Udacity Self Driving Car Engineer NanoDegree /

Sensor Fusion & Object Detection

Questions to Answer

Write a short recap of the four tracking steps and what you implemented there (EKF, track management, data association, camera-lidar sensor fusion). Which results did you achieve?

For me, it was very difficult to apply theories such as Kalman filter and Mahalanobis to code. So, I organized what I understood for each step and summarized what I learned.

(Kor) 저에게는 kalman filter와 mahalanobis등의 이론을 코드로 응용하는 것이 아주 어려웠습니다. 그래서 각 Step별로 제가 이해한 것을 정리하고 배운 것을 요약하였습니다.

#STEP 1 - EKF :

I first saw the Kalman filter in the mid-term assignment, 3D object detection,

and I didn't really understand it at the time. However, I learned more about it in this assignment. I studied the roles of the F, Q, Predict, Update, Gamma, and S functions in the Filter.py file. The F function returns a system matrix, which represents the state change (based on a constant velocity model) of the system.

The matrix given by the mentor omits the time-varying element for the Z coordinate, so the need for Z-direction data is set low, so I understood that it is suitable for initial debugging and performance optimization.

Q returns a process noise covariance matrix, which represents the uncertainty of the prediction process,

and Predict predicts the state (x) and covariance (p) to the next point in time and updates track.x and track.p. I confirmed that the Update function updates the covariance based on the measurement value, and in the Kalman Filter, the residual (gamma), covariance (s), and Kalman

gain (k) are calculated to modify the state and covariance. K, or Kalman gain, reflects the relative reliability of the measured value (sensor data) and the predicted value (model data). We learned that when the K value is large, the measured value is more reliable, and when the K value is small, the predicted value is more reliable.

The gamma function is a function that calculates the residual (measured value - predicted value = difference), and S calculates the covariance matrix of the residual.

However, we saw that P (covariance matrix) is used once each in the predict, update, and S functions.

In the Predict function, the meaning of P = F·P·FT+Q is used to reflect the system uncertainty when predicting the next state, and F is the state transition, Q is the process noise (system uncertainty), and in other words, P is used in the prediction step to calculate the uncertainty of the next state.

And we also learned that since this step is predicting the future state, there is a possibility that P, or uncertainty, may increase.

The role of P in the Update function is to reduce covariance and correct the state. P=(I−K·H)·P:

K** is the Kalman gain that adjusts the reliability of the measured value and the predicted value.

In the update step, the state is compensated using the residual γ(gamma) and K, and P is reduced to reflect the uncertainty of the compensated state.

In other words, the key is that the update step "compensates the predicted state with the measured value",

so it reduces the uncertainty of the prediction by reducing P.

In the S( ) function, P is used to calculate the residual covariance.

S = H*P* HT + R

S is the covariance matrix of the residual gamma, which means the result of combining the uncertainty of the measured value and the uncertainty of the predicted value, and in other words, S can be said to be the reliability of the residual that represents the difference between the predicted value and the measured value.

P is the covariance matrix of the current state in this equation, and this value is used to calculate

K(Kalman gain), and determines the compensation accuracy in the update step. Here, P is the covariance matrix of the current state.

In other words, P is the key to tracking. P quantitatively expresses the uncertainty about the state of the system, and in the prediction phase, the uncertainty increases, and in the update phase, it is compensated by the measured value and the uncertainty is reduced.

We learned that this plays a key role in efficiently tracking objects.

Ultimately, this P is also used for track management.

(kor) 저는 mid term과제였던 3D object detection에서부터 kalman 필터를 처음보게 되었는데,

그 당시에는 거의 이해하지 못했었습니다. 하지만 이번 과제에서 그것에 대해서 더 알게 되었습니다. Filter.py파일에서 F ,Q ,Predict, Update ,Gamma, S함수의 역할에 대해서 공부하게 되었습니다. F함수는 시스템 행렬을 반환, 즉 시스템의 상태변화(등속 모델기반)을 나타내며

멘토가 주신 행렬은 Z좌표에 대한 시간변화요소가 생략되어 있으므로 Z방향 데이터의 필요성이 낮게 설정되어 있어서, 초기 디버깅과 성능 최적화에 적합함으로 이해했습니다.

Q는 프로세스 노이즈 공분산 행렬을 반환하며 예측과정의 불확실성을 나타내고

Predict는 상태(x)와 공분산(p)를 다음 시점으로 예측하며 track.x와 track.p를 업데이트함을 확인하였습니다. Update함수는 측정값을 기반으로 공분산을 업데이트 함을 확인했고, kalman Filter에서 잔차(gamma)와 공분산(s) 그리고 칼만이득(k)를 계산하여 상태와 공분산을 수정합니다.

K 즉,칼만이득은 측정값(Sensor Data)와 예측값(모델 데이터)의 상대적인 신뢰도를 반영하는 것으

로, K값이 크면 측정값을 더 신뢰, K값이 작으면 예측값을 더 신뢰한다는 것을 알게 되었습니다.

gamma함수는 잔차(측정값 – 예측값 = 차이)를 계산하는 함수이며 S는 잔차의 공분산 행렬을 계산함을 확인했습니다.

다만 여기서 P(공분산 행렬)이 predict, update, S함수에서 각각 한번씩 사용되는 것을 보았는데

Predict 함수에서 $P=F \cdot P \cdot FT+Q$ 의미는 다음 상태를 예측할 때의 시스템 불확실성을 반영하는데 사용되며, F는 상태전이, Q는 프로세스 노이즈(시스템의 불확실성)이고 즉, 예측단계에서 P를 사용하여 다음 상태의 불확실성을 계산해주는 것입니다.

그리고 이 단계에서는 미래상태를 예측하는 것이므로 P즉 불확실성은 늘어날 수도 있다는 가능성이 있다는 것도 배웠습니다.

Update함수에서의 P의 역할은 공분산 감소 및 상태보정입니다.

P=(I−K·H)·P:

K**는 측정값과 예측값의 신뢰도를 조율하는 칼만 이득입니다.

업데이트 단계에서는 잔차 γ(gamma)와 K를 사용하여 상태를 보정하며, 보정된 상태의 불확실성을 반영하여 P를 줄입니다.

즉, 핵심은 업데이트 단계는 "예측된 상태를 측정값으로 보정"하므로

P를 감소시켜 예측의 불확실성을 줄입니다.

S( )함수에서는 P는 잔차 공분산 계산으로 사용됩니다.

