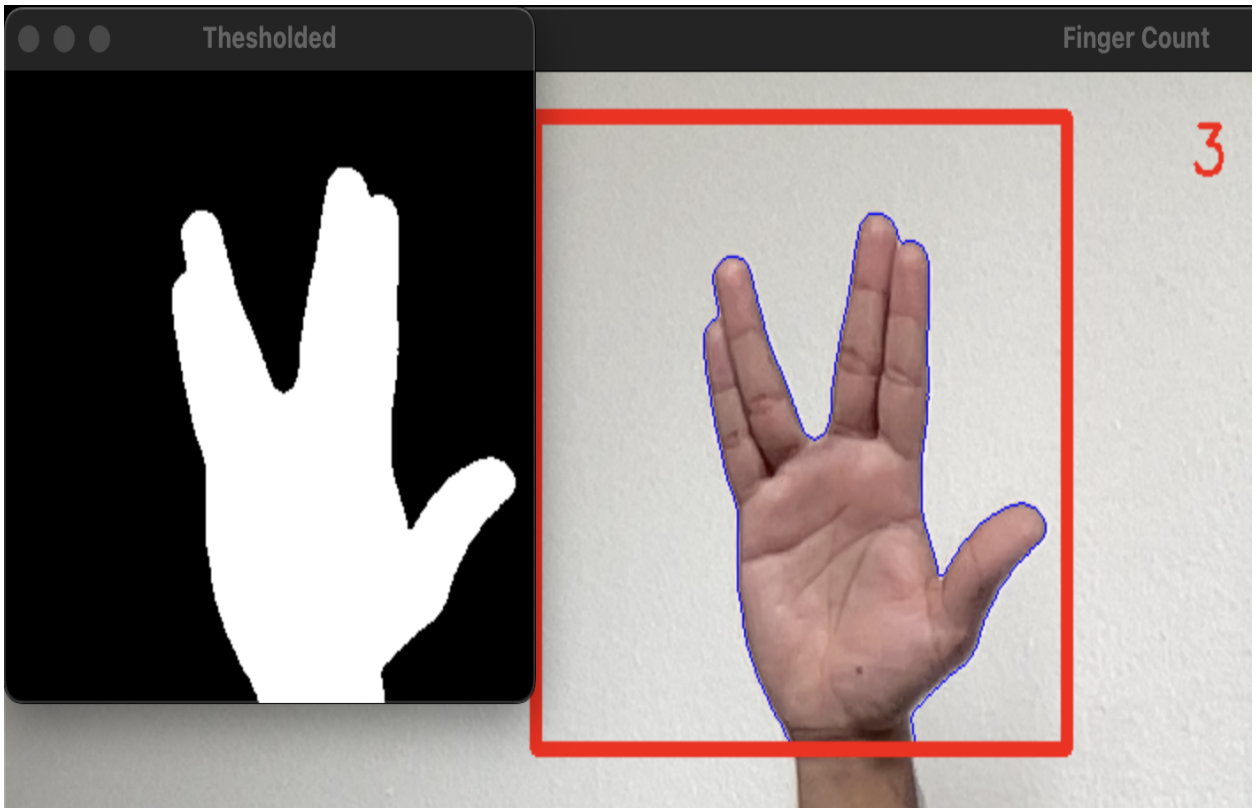
PROJECT REPORT

Parametric Evaluation & Robustness Testing of Hand Segmentation for Counting Fingers in Real-Time

CSCI 5722: Computer Vision

Instructor: Ioana Fleming



Rohit Kharat

04.24.2022

INTRODUCTION: PROJECT IDEA & GOALS

The concept of hand segmentation and counting fingers has been widely used in more complex tasks such as gesture recognition. Gesture recognition has found its application in communication, artificial intelligence, robotics, and the integration of Human-Computer Interaction (HCI).

For our final project, I tested and evaluated the effect of several parameters on hand segmentation and determined the robustness of the algorithm to count fingers in real-time. The parametric study involved studying and evaluating the parameters such as blur method, kernel size, number of neighbors, distance measures, thresholding method, and the threshold value. The robustness of the detection was tested in real-time by blurring the frame and increasing/decreasing brightness.

The tools used for our project are Visual Studio Code and Jupyter Notebook. I used Python language for developing the code and the libraries used are OpenCV, NumPy, Argparse, and sklearn.

