small number of distinct and marked features. This way the required memory usage can be decreased by several orders of magnitude (Kummerle et al. 2019).

In (Kuutti et al. 2018) it was demonstrated that, in terms of performance, LiDAR techniques hold the most promise for the localization of autonomous applications; however, the high power and processing requirements, as well as the high cost, make them unfeasible in terms of cost-efficiency and commercialization. As a result, more LiDAR technology optimization, or alternative approaches such as ground penetrating radar localization or vision-based localization within LiDAR maps could pave the way to commercially viable systems. However, before mass deployment, additional research work to validate the robustness of these systems, as well as validating their performance under a variety of driving conditions and refining operation parameters, will be needed.

In this work, a vehicle localization approach, referred to as RT_MCL, for urban environments is proposed. The RT_MCL relies on pole-like landmarks extracted from lidar/radar fused data. Pole-like landmarks typically take several forms such as traffic signs, telegraph poles, tree trunks, streetlamps, and bollards. Several features make them very convenient and reliable in-vehicle localization: a) they are abundant and widespread in urban areas, b) they rarely get dislocated and stay in their locations for long-time, c) they have well-defined rounded geometrical shapes that do not get affected with season or weather.

The RT_MCL matches the detected poles by the ego-car sensors with that being registered in a global reference map. This high-resolution map is being developed precisely off-line using lidar/radar data covering the same which the ego-car will drive on. The environment in the reference map is modeled, instead of as spatial poles of fixed positions, as probabilistic poles whereby every pole is represented by its Gaussian distribution over its position-prospect values. Consequently, Bayesian inference (employed by a particle filter) can favorably weight parts of the map most likely to be relevant to the ego-car pose, thereby reducing uncertainty and catastrophic errors. Moreover, to increase the precision of poles detection the RT_MCL uses the Unscented Kalman Filter (UKF) for sensors data fusion to combine the merits of both lidar and radar and reduces scattering.

In contemporary research, several endeavors are carried out to address the problem of vehicle localization utilizing pole-like landmarks extracted from lidar point-clouds. This problem can be divided into two subproblems: a) the first one is the pole detection and its position estimation, b) the second one is the pole-based ego-car pose estimation. For example, Weng et al. (Weng et al. 2018), developed a pole detector by partitioning the space around the ego-car and counts the number of reflected-scan points per voxel. Detecting poles is done by identifying the vertically connected stacks of voxels which all exceed a certain pre-specified threshold. Furthermore, the detector fits a cylinder to all the points associated with the identified stacks of voxels using RANSAC (Fischler et al. 1981). A particle filter combined with the nearest-neighborhood data association is then used for ego-car pose estimation. Another example, Sefati et al. (Sefati et al. 2017) focuses his approach of pole detection on removing the ground plane from the point-cloud received from sensors. The remaining points are projected onto a horizontal regular grid, then the neighboring cells are clustered based on occupancy and height, and afterward fit a cylinder to each cluster. The data association is done using the nearest-neighborhood and the pose estimation is performed using a particle filter. Kummerle et al. (Kummerle et al. 2019) refine further the pose estimate by augmenting the Sefati et al.'s pole detection method (Sefati et al. 2017) by fitting planes to building facades constructed from point clouds and fitting lines to lane markings in stereo camera images. Kummerle et al. (Kummerle et al. 2019) adopts a Monte Carlo method to solve the data association problem but uses nonlinear least-squares optimization to refine the estimated pose. Schaefer et. al. in (Schaefer et al. 2019) present a complete landmark-based localization system that relies on poles extracted from 3-D lidar data and is divided into 3 modules: the pole extractor, the mapping module, and the localization module. The pole detector does not only consider the laser ray endpoints, but also the free space in between the laser sensor and the endpoints, and to demonstrate reliable and accurate vehicle localization based on a map of pole landmarks on large time scales. The approach is tested on a long-term dataset that contains 35 hours of data recorded over 15 months – including varying routes, construction zones, seasonal and weather changes, and lots of dynamic objects.

The contribution of this paper can be enumerated as follows:

1. Tailoring the UKF algorithm to fuse multiple radars and lidars data to reduce scattering and achieve more accurate pose estimates for stationary pole-like objects around the ego car in real-time.
2. Tailoring the PF algorithm to employ pole-like landmarks for ego-car pose estimation in real-time.
3. Employing a high-order-generic-object-motion model (5 state variables that suits the most common road-objects in the development of the UKF and PF to generate more accurate estimates and improve the overall performance.
4. Representing the poles in the reference map in a probabilistic form that allows Bayesian inference implemented by the PF to contain map uncertainties and reduce localization errors.
5. Evaluating the gain of fusion by testing the UKF on three different cases (lidar+radar, lidar only, and radar only).
6. Evaluating the PF performance using different particle populations and under various map uncertainties.
7. Evaluating the real-time performance of both the UKF and PF on a moderate computational platform.
8. Employing the GB-DBSCAN clustering algorithm to detect potential objects from the lidar/radar fused data and finding their centroids.
9. Employing the RANSAC algorithm to extract the pole parameters by fitting circles (which represent poles shape) to the clusters identified by the GB-DBSCAN.
10. Employing the ICP algorithm for the data association between the detected poles in sensors data and the registered poles in the reference map.

II. THE RT_MCL METHOD OVERVIEW

The flowchart of the proposed RT_MCL autonomous car localization method is shown in Figure 1. The input to the proposed algorithm can be sorted into four items:

a) The reading of a Global Navigation Satellite Systems (GNSS) such as GPS integrated with that of an Inertial Motion Units (IMU) to provide the initial estimate of the ego-car pose. The GPS provides a low accuracy initial position and the IMU provides the incremental change in this position as well as the heading angle estimate. The main principle of GPS-IMU fusion is correcting accumulated errors of dead reckoning in intervals with absolute position readings (Suhr et al. 2017). The fused output is used in the initialization step of the particle filter.

b) The odometry readings in terms of the ego-car velocity and yaw rate are filtered from high-frequency noise and used as a control input to the particle filter.

c) The lidar/radar measurement data are fused using the proposed UKF to detect objects in the ego-car surroundings. The fused data are then clustered using the GB-DBSCAN algorithm (Dietmayer et al. 2012) to identify potential objects. The algorithm is tuned in a way to identify more likely pole-like static objects. The Doppler velocity of radar detections is used to discard detections originating from moving objects.

d) An off-line high definition 3-D point cloud reference map with identified pole-like landmarks coordinates is used as an input to the RT_MCL localizer. The identified pole-like objects by the GB-DBSCAN algorithm are then aligned with the identified pole-like landmarks in a process called "data association". In other words, the association of the coordinates of the poles resulted from the clustering with that from the map using the Iterative Closest Point (ICP) algorithm (Lu et al. 1997). This association is very crucial for localization accuracy.

The core of the RT_MCL localizer is the particle filter (Thrun, S. 2002), which will be explained later in detail. The particle filter uses the four inputs to determine the pose of the ego-car with much higher accuracy than the one received from the GPS/IMU fusion.

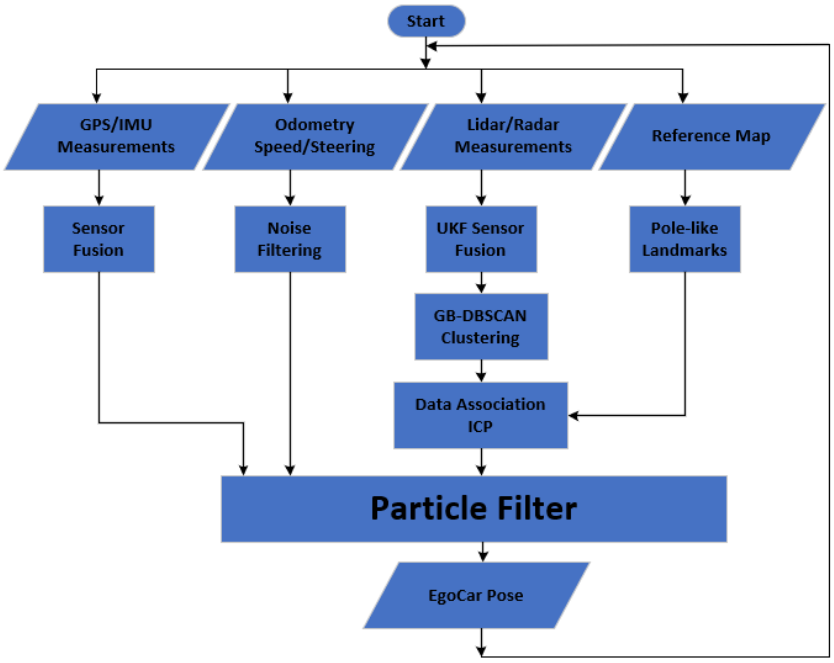

Figure 1. RT_MCL roadmap.

