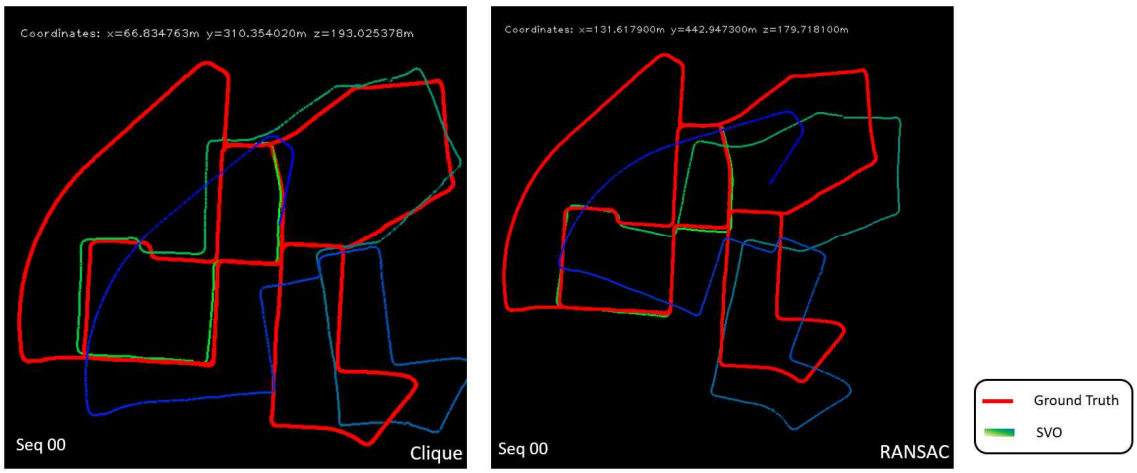
Robot Vision Project

PoseNet-Pytorch을 활용한 초기값 설정과 Visual Odometry

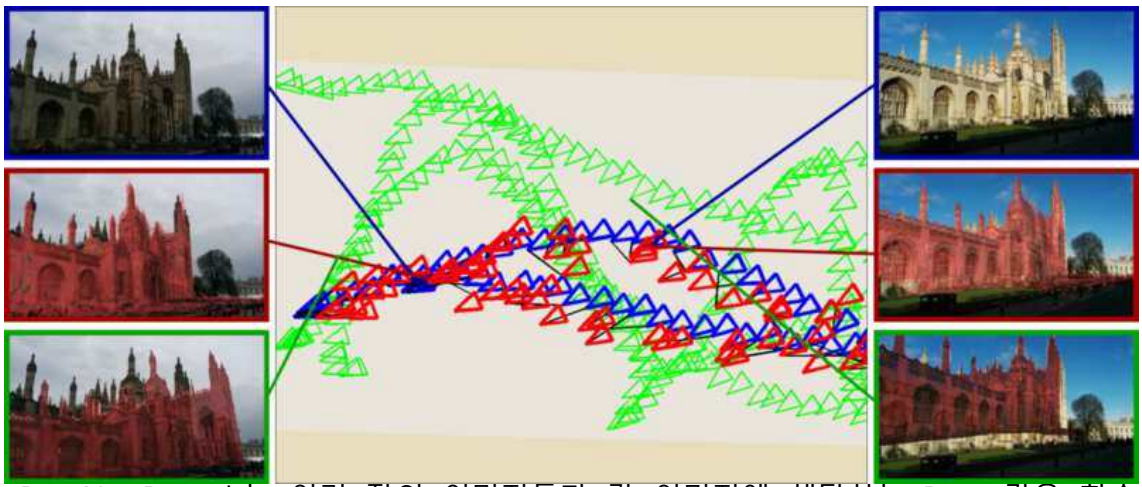
201818689 하창완

코드 Github : https://github.com/cwha0212/Robot-vision-project

1. 서론



Visual Odometry란 카메라를 통해 들어오는 Visual 정보를 활용하여 6-DOF를 구하는 것이다. 여기서 6-DOF는 3차원에서의 총 6가지 운동 방향을 말하며, 3개의 Position(x, y, z)과 3개의 Orientation(Yaw, Pitch, Roll)을 뜻한다. 두 개의 이미지가 있으면, 두 이미지 사이의 Rotation과 Translation을 구하고 이를 계속 누적해가며 각 이미지의 Pose를 구할 수 있다. 이를 활용한 분야로는 로보틱스, 웨어러블, 증강현실, 자율주행 등이 있다. 특히 Visual Odometry는 Wheel Odometry와 비교하였을 때 울퉁불퉁한 지면과 같은 불리한 조건들의 영향을 받지 않는다는 장점이 있다.



