Sensor fusion via UKF to track obstacles

1. **Introduction**

In the last project, I have completed the sensors fusion by the extended Kalman filter (EKF) to track object, and obtained satisfying results. Nevertheless, the applied kinetic model was too simple to represent the actual scenario. In addition, the EKF handles the nonlinear system through introducing the Jacobian matrix, which approximate the state transform function with first-order Taylor expansion and a large error maybe exists.

In this project, I am trying to work on the same topic, and there are two major improvements:

The advanced Constant Turn Rate and Velocity (CTRV) kinetic model will be applied. The previous Constant Velocity (CV) model assumes that the obstacles move along the straight line, which is extremely difficult to meet in practice, while the CTRV model can simulate the curved path of the object.

The process of state update and prediction is established via unscented Kalman filter (UKF). The EKF and UKF are always discussed together. On the one hand, they are designed to solve the same problem – the traditional Kalman filter cannot be used to estimate the nonlinear system. On the other hand, the UKF and EKF have different principle. The EKF handles the problem by approximating the nonlinear system with first-order Taylor expansion, while UKF describes the characteristic of the system through a set of sigma points to avoid the possible error of EKF. There are lots of preferences proving that the UKF is better than EKF. Herein, I am going to seek the confirmation.

1. **CTRV kinetic model**

Different with CV model, the CTRV model can represent the curved path of obstacle, ss shown in Figure 1. The state vector of CTRV model can be described as:

(1)

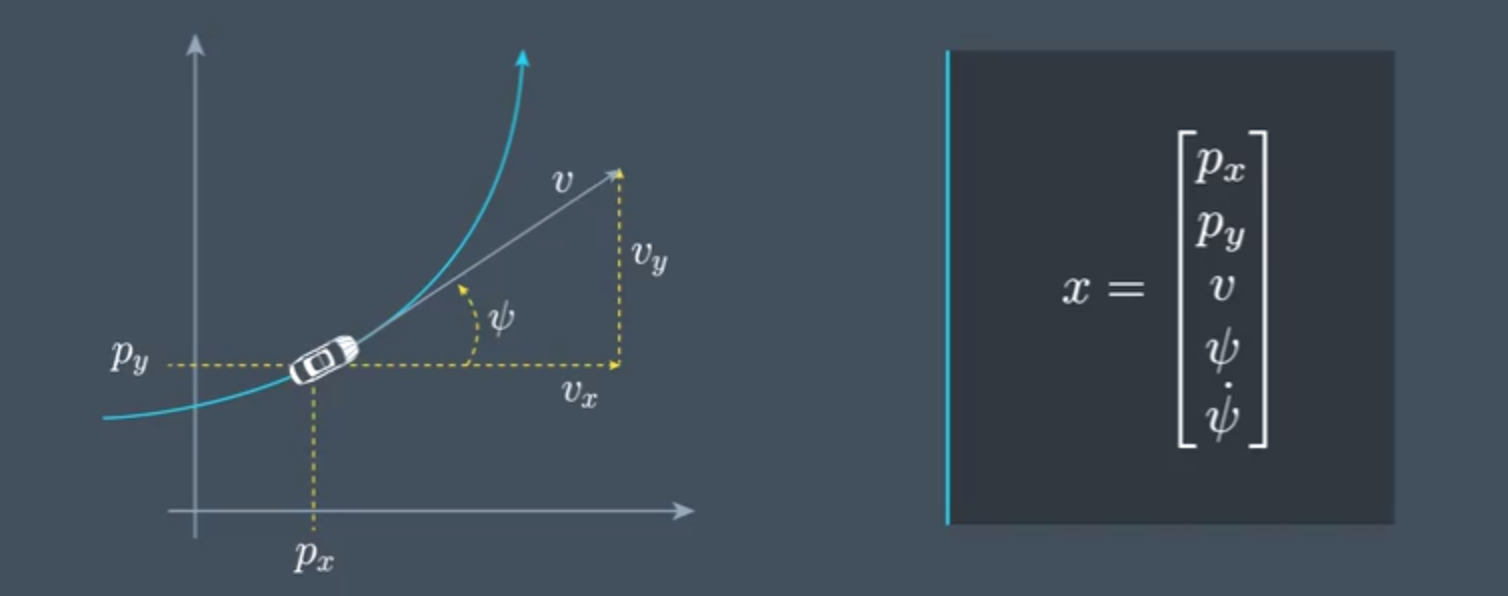


Figure 1 The diagram of CTRV model

Here, *ψ* denotes the yaw angle and its derivatives is the turn rate.

Usually, we use the following equation to describe the relationship between two adjacent sampling time:

(2)

The variable μ can be seen as the process noise.

Now, we need to determine the expression of function *f*.

The CTRV model assumes that the velocity and turn rate of obstacle are constant, so

(3)

(4)

(5)

Δt is the sampling time. μa and μψ denote the process noise of acceleration and yaw angle, and they are assumed to have Gaussian distribution as: μa ~ N(0, σa2), μψ ~ N(0, σψ2).