III. THE UNSCENTED KALMAN FILTER OVERVIEW

The Kalman Filter (KF) (Zarchan et al. 2013) is a system of equations working together to form a predictor-update cyclic optimal estimator that minimizes the estimated error covariance. The KF estimates the state $x \in R^n$ given the measurement $z \in R^m$ of a discrete-time controlled process that is modeled by a set of linear stochastic difference equations.

However, as KF is only limited to linear processes, and accordingly, it is not suitable to the radar measurement process which is inherently nonlinear. Therefore, the unscented Kalman filter is introduced (Wan et al. 2000) to overcome this limitation. The UKF is a derivative-free alternative to EKF (Einicke 1999) that uses a deterministic sampling approach. The UKF utilizes the predict-update two-step process as well, however, they are now augmented with further steps like generation and prediction of sigma points as shown in Figure 2.

In the UKF process, the state Gaussian distribution is represented using a minimal set of carefully chosen sample points, called sigma points. $n_x = 2n + 1$ sigma points are selected based on the following rule:

$$X_k = \left[x_k \;\; x_k + \sqrt{(\lambda + n_x)P_k} \;\; x_k - \sqrt{(\lambda + n_x)P_k}\right] \quad (1)$$

where X_k is the sigma-point matrix which includes n_x sigma-point vectors, λ is a design parameter that determines the

spread of the generated sigma points and usually takes the form $\lambda = 3 - n_x$.

In the sigma-point prediction step, each generated sigma point is inserted in the UKF nonlinear process model given in Eq. (2) to produce the matrix of the predicted (estimated) sigma points, which has an $n \times n_x$ dimension.

$$\hat{X}_{k+1} = f(X_k, v_k) \tag{2}$$

In the next step, the predicted state-mean and covariance matrices are calculated from the predicted sigma points as given in Eq. (3):

$$\begin{aligned} \hat{x}_{k+1} &= \sum_{i=0}^{n_x} w_i \hat{X}_{k+1,i} \\ \hat{P}_{k+1} &= \sum_{i=0}^{2n_x} w_i \left(\hat{X}_{k+1,i} - \hat{x}_{k+1}\right)\left(\hat{X}_{k+1,i} - \hat{x}_{k+1}\right)^T \end{aligned} \tag{3}$$

where $w_i's$ are the sigma-point weights that are used here to invert the spreading of the sigma points. These weights are calculated as shown in Eq. (4):

$$\begin{aligned} w_i &= \frac{\lambda}{\lambda + n_x}, \ i = 0 \\ w_i &= \frac{1}{2(\lambda + n_x)}, \ i = 1 \ldots n_x \end{aligned} \tag{4}$$

In the measurement prediction step, each generated sigma point is inserted in the UKF nonlinear measurement model given in Eq. (5) to produce the matrix of the predicted measurement sigma points, which has an $n \times n_x$ dimension.

$$\hat{Z}_{k+1} = h\left(\hat{X}_{k+1}\right) \tag{5}$$

In the next step, the predicted measurement-mean-and-covariance matrices are calculated from the predicted sigma points as well as the measurement noise covariance matrix R as given in Eq. (6):

$$\begin{aligned} \hat{z}_{k+1} &= \sum_{i=0}^{n_x} w_i \hat{Z}_{k+1,i} \\ S_{k+1} &= \sum_{i=0}^{2n_x} w_i \left(\hat{Z}_{k+1,i} - \hat{z}_{k+1}\right)\left(\hat{Z}_{k+1,i} - \hat{z}_{k+1}\right)^T + R \\ R &= E\{\omega_k . \omega_k^T\} \end{aligned} \tag{6}$$

where $w_i's$ are the sigma-point weights that are determined using Eq. (4).

The final step is the UKF state update, where the UKF gain matrix (K) is calculated as in Eq. (7) using the calculated cross-correlation matrix (T) between the sigma points in the state space and the measurement space. The gain is used to update the UKF state vector (x) as well as the state covariance matrix (P).

$$\begin{aligned} T_{k+1} &= \sum_{i=0}^{2n_x} w_i \left(\hat{X}_{k+1,i} - \hat{x}_{k+1}\right)\left(\hat{Z}_{k+1,i} - \hat{z}_{k+1}\right)^T \\ K_{k+1} &= T_{k+1} S_{k+1}^{-1} \\ x_{k+1} &= \hat{x}_{k+1} + K_{k+1}(\hat{z}_{k+1} - z_{k+1}) \\ P_{k+1} &= \hat{P}_{k+1} - K_{k+1} S_{k+1} K_{k+1}^T \end{aligned} \tag{7}$$

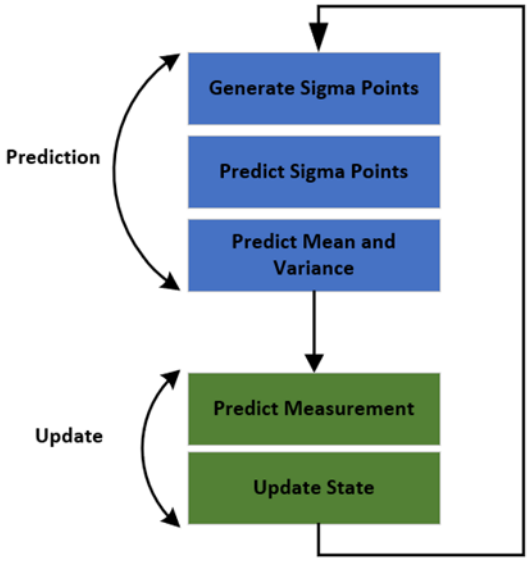

Figure 2. UKF roadmap.

IV. THE MOVING OBJECT MODEL

The state of the moving object (Schubert et al. 2008) is determined by the five variables grouped into the state vector x shown in Eq. (14), where p_x, and p_y are the object position in the x and y-axis respectively as shown in Figure 2, v is the

magnitude of object velocity derived from its x and y components, v_x and v_y respectively. ψ is the yaw angle (object orientation) and $\dot{\psi}$ is rate of change of the object-yaw angle.

$$x = \begin{bmatrix} p_x \\ p_y \\ v \\ \psi \\ \dot{\psi} \end{bmatrix}, v = \sqrt{v_x^2 + v_y^2}, \psi = tan^{-1}\frac{v_y}{v_x} \tag{8}$$

The nonlinear $x_{k+1} = f(x_k, \nu_k)$ difference equation that describes the motion model of the object is derived based on the state vector x and presented in Eq. (15) and (16).

$$x_{k+1} = x_k + \begin{bmatrix} \frac{v_k}{\dot{\psi}_k}\left(sin\left(\psi_k + \dot{\psi}_k \Delta t\right) - sin(\psi_k)\right) \\ \frac{v_k}{\dot{\psi}_k}\left(-cos\left(\psi_k + \dot{\psi}_k \Delta t\right) + cos(\psi_k)\right) \\ 0 \\ \Delta t \\ 0 \end{bmatrix} + \nu_k \tag{9}$$

$$\nu_k = \begin{bmatrix} \frac{1}{2}(\Delta t)^2 cos(\psi_k).\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2 sin(\psi_k).\nu_{a,k} \\ \Delta t.\nu_{a,k} \\ \frac{1}{2}(\Delta t)^2.\nu_{\ddot{\psi},k} \\ \Delta t.\nu_{\ddot{\psi},k} \end{bmatrix} \tag{10}$$

$$\begin{aligned} &\Delta t = t_{k+1} - t_k \\ &\nu_{a,k} \sim \mathcal{N}(0, \sigma_a^2) \\ &\nu_{\ddot{\psi},k} \sim \mathcal{N}(0, \sigma_{\ddot{\psi}}^2) \end{aligned} \tag{11}$$

where Δt is the time difference between two consecutive samples, $\ddot{\psi}$ is the yaw acceleration, a is the longitudinal acceleration, $\nu_{a,k}$ is the longitudinal acceleration noise at sample k with a standard deviation σ_a^2, $\nu_{\ddot{\psi},k}$ is the yaw acceleration noise at sample k with a standard deviation $\sigma_{\ddot{\psi}}^2$.

