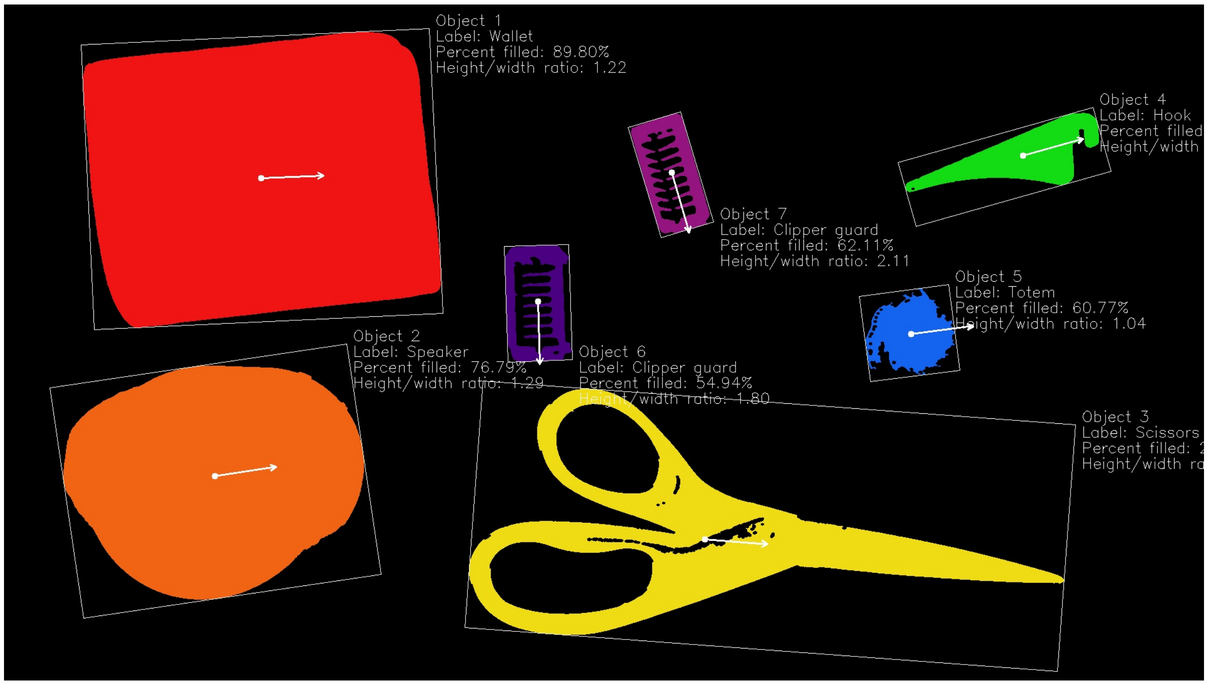
Input the object number and label

Object number	3
Object label	Pen

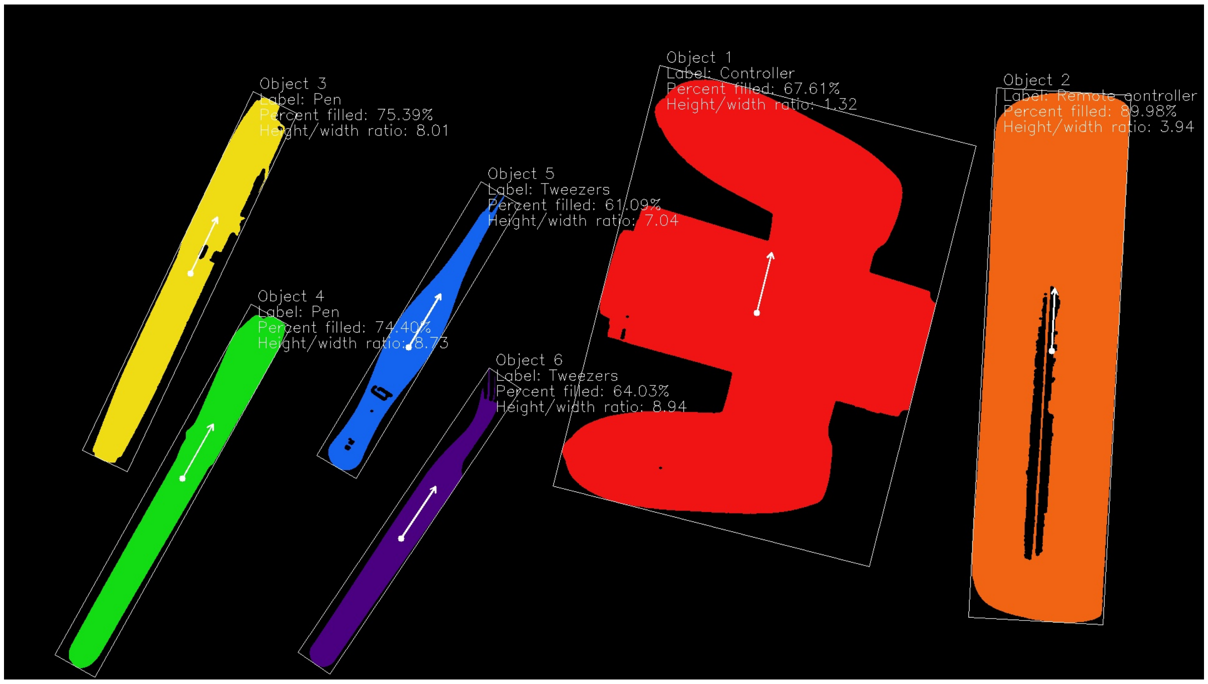
Pic 2, the prompt asking for number and label

