차량지능기초과제1

20163362 김성준

1.자율주행 인지관련 Data set 조사. 정리

포드 자율 주행 차량 데이터 세트

2017-18 년 도내 포드 자율주행차 차량이 수집한 도전적인 다중 에이전트 계절 데이터세트를 제공합니다. 차량은 디트로이트 공항, 고속도로, 도심, 대학 캠퍼스 및 교외 지역을 포함한 운전 시나리오가 혼합 된 미시간의 노선에서 수동으로 운전되었습니다.

데이터는 역동적 인 도시 환경에서 경험 날씨, 조명, 건설 및 교통 조건의 계절적 변화를 제시한다. 이 데이터 집합은 자율 주행 차량 및 다중 에이전트 시스템을 위한 강력한 알고리즘을 설계하는 데 도움이 될 수 있습니다. 데이터 집합의 각 로그는 타임 스탬프가 찍혀 있으며 모든 센서의 원시 데이터, 교정 값, 포즈 궤적, 지상 진실 포즈 및 3D 맵의 원시 데이터가 포함되어 있습니다. 모든 데이터는 ROS(오픈 소스 로봇 운영 체제)를 사용하여 시각화, 수정 및 적용할 수 있는 Rosbag 형식으로 제공됩니다.

포드 차량은 디트로이트 지역을 통과하는 평균 66km 의 경로를 횡단했다. 각 드라이브에는 DTW 공항, 미시간 디어본 대학교 캠퍼스 및 이미지에 표시된 주거 지역 사회가 포함됩니다. 각 실행은 도시 환경에서 다양한 기능을 캡처하는 경로에 약간의 변화가 있어 연구 커뮤니티에 유용한 데이터를 제공합니다.

데이터 집합에는 다음 센서의 전체 해상도 타임 스탬프 데이터가 포함되어 있습니다.

- 4 개의 HDL-32E 벨로다인 3-D 리다
- 6 포인트 그레이 1.3 MP 카메라
- 1 포인트 그레이 5 MP 대시 카메라
- Applanix POS-LV IMU

데이터 집합에는 다음이 포함됩니다.

- 3D 접지 반사도 맵
- 3D 포인트 클라우드 맵
- 6 DoF 지상 진실 포즈

- 3 DoF 현지화 포즈
- 센서 변환 및 교정

프레임 좌표

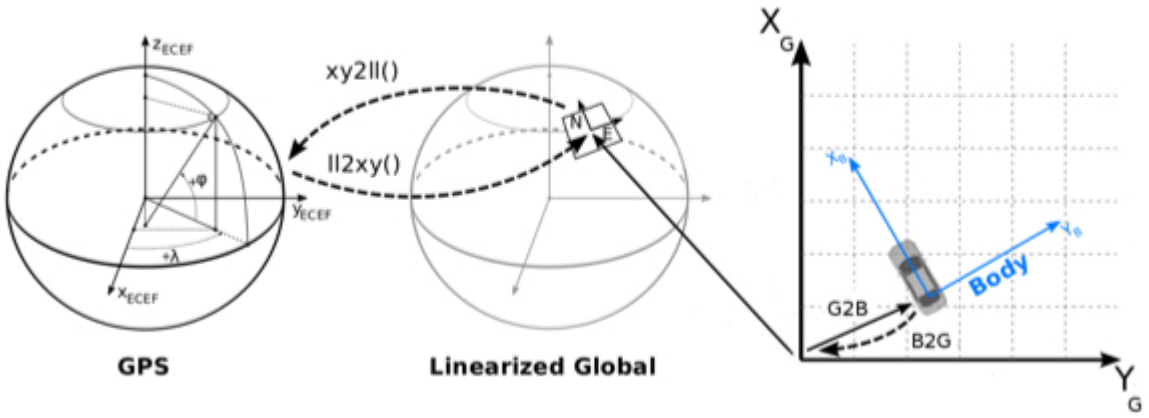
이 데이터 집합에 사용되는 좌표 프레임에는 센서 프레임, 바디 프레임, 로컬 프레임, GPS 프레임 및 전역 프레임이 포함됩니다.

X 축이 앞으로 가리키는 차량의 차체 프레임, 오른쪽에 Y 축, 아래쪽을 가리키는 Z 축입니다. 바디 프레임의 원점은 차량의 리어 액슬의 중심으로 정의됩니다.

센서는 내부 센서 프레임에 비해 데이터를 기록하므로 바디 프레임과 관련이 있어야 합니다. 차량 본체 프레임을 기준으로 센서 프레임의 위치는 외외 파라미터를 사용하여 설명된다.