If the $\dot{\psi}$ is zero, and to avoid dividing by zero in Eq. (9), the following approximation is used to calculate the prediction of p_x, and p_y:

$$\begin{aligned} p_{x_{k+1}} &= p_{x_k} + v_k cos(\psi_k)\Delta t \\ p_{y_{k+1}} &= p_{y_k} + v_k sin(\psi_k)\Delta t \end{aligned} \tag{12}$$

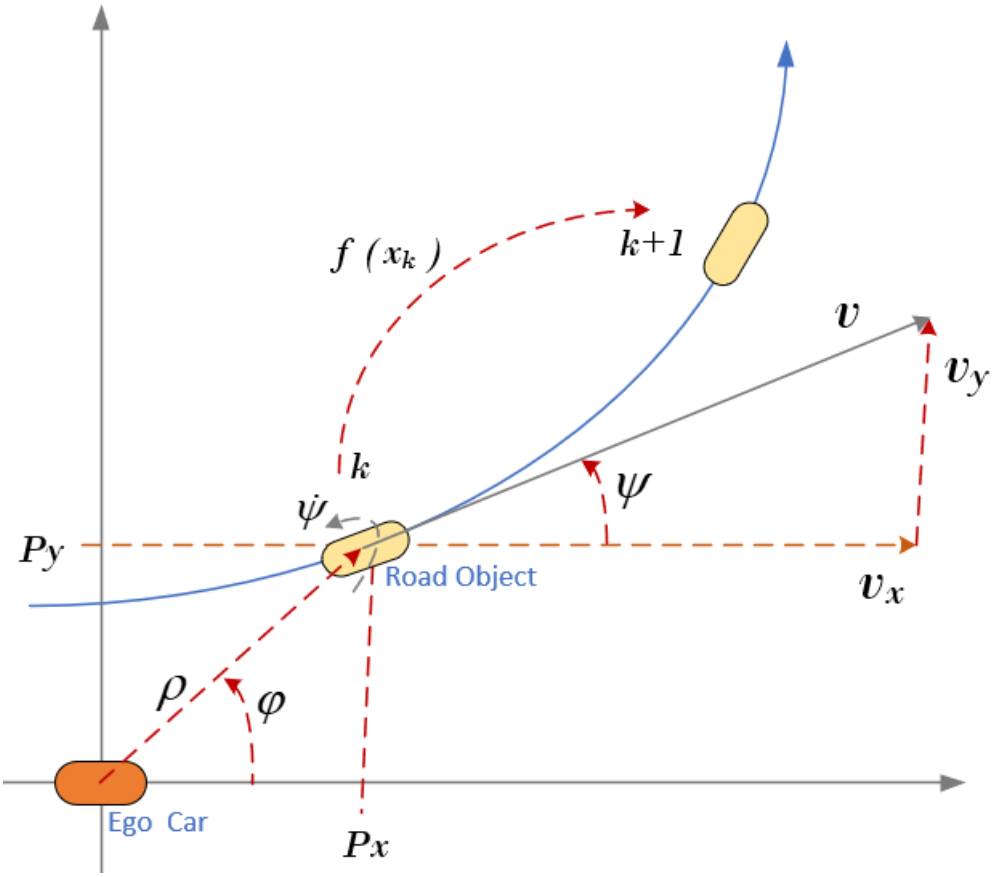

Figure 3. An object motion model.

V. SENSOR FUSION USING UKF

The processed lidar measurement vector includes the moving object centroid position (p_x and p_y) in cartesian coordinates (Eq. (13)), while the radar measurement vector includes the moving object centroid position (ρ, φ) and radian velocity ($\dot{\rho}$) in polar coordinates as represented by Eq. (14). The mapping function that specifies how lidar Cartesian coordinates got mapped to the radar polar coordinates is given as well in Eq. (15).

$$z_{lidar} = \begin{pmatrix} p_x \\ p_y \end{pmatrix}, z_{radar} = \begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix} \tag{13}$$

$$h(x) = \begin{pmatrix} \rho \\ \varphi \\ \dot{\rho} \end{pmatrix} = \begin{pmatrix} \sqrt{p_x^2 + p_y^2} \\ arctan\left(\frac{p_y}{p_x}\right) \\ \frac{p_x v_x + p_y v_y}{\sqrt{p_x^2 + p_y^2}} \end{pmatrix} \tag{14}$$

$$p_x = \rho cos(\varphi),\ p_y = \rho sin(\varphi) \tag{15}$$

According to Figure 3, each sensor has its own prediction update scheme, however, both sensors share the same state prediction scheme. The belief about the object's position and velocity is updated asynchronously each time the measurement is received regardless of the source sensor.

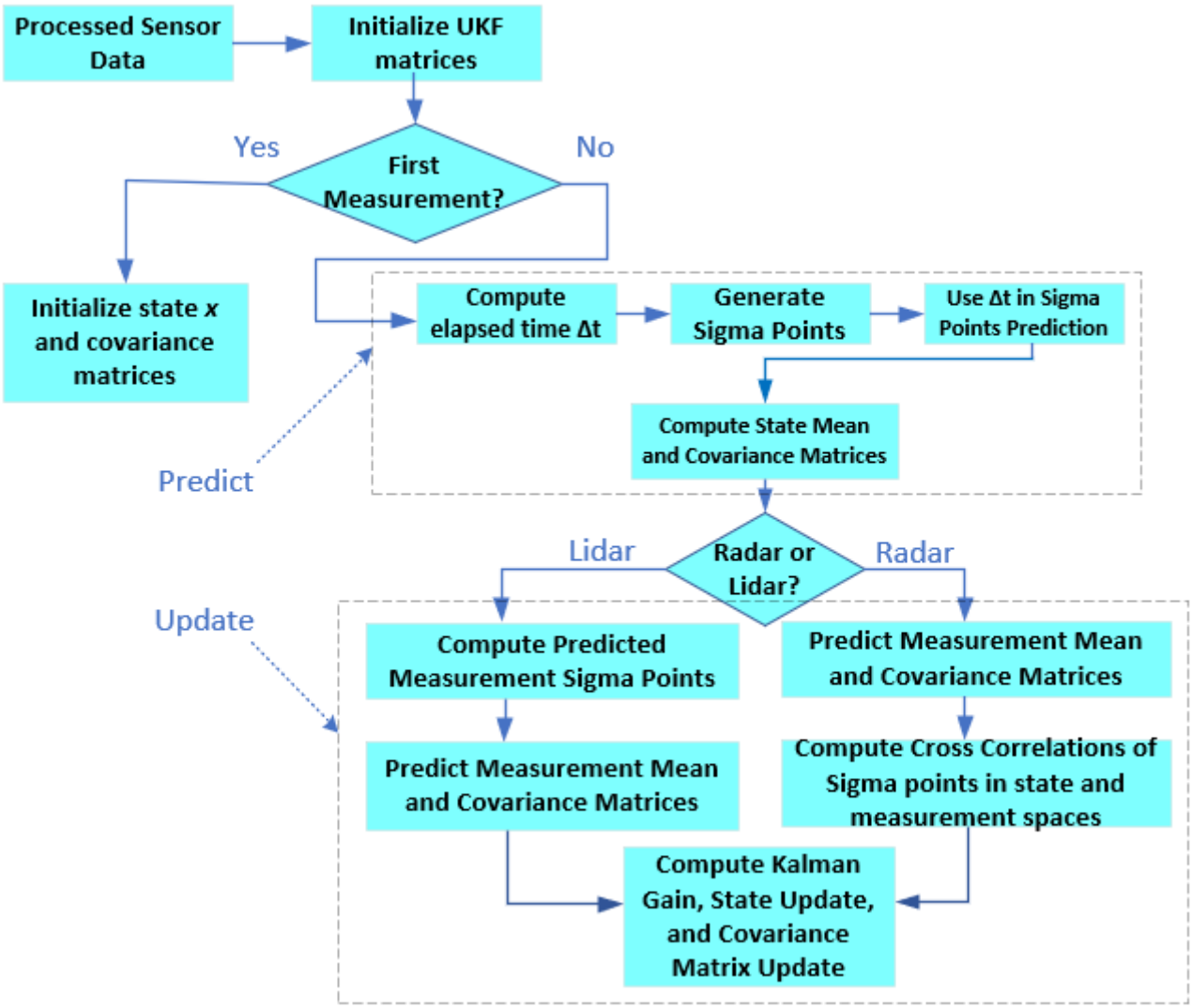

Figure 4. Lidar and radar data fusion using UKF.