6. Classify new images

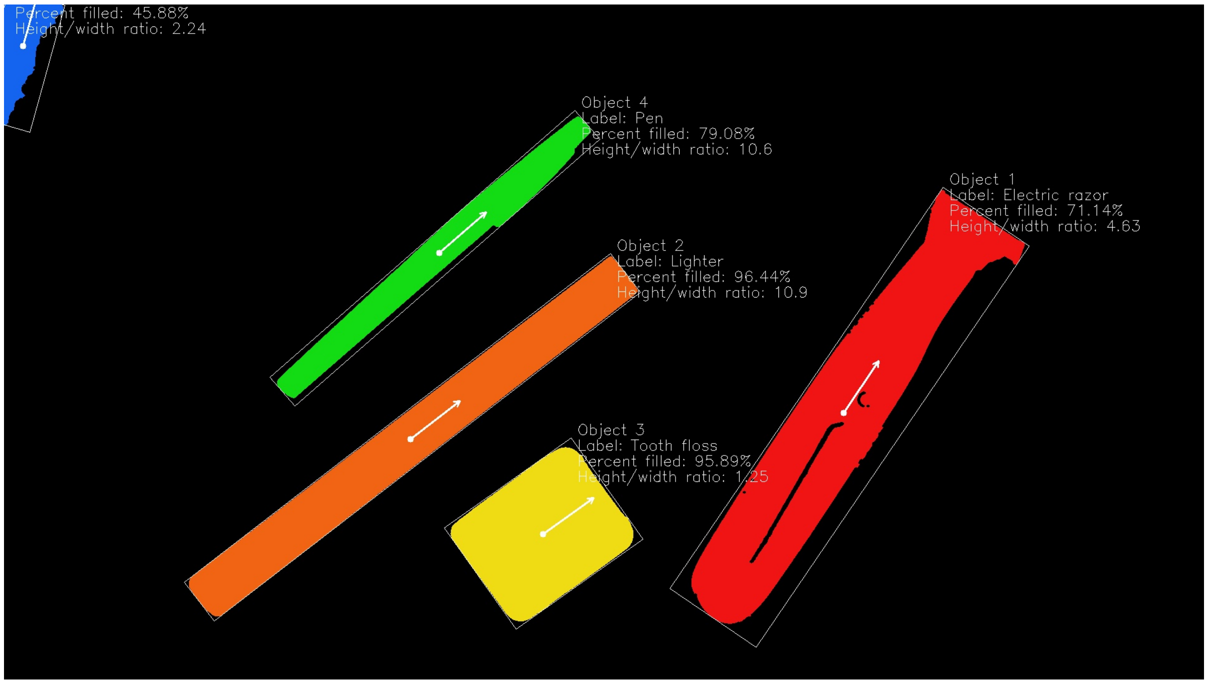
I implemented a nearest neighbor classifier and used scaled Euclidean distance as the distance metric. In total I use the system to classify 17 different objects in 13 different labels. The results are shown in the following images.


```
Object 1
Label: Wallet
Percent filled: 89.80%
Height/width ratio: 1.22

Object 4
Label: Hook
Percent filled
Height/width

Object 7
Label: Clipper guard
Percent filled: 62.11%
Height/width ratio: 2.11

Object 5
Label: Totem
Percent filled: 60.77%
Height/width ratio: 1.04

Object 2
Label: Speaker
Percent filled: 76.79%
Height/width ratio: 1.29

Object 6
Label: Clipper guard
Percent filled: 54.94%
Height/width ratio: 1.80

Object 3
Label: Scissors
Percent filled: 2
Height/width ra
```

Wallet, Speaker, Clipper guard, Scissors, Totem, Hook



Pen, Tweezers, Controller, Remote controller



Pen, Lighter, Tooth floss, Electric razor

7. Implement a different classifier

I also implemented a k-nearest neighbor classifier. I tested it with k=3. The result is not very different. In order to enable the k-nearest neighbor classifier to work, I need to let the database have more than 3 examples for each object.