PoseNet-Pytorch는 여러 장의 이미지들과 각 이미지에 해당하는 Pose 값을 학습하여 새로운 이미지가 입력으로 들어왔을 때 해당하는 이미지의 Pose 값을 출력한다. 딥러닝 학습 기법의 하나인 Stochastic gradient descent를 활용하여 학습이 오래 걸리는 문제를 개선하였고, 적절한 batch 설정을 통해 조금씩 학습하여 계산해 나아간다. 또한, 학습데이터가 부족한 경우 사용하는 Transfer learning을 통해 이미 학습되어있는 모델을 재사용하여 데이터 부족 문제를 극복하였다.

프로젝트에서는 PoseNet-Pytorch와 Visual Odometry를 활용하여 학습된 위치로부터 어떤 경로로 이동하였는지를 확인할 수 있도록 하였다. 이를 통해 우리는 GPS

정보를 이용한 것이 아닌 오로지 카메라 정보만을 사용하여 어느 지역에서 어떻게 이동하였는지 알 수 있을 것이다.

2. 본론

위의 사진은 사용한 코드들의 종류와 폴더 분류이다. data 폴더에는 이미지 파일과 PoseNet-Pytorch 학습이 완료된 모델이 저장되어있고 result 폴더에는 Visual Odometry를 통해 얻은 Pose 값과 결과 사진이 저장되어있다. 큰 틀에서의 구현 방법은 다음과 같다.

1. Visual Odometry를 이용하여 각 이미지의 Pose 정보를 취득한다.

Visual Odometry의 경우 ORB를 사용하여 Visual Feature 들을 추출하고 bf matcher를 사용하여 Feature matching을 하였다. 이렇게 매칭시킨 점들을 이용하여 Essential Matrix를 생성하고 Pose를 구하였다. 이때, PoseNet-Pytorch 학습을 위해서는 Matrix 형태가 아닌 Quaternion 형태를 사용하여야 하므로 적절하게 변환하여 준다. 여기서 구한 정보는 txt 파일에 저장하고, 누적시켜 Trajectory를 생성한다.

Visual Feature 검출에는 ORB 알고리즘을 사용하였는데, ORB는 (Oriented FAST and Rotated BRIEF)로 FAST, BRIEF, 해리스 코너 알고리즘을 결합한 알고리즘이다.

$$R = \begin{bmatrix} 1-2q_2^2-2q_3^2 & 2q_1q_2-2q_0q_3 & 2q_1q_3+2q_0q_2 \\ 2q_1q_2+2q_0q_3 & 1-2q_1^2-2q_3^2 & 2q_2q_3-2q_0q_1 \\ 2q_1q_3-2q_0q_2 & 2q_2q_3+2q_0q_1 & 1-2q_1^2-2q_2^2 \end{bmatrix}$$

Quaternion이란 3차원 사물의 회전을 표현하는 하나의 방법으로 $q_w\, q_x\, q_y\, q_z$ 4개의 값으로 이루어집니다. 이때, q_w는 스칼라값, $q_x\, q_y\, q_z$는 허수 i j k에 각각 매칭되는 값입니다. 사진은 Quaternion을 Rotation Matrix로 변환하는 식을 보여주는데 $q_0\, q_1\, q_2\, q_3$은 각각 $q_w\, q_x\, q_y\, q_z$를 뜻한다.

$$|q_0| = \sqrt{\frac{1 + r_{11} + r_{22} + r_{33}}{4}}$$

$$q_1 = \frac{r_{32} - r_{23}}{4q_0}$$

$$q_2 = \frac{r_{13} - r_{31}}{4q_0}$$

$$q_3 = \frac{r_{21} - r_{12}}{4q_0}$$

위의 그림은 반대로 Rotation Matrix를 Quaternion으로 변환하는 식을 보여주고 있다.

2. 획득한 Pose 정보와 이미지를 PoseNet-Pytorch 학습을 진행한다.

PoseNet-Pytorch의 경우 ResNet 50을 기반으로 학습을 진행하였다. epochs는 200으로 설정하였으며, 생성된 모델 중 가장 결과값이 좋은 모델인 best_net.pth를 사용하였다.

3. 최종적으로 나온 모델을 활용하여 초기값 Pose 정보를 취득한다.

PoseNet-Pytorch Open source의 argument 들을 변경하고 필요한 부분만 남겨 파일을 만들었다. 그리고 첫 이미지를 입력하여 Pose Estimation을 하였다.

4. 초기값을 지정해두고 Visual Odometry를 진행한다.

첫 이미지의 Pose에 기존 Visual Odometry 코드를 진행하여 각 이미지의 Pose를 구하였다. 1.과 같이 txt 파일에 저장하고 Trajectory를 생성한다.

사용한 코드는 총 6개이다.