In Figure 3, after computing the elapsed time between consecutive sensor reading (Δt), the sigma points (X_k) are generated using Eq. (1), and then a next-time-step prediction for sigma points ($\hat{X}_{k+1}$) is carried out using Eq. (2) while employing the moving object nonlinear motion model given in Eq. (9). The resulted predicted sigma points are then used to compute the state mean ($\hat{x}_{k+1}$) and covariance ($\hat{P}_{k+1}$) matrices using Eq. (3).

Then the fusion technique thus branches into two directions based on the source of the last sensor data measurement. If the source is a radar and employing the nonlinear radar measurement model (Eq. (14)), the predicted measurement sigma points ($\hat{Z}_{k+1}$) are calculated from the predicted state sigma points ($\hat{X}_{k+1}$) using Eq. (5). Then, the predicted measurements ($\hat{z}_{k+1}$) and their covariance matrix (S_{k+1}) are calculated based on Eq. (6) using the measurement noise covariance matrix R_{radar} given in Eq. (15). Then, $\hat{x}_{k+1}$ and $\hat{z}_{k+1}$ and are used to compute the cross-correlation matrix (T_{k+1}) between the sigma points in the state space ($\hat{X}_{k+1}$) and the measurement space ($\hat{Z}_{k+1}$) as in Eq. (7). Based on this cross-correlation matrix, the Kalman filter gain (K_{k+1}) is then calculated and used to compute the updated object's state vector (x_{k+1}) and covariance matrix (P_{k+1}) as shown by Eq. (7).

$$R_{radar} = \begin{bmatrix} \sigma_\rho^2 & 0 & 0 \\ 0 & \sigma_\varphi^2 & 0 \\ 0 & 0 & \sigma_{\dot{\rho}}^2 \end{bmatrix} \tag{16}$$

where σ_ρ is the noise standard deviation of the object radial distance, σ_φ is the noise standard deviation of the object heading (bearing), $\sigma_{\dot{\rho}}$ is the noise standard deviation of the object yaw rate.

If the measurement data source is a lidar sensor and employing the linear lidar measurement model (H_{lidar}) shown in Eq. (16), the predicted measurement sigma points ($\hat{Z}_{k+1}$) is directly calculated from ($\hat{X}_{k+1}$). Then, the predicted measurements ($\hat{z}_{k+1}$) and their covariance matrix (S_{k+1}) are calculated based on Eq. (6) using the measurement noise covariance matrix R_{lidar} given in Eq. (17). Then, $\hat{x}_{k+1}$ and $\hat{z}_{k+1}$ are used to compute the cross-correlation matrix (T_{k+1}) between the sigma points in the state space ($\hat{X}_{k+1}$) and the measurement space ($\hat{Z}_{k+1}$) as in Eq. (7). Based on this cross-correlation matrix, the Kalman filter gain (K_{k+1}) is then calculated and used to compute the updated object's state vector (x_{k+1}) and covariance matrix (P_{k+1}) as shown by Eq. (7).

$$H_{lidar} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$
$$R_{lidar} = E[\omega . \omega^T] = \begin{bmatrix} \sigma_{p_x}^2 & 0 \\ 0 & \sigma_{p_y}^2 \end{bmatrix} \quad (17)$$

where σ_{p_x} and σ_{p_y} are the noise standard deviations for the object x and y positions respectively.

VI. CLUSTERING AND DATA ASSOCIATION

Figure 3 presents the lidar and radar data fusion technique employing the UKF. The UKF produces point clouds that provide information about objects in the ego-car surrounding. Clustering is a key tool to extract these objects' information (geometry and poses) from UKF point clouds. The objective of the clustering is to represent each object in the form of a source-point model to reduce computational cost and memory requirements.

UKF data processing is performed using clustering and association algorithms. DBSCAN (Sander et. al. 1996) is an unsupervised clustering algorithm that groups together data points if the density of the points is high enough. It requires two parameters to determine the density. The first parameter is ε describing the radial distance from a point p, that is being evaluated. The second parameter is *minPts*, which is the least number of detections that must be within a distance ε from p, including p itself, to form a cluster. By choosing ε and *minPts*, it is then possible to decide the necessary density for a group of points to form a cluster, however, these fixed parameters are not convenient if various types and topologies of road objects need to be detected. As an improvement to this algorithm, GB-DBSCAN is introduced (Dietmayer et al. 2012). It works in the same manner as DBSCAN but does not have fixed parameters. Instead, a polar grid is created according to the radial and angular resolution of the sensor. Instead of looking at a circular search area with a fixed radius, GB-DBSCAN can use a more dynamic, elliptic, search area. The most distinctive feature of a pole-like object is that the density of point cloud at its position is far greater than its surrounding.

While GB-DBSCAN is used for coarse clustering, the RANSAC (Fischler et al. 1981) is used to fine-tune the clustering and associate geometrical shape proposals to potential coarse clusters. Therefore, RANSAC here is used to fit a circle (which represents poles shape) to all points (N) in each cluster. After successful fitting, RANSAC can extract the pole parameters (center ($\hat{x}_c, \hat{y}_c$) and radius ($\hat{r}$)) from the fitted circles by solving the following equation:

$$\min \left\{ \frac{1}{N} \sum_{i=1}^{N} \left[\sqrt{(x_i - \hat{x}_c)^2 + (y_i - \hat{y}_c)^2} - \hat{r} \right]^2 \right\} \quad (18)$$

Establishing a matching link between the detected pole-like objects in the lidar/radar data (the source) and the pole-like landmarks in the reference map (the target) is referred to as the data association step. This step is necessary for the proper execution of the particle filter. The data association step is carried out using the Iterative Closest Point (ICP) algorithm. Instead of working on the whole point-clouds in both the source and target as per the standard application of this algorithm (Lu et al. 1997), only the centroids of pole-like objects are considered for the matching process. This way a huge memory and processing time will be saved.

The ICP algorithm works by iterating a two-step procedure until convergence. The first step is matching each point in a set of source points, X (lidar/radar data), to the closest point in a set of target points, Y (reference map), and the second step is finding the optimal transform between the source and target sets, given the assignments. Matching points by distance is a computationally efficient operation if a k-d tree data structure is used to store Y. Here, the points in X is denoted as x_i and the matched (closest) point from x_i in Y is denoted as y_i. The basic 2D ICP version minimizes the sum of squared distances between source and target points to find the rotation angle φ, encoded by a rotation matrix $R(\varphi)$, and the translation t.

$$min_{\varphi,t} \left\{ \sum_{i=1}^{N} \left(y_i - (R(\varphi)x_i - t) \right)^T \left(y_i - (R(\varphi)x_i - t) \right) \right\} \quad (19)$$

To perform the data association, both observations and landmarks in the reference map should have the same coordinate system. Observations in the ego-car coordinate system (x_c and y_c) can be transformed into map coordinates x_m and y_m by ratifying them through a homogenous transformation matrix shown in Eq. (20) that performs rotation and translation using map particle/ego-car coordinates (x_p and y_p), and the rotation angle θ.

$$\begin{bmatrix} x_m \\ y_m \\ 1 \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta & x_p \\ sin\theta & cos\theta & y_p \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} \quad (20)$$

VII. PARTICLE FILTER OVERVIEW

Particle filtering uses a finite set of particles to represent the posterior distribution $bel(x_t)$ of some stochastic process given noisy and/or partial observations $p(z_t|x_t)$. It is an approximate realization of the recursive Bayesian filter stated in Eq. (21)

where ζ is a normalization factor.

$$bel(x_t) \leftarrow \zeta\ p(z_t|x_t)\ bel(x_{t-1}) \tag{21}$$

Typically, the number of particles, M should be large enough to represent the belief $bel(x_t)$ accurately to some extent. The set of particles are denoted at time step t as

$$\chi_t = \left\{x_t^{[i]} | 1 \le i \le M\right\} \tag{22}$$

Each particle $x_t^{[i]}$ is a hypothesis about the actual state at time t. TABLE I describes a simple implementation of the particle filter and Figure 5 depicts the whole flowchart. The recursive procedure is performed whenever a measurement update (z_t) along with a new set of odometry data (u_t) becomes available.

TABLE I. PARTICLE FILTER PSEUDO CODE.

Procedure *Particle Filter* (χ_{t-1}, u_t, z_t):
Input: Set of particles χ_{t-1} at time $(t-1)$, control inputs u_t, and a set of measurements z_t. **Output:** The updated set of particles χ_t at time t.
Begin 1. Initialize Particles: $\bar{\chi}_t = \chi_t = \emptyset$. 2. *For* $m = 1$ *to* M *do* i. *generate* $x_i^{[m]} \sim p\left(x_t | u_t, x_{t-1}^{[m]}\right)$ ii. $w_t^{[m]} = p\left(z_t | x_i^{[m]}\right)$ iii. $\bar{\chi}_t = \bar{\chi}_t + \langle x_i^{[m]}, w_t^{[m]} \rangle$ iv. End for loop 3. *For* $m = 1$ *to* M *do* i. *draw* i *with probability* $\alpha w_t^{[i]}$ ii. *add* $x_t^{[i]}$ *to* χ_t iii. End for loop 4. Return χ_t End.

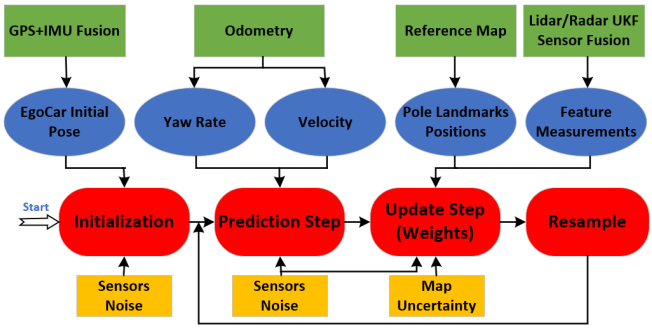

Figure 5. Particle Filter Algorithm Flowchart.

The prediction step is implemented by the loop at line 2. One state hypothesis ($x_i^{[m]}$) is generated for each particle based on its current state and the state transition distribution $p\left(x_t | u_t, x_{t-1}^{[m]}\right)$ which is computed using the particle (car) motion model described in Section IV. An importance factor (weight) $w_t^{[i]}$ is calculated or updated for each newly generated hypothesis using a multivariate Gaussian probability density function for each observation and combine the likelihood of all the observations by taking their products as given in Eq. (23):

$$w_t^{[m]} = \prod_{i=1}^{N} \frac{exp\left(-\frac{1}{2}\left(z_i^{[t]} - \mu_i^{[t]}\right)^T \Sigma^{-1} \left(z_i^{[t]} - \mu_i^{[t]}\right)\right)}{\sqrt{|2\pi\Sigma|}} \tag{23}$$

where $z_i^{[t]}$ is the i^{th} landmark observation for particle m at step t, $\mu_i^{[t]}$ is the predicted (mean) measurement for the landmark corresponding to the i^{th} observation at step t, Σ is the covariance matrix of the measurements, and N is the total number of measurements for one particle.

Afterward, in lines 3, new M particles are resampled from the previous set of particles ($\bar{\chi}_t$) proportionally to their importance factors ($\alpha w_t^{[i]}$) that have been determined in line 2.ii, where α is a normalization factor. This resampling yield χ_t, the updated posterior approximation. Note that χ_t generally will contain duplicates, taking the places of particles that were not drawn in line 3.i as they have evolved into less likely hypotheses.

To check the convergence of the particle filter, the weighted-average error ($Error_{weighted}$) of all the particles is used as a convergence indicator. The $Error_{weighted}$ is computed as given in Eq. (24), by simply calculating the root squared error between each particle state p_i and the ground truth g and multiply it by the particle's weight, and then sum the product for all

particles and divide the summation by the aggregated particle weights.

$$Error_{weighted} = \frac{\sum_{i=1}^{M} w_i \sqrt{|p_i - g|}}{\sum_{i=1}^{M} w_i} \quad (24)$$

VIII. IMPLEMENTATION OF THE RT_MCL

Both UKF and PF are implemented using the high-performance language GCC C++ (GCC C++ 2020) on Ubuntu Linux operating system (Ubuntu Linux 2020). This combination is fitted for the required real-time performance (Nagiub et al. 2013). A C++ numerical solver, matrix, and vector operations package "Eigen" (Eigen 2020) is used to numerically calculate the object model and effectively performing the predict and update steps.

The object motion model described by Eq. (9-11) includes several noise parameters that need to be carefully set. Table II presents the fine-tuned parameters for both UKF and PF.

TABLE II. THE UKF AND OBJECT MODEL NOISE PARAMETERS.

Parameter	UKF/PF	Parameter	UKF
σ_a *m/sec²*	1.0	σ_{p_y} (lidar) *m*	0.15
$\sigma_{\ddot{\psi}}$ *rad/sec²*	0.6	σ_{ρ} (radar) *m*	0.3
$\sigma_{\dot{\psi}}$ *rad/sec*	0.06	σ_{φ} (radar) *rad*	0.03
σ_{p_x} (lidar) *m*	0.15	$\sigma_{\dot{\rho}}$ (radar) *m/sec*	0.3

The UKF design is considered consistent if the estimation error is unbiased, i.e. has zero-mean, and that the actual mean square error of the filter matches the filter-calculated state covariance. As a measure of filter consistency, the time-average Normalized Innovation Squared (NIS) (Piché, R. 2016) can be used to finetune the noise parameters. The metric, described by Eq. (25), is used to calculate the NIS value at each sample k and then averaging these values ($NIS_{Average}$) over a moving window of measurements of length N.

$$NIS_k = (z_{k+1} - \hat{z}_k)^T S_k^{-1} (z_{k+1} - \hat{z}_k)$$
$$NIS_{Average} = \frac{1}{N} \sum_{k=1}^{k=N} NIS_k \quad (25)$$

The proper initialization of the UKF is very crucial to its subsequent performance (Zhao et al. 2017). The main initialized variables are the state estimate vector (x) and its estimate covariance matrix (P). The first two terms of the state vector x given by Eq. (8) are p_x and p_y which are simply initialized using the first received raw sensor measurement. For the other three terms of the state vector, intuition augmented with some trial-and-error is used to initialize these variables as listed in Table III. The state covariance matrix is initialized as a diagonal matrix that contains the covariance of each variable estimate (Eq. (26)).

$$P = diag\left(\sigma_{\hat{p}_x}^2, \sigma_{\hat{p}_y}^2, \sigma_{\hat{v}}^2, \sigma_{\hat{\psi}}^2, \sigma_{\hat{\dot{\psi}}}^2\right) \quad (26)$$

TABLE III. INITIALIZATION OF UKF STATES.