8. Evaluating performance of the system

The system is good at classifying most of the objects. But it sometimes confuses the objects with similar shapes, such as wallet and tooth floss (square shape objects), pen, tweezers and lighter (thin long rectangles).

	Wallet	3											2	
	Pen		4						1					
	Speaker	1		4										
	Clipper guard				5									
	Scissors					5								
	Totem						5							
Actual label	Hook							5						
	Tweezers		2						3					
	Remote controller									5				
	Controller										5			
	Lighter		1									4		
	Tooth floss	1											4	
	Electric razor													5
		Wallet	Pen	Speaker	Clipper guard	Scissors	Totem	Hook	Tweezers	Remote controller	Controller	Lighter	Tooth floss	Electric razor
		Predicted label												

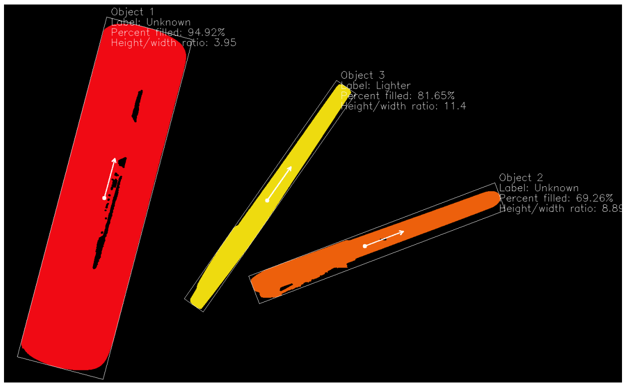
9. Video demo of the system

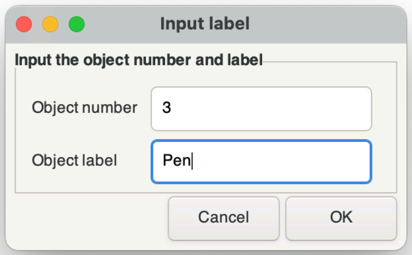
https://drive.google.com/file/d/11xr3kna2roguuYvSC6OMji48qHacTay3/view?usp=share_link

Extensions

1. GUI for setting labels and collecting training data

As shown in task 5, I developed a workflow, where users can see the objects with their object numbers and labels. The label might be “Unknown” if the object is brand new, or might be wrong. Either way, users can press “n” on the keyboard so the system will pop up a window and ask the user to input the object number and label. The system will store the data into the training data set. The same method can also be used to collect new feature values for the same object.





Input the object number and label

Object number: 3

Object label: Pen

Pic 1, region map with numbers and labels

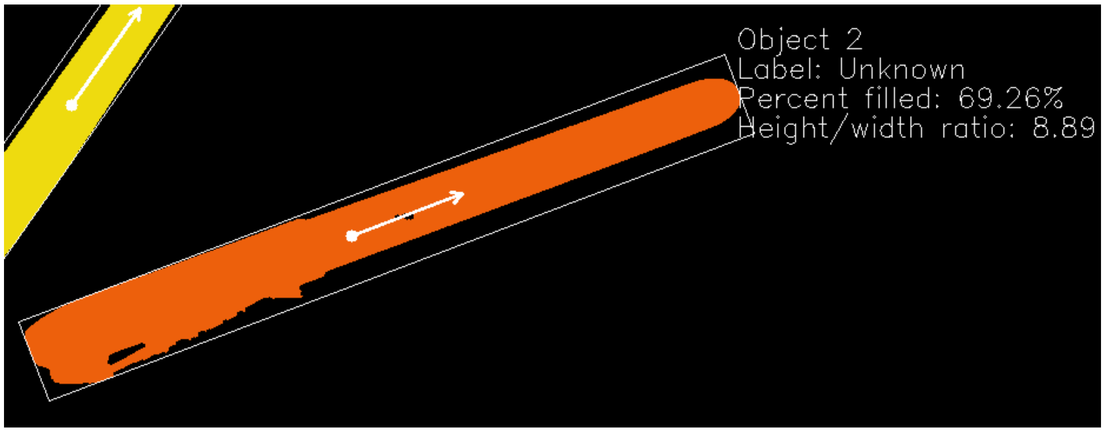
Pic 2, the prompt asking for number and label

2. 17 different objects with 13 different labels

As shown in task 6, I collected 17 different objects in total. These objects have 13 different labels in total. They are wallet, speaker, 2 clipper guards, scissors, totem, hook, 3 pens, 2 pairs of tweezers, controller, remote controller, lighter, tooth floss, electric razor.

