GEORG-AUGUST-UNIVERSITÄT GÖTTINGEN
INSTITUTE OF COMPUTER SCIENCE

Practical Course: Data Fusion
Report

Bianca Vetter (21681974) & Valerius Mattfeld (11580056)

September 2021

INTRODUCTION

The Practical Course: Data Fusion took place as a Blockkurs from 6th September 2021 to 24th September 2021 followed by an online presentation on the 27th September 2021. The course, led by Prof. Dr. Marcus Baum and M.Sc. Laura Marie Wolf, was attended as an Master module (M.Inf.1822) with a workload of 6 ECTS and organised as a worksheet.

At the introductory meeting, the students were spilt groups consisting of two to three people that worked daily on the given worksheet that included 8 exercises. The exercises were based on the public nuScenes [1] data set that includes a variety of driving situations. The data consists of 20 seconds long video snippets of vehicle in traffic also described in form of the necessary sensor data of a self-driving car. The set also includes annotations and the coordinates of the vehicle and the respective sensors.

The goal was to implement and evaluate nonlinear filtering methods. Online meetings that were held daily served as progress reports and a possibility for the groups to ask questions that would clarify the task at hand or ask for help regarding the occurred problems. It was mandatory to attend those meetings at least two times a week. All the exercises were processed in Python.

EXERCISE 1

The first exercise was about preparation and getting to know the theory about the nuScenes [1] data set. Downloading the data and setting everything up consisted on following the steps described in the worksheet . Explaining the terms used in the data set was also quite uncomplicated to find since everything was listed and summarised in the nuScenes [1] tutorial.

However, some of the terms were relatively abstract since they could mean different things. This problem was solved in the following exercise. We were aware that there were two coordinate systems, but it was still unclear how they are defined. A *"guide"* on how to pick different scenes and samples was also included in the tutorial and helped us build the connection to the data base schema also listed on the website. Rendering a scene was also unproblematic due to good examples.

EXERCISE 2

Although we took a look at the data structure in the exercise 1, it was mostly theoretical. The reason the exercise 2 took quite a bit of time was that it was the first time actually handling and working with the data set in practice. Besides that, the goal of the exercise was to extract and prepare the data that will be needed in the following tasks. Here is also where the two coordinate systems from the first exercise come in play.

After extracting the data and taking the time to inspect what it consists of, it became clear that the two coordinate systems were vehicle-centric (translation and rotation of each of the radar

points to the midpoint of the rear vehicle axle) and map-centric (translation and rotation of the vehicle given with respect to global coordinate system). The task at hand was to transform the radar points into the global coordinate system, see `data_fusion/utils/data_parsing/py` Ln. 27 - 46. [2]

Following that, the radar points, the Doppler velocity, the bounding boxes of the annotated vehicles and the ego vehicle position were to be plotted. In the first phase, it was about figuring out how to plot these for one sample and the second phase consisted of applying the same steps on all samples of a scene, for those to then be plotted as a video.

For this exercise we also asked quite a bit of questions at the online meetings and also kept on coming back and improving on it even after proceeding with the worksheet. We were able to finalize the exercise after we realised that we extracted the wrong data by mistake. A simple mistake as a wrong list index had cost us three days worth of work.

The end result included three plots: A plot of all the radar points during the scene, a plot of the Doppler velocities for every vehicle and a plot of the bounding boxes also of all the vehicles appearing in the scene.

EXERCISE 3

In a way, exercise 3 was similar to the previous way because the goal was also preparing the data and measurements that would be then used in the next exercises.

Firstly, from the scene picked in the previous task, we filtered the vehicles that were not moving by using the `vehicle.moving` attribute. After that, we kept on narrowing the vehicles down, targeting the token of the vehicle that was driving in front of the ego vehicle. We were able to get the token of this vehicle by using the technique of the bounding boxes as we did in exercise 2, which we then proceeded to save in a global variable for further use. The bounding box is described in the data set as a width, height and the lower left corner coordinate.

