DLIP : LAB2

Author : HanMinung

Student ID : 21800773

School of Mechanical and Control Engineering

LAB : Facial Temperature Measurement with IR images

Date : 2023.04.04

Github : https://github.com/HanMinung/DLIP

1. Purpose

       The purpose of this experiment was to estimate a person's body temperature using infrared (IR) imaging in the C++ openCV environment. This was achieved by capturing an IR image of the person using a thermal camera, converting the image to the HSV color space, extracting the intensity channel, and calculating the average intensity value of the image. Finally, a mapping equation was used to map the average intensity value to a temperature value, providing an estimate of the person's body temperature. This experiment aimed to demonstrate the use of openCV and IR imaging for non-contact temperature measurements and showcase the potential applications of this image processing technology.

2. Procedure

Accurate preprocessing is essential for achieving accurate image post-processing. In this experiment, the original IR image was split into three channels - H (hue), S (saturation), and V (intensity) - and only the V channel was extracted as a grayscale image. A thresholding process was then applied to the HSV image to identify a contour using 'inrange' function. However, even after applying inrange, there was a risk of disconnection in some areas of the face, such as the nose, due to varying contrast levels. Therefore, to address this issue, a dilation operation (Morphology) was performed twice to ensure that the contour of the object of interest was complete and well-defined for accurate temperature estimation. The processing flow chart of this experiment is as follows :
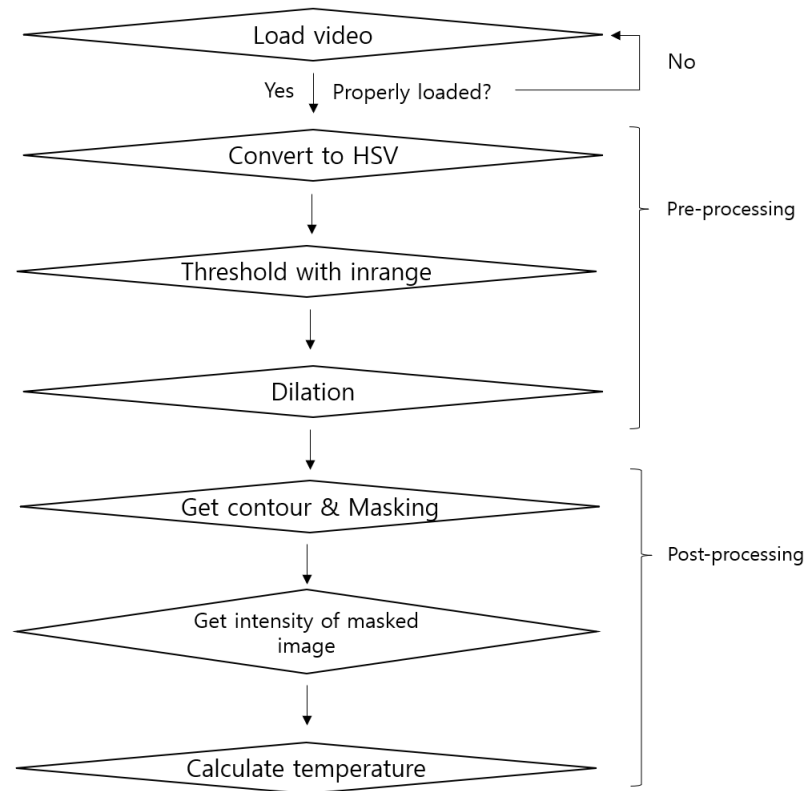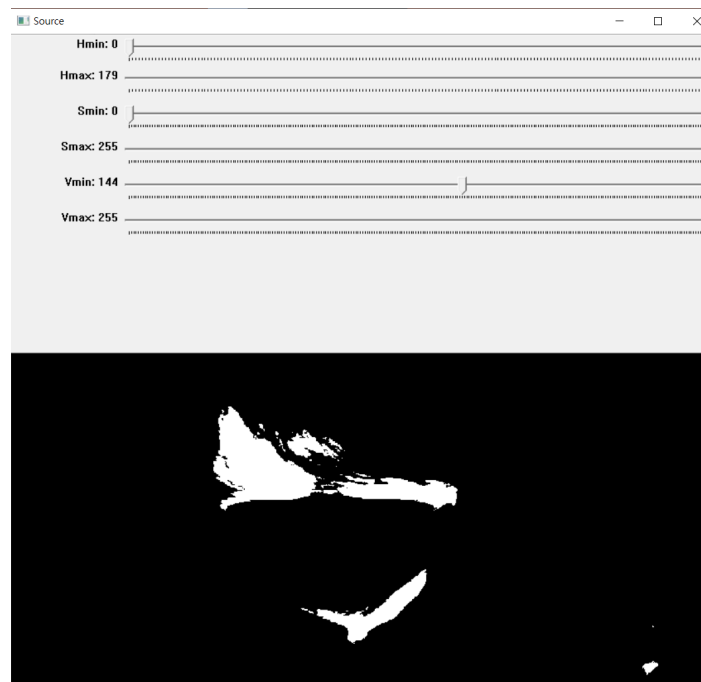

3. Preprocessing

3.1. Proper range of HSV

I found proper range of `[hmin, hmax, smin, smax, vmin, vmax]` to get the binary image which is applied 'inrange' function. Proper value of each components that I found with that technique is as follows :

$$[hmin, hmax, smin, smax, vmin, vmax] = [0, 179, 0, 255, 144, 255]$$

3.2. Preprocessing

Even after applying inrange, there was a risk of disconnection in some areas of the face, such as the nose, due to varying contrast levels. It was determined that applying a morphology technique such as dilation to make the white parts of the binarized image smoothly connect is necessary.