$S = H*P* H^T + R$

S는 잔차 gamma의 공분산 행렬로 측정값의 불확실성과 예측값의 불확실성을 결합한 결과를 의미하고 즉, S는 예측값과 측정값 간의 차이를 나타내는 잔차의 신뢰도라고 할 수 있습니다.

P는 이 식에서 현재 상태의 공분산 행렬이며, 이 값은 K(칼만이득)계산에 사용되며, 업데이트 단계에서 보정 정확도를 결정한다는 것 여기서 P는 현재 상태의 공분산 행렬입니다.

즉 P는 추적의 핵심입니다.

P는 시스템의 상태에 대한 불확실성을 정량적으로 표현하며, 예측단계에서는 불확실성이

증가하고, 업데이트 단계에서는 측정값으로 보정되어 불확실성이 감소됩니다.

이는 물체를 효율적으로 추적하는 데 핵심적인 역할을 한다는 것을 배웠습니다.

결국 이 P는 track관리에도 사용이 됩니다.

#STEP 2 – Track Management :

The P used in Filter.py is also used importantly in Step 2. The P value passed in the Filter.py file is used as a criterion for evaluating the stability of the track in the Trackmanagement.py file, and for deleting the track if it does not satisfy a certain condition.

In the initialization function of the track class in initialization (__init__), P is set to the initial state covariance matrix of the track. The three lines of code below reflect this.

self.P = np.zeros((6,6))

self.P[0:3, 0:3] = P_pos

self.P[3:6, 3:6] = P_vel

In addition, in the manage_tracks function, P is used as a criterion for determining whether the track will be deleted.

If the diagonal element of the covariance matrix is greater than params.max_P, the track will be deleted. if track.P[0, 0] > params.max_P or track.P[1, 1] > params.max_P:

delete_tracks.append(track)

It is also used in the handle_updated_track function.

print(f"Track ID {track.id} - Score: {track.score}, State: {track.state}, P[0,0]: {track.P[0,0]}, P[1,1]: {track.P[1,1]}")

That is, the P value is output as debugging information related to state transition.

The meaning and role of the covariance P represents the uncertainty of the tracking state of the track.

P[0, 0] and P[1, 1] represent the uncertainty of the position (x, y) dimension, respectively, and the larger the value, the lower the prediction reliability of the track.

If P is too large in manage_tracks, the track is judged to be unreliable and is deleted.

In filter.py, P updated through predict and update is used to evaluate the track status in trackmanagement.py. The two files are closely linked, and the update of P in filter.py directly affects the maintenance/deletion of the track in trackmanagement.py.

Ultimately, throughout step1 and step2, P is an important criterion for evaluating the reliability of the track.

And the score (track score) set in the trackmanagement.py file itself is an important factor in evaluating the reliability of the track. The handle_updated_track function increases the score and changes the state according to the score.

The manage_track function receives the changed score and state and determines whether to delete the track along with the P passed from filter.py.

Delete old tracks

```
# 오래된 트랙 삭제

        delete_tracks = []

        #멘토님 두번째 코드

        for i in range(len(self.track_list)):

            track = self.track_list[i]

            if track.state == 'confirmed' and track.score < params.delete_threshold:

                delete_tracks.append(track)

                continue

            if track.score < 0.15:

                delete_tracks.append(track)

                continue

            if track.P[0, 0] > params.max_P or track.P[1, 1] > params.max_P:

                delete_tracks.append(track)
```

In this way, we have confirmed that Filter.py and trackmanagement.py of STEP1 and STEP2 are connected to each other (organically) to track objects and manage tracks. However, unfortunately, the RMSE Plot is not output in step2 yet. We plan to debug it and receive feedback after submitting the assignment to revise it. Also, I still don't fully understand the Kalman filter, so I plan to study it further after the project.

(kor) Filter.py에서 사용된 P는 Step 2에서도 중요하게 사용됩니다. Filter.py파일에서 넘겨준P값은 Trackmanagement.py파일에서 공분산 행렬 P는 트랙의 안정성을 평가하고, 특정 조건을 만족하지 못할 경우 트랙을 삭제하는 기준으로 사용됩니다.

초기화(__init__)에서 track 클래스의 초기화 함수에서 P는 트랙의 초기 상태 공분산행렬로 설정됩니다. 아래의 3줄의 코드가 그 반영입니다.

self.P = np.zeros((6,6))

self.P[0:3, 0:3] = P_pos

self.P[3:6, 3:6] = P_vel

또한 manage_tracks함수에서 P는 트랙이 삭제 될지를 결정하는 기준으로 사용됩니다.

공분산 행렬의 대각요소가 params.max_P보다 클 경우 해당 트랙은 삭제 됩니다.

if track.P[0, 0] > params.max_P or track.P[1, 1] > params.max_P:

    delete_tracks.append(track)

또한 handle_updated_track함수에서도 활용됩니다.

print(f"Track ID {track.id} - Score: {track.score}, State: {track.state}, P[0,0]: {track.P[0,0]}, P[1,1]: {track.P[1,1]}")

즉, P값은 상태전환과 관련된 디버깅 정보로 출력이 됩니다.

공분산 P의 의미와 역할은 트랙의 추적상태에 대한 불확실성을 나타냅니다.

P[0, 0]과 P[1, 1]은 각각 위치(x, y) 차원의 불확실성을 나타내며, 값이 클수록 트랙의 예측 신뢰도가 낮아짐을 의미합니다.

manage_tracks에서 P가 너무 큰 경우 해당 트랙은 신뢰할 수 없다고 판단되어 삭제됩니다.

Filter.py에서 predict와 update를 통해 업데이트된 P는 trackmanagement.py에서 트랙상태를 평가하는데 사용됩니다. 두 파일은 서로 밀접하게 연결되어 있으며 filter.py에서의 P의 갱신이 trackmanagement.py에서의 트랙 유지/삭제에 직접적인 영향을 미칩니다.

결국 step1과 step2에 걸쳐서, P는 트랙의 신뢰도를 평가하는 중요한 기준입니다.

그리고 trackmanagement.py파일 자체에 설정된 score(트랙점수)는 트랙신뢰도 평가의 중요 요소입니다. handle_updated_track함수에서 점수가 증가하고 점수에 따라 state를 변경합니다.