데이터 집합에 사용되는 GPS 글로벌 프레임은 지구의 WGS-84 구역 구형 스페로이드 모델을 사용합니다.

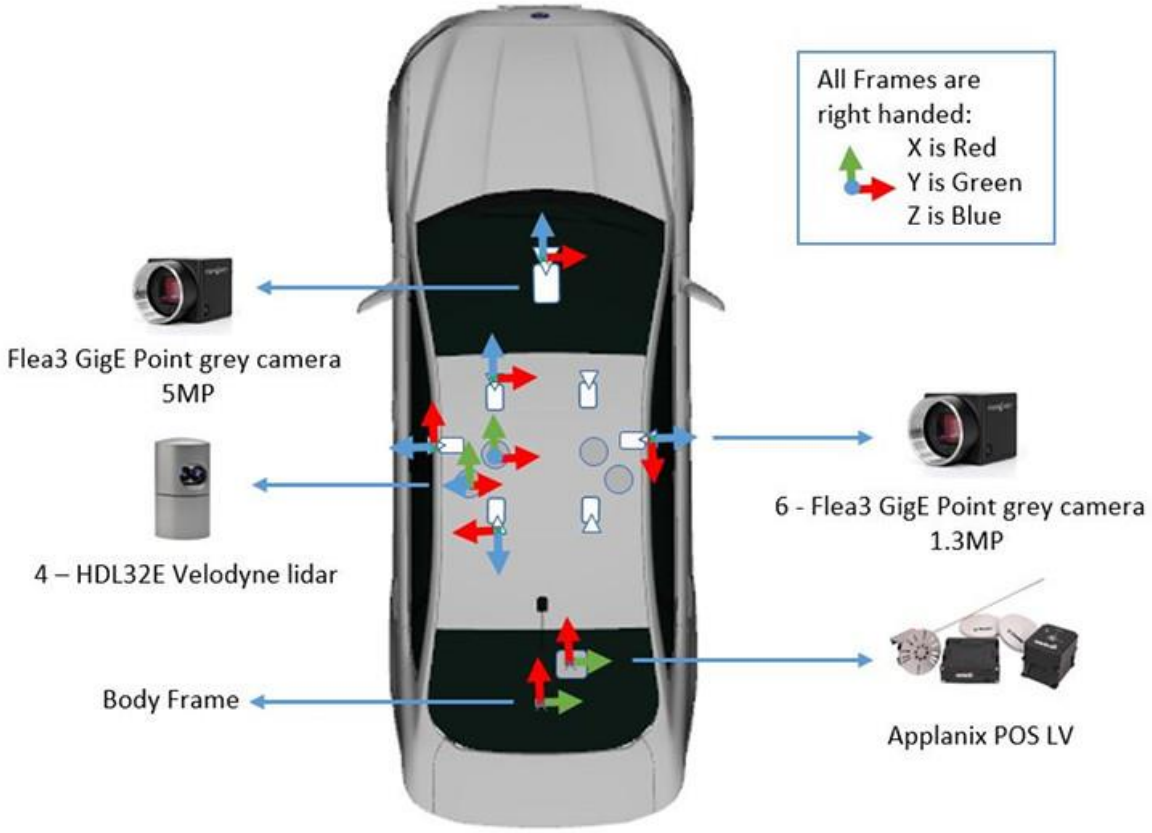
데이터 집합의 맵은 전역 프레임의 선형버전으로 표시됩니다. 선형화된 글로벌 프레임은 GPS 타원에서 위도 및 경도 선형화 지점에 연결됩니다. 데이터 집합은 북동-다운 규칙을 따릅니다.



카메라

각 로그는 차량의 모든 7 카메라의 이미지가 포함되어 있습니다. 두 개의 전면 카메라는 15Hz 에서 작동하는 1.3 MP 스테레오 쌍입니다. 두 측면 단안경 카메라는 15Hz 에서 작동합니다. 7 번째 전면 대시 카메라는 6Hz 에서 5MP 이미지를 생성합니다. 카메라 이미지와 관련된 모든 타임스탬프는 카메라가 트리거된 시간입니다.

모든 이미지는 본질적인 매개 변수를 사용하여 수정되었으며 PNG 이미지로 저장되었습니다. 모든 1.3 MP 카메라는 이미지가 동일한 타임스탬프와 동기화되도록 동시에 트리거됩니다. 각 rosbag 에는 시각화를 위해 전면 왼쪽 카메라의 미리보기 이미지가 포함되어 있습니다.