Parameter	UKF	Parameter	UKF
p_x *m*	1st raw x-reading	p_y *m*	1st raw y-reading
v *m/sec*	0.0	ψ *rad*	0.0
$\dot{\psi}$ *rad/sec*	0.0	$\sigma_{\hat{p}_x}$ *m*	1.0
$\sigma_{\hat{p}_y}$ *m*	1.0	$\sigma_{\hat{v}}$ *m/sec*	$\sqrt{1000}$
$\sigma_{\hat{\psi}}$ *rad*	$\sqrt{1000}$	$\sigma_{\hat{\dot{\psi}}}$ *m/sec²*	$\sqrt{1000}$

To check the performance of the UKF, in terms of how far the estimated results from the true results (ground truth), the Root Mean Squared Error (RMSE) given in Eq. (27) is used. The metric is calculated over a moving window of measurements of length N. x_k^{est} is the estimated state vector of the UKF, and x_k^{true} is the true state vector supplied by the simulator or given as training data during the UKF design phase.

$$RMSE = \sqrt{\frac{1}{N} \sum_{k=1}^{k=N} (x_k^{est} - x_k^{true})^2} \quad (27)$$

The initialization of the PF is very crucial as well for its performance. Therefore, it is carried out as follows:

a) The number of particles is set to $M = 50$. In the literature (Thrun, S. 2002; Levinson et al. 2007), this number usually ranges from 100 to 1000, however, it is a compromise between accuracy and computational speed. Several experimental trials are carried out and show that 50 produces the required accuracy and real-time performance.

b) The PF state vector (particles poses) are initialized from the output of the GPS+IMU fusion ($p_{x_{GPS}}, p_{y_{GPS}}, \theta_{GPS}$) as follows:

$$p_x^{[m]} \sim \mathcal{N}(p_{x_{GPS}}, \sigma_{x_{GPS}}^2 + \sigma_{x_{artificial}}^2)$$
$$p_y^{[m]} \sim \mathcal{N}(p_{y_{GPS}}, \sigma_{y_{GPS}}^2 + \sigma_{y_{artificial}}^2) \quad (28)$$
$$\theta_{Particle}^{[m]} \sim \mathcal{N}(\theta_{GPS}, \sigma_{\theta_{GPS}}^2 + \sigma_{\theta_{artificial}}^2)$$

where $p_x^{[m]}$, $p_y^{[m]}$ and $\theta_{Particle}^{[m]}$ represent particle m initialized pose. $\sigma_{x_{GPS}}$, $\sigma_{y_{GPS}}$, and $\sigma_{\theta_{GPS}}$ represent the GPS+IMU reading noise standard deviations. $\sigma_{x_{artificial}}$, $\sigma_{y_{artificial}}$, and $\sigma_{\theta_{artificial}}$ represent the artificial noise added to each initial

particle position for the purpose of randomization that helps in the conversion of the PF [25] (). The values of these parameters are listed in Table IV.

c) The particle importance weights are all initialized with the uniform distribution $w^{[m]} = \frac{1}{M}$.

d) As RT_MCL is using probabilistic maps, and each pole-like landmark is represented by Gaussian distributions $\mathcal{N}(p_{x_{Pole}}, \sigma^2_{x_{Pole}})$ and $\mathcal{N}(p_{y_{Pole}}, \sigma^2_{y_{Pole}})$ to model the uncertainties in their positions. The values of the standard deviations $\sigma_{x_{pole}}$ and $\sigma_{y_{pole}}$ are listed in Table IV.

TABLE IV. INITIALIZATION OF THE PARTICLE FILTER.

Parameter	PF	Parameter	PF
$\sigma_{x_{GPS}}$	0.3 m	$\sigma_{x_{artificial}}$	10 m
$\sigma_{y_{GPS}}$	0.3 m	$\sigma_{y_{artificial}}$	10 m
$\sigma_{\theta_{GPS}}$	0.01 rad	$\sigma_{\theta_{artificial}}$	0.05 rad
$\sigma_{x_{pole}}$	0.3 m	$\sigma_{y_{pole}}$	0.3 m

To check the performance of the PF, in terms of how far the estimated poses from the ground truth, the cumulative mean absolute error for each pose variable given in Eq. (29) is used. The metric is calculated over a moving window of measurements of length N. $x_i^{best}, y_i^{best}, \theta_i^{best}$ are the estimated pose variables of the PF, and $x_i^{gt}, y_i^{gt}, \theta_i^{gt}$ are the ground truth variables supplied by the simulator or given as training data during the PF design phase.

$$\begin{aligned} X_{error} &= \frac{1}{N}\sum_{i=1}^{N}\left|x_i^{best} - x_i^{gt}\right| \\ Y_{error} &= \frac{1}{N}\sum_{i=1}^{N}\left|y_i^{best} - y_i^{gt}\right| \\ Yaw_{error} &= \frac{1}{N}\sum_{i=1}^{N}\left|\theta_i^{best} - \theta_i^{gt}\right| \end{aligned} \tag{29}$$

IX. TESTING AND EVALUTION RESULTS

Extensive trials-and-errors attempts are used to tune the hyper-parameters of the *RT_MCL*. However, to be more consistent and accurate, numerical Key Performance Indicators (KPIs*)* are constructed and coded as in Eq. (24), Eq. (26) and Eq. (28) to evaluate the performance of the localization technique under the given set of hyper-parameters (Farag, W. 1998).

Several test tracks have been used to evaluate the performance of the *RT_MCL* under different sets of hyper-parameters in an iterative tuning process. An example of these test tracks is shown in Figure 6. This track is 754-meter long with several curvatures and includes 42 pole-like landmarks to emulate urban driving.

Table III presents the testing results of the lidar/radar fusion algorithm that uses the UKF. The performance evaluation is carried out on test tracks to detect road objects (cyclists, cars, pedestrians, and pole-like landmarks). The RMSE KPI (Eq. (26)) is used to evaluate the UKF performance based on the five state variables: p_x, p_y, v_x, v_y, and ψ. The KPI is comparing each estimated state variable to its ground-truth value and finding the error. The lower the value of the KPI the better the performance.

TABLE V. PERFORMANCE EVALUATION OF THE UKF.

State var	Cyclist	Car	Pedestrian	Pole
p_x	0.0648	0.1857	0.0652	0.0324
p_y	0.0809	0.1899	0.0605	0.0433
v_x	0.1452	0.4745	0.5332	0.0032
v_y	0.1592	0.5075	0.5442	0.0054
ψ	0.0392	0.2580	0.2075	0.0075

To assess the significance of the fusion between lidar and radar in tracking. The UKF is tested in one time with measurements from lidar alone, and another time with measurements from radar alone. The results reported in Table V show how fusion makes the difference and substantially improves accuracy. The estimation of all state variables is spectacularly improved. For example, the RMSE of x-position (p_x) estimation is reduced by 60% compared to "lidar-alone" and 60% compared to "radar-alone" estimations. Moreover, the RMSE of x-velocity (v_x) estimation is reduced by 30% compared to "lidar-alone" and 26% compared to "radar-alone" estimations. The NIS values are calculated as well for "lidar-alone" and "radar-alone" cases to test the consistency of the UKF in their cases. The reported values show that fusion significantly improves the consistency. The NIS values that exceed the 95%-threshold have been reduced by 31% compared to the "lidar-alone" and 38.5% compared to the "radar-alone" ones.

TABLE VI. SENSOR FUSION EVALUATION OF THE UKF (BICYCLE TRACK).

	Lidar+Radar	Lidar Only	Radar Only
RMSE - p_x	0.0648	0.1612	0.2031
RMSE - p_y	0.0809	0.1464	0.2539
RMSE - v_x	0.1452	0.2082	0.1971

RMSE - v_y	0.1592	0.2129	0.1871
RMSE - ψ	0.0392	0.0540	0.0480
NIS - Average	2.2797	1.6941	2.6576
NIS - Min	0.0012	0.04874	0.11309
NIS - Max	14.749	12.997	12.183
NIS > 95% Threshold	2.2%	3.2 %	5.2 %

Figure 6, Figure 7, and Figure 8 present an example of the testing and simulation results of the MCL algorithm that uses the combination of the UKF and PF while employing a probabilistic reference map. The performance evaluation is carried out on test tracks to detect road pole-like landmarks using the UKF and using these detections by the GB-DBSCAN and the PF to localize the ego-car on the global map. In the mentioned figure, both the estimated-pose values and the ground truth are drawn on top of each other due to the reported small errors as displayed in Table VII. The RMSE KPI (Eq. (28)) is used to evaluate the PF performance based on the three pose variables: x, y, and θ. The KPI is comparing each estimated state variable to its ground-truth value and finding the error. The lower the value of the KPI the better the performance. Several experiments have been carried out using the PF with different numbers of particles to optimize its real-time performance. The results in Table VII show that the number of particles can go down to "25", and the PF can still produce good results, fast execution time with robust convergence (while "15" is divergent). However, to ensure more robustness, the number of particles of "50" is considered the most appropriate selection with a delicate balance between the achieved accuracies, real-time performance, and convergence. It is clear from the table that above 50 not many improvements in precision are achieved and a margin of safety is required to enhance robustness, therefore 25 is not selected.

