Unscented Kalman Filter Project

Jean-François Vanreusel

Overview

In this project, we use **Unscented Kalman Filter** approach to track a cyclist that travels around our car.

Data is provided through 2 types of sensors: **Lidar** and **Radar**. Lidars only provide information on the position. Radars provide information on both position and velocity.

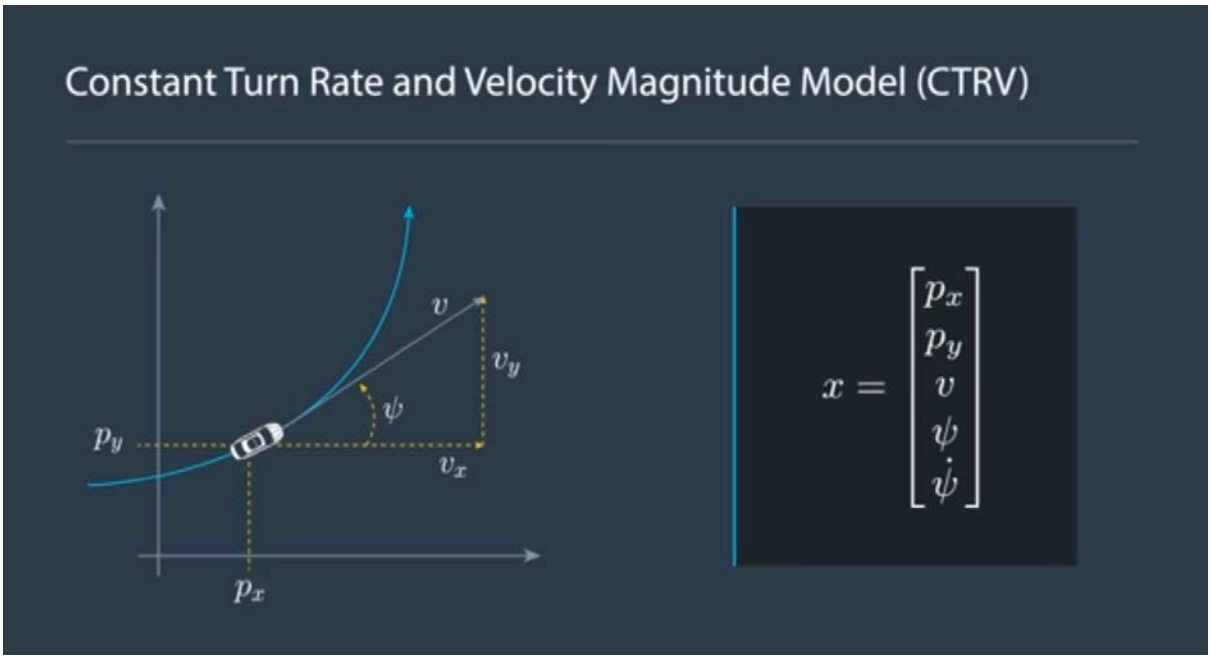
Our software returns predicted positions of the cyclist from multiple data sets.

It also returns the accuracy of our software by comparing our predictions with provided ground truth data. Accuracy is measured through **Root-Mean-Square-Estimates** and **NIS** methods.

