Unscented Kalman Filter Project

Objective: to implement Unscented Kalman Filter (UKF) with the CTRV model

Implementation:

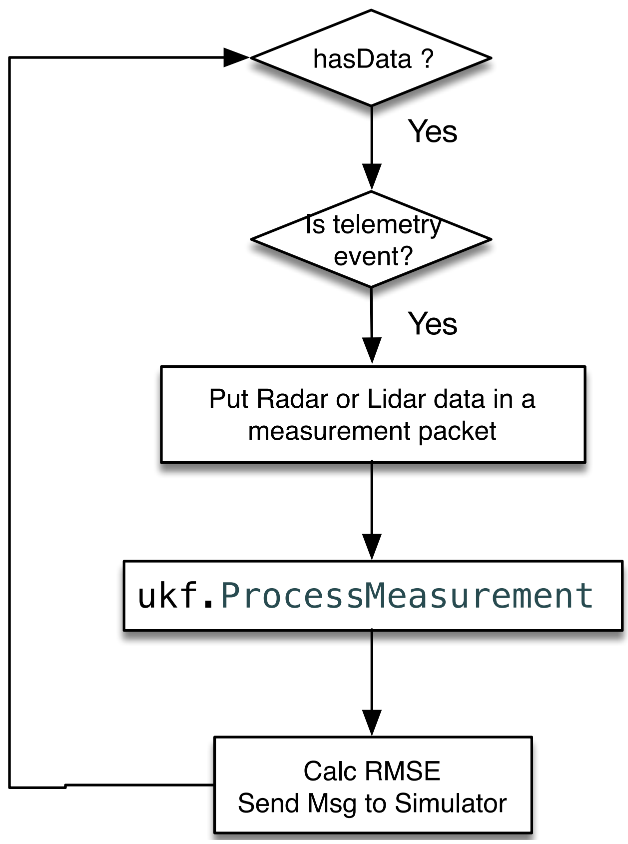

Fig. 1 Flow Chart for the Main Routine

Fig. 1 shows the flow chart of main.cpp. It listens on the socket to get data from simulator. When data arrive, it checks event type telemetry and put either Radar or Lidar data to a measurement packet for UKFto process measurement.

I made two changes in the main.cpp. First, I added three optional arguments in the command line.
std_a - float value of process noise standard deviation longitudinal acceleration
std_yawdd - float value of process noise standard deviation yaw acceleration
l or r - to disable Lidar or Radar

For example, the following command disables Radar and uses 1.0 for std_a and std_yawdd.
>./UnscentedKF 1 1 r

Command >./UnscentedKF works as the same with both Radar and Lidar measurement and default standard derivations.

Secondly, I only push ground truth and estimation into vectors for RMSE calculation when ProcessMeasurement returns true to prevent from getting incorrect RMSE when no estimation is available from measurement.

