Hand gesture and finger count recognition

Zdrenghea Maria-Iulia
Technical University of
Cluj-Napoca
Email: zdrenghea.va.maria@student.utcluj.ro

***Abstract*—In this project, I am going to develop a hand gesture and finger count recognition app. The project will be used on a laptop and it will be able to recognize how many fingers I hold up.**

I. Introduction

The task is to develop a system capable of recognizing hand gesture from camera. This system can be also integrated into various applications, such as virtual assistants, educational tools and communication devices. It can be used for deaf people who are taking a quiz. They choose an answer based on how many fingers they are holding up

For this particular project we are looking in recognizing how many fingers a person holds up.

The goal of this project is to be be able to detect hand gestures and count fingers for a person who doesn't know how to do it or how to interpret it. The program will be able to detect a how many firngers are hold up from the camera. We will develop methods to extract relevant features from the image such as hand shape and position.

The implementation will require image processing: which will be able to implement algorithms for processing the image as well as background subtraction is performed using the MOG2 algorithm. Then, non-human objects are darkened by applying specialized filters designed to detect human skin color. After this, the median filter is used to reduce non-uniform noise in the image signals as much as possible. The image is then converted to the HSV color space and subsequently to grayscale. Finally, as the last filtering step, the Otsu algorithm is combined with binary thresholding to mask values below the threshold with black.

My contribution will primary focus on implementing the image processing techniques, feature extraction algorithms in C++.

II. Bibliography Study

There are a few steps that first need to be documented when creating such a big project. First we are interested just in some features from the image, so we are going to define a Region of Interest, which will be filtered and then we are going to perform some operations on it. Secondly we are going to define a contour that joins all points along the boundary of the image with the same intensity. Third we are defining a convex hull as the intersection of all convex sets containing a given subset of Euclidean space.

When discussing background reduction and skin color filtering we are first interested in determining the probability density function for for each pixel in the image. Because the camera is stationary we can look at the background as a picture. We determine the variances for pixel intensity levels using a single Gaussian model.

For identifying the skin we are combining various color spaces RGB and Gray transform.

For finding the contour of the hand we perform geometric operations on the hand image which is black and white and in the ROI. In OpenCV we can find the edges of objects with the findContours function. The convex hull algorithm is an algorithm that finds the line segments whose contours connect the outermost points of a known set of points. At the same time we will use the points where these line segments meet for out operations. By counting the extreme points we will count the number of fingers present in the image.

III. Method

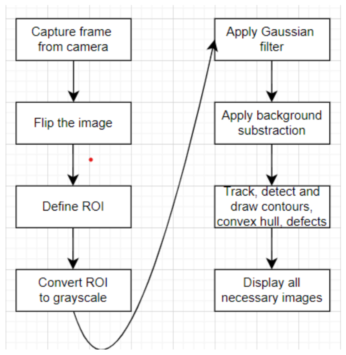

Capture frame from camera
Flip the image
Define ROI
Convert ROI to grayscale
Apply Gaussian filter
Apply background substraction
Track, detect and draw contours, convex hull, defects
Display all necessary images

Fig. 1. Overview figure

There are several steps involved in solving this problem.

First we are going to capture the video. For that we are using VideoCapture and we are constantly reading the frame.

Secondly we do are defining the region of interest as a square, there we are going to place our hand and then we will convert it to GrayScale then we will apply a blur and also dilate the image to decrease the noise.

Now I will explain how we track how many fingers we hold up. First we are going to initialize a variable that counts the number of fingers and a string which will print on the final image the number from the previous variable that counts the fingers. After that we declare a vector that has a vector of points that keeps the contours and we also declare a vector to store the hierarchy of the contours.

First we apply Gaussian filter on the image that has the ROI to smothen the image and reduce noise, then we apply binary thresholding to convert the image into binary, after that we apply the canny edge detector to the threshold image and then we apply find contours on the binary image to get the vector of contours.

After that we create a new image that has the same size as the canny image, but it is a black image because it will be used to draw the contours. Then we check if there are any contours found and if we find them we loop through each detected contour to find the largest one by comparing their sizes.

Then we are declaring some vectors for storing important variables that help us find the points of convexity defects. After this we are declaring some rectangles to store the minimum area rectangle for each contour.

Now for drawing the contours and hull points we are going to iterate through the contours, check if the are is greater than 5000, this threshold is used to filter small contours that are not significant. We compute the convex hull of the current contour. The convex hull is the smallest polygon that can enclose the contour points. Computes the convexity defects of the current contour with respect to its convex hull. Convexity defects are the deviations of the contour from the convex hull. Now we check if the current contour is the largest contour found and if so, we compute the minimum area rectangle that can enclose the current contour. Now we loop through each point in the convex hull, retrieve the points in the original contour and add then to the hull points and reset count variable to 0, this is used to count convexity defects. Now we go through each convexity defects of the current contour, filter the defects add it to the defect, we draw a circle on the endpoint of the defect and then increment count. In the end based on the count of the significant defects we assign a value to ’a’. Then we draw all necessary information on the images. The count is put on the initial image, the contour and convex hull points on the final image and we approximate the contour to a polygon. And in the end we draw the line connecting the corner points of the minimum area rectangle.