그 변경된 score와 state를 manage_track함수에서 받아서 filter.py에서 넘겨준 P와 함께 트랙삭제 여부를 결정합니다.

```
# 오래된 트랙 삭제
    delete_tracks = []
    #멘토님 두번째 코드
    for i in range(len(self.track_list)):
        track = self.track_list[i]
```

```
if track.state == 'confirmed' and track.score < params.delete_threshold:
    delete_tracks.append(track)
    continue
if track.score < 0.15:
    delete_tracks.append(track)
    continue
if track.P[0, 0] > params.max_P or track.P[1, 1] > params.max_P:
    delete_tracks.append(track)
```

이와 같이 STEP1과 STEP2의 Filter.py와 trackmanagement.py가 서로 연결되어서(유기적으로)

물체를 추적,트랙을 관리하게 됨을 확인하였습니다. 하지만 아쉽게도 아직 step2에서 RMSE Plot출력되고 있지 않습니다. 디버깅을 할 계획이며, 과제제출 후 feedback을 받아서 수정하려고 합니다.

또한 아직 칼만필터에 대해서 완전히 이해한 것은 아니기 때문에 프로젝트 이후에도 더 공부하려고 합니다.

#STEP 3 – Data Association :

In this step, by configuring the association.py file, it was well organized to learn how to effectively handle realistic data problems such as Mahalanobis distance and gating to associate tracks and measurements, the effect of data association in a multi-target tracking environment, and ghost tracks.

We learned how to complete the Data Association by configuring each function in the file.

The associate function calculates the Mahalanobis between track_list and meas_list, applies gating (threshold), and determines whether to associate measurements and tracks. We confirmed the logic of storing the association results in the association_matrix and writing unassociated tracks to unassigned_tracks and unassigned_meas.

In the get_closet_track_and_meas function, we configured the logic to find the closest track and measurement and return it (return track_ind, meas_ind).

In the gating function, we configured the code to associate tracks and measurements if the Mahalanobis distance does not exceed the threshold. In the MHD function, the Mahalanobis distance is calculated to quantify the relationship between the track and the measurement value.

Finally, in the associate_and_update function, all of the above functions are integrated to update the track status, update the status and score of the related track, and configure a structure to manage unrelated items.

I still don't understand Mahalanobis well, so I will study it more later.

(Kor) 이 단계에서는 association.py파일을 구성함으로써, Mahalanobis 거리와 gating을 활용한 트랙과 측정값 연관, 다중 표적 추적 환경에서 데이터의 연관의 효과, 고스트트랙과 같은 현실적인 데이터 문제를 효과적으로 처리하는 방법을 학습하도록 잘 짜여져 있었습니다.

파일내에 각 함수들을 구성함으로써 Data Association을 완성하는 것을 배웠는데,

associate함수는 track_list와 meas_list간의 Mahalanobis를 계산하고 gating(문턱값)을 적용하여 측정값과 트랙을 연결시키질 결정하고, 연관결과를 association_matrix에 저장시키고 연관되지 않은 트랙은 unassigned_tracks와 unassigned_meas에 기록하는 로직을 확인했습니다

get_closet_track_and_meas함수에서는 가장 가까운 트랙과 측정값을 찾아 반환(return track_ind, meas_ind)하는 로직을 구성하였고

gating함수에서는 Mahalanobis거리가 임계값을 초과하지 않는 경우 트랙과 측정값을 연관시키는 코드를 구성하였습니다.

MHD함수에서는 mahalanobis거리를 계산하여서 트랙과 측정값간의 관계를 수치화 하는 것을 구성하여습니다.

최종적으로 associate_and_update함수에서는 위의 모든 기능을 통합하여서 트랙상태를 업데이트하고 연관된 트랙의 상태 및 점수를 업데이트하고 연관되지 않은 항목은 관리하는 구조를 구성했습니다.

지금도 mahalanobis에 대해서는 잘 이해하지 못하였기 때문에 추후 더 공부해보겠습니다

#STEP 4 – camera-lidar sensor fusion :

In this step, we learned how to fuse lidar and camera data by configuring the measurements.py file,

updating the track using camera data, completing the sensor fusion model, and finally

visualizing the RMSE to check if the track is tracked and if there is a ghost track.

That is, it was a process of learning how to utilize sensor fusion (Lidar + camera) and Jacobian, and ultimately implementing the integration of multiple sensors to lower the RMSE and solve the ghost track problem.

In the get_hx function, we configured the logic to nonlinearly transform the camera data from the vehicle coordinate system to the sensor coordinate system and project it to the image coordinate system, and in the get_H function, we calculated the Jacobian matrix and provided a linear approximation to the get_hx function. Instead of directly using the H value returned from the get_H function in the get_hx function, the get_H function is called in the update function in the filter.py file, and the get_hx function is called in the gamma function in the filter.py file to calculate gamma and return it again, and this gamma function is called in the update function to calculate the hx variable calculated in gamma, i.e. h(x). Through this project, I learned important algorithms and implementation methods at each stage, such as EKF, track management, data association, and sensor fusion. In particular, I learned how to understand the role of the covariance matrix and the prediction and update principles of the Kalman filter, and how to obtain more reliable tracking results by integrating multi-sensor data. It was difficult to implement mathematical theory into code and understand the sophisticated intention of the system where multiple files interact with each other, but it was a very helpful task.

(Kor) 이 단계에서는 measurements.py파일을 구성함으로써 라이다와 카메라 데이터를 융합하고

카메라데이터를 이용해 트랙을 업데이트하며, 센서융합모델을 완성하며, 또 최종적으로

RMSE의 시각화를 통해 트랙이 추적되고 고스트 트랙이 있는지 확인하는 것을 배웠습니다.

즉,센서융합(Lidar + camera)와 Jacobian을 활용하기, 그리고 결국 최종적으로 여러 센스를 통합하여 RMSE를 낮추고 Ghost track 문제를 해결하는 것을 구현하는 것을 배우는 과정이었습니다.

Get_hx함수에서 카메라 데이터를 비선형 변환하여 차량좌표계에서 센서 좌표계로 변환하고 이미지 좌표계로 투영하는 로직을 구성했고 get_H함수에서 Jacobian 행렬을 계산하여 get_hx함수에 선형 근사를 제공하였습니다. Get_H함수에서 return하는 H값을 get_hx함수에서 바로 가져다 쓰지는 않고 filter.py파일내의 update함수에서 get_H함수를 호출하였고,또한 filter.py 파일내에 gamma 함수에서 get_hx함수를 호출하여서 계산후 gamma를 다시 return하고 이 gamma함수를 update

함수에서 호출하여서 gamma내에서 계산된 hx변수 즉,h(x)를 계산합니다.