교정

데이터 집합과 함께 모든 센서에 대한 외재 교정 파일을 바디 프레임에 제공합니다. 총 12 개의 파일이 제공되어 카메라 7 개, 리다르 4 개, imu 파일 1 개등이 포함되어 있습니다. 모든 교정 값은 yaml 파일의 형태로 되어 있으며 ROS 명명방법을 준수합니다.

포즈

각 로그는 글로벌 프레임에 차량의 다음과 같은 포즈 (x, y, z, 롤, 피치, yaw)를 포함합니다.

- 원시 포즈 – 글로벌 프레임으로 변신한 아플랑닉스 POS-LV 의 통합 GNSS 포즈
- 지상 진실 포즈 - 전체 번들 조정을 사용하여 생성 된 포즈
- 지역화된 포즈 – 접지 반사도 기반 현지화를 사용하여 (x, y, yaw)에서 수정된 포즈

활용예

자율주행의 핵심 기술(인지, 측위, 판단 및 제어) 중 인지 기술은 차량 주변의 물체 궤적을 찾아내는 기술(검출)이며, 이 궤적의 다양한 소스(장애물/차량의 상대적임 움직임 정보들의 조합)를 통해 예측 기술을 학습할 수 있는 실제적인 정보를 얻게 된다. 테슬라는 이것을 “자동 라벨링”이라고 하고, 이것은 일부의 데이터 학습을 통해 그 나머지의 데이터를 예측하는 과정(예. 일부 영상만 보여주고 다른 부분을 추측하는 것과 같은 과정으로 전체 10초 녹화된 영상에서 처음 5초의 저장된 정보를 기반으로 다음 5초를 예측하는 과정)이며, 딥러닝 중의 한 형태인 “자체-교사학습(Supervised learning)”의 입력 데이터로 사용된다.

2) 자율주행 인지에 관련된 2종 이상 Open Source 조사, 정리

keras-yolo3 패키지를 이용하여 Yolo와 tiny Yolo 기반으로 이미지와 영상 Object Detection 수행

- 다크넷에서 Pretrained 된 yolo/tiny-yolo weights 모델을 다운로드
- 다운로드한 다크넷 weight 파일을 기반으로 keras-yolo3 에서 사용할 수 있는 weight 파일로 변환 후 이를 이용하여 Object Detection 수행

In [ ]:

```
import os
import sys
import random
import math
import time
import numpy as np
import tensorflow as tf
import matplotlib
import matplotlib.pyplot as plt
```

Local Directory 상에서 yolo package를 import 함.

- keras-yolo3 는 setup 을 제공하지 않으므로 local directory 상에서 바로 package 를 import 함.
- 이를 위해 keras-yolo3 를 system path 에 추가
- keras-yolo3 디렉토리에 있는 yolo.py 에서 YOLO class 를 import 하여 사용.

In [ ]:

```
LOCAL_PACKAGE_DIR = os.path.abspath("./keras-yolo3")
sys.path.append(LOCAL_PACKAGE_DIR)

from yolo import YOLO
```

In [ ]:

```
# YOLO 클래스는 model_path, achors_path, classes_path 를
model_data 밑에 파일로 가짐. 변경을 위해서는 yolo.py 파일에서 YOLO 클
래스코드를  직접 변경 필요.

print(LOCAL_PACKAGE_DIR)
!ls /home/chulmin.kwon45/DLCV/Detection/yolo/keras-yolo3
!cat /home/chulmin.kwon45/DLCV/Detection/yolo/keras-
yolo3/yolo.py
```

YOLO 객체를 사용하기 위한 모델 파일 설정 및 소스 코드 변경

- 다크넷에서 Yolo V3 Weight 모델 파일을 다운로드 받은 뒤 이를 keras-yolo3 용으로 모델 파일 변경
- model_data 디렉토리 밑에 yolo_anchors.txt, coco_classes.txt 가 있는지 확인.

In [ ]:

```
# keras-yolo3 디렉토리 밑에서 아래 명령어로 다크넷에서 Yolov3 weight
를 다운로드 받고 이를 keras-yolo3 용으로 모델 파일 변경
#!wget https://pjreddie.com/media/files/yolov3.weights
#!python convert.py yolov3.cfg yolov3.weights
model_data/yolo.h5
# convert.py 를 수행하면 yolo anchor 값이 yolo_anchors.txt 파일이 자
동으로 생성됨.
# coco label 과 클래스 매핑은 0 부터 매핑함.(0 => person)
```