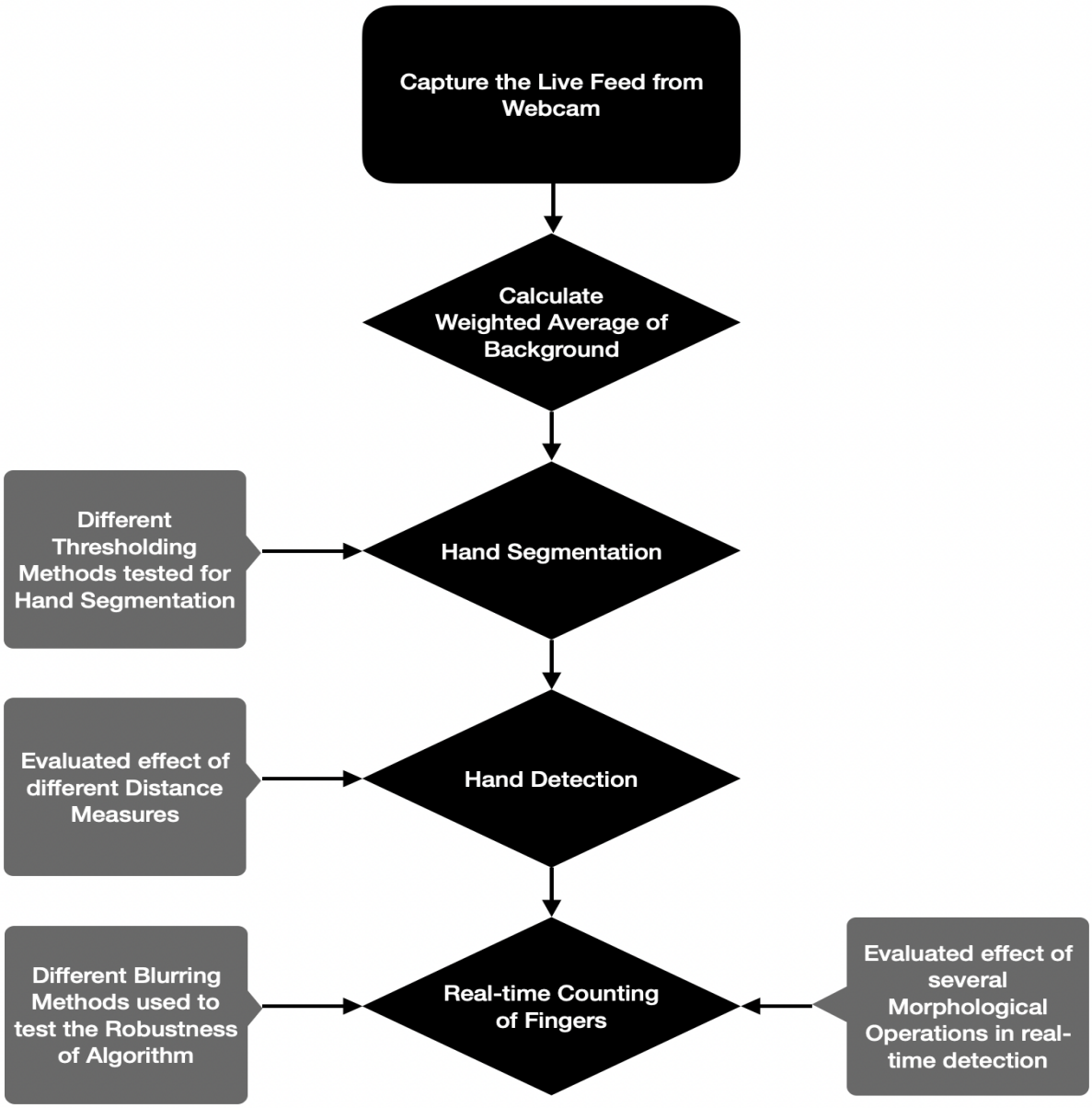
OVERVIEW OF TESTING FILES: HYPOTHESIS & OBJECTIVES

Following is a brief description of the four python files programmed for this project:

1. Capstone_Threshold.py: This python file is used to test the effect of different threshold methods and assess the quality of detection of fingers with changes in parameters. The parameters used for testing are the threshold value and number of neighbors. Our other objective was to determine the best threshold method and to determine the best and worst parameters for segmentation.
2. Capstone_DistMeas.py: This python file is used to evaluate the effect of different distance measures on the counting of fingers. The distance measures used are euclidean, haversine, manhattan, and cosine.
3. Capstone_Morpho.py: This python file is used to determine the effect of different morphological operations on the counting of fingers. The morphological operations performed are eroded, opening, closing, and gradient.
4. Capstone_BlurSmooth.py: This python file is used to test the robustness of detecting the number of fingers in real-time even if the frame is blurred. The blurring methods tested are Gaussian blurring, filter2D, simple blur, median blurring, and bilateral filtering.

COMPUTER VISION PIPELINE: ALGORITHMIC APPROACH

Our pipeline consists of four main parts - calculation of the accumulated average weight of background, segmentation of hand using thresholding methods, counting fingers using convex hull, and real-time capturing the frame to show count of fingers and segmented hand. We used traditional computer vision methods employing OpenCV and avoided the usage of deep learning frameworks for parametric study. The flowchart below shows the overall steps of the computer vision pipeline:



PROCEDURE: COMPUTER VISION METHODS

The pipeline starts with capturing the live feed of the web camera and takes a few seconds to calculate the weighted average of the background. The computation of the accumulated average is carried out by the OpenCV function 'accumulateWeighted' which takes the initial predefined accumulated weight and the next frame of background captured by the webcam.

