Capstone Design

지도교수 김윤관

가시광선과 적외선 영역에서 영상처리를 이용한 화재 감지 시스템 연구

김영수

권지헌

한지웅

홍단비

< 목 차 >

1. 서 론

1) 개발 동기

정보통신 전자 공학부생으로서 학사를 졸업하기 전에 하나의 시스템을 설계 해보고 싶었다. 학사과정에서 배운 모든 지식들을 단방향으로 습득하기만 하고, 지식들을 응용하여 나만의 지식으로 만들어 보지는 못하였다. 따라서, 이번 Capstone Design 교과목을 기회로 남들에게 도움이 될 수 있는 [1]Embedded System을 설계 하고자 한다.

2) 개발 목적

하나의 Embedded System을 만들기 위해서는 필요한 배경 지식이 매우 많다. 하드웨어부터 소프트웨어까지 모든 지식을 갖추기엔 학사과정에서는 힘들다. 따라서 우리는 하드웨어 플랫폼은 이미 구현되어 있는 Raspberry Pi를 사용하였다.

Raspberry Pi로 어떤 기능을 가지는 Embedded System을 만들까 하고 고민하던 중, 팀원이 운영하는 가게에 도움이 될 만한 기능을 가지는 시스템을 만들기로 하였다.

가게에서 필요한 Embedded System을 뭔지 고민한 끝에, 외부 침입 감지와 화재 감지를 할 수 있는 시스템을 만들기로 생각했는데, 최종적으로 우리는 가게 또는 사무실 같은 소규모의 공간에서 화재 감지를 할 수 있는 Embedded System을 만들기로 결정하였다.

3) 필요성

기존에 있는 화재 감지 시스템은 센서 기반에서부터 영상처리까지 다양하게 존재하고 있다. 센서 기반 화재 감지 시스템은 신뢰성이 낮아 독립적으로 잘 사용하지는 않는다.

---

[1] 특정 기능을 수행하는 규모가 있는 전자적 시스템으로 구성되는 컴퓨터 시스템

영상처리 기반 화재 감지 시스템은 가격이 비싸 가게와 같은 소규모 공간에서 사용하기엔 부담스러운면이 있다.

우리는 Raspberry Pi와 Pi Camera를 사용하여 프로젝트를 진행하여, 값싼 가격에 CCTV 역할을 하도록 웹스트리밍 서비스와 특정 기능인 화재 감지를 동시에 제공하고자 한다.

나아가 현재 사용되고 있는 화재 감지 시스템보다 신뢰성이 높은 시스템을 만드는데 목적을 둔다.

4) 차별성

영상처리 기반의 화재 감지는 크게 열영상을 이용하는 방법과 칼라영상을 이용하는 방법으로 분류한다. 열영상을 이용하는 방법 사물의 온도가 높고 낮음이 영상의 명도 정보로 반영되는 적외선 열영상에서 높은 명도값을 갖는 영역을 화재로 분류한다. 반면에 컬러영상을 이용하는 방법은 칼라영상에서 사물의 색 정보를 분석하여 화재나 연기와 동일한 컬러특성을 갖는 영역을 화재로 분류한다.

본 과제에서는 일반 카메라의 CCD 센서를 통해 가시광선 영역에서 컬러영상과 적외선 영역의 열영상을 분리, 획득하고 컬러영상과 열영상에서 얻어지는 이미지 데이터들의 관계, 차이를 이용하여 화재를 감지 하는데 신뢰성이 높은 영상처리 기법들을 설계하는데 목적이 있다.

다음은 가시광선 영역에서 찍은 사진과 적외선 영역에서 찍은 사진을 나타낸 그림이다.

< 그림 : 가시광선 영역과 적외선 영역에서의 이미지 >

이처럼 가시광선 영역에서 얻는 이미지와 적외선 영역에서 얻는 이미지는 서로 다른 특징을 가지고 있다.

적외선과 가시광선 영역을 모두 이용한 시스템은 기존에 있는 컬러영상과 센서 기반의 화재 감지기술의 오작동을 줄이고, 화재 감지 이후 기반으로 신속한 네트워크 통신을 통해 담당자에게 알림으로서 신속한 후속조치가 이루어 질 수 있는 시스템을 개발하는 데 궁극적인 목표를 두고있다. 그리고 기존 센서 기반과 서로 독립된 컬러영상과 열영상을 기반으로한 화재감지 기술의 단점을 보완하고 화재 감지의 실효성을 높이기 위한 기술을 제안한다.

과제의 준비 기간과 팀원의 역량 및 개인별 할당량을 고려해서 CCD 센서를 통해 얻은 영상 데이터를 가시광선 영역과 적외선 영역으로 분리하는 과제를 수행하는 대신 일반 카메라와 적외선 카메라를 나눠 사용하고, Server-Client 모델을 이용하여 2개의 Client에서 영상 데이터를 Server가 받고, Server에서 이미지 프로세싱을 하도록 하였다.

본 학기 기간에는 이처럼 가상적으로 구현하고, 최종 과제는 향후과제에서 다루도록 한다.

5) 배경 지식

5-1) Raspberry Pi

라즈베리 파이(영어: Raspberry Pi)는 영국의 라즈베리 파이 재단이 학교에서 기초 컴퓨터 과학 교육을 증진시키기 위해 만든 싱글 보드 컴퓨터이다. 라즈베리 파이는 그래픽 성능이 뛰어나면서도 저렴한 가격 특징을 갖고 있다.

라즈베리 파이는 브로드컴의 BCM2835 단일 칩 시스템을 사용하며, 이 칩에는 ARM1176JZF-S 700 MHz 프로세서, 비디오코어 IV GPU 와 256 메가바이트 RAM 이 들어 있다. 라즈베리 파이는 하드 디스크 드라이브나 솔리드 스테이트 드라이브를 내장하고 있지 않으며, SD 카드를 외부 기억장치로 사용한다.

라즈베리파이는 인터넷, 1080p 영상재생, 웹서버, FTP 서버, 기기컨트로(임베디드) 등 다양한 용도로 사용될 수 있다. 특히 라즈베리파이는 영상만 따로 처리할 수 있는 GPU 라는 것이 존재하여 고해상도의 영상도 무리없이 사용할 수 있다.

5-2) Infrared Ray

적외선은 가시광선의 적색보다 파장이 긴 영역을 말한다. 적외선은 일반적으로 열선으로 많이 알려져 있으나, 열선이 적외선에 포함될 뿐 모든 적외선이 열선은 아니다. 적외선 사진에서 주로 말하는 적외선이란 앞에서 말한 적외선 영역 중 가시광선에 가까운 파장을 근적외선이라 하고, 전자파에 가까운 파장을 원적외선이라 한다. 이미지 센서가 포착할 수 있는 영역이 근적외선 영역의 일부이기 때문에 나머지 영역의 적외선은 일반적인 디지털카메라로는 기록되지 않는다.

적외선 열 영상에서의 화재 검출은 물체나 화재영역의 온도가 열 영상에서 명도 정보로 반영되어 이를 분석하면 되므로 컬러영상에서의 화재 검출 보다 알고리즘의 처리량과 복잡성을 줄일 수 있는 이점이 있다.

5-4) OpenCV

OpenCV(Open Computer Vision)은 오픈 소스 컴퓨터 비전 C 라이브러리이다. 원래는 인텔이 개발하였다. 윈도, 리눅스 등의 여러 플랫폼에서 사용할 수 있다. 실시간 이미지 프로세싱에 중점을 둔 라이브러리이다. 인텔 CPU에서 사용되는 경우 속도의 향상을 볼 수 있는 Intel Performance Primitives (IPP)를 지원한다.

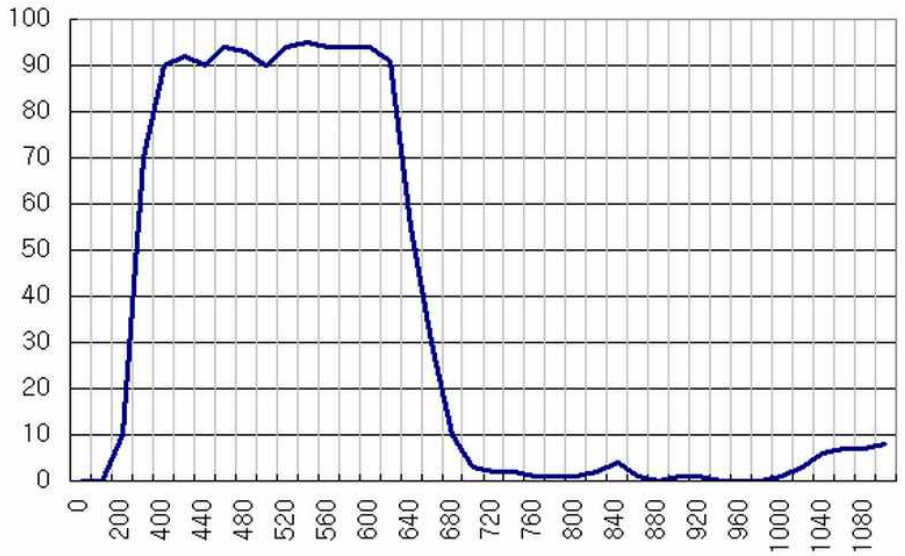
2. 가시광선 영역에서 화재 검출

가시광선 영역에서 화재를 검출하기 위해서 적외선 차단 필터가 부착된 일반 디지털 카메라를 사용한다. 디지털 카메라는 부가장치를 사용하지 않더라도 이미지 센서의 분광감광영역(Spectral Sensitivity Range)이 대략 380~1200nm이며, 이것은 근적외선 영역을 일부 포함하는 수치이다. 가시광선(대략 400~700nm)의 영역을 촬영하기 위해서는 이미지 센서의 앞부분에 적외선 차단 필터를 사용하여 700nm를 초과하는 광선을 차단한다. 이후 가시광선 영역에서 사용되는 카메라는 칼라 카메라라고 명칭한다.