TABLE VII. THE PARTICLE FILTER WITH DIFFERENT NUMBER OF PARTICLES.

#Particles	x-error	y-error	Yaw-error	Exec. Time
15	122.34	33.002	1.5959	0.268 ms
25	0.1382	0.1240	0.0048	0.486 ms
50	0.1143	0.1154	0.0040	0.739 ms
100	0.1154	0.1071	0.0037	1.224 ms
150	0.1098	0.1060	0.0037	2.086 ms
200	0.1102	0.1039	0.0036	2.403 ms

The PF performance is also studied under various uncertainties of the reference map. The uncertainties are modeled by the standard deviation of the positions of the pole-like landmarks stated in the reference map. Table VIII shows that an accurate reference map is crucial to the pose estimation of the ego-car; however, the RT_MCL shows that it can handle uncertainties up to 2.0 meters ($2\sigma_{pole}$) in map-poles poses and still can localize the ego-car with less than 30 cm of error.

TABLE VIII. EFFECT OF THE LANDMARK STANDARD DEVIATION.

$\sigma_{x_{pole}}$	$\sigma_{y_{pole}}$	x-error	y-error	Yaw-error
0.3	0.3	0.1143	0.1154	0.0040
0.5	0.5	0.1730	0.1633	0.0057
1.0	1.0	0.2926	0.2736	0.0098

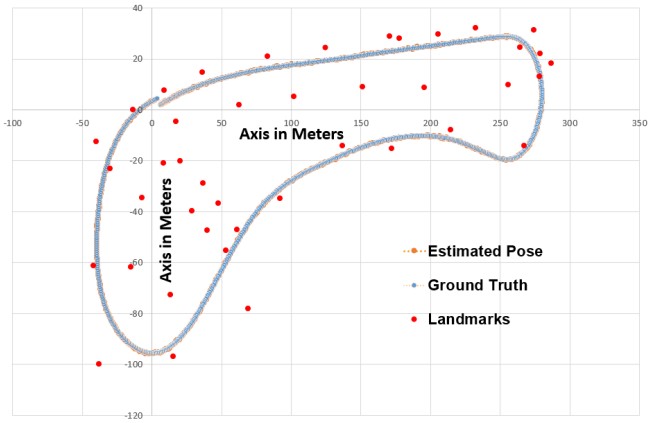

Figure 6. Ego-car localization results in the test track.

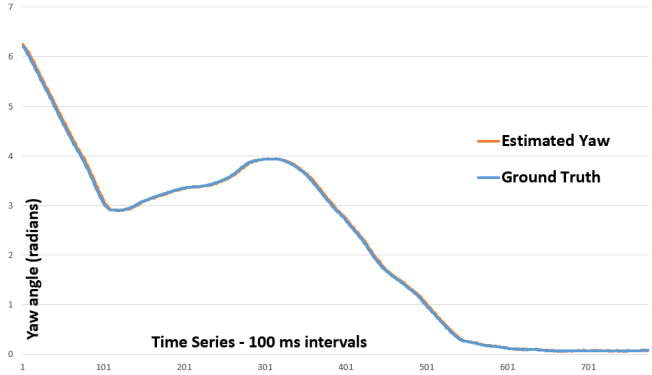

Figure 7. Ego-car orientation estimation in the test track.

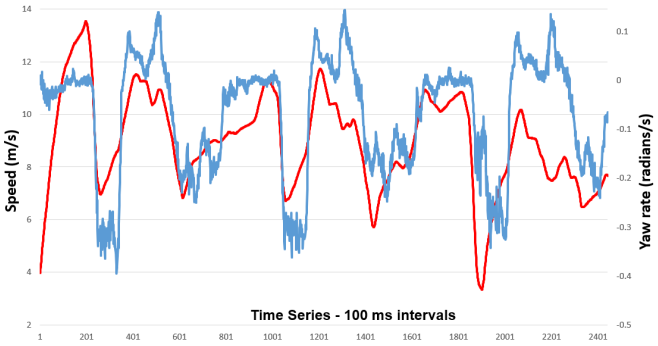

Figure 8. The ego-car speed and yaw rate during driving 3 laps in the test track.

Figure 9 shows the convergence performance of the particle filter during the initialization phase using 50 particles. The error stabilizes after 100 time-steps. Figure 10 as well shows the performance of the particles' weights for one lap travel on the test track. The figure shows that the best weights are significantly higher the average weights which a sign of convergence robustness. Moreover, there is a kind of inverse relationship between the number of detected landmarks and the value of the best and average weight. Eq. (22) can be rewritten in a more streamlined form in Eq. (23). The later equation shows the weight values are the product of the likelihood of the pole-landmark observation represented by a multivariate Gaussian probability density function. The higher the number of observed landmarks the more the chance that some of these landmarks have very small likelihood values that bring the whole product down. After many experiments, it has been found that the reasonable value for the number of detected landmarks at each time-step for the robust running of the RT_MCL lies in the range of 4→12 landmarks.

$$w_t^{[m]} = \prod_{j=1}^{N} \frac{exp\left(-\frac{\left(z_{x_j}^{[t]}-\mu_{x_j}^{[t]}\right)}{2\sigma_{x_{pole}}^2}-\frac{\left(z_{y_j}^{[t]}-\mu_{y_j}^{[t]}\right)}{2\sigma_{y_{pole}}^2}\right)}{2\pi\sigma_{x_{pole}}\sigma_{y_{pole}}} \tag{23}$$

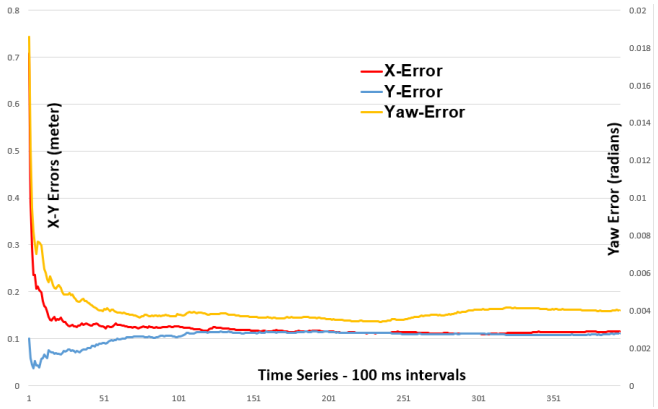

Figure 9. Performance during the particle filter initialization.

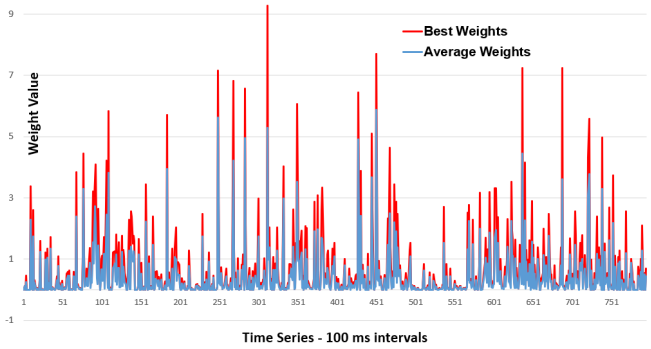

Figure 10. Performance of particle weights during driving one lap.

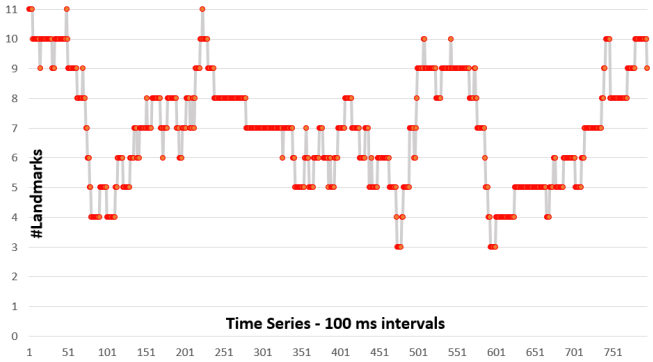

Figure 11. The number of detected pole-like landmarks during one lap.