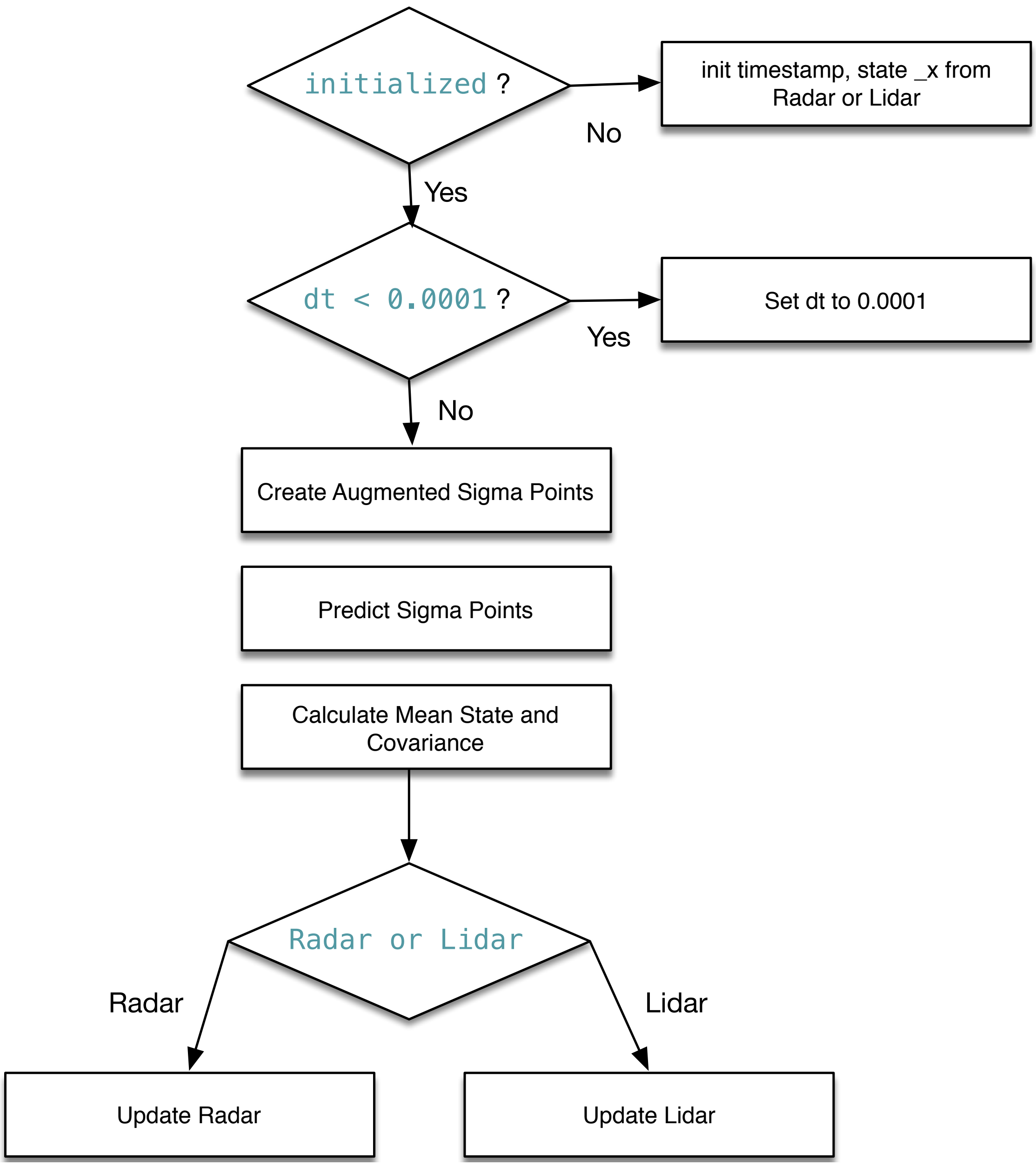

Fig. 2 UKF Process Measurement Flow Chart

Fig. 2 shows the flow chart of Process Measurement of UKF. Object UKF is created as a local variable in main.cpp. Its constructor is invoked to initialize its internal variables.

The following variables are initialized in the contractor gf UKF.
x_: state vector size of five for position (x,y), velocity (v), yaw angle, and yaw rate
P_: covariance matrix
std_a_: process noise standard deviation longitudinal acceleration, init to 0.5
std_yawdd_: Process noise standard deviation yaw acceleration, init to 0.6
There are five measurement noise standard deviation for Radar and Lidar from manufactures.
n_x_: state dimension set to 5
n_aug_: augmented dimension set to 7 (n_x + noises)
n_z_: Radar measurement dimension set to 3 for ro, phi, and r_dot
n_l_: Rider measurement dimension set to 2 for x and y
lambda_: spreading parameter, set 3 - n_aug_
n_sigma: number of sigma points, set 2 * n_aug_+1
weights_: weight vector for sigma points
Xsig_pred_: matrix for predicted values of sigma points dimension n_aug_ x n_sigma_
time_us_: last seen timestamp in micro-second
is_initialized_: flag initialized to false

When ProcessMeasurement is called the very first time, timestamp is saved. If it's from Lidar, I only save position (x, y) into the state x_. If it's from Radar, I convert polar vector to cartesian coordinate. Phi is the angle of position (x,y) not velocity yaw angle. However, I use it to estimate the initial value for yaw and v. Rho_dot can tell us whether the object is moving closer or away. That's a better estimation to start than zeros.

```
// Convert radar from polar to cartesian coordinates
double rho = measurement_pack.raw_measurements_[0];
double phi = measurement_pack.raw_measurements_[1];
double rho_dot = measurement_pack.raw_measurements_[2];
x = rho * cos(phi);
y = rho * sin(phi);
yaw = phi;
double vx = rho_dot * cos(phi);
double vy = rho_dot * sin(phi);
v = sqrt(vx*vx + vy*vy);
if (rho_dot < 0)
   v = -v;
```

Generating Sigma Points
UKF::AugmentedSigmaPoints() creates sigma points in a matrix by formula 1.

$$X_{k|k} = [x_{k|k} \qquad x_{k|k} + \sqrt{(\lambda + n_x)P_{k|k}} \qquad x_{k|k} - \sqrt{(\lambda + n_x)P_{k|k}}]$$

remember that $x_{k|k}$ is the first column of the Sigma matrix.

$x_{k|k} + \sqrt{(\lambda + n_x)P_{k|k}}$ is the second through $n_x + 1$ column.

$x_{k|k} - \sqrt{(\lambda + n_x)P_{k|k}}$ is the $n_x + 2$ column through $2n_x + 1$ column.

Formula 1 Generating Sigma Points

Sigma Point Prediction

UKF::SigmaPointPrediction() implements Sigma Point prediction by using Formula 3. Thereafter, Formula 4 is used in UKF::PredictMeanAndCovariance() to calculate mean state and covariance matrix.

$$\text{Augmented State} = x_{a,k} = \begin{bmatrix} p_x \\ p_y \\ v \\ \psi \\ \dot{\psi} \\ \nu_a \\ \nu_{\ddot{\psi}} \end{bmatrix}$$

Note: The mean of the process noise is zero.

$$\text{Augmented Covariance Matrix} = P_{a,k|k} = \begin{bmatrix} P_{k|k} & 0 \\ 0 & Q \end{bmatrix}$$

Formula 2 Augmented state and Augmented Covariance matrix

$$x = \begin{bmatrix} p_x \\ p_y \\ v \\ \psi \\ \dot{\psi} \end{bmatrix}$$

If $\dot{\psi}_k$ is not zero

$$\text{State} = x_{k+1} = x_k + \begin{bmatrix} \frac{v_k}{\dot{\psi}_k}(sin(\psi_k + \dot{\psi}_k \Delta t) - sin(\psi_k)) \\ \frac{v_k}{\dot{\psi}_k}(-cos(\psi_k + \dot{\psi}_k \Delta t) + cos(\psi_k)) \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 cos(\psi_k)\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 sin(\psi_k)\nu_{a,k} \\ \Delta t \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 \nu_{\ddot{\psi},k} \\ \Delta t \cdot \nu_{\ddot{\psi},k} \end{bmatrix}$$

If $\dot{\psi}_k$ is zero

$$\text{State} = x_{k+1} = x_k + \begin{bmatrix} v_k cos(\psi_k)\Delta t \\ v_k sin(\psi_k)\Delta t \\ 0 \\ \dot{\psi}_k \Delta t \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{2}(\Delta t)^2 cos(\psi_k)\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 sin(\psi_k)\nu_{a,k} \\ \Delta t \cdot \nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 \nu_{\ddot{\psi},k} \\ \Delta t \cdot \nu_{\ddot{\psi},k} \end{bmatrix}$$

Formula 3 Sigma Point Prediction

Weights

$$w_i = \frac{\lambda}{\lambda + n_a}, i = 1$$

$$w_i = \frac{1}{2(\lambda + n_a)}, i = 2 \ldots n_\sigma$$

Predicted Mean

$$x_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i X_{k+1|k,i}$$

Predicted Covariance

$$P_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i (X_{k+1|k,i} - x_{k+1|k})(X_{k+1|k,i} - x_{k+1|k})^T$$

Formula 4 Predicated Mean and Covariance Calculation

Update Phase

UKF::UpdateRadar() applies Formula 5 to update for Radar measurement. UKF::UpdateLidar() is the counterpart for Lidar measurement update. UpdateRidar is simpler due to linear and only two measurements (x,y).