The next step was to filter all the radar points that lie inside the bounding box of the vehicle. In addition, a small margin was added to make sure that all radar points were included. Those points inside the box were then averaged and plotted in a video that also included the extracted ground truth measurements that serve as a reference point. The added margin was initially set relatively high just so we could have a measurement for every timestamp. The reason was a simpler initial processing of the next few tasks. However, after working out some of the following exercises, the margin was readjusted in a much more reasonable one (about 30 cm) resulting in missing measurements in some of the samples. This further led to readjusting all of the following exercises.

The result consists of a final plot with a ground truth marker, a marker for the averages and lastly, a marker representing the ego vehicle, see `notebooks/prep.ipynb` [2].

EXERCISE 4

The exercise number 4 took us by far the longest to understand. The goal was to implement the Kalman filter algorithm.

It took quite a bit of research to understand what the Kalman filter was and how it works. After the initial research, we could start with the implementation. This took some time as it was unclear how the variables/parameters from the worksheet translated to the given presentation and to the materials we found online. It was also unclear how some of the missing parameters were to be initialised if at all. This issue would be resolved and the parameters were clarified at the daily online meeting.

We proceeded to write the time update formulas and the measurement formulas. Those were broken up in few methods that we then *"puzzled"* together in the final plotting method.

After we finally had a plot so we could visualise what we were doing, it was clear that, although we followed the algorithm, that the plot was faulty. After multiple rechecking of all the formulas and the implemented algorithm, it became clear that the fault was either a wrong initialisation or that we fed the methods the wrong parameters. We were able to resolve this issue again at the daily meeting. The mistake was in the measurement update formulas that were supposed to have the ground truth as a parameter. This led to us finally fully understanding how the algorithm of the Kalman filter works and we were able to successfully implement it and get a correct plot.

This is also an exercise where the missing radar points come into play since we need this data to update the measurements in the Kalman filter algorithm. We just continued to loop the predictions and as soon as we got the necessary data, we proceed with the measurement update as per usual. The second option to solve this problem would be not to have a constant time within the time update formulas, but instead to calculate them depending on the missing data.

The final result is again a plot with two attributes: firstly the predictions from our algorithm and secondly, the ground truth that, once again, served as a reference point, see `notebooks/prep.ipynb` [2].

EXERCISE 5

The exercise 5 on the worksheet was skipped as it was an exercise intended for the bachelor students attending this course.

EXERCISE 6

This exercise was similar to the exercise number 4, since the goal was also to implement a Kalman filter, except this time it was the extended Kalman filter.

Since we knew from the previous task which and how the parameters were to be initialised, this exercise came together much quicker. Almost all of the problems we had could be categorised in the following two categories: How/where to initialise the ground truth data and the Python library Sympy [3]. Neither of us have worked with the Sympy library before and therefore, most of our research was limited to the Sympy documentation.

The most time-consuming problem was including the values of the global variables into formulas. For example, after we calculated the Jacobian matrix of the multivariate time function listed on the worksheet with the variables declared as symbols, there was a problem with inserting the actual value of α, the angle in the trigonometric functions.

Although the syntax should be $cos(\alpha)$, we constantly got the error *Symbol has no attribute cos*. This happens because the command `cos(alpha)`, for some reason, gets interpreted as `alpha.cos()`.

It is still unclear to us why this happens except that it is somehow linked to the `lambdify` function. We found a workaround which was the subs function. The important thing to notice here is that the float variables that hold the to-be-implemented values need to have different names than the Symbol type variables.

Since we lost quite a bit of time trying to figure all of this out, we concluded a bit refactor was necessary. Which is why the vehicle data has a separate class `Vehicle` that includes the annotations and the bounding box information. We have worked completely in a Jupyter Notebook up until this point. Restructuring the project into a separate pythonic module in conjunction with utilising the import syntax helped to reclaim a better overview over the used code, results and scope issues typically encountered while realising a project using notebooks. The final structure and intentions behind every submodule and file can be found in the `README.md` [2].

