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.

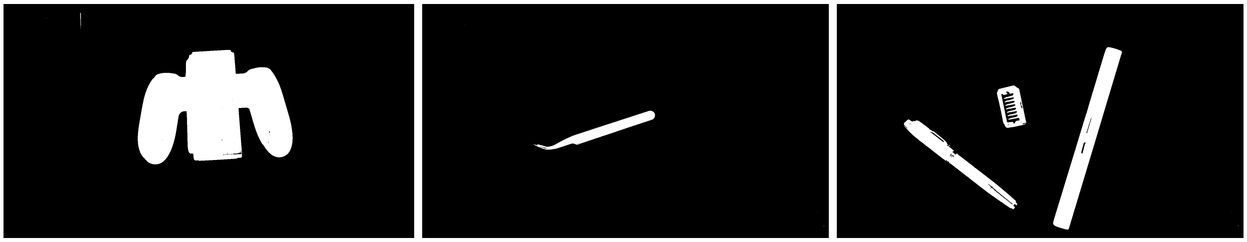
To recognize the objects, the project is carried through the following tasks: thresholding the input video to turn the frames into binary images, cleaning up the binary images by performing morphological filtering, segmenting the images into regions by performing connected components analysis, computing 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.

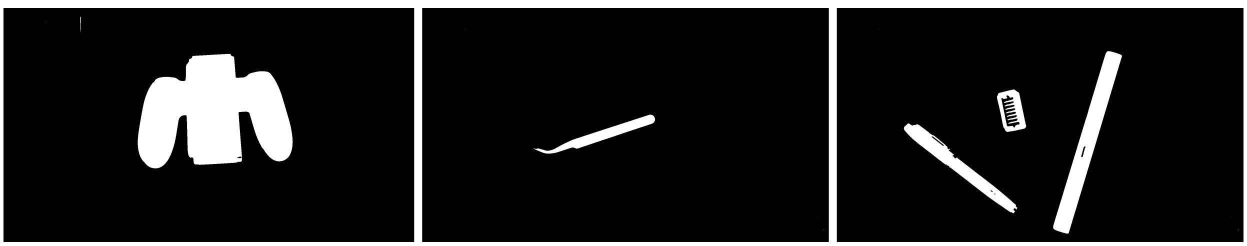
Original video frames



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

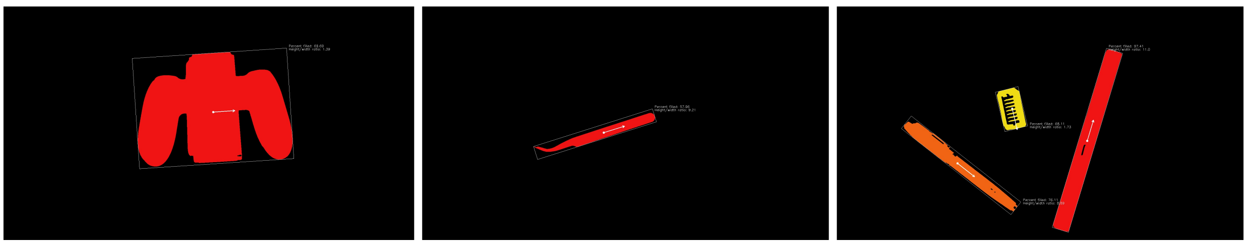
3. Segmenting the images into regions

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

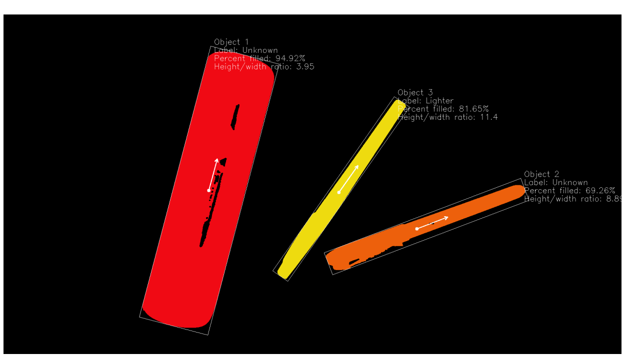
Similar to segmenting the images into regions, I only compute the features for the largest 7 foreground regions and ignore the regions that are smaller than 1000 pixels. For each region, I create a mask for it and compute its moments. I use the built-in moments function to achieve this. I used μ_{11}, μ_{20} and μ_{02}, 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.

