

## CS5330: Project 3

# Real-time 2-D Object Recognition

### **Authors:**

**Name:** Ravi Shankar Sankara Narayanan  
**NUID:** 001568628

**Name:** Haritha Selvakumaran  
**NUID:** 002727950

### **Short description:**

This project is a real-time 2-D object recognition program. It captures video from a webcam and performs preprocessing operations like thresholding, morphological filtering and connect components analysis to identify distinct objects in each frame.

For each identified object, the program calculates certain features like bounding box ratio, axis aligned bounding box co-ordinates, oriented bounding box co-ordinates and percentage of bounding box filled. These features can be saved to a database (text file) along with user-defined labels.

Lastly the program utilizes this saved database file to recognize objects using Nearest Neighbor or K-Nearest Neighbors algorithms. Additionally, there is also an evaluation mode that will calculate a confusion matrix for pre-defined set of labels(classes).

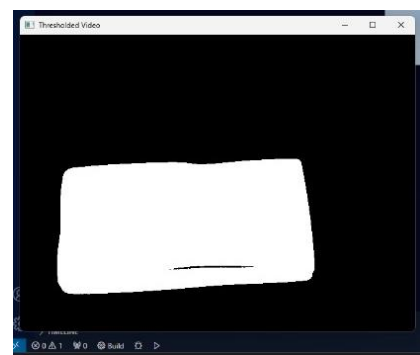
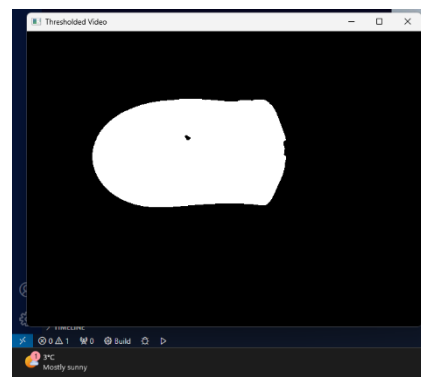
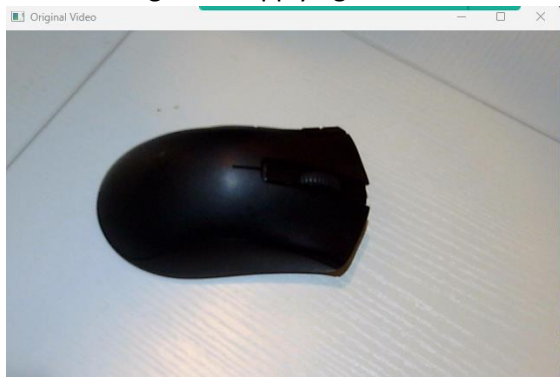
The program also displays multiple windows for displaying the original feed from camera, video after applying thresholding, video after applying morphological filter and video after applying connected component analysis and applying bounding boxes (both axis-aligned bounding box and oriented bounding box)

### **Tasks and Outputs**

#### **Task 1 Threshold the input video:**

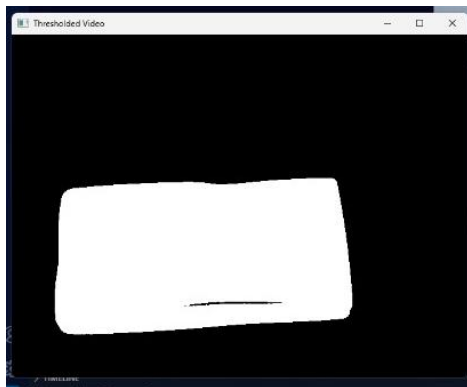
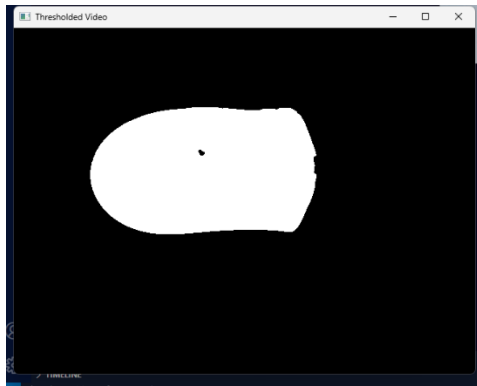
The thresholding function in my program takes an input frame from the webcam and applies a Gaussian blur to reduce noise. Then it applied a binary threshold to separate the objects of interest from the background. The threshold value is determined automatically using 'Otsu's' method. This is a simple yet effective thresholding method. Adaptive thresholding was tested but it computationally expensive and the frame rate of the live video output was so low.

The thresholding operation was performed on multiple objects and the results with the original image and the image after applying the threshold filter is attached below:



### Task 2 Clean up the binary image:

The function in my program will first perform an opening operation which is useful for removing noises. Then it performs a closing operation which will close all small holes in the object. The results of applying the morphological filter on the image after applying the thresholding filter is attached below:

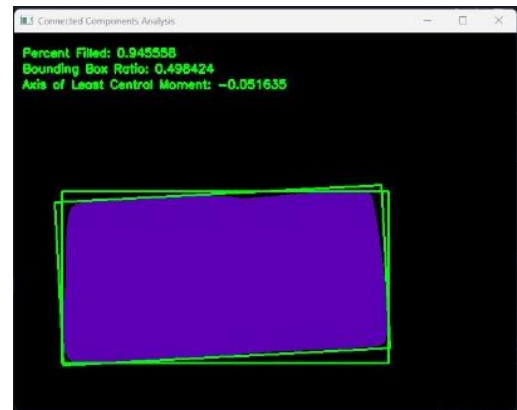
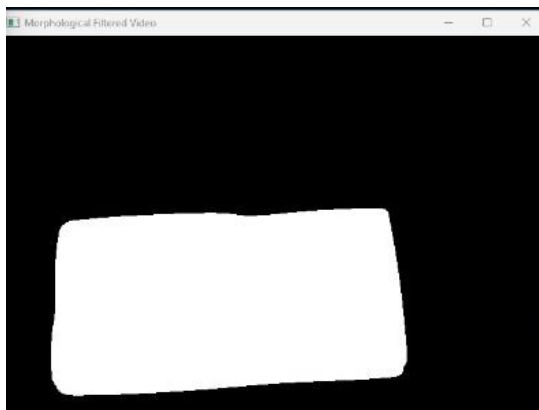
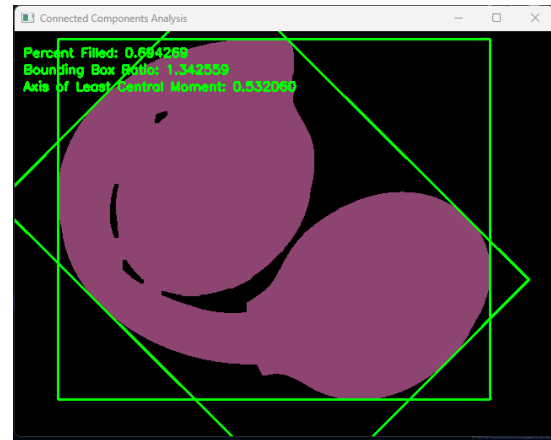
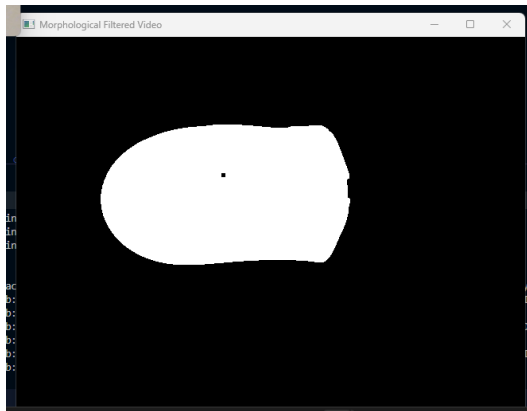


### Justification:

The morphological operations in my code include opening and closing operations. Opening, which is erosion followed by dilation, is used to remove small noise like reflections or shadows not part of the actual objects. Closing, which is dilation followed by erosion, is used to fill small holes within the objects that were not correctly thresholded. The structuring element, a 5x5 rectangle, defines the neighborhood for these operations. It's large enough to remove noise and fill holes without altering the objects' size or shape significantly. The choice of operations and parameters are well suited for this image characteristics and application requirements.