The issue with no data related to some samples we solved the same way we did in exercise 4. The final plot also looks pretty similar, as we have also the predictions the ground truth plotted. Although this filter may have a bigger difference than the basic Kalman filter in the beginning it seems to be more precise later on.

EXERCISE 7

The goal of exercise 7 was to implement the unscented Kalman filter. We assumed it would be closely related to the previous two exercises. However, we found this one to be really hard to understand. We were to use Wikipedia as our reference point. The different variables and parameters of different functions were very unclear. The notation was also ambiguous and it took us quite some time to figure it out. In retrospective, one could say that it was bold for us to assume that we would be able to finish this exercise within one day, but we still think that the time we sacrificed for the big refactor was absolutely necessary and worth it in the long run.

The unscented Kalman filter algorithm uses the, so called, sigma points that make it a better approach when it comes to non linear transformations. As we learned, the unscented filter has the upper hand when compared to the extended filter. This becomes apparent when dealing with highly nonlinear functions. This due to the more appropriate sampling technique - the unscented transformation. [4]

Unfortunately, this exercise was not completed due to the fact that we get a negative definite matrix that, for that reason, cannot be decomposed using the Cholesky decomposition. The problem seems to be already present when calculating the prediction, but we are still unsure as to why.

Even though we did not complete this exercise, we could see one important benefit of this approach in contrast to the extended Kalman filter, which is the (non)-calculation of the Jacobian matrix. Although the given functions did not pose a problem for us in this particular situation, more complex functions could be very heavy on the computation.

Since we did not fix the algorithm, we were not able to finish the exercise 7 since this required testing the combination of the extended Kalman filter and the unscented Kalman filter. Furthermore, this also meant that we could not fully process the exercise 8.

EXERCISE 8

The purpose of the last task was to evaluate the different filters we implemented by computing the root mean squared error of the ground truth and the predictions from all the filters. This is also what we represented visually by plotting those measurements.

We unfortunately did not have enough time to try the algorithms on different vehicle instances, but that is definitely something our team is keen to keep on experimenting with even after the official course has ended.

CONCLUSION

Although we did not fully complete all of the tasks, we would still call this course successfully completed. We learned quite an amount of new concepts, we could see the improvement of our approaches in every new exercise we would start and the visions of the end results were more clear with every newly finished task which would make the planing more efficient.

Extending the Kalman Filter into multiple variations and reflecting on their results after was a satisfying experience. We were seeing results of different concepts and had a hands-on approach on an otherwise theoretical topic.

In our opinion, this was also the first time we were actually able to fully utilize our linear algebra knowledge.

Although we both had a fair amount of Python experience in the past, this was a new way to take advantage of this knowledge.

REFERENCES

[1] Holger Caesar et al. “nuScenes: A multimodal dataset for autonomous driving”. In: *arXiv preprint arXiv:1903.11027* (2019).

[2] Valerius Mattfeld and Bianca Vetter. *Practical Course on Data Fusion, 2021*. Sept. 2021. URL: https://github.com/valerius21/x_mean.

[3] Aaron Meurer et al. “SymPy: symbolic computing in Python”. In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[4] Wikipedia contributors. *Unscented Kalman filter*. [Online; accessed 20-September-2021]. 2021. URL: https://en.wikipedia.org/wiki/Kalman_filter#Unscented_Kalman_filter.

WORKSHEET

The worksheet can be found on the next page.

Practical Course: Data Fusion
Summer Semester 2021
Project Description

University of Göttingen
Institute for Computer Science
Data Fusion Group
Prof. Dr.-Ing. Marcus Baum
http://www.uni-goettingen.de/fusion

The following tasks should act as a guidance on your way to the project goal. You are welcome to experiment with the data. Tasks marked with (Bachelor) are mandatory for bachelor students and and optional for master students. Vice versa tasks marked with (Master) are mandatory for master students and optional for Bachelor students. However, you are free to try all of them.

You can solve the tasks either in regular python scripts or in jupyter notebooks. You should prepare a small presentation or jupyter notebook for the last day of the practical course (September 24) to present your results. You should write a report of 6 pages summarizing your main results and insights until the end of the semester (September 30).