Deliverables

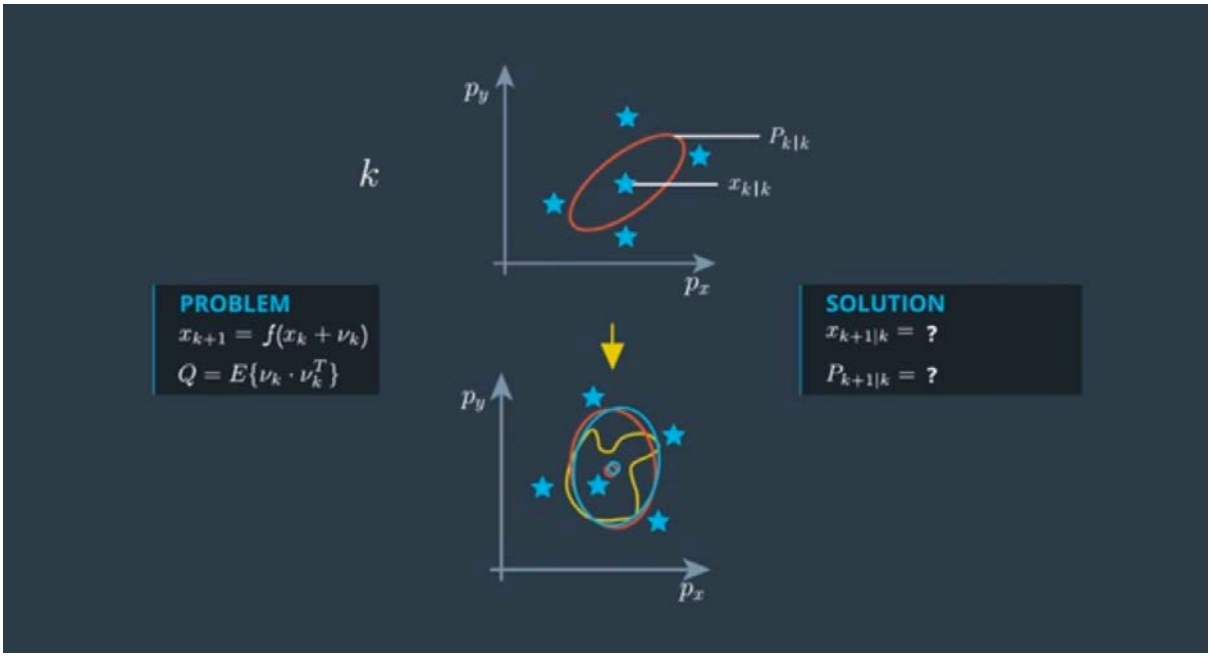
- **Unscented Kalman Filter Project.pdf** – You're reading it ☺.
- **Main.cpp** – Main file for the project. It reads in the data, creates an instance of UKF and provides the accuracy of the results
- **UKF.*** - UKF algorithm.
- **Tools.*** - file that includes the code for the RMSE assessment.
- **Measurement_package.*** - defines the measurement class to read in data from Radar or Lidar.
- **Ground_truth_package.h** – defines the ground_truth class
- **Eigen** – Folder and files that provide Eigen support for matrices operations
- **Data** – Folder with our input and output files for 3 data sets.

Overall UKF Algorithm



In this project, the state of the cyclist (not the car as shown above) is defined by 5 variables: Px (position on X axis), Py (position on Y axis), v (velocity), yaw (angle of the turn) and Yaw rate (change rate in angle).

With Extended Kalman filters, we used the Jacobian matrix to approximate the non-linear update function. With Unscented Kalman Filters, we have non-linear prediction and update functions, so we apply a different technique by we generating sigma points that approximate our functions.



As part of UKF algorithm, we perform 3 key steps:

1. **Initialization step** - where we set our State variables, Covariance matrix.
2. **Prediction step** - where we predict the position of the cyclist at time k+1. To do so, we generate sigma points (representative of our function) and pass them through our process function to create the predicted sigma points. We also compute the Mean and Covariance for the predicted state.
3. **Update step** - where we combine the recent measurement with our prediction. We first need to update our predicted state into our measurement state so we can fine tune it based on our recent measurement. The Update step needs to be performed for both LIDAR and RADAR data.

Build instructions

To compile and build the code, simply type in the following commands:

1. Clone this repo.
2. Make a build directory: `mkdir build && cd build`
3. Compile: `cmake .. && make`
4. Run it: `./UnscentedKF ../data/obj_pose-laser-radar-synthetic-input.txt output.txt`. You can find some sample inputs in 'data/'.