저는 이번 프로젝트를 통해 EKF, 트랙 관리, 데이터 연관, 센서 융합 등 각 단계에서 중요한 알고리즘과 구현 방법을 학습했습니다. 특히, 공분산 행렬의 역할과 칼만 필터의 예측 및 업데이트 원리를 이해하며, 다중 센서 데이터를 융합해 보다 신뢰도 높은 추적 결과를 얻는 방법을 배웠습니다. 수학적 이론을 코드로 구현하고 여러 파일이 서로 상호작용하는 시스템의 정교한 의도를 이해하기가 어려웠습니다만 큰 도움이 된 과제였습니다.

Which part of the project was most difficult for you to complete, and why?

It was very difficult to understand the Kalman Filter, implement it in code, and understand the system in which each file interacts with each other.

In addition, in steps 1 to 3, the update file of filter.py operates only with Lidar data, but in step 4, it was difficult to understand that filter.py processes not only Lidar but also camera data using the object called meas.sensor.

To be honest, I still can't say that I understand it 100%. So even after submitting the assignment, I plan to study more about theories such as Kalman filter and Jacobian matrix, and the interaction of each file.

(Kor) Kalman Filter에 대한 이해와 그것을 code로 구현하는 것, 각 파일들이 서로 상호작용하도록 연결되어 있는 시스템에 대해서 이해하는 것이 무척 어려웠습니다.

또한 step 1~3까지는 Lidar 데이터만으로 filter.py의 update파일이 동작하는데, step4에서 filter.py가 meas.sensor라는 객체를 Lidar뿐만 아니라 camera 데이터도 처리하게 된 다는 것이 이해하기 어려웠습니다.

솔직히 지금도 100%이해했다고는 말할 수 없습니다. 그래서 과제 제출후에도 칼만필터와 jacobian행렬등의 이론과 각 파일들의 상호작용에 대해서 더 공부하려고 합니다

Do you see any benefits in camera-lidar fusion compared to lidar-only tracking (in theory and in your concrete results)?

Steps 1~3 were performed only with Lidar data, and Steps 3 and 4 were performed with the camera. In the filter.py file, the predict and update functions only used Lidar data.

In Step 4, the camera data is finally used, and in the init function in the class sensor in measurements.py, the camera data is received with the code elif name == 'camera', and the get_hx and get_H functions activate camera processing with the condition self.name =='camera', and the in_fov function is implemented as a logic that only processes camera data.

In the end, camera data processing is activated in Step 4, which is implemented with the processing process above.

If you compare the result images of Step 3 and Step 4, they seem similar. The detected vehicles and the initialized tracks indicated in red also look similar. Since the Lidar data itself is already accurate, it does not seem that using the camera together will detect more vehicles in the image. The RMSE values that are finally plotted are the same for Step 3 and Step 4. If it was supplemented with a camera,

there should have been a clear change, but I guess it seems that there is no difference because the Lidar data provided by default was of good quality.

However, in step 3, the processing results for each frame are finished at around 5~20,

while in step 4, it shows a high score of 10~30. In step 3, there are cases where the score is 30, but I observed that it tends to be more frequent in step 4.

If it were possible to output the score, I guess we would be able to confirm that step 4 is improved compared to step 3.

As a result, the results of step 3 and step 4 may have been similar because the environment of the given data is a good environment of a clear day and not a very complex environment. And the RMSE results of step 3 and step 4 are almost the same, but this seems to be because the provided Lidar data is of high quality. However, the fact that the score tends to be higher in step 4 indirectly shows that the track reliability has improved due to camera-lidar fusion. In particular, in real environments, camera data can provide visual information that is difficult to supplement with Lidar, so we expect greater benefits in complex road environments in the future. In particular,

we expect cameras to be much more powerful than Lidar in detecting people (pedestrians).

(Kor) Step 1~3에서는 Lidar 데이터만으로 진행했고 Step 3,4에서는 camera와 함께 진행되었습니다. Filter.py파일에서 predict와 update함수는 Lidar데이터만을 활용했습니다.

Step4에서 드디어 camera 데이터가 사용이 되는데 measurments.py에 class sensor내의 init함수에서 elif name == 'camera' 코드로 camera data를 받고 get_hx함수와 get_H함수는 self.name =='camera' 조건으로 camera 처리를 활성화하고, in_fov함수는 camera 데이터만을 처리하게 작동하는 로직으로 구현되어 있습니다.

결국 위의 처리과정으로 구현된 step4에서 camera데이터 처리가 활성화 됩니다.

Step3의 결과영상과 step4의 결과 영상을 비교해보면 유사한 듯 합니다. 검출해 내는 차량이나 red색상으로 표기되는 initialized track도 비슷해 보입니다. 이미 Lidar데이터 자체가 정확하기 때문에 camera를 함께 사용한다고 해서, 영상에서 더 많은 차량을 검출해낸다거나 하지는 않는것으로 보입니다. 최종적으로 Plot 되는 RMSE값도 step3와 step4가 동일합니다. Camera로 보완했다면

분명히 변화가 있어야 할텐데, 제 추측으로는 기본적으로 제공된 Lidar데이터가 좋은품질이었기 때문에 이렇게 차이가 없는듯 합니다.

다만 step3에서는 각 frame별 processing결과를 보면 score가 5~20정도에 마무르는 반면,

Step4에서는 10~30등 높은 score를 보이고 있습니다. Step3에서도 부분적으로 score가 30이 되는 경우들이 있지만 step4에서 더 많이 그런 경향이 보이는 것을 관찰하였습니다.

만약 score에 대해서 출력할 수 있다면, step3보다 step4가 향상된 것을 확인할 수 있을 것이라고 추측합니다.

결과적으로 주어진 data의 환경이 맑은 날의 좋은 환경이고 그다지 복잡한 환경이 아니기 때문에

Step3와 step4의 결과가 비슷하게 나왔을 수가 있습니다. 그리고 Step 3과 Step 4의 RMSE 결과는 거의 동일하지만, 이는 제공된 Lidar 데이터가 고품질이기 때문으로 보입니다. 그러나 Step 4에서 score가 더 높은 경향을 보이는 점은 Camera-Lidar 융합으로 인해 트랙 신뢰도가 향상되었음을 간접적으로 보여줍니다. 특히, 실제 환경에서는 Camera 데이터가 Lidar로 보완하기 어려운 시각적 정보를 제공할 수 있으므로, 향후 복잡한 도로 환경에서 더 큰 이점을 기대할 수 있을 것으로 생각합니다. 특히 사람(보행자)를 검출해 내는데는 camera가 Lidar보다는 훨씬 강력할 것으로 예상합니다.

Which challenges will a sensor fusion system face in real-life scenarios? Did you see any of these challenges in the project?