Next, we have made a function for hand segmentation which takes the frame and the threshold value as inputs and gives the thresholded image and maximum contoured object i.e. hand as output. The absolute difference between the previous frame and the current frame is calculated to separate the foreground from the background. Several thresholding methods are used to evaluate their effect on separating hands from the background. The thresholding types tested in the Capstone_Threshold.py are binary, inverse binary, truncate, the threshold to zero, the threshold to zero inverted, the Otsu algorithm, and the Triangle algorithm. OpenCV function 'threshold' is used to implement these several thresholding types with a user-defined threshold value. Adaptive thresholding is also implemented for the user to test with two adaptive methods i.e. mean and gaussian. OpenCV's 'findContours' function is used to get the external contours and the maximum area contour is returned as we want to get the hand from the background.

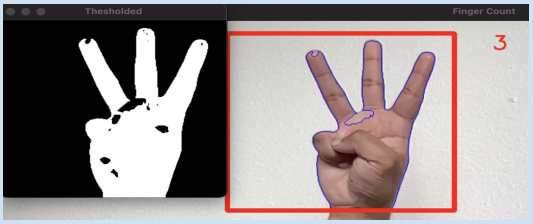
The next function 'countFingers' takes the thresholded image and the hand's external contours and returns the number of fingers held in front of the camera. We used the OpenCV function 'convexHull' which takes the contoured hand and returns at least 4 points formed by the polygon (i.e. hand). We find the center of the polygon and compute the distance between the center and the extreme points. In the file Capstone_DistMeas.py, we implemented 4 different distance measures i.e. euclidean, haversine, manhattan, and cosine to understand the effect of each on forming the region of interest (ROI) in the frame. The ROI is a circle with an 80% radius of the largest distance between the center and the extreme points. 'Bitwise_and' function is used to extract the hand segment from the circular ROI. Finally, external contours help us to get the hand contours from the circular area and a loop is used for detecting the number of fingers from the ROI. The fingers are counted based on two conditions: 1) The upper region of the wrist should be used to form the polygon 2) 25% of the circumference of the circular ROI is used to limit the number of points that fall outside of the hand segment. If both these conditions are satisfied, we increment the count and display it on the frame

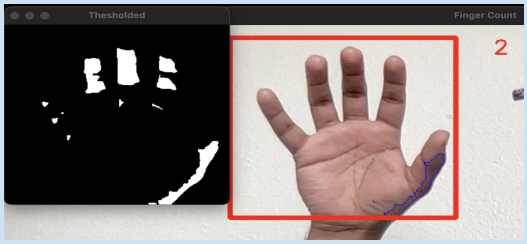
for the user.

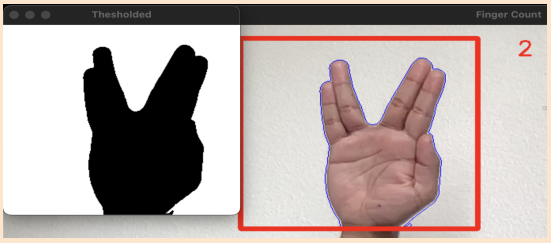
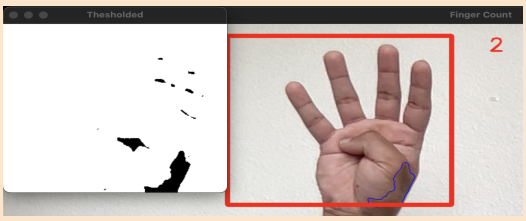
The file Capstone_BlurSmooth.py has several blurring and smoothing methods implemented which test the robustness of the algorithm in real-time as the kernel size is increased to blur and smoothen the frame. The effect of blurriness affects the performance of detection of hand segments particularly fingers after some number of kernel sizes and sigma value. The file Capstone_Morpho.py determines the effect of several morphological operations performed in real-time on the frame. Some morphological operations tend to help in enhancing the detection of hand segments and eventually give better performance in counting the number of fingers.

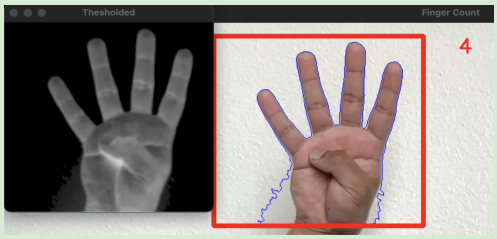
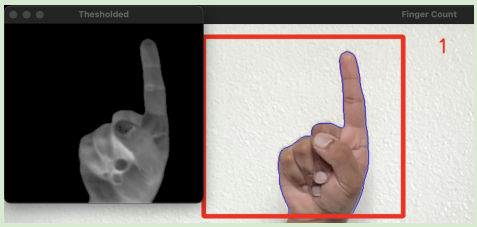
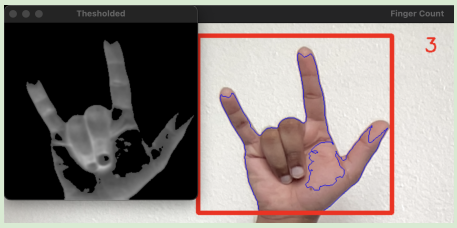
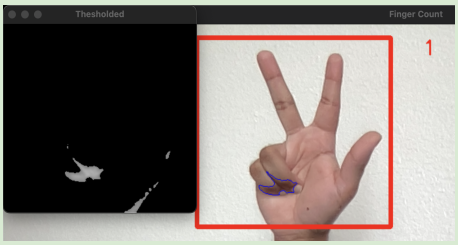
PARAMETRIC STUDY: IMAGE ANALYSIS