In [ ]:

```
import sys
import argparse
from yolo import YOLO, detect_video
#keras-yolo 에서 image 처리를 주요 PIL 로 수행.
from PIL import Image

# YOLO 객체 생성. config 는 default 로 keras-yolo3 디렉토리에 있는
yolov3.cfg 를 적용.
config_dict = {}
yolo = YOLO(model_path='~/DLCV/Detection/yolo/keras-
yolo3/model_data/yolo.h5',
            anchors_path='~/DLCV/Detection/yolo/keras-
yolo3/model_data/yolo_anchors.txt',
            classes_path='~/DLCV/Detection/yolo/keras-
yolo3/model_data/coco_classes.txt')
```

단일 이미지 Object Detection

In [ ]:

```
# 원본 이미지 보기
```

```
img =
Image.open(os.path.join('../../data/image/beatles01.jpg'))

plt.figure(figsize=(12, 12))
plt.imshow(img)
```

In [ ]:

```
# yolo.detect_image() 메소드는 PIL package를 이용하여 image 작업 수행. keras-yolo3/font 디렉토리를 상위 디렉토리로 복사 해야함.
```

In [ ]:

```
img =
Image.open(os.path.join('../../data/image/beatles01.jpg'))
detected_img = yolo.detect_image(img)

plt.figure(figsize=(12, 12))
plt.imshow(detected_img)
```

Video Object Detection 수행

In [ ]:

```
import cv2
import time

def detect_video_yolo(model, input_path, output_path=""):

    start = time.time()
    cap = cv2.VideoCapture(input_path)

    #codec = cv2.VideoWriter_fourcc(*'DIVX')
    codec = cv2.VideoWriter_fourcc(*'XVID')
    vid_fps = cap.get(cv2.CAP_PROP_FPS)
    vid_size =
(int(cap.get(cv2.CAP_PROP_FRAME_WIDTH)),int(cap.get(cv2.CAP_PR
OP_FRAME_HEIGHT)))
    vid_writer = cv2.VideoWriter(output_path, codec, vid_fps,
vid_size)

    frame_cnt = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    print('총 Frame 갯수:', frame_cnt, '원본 영상 FPS:',vid_fps)
    index = 0
    while True:
        hasFrame, image_frame = cap.read()
        if not hasFrame:
            print('프레임이 없거나 종료 되었습니다.')
            break
        start = time.time()
```

```
        # PIL Package 를 내부에서 사용하므로 cv2 에서 읽은 image_frame
array 를 다시 PIL 의 Image 형태로 변환해야 함.
        image = Image.fromarray(image_frame)
        # 아래는 인자로 입력된 yolo 객체의 detect_image()로 변환한다.
        detected_image = model.detect_image(image)
        # cv2 의 video writer 로 출력하기 위해 다시 PIL 의 Image 형태를
array 형태로 변환
        result = np.asarray(detected_image)
        index +=1
        print('#### frame:{0} 이미지 처리시간:{1}'.format(index,
round(time.time()-start,3)))

        vid_writer.write(result)

    vid_writer.release()
    cap.release()
    print('### Video Detect 총 수행시간:', round(time.time()-
start, 5))
```

In [ ]:

```
detect_video_yolo(yolo,
'../../data/video/Night_Day_Chase.mp4',
'../../data/output/Night_Day_Chase_yolo_01.avi')
```

tiny Yolo를 이용하여 이미지와 영상 object detection 수행.

- tiny yolo weights 파일은 https://pjreddie.com/media/files/yolov3-tiny.weights 에서 다운로드 받을 수 있음.
- 다운로드 받은 tiny-yolo weight 파일을 keras-yolo3 에서 사용할 수 있게 Convert 수행 후 YOLO 객체에서 로딩하여 사용

In [ ]:

```
# wget https://pjreddie.com/media/files/yolov3-tiny.weights
#!python convert.py yolov3-tiny.cfg ./model_data/yolov3-
tiny.weights model_data/yolo-tiny.h5
```

tiny yolo weight 파일과 anchor 파일, coco 클래스 파일을 YOLO 객체 생성 시 인자로 입력

In [ ]:

```
tiny_yolo = YOLO(model_path='~/DLCV/Detection/yolo/keras-
yolo3/model_data/yolo-tiny.h5',
          anchors_path='~/DLCV/Detection/yolo/keras-
yolo3/model_data/tiny_yolo_anchors.txt',
          classes_path='~/DLCV/Detection/yolo/keras-
yolo3/model_data/coco_classes.txt')
```

단일 이미지 Object Detection

In [ ]:

```
img =
Image.open(os.path.join('../../data/image/beatles01.jpg'))
detected_img = tiny_yolo.detect_image(img)

plt.figure(figsize=(12, 12))
plt.imshow(detected_img)
```

Video Object Detection

In [ ]:

```
detect_video_yolo(tiny_yolo,
'../../data/video/secretlife_pet.mp4',
'../../data/output/secretlife_pet_yolo01.avi')
```