State Vector

Cross-correlation Matrix

$$T_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i (X_{k+1|k,i} - x_{k+1|k})\,(Z_{k+1|k,i} - z_{k+1|k})^T$$

Kalman gain K

$$K_{k+1|k} = T_{k+1|k} S_{k+1|k}^{-1}$$

Update State

$$x_{k+1|k+1} = x_{k+1|k} + K_{k+1|k}(z_{k+1} - z_{k+1|k})$$

Covariance Matrix Update

$$P_{k+1|k+1} = P_{k+1|k} - K_{k+1|k} S_{k+1|k} K_{k+1|k}^T$$

$$\dot{\rho} = \frac{p_x cos(\psi)v + p_y sin(\psi)v}{\sqrt{p_x^2 + p_y^2}}$$

Predicted Measurement Mean

$$z_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i Z_{k+1|k,i}$$

Predicted Covariance

$$S_{k+1|k} = \sum_{i=1}^{n_\sigma} w_i (Z_{k+1|k,i} - z_{k+1|k})(Z_{k+1|k,i} - z_{k+1|k})^T + R$$

$$R = E(w_k \cdot w_k^T) = \begin{bmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\varphi^2 & 0 \\ 0 & 0 & \sigma_{\dot{\rho}}^2 \end{bmatrix}$$

Formula 5 Formula for State and Covariance Update

Tools

Tools has two functions, CalculateRMSE and normalize_angle. CalculateRMSE calculates RMSE between ground truth and estimation, the same as Project 1. Normalize_angle normalizes angle to be between π and $-\pi$. Lecture uses while loop to get rid of 2π is not a practice solution.

```
if (x > M_PI) {
      x = fmod(x, pi2);
      if (x > M_PI)
         x -= pi2;
  }
  if (x < -M_PI) {
      x = fmod(x, -pi2);
      if (x < -M_PI)
         x += pi2;
  }
```

Results

Table 1 shows testing results when I ran using the simulator against dataset 1 and dataset2 including results from Radar or Lidar only tests. Chart 1 shows charts for results from dataset 1. Std_a 0.5 and std_yaw 0.6 seems to have good results for both datasets. I set them in the UKF C++ initialization.

		RMSE	Dataset 1			Dataset 2				
std_a	std_yaw	X	Y	VX	VY	X	Y	VX	VY	
0.1	0.1	0.125	0.1352	0.4171	0.3164					
0.2	0.2	0.0741	0.1012	0.3542	0.2472					
0.3	0.3	0.0635	0.0918	0.338	0.2259					
0.4	0.4	0.0612	0.0879	0.3322	0.2171	0.0638	0.0604	0.3397	0.3026	
0.5	0.5	0.0612	0.0859	0.3302	0.2135	0.0635	0.0598	0.3389	0.3001	
0.5	0.6	0.0605	0.0862	0.3299	0.2131	0.0633	0.0591	0.3393	0.2992	
0.6	0.5	0.0624	0.0846	0.3302	0.213	0.0641	0.0609	0.3391	0.3009	
0.6	0.6	0.0617	0.0848	0.3299	0.2125	0.0638	0.0601	0.3395	0.2999	
0.7	0.7	0.0624	0.0842	0.3305	0.213	0.0643	0.0608	0.3408	0.3008	
0.8	0.8	0.0632	0.0838	0.3317	0.2145					
0.9	0.9	0.0639	0.0837	0.3332	0.2167					
1	1	0.0647	0.0836	0.335	0.2193					
2	2	0.0701	0.0857	0.3557	0.2539					
3	3	0.0748	0.0885	0.3827	0.3085					
4	4	0.0765	0.0915	0.4003	0.3292					
5	5	0.0787	0.0942	0.423	0.3654					
6	6	0.0806	0.0966	0.4461	0.4002					
8	8	0.0833	0.1005	0.49	0.4644					
10	10	0.0855	0.1037	0.5322	0.5239					
15	15	0.0895	0.1097	0.6273	0.6561					
20	20	0.0922	0.1139	0.7112	0.7706					
0.5	0.6	0.0899	0.0938	0.6029	0.2312	0.0833	0.0739	0.3832	0.3475	laser
		0.1536	0.1971	0.4278	0.3072	0.1831	0.2066	0.3871	0.4991	radar