Code Overview

In this project, the key algorithms are in the ukf.cpp and tool.cpp files.

Ukf.cpp

This file includes the UKF algorithm. The UKF class includes the variables and methods necessary to perform the Kalman filter.

UKF Methods:

- **UKF** – initialize all the instance variables. In particular, this is where we initialize our weights vector.
- **ProcessMeasurement** – use the first measurement to initialize the state. For subsequent measurements, it converts the time into seconds. It also checks that the time between the last two measurement is not too big. If it's too big we split it into smaller chunks and perform the **Prediction** function multiple times. We then run the **Update** function based on the nature of the signal (LIDAR or RADAR).
- **NormalizeAngle** – This function normalizes the angle to the [-PI, PI] range. Normalizing the angles help us keep the covariance matrix in the right range.
- **Prediction** – the prediction function predicts the state of the cyclist at time k+1. It first create sigma points (step 1), it then passes those sigma points to the Prediction function (step 2) and then compute the Mean and Covariance for the predicted state (step 3).

- **PreditionRadarSpace** – The Prediction is still in the "state" space. We need to convert it to the Radar space so we can combine it with RADAR data in the UpdateRadar function. We don't need such conversion for the LIDAR space since Px, Py are part of the Lidar space.
- **UpdateLidar –** we use the Extended Kalman Filter function here. We can do this because we deal with a linear function.
- **UpdateRadar –** uses the Unscented Kalman Filter functions to update the state based on Radar data. The prediction has already been changed to the Radar space through the PredictionRadarSpace function.

Tools Methods:

- **CalculateRMSE** – this function computes the Root-Mean-Squared Error that we use to validate our results.

Results

Test File 1: obj_pose-laser-radar-synthetic-input.txt
We ran the UKF software against the data file 1 and obtained the following results:

Final estimated space:

```
  -7.02388
   10.8853
   4.98148
 -0.021433
-0.0519749

RMSE
0.0694128
0.0809105
 0.294646
 0.205604
```

The results are within the expected range ([.09, .10, .40, .30]).

Test File 2: sample-laser-radar-measurement-data-1.txt

We ran the UKF software against the data file 2 and obtained the following results:

Final estimated state:

```
 11.3791
-1.91173
-2.66904
-1.74594
0.514875

RMSE
0.0839282
0.0937808
 0.607109
  0.60134
```

Those results are mostly within the expected range for this data set. RMSE <=[0.09, 0.09, 0.65, 0.65].

Test File 3: sample-laser-radar-measurement-data-2.txt

We ran the UKF software against the data file 3 and obtained the following results:

Final estimated state:

```
   204.063
   36.1251
   1.37179
 -0.382745
-0.0508767
```

```
RMSE
0.170268
0.177715
0.202657
0.267382
```

The RMSE are within the expected range: RMSE <= [0.20, 0.20, 0.55, 0.55].

Parameters fine-tuning

Process Noise Covariance

Some parameters were given to define the Process Noise Covariance (Q). For the acceleration noise, we usually pick ½ of the maximum acceleration for the item that we track. For a bicycle in an urban environment, we believe that it would be about 3 m/s^2. So, if we divide this by 2, we use 1.5 as the standard deviation longitudinal acceleration.

For the standard deviation associated to the yaw acceleration, we looked at the data and observed small changes. By fine tuning this parameter we ended up with 0.5 standard deviation.

Using NIS, we checked if we are using parameters that are too conservative for data file 1 (obj_pose-laser-radar-synthetic-input.txt).