본 과제를 위해서 적외선 차단 필터를 사용하기도 하지만, 일반적으로 좋은 품질의 디

지털 사진을 얻기 위해 가시적으로 확인할 수 없는 신호를 제거하는데도 사용한다.

다음 사진은 적외선 차단 필터를 적용한 투과율 그래프인데 700nm를 초과하는 광선을 차단하는 것을 확인할 수 있다.



< 그림 : 적외선 차단 필터를 적용한 투과율 그래프 >

1) RGB 이미지 데이터 형식

RGB 가산혼합은 빛의 삼원색을 이용하여 색을 표현하는 방식이다. RED, GREEN, BLUE 세종류의 광원을 이용하여 색을 혼합하며 색을 섞을수록 밝아지기 때문에 '가산혼합'이라고 한다. 가시광선 영역에서는 RGB 이미지 데이터 포멧으로 이미지 데이터를 받고 프로세싱을 한다. RGB 포멧으로 화재의 컬러 모델인 붉은 계열의 색상 구별하는데 집중적으로 예비 후보 영역을 검출 할 수 있다.

2) 영상 잡음 제거

디지털 영상에서 소금과 후추 노이즈(Salt-and-Pepper Noise)와 같은 임펄스(Impulse) 잡음을 제거하기 위해 Blurring 처리를 하였다. 영상의 Edge를 잘 보존하기 위해서 단순

Blurring으로 처리하였다.

다음 그림은 OpenCV 라이브러리의 cvSmooth 함수에서 단순 블러링으로 이웃 픽셀간의 평균을 도출하는 방법을 사용하여 노이즈를 제거한 그림이다.

< 그림 : 영상 잡음 제거 전후 사진 >

3) 정적인 후보 화재 영역 검출

1차 화재 후보 영역을 검출하기 위해 문턱값 처리를 한다. 불의 RGB 컬러 모델을 기준으로 픽셀의 RGB 데이터 값 불의 RGB 컬러 모델에 속하지 않는다면 모두 검은색으로 처리해준다. 즉, 가시광선 영역에서는 RGB 포멧으로 불의 색상 위주로 화재 후보 영역을 검출한다.

환경에 따라 불의 RGB 모델이 달라질 수 있음을 염두해 둔다.

3-1) 문턱값 처리

< 그림 : 문턱값 처리된 이미지 >

불의 RGB 모델에 따라 후보 영역들을 검출 했지만, 불의 RGB와 비슷한 빨간색 계열의 물품이나 옷, 형광등 등도 불과 함께 검출이 된다. 이와 같은 영역들은 정적인 후보 화재 영역이라고 한다.

4) 동적인 후보 화재 영역 검출

불의 RGB 모델로 문턱값 처리를 하여 검출된 후보 화재 영역은 불이 아닌 후보 영역들과 구별하기 위해서는 또 다른 처리 과정이 필요하다.

불이 아닌 후보 영역들은 연속되는 영상에서 명도 세기 또는 색상 변화량이 적은 정적인 특성을 가지는 반면, 불의 경우는 화염의 움직임이나 확산 현상에 의해 연속되는 영상에서 명도 세기 또는 색상 변화량이 큰 동적인 특성을 가진다. 이러한 물체의 명도 변화를 영상의 시공간 영역에서 변화량을 구하여 수치화하고, 이를 정해진 기준치로 분류하면 정적인 오류 요소의 화재 오 검출을 낮출 수 있으며, 화재 검출의 신뢰성 또한 높일 수 있는 장점이 있다.

동적인 후보 화재 영역을 검출하기 위해서 필요한 알고리즘은 다음과 같다.

4-1) 라벨링 알고리즘

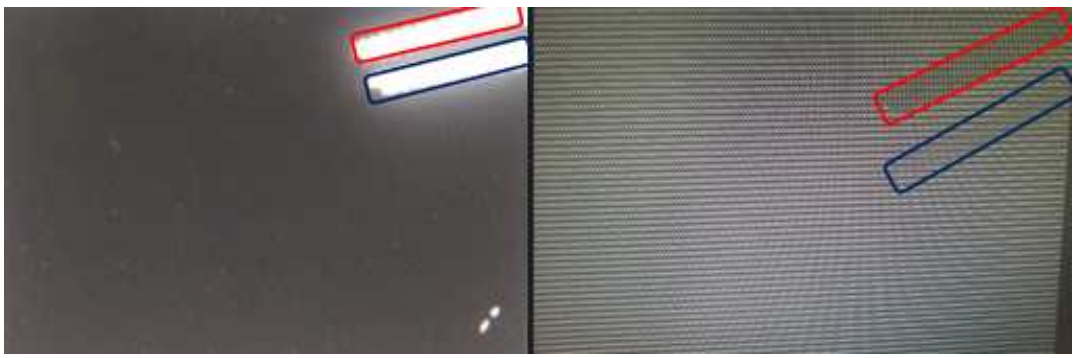
각 후보 영역들에 대해서 시공간 영역에서 움직임의 변화량을 측정해야 한다. 변화량

을 알기 위해서는 후보 영역들을 객체화 시킬 필요성이 있다. 이를테면 프로세싱 할 수 있는 기준이 한 개의 영상 프레임 단위가 아닌 하나의 객체 단위로 바꿀 수 있다. 각 후보 영역 하나하나를 하나의 객체라고 생각하면 연속되는 영상 프레임에서 후보 영역들간의 변화량을 측정 할 수 있다.

본 과제에서 후보 영역들을 객체화 시키기 위해서 라벨링 알고리즘을 사용한다. 라벨링은 인접한 화소에 같은 번호(Label)를 붙여서 그룹을 짓는 일을 말한다.

다음 그림은 후보 화재 영역인 형광등에 대해 문턱값 처리를 하고 라벨링 알고리즘을 적용한 사진이다. 왼쪽 사진은 원본 사진에 대해 문턱값을 처리한 사진이고, 오른쪽 사진은 라벨링 알고리즘을 실시하여 한 프레임의 픽셀단위마다 번호를 부여한 그림이다.

객체에 대해서는 순서대로 1부터 증가하는 값을 가지고 객체가 아닌 배경에서는 0인 번호를 가지게 된다. 빨간색 테두리를 가지는 객체1은 숫자 1의 번호를 갖게 되고, 파란색 테두리를 가지는 객체2는 숫자 2의 번호를 갖게 된다. 따라서 숫자 1과 2를 이용해서 후보 영역들을 프로세싱 할 수 있다.



< 그림 : 라벨링이 된 이미지 >

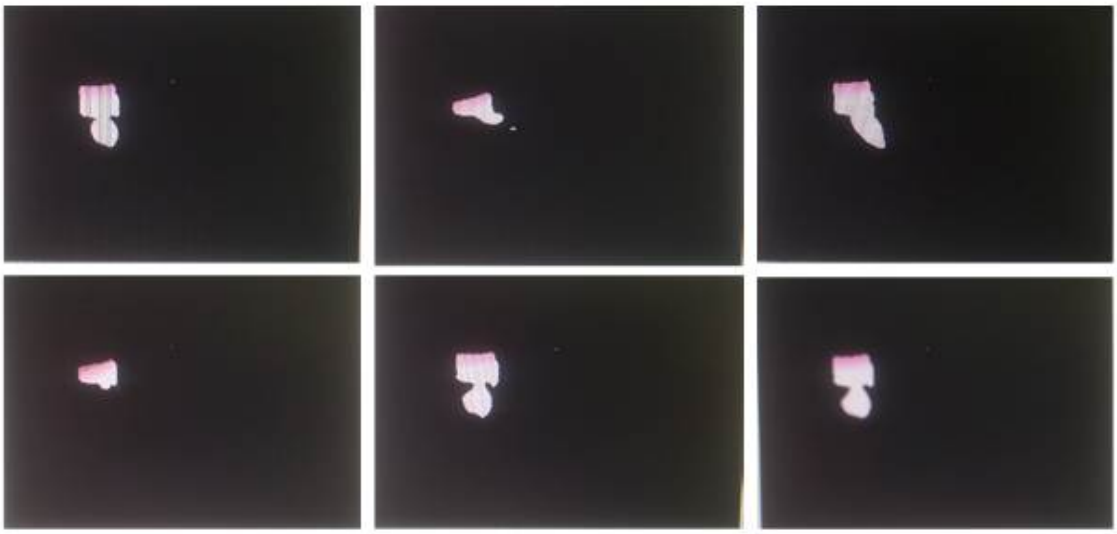
4-2) 후보 영역들의 움직임 측정

문턱값 처리를 통해 얻어낸 후보 화재영역들에 대해 동적인지 여부를 확인하기 위해서는 연속되는 프레임들의 정보를 통해 후보 화재영역들의 움직임을 측정한다. 각 후보 화재영역들에 대해 움직임을 측정하기 위해 다음과 같은 과정을 거친다.

문턱값 처리 후 하나의 후보 영역이 생기면, 해당 프레임을 기준으로 연속되는 10개의 프레임에 대해 후보 영역들에 대한 중점을 모두 저장하고, 연속되는 프레임마다 중점의 변화량을 측정한다. 10개의 프레임에 대해서 중점의 변화량의 평균을 내면 후보 화재영역에 대한 움직임을 측정할 수 있다.