Table 1 RMSE Results from Various Values of std_a and std_yaw

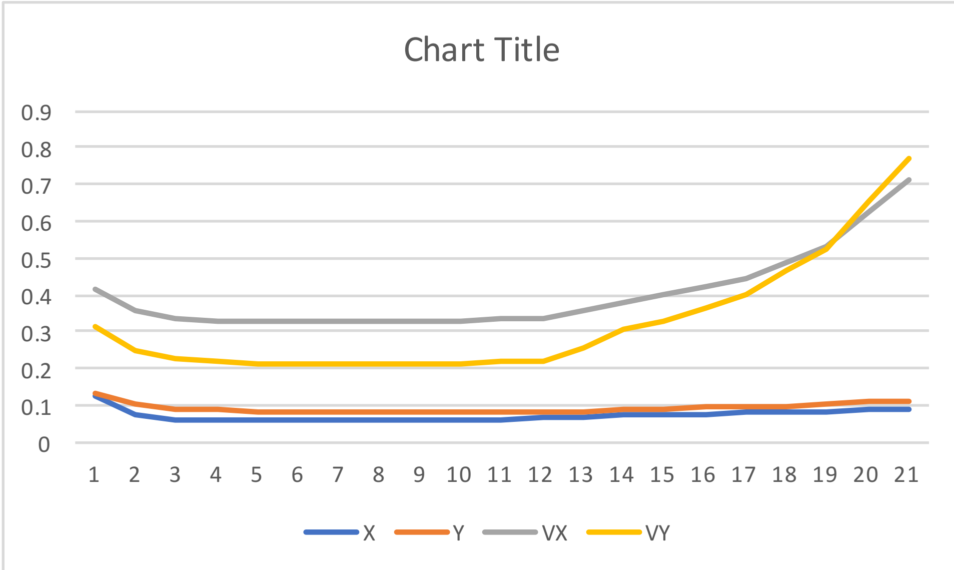

Chart 1 (X, Y, VX, VY) RMSE vs various std_a and std_yaw Dataset 1

Table 2 shows results from my EKF project (project 1). UKF has better results than EKF,

RMSE	x	y	Vx	Vy
Both	0.096198	0.0852897	0.413292	0.480286
Laser Only	0.122191	0.0983799	0.582513	0.456699
Radar Only	0.197623	0.264278	0.456697	0.679961

Table 2 Results from EKF (project 1)