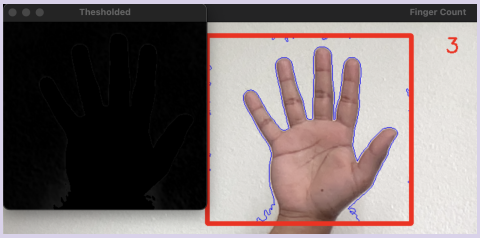
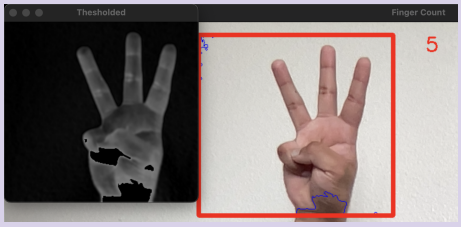
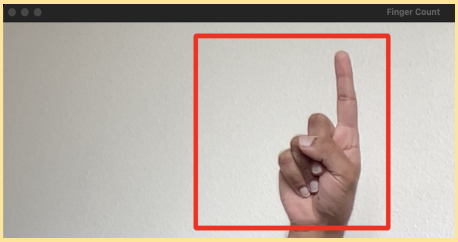
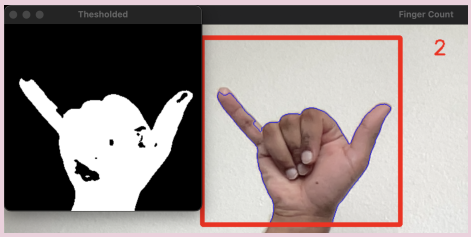
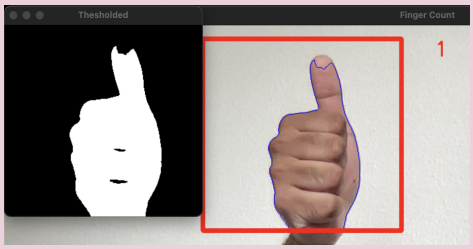
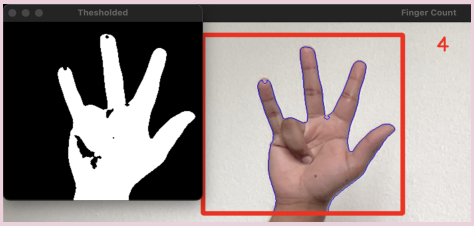
A) Evaluation of different Thresholding Methods and their Parameters

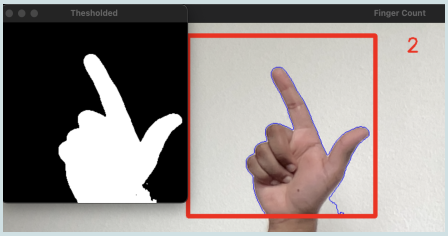
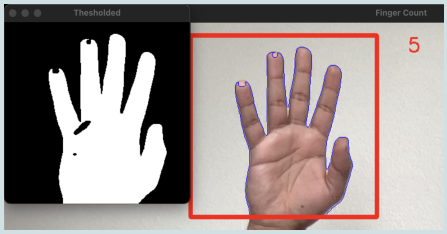
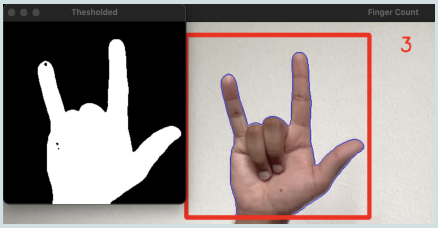
Thresholding Type	Threshold Value	Output
THRESH_BINARY	25	
THRESH_BINARY	50	
THRESH_BINARY	75	

Thresholding Type	Threshold Value	Output
THRESH_BINARY	127	

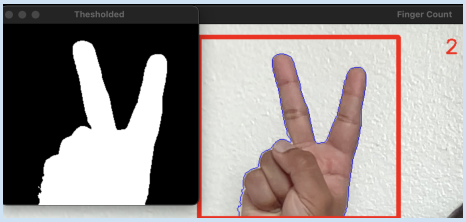
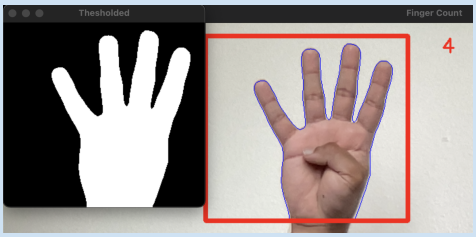
Thresholding Type	Threshold Value	Output
THRESH_BINARY_INV	25	
THRESH_BINARY_INV	75	
THRESH_BINARY_INV	125	
THRESH_TRUNC	25	