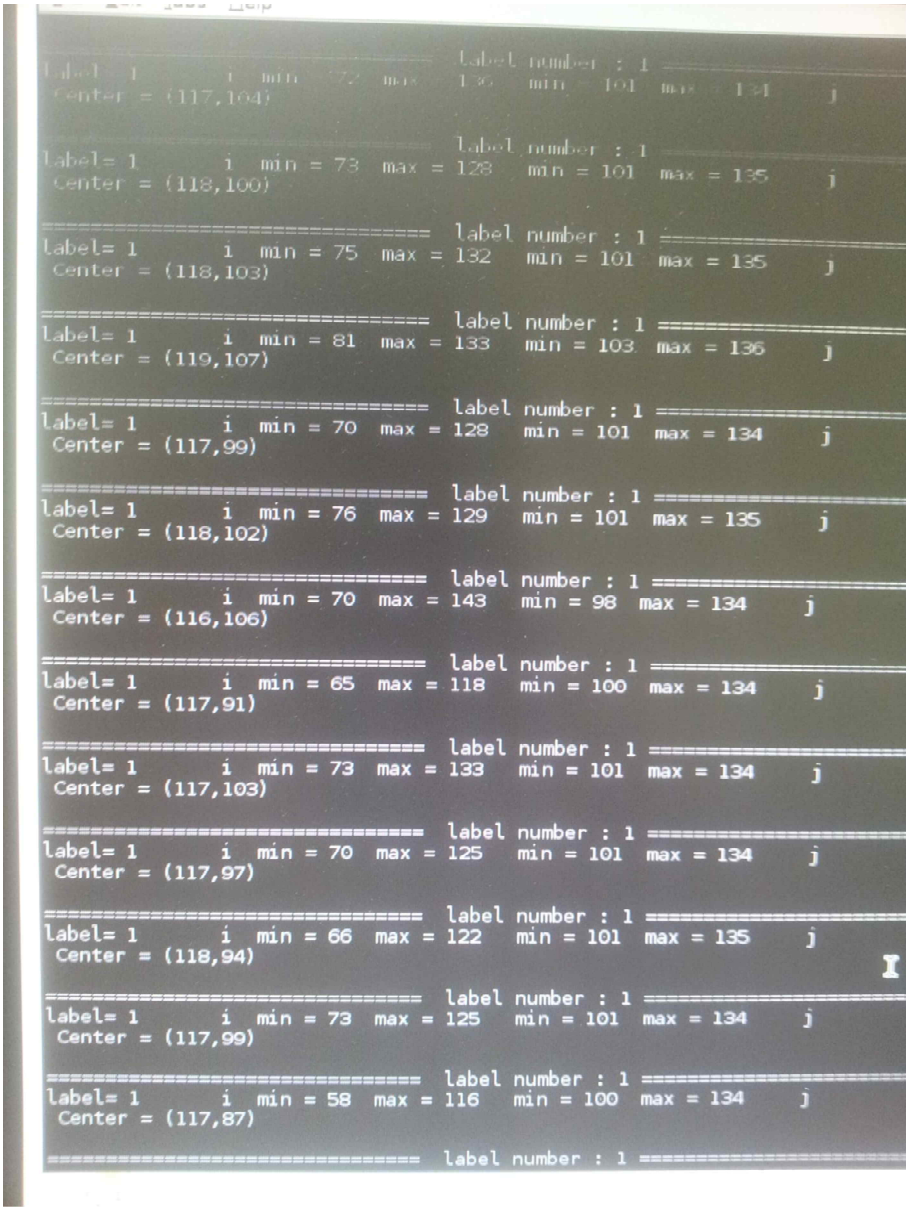
여기서 주의 해야 할 점은 연속되는 프레임에서 중점의 변화량을 측정하기 전 한가지 전제 조건이 있다. 외부요인을 배제하기 위해서 연속되는 프레임간의 라벨 수는 같아야 한다. 예를 들어 사람이 라벨링 된 영역을 지나다니면, 하나의 라벨이 둘 또는 그 이상으로 라벨이 더 생기는 현상이 발생 할 수도 있다. 이는 동적인 움직임을 판단하기 위한 기준을 벗어나기 때문이다.

다음 그림은 연속된 프레임 속에서의 불의 동적인 움직임을 나타낸 그림이다.



< 그림 : 연속된 프레임 속의 불 화면 >

다음 그림은 동적인 불꽃의 중점을 추적하기 위해 디버깅한 화면이다. Label =1 을 통해 하나의 라벨이 연속적으로 생기는 것을 확인할 수 있고, x,y 좌표의 min, max를 구하여 중점(Center)를 얻는 것을 확인할 수 있다. 시간 변화량에 따라 중점의 좌표가 변화하는 것을 알 수 있다.



< 그림 : 동적인 움직임을 가지는 불 이미지 >

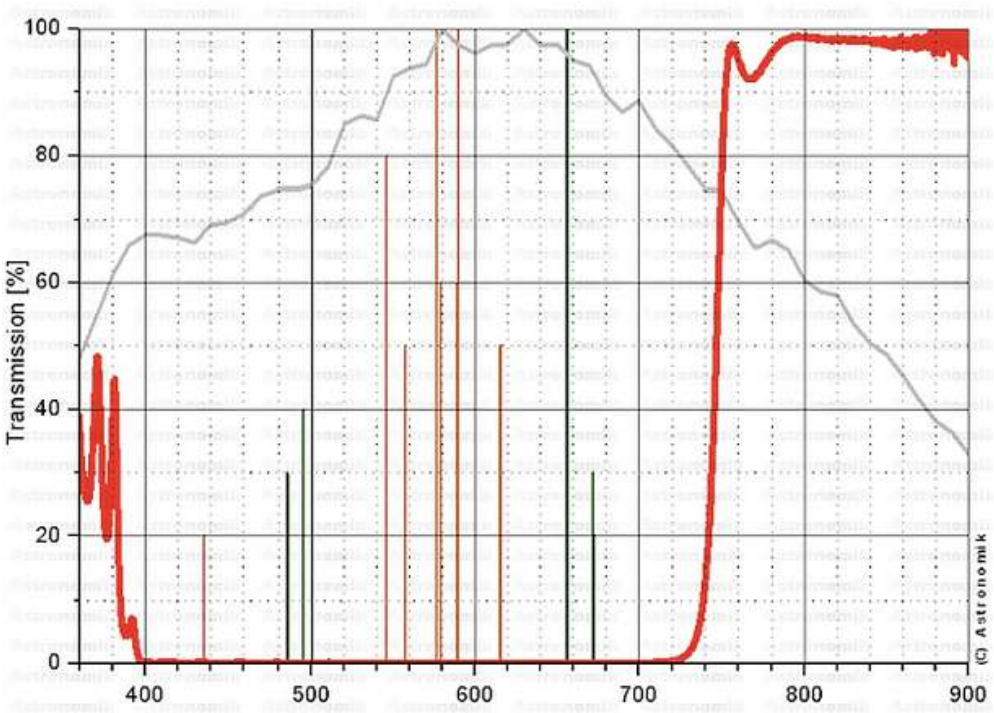
3. 적외선 영역에서 화재 검출

적외선 영역에서 화재 검출을 하기 위해서는 적외선 카메라를 사용해야 하지만, 개발환경 특성상 IR filter를 제거한 카메라를 사용하였다. 이후 적외선 영역에서 사용되는 카메

라는 편의상 적외선 카메라라고 명칭한다. 그리고 가시광선 영역을 제거하기 위해 인화된 필름을 부착하였다.

< 그림 : 적외선 카메라와 인화된 필름(네거티브 필름) >

투과되는 광선의 범위를 살펴보면 다음 그림과 같다. (빨간색 실선)



< 그림 : 인화된 필름과 적외선 필터를 제거한 투과율에 대한 그래프 >

1) YCbCr 이미지 데이터 형식

YCbCr컬러공간은 디지털 비디오에서 널리 사용된다, 이 포맷에서, 휘도정보는 단일 성분 Y에 의해 표현되며, 칼라정보는 두 개의 칼라 차이 성분들 Cb와 Cr로서 저장된다. 성분 Cb는 청색 성분과 기준 값간의 차이이며, 성분 Cr은 적색 성분과 기준 값간의 차이이다. RGB영상을 YCbCr영상으로의 변환은 아래의 식으로 표현 할 수 있다.

$$Y = 0.257*R+0.504*G+0.098*B+16;$$
$$Cb = -0.148*R-0.291*G+0.439*B+128;$$
$$Cr = 0.439*R-0.368*G-0.071*B+128;$$

YCbCr에서 Y는 휘도(빛의 양)이고,Cb와 Cr은 색차(크로마)를 나타낸다. 이 방식은 RGB보다 색상의 분리 및 전달 효과는 약하지만, 적은 데이터로 보다 많은 색상을 나타낼 수 있는 장점을 가진다.

앞에서 언급한 바와 같이 적외선 영역에서는 밝기에 민감하게 이미지화 된다. 따라서 밝기에 더 초점을 맞춘 이미지 데이터 형식인 YCbCr로 변환하여 사용하였다.

2) 영상 잡음 제거

가시광선 영역에서와 동일함으로 생략하도록 한다.

3) 정적인 후보 화재 영역 검출

적외선 영역에서 마찬가지로 정적인 영역을 검출해 후보 화재영역을 선정한다. 문턱값 처리를 하는데 가시광선과는 달리 적외선 영역에서는 YCbCr 이미지 포멧을 사용하기 때문에 문턱값은 Y에 맞춰 처리한다.

다음 그림은 적외선 영역에서 문턱값 처리한 그림인데, 형광등의 불빛은 문턱값에 의해 걸러진 것을 확인 할 수 있다.