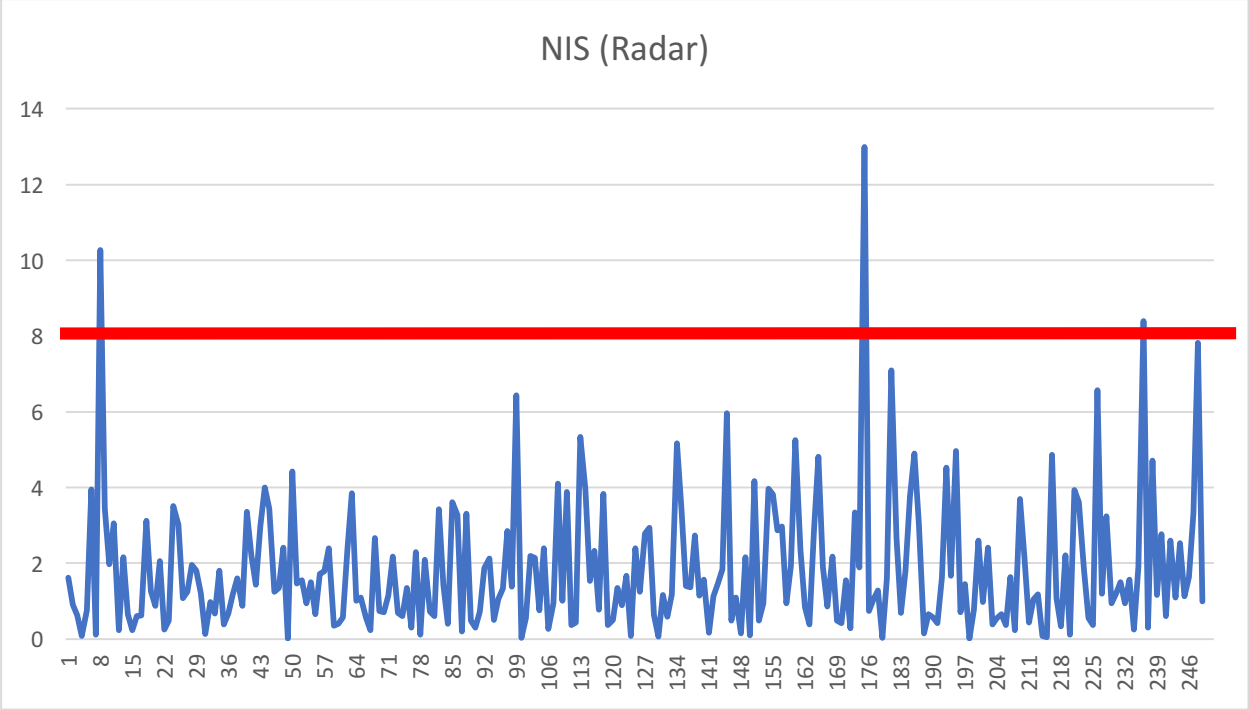
NIS follows a Khi-square distribution.

$\varepsilon \sim \chi^2$

df	χ^2.950	χ^2.900	χ^2.100	χ^2.050
1	0.004	0.016	2.706	3.841
2	0.103	0.211	4.605	5.991
3	0.352	0.584	6.251	7.815
4	0.711	1.064	7.779	9.488
5	1.145	1.610	9.236	11.070