1. image_extracrion.py : 영상에서 이미지를 추출하는 코드

```
영상파일을 불러온다.
영상파일을 성공적으로 불러왔는지 확인한다.
while 영상파일이 계속 열려있다면
  do 3번째 프레임마다 영상을 저장한다.
```

2. visual_odom.py : Visual Odometry를 진행하는 코드

```
ORB Feature 추출기를 생성한다.
bf matcher를 생성한다.
이미지 경로에서 이미지 리스트를 추출한다.
이미지 리스트를 정렬하고 첫 번째, 두 번째 이미지를 읽어온다.
이미지에 대한 Feature를 추출한다.
Feature 들을 매칭해준다.
매칭된 점들은 pts1과 pts2에 저장하여준다.
Essential Matrix와 Pose 값을 구해준다.
Pose를 행렬 형태에서 Quaternion 형태로 변환하여준다.
Quaternion 값을 list에 저장한다.
for 이미지 수만큼:
 새로운 이미지를 불러와 Feature를 추출한다.
 바로 이전 이미지와 Feature Matching을 한다.
 매칭된 점들은 pts1과 pts2에 저장하여준다.
 Essential Matrix와 Pose 값을 구해준다.
 Pose를 행렬 형태에서 Quaternion 형태로 변환하여준다.
 Quaternion 값을 list에 저장한다.
 Pose의 x, z 위치를 창에 나타내어준다.
Quaternion 값을 저장할 파일을 만든다.
값을 txt 파일에 모두 저장한다.
```

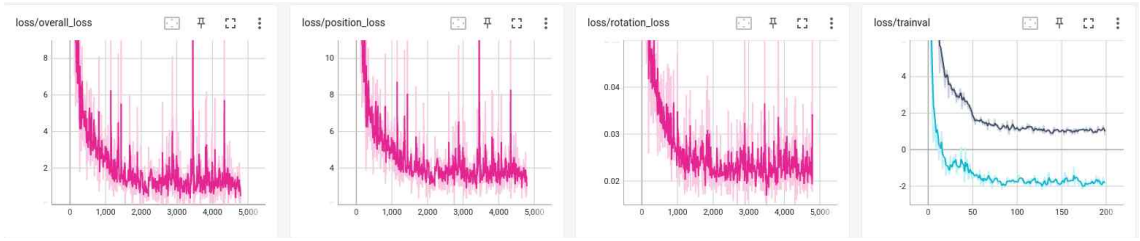
3. data_loader.py : 학습을 진행할 파일을 불러오는 코드
4. model.py : 학습의 기반이 된 model을 불러오는 코드
5. solver.py : 학습을 진행할 파일, 기반이 되는 model과 학습된 model을 이용하여 pose estimation을 진행하는 코드
6. posenet+visual_odom.py : 학습된 데이터를 이용하여 초기값 설정 후 Visual Odometry를 진행하는 코드

```
첫 번째 이미지를 불러온다.
data_loader를 통해 데이터들을 불러온다.
solver를 통해 결과값을 얻어낸다.
결과값(Quaternion)을 Matrix 형태로 변환한다.
```

해당 코드를 visual_odom.py의 line 7 이후 추가했다.

3. 결론

데이터의 경우 7호관의 앞부분을 지나면서 촬영한 데이터와 데이터가 끝나기 두 발자국 전부터 시작하여 주차장을 관통하여 지나면서 촬영한 데이터를 사용하였다. 7호관의 앞부분은 371장의 이미지, 주차장 이미지는 375장이 나왔다. 학습에는 7호관 앞부분 이미지를 사용하였고, 주차장 이미지로 최종 Visual Odometry를 실시하였다.



PoseNet-Pytorch 결과값이다. best_net.pth의 경우 pos오차 1.5, ori오차 0.025 정도로 나왔다.

