OpenCV

[Python3]OpenCV 곡선 차선 인식 프로젝트 - 차선유지,핸들(2)

큐
2019. 6. 5. 1:04

이웃추가

프로젝트 기간

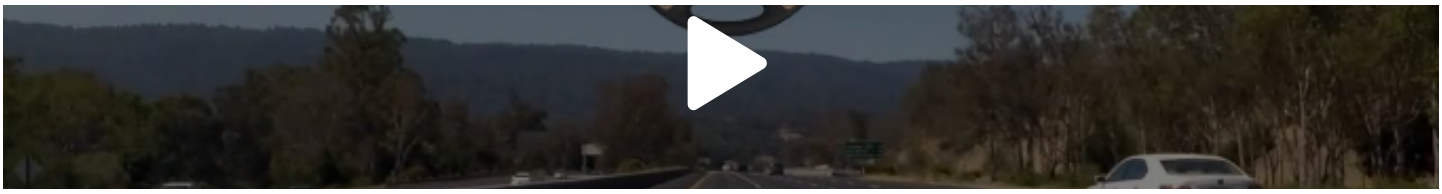
2019.06.02 - 2019.06.03 (약 2일)
독학하였음을 밝힙니다.

06.02 : 차선 곡률 계산 및 표시, 차선 중앙으로부터의 오프셋 계산 및 표시, 차량 핸들 표시
06.03 : 상단 그라데이션, 곡률 계산 알고리즘 수정, 핸들 자동 조향 구현, 차선 이탈 주의 알림

17 47

OpenCV 곡선 차선 인식 프로젝트 - 차선유지,핸들(2)

850 0



00:00 00:50 자동

OpenCV 곡선 차선 인식 프로젝트 - 차선유지,핸들(2)

▼ 이전 글(차선 인식 편) 보러 가기 ▼

[Python3]OpenCV 곡선 차선 인식 프로젝트 - 차선 인식(1)

프로젝트 기간2019.03.03 - 2019.03.05 (약 3일)독학하였음을 밝힙니다.03.03 …

blog.naver.com

Introduction

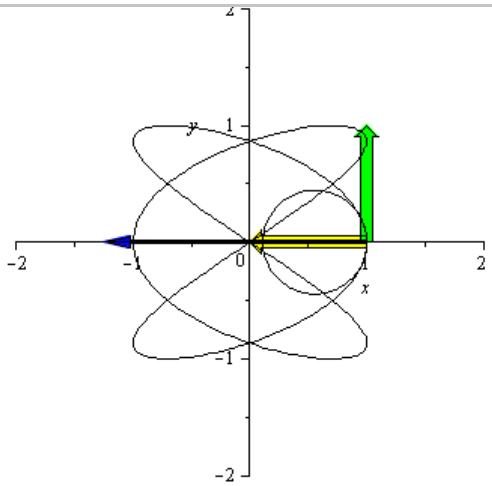
17 47

Perspective Transform, Sliding Window Search 등의 기법을 이용하여 곡선 차선 인식을 구현하는 과정을 설명했었습니다.

대략의 프로젝트 작업 환경은 다음과 같습니다.

> **CPU:** Intel Core i7-8550U
> **RAM:** 8GB
> **Graphic Card:** Intel UHD Graphics 620 (외장 그래픽 카드 없음)
> **OS:** Windows 10 Pro
> **System:** x64
> **Editor:** Jupyter Notebook (Python3)

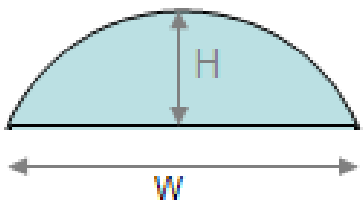
Radius of Curvature



각 위치에서 그려지는 원의 반지름이 곡률 반경!

곡률 반경 또는 곡선 반경이라고 불리는 Radius of Curvature은 차량의 자동 조향 시스템에 가장 중요한 정보를 제공합니다. 곡률 반경을 기반으로 핸들(Steering Wheel)을 얼마나 꺾을 것인지가 결정되기 때문인데요, 이 곡률 반경을 구하는데 만만찮은 시행착오를 겪었습니다.
프로젝트에서 수식적으로 구할 수 있었던 변수는 하나 뿐이었는데 구글에 검색하여 나오는 Radius of Curvature 공식은 최소한 두 개의 변수 값을 알고 있어야 결정되는 수식이었기 때문입니다. 또한 다른 프로젝트에서 찾아낸 곡률 반경 계산 코드는 대략적인 계산인지, 아니면 제 데이터셋에 맞지 않는 것인지 오차가 심했습니다. 곡률 반경이 너무 크게 나와서 이를 핸들에 적용하였을 때 Steering Ratio를 100000 : 1 이상으로 두어야만 핸들이 의미 있는 움직임을 보였는데, 그마저도 곡률 반경의 값 변화 폭이 너무 크다 보니(0.5초에 300-60000 사이를 날뜀) 핸들이 갑자기 180도 넘게 움직이기도 했습니다. 결국 해당 코드가 적용 불가능하다고 판단했던 저는 아래 사이트의 공식을 토대로 대안 코드를 작성했습니다.

Radius of an Arc or Arch - Math Open Reference
Radius of an arc or segment Definition: The radius of an arc or segment i…
www.mathopenref.com



17 47

$$Radius\ of\ Curvature(R) = \frac{W}{2} + \frac{W^2}{8H}$$

W는 프로젝트에서 수식적으로 구할 수 있었던 변수이며 H가 문제였습니다. 따라서 H 값을 최대한 근사하게 구하기 위해 직선의 기울기를 이용하였습니다. 먼저 선분 W의 양 끝 A, B를 지나는 직선의 기울기 m과 A, B의 중점 M의 좌표를 구합니다. Arc의 중점은 '점 M을 지나면서 기울기가 -m^-1인 직선' 위에 위치할 것이므로 Arc 위의 점 C와 선분 AB의 중점 M을 잇는 직선의 기울기가 -m^-1에 최대한 근접하는 C의 위치를 찾으면 됩니다. Arc 위의 점들은 이전에 Sliding Window Search를 하면서 좌표를 구해두었기 때문에 이 방법은 곡률 반경을 구하는데 대안이 될 수 있습니다.

```
0.1621262209026911 361
0.13586846949172024 362
0.10986842982623793 363
0.08412114002737046 364
0.058621757511781086 365
0.0333655555385432O4 366
0.00834791896706108 367
0.016435657564039224 368
0.04098957440109818 369
0.06531812922871497 370
0.08942552023215902 371
0.11331584900158762 372
0.1369931233354417 373
```

-m^-1에 근접할수록 0에 가까워지도록 했다.

```
# ARC_M_Y: 호의 중점 C의 y좌표값
L_H = math.sqrt((L_FIT_X[L_ARC_M_Y]*x_mpp - (L_FIT_X[0]*x_mpp + L_FIT_X[-1
R_H = math.sqrt((R_FIT_X[R_ARC_M_Y]*x_mpp - (R_FIT_X[0]*x_mpp + R_FIT_X[-1
L_W = math.sqrt((L_FIT_X[0]*x_mpp - L_FIT_X[-1]*x_mpp)**2 + (720*y_mpp)**2
R_W = math.sqrt((R_FIT_X[0]*x_mpp - R_FIT_X[-1]*x_mpp)**2 + (720*y_mpp)**2

L_RAD = ((L_H/2) + (L_W**2 / (8*L_H)))
R_RAD = ((R_H/2) + (R_W**2 / (8*R_H)))
```

마지막으로 C와 M을 이용해 H를 구하고, 현재 pixel인 단위를 meter로 변환한 뒤 공식에 대입하여 Radius를 구합니다.

17 47

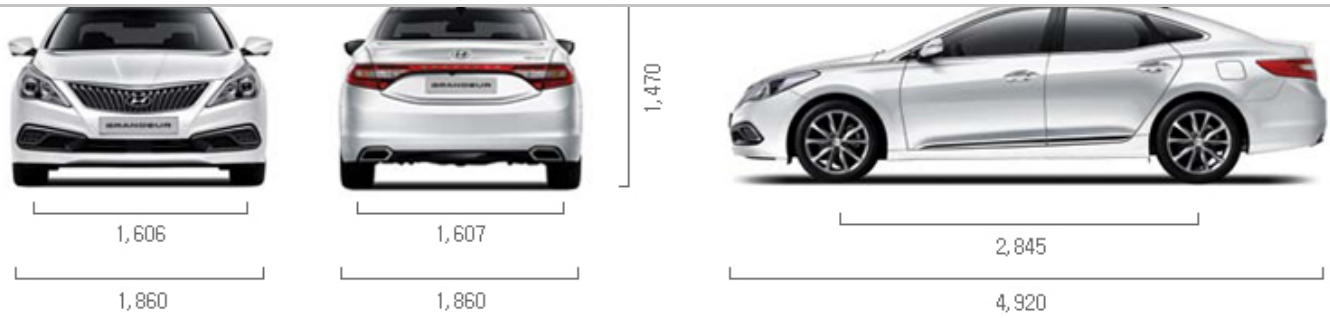
Center-Line Offset

차선 중심으로부터의 떨어진 정도는 (카메라 가로 해상도의 절반 - 차선 중앙 x좌표)로 알 수 있습니다. 차선 중앙의 x좌표는 Sliding Window Search를 하면서 얻은 두 차선의 x 좌표의 중점과 같습니다. 계산하고 나니 무언가 허전하길래, 중앙 오프셋을 계산하는 것 외에도 얼마나 치우쳐져있는지 시각적으로 볼 수 있게끔 cv2.fillPoly()를 이용해 차선 위에 오프셋 영역을 추가적으로 표시했으며, 그 오프셋이 15cm를 초과할 경우 오프셋 영역을 빨간색으로 바꿔 칠하고 차선 이탈 주의 알림을 띄우도록 하였습니다. 차선 유지 시스템처럼 오프셋이 핸들의 꺾는 각도에 영향을 주도록 할 수도 있었는데, 프로젝트에 사용된 영상이 워낙에 좌측으로 치우쳐져 있어서 이것까지 적용하면 핸들 steering이 만족스럽지 않을 것 같아 보듀했습니다.

Steering Wheel

제일 구현해보고 싶었고 중요한 파트입니다. 예상대로 핸들을 우측으로 30도 꺾는다고 해서 꼭 차량이 30도 우측으로 도는 것은 아니었습니다.

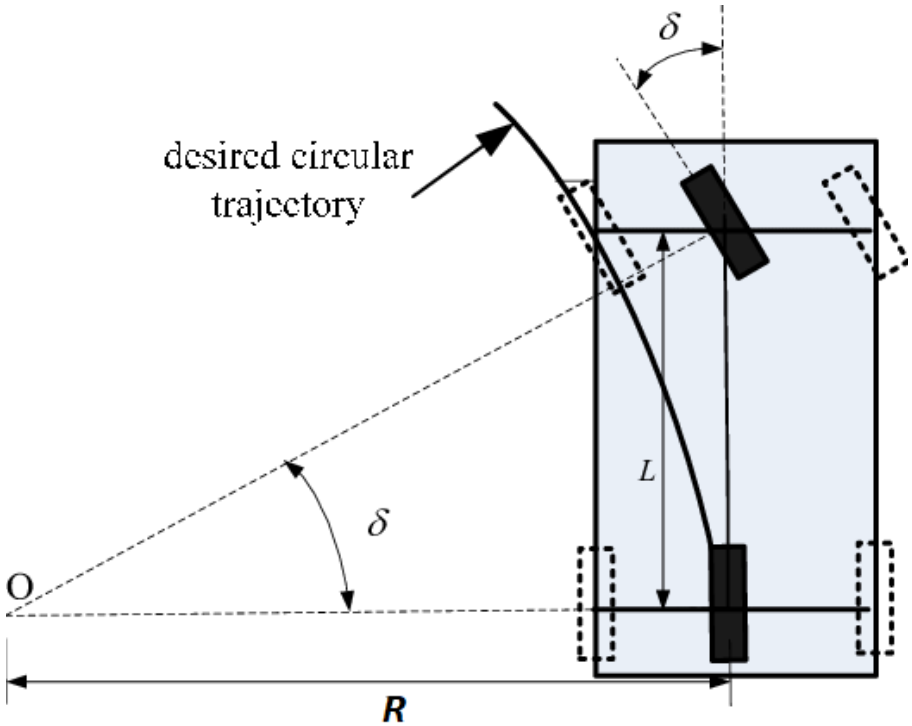
핸들의 회전 각과 차량의 회전 각 사이에는 wheel base(축거)와 steering ratio(조향비)가 관여합니다. wheel base는 차량 스펙마다 값이 다르기 때문에 저는 익숙하게 타고다녔던 현대자동차 그랜져HG 모델을 기준으로 계산하기로 했습니다. steering ratio는 일반 승용차의 경우 12:1 ~ 20:1 사이라고 하는데, 영상에서 핸들이 회전하는 것을 눈에 띄게 표현하고 싶어 20:1로 채택했습니다. 이는 실제 차량을 1° 회전시키기 위해서 핸들을 20° 꺾어야 한다는 의미입니다.