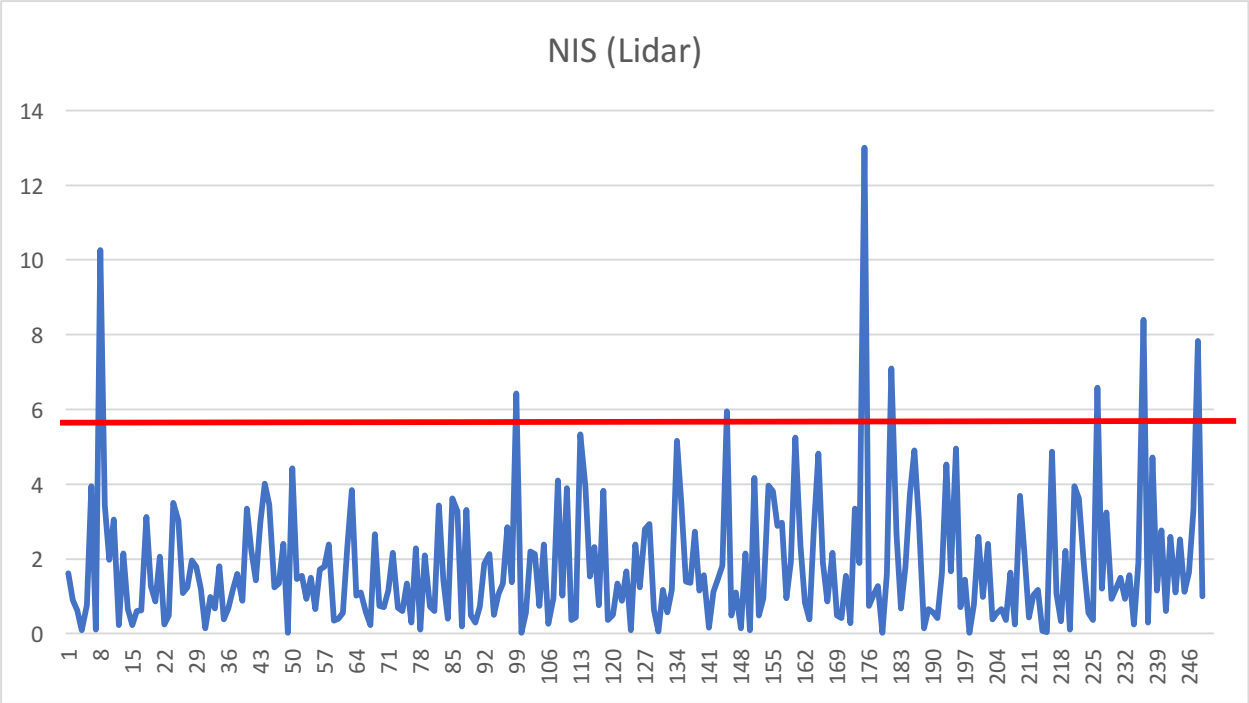
for Radar, we have three coordinates (or degrees of freedom (df)). Epsilon at 7.8 for DF=3 means that only 5% of the distribution has a higher NIS.

For Lidar, we only have 2 coordinates (i.e. DF=2), so we want to see most of our NIS measures below 5.991.



NIS measures roughly come below the 7.8 bar (in red) which indicates when our predictions and estimates are accurate in about 95%. This shows that the Radar parameters are about right.

For LIDAR data, we have the following graphic



It also looks that most of our data is below the red line defined by a value of NIS @ 5.91.

UKF versus EKF results comparison

File	EKF	UKF
sample-laser-radar-measurement-data-1.txt	Accuracy - RMSE: 0.0651649 0.0605378 0.54319 0.544191	RMSE 0.0839282 0.0937808 0.607109 0.60134
sample-laser-radar-measurement-data-2.txt	Accuracy - RMSE: 0.185496 0.190302 0.476754 0.804469	RMSE 0.170268 0.177715 0.202657 0.267382

We compared our results for Test File 2 and Test File 3 with the RMSE that we had achieved with Extended Kalman Filter. For Test File 2, we observed slightly worst results than with EKF. Maybe some parameters can be better tuned up for this dataset. For Test File 3, our results are improved – especially when it relates to velocity.

Sensor Fusion versus Radar or Lidar only

We used our Test File 1 to compare the results when we were using just Radar or Lidar data.

	Radar Only	Laser (Lidar) Only
Predicted state	-7.15165	4.21014

	10.7197 -4.7452 9.39279 -0.0287103	5.75089 -4.34197 -35.5266 -4.37962
RMSE	0.627552 1.27323 1.60289 2.02533	14.2385 5.18555 4.17708 3.99674

By using sensor fusion, we obtain the best results. Then, Radar on its own shows good results for the UKF algorithm. Laser data on its own provided much higher RMSE results.