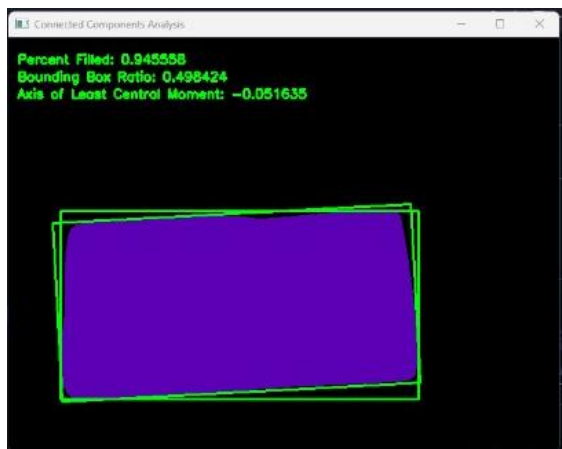
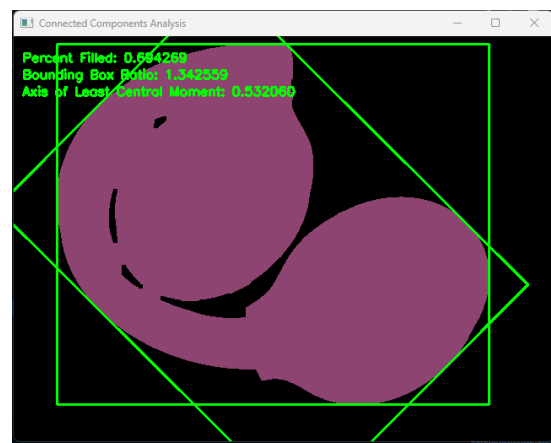
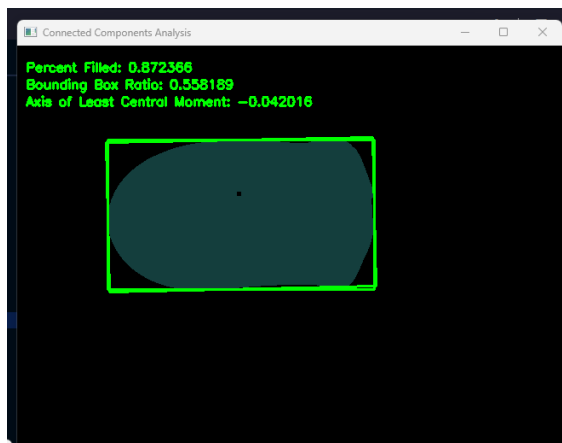
### Task 3 Segment the image into regions:

My program applies the connected components analysis on the input image, which labels each connected component (or region) in the image with a unique identifier. The function then creates an output image where each region is colored differently, effectively segmenting the image into distinct regions. The result after applying the connected component analysis to the input after applying morphological filter is attached below:



#### Task 4 Compute features for each major region:

The program calculates various features for a specific region in an image. These features include the axis-aligned bounding box, the oriented bounding box, the percent of the bounding box that is filled, the ratio of the bounding box's width to its height, and the axis of least central moment. These features will be used for object recognition. The program calculates both: axis aligned bounding box and oriented bounding box. The output images after extracting the feature vector over the region is attached below:

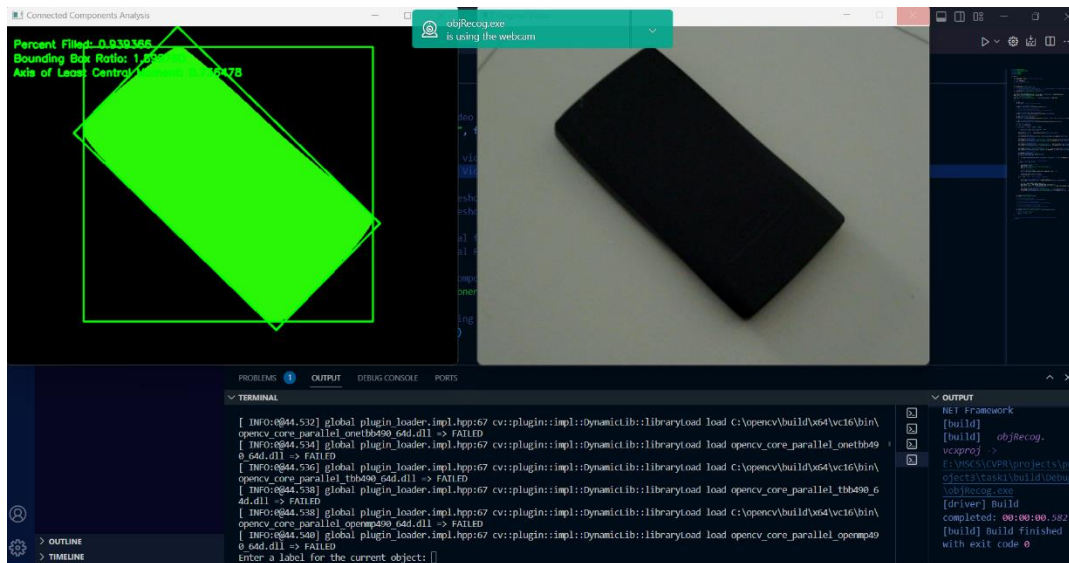


The Axis-Aligned Bounding Box (AABB) is the smallest rectangle, with sides parallel to the axes, that can contain the object. The Oriented Bounding Box (OBB) is the smallest rectangle that can contain the object, but unlike the AABB, it can rotate to best fit the object.

While the OBB can provide a better fit for non-axis-aligned objects, it's more computationally expensive due to rotation. And OBB is useful for computing rotation-invariant features, like the orientation of the object.