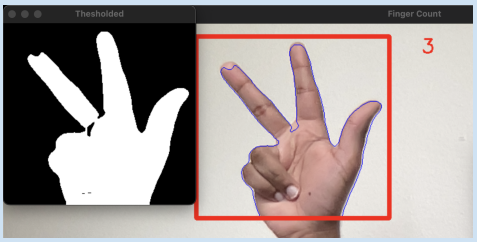
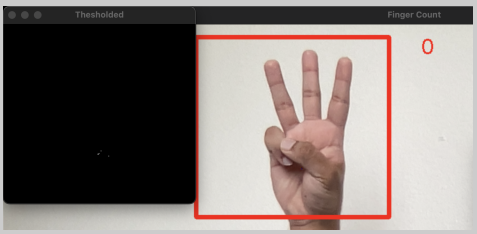
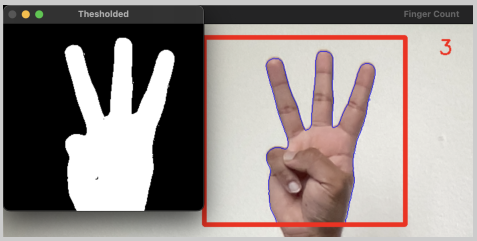
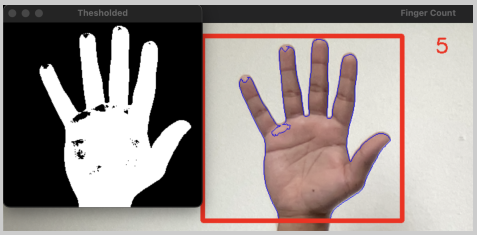
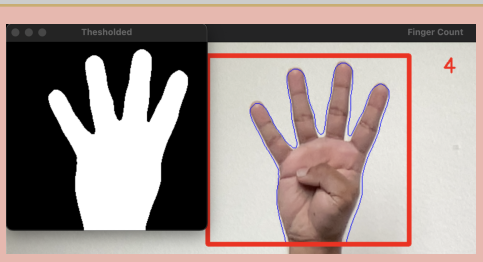
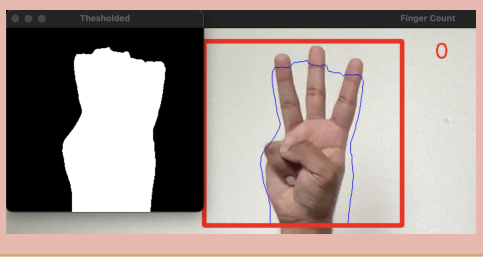
Thresholding Type	Threshold Value	Output
THRESH_TRUNC	75	
THRESH_TRUNC	127	
THRESH_TOZERO	25	
THRESH_TOZERO	50	
THRESH_TOZERO	75	
THRESH_TOZERO	127	

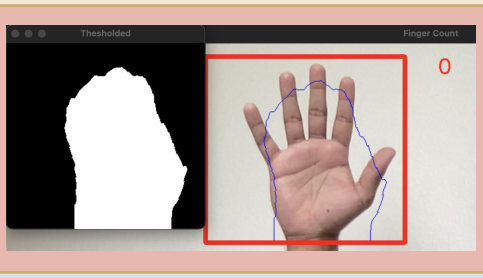
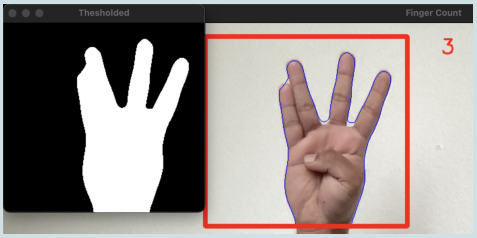
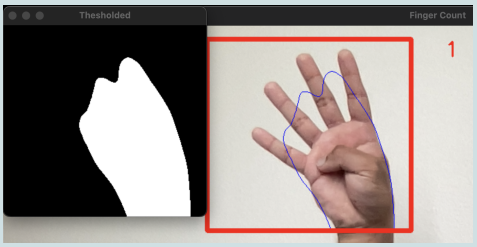
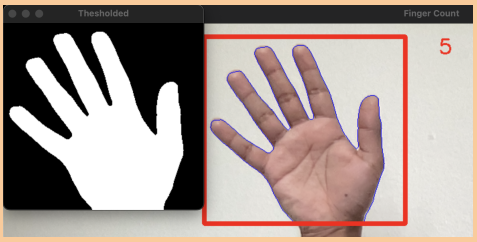
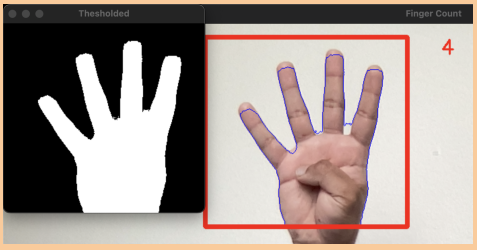
Thresholding Type	Threshold Value	Output
THRESH_TOZERO_INV	25	
THRESH_TOZERO_INV	125	
THRESH_MASK	25, 50, 127	
THRESH_OTSU	25	
THRESH_OTSU	100	
THRESH_OTSU	127	