단위 : mm, 윤거는 245/45 R18 타이어 기준

이제 wheel base와 steering ratio를 적절히 정했으니 angle을 구하면 됩니다. 검색해보면 복잡한 공식이 나오기도 하는데, 프로젝트를 간단히 하기 위해 하단의 심플한 사진에서 수식을 도출했습니다. 요구되는 실제 회전 각이 세타(θ)라면 삼각비를 이용하여 세타를 구한 뒤 차량의 steering ratio를 적용하고, 곡률 반경과 휠베이스에 의해 항상 양(+) 값이 나올 것이므로 angle 방향을 결정해줄 변수를 추가적으로 곱해줍니다.

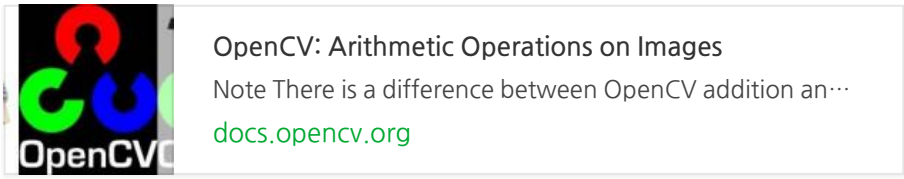
오차를 수정한 radius를 수식에 넣어 angle을 도출하고 핸들에 적용하였더니 이전처럼 핸들이 거의 움직이지 않거나 갑자기 많이 꺾이는 등의 이상 현상 없이 만족스러운 결과가 나왔습니다.



Definition of Steering Angle

참고로 왼쪽으로 꺾는 것이 양(+) 값이고, 오른쪽으로 꺾는 것은 음(-) 값입니다. 왼쪽/오른쪽 여부는 Sliding Window Search로 얻은 차선 x 좌표 배열의 인덱스 0 값과 인덱스 -1 값을 비교하여 판별했습니다.

그리고 핸들이나 경고 아이콘과 같은 4채널 PNG를 프레임 위에 그리기 위해 3가지 방법을 시도했었는데, 가장 마지막에 시도한 방법이 괜찮았습니다. 첫 번째 사진은 OpenCV docs를 보고 따라했던 Bitwise Operation인데.. 보시다시피 테두리가 영 불편합니다.



두 번째는 Bitwise Operation이 너무 별로이길래 그냥 제가 수정해버린 결과입니다. 핸들 이미지를

```
# 완성된 핸들 이미지를 배경 위에 그린다
pos_x = 540
pos_y = 110
for i in range(210):
    for j in range (210):
        if(wheel[i][j][3] < 100): # threshold = alpha(100)
            continue
        else:
            wheel_rgb = wheel[i][j][:3]
            img[i+pos_y][j+pos_x] = wheel_rgb
```

세 번째는 두 이미지의 픽셀 색상과 오버레이 할 이미지의 alpha 값을 적절히 연산하여 테두리까지 부드럽게 그리는 방법입니다. 버리는 픽셀이 없기에 상당히 부드럽게 나옵니다. (렌더링 속도를 바친 대가로요..)

```
# 완성된 핸들 이미지를 배경 위에 그린다
pos_x = 540
pos_y = 110
for i in range(210):
    for j in range (210):
        alpha = wheel[i,j,3] / 255.0
        img[i+pos_y,j+pos_x,0] = (1. - alpha) * img[i+pos_y,j+pos_x,0] + a
        img[i+pos_y,j+pos_x,1] = (1. - alpha) * img[i+pos_y,j+pos_x,1] + a
        img[i+pos_y,j+pos_x,2] = (1. - alpha) * img[i+pos_y,j+pos_x,2] + a
```