#### Task 5 Collect training data:

After calculating the feature vector, my program will save the all these features along with a user-defined label to a text file when the user presses the 'n' key. The terminal asks for label name from the user and then saves it to the text file. The output when a user presses the 'n' key is attached below:



The user enters the label and presses enter to save the feature vector for the calculator. The file containing the feature vector look like this:

```
calculator 0.93926 1.9017 0.736396 311.789 215.242 401.135 210.935 44.1048
calculator 0.953839 1.99242 0.0425905 261.187 233.58 412.437 207.004 2.72631
headphones 0.642959 0.731603 -0.323992 342.096 246.487 394.981 539.884 84.8696
smartwatch 0.563821 1.08369 0.623733 292.832 241.378 175.178 161.65 2.2026
smartwatch 0.792119 1.39137 0.685561 283.168 227.824 191.179 137.404 31.7014
bull_statue 0.59028 1.62133 0.791042 298.66 240.38 209.8 129.4 53.1301
bull_statue 0.696716 1.83153 0.218117 287.026 178.94 199.848 109.116 23.1986
optical_mouse 0.844267 1.65981 0.699835 255.718 209.767 312.789 188.449 43.1524
optical_mouse 0.861521 1.78022 0.424691 277.421 189.988 328.38 184.461 22.0295
pen 0.756134 7.14135 0.282362 325.735 219.076 361.581 50.6321 16.8735
pen 0.621983 0.137832 -0.8701 334.252 190.782 46.241 335.488 42.1579
pen 0.679834 7.92258 0.605309 301.086 327.011 347.281 43.8344 32.451
fork 0.386615 6.50483 0.666236 222.334 318.845 489.702 75.2828 34.3376
fork 0.567823 6.81873 0.51752 267.442 307.748 490.33 71.9093 27.2236
bottlecap 0.808413 1.13462 1.07548 318.538 239.997 167.428 147.564 85.6013
bottlecap 0.806227 1.04151 0.867219 319.586 204.664 153.738 147.61 62.1985
wallet 0.924678 1.24094 0.711947 263.545 254.527 294.472 237.297 49.0377
wallet_opened 0.891978 2.44977 0.539728 335.297 215.38 574.852 234.656 31.1688
specs_case 0.896096 2.32631 0.883984 311.239 252.644 436.452 187.616 51.6325
```

Each line refers to an object with its label and its features. Multiple instances of the same object represent the different orientation their feature vectors were saved in. The features calculated and saved are:

- **Label:** This is the name of the object as entered by the user
- **Percent Filled:** This is the percentage of the bounding box that is filled with the object.
- **Bounding Box Ratio:** This is the ratio of the width to the height of the bounding box.
- **Theta (Axis of Least Central Moment):** This is the angle of the axis of least central moment of the object.

- **Oriented Bounding Box Center X:** This is the x-coordinate of the center of the oriented bounding box.
- **Oriented Bounding Box Center Y:** This is the y-coordinate of the center of the oriented bounding box.
- **Oriented Bounding Box Width:** This is the width of the oriented bounding box.
- **Oriented Bounding Box Height:** This is the height of the oriented bounding box.
- **Oriented Bounding Box Angle:** This is the angle of the oriented bounding box.

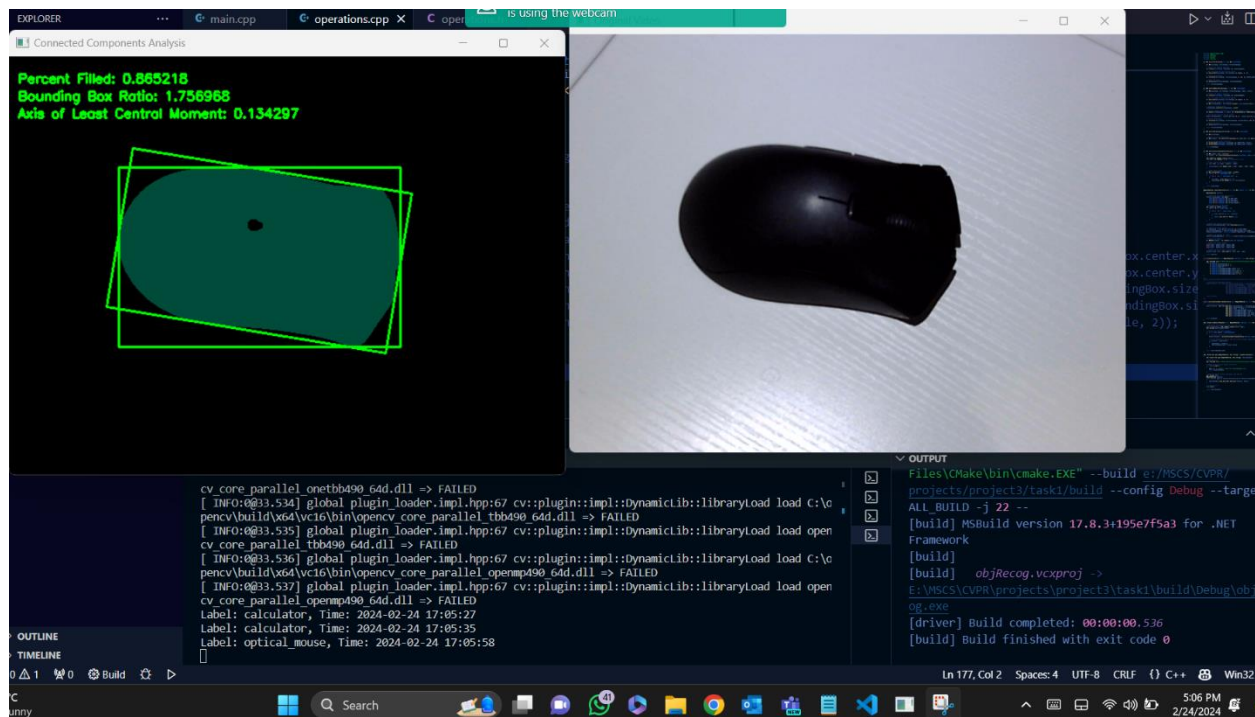
### Task 6 Classify new images:

The program first reads the file containing data about known objects, including their labels and various features. It stores this data in a vector of pairs, where each pair consists of an object type that has features and a string label.

Secondly, the program calculates the features of the current object in the camera and the database of known objects and finds the object in the database that is most like the input object. It does this by calculating the Euclidean distance between the input object's features and each known object's features and returning the label of the known object with the smallest distance.