This project was conducted in a relatively simple environment and did not cover all the complex real-life challenges mentioned above. However, some issues were indirectly confirmed:

Since the lidar data was very accurate, it seemed that the camera data did not improve the tracking, which is probably due to the very clear daytime environment. I think that the data input by the camera will greatly improve the performance in various environments such as at night, when it is raining, and for tasks such as distinguishing between pedestrians, bicycles, and motorcycles. Also, although the results of the fusion between sensors were good, since the autonomous driving system that operates in real-time is not yet available, I think that there may be some system delays when applying these codes to hardware.

In conclusion, since this project was conducted based on stable sensor data, it did not cover all the problems that can occur in a real complex environment. However, I was able to confirm the benefits of sensor fusion and learn how the theoretical basis (Kalman filter) can be utilized to solve real-life problems.

(Kor) 이 프로젝트에서는 비교적 단순한 환경에서 진행되었으며, 위와 같은 복잡한 실생활 도전 과제를 모두 다루지는 않았습니다. 그러나 일부 문제를 간접적으로 확인할 수 있었습니다:

라이다 데이터가 매우 정확했으므로, 카메라 데이터가 tracking 을 향상시키지는 못하는 것으로 보였는데, 그것은 매우 깨끗한 주간환경 때문으로 판단됩니다. 야간, 폭우가 내릴 때 등의 다양한 환경과 보행자,자전거와 모터사이클 구분하는 것과 같은 일에는 카메라로 입력된 데이터들이 큰 성능 향상을 도울 것이라고 생각합니다. 또한 센서 간의 융합 결과는 양호했으나, Real-time 으로 작동하는 자율주행시스템이 아직은 아니므로, 이 코드들을 하드웨어에 응용할 경우 어떤 시스템의 지연(delay) 생길수도 있다고 생각합니다.

결론적으로, 이 프로젝트는 안정적인 센서 데이터를 기반으로 진행되었기 때문에, 실제 복잡한 환경에서 발생할 수 있는 문제를 모두 다루지는 않았습니다. 그러나 센서 융합의 이점을 확인하고, 이론적 기반(kalman filter)이 실생활 문제를 해결하는 데 어떻게 활용될 수 있는지 배울 수 있었습니다.

Can you think of ways to improve your tracking results in the future?

In this project, I used precomputed detection provided by Udacity. The provided dataset was great and I could see good performance except for STEP2.

However, I would like to improve the task in the following ways:

Integrating lane detection and YOLO:

In my previous Udacity project (self-driving car Nano degree course in 2020), I did lane detection for object detection and personally implemented YOLO. Integrating these techniques into the current tracking framework can help improve detection accuracy by adding complementary information. For example, lane detection can help filter out objects outside the drivable area and YOLO can be used to detect objects that the lidar may miss, such as pedestrians or small objects. I think that in real driving environments, nighttime environments and situations where pedestrians, bicycles, and motorcycles appear can significantly affect the performance of the lidar sensor. I think that by leveraging the camera data enabled in step 4 and combining it with other techniques such as YOLO, we can solve this task more effectively. Parameter Tuning:

I think fine-tuning the parameters in params.py can reduce the RMSE and slightly improve the stability of the tracking.

Experiment with Data Association Algorithm:

Currently, this project uses a simple Nearest Neighbor Association method. I plan to apply more modern algorithms such as Joint Probabilistic Data Association (JPDA) or Global Nearest Neighbor (GNN).

Challenge Kalman Filter Performance:

Tuning the Kalman filter to estimate additional properties such as object size (width, length, height) could be another way to improve the tracking performance. I don't think it will give dramatic results, and I don't fully understand the implementation yet, but I think it will allow for slightly improved tracking.

Conclusion:

I am still learning the theory and practice of sensor fusion and tracking, but this project motivates me to think about how to integrate techniques I have done in the past such as lane detection and YOLO.

But for now, I will just use the configuration provided in the course and focus on understanding and improving the basics such as parameter tuning and effective data association. Since this is a curriculum created by industry geniuses, I think it is important to first understand and follow their intentions.

(Kor) 이 프로젝트에서 저는 Udacity 에서 제공하는 사전 계산된 탐지를 사용했습니다. 제공된 데이터 세트는 훌륭했고, STEP2 를 제외하고는 좋은 성능을 볼 수가 있었습니다.

하지만 저는 아래와 같은 방법으로 과제를 개선하고자 합니다.

차선 감지 및 YOLO 통합 :

이전 (2020 년의 self driving car Nano 학위과정) Udacity 프로젝트에서 객체 감지를 위해 차선 감지를 수행했고 개인적으로 YOLO 를 구현했습니다. 이러한 기술을 현재 추적 프레임워크에 통합하면 보완 정보를 추가하여 탐지 정확도를 높일 수 있습니다. 예를 들어, 차선 감지는 주행 가능 영역 외부의 객체를 필터링하는 데 도움이 될 수 있으며 YOLO 는 보행자나 작은 객체와 같이 라이더가 놓칠 수 있는 객체를 감지하는 데 사용될 수 있습니다. 실제 주행환경에서 야간 환경과 보행자, 자전거, 모터사이클이 등장하는 상황은 라이다 센서 성능에 상당한 영향을 미칠 수 있다고 생각합니다. 4 단계에서 활성화한 카메라 데이터를 활용하고 YOLO 와 같은 다른 기술과 결합함으로써 이러한 과제를 보다 효과적으로 해결할 수 있다고 생각합니다.

매개변수 튜닝:

params.py 의 매개변수를 미세 조정하면 RMSE 를 줄이고 추적의 안정성을 약간은 개선할 수 있다고 생각합니다.

데이터 연관 알고리즘 실험:

현재 이 프로젝트는 간단한 최근접 이웃 연관 방법을 사용합니다. 저는 Joint Probabilistic Data Association(JPDA) 또는 Global Nearest Neighbor(GNN)와 같은 좀 더 최신의 알고리즘을 적용해 볼 계획입니다.

칼만 필터 기능 향상에 도전:

객체 크기(너비, 길이, 높이)와 같은 추가 속성을 추정하도록 칼만 필터를 조정하는 것은 추적 성능을 개선하는 또 다른 방법이 될 수 있습니다. 극적인 결과를 주지는 않으리라 생각합니다만, 그리고 아직 구현을 완전히 이해하지는 못했지만, 이를 통해 약간은 향상된 추적이 가능하리라고 생각합니다.

결론:

저는 아직 센서 융합 및 추적의 이론과 실무를 배우고 있지만, 이번 프로젝트는 제가 과거에 수행했던 차선 감지 및 YOLO 등의 기술을 통합하는 방법에 대해 생각하도록 동기를 부여합니다.