Rendering

이번에도 moviepy를 이용하여 편집된 영상을 렌더링하였습니다. 수행할 코드에 for이나 if문이 추가될 때마다 렌더링 시간이 늘어나서, 중간점검을 할 때엔 100프레임 정도까지만 렌더링하여 확인하곤 했습니다. 저전력 노트북 CPU만으로 이것저것 기능을 더 추가하기엔 슬슬 버겁지 않나 싶은데요, 사실 차량 인식 기능 하나만 더 추가해보려 했는데 상황을 보고 코드를 끼워야겠습니다. PNG 이미지 등을 프레임에 그릴 때마다 반복문에 의해 렌더링 시간이 꽤 늘어나니 이미지는 최대한 자제하면 좋을 것 같습니다. 그리고

Conclusion

이전보다 영상이 볼 거리가 생긴 것 같아 뿌듯합니다.^^ 그런데 3월에 곡선 차선 인식 구현해둔 걸 6월에 steering 시키다니(...) 그동안 시험도 봤었고 다든 프로그램 개발도 했었다는 변명밖에 나오지 않네요. 이번 프로젝트는 제가 미분적분학 진도를 나가야 해서 이틀 모두 밤을 샜습니다. 그래도 크게 막힌 것 없이 의도한 바의 결과가 나오니 다행입니다.

그런데 처음에 적용한 곡률 반경(Radius of Curvature) 코드가 왜이리 심한 오차를 냈는 지는 아직 모듭니다. 코드 출처가 Udacity 학생 것이었는데.. 다들 핸들에 적용을 안해보신 걸까요? 아니면 제가 필터링을 약하게 해서 과 데이터로 오차가 커진걸까요? 만일 Radius of Curvature 파트에서의 H를 수식으로 바로 알아낼 수 있다면 제보 부탁드립니다. 현재는 프레임 별로 H 값을 알아내기 위해 두프 200번을 돌려서 계산하고 있는데 제가 두프 돌리는걸 별로 좋아하지 않습니다. steering wheel angle에 적용 가능한 O(1)의 알고리즘을 아시는 분은 꼭 알려주세요!

이 프로젝트.. 3번째 포스팅을 하게 될 지 모르겠으나 추가할 만한 재미거리가 있다면 언제든 다시 찾아오겠습니다.
CPU가 뻗는 그날까지.. ㅋㅋ

파워링크 광고

카페24 PYTHON
업계 최고의 기술력과 노하우! 합리적 가격! 결제 즉시 사용 가능!

한국정보교육원
IT전문기관 자바(JAVA) 빅데이터 파이썬 개발자 100%국비지원 신림역6번출구

요즘 애들의 커리어 플랫폼
원티드에서 주35시간 근무, 재택근무, 연봉상위 등 1만 개의 멋진 회사 탐색

17 47

큐
IT·컴퓨터
청정수 컴맹과 귀여운 컴퓨터의 이야기
이웃추가

#소통원해요
광고
강서구 댕댕브라더스
하이앤드머신 카페
분위기 좋은 애견동반카페
리뷰 4

이 블로그 OpenCV 카테고리 글

[Python3]OpenCV 곡선 차선 인식 프로젝트 - 차선유지,핸들(2)
2019. 6. 5.
17 47

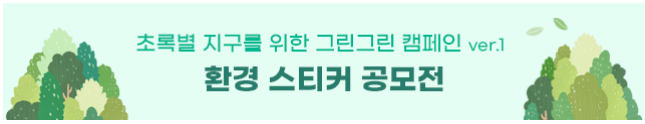
[Python3]OpenCV 곡선 차선 인식 프로젝트 - 차선 인식(1)
2019. 3. 12.
13 91

이 글에 링크된 글

[Python3]OpenCV 곡선 차선 인식 프로젝트 - 차선 인식(1)
큐 2019. 3. 12.
13 91

17 47

큐의 Qriosity Log



PC버전으로 보기