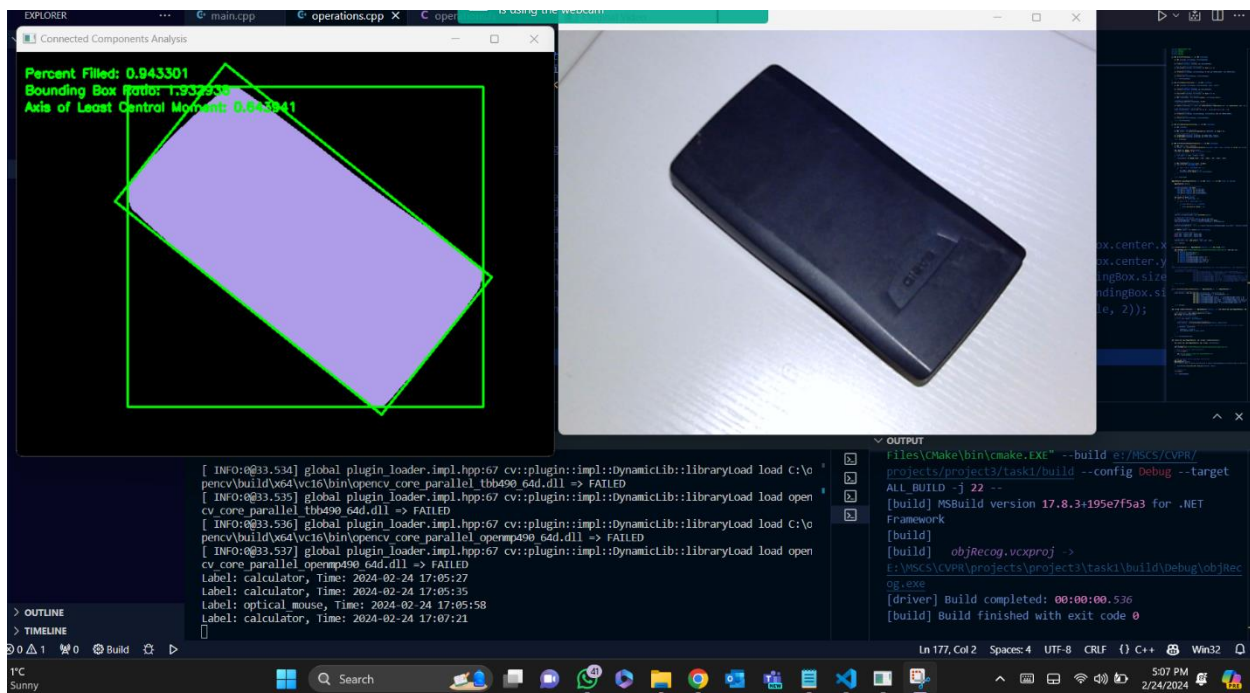
When the user presses key 'r' the program enters recognition mode and prints the label of the object in the database that is most similar to the object in the camera frame. The label on the output camera feed displays only for a few seconds so we have printed the label on the terminal

The output of this function is attached below:

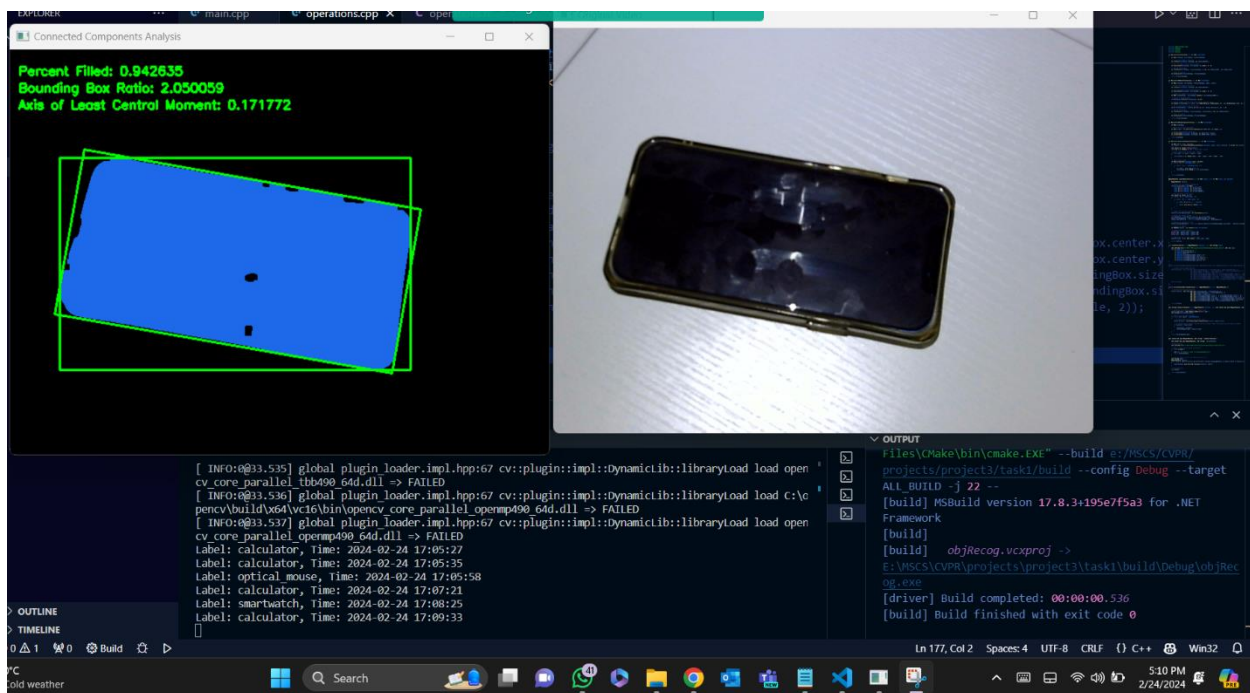


As we can see from the terminal output, the input image is correctly recognized as optical\_mouse. A few other objects were also tested with the program.





And the program returns the most similar object if it couldn't find the correct label of the input object.



In the above result, the program encounters the mobile phone object for the first time and returns the output as calculator which reasonably is like a mobile phone.



## Task 7 Evaluate the performance of your system:

For evaluating the performance of the program and to create a 5x5 confusion matrix five objects were taken from the full set of objects. These objects were later assigned some indices.

```
// Map the labels to indices
std::map<std::string, int> labelToIndex;
labelToIndex["bull_statue"] = 0;
labelToIndex["pen"] = 1;
labelToIndex["smartwatch"] = 2;
labelToIndex["headphone"] = 3;
labelToIndex["scissors"] = 4;
```

Their feature vector is attached below:

```
File Edit View

bull_statue 0.573916 1.54506 0.240917 299.906 183.508 212.669 137.644 21.5014
bull_statue 0.668431 1.7899 1.05073 260.945 186.662 208.111 116.27 49.7636
bull_statue 0.533503 1.40545 0.452623 274.936 159.437 208.904 148.639 19.6121
bull_statue 0.678941 1.91945 0.00460937 311.304 157.593 207.031 107.86 7.49586
bull_statue 0.528289 0.502036 -1.44652 282.351 186.366 118.574 236.186 6.67447
bull_statue 0.534951 0.535487 -1.31605 325.443 215.304 122.004 227.837 26.811
pen 0.682142 8.18409 0.173345 255.496 242.577 338.546 41.3664 8.01709
pen 0.609686 9.53058 1.44758 250.003 257.992 329.229 34.5445 81.9331
pen 0.688554 0.124922 -0.216955 243.387 186.378 41.4545 331.843 79.3543
pen 0.725682 7.84027 0.141758 214.861 195.402 358.247 45.6932 8.74616
pen 0.730419 8.28603 0.970625 266.999 160.744 349.668 42.1997 54.6434
pen 0.79617 8.53648 0.353153 246.582 202.135 375.634 44.0033 20.2249
smartwatch 0.755181 1.46561 0.946016 257 219.5 195.869 133.643 45
smartwatch 0.628471 1.11573 1.05171 252.946 185.816 170.298 152.634 84.0938
smartwatch 0.792808 1.86193 1.36689 246.212 171.558 185.134 99.4309 78.6901
smartwatch 0.550583 1.32384 1.26724 258.177 158.706 180.447 136.305 75.9638
smartwatch 0.787226 1.43708 0.87348 265.396 161.361 190.598 132.628 47.663
headphone 0.6815 0.771283 -0.376095 332.55 203.353 409.006 530.41 53.7853
headphone 0.698781 0.720669 -0.462415 308.41 224.089 384.524 533.565 73.9534
knife 0.806668 6.45551 0.315455 335.569 230.538 541.341 83.8573 19.2171
knife 0.79537 6.44452 0.335576 305.175 266.602 544.388 84.4731 18.178
scissors 0.348412 2.89849 0.578041 293.306 287.436 546.229 188.453 44.5474
scissors 0.381212 0.395403 -0.546007 301.637 276.165 205.929 520.809 72.6619
scissors 0.365236 0.430964 -0.256986 300.007 266.767 230.162 534.062 88.4015
scissors 0.319228 0.487881 -1.18539 251.266 238.549 235.9 483.52 33.2875
specs_case 0.903871 2.16541 0.375073 276.154 218.611 437.534 202.056 20.2826
specs_case 0.88088 0.474087 -0.668969 323.345 172.88 203.107 428.416 53.7462
specs_case 0.900161 0.48124 -0.3824 332.233 282.913 214.634 446.003 68.0661
specs_case 0.896273 2.33626 0.614072 283.176 182.054 433.72 185.647 35.5377s
```