1) NuScenes dataset

a) You require Python 3 on your device. The nuScenes devkit is tested for Python 3.6 and 3.7. Then install the nuScenes devkit via

```
pip install nuscenes-devkit
```

For a more detailed guide using a conda environment, visit
`https://github.com/nutonomy/nuscenes-devkit/blob/master/docs/installation.md`

b) Visit `https://www.nuscenes.org/download`, create an account, and download the *Mini*-subset of the *Full dataset (v1.0)*.

c) Visit `https://www.nuscenes.org/nuscenes#tutorials` for a basic introduction to the functionalities of the nuScenes devkit (select the nuScenes tutorial) and get to know the data format. Explain the terms scene, instance, sample, and sample_annotation in your own words. What is a keyframe? Which coordinate systems are used and which data are given in which coordinate system?

d) Look into the different scenes, pick out a sample and render it. What challenges can you identify for detecting the vehicles using only the radar point cloud?

2) Prepare radar data

We will be using the radar data for tracking. A radar data point consists of Cartesian coordinates and a Cartesian vector specifying the component of the object's velocity in direction of the sensor. In order to get a surrounding view we have to combine the data from all radar sensors.

a) Write a function that extracts the radar points for a given sample from the binary files (check the function `from_file()` of the class `RadarPointCloud` in nuscenes.utils.data_classes). Extract the Cartesian coordinates and Doppler velocity (`v_comp`) and transform them into the global coordinate system[1]. Combine the data from all radar sensors.

b) Plot the radar points including Doppler velocity for a sample of your choice.

c) Plot the bounding boxes of the annotated vehicles and the position of the ego vehicle.

d) Repeat this process for all samples of a scene and combine the plots to a video.

3) Prepare measurements

We will select a vehicle to track and prepare the measurements.

a) Select a vehicle from your sample which is annotated in most of the frames.

b) Filter the radar points that lie inside the vehicle's bounding box. It may be necessary to add a small margin around the bounding box to capture all points belonging to the vehicle.

c) Average all points inside the bounding box to receive a single measurement per timestep.

d) Generate a plot with the vehicle's measurements for all keyframes of the scene. Add the annotated true position (ground truth) of the vehicle.

[1] Hint: check `transform_matrix` in nuscenes.utils.geometry_utils.

4) Kalman Filter

We are going to track a single vehicle using a Kalman filter[2]. To track the vehicle, we define its state as a $2D$ Cartesian position and velocity $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & v_1 & v_2 \end{bmatrix}^{\mathrm{T}}$. As we do not know the exact state, we model it as a Gaussian random variable $\mathbf{x} \sim \mathcal{N}(\hat{\mathbf{x}}, \mathbf{C})$ and are going to estimate its mean $\hat{\mathbf{x}}$ and covariance $\mathbf{C}$.

a) For the time update, we assume a constant velocity model, with transition matrix

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & t & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

which moves the state t seconds into the future, no control input and process noise covariance matrix

$$\mathbf{Q} = \begin{bmatrix} 0.5t^2 \\ 0.5t^2 \\ t \\ t \end{bmatrix} \sigma_v^2 \begin{bmatrix} 0.5t^2 & 0.5t^2 & t & t \end{bmatrix}.$$

Implement the Kalman filter time update formulas.

b) For the measurements, we assume one $2D$ measurement $\mathbf{y}_k$ of the state's position per time step. We do not incorporate the Doppler velocity here. Implement the Kalman filter measurement update with measurement matrix

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix},$$

and measurement noise covariance matrix $\mathbf{R} \in \mathbb{R}^{2\times 2}$.

c) Use the first two appearances of the vehicle to initialize its state. Apply the Kalman filter to estimate a track of the vehicle based on the measurements. Experiment with values for the initial covariance $\mathbf{C}_{0,0}$, process noise variance σ_v^2, and measurement noise covariance $\mathbf{R}$ and observe for which settings your estimate follow the true target best. Add the track to the plot with the ground truth.