IV. Evaluation and Results

To evaluate the effectiveness of our image processing workflow, we conducted a series of experiments on a diverse set of images and real-time video feeds. The images included various hand gestures and backgrounds to test the robustness and accuracy of our approach. We utilized a standard webcam with a resolution of 720p and conducted tests under different lighting conditions to simulate real-world scenarios.

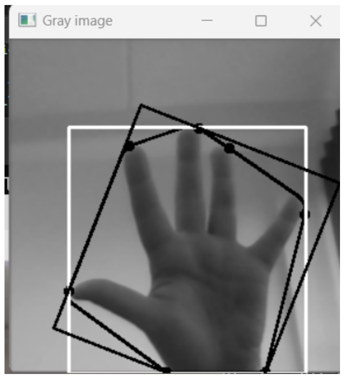


Fig. 2. Enter Caption

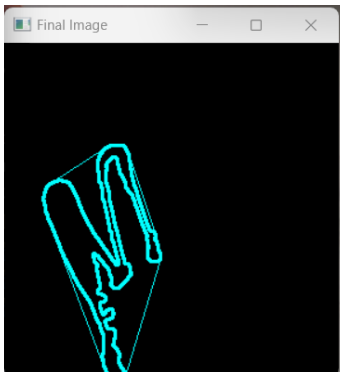


Fig. 3. Enter Caption

V. Conclusion

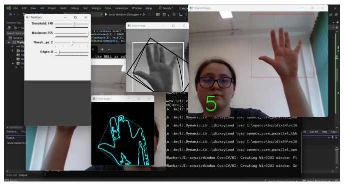
In this project, we addressed the problem of real-time hand gesture recognition using image processing techniques with OpenCV. The objective was to detect and identify hand gestures in a video feed captured from a webcam, utilizing various image processing steps such as background subtraction, noise reduction, edge detection, and contour analysis. Our approach was based on leveraging the MOG2 algorithm for background subtraction, combined with custom filters to detect human skin color, and a series of transformations to accurately identify and count fingers.

The project successfully demonstrated the capability to recognize and count hand gestures in real-time. The system was able to:

- Detect the hand in various lighting conditions and backgrounds.
- Detect the hand in various lighting conditions and backgrounds.
- Identify and count fingers with a high degree of accuracy.
- Provide a visual output showcasing the processed steps, from initial capture to final gesture recognition.

While the solution performed well, there are areas for potential improvement:

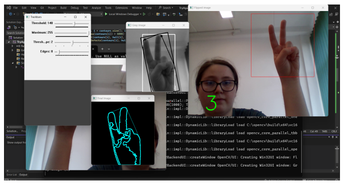
- Enhanced Skin Detection: Integrating additional color spaces and machine learning techniques to improve the



Threshold: 140
Maximum: 255
Thresh...pe: 2
Edges: 8

```
in::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16
in::impl::DynamicLib::libraryLoad load opencv_core_parallel_tbb
in::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16
in::impl::DynamicLib::libraryLoad load opencv_core_parallel_ope
2BackendUI::createWindow OpenCV/UI: Creating Win32UI window: Fl
2BackendUI::createWindow OpenCV/UI: Creating Win32UI window: Gr
```

Fig. 4. Enter Caption



Threshold: 140
Maximum: 255
Thresh...pe: 2
Edges: 8

```
in::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16
in::impl::DynamicLib::libraryLoad load opencv_core_parallel_tbb
in::impl::DynamicLib::libraryLoad load C:\opencv\build\x64\vc16
in::impl::DynamicLib::libraryLoad load opencv_core_parallel_ope
2BackendUI::createWindow OpenCV/UI: Creating Win32UI window: Fl
2BackendUI::createWindow OpenCV/UI: Creating Win32UI window: Gr
```

Fig. 5. Enter Caption

robustness of skin color detection under various lighting conditions.
- Optimization for Speed: Optimizing the processing pipeline for faster execution, which is crucial for real-time applications.
- Advanced Noise Reduction: Exploring more advanced noise reduction techniques to further enhance image quality and contour detection accuracy.
- Gesture Recognition: Expanding the functionality to include specific gesture recognition and classification, enabling more interactive applications.
- Integration with Other Sensors: Combining visual data with other sensor inputs, such as depth sensors, to improve the accuracy and reliability of hand detection.