

2020년 졸업 작품 연구일지 20

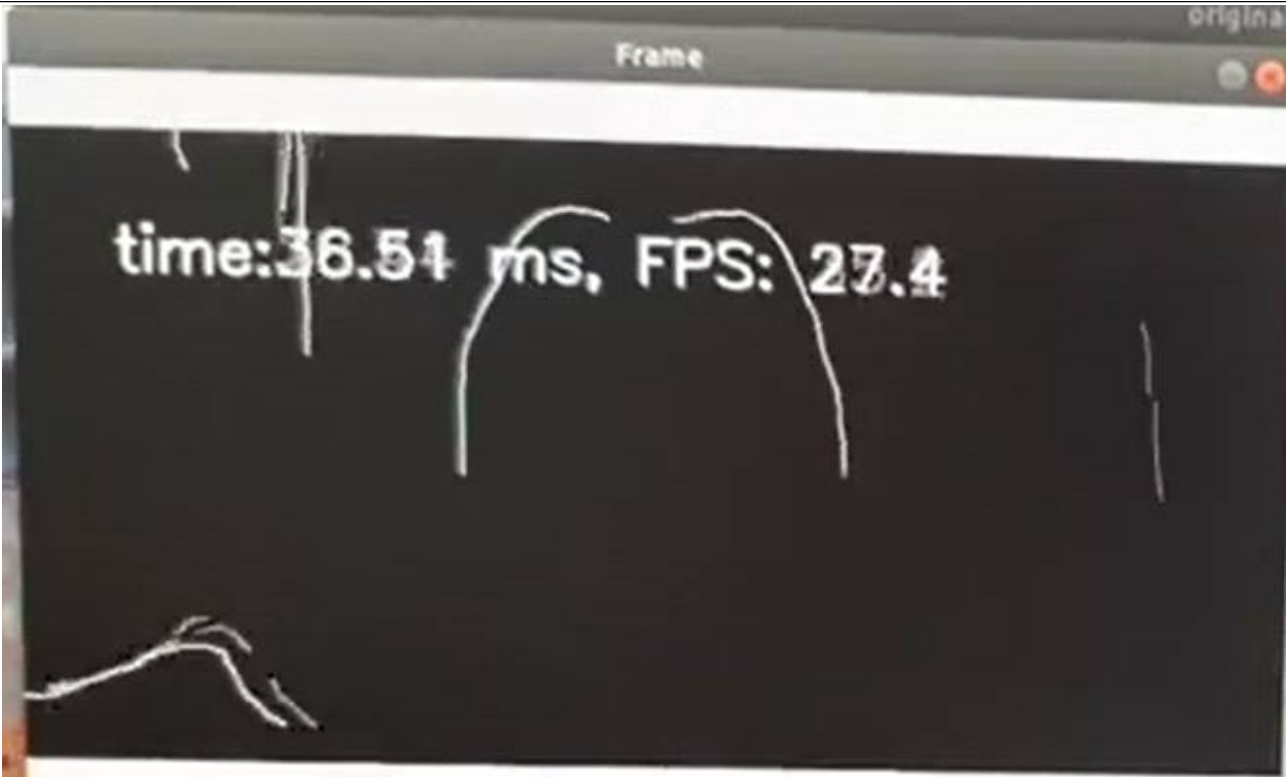
작품명	Adaptive Cruise Control by Monocular Depth Estimation	담당교수	조 용 범 (인)
조원	김용준 이준석		
지난 과제	Digging Into Self-Supervised Monocular Depth Estimation		
다음 과제	Digging Into Self-Supervised Monocular Depth Estimation		

목차

- 1. 앞차와의 거리측정을 위한 영상처리
- 2. Find Distance & Prepare to activate motor(ROS)

1. 앞차와의 거리측정을 위한 영상처리

이번 주는 depthmap image를 edge detect등 여러 가지 영상처리를 통한 데이터 추출을 시도했다.



edge detect를 이용하여 앞차의 형태 파악



앞차의 크기를 pixel단위로 접근하여 파악
결과 :

1. 앞차를 특정할 수 있다고 판단
2. 자동 주행이 아닌 앞차를 특정하고 따라가는 형태이기 때문에 영상을 바꿀 필요를 느낌
3. 주행 중 다른 차가 끼어드는 것은 영상에서 충분히 감속이 가능
4. 보드를 다시 세팅하여 다음주까지 구현을 목표

2. Find Distance & Prepare to activate motor(ROS)

1) Find Distance

전방차량을 찾아내었기 때문에, 이제 앞 차량과의 거리를 측정해서 threshold 보다 낮으면 속도를 감속하여 운행하고, 높으면 속도를 증가시켜 차간거리를 좁히고자 한다. 이를 위해 지난번 사용했던 코드를 가져와 사용하기로 하였다.