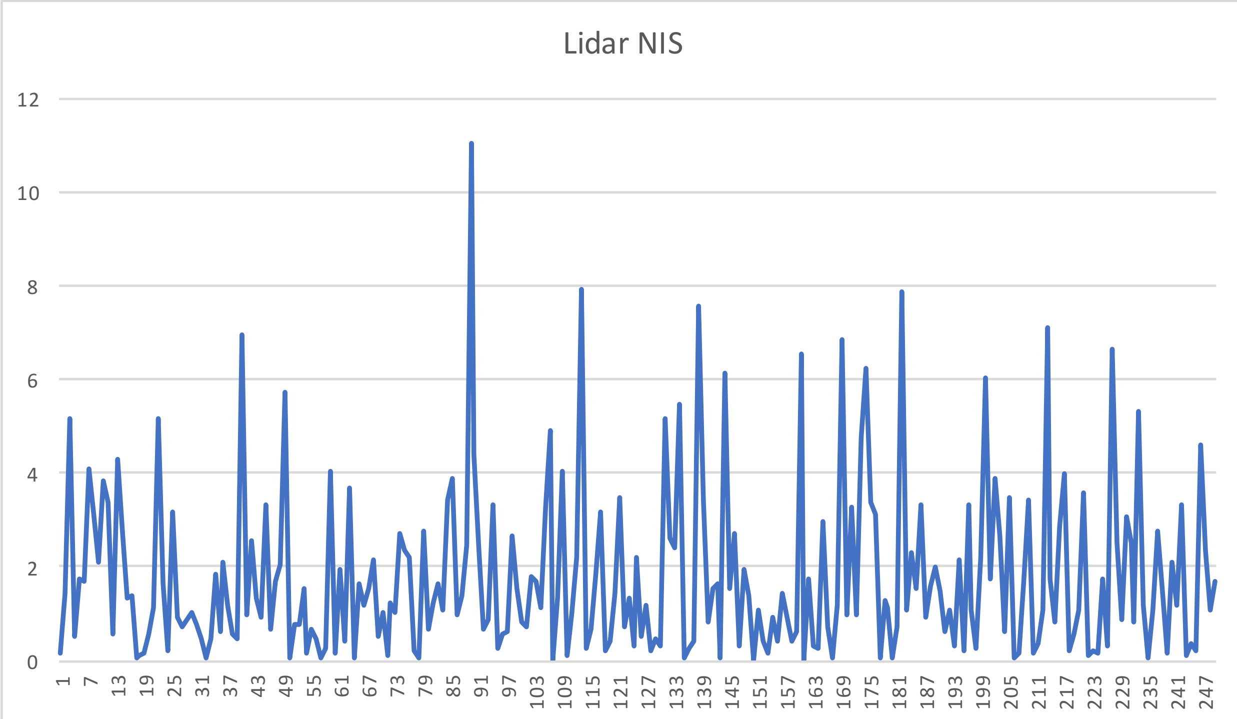

Chart 2 Lidar NIS for Dataset 2

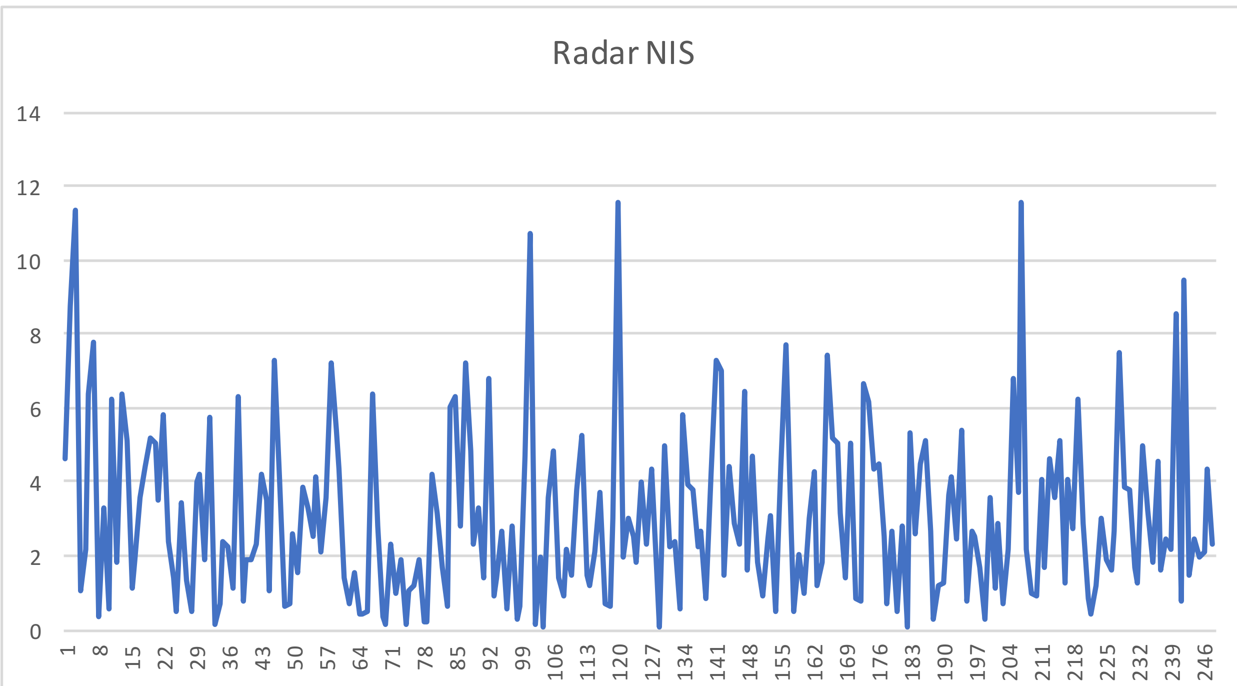

Chart 3 Radar NIS for Dataset 2