To create the confusion matrix, we created some hooks on the program to check if the result of the program is correct or wrong. If it's correct the user presses 'y' and if the answer of the program is incorrect, the user presses 'n' and enters the right label for the input object.

```
✓ TERMINAL
Predicted label: pen. Is this correct? (y/n): y
Predicted label: smartwatch. Is this correct? (y/n): y
Predicted label: bull_statue. Is this correct? (y/n): n
Enter the correct label: smartwatch
Predicted label: smartwatch. Is this correct? (y/n): y
Predicted label: headphone. Is this correct? (y/n): y
Predicted label: headphone. Is this correct? (y/n): y
Predicted label: headphone. Is this correct? (y/n): y
Predicted label: scissors. Is this correct? (y/n): y
Predicted label: scissors. Is this correct? (y/n): y
Predicted label: smartwatch. Is this correct? (y/n): n
Enter the correct label: bull_statue
Predicted label: bull_statue. Is this correct? (y/n): y
```

The 5x5 confusion matrix is attached below:

```
Confusion Matrix:
1 0 3 0 0
0 4 0 0 0
1 0 2 0 0
0 0 0 3 0
0 0 0 0 2
```

A more detailed confusion matrix is attached below:

	bull_statue	pen	smartwatch	headphone	scissors
bull_statue	1	0	3	0	0
pen	0	4	0	0	
smartwatch	1	0	2	0	0
headphone	0	0	0	3	0
scissors	0	0	0	0	2

With the limited amount of data and from the above confusion matrix we can observe that the model works good at recognizing most objects correctly but mixes a small statue of a bull with smartwatch and vice versa.

Note: For generating the confusion matrix and for evaluation of the code using 5 objects, use the database file: object\_db\_task6.txt. For the rest of the recognition use: object\_db\_task9.txt

### Task 8 Capture a demo of your system working:

Link: <https://drive.google.com/file/d/1SzISVRnYJVLVcfKGcVSm7WtYe3TQBjaR/view?usp=sharing>

### Task 9 Implement a second classification method:

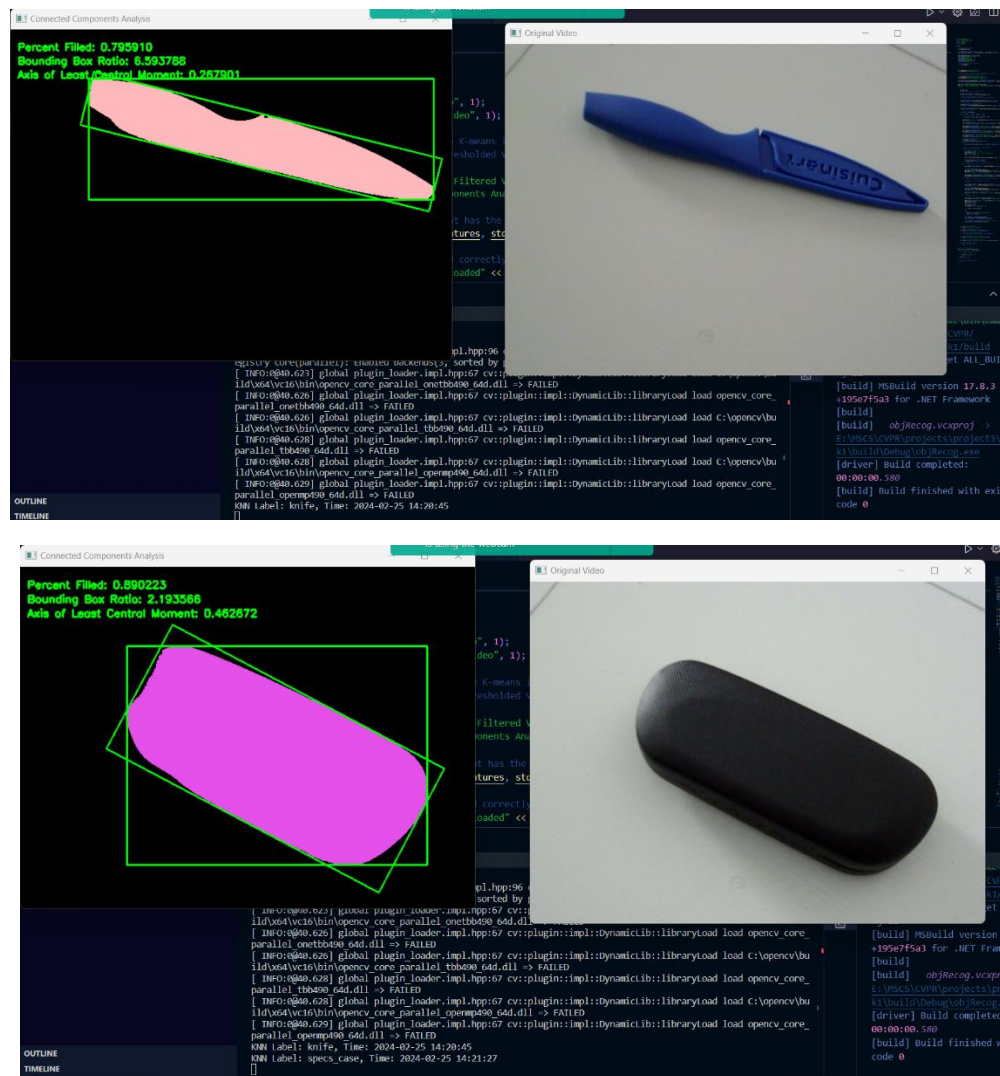
The program uses K- Nearest Neighbors function as a second classification method. It takes the following input the features of input object, a vector of pairs from the feature database (each containing a feature of the object in the database and their label), and an integer K. The function is designed to find the K nearest neighbors to the input object in the object database. It uses a priority queue to store the K nearest neighbors, ordered by distance. The function then iterates over the object database, calculating