먼저 이들은 차량과의 거리를 구할 때 시점부터 바꾸고 시작하였는데 단안 카메라를 통해 보이는 세상은 1점 투시도와 같은 형태의 세계이기 때문에 앞 뒤 거리측정을 하는데는 다소 어려움이 있다. 따라서 모든 pixel들을 항공뷰처럼 위에서 바라보았을 때의 관점으로 변환하는 과정을 사용하였다. (가까운 부분은 축소시키고, 먼 부분은 늘린다.) 이 때 사용했던 변환함수는 “projection.p”라는 파일에 들어있다.

또한 이 계산법은 1280*720 pixel화면에 적합한 방법이다. 따라서 우리가 영상의 크기를 조절해주는 경우, 비례해서 거리를 곱하거나 나눠 계산해 줘야한다.

```
def calculate_position(self, bbox):
    if (self.has_position):
        pos = np.array((bbox[0]/2+bbox[2]/2, bbox[3])).reshape(1, 1, -1)
        dst = cv2.perspectiveTransform(pos, self.transform_matrix).reshape(-1, 1)
        return np.array((self.warped_size[1]-dst[1])/self.pix_per_meter[1])
    else:
        return np.array([0])
```

: 함수에서 보이는 dst가 차량을 검출한 부분의 영역 (bbox)의 영역을 토대로 시점변환 계산을 진행한 뒤 얻어진 값을 저장해준다.

warped_size(차선 및 차량과의 거리 계산을 할 때 사용하는 영상의 범위)에서 dst만큼을 제외한 값이 작을수록 해당 영상범위 내에 차량이 점유하고 있는 부위가 넓다는 의미일 것이고 이는 그 차량과의 거리가 멀지 않다는 결론이 나온다. 이 계산을 위해 넣은 상수값이 pix_per_meter이다. (이 역시 projection과정 중에 계산된 결과이다.)

위에서 사용한 방식을 차용하여, bbox대신에 우리가 검출한 edge의 pixel값을 넣어 결과를 얻고자 한다.

2) Prepare to activate motor(ROS)

ROS을 활용하여 통신하고, teensy 칩으로 모터를 제어하는 방법을 채택하였다.

```
import rospy
from std_msgs.msg import String

rospy.init_node('tutorial')
publisher = rospy.Publisher('/YOLO',String,queue_size = 10)
rate = rospy.Rate(20)

while not rospy.is_shutdown():
    publisher.publish('YOLO')
    rate.sleep()
```

Publisher

```
def laser_listener():
    rospy.init_node('laser_listener', anonymous=True)
    rsub = rospy.Subscriber('/scan', LaserScan, callback, queue_size=1)
    sub = rospy.Subscriber('/YOLO', String, call2)
    rospy.spin()

if __name__ == '__main__':
    laser_listener()
```

Subscriber (Listener)

: 과거에 obstacle avoid 시연에서 사용했던 코드 중 일부이다. 코드 상에서는 다음과 같이 퍼블리셔와 리스너가 나뉘어있으며, 이들을 통해 계산된 distance결과를 계속 publish로 ROS상에 올려줄 수 있을 것으로 기대된다. 리스너 쪽에서 거리에 따라 응답할 필요가 있을 때마다, 사전에 등록해둔 퍼블리셔 이름을 검색하여 연관데이터를 얻어 갈 수 있다.

아두이노 상에서 이용하기 쉽도록 이렇게 퍼블리셔 -> 리스너를 통해 획득한 데이터는 다시 시리얼을 통해 목적하는 Forward Pin과 Steering Pin 제어에 전달할 계획이다.

```
void motortest1()
{
    analogWrite(ForwardPin, 311);
    analogWrite(SteeringPin, map(90, 0, 180, 205, 409));
}
```

: 다음과 같이 활용할 수 있으며, map은 normalize를 위한 함수일 뿐 int값을 넣어도 된다.

모터의 출력을 제어하는 Forward Pin은 실험해 본 결과 311 정도면 충분한 속력이 나왔으며, 307 정도면 멈추었다.

이제 threshold 이내의 거리에서는 307 근처의 값으로 속력을 늦추거나 멈춰주고, threshold 이후의 거리에서는 311또는 그 이상의 값을 넣어줘서 가속시켜주는 단순한 테스트를 다음주에 진행해볼 예정이다.

자체 평가

영상처리는 금방 구현에 성공하였으나 보드를 모터까지 굴러가도록 구현하기 위한 세팅이 처음부터 다시하는 것이 안타깝다. 이번 주에 가능하면 성공시키고 싶었으나 다음 주까지 가능할듯하다

평가자
한마디

()