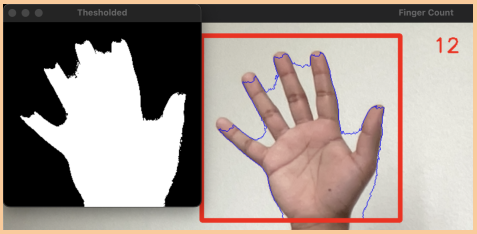
Thresholding Type	Threshold Value	Output
THRESH_TRIANGLE	25	
THRESH_TRIANGLE	100	
THRESH_TRIANGLE	125	

B) Testing the robustness of counting fingers by using different blurring methods and changing the kernel sizes & sigma values

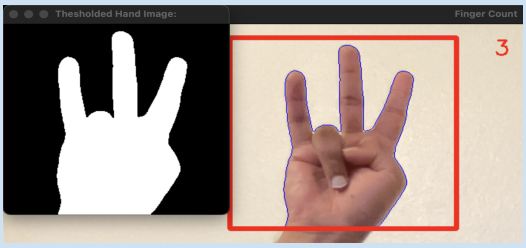
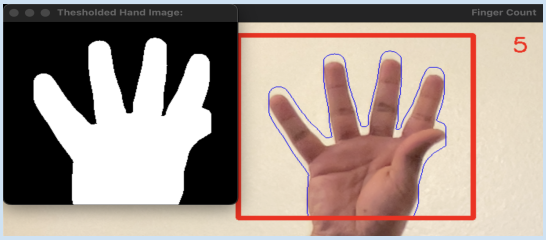
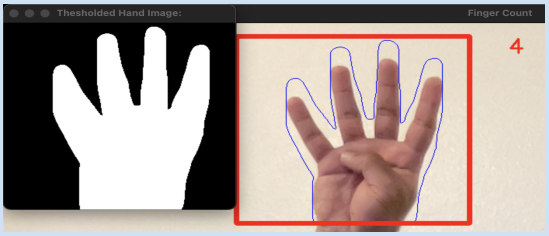
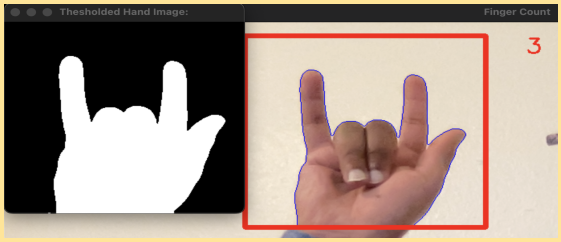
Blurring/Smoothing Method	Kernel Size	Sigma	Output
Gaussian Blur	7	0	
Gaussian Blur	9	10	

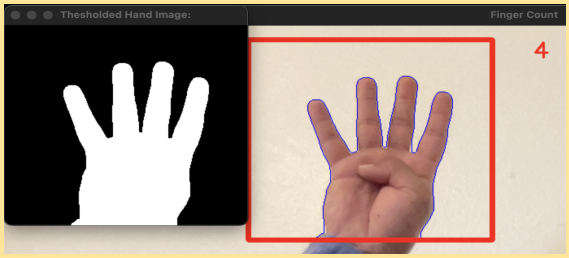
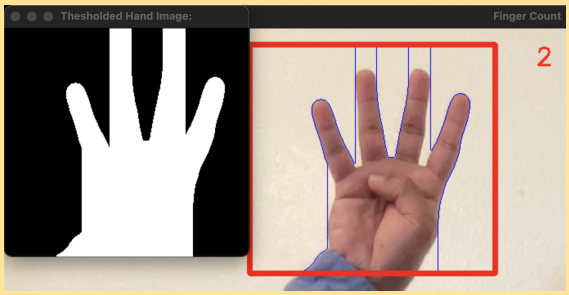
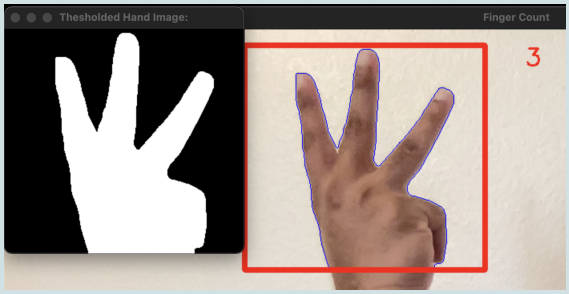
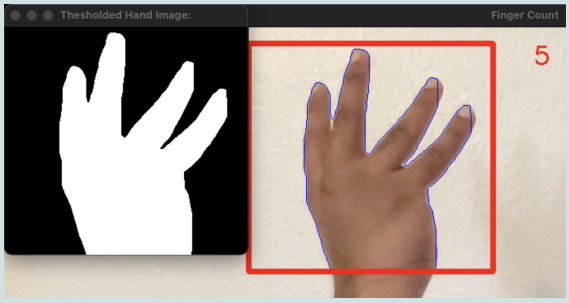
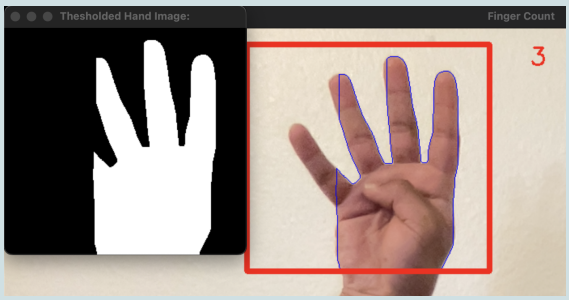
Blurring/Smoothing Method	Kernel Size	Sigma	Output
Gaussian Blur	11	50	
Filter 2D	7	-	
Filter 2D	1	-	
Filter 2D	0.5	-	
Blur	17	-	
Blur	55	-	

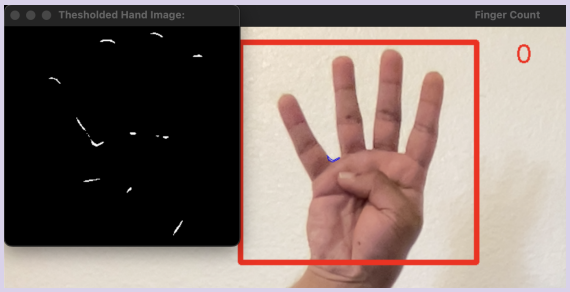
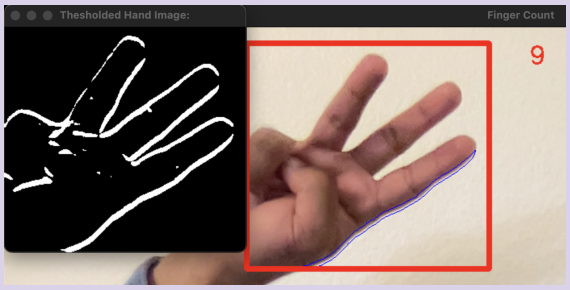
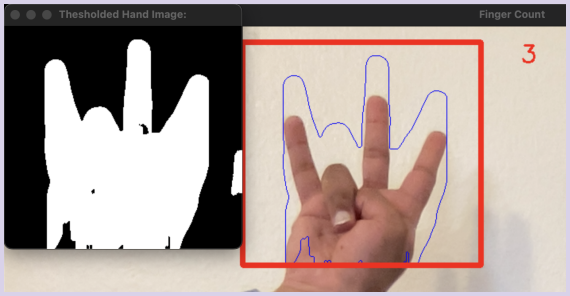
Blurring/Smoothing Method	Kernel Size	Sigma	Output
Blur	101	-	
Median Blur	19	-	
Median Blur	75	-	
Bilateral Filter	11	-	
Bilateral Filter	37	-	

Blurring/Smoothing Method	Kernel Size	Sigma	Output
Bilateral Filter	63	-	

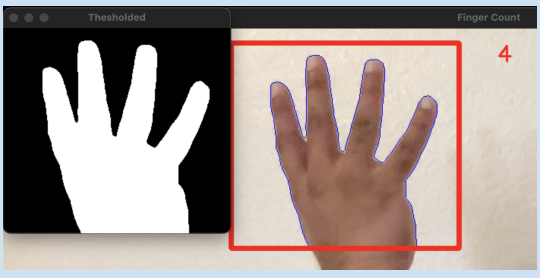
C) Determination of several morphological operations on counting fingers

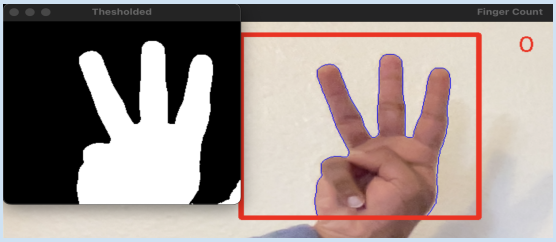
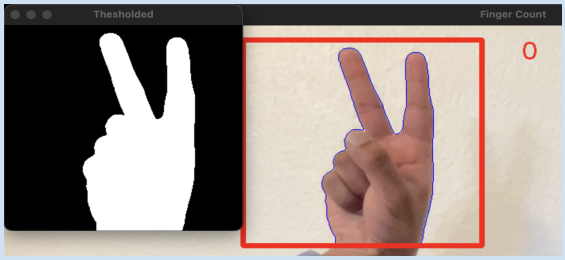
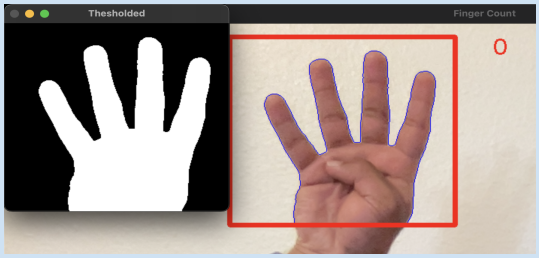
Morphological Operation	Kernel Size	Output
Erode	1	
Erode	7	
Erode	15	
MORPH_OPEN	7	