< 그림 : 문턱값 처리된 이미지 >

이처럼 적외선 영역에서는 작은 불빛이나 빛이 없는 빨간 색상의 계열들은 문턱값 처리로 걸러 낼 수 있다.

이렇게 가시광선 영역과 적외선 영역에서 다른 기준으로 후보 화재영역을 선정함으로써 신뢰성을 좀 더 높일 수 있다.

4) 동적인 후보 화재 영역 검출

가시광선 영역에서와 동일함으로 생략하도록 한다.

4. 최종 화재 검출

적외선, 가시광선 영역에서 얻은 이미지 데이터를 프레임 단위로 서버에서 전송 받아, 서버에서는 최종적으로 적외선 영역에서 받은 이미지 데이터와 가시광선 영역에서 받은 이미지 데이터를 프로세싱 하여 화재를 검출한다.

이런 절차를 거쳐 독립적인 플랫폼을 가지는 일반 카메라에서의 영상처리 보다 신뢰성을 높일 수 있는 화재 감지 시스템을 만들 수 있다.

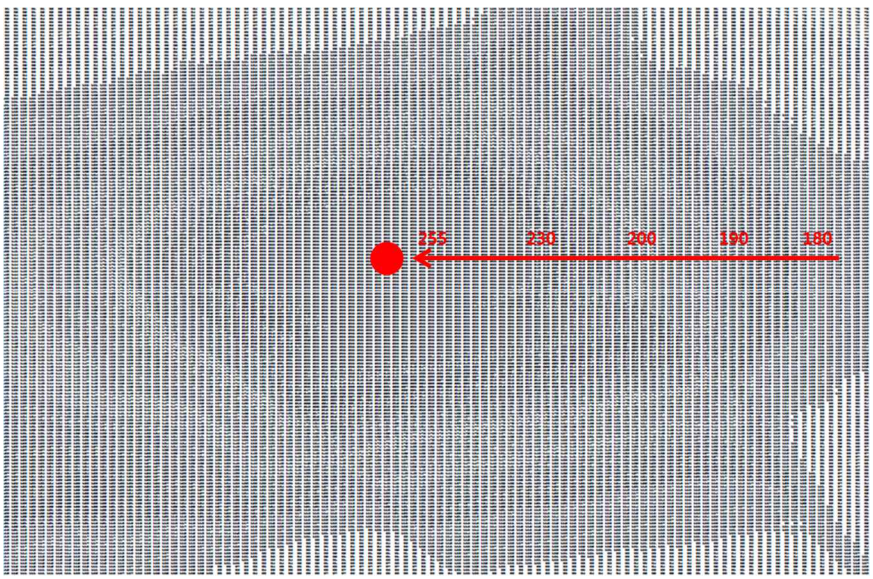
1) 두 영역에서 얻은 이미지 데이터 비교

Client들로부터 가시광선 영역의 프레임과 적외선 영역의 프레임을 받으면, 화재 영역이 서로 일치하는지 확인한다. 이 과정을 통해 가시광선 또는 적외선 하나의 영역에서만 감지된 후보 영역들을 제외 시킬 수 있다.

< 그림 : 각 영역에서의 중점 일치 확인 >

최종적으로 불을 감지하는 방법은 불의 모델인지 확인한다. 실제 불을 RGB 또는 YcbCr 데이터 값으로 뽑아보면 다음과 같은 모델을 가진다. 다음 그림은 실제 Y의 값 메모장에 출력한 것으로 색깔이 진해질수록 255에 가깝다는 것을 뜻한다. 즉, 데이터 값을 축소한 그림이다.

불은 Core로 향할수록 Y값이 점진적으로 증가하는 것으로 나타난다. 따라서 최종적으로 불의 모델의 특성을 이용하여 최종적으로 화재 후보영역을 검출할 수 있다.

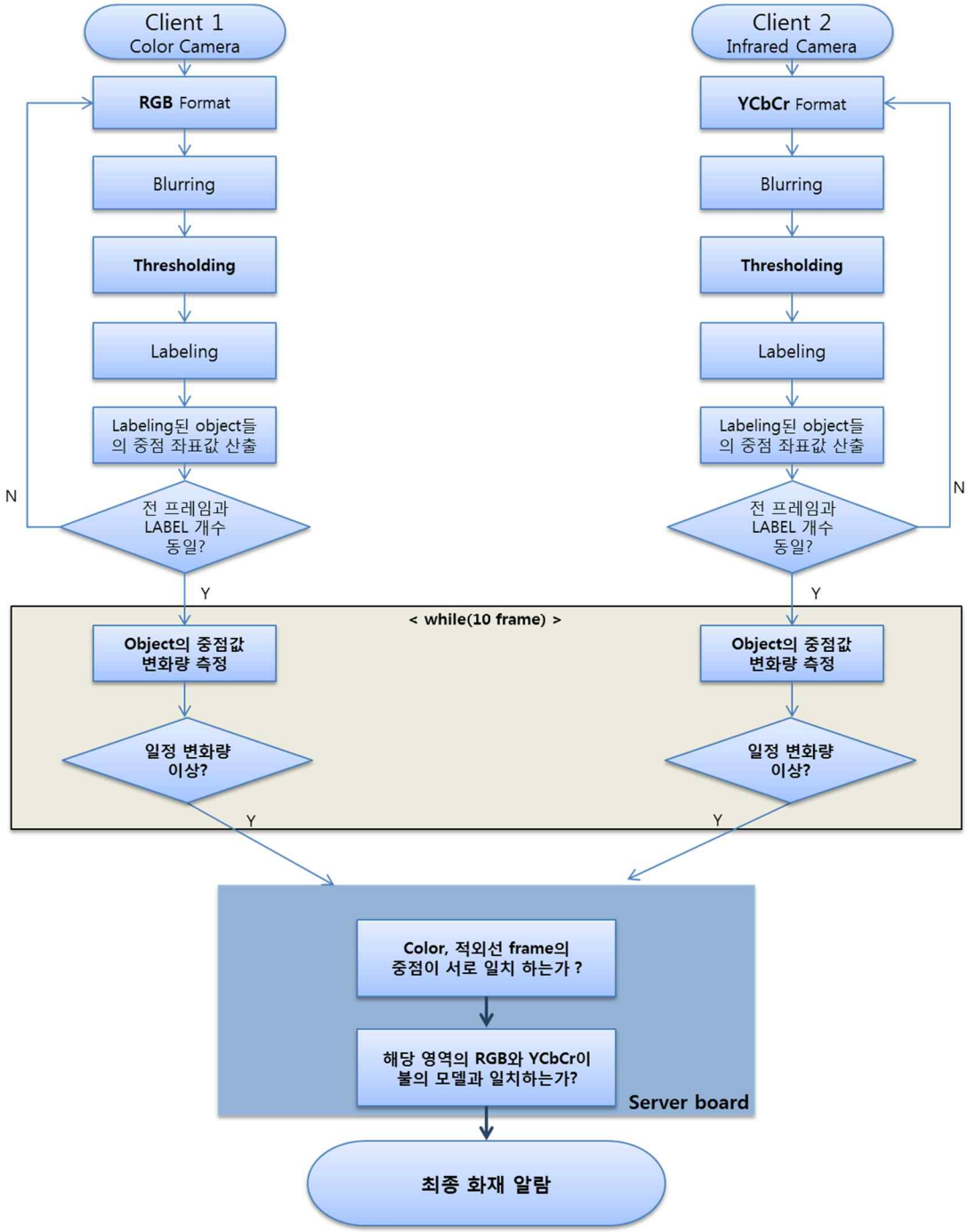


< 그림 : 데이터 값을 통한 불의 모델 >

그리고 불꽃만의 RGB 비율을 가지고 있어서, RGB 포멧에서도 불의 모델을 확인 할 수 있다.

2) 전체 시스템 Flow Chart

가상적으로 Client-Server 모델로 구현한 시스템의 흐름은 다음과 같다.



< 그림 : 전체 시스템 Flow Chart >

[단계 1] 각 라즈베리파이 보드의 라즈베리파이 전용 카메라(일반, 적외선)를 통하여 영상 획득.

- 2592 X 1944(정적 이미지), 1080p30, 720p60 및 640x480p 60/90 비디오.

[단계 2] 화재 후보 영역 판단 및 추출.

- Color camera는 RGB format, 적외선 camera는 YCbCr format을 기반으로 processing한다.

- blurring(전처리 과정)을 통하여 영상의 잡음을 제거한다.
- 1차 화제 후보영역 검출, (Thresholding을 통하여 (RGB는 불꽃 색의 비율에 초점을 맞추고, YCbCr은 불꽃의 밝기에 초점을 맞춘다.) 정적인 측면에서 화재라고 의심이 되는 부분을 검출한다.
- 정적인 측면에서 검출된 1차 화제 후보영역을 Labeling을 통해 영역간 구분을 짓도록 한다.
- 각 object(후보영역)의 중점 좌표 값을 구하여 2차 화제 후보 검출의 기준을 얻는다.
- 2차 화제 후보영역 검출, (후의 10 frame동안 frame간 label 개수와 중점 좌표 값의 변화량을 이용.) 동적인 측면에서 화제라고 의심이 되는 부분을 검출한다.
- Color, 적외선 camera 중 후보 영상이 검출되면 동기화를 맞춰 서버 보드에 전송할 수 있도록 한다.

[단계 3] 후보 영역을 TCP/IP 통신을 통해 서버로 전송.