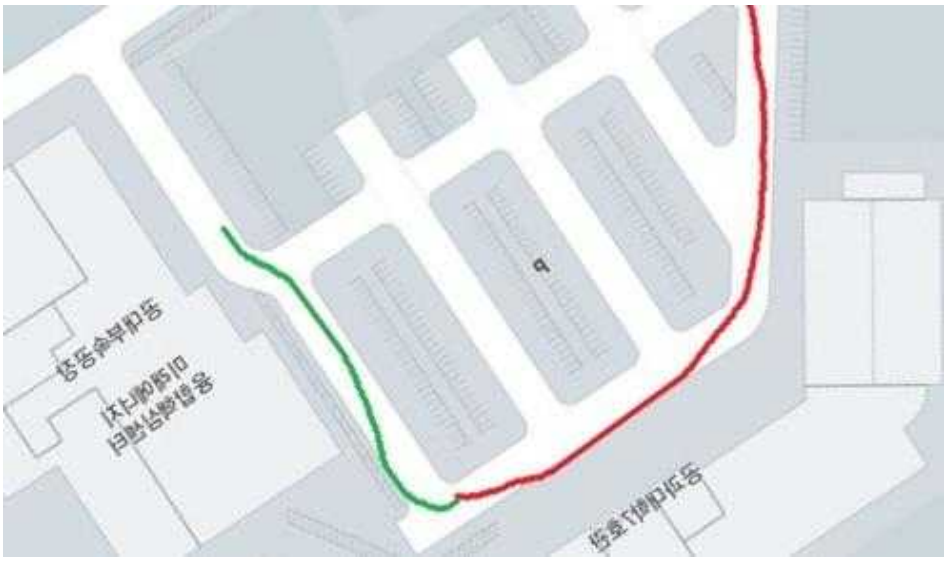
frame00332.jpg -144.71197841366364 -8.854774695738 59 217.92063638476623 0.8454735978143294 0.013510889886166692 -0.5287849580795028 -0.07333702586162885
frame00333.jpg -145.54624717263 29 -9.054225710700628 218.43465479925985 0.8415009885229928 0.008893085194445491 -0.5353389071621595 -0.07217516074778281
frame00334.jpg -146.49074384300567 -8.980212246761797 218.7547298635971 0.8352414992111602 0.007720263438119639 -0.5451274445013669 -0.07175029462895592

7호관 앞 데이터 Pose

frame00000.jpg -145.97046 -8.987721 214.62396 0.8279541 0.013314937 -0.55588025 -0.0728825
frame00001.jpg -146.96935381334927 -8.957427354862848 214.58802642480208 0.8316476241611076 0.012865835466046962 -0.5499323206911746 -0.07596816107316193
frame00002.jpg -147.95530142981175 -8.801416879838596 214.5282934069972 0.8276749562547815 0.012516052670482499 -0.5560283184176489 -0.0750336157975903

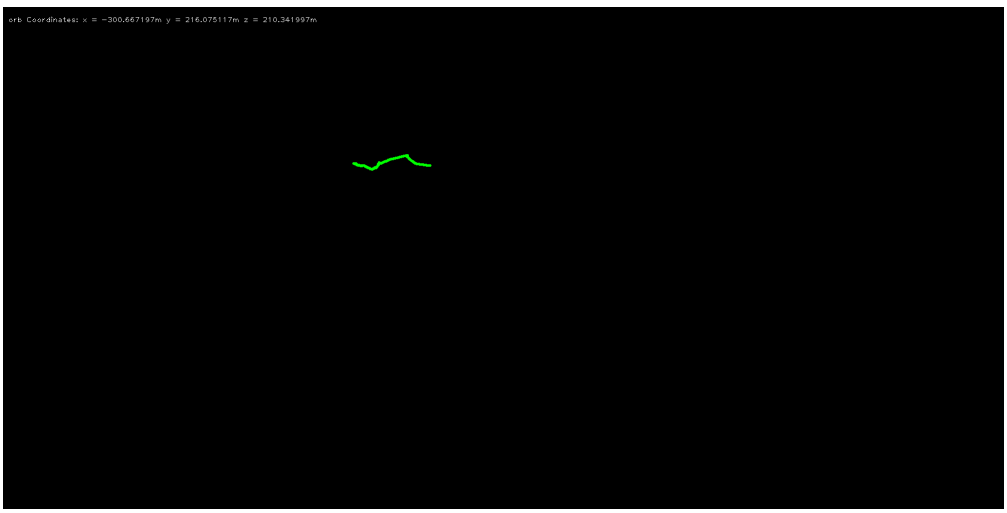
7호관 주차장 데이터 Pose

PoseNet-Pytorch 결과값으로 나온 txt파일의 일부와 이에 해당하는 이미지의 사진들이다. x y z qw qx qy qz값 모두 적절한 값을 취득하였다.



데이터 촬영 시 걸어갔던 경로를 대강 지도에 나타낸 그림이다.

7호관 앞부분에 대한 Viusual Odometry를 진행한 결과이다. 7호관의 앞부분 모양처럼 잘 나왔다.



초기값 설정 후 Visual Odometry를 진행한 모습이다. 7호관 앞부분에 대한 Visual Odometry 결과가 끝나는 지점부터의 위치를 잘 잡아내고 있는 모습이다.
다만, Rotation에 대한 오차 때문인지 전체적인 모양에서는 오차가 어느정도 발생한 것을 확인할 수 있었다.

[참고사이트]
https://github.com/cwha0212/posenet_pkg
https://github.com/youngguncho/PoseNet-Pytorch
https://github.com/Voilier-bsc/mono_vis_odom_python/blob/main/vis_odom.py
https://avisingh599.github.io/vision/visual-odometry-full/
https://danceswithcode.net/engineeringnotes/quaternions/quaternions.html
[참고논문]
PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization