**R E P O R T**

**과목명** OOOO

**담당교수** OOO 교수님

**이름** OOO

**학과명** OOOOO과

**학번**  2000112345

**Term Project**

**-시각장애인 보도블럭 인식하기**

**과목명**  컴퓨터비전프로그래밍

**담당교수**  김성호

**학과명**  전자공학과

**학번**  21611663

**이름**  이주형

**차 례**

**들어가며**

**01 Flow chart**

**02 Homography**

**03 Color**

**04 Histogram Equalization**

**05 Canny edge & Hough line**

**06 Find Blob**

**07 Result**

**08 Conclusion**

**09 Reference**

|  |  |
| --- | --- |
| **들어가며**  **01**  **Flow chart**  **02. Homography**  **03**  **color**  **04**  **Histogram equalization**  **04**  **Canny edge &**  **Hough line**  **05**  **Find blob**  **06**  **Result**  **07**  **Conclusion**  **08**  **Reference** | 컴퓨터 비전 텀 프로젝트로 “시각장애인 보도 블록 인식하기”라는 주제를 선정했다.  길을 걷다가 끊기거나 파손된 보도블록들이 있는 것을 보고 이렇게 관리되지 않는 블록들이 있을 때에도 시각 장애인들이 제약없이 보도블록을 인식할 수 있을까 라는 생각으로 컴퓨터비전 텀 프로젝트 주제를 선정하게 되었다. 주제를 선정하고 찾아보니 시력이 조금이라도 남아있는 분들이 길을 잘 따라갈 수 있도록 눈에 잘 띄는 색인 노란색을 사용했다는 것을 알 수 있었다. 또 관리되지 않는 보도블록에 대한 항의가 많았다. 특히 횡단보도를 건너는 분들을 잘못된 방향으로 이끄는 블록들이 문제가 심했다.    <파손된 보도블럭>    <도중 끊긴 보도블럭>    <잘못된 방향으로 횡단보도를 건너게 하는 보도블럭>  그래서 이런 문제를 해결하기 위해 보도블럭이 배치된 규칙을 찾아보았다. 보도블럭은 직선과 동그라미로 이루어진 두 종류이며 각각 멈춤과 직진의 의미를 담고 있었다.  ìê°ì¥ì ì¸ ë³´ëë¸ë­ì ëí ì´ë¯¸ì§ ê²ìê²°ê³¼  <시각장애인용 보도블럭>  따라서 보도블럭 두 개의 의미를 컴퓨터가 먼저 인식할 수 있도록 하는 것을 과제의 목표로 삼았다.    <순서도>    우선 사진을 찍는 방향과 이용자가 서 있는 각도와 상관없이 블록을 인식하기 위해서 homography를 이용해 정면에서 보는 사진으로 변경할 수 있으면 좋을 것 같았다. 그래서 네 꼭짓점의 값을 받아와 틀어져있고 누워져있는 평면을 정면 위에서 바라보는 평면으로 바꿔 보았다. 이것은 homography행렬을 이용한 변환으로 카메라의 tilt나 위치에 상관없이 같은 평면을 서로 매칭할 수 있는 기법이다. 하지만 영상으로 output을 낼 것이었기 때문에 매번 꼭짓점을 받아와 출력하는 것은 번거롭다고 생각했다. 계속 변하는 영상에서 꼭짓점을 자동으로 인식하기 위한 아이디어를 생각해보았는데, 노란색으로 인식되는 지점의 꼭짓점을 받아오는 것이 떠올랐다.    <homography 행렬 연산을 이용해 원하는 평면을 편 모습>  노란색으로 인식되는 픽셀들의 가로세로 최소,최대값을 이용하면 꼭짓점을 받을 수 있을 것 같았다. 하지만 아래 그림처럼 노란색을 인식하는 범위를 잘못 지정하는 경우 주변 또한 노란색으로 같이 인식되어버렸다. 평면이 제대로 보여지지 못하는 문제점이 있어 homography 연산 및 변환을 중단하고 노란색으로 따온 부분에서만 진행을 하기로 했다.    <노란색이 잘못 인식되는 경우>  아래는 해당 소스이다. Mat homo 함수의 경우 내용이 많아 부분만 캡쳐했다.      <네 꼭짓점 추출을 위한 마우스 클릭이벤트 함수>    <평면으로 변경 시에 사용되는 보간 함수>    <homography 행렬을 연산하고 반환하는 함수>  다음으로는 노란색 부분만 따와서 순서를 진행한다. HSV로 채널을 분리해서 Hue가 10~30 사이인 값을 가지는 픽셀 값을 가져와 할당하는 과정을 진행했다.  C:\Users\User\Desktop\이주형_컴비전\TermP\Project1\1.jpg      <노란색만 따온 mask>  C++은 matlab과 다르게 for문을 직접 돌리며 픽셀에 하나하나 접근한다.    <노란색부분을 검출하는 소스>  이 과정을 생각하지 않고 진행하던 중 edge가 많이 깨져서 동그라미를 제대로 인식하지 못하고 주변의 물체를 동그라미로 잡는 경우도 많이 발생해서 전처리 과정을 거쳐야겠다고 생각하고 추가한 부분이다. 처음에는 노란색을 추출하는데 노란색과 다른 색의 대비를 이용해서 mask를 더욱 깨끗하게 얻어 낼 수 있지 않을까해서 찾아보았는데 찾은 함수는 gray input 영상에서 gray output 영상을 내는 함수였다. 따라서 노란색 검출 이후 부분에 적용해보았더니 edge가 선명하게 살아나는 효과를 볼 수 있었다.      <contrast를 높이기 위한 histogram equalization>    <histogram equalization 소스>  Histogram 변환을 하고나서 살아난 edge를 검출해서 blob을 인식하려고 한다.  Edge를 얻는 방법으로 우선 canny edge를 수행했다. Blob 인식이 목적이므로 Canny edge 후 강한 edge만을 따라서 테두리를 그리는 함수(drawcontours)를 사용했다. 이 함수를 사용하기 위해 알맞은 자료형을 찾는 것이 상당히 까다로웠다. 이 때 파이썬 코드의 장점을 알게 된 것 같다. 특히나 opencv는 자료형이 많아 출력을 했는데도 사진이 보이지 않는 경우가 있고 값을 변환하다 data가 잘리는 경우도 많다고 한다. 이 과정을 진행하면서 hough line 함수도 같이 사용했는데, 이 함수를 사용한 이유는 사용자의 보행방향을 강한 edge로 그려주고 끊김이 있는 보도블럭에도 진행방향을 알려줄 수 있는데 쓰일 수 있지 않을까에서였다.    맨마지막 사진처럼 findcontour, drawcontour 함수로 edge의 테두리만 따고 나니 동그라미가 더 선명해진 것을 볼 수 있었다. 이 과정에서 사용된 허프 변환은 무인 자동차 lane 인식에도 많이 사용된다고 한다.  마지막 과정으로 blob을 찾는다. 이 과정에서 대부분의 시간이 소요되었다. 처음에는 houghcircle을 사용했는데, mask를 따온 영상에서도 아래와 같이 line을 따라 circle이 그려졌다. Circle을 그리는데 왜 line을 따라 그리는지 이해가 안되었다. 함수의 radian 값을 조정해보아도 결과는 똑같았고 canny edge의 threshold 값을 건드리면 아예 circle이 출력되지 않기도 했다. 또 어떤 때는 원의 중심을 최소 영상크기의 1/100배로 잘게 잡았더니 mask가 없는 부분에서도 원의 중심이 잡혀서 화면이 아주 빨갛게 되기도 했다. 그러던 중 수업시간에 배운 blob detect를 사용해서 다시 시도해보았으나    오히려 더 작은 blob들이 검출되어서 이 마저도 효율적이지 못했다. Circle을 따내기 위해 edge도 다 따고 contour도 다 땄는데 뭐가 문제인지 도무지 감이 오지 않았다. 이번에는 영상에다 blob을 찾는 것을 적용해 보았다. 다른 영상입력에서는 노란색이 별로 많이 검출되지도 않았다. 그런데 blob을 찾는 것을 보고 아주 약한 edge에서도 blob이 detect 된다는 것을 깨닫고 이 방법은 연산량이 너무 많겠다, 내가 원하는 함수가 아닌 것 같다는 생각을 하고 다시 houghcircle 함수를 사용하기 위해 검색을 해보았다.      검색을 통해 Hough circle의 다른 함수를 이용해서 mindist와 maxdist의 값을 설정해줄 수 있는 함수를 사용할 수 있었다. 그래서 평균 반지름 값을 계산해 원하는 크기의 blob만 추출할 수 있었다. 이렇게 찾은 blob이 일정 개수 이상 화면에 나타나면 stop신호를, 일정개수에 도달하지 못하면 go를 화면에 출력했다.      위 아래 그림을 비교해보면 동그라미의 개수가 확실히 차이 나는 것을 알 수 있다. 하지만 인식해야 할 동그라미 수가 많아질수록 처리할 것이 많아서 영상처리속도가 느려지는 것을 알 수 있다.  Opencv로 과제를 진행하는 동안 형 변환 문제 등의 어려움을 마주했다. 특히 Opencv C/C++에서 주로 사용해왔던 것으로 알고 있었는데 파이썬이라는 언어도 많이 발전해서 opencv 코드가 대부분이고 형변환에 상관하지 않을 수 있어서 간단하다는 것을 많이 느꼈다. 또 직접 구현하기 어려운 영상처리코드를 내장해서 한 줄의 함수로 구현한 이 영상처리 라이브러리가 많이 쓰이는 이유를 알게 되었다.  <https://medium.com/@mrhwick/simple-lane-detection-with-opencv-bfeb6ae54ec0>  <https://opencvlib.weebly.com/cvinranges.html>  <http://blog.naver.com/PostView.nhn?blogId=samsjang&logNo=220592858479&redirect=Dlog&widgetTypeCall=true> |