하지만 현재로서는 과정에서 제공하는 구성을 그대로 사용하고 매개변수 튜닝 및 효과적인 데이터 연결과 같은 기본 사항을 이해하고 개선하는 데 중점을 두고 있습니다. 업계의 천재들이 만들어낸 교육과정이므로 먼저 그들의 의도를 이해하고 따라가는 것이 중요하다고 생각합니다.

Summary of results for each step

Project - Step 1:

Goal :

- **Tracking single object lidar data over time using the Extended Kalman Filter (EKF)**
- **Implemented to ensure that the average RMSE is less than 0.35 in the RMSE plot**

I Realized that below setting is the main intention.

exec_tracking = ['perform_tracking']: Activating tracking with EKF.

exec_visualization = ['show_tracks']: Visualize Tracking results

configs_det = det.load_configs(model_name='fpn_resnet'): Using ResNet Model .

in this step 1, Filter.py file is important. The core of EKF logic.

##Filter.py : In the predict() function, the system matrix, F and the process noise covariance Q are calculated and the state and covariance are updated.

##Measurements.py : function get_hx() , get_H() is important. This file dealing with measurements data.

##Objdet_detect.py: Using ResNet model , processing radar data.

Results:

EKF prediction works well.

The below image shows RMSE is under 0.35.

(Kor) 아래 설정이 주요 의도라는 것을 알았습니다.

exec_tracking = ['perform_tracking']: EKF로 추적 활성화.

exec_visualization = ['show_tracks']: 추적 결과 시각화

configs_det = det.load_configs(model_name='fpn_resnet'): ResNet 모델 사용.

이 1단계에서는 Filter.py 파일이 중요합니다. EKF 논리의 핵심입니다.

##Filter.py: predict() 함수에서 시스템 행렬 F와 프로세스 잡음 공분산 Q가 계산되고 상태와 공분산이 업데이트됩니다.

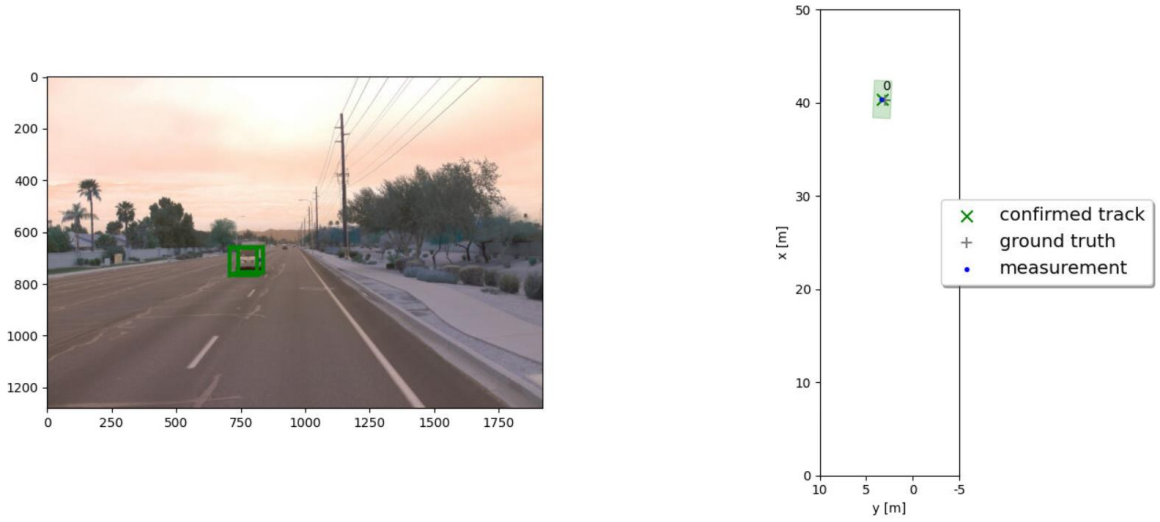
##Measurements.py: get_hx(), get_H() 함수가 중요합니다. 이 파일은 측정 데이터를 처리합니다.

##Objdet_detect.py: ResNet 모델을 사용하여 레이더 데이터를 처리합니다.

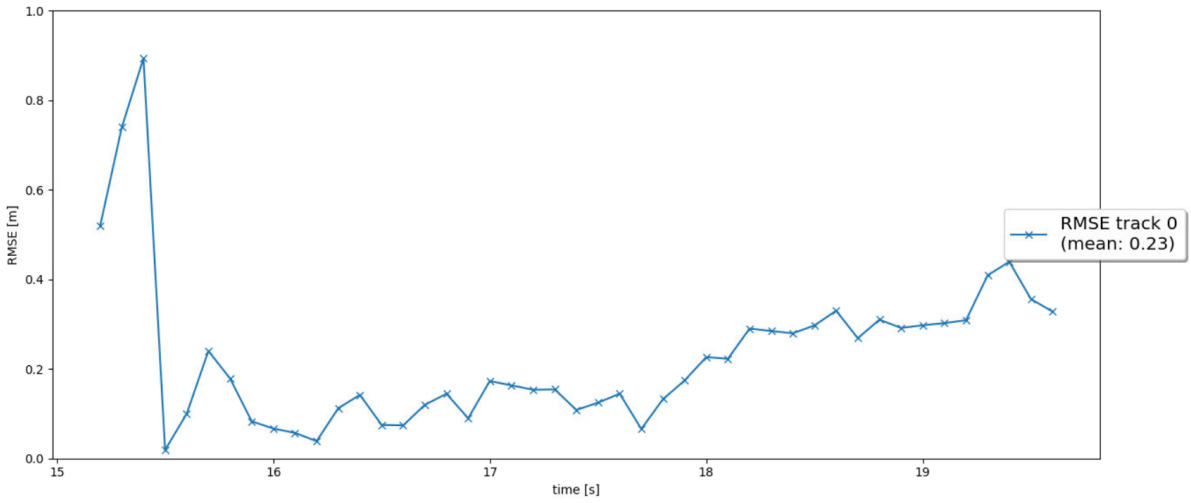
결과:

EKF 예측이 잘 작동합니다.

아래 이미지는 RMSE가 0.35 미만임을 보여줍니다.



(Step 1 Tracking Screen)



(Step 1 RMSE Plot)

Project - Step 2:

Goal :

- **Initialize Track to single object :**

  initialized → tentative → confirmed (Deletion management)

- **Plot RMSE with Single object :**

  RMSE plot should be single graph

##Trackmanagement.py : main logic (initialize track, manage scores, delete track),

Manage_tracks function has considering delete track & scores reduction)

Handle_updated_track function performs track score increase & status.