[단계 4] 수신한 칼라 및 적외선 영상을 비교 분석하여 화재 / 비 화재를 판단.
- 각 frame 간 중점이 일정한 거리에 있는지 판단한다.
- 서로 RGB, YCbCr format으로 각각 변환하였을 경우 어느정도 일치하는지 판단하여 화재로 감지한다..

[단계 5] 최종적으로 화재로 판단된 경우 빠른 조치를 위해 메시지를 전송.

5. 결론

현재 사용되고 있는 불 감지 시스템은 오검출률이 높아 신뢰성에 문제가 되고 있다. 따라서 본 과제는 가시광선 영역과 적외선 영역을 서로 다르게 영상 처리를 하여 화재 검출을 하는데 좀 더 신뢰성을 높이고자 하였다. 가시광선이나 적외선 영역에서 찍히는 이미지는 결국 사람이 보기 위해서 RGB를 갖는 색상 데이터로 표현되는데, 가시광선은 사람의 눈에 보이는 그대로 RGB로 옮겨 이미지화를 하지만, 적외선은 물체에 흐르는 열 파장의 세기를 감지하여 RGB로 이미지화 한다. 따라서 같은 사진에 대해 똑 같은 RGB 포멧을 가지지만, 다르게 표현된다. 우리는 이렇게 다르게 표현되는 것을 이용하였다. 가시광선 영역에서는 물체의 색상을 강조하여 표현이 되고, 적외선 영역에서는 물체의 빛을 강조하여 표현이 되었다.

화재 검출 알고리즘은 크게 3단계로 나눠 이루워졌다. 첫 번째, 불과 비슷한 색상과 밝기를 가지는 영역을 제외한 영역은 보두 제외시켰고, 두 번째, 랜덤한 움직임을 가지는 불의 특성을 이용하여 움직임의 변화를 관찰하였다. 세 번째, 최종적으로 가시광선 영역에서 인지한 영역과 적외선 영역에서 인지한 영역이 불의 모델과 적합한지를 확인하였다.

이런 과정을 통하면, 기존에 독립적인 가시광선 영역이나 적외선 영역을 가지는 플랫폼의 단점인 빨간 계열의 색상과 불과 비슷한 밝기를 가지는 햇빛, 형광등 등을 정확히 구별하기 어려운 점을 보완할 수 있다.

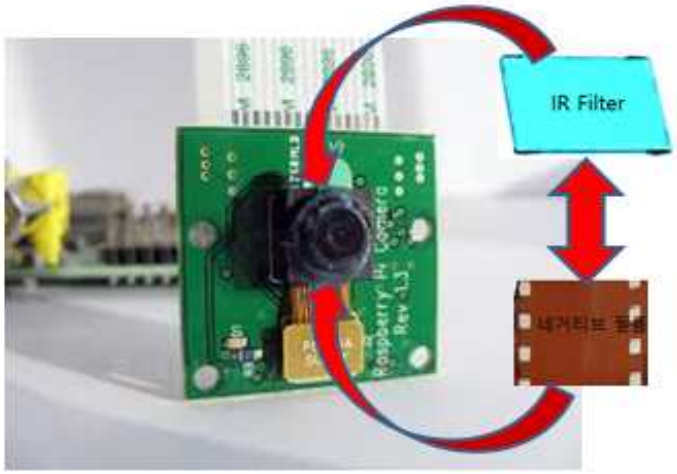
구현 정도는 동적인 영역을 감지하는 것과 네트워크 통신까지 되었으나, 디버깅을 하지않아 실험까지 진행하기엔 어려움이 있었다. 하지만, 알고리즘 부분적인 내용은 실험을 통하여 진행하였다.

부분적인 내용을 통해 불 감지 시스템을 만들 수 있다고 일반화시키는 어렵지만, 이러한 일련의 절차를 가지는 것에 대해서는 의미있다고 생각한다. 그리고 적외선과 가시광선 영역을 함께 사용하는 것은 실험을 통해 신뢰성을 가지는 것을 확인하였다.

향후 계획과 함께 좀 더 연구하고 동적인 움직임을 감지하는 알고리즘을 좀 더 보완하면 실생활에 적용할 수 있는 화재 감지 시스템을 만들 수 있다고 생각한다.

6. 향후 계획

본 과제는 가시광선 영역과 적외선 영역을 하나에 카메라에서 분리하지 못하여, Server-Client 모델을 사용하여 가시광선 영역과 적외선 영역을 마치 분리 한 것 처럼 프로세싱을 하였다. 하지만, 이는 가상적으로 구현한 것으로 현실성이 떨어진다. 우리는 이후에 실질적으로 사용할 수 있는 플랫폼을 만들어보기로 구상하였다.



< 그림 : 향후 플랫폼 상상도 >

센서가 감지하는 신호단에서 가시광선 영역과 적외선 영역을 분리하는 것은 어려움이 많을 것이라 예상이 되어, 센서 앞부분 필터를 통해서 시간차로 적외선 영역, 가시광선 영역을 나눠 이미지 데이터를 받을 수 있도록 한다. 필터가 교체되는 시점과 영상을 받아들이는 시점을 동기화 시키는게 문제가 될 수 있지만, 어느 정도 속도를 맞추면 가능할 수 있다고 생각한다.

7. 참고 문헌

[1] 趙宰暾, 영상처리를 이용한 화재감지기 검사방법, 학위논문

[2] 황준철, 컬러 영상처리 기반의 3-step 화재검출 알고리즘 연구, 학위논문

[3] 장복규, 주파수 분석 기반의 적외선 열영상 화재 검출 알고리즘에 관한 연구, 학위논문

[4] 한성수, 적외선 카메라를 이용한 손 인식 인터페이스에 관한 연구, 학위논문

[5] 열영상 인식 기술과 USN 기술을 응용한 화재감지 시스템 설계

[6] 장대웅, YCbCr 컬러 모델과 질감을 이용한 불 검출 기법에 대한 연구, 학위논문

[7] Fire detection using statistical color model in video sequences

[8] NG CHING HAU, Fire and smoke detection based on low-cost camera

8. 부록

Client

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv/cv.h"
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <unistd.h>
```

```
#define SIZE sizeof(struct sockaddr_in)

#define MAXX(x,y)     ((x)>(y)?(x):(y))
#define MINN(x,y)     ((x)<(y)?(x):(y))

#define CENTER(min,max)                    ((min)+(max)) / 2

#define WIDTH_C                  320
#define HEIGHT_C     240

#define MAX_LABEL   1000
#define EQ_INTV                  5

#define TH_B            200
#define TH_G            210
#define TH_R            220

#define TEST_P                   1
#define MAP_TEST     0


void catcher(int sig);
int newsockfd, sockfd;
struct sockaddr_in server = {AF_INET, 9009};

void wait(float seconds);
void tcp_client(void);

int glabel, glo_i, debugC,   i,j, g,b, m,n, p,q;
int label = 0;
int labeling[HEIGHT_C][WIDTH_C];
int maxL,minL;
int min_axis[MAX_LABEL][2];
int max_axis[MAX_LABEL][2];

CvCapture *capture;
IplImage *img0;
IplImage *img1;

int is_data_ready = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void *streamClient(void *arg);
void quit(const char *msg, int retval);

void printvalue(FILE *rfp, FILE *gfp, FILE *bfp, IplImage *frame);
void Threshold(IplImage *img);
void Labeling(IplImage *img);
```

```
void frame_init(void);
void debugging(void);
void pre_image_set(IplImage *img);
void print_labeling(void);
void print_yellow_rect(IplImage *img);


int main(int argc, char ** argv)
{

                pthread_t thread_s;
                int key;
        server.sin_addr.s_addr = inet_addr("127.0.0.1");

        capture = cvCaptureFromCAM(0);
        if(!capture)    quit("cvCapture failed", 1);

                for(i=0;i<MAX_LABEL;i++)
                {
                        min_axis[i][0] = WIDTH_C-1;
                        min_axis[i][1] = HEIGHT_C-1;
                }

        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_WIDTH, WIDTH_C);
        cvSetCaptureProperty(capture, CV_CAP_PROP_FRAME_HEIGHT, HEIGHT_C);

                img0 = cvQueryFrame(capture);
                img1 = cvCreateImage(cvGetSize(img0), IPL_DEPTH_8U, 3);
                cvZero(img1);


                //      printf("img1->imageSize = %d₩n", img1->imageSize);
                //      printf("img1->widthStep = %d₩n", img1->widthStep);
                //      printf("img1->height = %d₩n", img1->height);
                //      printf("img1->width = %d₩n", img1->width);

                cvNamedWindow("Client", CV_WINDOW_AUTOSIZE);
                //cvNamedWindow("Unblurred", CV_WINDOW_AUTOSIZE);
                //cvNamedWindow("Client_Test", CV_WINDOW_AUTOSIZE);


                fprintf(stdout, "width: %d₩nheight: %d₩n₩n", img0->width, img0->height);
                fprintf(stdout, "Press 'q' to quit ₩n₩n");

                // Connect to Server
                //tcp_client();

                // Start Thread...
                //if(pthread_create(&thread_s, NULL, streamClient, NULL))       quit("pthread_create failed", 1);
```

```
		while(key!='q')
		{
			wait(0.1);
			img0 = cvQueryFrame(capture);
			if(!img0)	break;
			img0->origin = 0;
			//cvFlip(img0, img0, 1);
			cvCopy(img0, img1, NULL);

////////////////////////////////////////////////////////////////////
////////////		Image Processing		////////////////
////////////////////////////////////////////////////////////////////

			pthread_mutex_lock(&mutex);

			// Step 1 : blurring
			cvSmooth(img1, img1, CV_BLUR,3,0,0,0);

			// Step 2 : Threasholding
			Threshold(img1);