Binary image	Dilated image

3.3. Split to H, S, V

To extract the intensity value for body temperature measurement, the HSV scale image must be split into H, S, and V forms, and the grayscale image of V must be extracted. The resulting image of the process is as follows :

3.4. Getting contour & Masking

In the image which is binarized with the process by applying 'inrange' function, contours of one's face can be extracted. Later in the experiment, a `bitwise_or` operation was performed on the contour image that includes the face, and the black background image with same size. This resulted in a single binary image where all parts except the face area were blacked out. To accurately perform this process, it was essential to extract only the intensity values corresponding to the face region by performing a bitwise_and operation with the grayscale image. Accurate contouring was also crucial in ensuring that the resulting binary image was correctly aligned with the face region. Additionally, it was necessary to confirm the accuracy of the result through visual inspection and analysis to ensure that the temperature estimation was based on a precise and reliable contour of the face. The following results are as follows :

Contouring	
Mask	
Masking result	

3.5. Reshaping & Sorting

Based on the masked result, intensity and the following temperature can be calculated by reshaping masking result **(3.4. Getting contour & Masking)**. The mapping equation that I used is as follows :

```
// Sorting
graySort = maskResult.reshape(0, 1);
cv::sort(graySort, graySort, SORT_DESCENDING);

// Intensity to temperature
float PIX2DEG(float val) {

    return (float)round(15.0 * val / 255.0 + 25.0);
}
```

4. Result video

Link : https://youtu.be/FIHCVJwhP5s

Github : https://github.com/HanMinung/DLIP

5. Appendix

- main statement

```
void main() {

    VideoCapture cap("IR_DEMO_cut.avi");

    cap.set(CAP_PROP_FRAME_WIDTH, videoWidth);
    cap.set(CAP_PROP_FRAME_HEIGHT, videoHeight);

    Mat faceMask = cv::Mat::zeros(videoWidth, videoHeight, CV_8UC1);

    if (!cap.isOpened())    printf("Video Loading Failed ...!");

    while (true) {

        cap >> src;
        preProcessing();

        findContours(dstMorph, contours, hierarchy, RETR_EXTERNAL,
CHAIN_APPROX_SIMPLE);

        processContour(src, contours, hsvGray, "area", criticalMin);

        printImg("Src video", src);
```

```cpp
            if (waitKey(1) == 27)
                break;
    }
}
```

- Function : Process contour

```cpp
void processContour(cv::Mat& image,
std::vector<std::vector<cv::Point>>& contours , cv::Mat& hsvV,string
mode, int criticalMin) {

    Mat contourMask = Mat::zeros(image.size(), CV_8UC1);
    Mat faceMask = Mat::zeros(image.size(), CV_8UC1);
    Mat maskResult = Mat::zeros(image.size(), CV_8UC1);
    Mat prtGray = Mat::zeros(image.size(), CV_8UC1);
    Mat graySort;

    float maxGray = 0.0, sumGray = 0.0, avgGray = 0.0;
    int cnt = 0;

    if (mode == "area") {

        int maxArea = 0;
        int maxIdx = 0;

        for (int Idx = 0; Idx < contours.size(); Idx++) {

            int area = cv::contourArea(contours[Idx]);

            if (maxArea < area) {

                maxArea = area;
                maxIdx = Idx;
            }
        }

        if (maxArea > criticalMin) {

            cv::Rect BoundingBox = cv::boundingRect(contours[maxIdx]);
            cv::Moments mu = cv::moments(contours[maxIdx], false);
            cv::Point centroid = cv::Point(mu.m10 / mu.m00, mu.m01 /
mu.m00);
            cv::drawContours(image, contours, maxIdx, cv::Scalar(0, 0,
255), 2);
            cv::rectangle(image, BoundingBox, cv::Scalar(0, 0, 255),
2);

            drawContours(contourMask, contours, maxIdx, Scalar(255),
FILLED);
            bitwise_or(faceMask, contourMask, faceMask);
```

```cpp
            bitwise_and(faceMask, hsvV, maskResult);

            hsvV.copyTo(prtGray);

            graySort = maskResult.reshape(0, 1);
            cv::sort(graySort, graySort, SORT_DESCENDING);

            for (int idx = 0; idx < graySort.cols * 0.01; idx++) {

                if (graySort.at<uchar>(0, idx) != 0) {

                    if (maxGray <= graySort.at<uchar>(0, idx))  maxGray
= graySort.at<uchar>(0, idx);

                    sumGray += graySort.at<uchar>(0, idx);

                    cnt++;
                }
            }

            if (cnt != 0)   avgGray = (float)(sumGray / (float)cnt);

            putText(image, "Max temp : " +
to_string(int(PIX2DEG(graySort.at<uchar>(0, 0)))) + "  AVG temp : " +
to_string(int(PIX2DEG(avgGray))), Point(5,25), FONT_HERSHEY_COMPLEX,
0.7, Scalar(0,0,255));

            if (PIX2DEG(avgGray) >= 38.0)   putText(image, "WARNING !",
Point(5, 80), FONT_HERSHEY_COMPLEX, 1.5,                   Scalar(0,
0, 255));
        }
    }


    if (mode == "length") {

        for (int Idx = 0; Idx < contours.size(); Idx++) {

            int area = cv::arcLength(contours[Idx], true);

            if (area > criticalMin) {

                cv::Rect BoundingBox = cv::boundingRect(contours[Idx]);
                cv::rectangle(image, BoundingBox, cv::Scalar(0, 0,
255), 2);
                cv::Moments mu = cv::moments(contours[Idx], false);
                cv::Point centroid = cv::Point(mu.m10 / mu.m00 - 50,
mu.m01 / mu.m00 - 50);
                cv::drawContours(image, contours, Idx, cv::Scalar(0, 0,
255), 2);
            }
```

```
            }
        }
}
```