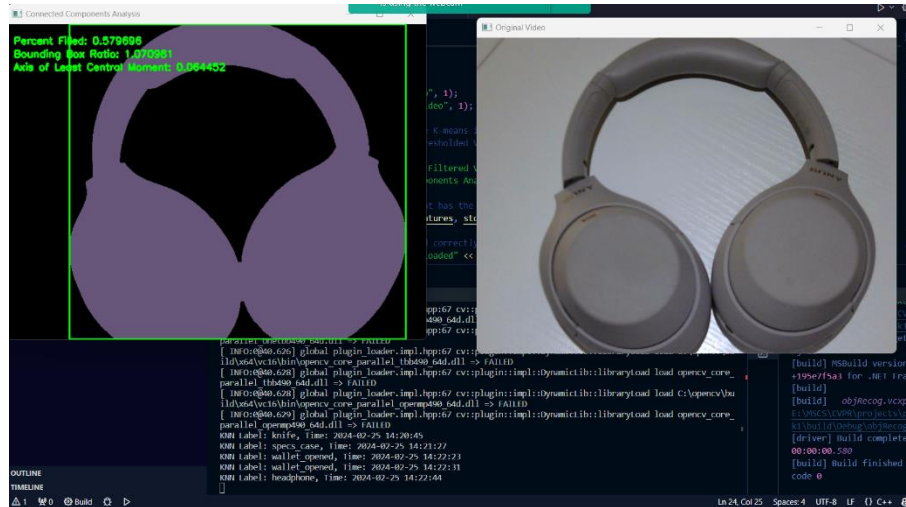
the distance to the input features for each object. If the current object is closer than the farthest neighbor in the queue, the farthest neighbor is replaced. After finding the K nearest neighbors, it calculates a weighted average of the labels, where the weight of each label is the inverse of the distance to the corresponding neighbor. The label with the highest average weight is then returned as the most likely label for the input features.

This implementation does not use a voting method, but instead uses a weighted average approach.

When a user presses the 'k' key the program executes the object recognition but by using the KNN method and prints the prediction in the terminal.

The output of using this function to identify objects is attached below:





In the above three images, if we look at the terminal, the predicted label for the input images is printed. And in our third result we can observe that the program fails to recognize the headphones the first couple of times but it predicts the label correctly the third time.

### Comparison with baseline program:

The baseline program which is nearest neighbor function used Euclidean distance to compare the feature vector and returns the label of the object that is closest to the input object. Whereas in KNN, the program finds the 'K' closest objects in the object database to the input features. It then calculates the weighted average of the labels of these objects, where the weight of each label is the inverse of the distance to the corresponding object. It returns the label with the highest average weight.

Based on the small training data, we have found that KNN is more robust, and it easily recognizes a wide range of objects that have complex shape. Though KNN is computationally expensive, it can detect objects very well when compared to baseline program. Moving forward with a larger dataset, the difference between KNN and baseline nearest neighbor function will only get big with KNN being more precise at recognizing objects well.

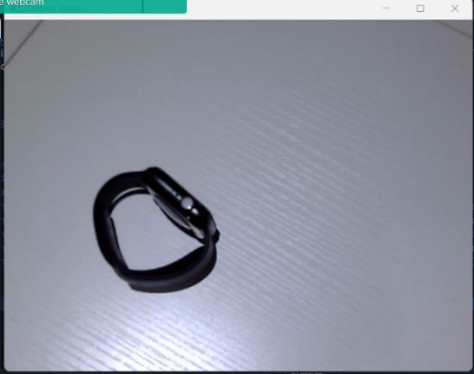
### Extensions:

#### 1. Multi-object recognition:

Our program in total can recognize up to 13 objects. The objects are calculator, headphones, smartwatch, bull statue, optical mouse, pen, fork, bottlecap, wallet. Wallet opened, specs case, knife, and scissors. Few sample output images are attached below:

Connected Components Analysis

Percent Filled: 0.485368  
Bounding Box Ratio: 1.250000  
Axis of Least Central Moment: 0.822075



Explorer: main.cpp, operations.cpp, is using the webcam

Output:

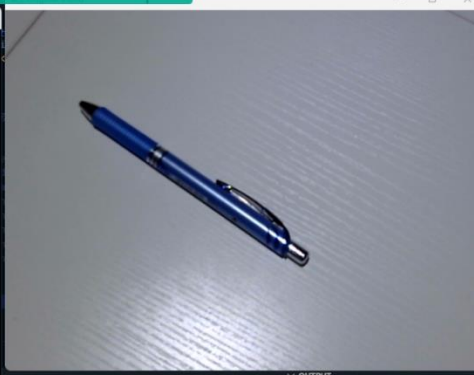
```
Files(CMake/bin/cmake.exe) --build e7/MSVC/CVPR/
projects/projects/tasks/build --config Debug --target
ALL_BUILD -j 22 --
[build] MSBuild version 17.8.3:195e7f5a3 for .NET
Framework
[build]
[build] obj\recog.vcxproj ->
E:\MSVC\CVPR\projects\project3\task1\build\Debug\obj\rec
og.exe
[driver] Build completed: 00:00:00.536
[build] Build finished with exit code 0
```

Timeline:

```
Label: calculator, Time: 2024-02-24 17:05:27
Label: calculator, Time: 2024-02-24 17:05:35
Label: optical mouse, Time: 2024-02-24 17:05:58
Label: calculator, Time: 2024-02-24 17:07:21
Label: smartwatch, Time: 2024-02-24 17:08:25
```

Connected Components Analysis

Percent Filled: 0.651682  
Bounding Box Ratio: 0.539214  
Axis of Least Central Moment: 0.596281



Explorer: main.cpp, operations.cpp, is using the webcam

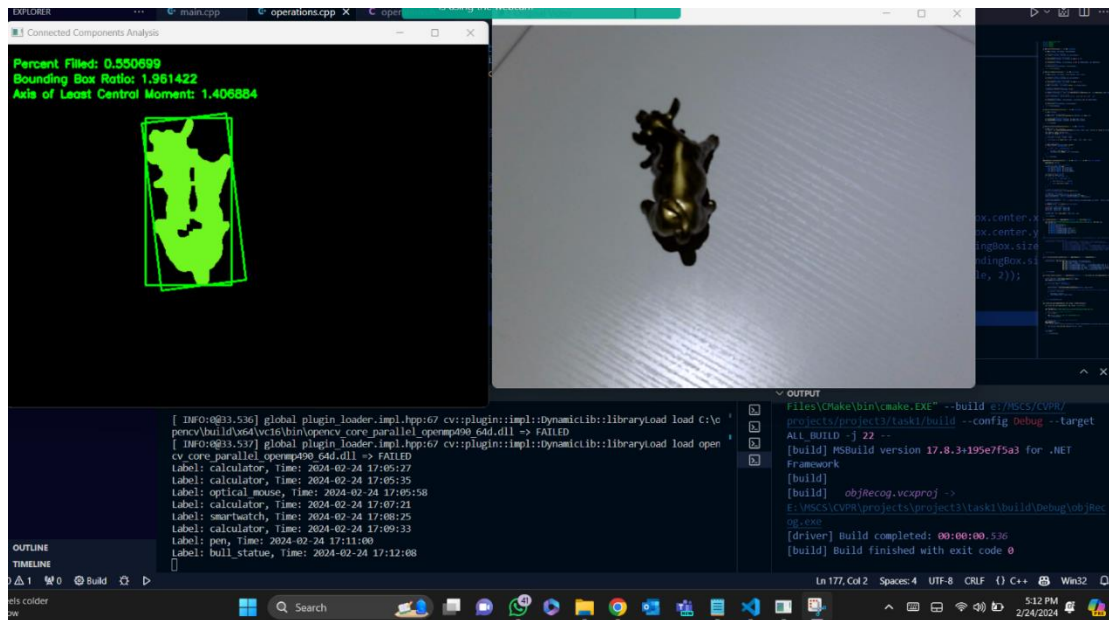
Output:

```
Files(CMake/bin/cmake.exe) --build e7/MSVC/CVPR/
projects/projects/tasks/build --config Debug --target
ALL_BUILD -j 22 --
[build] MSBuild version 17.8.3:195e7f5a3 for .NET
Framework
[build]
[build] obj\recog.vcxproj ->
E:\MSVC\CVPR\projects\project3\task1\build\Debug\obj\rec
og.exe
[driver] Build completed: 00:00:00.536
[build] Build finished with exit code 0
```