#Track Initialize :

- Initialize track with new measure data
- Function(manage_tracks) inspect the measure data and create new track with addTrackTolist
- Set Initial state with **'initialized'**. Set score '**1/params.window'**

#Delete Track:

- If track score is too low, or Covariance matrix 'P' is too big then delete it
- Deletion criteria refers 'params.delete_threshold' , 'params.max_P

#Manage Tracks:

- Increase the scores and Update the status as 'tentative' or 'confirmed'

##Loop_over_dataset.py :

- Visualize how tracks and metrics are managed with exec_visualization = ['show_tracks'].
- Use plot_rmse to compute the RMSE for each track and check whether the average RMSE satisfies the target.

Results:

Check points:

- RMSE: The average RMSE should be around 0.8 when tracking a single object.
- Visualization: Check that the tracks are correctly initialized and deleted, and that the console outputs the deleting track no.
- Performance: Evaluate that the tracks are not lost and are properly managed.

But actual result has not plot the RMSE. I will update this page with mentor's support.

(Kor) ##Trackmanagement.py: 주요 로직(트랙 초기화, 스코어 관리, 트랙 삭제),

Manage_tracks 함수는 트랙 삭제 및 스코어 감소를 고려함)

Handle_updated_track 함수는 트랙 스코어 증가 및 상태를 수행합니다.

#Track Initialize:

- 새 측정 데이터로 트랙 초기화

- Function(manage_tracks)는 측정 데이터를 검사하고 addTrackTolist로 새 트랙 생성

- 'initialized'로 초기 상태 설정. 점수 설정 '1/params.window'

#트랙 삭제:

- 트랙 점수가 너무 낮거나 공분산 행렬 'P'가 너무 큰 경우 삭제

- 삭제 기준은 'params.delete_threshold', 'params.max_P'를 참조

#트랙 관리:

- 점수를 높이고 상태를 'tentative' 또는 'confirmed'로 업데이트

##Loop_over_dataset.py:

- exec_visualization = ['show_tracks']로 트랙과 메트릭이 관리되는 방식을 시각화합니다.

- plot_rmse를 사용하여 각 트랙의 RMSE를 계산하고 평균 RMSE가 목표를 충족하는지 확인

결과:

체크 포인트:

- RMSE: 단일 객체를 추적할 때 평균 RMSE는 약 0.8이어야 합니다.

- 시각화: 트랙이 올바르게 초기화되고 삭제되었는지, 콘솔에서 삭제 트랙 번호를 출력하는지 확인합니다.

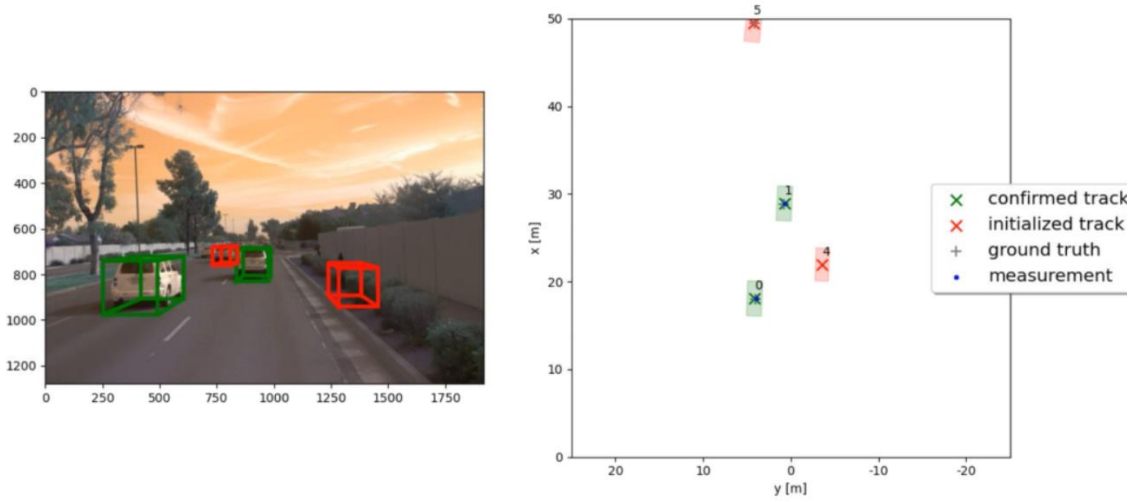
- 성능: 트랙이 손실되지 않고 적절하게 관리되는지 평가합니다.

하지만 실제 결과는 RMSE를 표시하지 않았습니다. 멘토의 지원으로 이 페이지를 업데이트하겠습니다.

Project - Step 3:

Goal :

- **Verifying Data Association :** Ensure that multiple tracks are correctly associated with multiple measurements. Ensure that each measurement is used at most once in the console output, and that each track is updated at most once
- **Verifying Ghost Track :** The visualization shows that there are no "ghost tracks" that do not actually exist. Initialized or provisional ghost tracks may be created, but these tracks should be deleted after several frames.
- **Save RMSE Plot:** If the correlation is working correctly and all tracks and measurements are correctly matched, generate and save an RMSE plot.



(Step 3 Tracking Screen)

- **Results:**

As a result, although it is a tracking using only Lidar, it has shown excellent tracking results. We will analyze these results together in step 4.

(Kor) 결과적으로 아직까지는 Lidar만을 활용한 추적이지만 훌륭한 추적결과를 보여주었습니다. 이 결과에 대한 분석은 step4에서 함께 수행하겠습니다.

Project – Step 4

Goal : The goal of Phase 4 is to implement a nonlinear camera measurement model and complete the sensor fusion module through camera-lidar fusion.

-- Accurate sensor fusion: Track update using lidar and camera measurements.