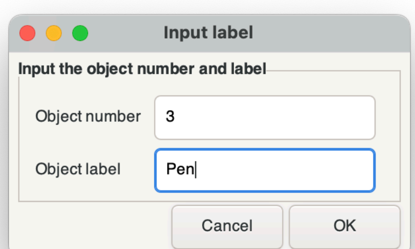


Region maps with axis or least central moment and oriented bounding box

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 to each object with an object number and an object label. The object number starts from 1, and is in descending order according to the object area size. Initially, the objects are labelled as "Unknown" (Pic 1). Then, when users want to give an object a label, or to assign a new label to an object, they press "n" on the keyboard and the system will use Zenity to display a GUI, prompting the user to input the object number and the object label (Pic 2). The system verify the input and store the label with the feature values (percent filled and height/width ratio) to a csv file, which acts as the database of the system. This file is read at the beginning of the program. The label is also updated to the memory so users can see the labels right after the changes are made. In this way we can efficiently collect the training data.



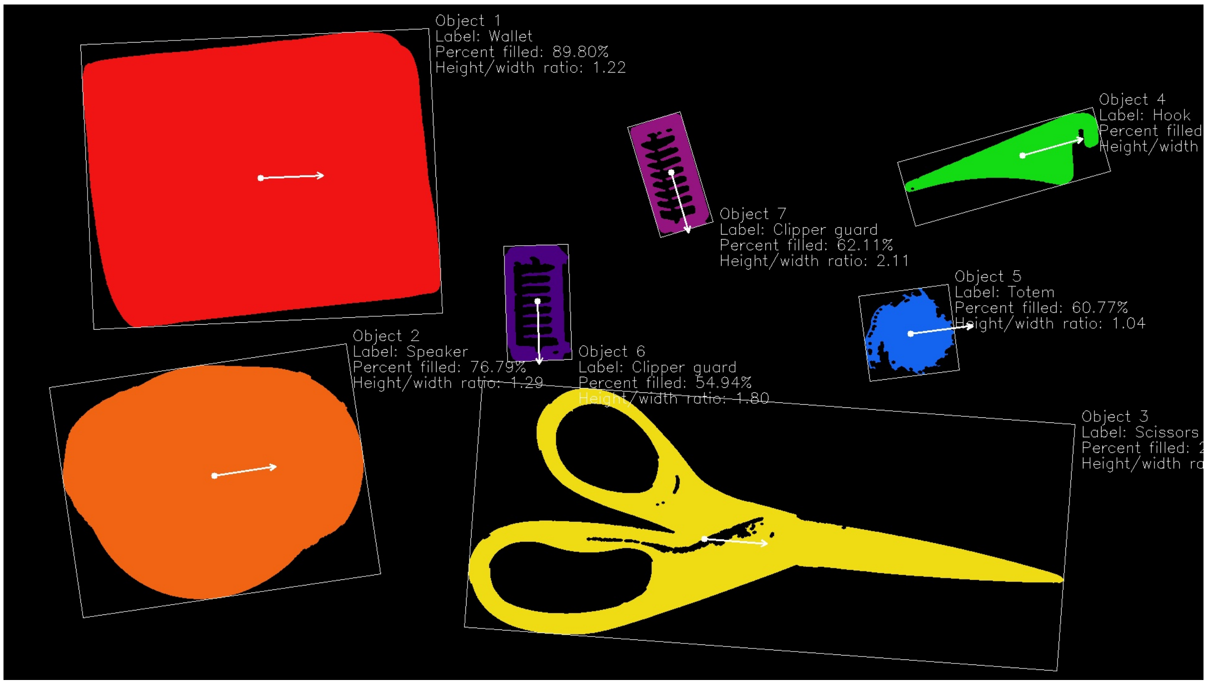


Pic 1, region map with numbers and labels

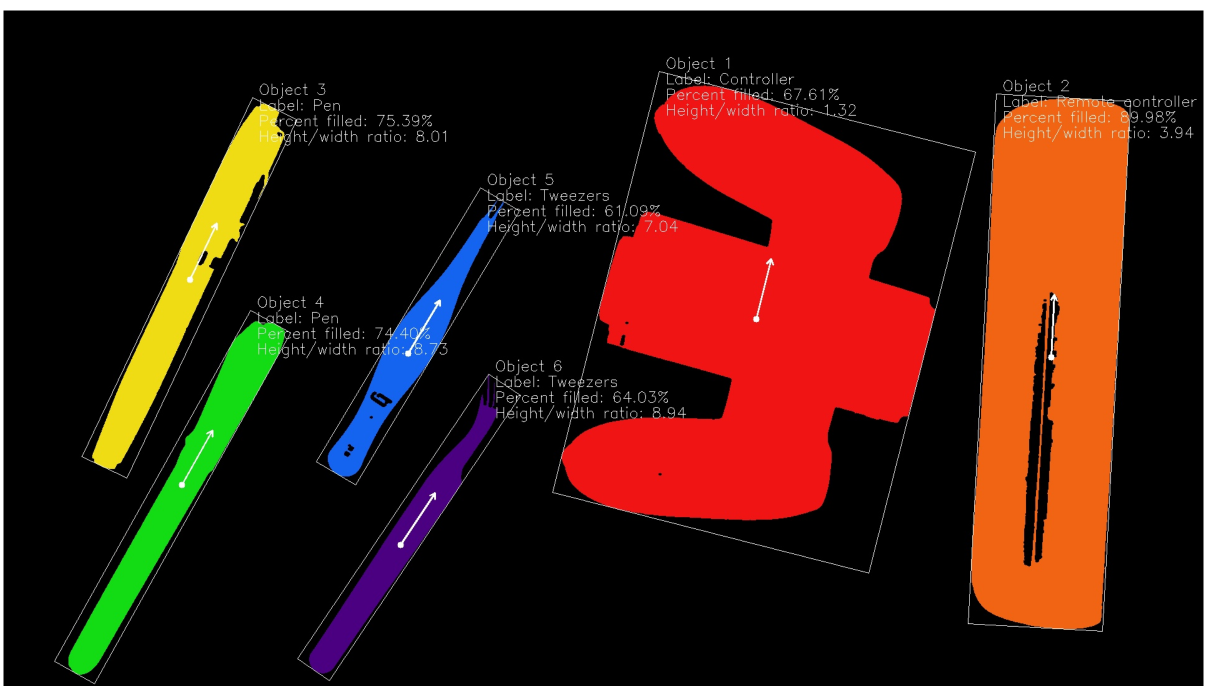
Pic 2, the prompt asking for number and label

6. Classify new images

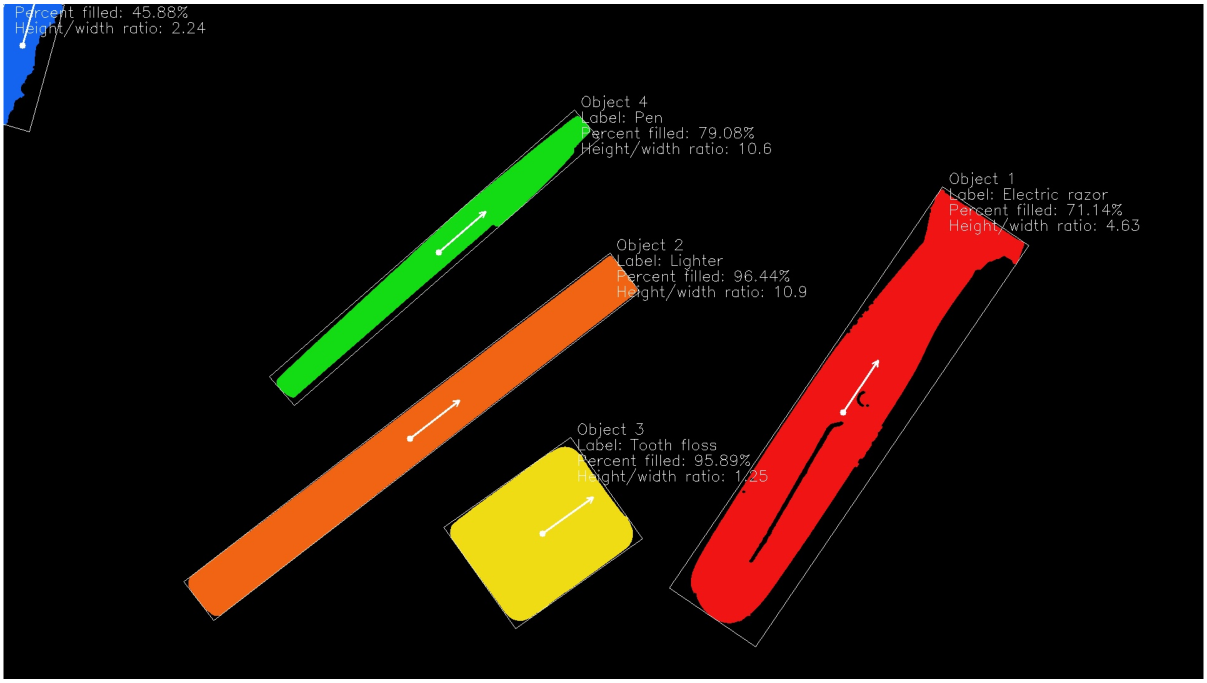
I implemented a nearest neighbor classifier and used scaled Euclidean distance as the distance metric. In total I use the system to classify 17 different objects in 13 categories. The results are as the following.



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. In order to enable the classifier to work, I need to let the database have more than 3 examples for each object.

8. Evaluate performance of the system

The system is mostly good at classifying the objects. But it also confuses some similar objects, such as wallet and tooth floss, pen and tweezers and lighter.

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

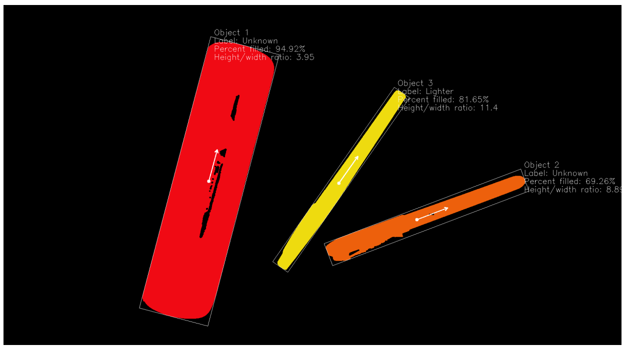
9. Video demo of the system

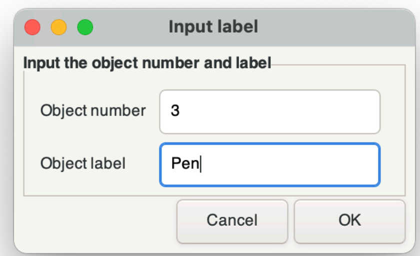
https://drive.google.com/file/d/11xr3kna2roguuYvSC6OMji48qHacTay3/view?usp=share_link

Extensions

1. GUI for managing DB and collecting data

As shown in task 5, I developed a GUI to allow user to input number and labels to collect training data set.





Pic 1, region map with numbers and labels   Pic 2, the prompt asking for number and label

2. 17 objects in 13 categories

As shown in task 6, I classified 17 objects in total. These objects belongs to 13 categories in total.

3. Showing unknown objects

As shown in task 5, the system can recognize unknown objects and learn new objects. The demo is also shown in task 5 and the video.

4. Classifying multiple objects in one image

As shown in task 6, the system can classify multiple object in one image.

5. K-means k=2 adaptive thresholding

For task 1, I also developed an adaptive thresholding with the use of k-means, k = 2 method. The method was not ideal so I didn't use it for the final product.

Reflections

In this project, I learned about how to perform 2D object recognition. I learned a lot about how to threshold an image to turn it into binary image; clean up images with different morphological filters; segment the images with connected components analysis; moments and oriented bounding boxes. They are very helpful for future projects.

Acknowledgments

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