Timeline:

```
Label: calculator, Time: 2024-02-24 17:05:27
Label: calculator, Time: 2024-02-24 17:05:35
Label: optical mouse, Time: 2024-02-24 17:05:58
Label: calculator, Time: 2024-02-24 17:07:21
Label: smartwatch, Time: 2024-02-24 17:08:25
Label: pen, Time: 2024-02-24 17:11:00
```

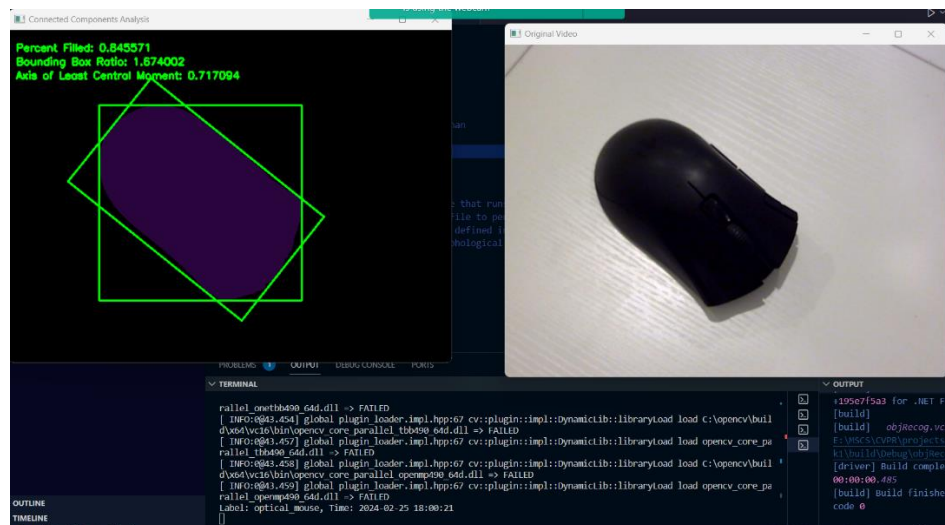




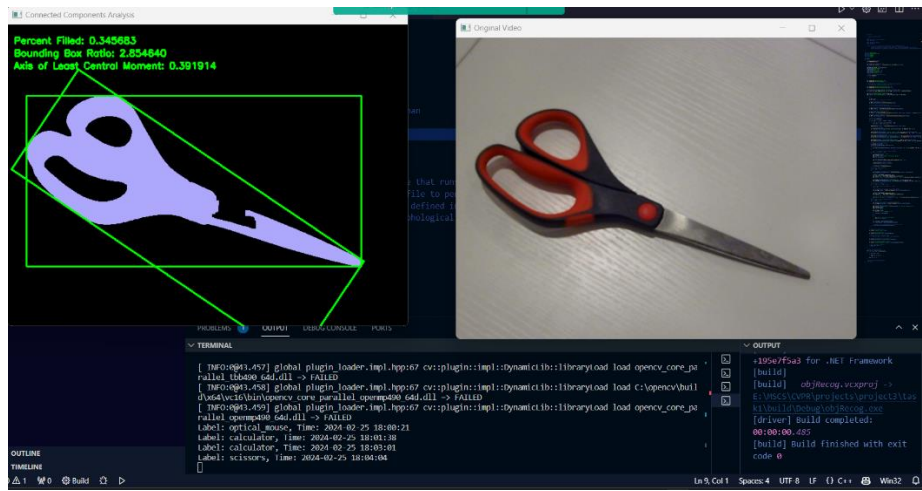
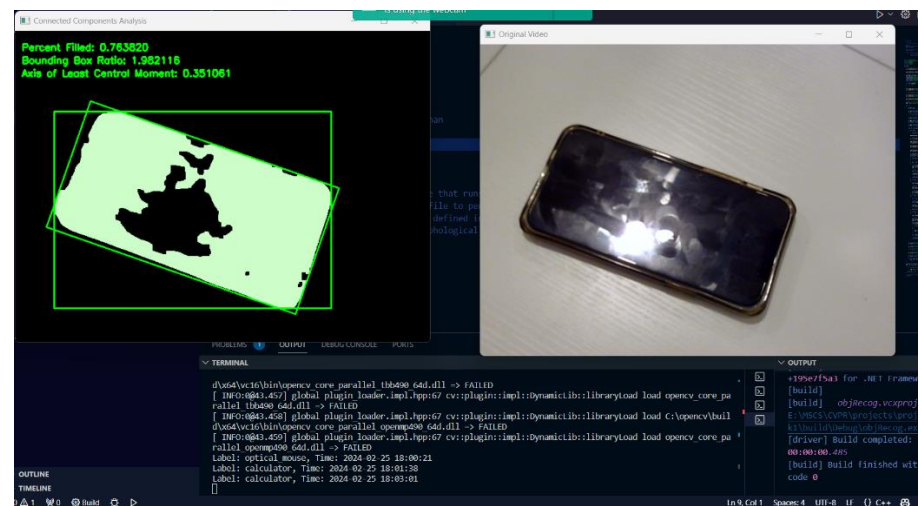
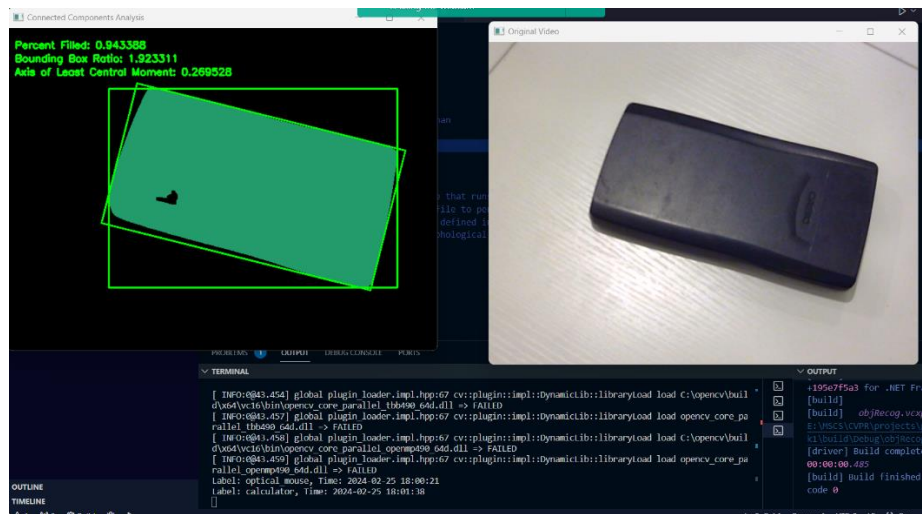
## 2. Testing the object recognition using various distance metrics

