COMSATS University Islamabad, Wah Campus

Project Submission Report

Subject: Programming Fundamentals

Project: Gesture Recognition System

Submitted to:

Dr. Samia Riaz

Submitted By:

Eman Zahra FA23-BCS-043-2A

Saad Zaffar Laghari FA23-BCS-169-2A

Date: May 26, 2024.

Executive Summary:

This project outlines the development of a C++ application that utilizes a camera to recognize hand gestures and perform corresponding actions. The system will be designed to detect the number of fingers shown and trigger predefined commands based on the recognized gesture.

The system will employ computer vision techniques to process camera input and identify the number of fingers extended in the frame.

Specific actions will be linked to each recognized gesture. For instance: 1 Finger: Open the Browser, 2 Fingers: Open Calculator, 3 Fingers: Open Spotify, 4 Fingers: Open Task Manager

This gesture recognition system offers a hands-free approach to interact with the computer, enhancing accessibility and convenience. The command mapping can be tailored to user preferences, allowing for personalized control over frequently used applications.

Introduction:

The evolution of human-computer interaction (HCI) constantly seeks new methods for natural communication. Gesture recognition holds immense potential in this domain, enabling users to interact with computers through hand movements. The system aims to bridge the gap between traditional input methods and a more natural, hands-free approach.

This document outlines the design and implementation of the system, focusing on its core functionalities of recognizing the number of fingers displayed in a camera frame and triggering pre-defined actions based on the recognized gesture.

Objective:

This project aims to develop a C++ gesture recognition system that detects the number of fingers shown through a camera. Based on the detected fingers (1-5), the system will trigger pre-defined actions like opening a browser, media player, or launching specific applications, providing a hands-free and intuitive way to interact with the computer.

Scope:

Target Audience:

This project is designed for two primary audiences:

- **General Users:** Individuals seeking a more intuitive way to interact with their computers. The system prioritizes ease of use, making it accessible to users with varying levels of technical expertise.
- **C++ Programmers:** The project provides practical application of core C++ concepts while delving into computer

vision techniques. Programmers can enhance their skillset by working with libraries like OpenCV and implementing image processing algorithms.

Programming Skills:

This project assumes a basic understanding of C++ programming concepts like:

- Variables and Data Types
- Control Flow Statements (if/else, loops)
- Functions
- User Input and Output

Experience with computer vision libraries like OpenCV is a plus but not mandatory. The project aims to provide a clear implementation path for beginners interested in exploring this field.

Project Code:

```
#include <opencv2/opencv.hpp>

#include <iostream>

#include <vector>

#include <string>

#include <cmath>

#include <Windows.h>

#include <shellapi.h>


using namespace cv;

using namespace std;
```

```
// Define platform-specific commands to open applications
void open_browser() {
    ShellExecuteW(0, 0, L"http://www.google.com", 0, 0 , SW_SHOW );
}
void open_calculator() {
    ShellExecuteW(0, 0, L"calc", 0, 0 , SW_SHOW );
}
void open_spotify() {
    ShellExecuteW(0, 0, L"spotify://", 0, 0 , SW_SHOW );
}
void open_task_manager() {
    ShellExecuteW(0, 0, L"taskmgr", 0, 0 , SW_SHOW );
}

// Function to keep the window always on top (Windows only)
void set_window_on_top(const string& window_name) {
    HWND hwnd = FindWindowA(NULL, window_name.c_str());
    if (hwnd) {
        SetWindowPos(hwnd, HWND_TOPMOST, 0, 0, 0, 0, SWP_NOMOVE | SWP_NOSIZE);
    }
}

int main() {
    // Initialize the last action time to prevent rapid actions
    double last_action_time = (double)getTickCount() / getTickFrequency();
    const double action_delay = 5.0; // 5 seconds delay between actions

    // Opening the Camera
    VideoCapture capture(0);
```

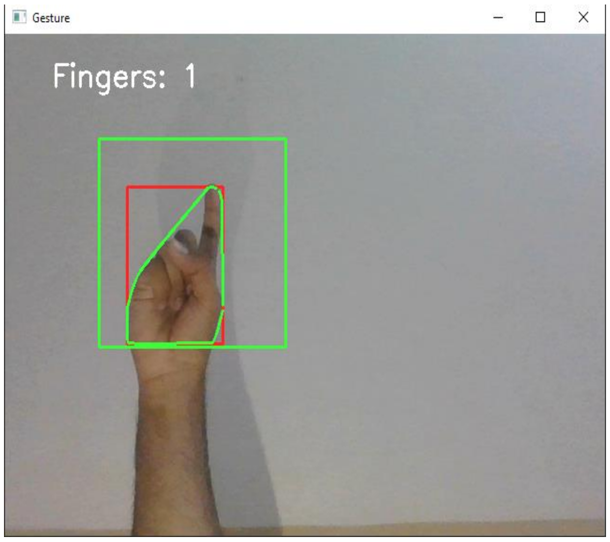
```
if (!capture.isOpened()) {
    cerr << "Error: Unable to open camera." << endl;
    return -1;
}
while (capture.isOpened()) {  // Loop until camera is closed
    Mat frame, crop_image, blur, hsv, mask2, dilation, erosion, filtered, thresh, drawing;

    // Capture frames from the camera
    capture >> frame;

    // Get hand data from the rectangle sub window (ROI)
    Rect roi(100, 100, 200, 200);
    crop_image = frame(roi);

    // Draw the ROI rectangle on the main frame
    rectangle(frame, roi, Scalar(0, 255, 0), 2);

    // Apply Gaussian blur with larger kernel size to smoothen the image
    GaussianBlur(crop_image, blur, Size(5, 5), 0);

    // Change color-space from BGR (default OpenCV) -> HSV (better for skin detection)
    cvtColor(blur, hsv, COLOR_BGR2HSV);

    // Create a binary image with white as skin colors and the rest black (for segmentation)
    inRange(hsv, Scalar(0, 20, 70), Scalar(20, 255, 255), mask2);

    // Kernel for morphological transformation (to clean up the mask)
    Mat kernel = getStructuringElement(MORPH_ELLIPSE, Size(7, 7));
```

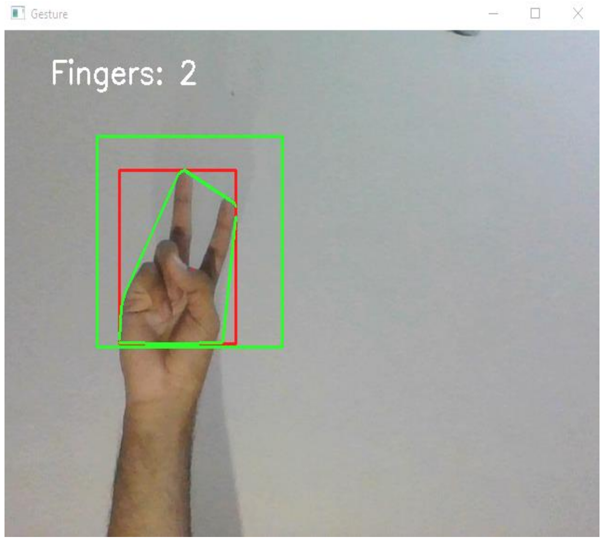
```
dilate(mask2, dilation, kernel, Point(-1, -1), 1);    // Expand white regions (skin)
erode(dilation, erosion, kernel, Point(-1, -1), 1);   // Shrink white regions to remove noise

// Apply Gaussian Blur and Threshold to further refine the mask
GaussianBlur(erosion, filtered, Size(3, 3), 0);
threshold(filtered, thresh, 127, 255, THRESH_BINARY);

// Show threshold image (mostly for debugging)
imshow("Thresholded", thresh);

// Find contours in the thresholded image (these represent the hand shape)
vector<vector<Point>> contours;
findContours(thresh, contours, RETR_TREE, CHAIN_APPROX_SIMPLE);

// Create a blank image for drawing contours later
drawing = Mat::zeros(crop_image.size(), CV_8UC3);

// If contours were found, proceed with hand analysis
if (!contours.empty()) {
  try {
    // Find contour with maximum area (assumed to be the hand)
    auto contour = *max_element(contours.begin(), contours.end(),
      [](const vector<Point>& a, const vector<Point>& b) {
        return contourArea(a) < contourArea(b);
      });

    // Create bounding rectangle around the contour (hand)
    Rect rect = boundingRect(contour);
    rectangle(crop_image, rect, Scalar(0, 0, 255), 2);
```

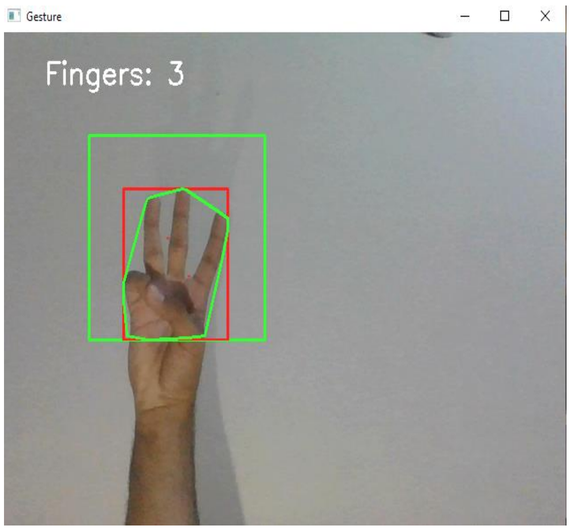
```
// Find convex hull (simplified outline of the hand)
vector<Point> hull;
convexHull(contour, hull);

// Drawing contour and hull on images
drawContours(drawing, vector<vector<Point>>{contour}, -1, Scalar(0, 255, 0), 2);
drawContours(drawing, vector<vector<Point>>{hull}, -1, Scalar(0, 0, 255), 2);

// Find convexity defects (indentations between fingers)
vector<int> hullIndices;
convexHull(contour, hullIndices, false);
vector<Vec4i> defects;
if (hullIndices.size() >= 3) { // Need at least 3 points for convexity defects
    convexityDefects(contour, hullIndices, defects);
}

// Count fingers based on convexity defects
int count_defects = 0;
for (const auto& defect : defects) {
    int startIdx = defect[0]; Point startPoint = contour[startIdx];
    int endIdx = defect[1]; Point endPoint = contour[endIdx];
    int farIdx = defect[2]; Point farPoint = contour[farIdx];

    // Calculate distances and angle (using cosine rule) to determine finger tips
    double a = norm(endPoint - startPoint);
    double b = norm(farPoint - startPoint);
    double c = norm(endPoint - farPoint);
    double angle = (acos((b * b + c * c - a * a) / (2 * b * c)) * 180) / 3.14;
```

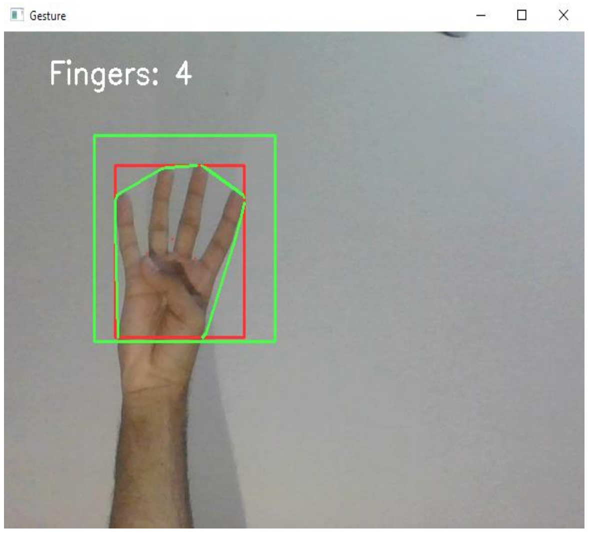
```
                // If angle is small enough, it will likely to be a finger tip
                if (angle <= 90) {
                    count_defects += 1;
                    circle(crop_image, farPoint, 1, Scalar(0, 0, 255), -1);
                }
                // Draw line is connecting start and end points (for a better visualization)
                line(crop_image, startPoint, endPoint, Scalar(0, 255, 0), 2);
            }
            // This will display the number of fingers detected
            putText(frame, format("Fingers: %d", count_defects + 1), Point(50, 50),
FONT_HERSHEY_SIMPLEX, 1, Scalar(255, 255, 255), 2);


            // Performs the actions based on number of defects
            double current_time = (double)getTickCount() / getTickFrequency();
            if (current_time - last_action_time > action_delay) {
                if (count_defects == 0) {
                    open_browser();
                } else if (count_defects == 1) {
                    open_calculator();
                } else if (count_defects == 2) {
                    open_spotify();
                } else if (count_defects == 3) {
                    open_task_manager();
                }
                last_action_time = current_time; // Update last action time to prevent rapid actions
            }
        } catch (const exception& e) {
            cerr << "Error: " << e.what() << endl;
            }}
        // Show the required images
        imshow("Gesture", frame);
        Mat all_image;
        hconcat(drawing, crop_image, all_image);
        imshow("Contours", all_image);


        // Close the camera if 'Escape' Key is pressed
        if (waitKey(1) == 27) break;
    }
    return 0;
}
```

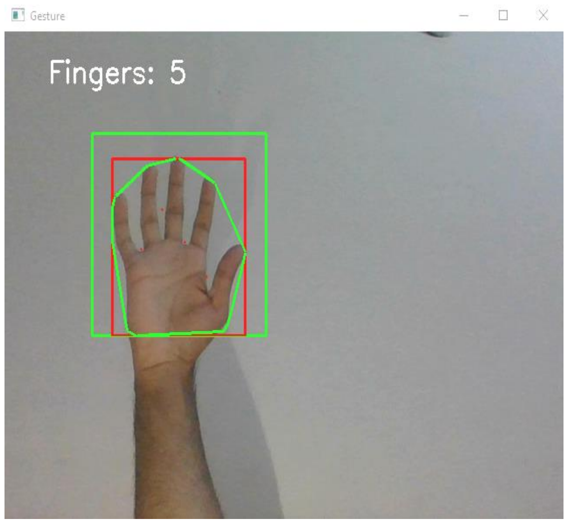
Project Images:











CONCLUSION:

This project presents a foundation for a C++ gesture recognition system with the potential to revolutionize human-computer interaction. The application offers a hands-free and intuitive user experience by enabling control through simple hand gestures. By leveraging computer vision techniques and user-defined commands, the system paves the way for a more natural and personalized way to interact with our computers.