The prediction function of Px and Py are complicated, but doable.

(6)

(7)

= (8)

(9)

The process noise of Px is also difficult to calculate. However, we can approximate the result by Equation (6). Although errors may be introduced, it is still feasible, as long as the yaw rate is not too high.

According to Equations (6)-(9), the nonlinear prediction function of Px can be written as:

(10)

In the same way, the prediction function of Py is:

(11)

In conclusion, the nonlinear function *f* can be described by Equation (3), (4), (5), (10) and (11).

1. **Sensor fusion by UKF**

**3.1 Generate sigma points**

The procedure of UKF is similar with EKF. The state vector is predicted via the kinetic model and then updated according to the measurement of Lidar or Radar. However, the UKF calculates the mean and covariance by a set of sigma points generated in advance. The UKF algorithm augments the sate vector to consider the influence of process noise. The augmentation vector is:

(12)

The sigma points are generated according to the augmentation vector and the number of sigma points is usually to be (2n+1). Here, n is 7, which is the dimension of augmentation vector. Why (2n+1)? Articles usually utilize this parameter but there is no specific proof to explain this question. I guess that this is just an empirical value, and I think the performance could be verified when we use other values, like (4n+1). But this is not the topic of this article and I will not spend time to expand it. Anyway, 15 sigma points are generated in this project according to the following equation.

(13)

(14)

(15)

Here, λ is a scaling parameter to determine the spread of sigma points around the Xaug. The value of λ is usually set to a small value, such as 3-n. *Paug* is the covariance of the augmented state vector, and it is composed by the initial covariance matrix P and covariance of process noise Q.

(16)

is the i-th row or column of the matrix square root of (n+λ)Paug. The square root of the matrix is computed via a Cholesky factorization.

**3.2 Prediction of sigma points**

Now, we have generated 15 sigma points to represent the system state at time k. The points set at time k+1 can be predicted through the nonlinear function *f*. Something need to be noted, the sigma point input to the nonlinear function is 7-dimension, while output with 5-dimension. All of the 15 sigma points are predicted and then their mean and covariance are calculated to describe the priori at time k+1. The calculation of mean and covariance are as follows:

(17)

(18)

In Equation (17) and (18), the item w(i) denotes the weight of *i*-th sigma point.

(19)

**3.3 Prediction of measurement**

The measurement of Lidar and Radar are different. Just like what we have done in EKF, the state vector should be converted to the measurement space before updating the state via sensors data. Here is the detail of conversion function.

For Lidar measurement, the conversion is pretty simple since the Lidar can only provide the information of Px and Py.

(20)

Based on Equation (20), all of the sigma points can be converted into the measurement space of Lidar.

For Radar measurement, the conversion is much more complicated. The radar doesn’t provide the information of position or velocity, instead it provides information under polar coordinate. The conversion of is shown in Equation (21).

(21)

Then, the mean and covariance of the sigma points in the measurement space can be computed:

(22)

(23)

Here, R is the covariance of the measurement noise. The question is: why the measurement noise is not considered via the augmented vector as Equation (12). The reason is: the process noise in Equation (12) is applied inside the prediction function, while the measurement noise applies outside the conversion function.

**3.4 Update the state at time k+1**

This is the final step. In this step, the Kalman gain matrix will be calculated to update the posterior estimation at time k+1. The Kalman gain matrix is:

(24)

Here, Tk+1|k is the cross covariance of X and Z. The calculation of Tk+1|k is shown as Equation (25).

(25)

Then, the update of state vector and covariance matrix P is:

(26)

(27)

1. **The terrible story**

Now, I have grasped the principle of UKF and the procedure of sensor fusion. The measurement data of Lidar and Radar is ready. I made a cup of coffee and sat in the sunlight and wrote the code on MATLAB. I was excited to run the code and waited for the display of the results.

Oops! Something went wrong. But I was not worried because the bugs were always inevitable. I carefully check all of the code but the problems were still there. Then, I spent a whole week solving the problem without any improvement, the results were divergent. I summarized the reasons as follows:

1. The covariance matrix of process noise and measurement noise are not appropriate because I have no idea about the information the noise.
2. The measurement data of Lidar and radar are not appropriate since the velocity of obstacle varies constantly, while the CTRV model assumes that the velocity is constant.
3. The sigma points spread too dispersive, which results in the divergence of the final results after several sampling period. I have tried to solve this problem but didn’t success.
4. There is another possible explanation: the UKF relies greatly on the initial value and the accuracy of the model. Things maybe go wrong from this point.

All in all, I still have not solve this problem and I feel a bit depressed. I will upload all of the materials to my Github, including the MATLAB codes. If you have any suggestion for me, please don’t hesitate to contact me: [zhengzha16@163.com](mailto:zhengzha16@163.com). I need your help!