Morphological Operation	Kernel Size	Output
MORPH_OPEN	19	
MORPH_OPEN	37	
MORPH_CLOSE	11	
MORPH_CLOSE	25	
MORPH_CLOSE	57	

Morphological Operation	Kernel Size	Output
MORPH_GRADIENT	3	
MORPH_GRADIENT	15	
MORPH_GRADIENT	43	

D) Evaluation of the effect of different distance measures in counting fingers

Distance Measure	Output
Euclidean	

Distance Measure	Output
Haversine	
Manhattan	
Cosine	

EFFECTS OF PARAMETERS: SUCCESSFUL & UNSUCCESSFUL IMPLEMENTATIONS

A) Evaluation of different Thresholding Methods and their Parameters

Different thresholding methods and values were used to evaluate their effect on hand segmentation. Most threshold types perform poorly with increasing threshold values except the Otsu and Triangle method. Some threshold types like Binary Inverted, Truncate, ToZero Inverted, and Mask performed worst with a range of threshold values. ToZero type works pretty well with segmentation as well as counting of fingers. The hand segmentation after using ToZero looks like an 'X-ray' image, giving it a look of an actual hand but still performing decently in counting fingers. The Otsu and Triangle type works well even with increasing noise in the frame.

B) Testing the robustness of counting fingers by using different blurring methods

Six different blurring/smoothing techniques were used to test the robustness of the detection of hand and counting fingers in real-time. 'GaussianBlur' function works well with an increasing number of kernel sizes and sigma values but starts to perform poorly after too much smoothing. The algorithm suffers mostly from the blurring of fingertips and thumbs, which leads to inaccurate segmentation. The 'filter2D' function performs better with lower kernel sizes but performs badly with kernel sizes greater than 3. All blurring methods perform badly with greater kernel sizes and leads to negligible segmentation of fingers and in some case even the wrist. The 'bilateralFilter' function performs well with higher kernel sizes but still degrades after some value. This function is the most resource-consuming and slowest compared to other smoothing methods.

C) Determination of several morphological operations on counting fingers

Four different morphological operations were performed on the frame after smoothing the edges in the frame. Morphological operations are helpful in effective segmentation and detecting contours in an image. I expected the same results for our application but in real-time. All operations performed exceptionally well in extracting hands from the background and counting fingers except 'MORPH_GRADIENT'. Other operations perform well at lower kernel sizes, but 'MORPH_GRADIENT performs somewhat better at higher kernel size values and gives a poor result at smaller values.

D) Evaluation of the effect of different distance measures in counting fingers

The objective of this test was to assess the effect of different distance measures for effective distinguishing of fingers. But, the tests confirmed that only the 'euclidean' measure performs best compare to other measures such as 'haversine', 'manhattan', and 'cosine'. The reason for the 'haversine' measure to perform badly is because it is better suited for measuring geometric distance on earth. The 'manhattan' measure measures the distance between two points along right axes and because of it fails to get the closest distance between two fingers. The 'cosine' measure works well with vectors and fails here to get the shortest distance.

CONTRIBUTIONS

Rohit Kharat: My contribution was in the parametric study, evaluation, and robustness testing of hand segmentation for counting fingers in real-time. The files submitted by me are as follows:

1. Capstone_Threshold.py
2. Capstone_BlurSmooth.py
3. Capstone_Morpho.py
4. Capstone_DistMeas.py

Reid Glaze: Reid's contribution to our project was to add functionalities to our algorithm and develop code that gives accurate results in extreme situations such as noise, blurriness, and contrast. The file submitted by Reid is

1. CVProject.py

THINGS LEARNED DURING THIS PROJECT

1. Parametric study and evaluation helped me to learn the effect of parameters on the effectiveness of a Computer Vision pipeline and to determine which parameters work and which do not.
2. How simple Computer Vision tasks such as thresholding, contours, blurring, and convex hull can be utilized for more complex tasks such as real-time hand segmentation and counting fingers.
3. Learned the importance of testing a Computer Vision pipeline for robustness in extreme natural conditions for better performance in real-world scenarios.
4. Developed the habit of getting to the root cause behind the successful and unsuccessful implementation.

ADVICE TO FUTURE STUDENTS

My most important advice to future students will be to enroll in the Computer Vision course even if you are from different background. I and Reid are from the Mechanical department but we learned a lot from this course. I find this course particularly helpful for me to understand machine vision fundamentals and to apply my learnings for practical applications. Computer vision has found its applications in almost every field and will continue to grow. If you are someone who is enthusiastic to learn computer vision and apply it in your domain, this course will help you to make your fundamentals strong and to understand computer vision libraries with ease.