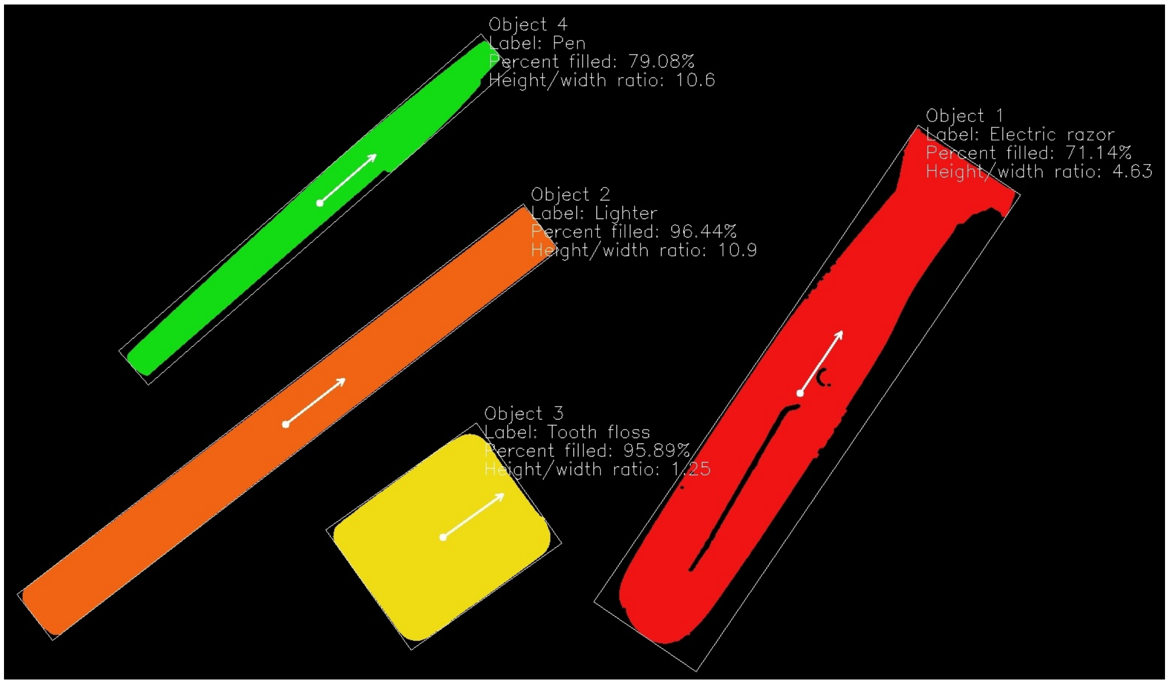
3. Showing unknown objects

As shown in task 5, the system can recognize unknown objects and display the label as “Unknown”. By specifying the object numbers and giving labels to them, the system can learn new objects. The demonstration is also shown in the video.



4. Classifying multiple objects in one image

As shown in task 6, the system can classify multiple objects simultaneously.



5. Adaptive thresholding with k-means clustering when k=2

For task 1, besides fix thresholding, I also developed an adaptive thresholding method by running k-means clustering with k = 2 on the first quarter of the shuffled pixels. This gives me the means of the two dominant colors in the image. I used the mean value of the two means as the threshold for fix thresholding. This method was not as ideal as it often classifies the shadow as the foreground so I didn't use it for the final result. But it is still programmed and can be activated with an optional parameter.

Reflections

In this project, I learned about how to perform real-time 2D object recognition. I practiced again on how to capture live video stream using a camera and display it. I learned about how to fix and adaptive threshold an image and turn it into a binary image; how to clean up an image if it has holes in the foreground or noises in the background, with different morphological filters; how to segment an image with connected components analysis to get the regional map of it; how to calculate moments and how to get axis of least central moment; how to use central moment angle to compute the oriented bounding box and then calculate the percent filled & height/width ratio as features; how to perform k-nearest neighbor classifying with multiple labels.

Acknowledgments

Professor Bruce Maxwell, for the development image set and the live coding.