### Manhattan Distance:

Apart from Euclidean distance we also tried to compare feature vectors of input object with feature vectors of the objects already stored in database using Manhattan Distance (L1 distance). The Manhattan distance is the sum of the absolute differences of their corresponding values. In this case, it's the sum of the absolute differences of the features of both the objects. The objects that were recognized using Euclidean distance are reused again for testing Manhattan distance. The results are attached below:



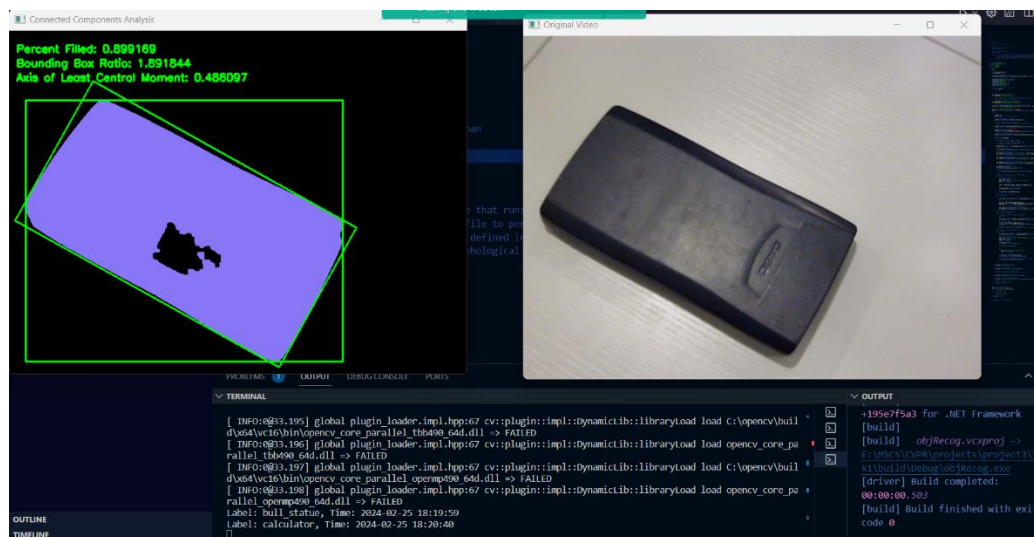
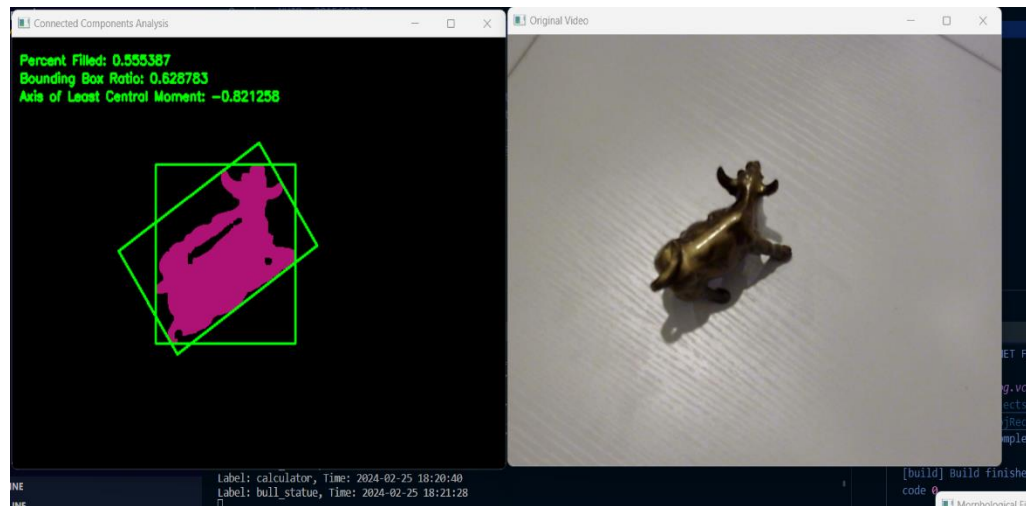


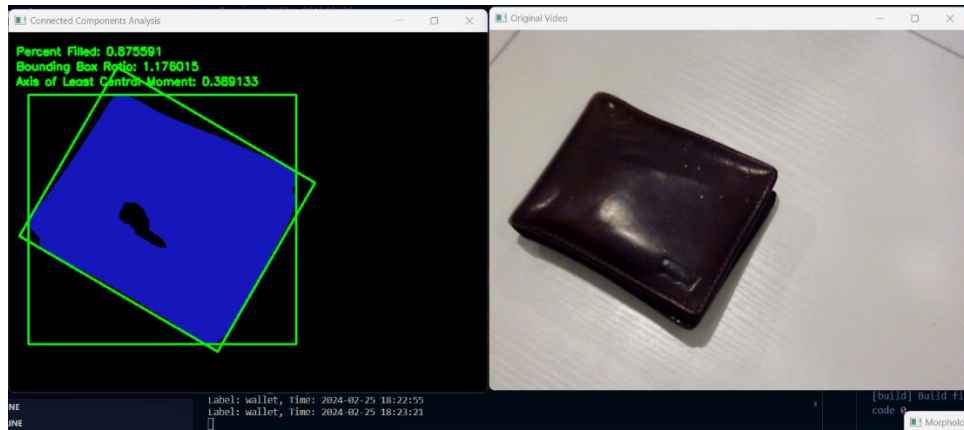


From the above images, if we look at the terminal, we are able to replicate the same result we obtained from using Euclidean distance metric.

### Cosine Distance:

We have also tested the program to use the cosine distance metric for comparing vectors. The cosine distance is a measure of the angle between two vectors, in this case, the vectors formed by the features of the two objects. It's calculated as 1.0 minus the cosine similarity, where the cosine similarity is the dot product of the two vectors divided by the product of their magnitudes. The output of using this function is attached below:





For the above three objects, the program was able to correctly predict the labels and print them in the terminal.

### Observation:

On comparing the performance of all three-distance metric, we can observe that all three perform well but based on the testing we have performed. Euclidean distance metric seems to recognize well than the other two.

### Reflections:

#### Ravi Shankar Sankara Narayanan:

This project has been very effective at teaching me how to build a simple real time object recognition without the use of complicated Neural Networks. Starting from thresholding to object recognition, the instructions were designed to help me clearly understand the data preparation for object recognition. And also, I learnt about the various features that can be extracted from an object and also the various distance metrics that can be used to compare object's features.

#### Haritha Selvakumaran:

Through the completing of this project, I've gained a deeper understanding of tasks like thresholding, feature extraction, and about morphological filters. I have also gained hands-on experience in preprocessing and analyzing images for object recognition. Implementation of morphological filters like erosion and dilation for cleaning up binary images gave me firsthand experience in removing noise, filling in gaps, and smoothing out object boundaries. Also learnt to perform region analysis and solidified my understanding of binary images in general. Overall, this project has laid a strong foundation to explore more advanced image processing and object recognition techniques.