5) Extended Kalman filter (Bachelor)

We will incorporate the Doppler velocities and implement the Extended Kalman Filter (EKF). We use the same state as before, but the measurements are now given as $\mathbf{y} = \begin{bmatrix} y_1 & y_2 & d_1 & d_2 \end{bmatrix}^{\mathrm{T}}$, with Doppler velocity $d = \begin{bmatrix} d_1 & d_2 \end{bmatrix}$ in Cartesian coordinates.

a) In order to compute the Doppler velocity of the predicted measurement, the object's velocity $\mathbf{v}$ is projected onto the vector from the sensor's position to the object's position, $\mathbf{r} = \mathbf{x} - \mathbf{s}$, where $\mathbf{s}$ is the position of the sensor. Implement the EKF measurement update using the measurement function

$$h(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & \frac{v_1 r_1 + v_2 r_2}{r_1^2 + r_2^2} r_1 & \frac{v_1 r_1 + v_2 r_2}{r_1^2 + r_2^2} r_2 \end{bmatrix}.$$

You can either compute the derivatives by hand or use, e.g., the `sympy` package.

b) Combine the EKF measurement update with the Kalman filter time update from the last exercise and try to find suitable values for $\mathbf{R}$. Add the resulting track to the plot and compare.

6) Extended Kalman filter (Master)

We will incorporate the Doppler velocities and implement the Extended Kalman Filter (EKF). We use the state $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & v & \alpha \end{bmatrix}^{\mathrm{T}}$ with Cartesian position, speed v and orientation α (counter clockwise angle between the vehicle's heading and the positive global x-axis) and 4-dimensional measurements $\mathbf{y} = \begin{bmatrix} y_1 & y_2 & d_1 & d_2 \end{bmatrix}$, with Doppler velocity $d = \begin{bmatrix} d_1 & d_2 \end{bmatrix}$ in Cartesian coordinates.

[2] Please refer to the uploaded slides.

a) Implement the EKF time update using the transition function, which propagates the state t seconds into the future.

$$a(\mathbf{x}) = \begin{bmatrix} x_1 + tv\cos(\alpha) & x_2 + tv\sin(\alpha) & v & \alpha \end{bmatrix}.$$

You can either compute the derivatives by hand or use, e.g., the `sympy` package.

b) In order to compute the Doppler velocity of the predicted measurement, the object's velocity (v, α) is projected onto the vector from the sensor's position to the object's position, $\mathbf{r} = \mathbf{x} - \mathbf{s}$, where $\mathbf{s}$ is the position of the sensor. Implement the EKF measurement update using the measurement function

$$h(\mathbf{x}) = \begin{bmatrix} x_1 & x_2 & \frac{v\cos(\alpha)r_1 + v\sin(\alpha)r_2}{r_1^2 + r_2^2} r_1 & \frac{v\cos(\alpha)r_1 + v\sin(\alpha)r_2}{r_1^2 + r_2^2} r_2 \end{bmatrix}.$$

c) Find suitable initializations for the state and the state covariance and experiment with different values for $\mathbf{Q}$ and $\mathbf{R}$. Apply the EKF to your vehicle. Add the track to the plot and compare.

7) Unscented Kalman filter (Master)

We will try another approach to deal with nonlinearities in filtering, the Unscented Kalman Filter (UKF), which uses fixed deterministic samples and does not require derivatives.

a) Check out the UKF at Wikipedia. Implement the UKF time update and measurement update using the same transition and measurement functions as with the EKF.

b) Apply the UKF to your vehicle. Add the track to the plot and compare.

c) You can also mix the EKF and UKF time and measurement update or try the UKF instead of the EKF in task 5. Do you see differences?

8) Evaluation

a) Compute the root mean squared error (RMSE) between the ground truth track and each track that you estimated. Which filter worked best?

b) Try your algorithms on different vehicle instances and try to find a parameterization that works on all of them.

c) You already have a plot with the filtered measurements and all tracks. Now add the tracks to the video from the beginning!

d) Which problems did you encounter? What shortcomings did your filters have? What could be improved?