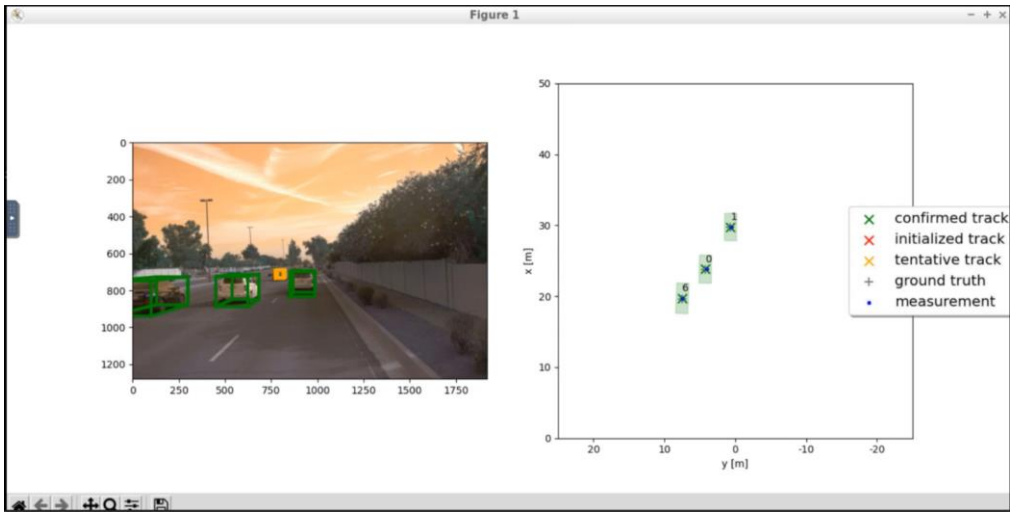
-- Console output: Show camera update after lidar update.

-- Visualization: No confirmed ghost tracks, no track loss.

-- RMSE plot: : 3 confirmed tracks. Average RMSE ≤ 0.25.

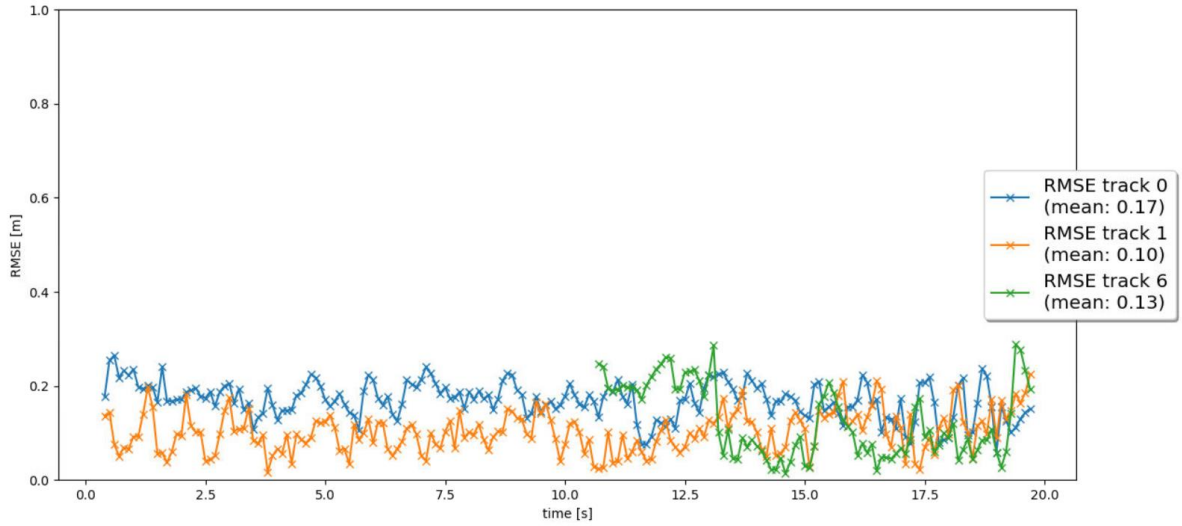
-- Keep tracks without track loss during sequence (0-200s).

Results : Step 3 Vs Step 4

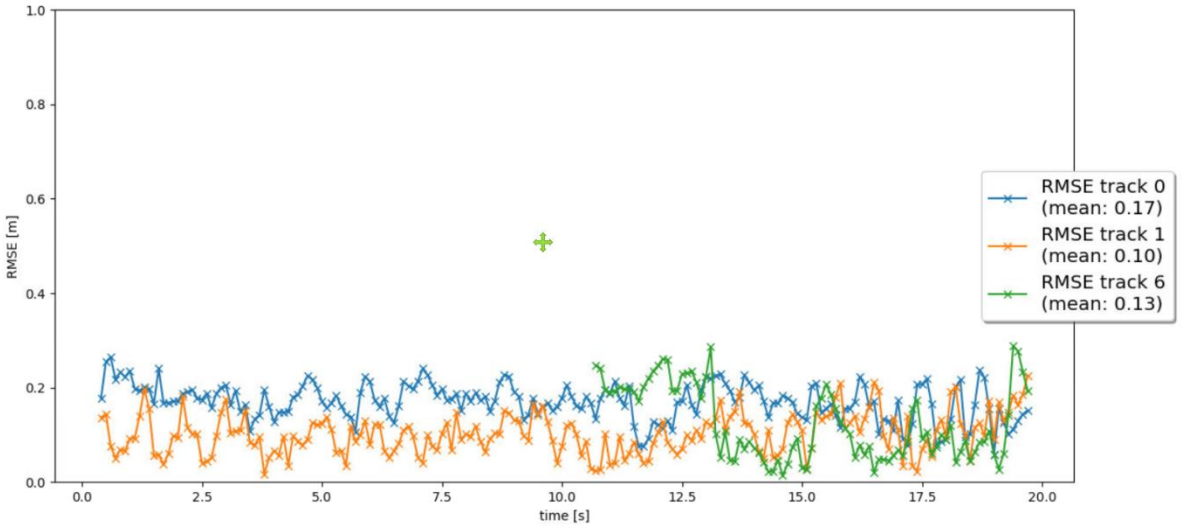


(Step 4 Tracking Screen)

Move Link : https://youtu.be/lf18ogjRAzY



(STEP 4 : RMSE Plot)



(STEP 3 : RMSE Plot)

As a result, the results of Step 3 using only lidar and Step 4 tracking objects with lidar + camera look the same. My guess is that the data provided by waymo is in a daytime environment with daylight, and there are not many vehicles, and there are no pedestrians or bikes, which is good for tracking, and since the lidar data itself is of high quality, I think there was no big difference in fusion with the camera. If the driving environment conditions were different, I think the results of fusion with the camera would definitely be different.

(Kor) 결과적으로 lidar만을 사용하는 Step3와 Lidar + camera로 객체를 추적하는 Step4의 결과가 동일하게 보입니다. 제 추측으로는, waymo에서 제공한 data가 day light가 있는 낮의 환경이고

차량도 많지 않고 보행자나 bike등이 없는 추적에 좋은 조건이며, Lidar의 Data자체가 고품질이기 때문에 camera와의 융합에서 큰 차이를 보이지 않은 것으로 판단합니다.

만약 다른 주행환경 조건이라면 camera와의 융합을 한 결과는 분명히 다를 것이라고 판단합니다.