			// Step 3 : Labeling
			//pre_image_set(img1);

			//if(debugC==100 || debugC==200 || debugC==300 || debugC==400 || debugC == 500)
			Labeling(img1);
			//print_labeling();
			//print_yellow_rect(img1);

			debugC++;
////////////////////////////////////////////////////////////////////////////////
		//			pthread_mutex_lock(&mutex);
//				cvCvtColor(img0, img1, CV_BGR2RGB);

				is_data_ready = 1;
				pthread_mutex_unlock(&mutex);

			//		cvCvtColor(img1, img0, CV_YCrCb2RGB);


				//cvCvtColor(img0, img1, CV_YCrCb);
		//		cvShowImage("Client", img1);
				cvShowImage("Client", img1);

				//cvSmooth(img1, img1, CV_BLUR,3,0,0,0);
				//cvShowImage("Client_Test", img1);

				frame_init();
```

```
                key = cvWaitKey(10);
        }

        if(pthread_cancel(thread_s))    quit("pthread_cancel failed", 1);

        cvDestroyWindow("Client");
        cvDestroyWindow("Unblurred");
        quit(NULL, 0);

    return 0;
}


void frame_init(void)
{
        label = 0, maxL = 0, minL = 0;

        for(int i=0; i<HEIGHT_C;i++)
                for(int j=0; j<WIDTH_C;j++)
                        labeling[i][j] = 0;


    for(int i=0;i<MAX_LABEL;i++)
    {
            min_axis[i][0] = WIDTH_C-1;
            min_axis[i][1] = HEIGHT_C-1;
    }

        for(int i=0;i<MAX_LABEL;i++)
        {
                        max_axis[i][0] = 0;
                        max_axis[i][1] = 0;
        }

}

void debugging(void)
{

}


void Labeling(IplImage *img)
{

        int i, j, r_ch, b_ch, g_ch, min_eq,   cnt, temp;
        int width, height, nchannels, step, offset;
        int *hash;
        int map_count_x = 0, map_count_y = 0;
```

```
unsigned char** map;
unsigned char** ptr;

width = img->width;
height = img->height;
nchannels = img->nChannels;
step = img->widthStep;

int eq_tb1[MAX_LABEL][2] = {{0,}};

map = (unsigned char**)malloc(sizeof(unsigned char*)*height);
for(i=0;i<height;i++)
{
        map[i] = (unsigned char*)malloc(sizeof(unsigned char)*width);
}

ptr = (unsigned char**)malloc(sizeof(unsigned char*)*height);
for(i=0; i<height; i++)
{
        ptr[i] = (unsigned char*)malloc(sizeof(unsigned char)*width);
}

// map, ptr init
for(i=0;i<height;i++)
{
        for(j=0;j<width;j++)
        {
                map[i][j]=0;
                ptr[i][j]=0;
        }
}



// Labeling Start : X, Y axis data store...

for(i=0; i< height; i++)
{
        for(j=0; j< width; j++)
        {
                        offset = j*nchannels;

                        // Labeling Start
                        if(img->imageData[i*step+offset+0] >= TH_B)
                            //img->imageData[i*step+offset+1] >= TH_G &&
                            //img->imageData[i*step+offset+2] >= TH_R)
                        {
```

```
                    if(i==0 && j==0)       // Case 1 : (0,0)
                    {
                            label++;
                            ptr[i][j] = label;
                            eq_tb1[label-1][0] = label;
                            eq_tb1[label-1][1] = label;
                    }
                    else if(i==0)          // Case 2 : (0,y)
                    {
                            if(ptr[i][j-1] != 0)
                                    ptr[i][j] = ptr[i][j-1];
                            else
                            {
                                    label++;
                                    ptr[i][j] = label;
                                    eq_tb1[label-1][0] = label;
                                    eq_tb1[label-1][1] = label;
                            }
                    }
                    else if(j==0)          // Case 3 : (x,0)
                    {
                            if(ptr[i-1][0] != 0)
                                    ptr[i][j] = ptr[i-1][j];
                            else
                            {
                                    label++;
                                    ptr[i][j] = label;
                                    eq_tb1[label-1][0] = label;
                                    eq_tb1[label-1][1] = label;
                            }
                    }
else        // Case 4 : Others...
{
            if( (ptr[i-1][j] !=0) && (ptr[i][j-1] !=0 ) )
            {
                    if(ptr[i-1][j] == ptr[i][j-1]) // if i-1 equal j-1
                                    {
                                            ptr[i][j] = ptr[i-1][j];
                                    }
                                    else // if i-1 not_equal j-1
                                    {
                            maxL = MAXX(ptr[i-1][j], ptr[i][j-1]);
                            minL = MINN(ptr[i-1][j], ptr[i][j-1]);

                                            ptr[i][j] = minL;

                    min_eq = MINN(eq_tb1[maxL-1][1], eq_tb1[minL-1][1]);

                            eq_tb1[(eq_tb1[maxL-1][1])-1][1] = min_eq;
```

```
                                                                        eq_tb1[maxL-1][1] = min_eq;
                                                                        eq_tb1[minL-1][1] = min_eq;

                                                                                //for(g=i;g>=0;g--)
                                                                                        //{
                                                                                        //
        if(labeling[g][j]==maxL)
                                                                                        //
        labeling[g][j] = minL;
                                                                                        //}

                                                                        //if( labeling[i-1][j] == maxL)
                                                                                        //{
                                                                        //if(TEST_P)            printf("     lC-- x:%d ", i);
                                                                        //if(TEST_P)            printf(" y:%d / ",j);
                                                                                                //lC--;
                                                                                        //}

                                                                        //if( labeling[i-1][j] < labeling[i][j-1] )
                                                                                //            labeling[i][j-1] = minL;

                                                                                }
                                                                        }
                                                                        else if( ptr[i-1][j] !=0 )
                                                                        {
                                                                                ptr[i][j] = ptr[i-1][j];
                                                                        }
                                                                        else if( ptr[i][j-1] !=0 )
                                                                        {
                                                                                ptr[i][j] = ptr[i][j-1];
                                                                        }
                                                                        //else if( labeling[i-1][j-1] !=0 )
                                                                        //{
                                                                        //            labeling[i][j] = labeling[i-1][j-1];
                                                                        //}
                                                        //else if( labeling[i-1][j] == 0 && labeling[i][j-1] == 0 && labeling[i-1][j-1] == 0 )
                                                                        else
                                                                        {
                                                                                //if(i!=159 && j!=119)
                                                                                //if(TEST_P)            printf("     else
lC++ x:%d ", i);
                                                                                //if(TEST_P)            printf("y:%d
/", j);
                                                                                label++;
                                                                                ptr[i][j] = label;
                                                                                eq_tb1[label-1][0] = label;
                                                                                eq_tb1[label-1][1] = label;
                                                                        }
                                                                }
```

```
                                        }
                        }
        }
        // Labeling End


        hash = (int*)malloc(sizeof(int)*(label));
        memset(hash,0,sizeof(int)*(label));

        for(int i=0;i<label;i++)
                hash[eq_tb1[i][1]-1] = eq_tb1[i][1];            // standard for eq_tb1[][1] to label
        cnt = 1;

        for(int i=0;i<label;i++)
                if(hash[i]!=0)
                        hash[i] = cnt++;
        for(int i=0;i<label;i++)
                eq_tb1[i][1] = hash[eq_tb1[i][1]-1];            // reassignment to eq_tb1 base on cnt(new
label numbering)


        for(int i=0;i<height;i++){
                for(int j=0;j<width;j++){
                        if(ptr[i][j]!=0){

                                temp = ptr[i][j];
                                map[i][j] = eq_tb1[temp-1][1];
                                if(i < min_axis[map[i][j]-1][0]){
                                        min_axis[map[i][j]-1][0] = i;
                                }
                                if(i > max_axis[map[i][j]-1][0]){
                                        max_axis[map[i][j]-1][0] = i;
                                }
                                if(j < min_axis[map[i][j]-1][1]){
                                        min_axis[map[i][j]-1][1] = j;
                                }
                                if(j > max_axis[map[i][j]-1][1]){
                                        max_axis[map[i][j]-1][1] = j;
                                }
                        }
                }
        }


        if(MAP_TEST)
        {
                printf("₩n₩n **********   original map   ********** ₩n");
                //for(i=height-1;i>=0;i--)
                //{
```

```
                        //for(j=width-1;j>=0;j--)
                        //{

                for(i=0;i<height;i++)
                {
                        for(j=0;j<width;j++)
                        {
                                //map_count_x++;
                                printf("%d", map[i][j]);

                                if(map_count_x == 10)
                                {
                                        //printf(".");
                                        map_count_x = 0;
                                }
                        }
                        //map_count_y++;
                        if(map_count_y == 10)
                        {
                                //printf("..");
                                map_count_y = 0;
                        }
                        printf("₩n");
                }
        }



        printf("₩n================================== label number : %d
==============================₩n", cnt-1);
        for(i=0;i<cnt-1;i++)
        {
                printf("label= %d  ", i+1);
                printf("   i  ");
                for(j=0;j<2;j++)
                {
                        printf("min = %d  max = %d  ", min_axis[i][j], max_axis[i][j] );
                }
                printf("  j   ");
                printf("₩n                                      Center                                  =
(%d,%d)",CENTER(min_axis[i][1],max_axis[i][1]),CENTER(min_axis[i][0],max_axis[i][0]));
                printf("₩n");
        }



/*
        for(j=0; j< img->height;j++)
```

```
	{
		for(i=0; i< img->width;i++)
		{
			offset = i*nchannels;

				if(labeling[i][j] != 0)
				{
					if(i < min_axis[labeling[i][j]][0])
						min_axis[labeling[i][j]][0] = i;
					if(i >   max_axis[labeling[i][j]][0])
						max_axis[labeling[i][j]][0] = i;
					if(j < min_axis[labeling[i][j]][1])
						min_axis[labeling[i][j]][1] = j;
					if(j > max_axis[labeling[i][j]][1])
						max_axis[labeling[i][j]][1] = j;
				}

		}
	}
*/

/*
	// Filtering Start -> make like this -> min, max X = 0, Y = 0

	for(int i=1;i<=lC;i++)
		if( max_axis[i][0] == 0 && max_axis[i][1] == 0 ||
		    max_axis[i][0] == min_axis[lC][0] && max_axis[i][1] == min_axis[i][1]   )
		{
			min_axis[i][0]		= 0;
			min_axis[i][1] = 0;
			max_axis[i][0]		= 0;
			max_axis[i][1]		= 0;
		}
*/

		free(hash);
		for(i=0;i<height;i++)
			free(ptr[i]);
		free(ptr);

		for(i=0;i<height;i++)
			free(map[i]);
		free(map);

		glabel = cnt -1;

}
```

```
void Threshold(IplImage *img)
{
	int i, j, r_ch, b_ch, g_ch;
	int width, height, nchannels, step, offset;

	width = img->width;
	height = img->height;
	nchannels = img->nChannels;
	step = img->widthStep;


	for(i=0; i< height;i++)
	{
		for(j=0; j< width;j++)
		{
			offset = j*nchannels;

// Thresholding Value Setting
if(//img->imageData[i*step+offset+0] <= TH_B &&
   //img->imageData[i*step+offset+1] <= TH_G &&
   img->imageData[i*step+offset+2] <= TH_R)
{
		img->imageData[i*step+offset] = 0;
		img->imageData[i*step+offset+1] = 0;
		img->imageData[i*step+offset+2] = 0;

}

		}
	}

	  //printf("Threshold End₩n");
}


void print_labeling(void)
{

	  int offset;

	  printf("₩n₩nlC : %d₩n₩n", label);

	  int debuc=0;
	  // Debugging
	  for(int i=0; i<HEIGHT_C;i++)
	  {
	  for(int j=0; j<WIDTH_C;j++)
	  {
				//offset = j*nchannels;
```

```
                                                //printf("%d", labeling[i][j]);
        }
                        printf("₩n");
                        debuc++;
                if(debuc%10 ==0)      printf(".");
        }


        for(int i=1;i<=label;i++)
        {
                printf("Label Number : %d₩n",i);
                printf("min X = %d     ",     min_axis[i][0] );
                printf("min Y = %d     ₩n", min_axis[i][1] );
                printf("max X = %d     ",     max_axis[i][0] );
                printf("max Y = %d     ₩n₩n", max_axis[i][1] );


        }
}



void print_yellow_rect(IplImage *img)
{
        int i, j, r_ch, b_ch, g_ch;
        int width, height, nchannels, step, offset;

        width = img->width;
        height = img->height;
        nchannels = img->nChannels;
        step = img->widthStep;

/*
        //test
        for(i=0; i< img->height; i++)
                for(j=0; j< img->width; j++)
                {
                        offset = j*nchannels;

                        for(int labelN; labelN < glabel; labelN++)
                                if( j == CENTER(min_axis[i][0],max_axis[i][0]) &&                       i       ==
CENTER(min_axis[i][1],max_axis[i][1]) )
                                {
                                        img->imageData[i*step+offset+0] = 0;
                                img->imageData[i*step+offset+1] = 0;
                                img->imageData[i*step+offset+2] = 0;

                                }
                }
```

```
*/


                    // Paint Yellow Rectangle


        for(i=0; i< img->height;i++)
            for(j=0; j< img->width;j++)
                {
                    offset = j*nchannels;

                                for(int labelN=0; labelN < glabel; labelN++)
                                {

                                        if(i == CENTER(min_axis[labelN][0],max_axis[labelN][0]))
                                        {
                                            for(int m = min_axis[labelN][1]; m <= max_axis[labelN][1]; m++)
                                            {
                                                    offset = m*nchannels;

                                                    img->imageData[i*step+offset+0] = 0;
                                                    img->imageData[i*step+offset+1] = 0;
                                                    img->imageData[i*step+offset+2] = 255;
                                            }
                                        }

                                        if(j == CENTER(min_axis[labelN][1],max_axis[labelN][0]))
                                        {
                                            for(int n = min_axis[labelN][0]; n <= max_axis[labelN][1]; n++)
                                        {
                                          offset = j*nchannels;

                                          img->imageData[ n *step+offset+0] = 0;
                                          img->imageData[ n *step+offset+1] = 0;
                                          img->imageData[ n *step+offset+2] = 255;
                                        }

                                        }
/*
                                        // at MAX Y
                                        if(i == max_axis[labelN][0])
                                        {
                                            for( int m = min_axis[labelN][1]; m <= max_axis[labelN][1]; m++ )
                                            {
                                                    offset = m*nchannels;

                                                    img->imageData[i*step+offset+0] = 0;
                                                    img->imageData[i*step+offset+1] = 255;
```

```
                                        img->imageData[i*step+offset+2] = 255;
                                }
                        }

                        // at MIN X
                        if(j == min_axis[labelN][0])
                        {
                                for(int n=min_axis[labelN][1]; n<=max_axis[labelN][1]; n++)
                                {
                                        offset = j*nchannels;

                                        img->imageData[ n *step+offset+0] = 0;
                                        img->imageData[ n *step+offset+1] = 255;
                                        img->imageData[ n *step+offset+2] = 255;
                                }
                        }

                        // at MAX X
                        if(j == max_axis[labelN][0])
                        {
                                for(int n=min_axis[labelN][1]; n<=max_axis[labelN][1]; n++)
                                {
                                        offset = j*nchannels;

                                        img->imageData[ n *step+offset+0] = 0;
                                        img->imageData[ n *step+offset+1] = 255;
                                        img->imageData[ n *step+offset+2] = 255;
                                }
                        }
*/

                                }
                        }



}



void pre_image_set(IplImage *img)
{

        int i, j, r_ch, b_ch, g_ch;
        int width, height, nchannels, step, offset;

        width = img->width;
        height = img->height;
```

```
	nchannels = img->nChannels;
	step = img->widthStep;


			for(j=0; j< img->height-EQ_INTV;j++)
	{
		for(i=0; i< img->width-EQ_INTV;i++)
		{
			offset = i*nchannels;

						for(m=0; m<=EQ_INTV;m++)
						{
							for(n=0; n<=EQ_INTV;n++)
							{
								if(//img->imageData[j*step+offset+0] >= TH_B &&
								   //img->imageData[j*step+offset+1] >= TH_G &&
								   img->imageData[j*step+offset+2] >= TH_R)
								{
									for(p=0;p<=m;p++)
									{
										for(q=0;q<=n;q++)
										{
											offset = (i+n-
q)*nchannels;

											// equal pre_pixel to
now_pixel
											img->imageData[(j+m-
p)*step+offset] = img->imageData[j*step+offset+0];
											img->imageData[(j+m-
p)*step+offset+1] = img->imageData[j*step+offset+1];
											img->imageData[(j+m-
p)*step+offset+2] = img->imageData[j*step+offset+2];
										}
									}
								}
							}
						}
		}
	}

}

void tcp_client(void)
{

		if(( sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
		{
			perror("socket call failed");
```

```
                exit(1);
        }
        printf("socket call success...₩n");

        if( connect(sockfd, (struct sockaddr *) &server, SIZE) == -1)
        {
                perror("connect call failed");
                exit(1);
        }
        printf("connect call success...₩n");

}

void* streamClient(void* arg)
{
        pthread_setcancelstate(PTHREAD_CANCEL_ENABLE, NULL);
        pthread_setcanceltype(PTHREAD_CANCEL_ASYNCHRONOUS, NULL);

        int imgsize = img1->imageSize;
        int bytes, i;

        while(1)
        {
                pthread_mutex_lock(&mutex);
                if(is_data_ready)
                {
                        bytes = send(sockfd, img1->imageData, imgsize, 0);
                        //printf("send complete ₩n");
                        is_data_ready = 0;
                }
                pthread_mutex_unlock(&mutex);

                //if(bytes != imgsize)
                //{
                        //printf("bytes != imgsize... ₩n");
                        //fprintf(stderr, "Connection closed ₩n");
                        //close(sockfd);
                        //if(connect(sockfd, (struct sockaddr *) &server, SIZE) == -1)
                        //              quit("byte!=imgsize connection failed ₩n", 1);
                //}

                pthread_testcancel();
                usleep(100);
        }
}

void printvalue(FILE *rfp, FILE *gfp, FILE *bfp, IplImage *img)
{
        int i, j, r_ch, b_ch, g_ch;
```

```
        int width, height, nchannels, step, offset;

        width = img->width;
        height = img->height;
        nchannels = img->nChannels;
        step = img->widthStep;

        for(j=0;img->height;j++)
        {
                uchar *data = (uchar*)(img->imageData + j*step);

                for(i=0; i< img->width;i++)
                {
                        offset = i*nchannels;
                        b_ch = data[offset];
                        g_ch = data[offset+1];
                        r_ch = data[offset+2];

                        fprintf(rfp, "%4d", r_ch);
                        fprintf(gfp, "%4d", g_ch);
                        fprintf(bfp, "%4d", b_ch);
                }
                fprintf(rfp,"₩n");
                fprintf(gfp,"₩n");
                fprintf(bfp,"₩n");
        }
        //fprintf(rfp,"₩n₩n");
        //fprintf(rfp,"₩n₩n");
        //fprintf(rfp,"₩n₩n");

}


void wait(float seconds)
{
        clock_t endwait;
        endwait = clock() + seconds * CLOCKS_PER_SEC;
        while(clock() < endwait) {}
}

void quit(const char* msg, int retval)
{
        if(retval == 0)
        {
                fprintf(stdout, (msg == NULL ? "" : msg));
                fprintf(stdout, "₩n");
        } else {
                fprintf(stderr, (msg == NULL ? "" : msg));
                fprintf(stderr, "₩n");
```

```
        }
        if(sockfd) close(sockfd);
          if(capture)   cvReleaseCapture(&capture);
        if(img1) cvReleaseImage(&img1);

        pthread_mutex_destroy(&mutex);
        exit(retval);
}
```

Server

```
#include "opencv2/highgui/highgui.hpp"
#include "opencv/cv.h"
#include <ctype.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>

#define SIZE sizeof(struct sockaddr_in)

void catcher(int sig);
int colorsockfd, redsockfd, sockfd;
struct sockaddr_in server = {AF_INET, 9009, INADDR_ANY};
void tcp_server(void);
void wait(float seconds);

void* Func_Client_Color(void *arg);
void* Func_Client_Red(void *arg);

void* Func_Super_Color(void *arg);
void* Func_Super_Red(void *arg);

void quit(const char *msg, int retval);

IplImage *color_img, *red_img;
pthread_mutex_t color_mutex = PTHREAD_MUTEX_INITIALIZER;
```

```
pthread_mutex_t red_mutex = PTHREAD_MUTEX_INITIALIZER;
int key;
int color_is_data_ready = 0;
int red_is_data_ready = 0;

pthread_t Th_Client_Color, Th_Client_Red;
pthread_t Th_SuperClient_Color, Th_SuperClient_Red;


int main(int argc, char ** argv)
{

        int th_num = 0;
        void *thread_result;

        //** width, height **//
        int width = 320;
        int height = 240;

        color_img = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);
        red_img = cvCreateImage(cvSize(width, height), IPL_DEPTH_8U, 3);
        cvZero(color_img);
        cvZero(red_img);

//      printf("img->imageSize = %d₩n ", img->imageSize);
//      printf("img->widthStep = %d₩n", img->widthStep);
//      printf("img->height = %d₩n", img->height);
//      printf("img->width = %d₩n", img->width);

//      cvNamedWindow("Server", CV_WINDOW_AUTOSIZE);

        tcp_server();

        //      Thread_Client_Color
        if( (colorsockfd = accept(sockfd, NULL, NULL)) == -1 )
                perror(" 1st accept call failed");
        printf(" First accept call success...₩n", th_num);

        if( pthread_create(&Th_Client_Color, NULL, Func_Client_Color, NULL))
                quit("Th_Client_Color Failed...", 1);

        //      Thread_Client_Red
        if( (redsockfd = accept(sockfd, NULL, NULL)) == -1 )
                perror(" Second accept call failed");
        printf(" Second accept call success...₩n", th_num);

        if( pthread_create(&Th_Client_Red, NULL, Func_Client_Red, NULL))
                quit("Th_Client_Red Failed...", 1);
```

```
        if( pthread_join(Th_Client_Color, &thread_result) )
                quit("Thread_Client_Color join failed...",1);
        if(thread_result == NULL)
        {
                //return 0;
                if( pthread_join(Th_Client_Color, &thread_result) )
                        quit("Thread_Client_Color join failed...",1);
                if(thread_result == NULL){
                        close(colorsockfd);
                        close(redsockfd);
                        return 0;
                }
                else{
                        close(colorsockfd);
                        close(redsockfd);
                        return 1;
                }
        }
        else{
                close(colorsockfd);
                close(redsockfd);
                return 1;
        }
} // end of main();
//cvDestroyWindow("Server");

                        ////////////////////////////
                        ////          Client    ////
                        ////////////////////////////

void* Func_Client_Color(void *arg)
{
        printf("Func_Client_Color Thread Start ! ₩n");
        if(pthread_create(&Th_SuperClient_Color, NULL, Func_Super_Color, NULL))
                quit("Th_SuperClient_Color create failed", 1);

    //fprintf(stdout, "Press 'q' to quit ₩n₩n");
    cvNamedWindow("Server_Color", CV_WINDOW_AUTOSIZE);

    while(key != 'q')
    {
            pthread_mutex_lock(&color_mutex);
            //printf(" cvShowImage mutex IN   ₩n");
            if(color_is_data_ready)
            {
                    cvShowImage("Server_Color", color_img);
                    color_is_data_ready = 0;
```

```
                }
                pthread_mutex_unlock(&color_mutex);
                //printf(" cvShowImage mutex OUT    ₩n");

                key = cvWaitKey(10);
        }

          if(pthread_cancel(Th_SuperClient_Color))
                        quit("pthread_cancel failed", 1);
        cvDestroyWindow("Server_Color");
        quit(NULL, 0);

}

void* Func_Client_Red(void *arg)
{
          printf("Func_Client_Red Thread Start ! ₩n");
          if(pthread_create(&Th_SuperClient_Red, NULL, Func_Super_Red, NULL))
                        quit("Th_SuperClient_Red create failed", 1);

          //fprintf(stdout, "Press 'q' to quit ₩n₩n");
        cvNamedWindow("Server_Red", CV_WINDOW_AUTOSIZE);

        while(key != 'q')
        {
                pthread_mutex_lock(&red_mutex);
                //printf(" cvShowImage mutex IN    ₩n");
                if(red_is_data_ready)
                {
                        cvShowImage("Server_Red", red_img);
                        red_is_data_ready = 0;
                }
                pthread_mutex_unlock(&red_mutex);
                //printf(" cvShowImage mutex OUT    ₩n");

                key = cvWaitKey(10);
        }

        if(pthread_cancel(Th_SuperClient_Red))
                  quit("pthread_cancel failed", 1);
        cvDestroyWindow("Server_Red");
        quit(NULL, 0);
}


                        ////////////////////////////////
                        ////            Super Client            ////
                        ////////////////////////////////
```

```
void* Func_Super_Color(void *arg)
{
        printf("-------Func_Super_Color Thread Start ! ₩n");
    int imgsize = color_img->imageSize;
    char sockdata[imgsize];
    int i,j,k,bytes;

    while(1)
    {
        for(i=0; i<imgsize; i+=bytes)
        {
     if((bytes = recv(colorsockfd, sockdata + i, imgsize-i, 0)) == -1)
                quit("recv failed", 1);
        }

        pthread_mutex_lock(&color_mutex);
        //printf("mutex in ₩n");
        for(i=0, k=0;i< color_img->height; i++)
        {
            for(j=0;j< color_img->widthStep; j++)
            {
  ((uchar*)(color_img->imageData + i*color_img->widthStep))[j] = sockdata[k];
                //((uchar*)(color_img->imageData + i*color_img->widthStep))[j+1] = sockdata[k+1];
                //((uchar*)(color_img->imageData + i*color_img->widthStep))[j+2] = sockdata[k+2];
            //k+=3;
            k++;
            }
        }
        color_is_data_ready = 1;
        pthread_mutex_unlock(&color_mutex);
        //printf("mutex out ₩n");

        pthread_testcancel();
        usleep(100);
    }
}

void* Func_Super_Red(void *arg)
{
       printf("-------Func_Super_Red Thread Start ! ₩n");
    int imgsize = red_img->imageSize;
    char sockdata[imgsize];
    int i,j,k,bytes;

    while(1)
    {
        for(i=0; i<imgsize; i+=bytes)
        {
```

```
        if((bytes = recv(redsockfd, sockdata + i, imgsize-i, 0)) == -1)
                quit("recv failed", 1);
            }

            pthread_mutex_lock(&red_mutex);
            //printf("mutex in ₩n");
            for(i=0, k=0;i< red_img->height; i++)
            {
                for(j=0;j< red_img->widthStep; j++)
                {
                    ((uchar*)(red_img->imageData + i*red_img->widthStep))[j] = sockdata[k];
                k++;
                }
            }
            red_is_data_ready = 1;
            pthread_mutex_unlock(&red_mutex);
            //printf("mutex out ₩n");
            pthread_testcancel();
            usleep(100);
        }

}


void tcp_server(void)
{

        if( ( sockfd = socket(AF_INET, SOCK_STREAM, 0 )) == -1)
        {
            perror("socket call failed");
            exit(1);
        }
        printf("sockcet call success...₩n");

        if( bind(sockfd, (struct sockaddr *) &server, SIZE) == -1 )
        {
            perror("bind call failed");
            exit(1);
        }
        printf("bind call success...₩n");

        if( listen(sockfd, 2) == -1 )
        {
            perror("listen call failed");
            exit(1);
        }
        printf("listen call success...₩n");

}
```

```
void wait(float seconds)
{
	clock_t endwait;
	endwait = clock() + seconds * CLOCKS_PER_SEC;
	while(clock() < endwait)		{}
}


void catcher(int sig)
{
//	close(newsockfd);
	exit(0);
}

void quit(const char* msg, int retval)
{
	if(retval == 0)
	{
		fprintf(stdout, (msg == NULL ? "" : msg));
		fprintf(stdout, "₩n");
	} else {
		fprintf(stderr, (msg == NULL ? "" : msg));
		fprintf(stderr, "₩n");
	}
//	if(sock) close(sock);
	if(color_img) cvReleaseImage(&color_img);

	pthread_mutex_destroy(&color_mutex);
	pthread_mutex_destroy(&red_mutex);

	exit(retval);
}
```