The many experimentations of the *RT_MCL* pipeline proved to be fast enough in execution to be used in real-time. Using an Intel Core i5 with 1.6 GHz and 8 GB RAM which is a very moderate computational platform, the following measurements (Table IX) are collected for the execution of the RT_MCL for a single ego-car pose estimation based on 12 pole-like landmarks.

TABLE IX. RT_MCL EXECUTION TIME FOR A SINGLE POSE ESTIMATION.

Task	Exec. time
UKF state estimation for 12 landmarks	12×0.439 ms
GB-DBSCAN + RANSAC + ICP Clustering and data association	0.835 ms
PF pose estimation	0.739 ms
Control code overhead – 20%	1.368 ms
Total	8.210 ms

Table IX shows it takes around 8.2 *ms* to execute the whole pipeline for single pose estimation. Most localization functions run at 10Hz to 30Hz speed. Therefore, the proposed RT_MCL satisfies comfortably even the upper end of this requirement.

By considering that the lidar/radar measurements are collected at approximately 30 fps rate (Yurtsever et al. 2020). Then the measurement cycle is 33.3 *ms* which is large enough to be utilized for considering more than 50 landmark detections using UKF according to the data in Table IX.

X. CONCLUSION

In this paper, a real-time Monte Carlo Localization (RT_MCL) method for autonomous cars is proposed, implemented, and described in detail. The method uses a tailored unscented Kalman filter to perform data fusion for the mounted lidar and radar devices on the ego-car. The raw data of the lidar/radar are getting fused using the UKF and then getting clustered using both GB-DBSCAN and RANSAC algorithms to produce the detected pole-like landmarks' poses. These detected landmarks are then associated with the ones in the supplied reference map using the ICP algorithm. Then, a tailored particle filter is designed to produce estimated ego-car poses measured on the global map coordinates.

The RT_MCL method is fully implemented using GCC C++ in addition to advanced math libraries to optimize its real-time performance. The design steps, initialization, and tuning of both the UKF and the PF are described in detail. The initialization and consistency evaluation of both filters has been explained as well. The generic object motion model employed by both UKF and PF is comprehensive and is described using five state variables.

The validation results show that the proposed method is reliably able to detect pole-like landmarks and to estimate the ego-car pose with an 11-cm mean error in real-time using only 50 particles. The measured throughput (execution time) using an affordable CPU proved that the RT_MCL pipeline is very suitable for real-time ADAS or self-driving car localization.

Both UKF and PF has shown that the RT_MCL pipeline can run at 30Hz while able to handle up to 50 landmark detections. The reference map is represented in a probabilistic form by representing each landmark position by its mean centroid and standard deviation. The RT_MCL shows it can handle uncertainties up to 2.0 meters in the landmark centroids and still can localize the ego-car with less than 30-cm of error.

In the future, it is intended to add a front-camera to the presented fusion technique and further investigate the benefits it will add to the overall localization performance. Furthermore, will augment the RT_MCL with other road objects like guardrails, sidewalks, curbs, and intersection features, etc.

REFERENCES

Carlevaris-Bianco, N. Ushani, A.K. and Eustice R.M. 2015. University of Michigan north campus long-term vision and lidar dataset. International Journal of Robotics Research, vol. 35, no. 9, pp. 1023–1035.

Dietmayer, K. Kellner, D. & Klappstein, J. 2012. Grid-based dbscan for clustering extended objects in radar data. IEEE Intelligent Vehicles Symposium, Alcala de Henares, Spain, June.

Eigen 2020. http://eigen.tuxfamily.org/index.php?title=Main_Page, accessed on 11th March.

Einicke, G.A. & White, L.B. 1999. Robust Extended Kalman Filtering. *IEEE Trans. Signal Process.*, 47(9):2596–2599, Sept.

Farag, W. 1998. Synthesis of intelligent hybrid systems for modeling and control. Ph.D. Thesis, *University of Waterloo*, Canada.

Farag, W. 2018. Recognition of traffic signs by convolutional neural nets for self-driving vehicles. *International Journal of Knowledge-based and Intelligent Engineering Systems*, IOS Press. 22(3):205-214.

Farag, W. 2019a. Traffic signs classification by deep learning for advanced driving assistance systems. *Intelligent Decision Technologies*, IOS Press, 13(3): 215-231.

Farag, W. 2019b. Safe-driving cloning by deep learning for autonomous cars. *International Journal of Advanced Mechatronic Systems,* Inderscience Publishers, 7(6):390-397.

Farag, W. 2019c. Cloning Safe Driving Behavior for Self-Driving Cars using Convolutional Neural Networks. *Recent Patents on Computer Science*, Bentham Science Publishers, The Netherlands, 12(2):120-127(8).

Farag, W. 2020a. A Comprehensive Real-Time Road-Lanes Tracking Technique for Autonomous Driving. *International Journal of Computing and Digital Systems (IJCDS)*, 9 (3):349-362.

Farag, W. 2020b. Complex Trajectory Tracking Using PID Control for Autonomous Driving. *International Journal of Intelligent Transportation Systems Research*, Springer, 18:356–366.

Farag, W. 2020c. Real-Time Detection of Road Lane-Lines for Autonomous Driving. *Recent Advances in Computer Science and Communications*, Betham Science, 13(2): 265-274.

Farag, W. 2020d. A Comprehensive Vehicle-Detection-and-Tracking Technique for Autonomous Driving. *International Journal of Computing and Digital Systems (IJCDS)*, 9 (4):567-580.

Farag, W. 2020e. A lightweight vehicle detection and tracking technique for advanced driving assistance systems. *Journal of Intelligent & Fuzzy Systems* 39(3):2693-2710, IOS Press.

Farag, W. 2020f. Complex Track Maneuvering using Real-Time MPC Control for Autonomous Driving. *International Journal of Computing and Digital Systems (IJCDS)*, 9(5):909-920.

Farag, W. 2021a. Real-Time Autonomous Vehicle Localization Based on Particle and Unscented Kalman Filters. *Journal of Control, Automation and Electrical Systems,* Springer, Vol. 32, pp. 309–325.

Farag, W. 2021b. Complex-Track Following in Real-Time Using Model-Based Predictive Control. *International Journal of Intelligent Transportation Systems Research*, Springer, April, Vol. 19, No. 4, pp. 112–127.

Farag, W. Saleh, Z. 2018a. Road Lane-Lines Detection in Real-Time for Advanced Driving Assistance Systems. *Intern. Conf. on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT'18)*, Bahrain, 18-20 Nov.

Farag, W. Saleh, Z. 2018b. Behavior Cloning for Autonomous Driving using Convolutional Neural Networks. *Intern. Conf. on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT'18)*, Bahrain, 18-20 Nov.

Farag, W. Saleh, Z. 2018c. Tuning of PID Track Followers for Autonomous Driving. *Intern. Conf. on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT'18)*, Bahrain, 18-20 Nov.

Farag, W. Saleh, Z. 2019a. An Advanced Road-Lanes Finding Scheme for Self-Driving Cars. *2nd Smart Cities Symposium (SCS'19)*, IET Digital Library, Bahrain, 24-26 March.

Farag, W. Saleh, Z. 2019b. An Advanced Vehicle Detection and Tracking Scheme for Self-Driving Cars. *2nd Smart Cities Symposium (SCS'19),* IET Digital Library, Bahrain, 24-26 March.

Farag, W. Saleh, Z. 2019c. MPC Track Follower for Self-Driving Cars. *2nd Smart Cities Symposium (SCS'19),* IET Digital Library, Bahrain, 24-26 March.

Fischler, M. & Bolles, R. 1981. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Comm. ACM*, 24(6):381–395, June.

GCC C++ 2020. https://gcc.gnu.org/, accessed on 11th March.

Kummerle, J. Sons, M. Poggenhans, F. Kuehner, T. Lauer, M. & Stiller, C. 2019. Accurate and efficient self-localization on roads using basic geometric primitives. 2019 IEEE International Conference on Robotics and Automation, May.

Kuutti, S. Fallah, S. Katsaros, K. Dianati, M. Mccullough, F. & Mouzakitis, A. 2018. A Survey of the State-of-the-Art Localization Techniques and Their Potentials for Autonomous Vehicle Applications. IEEE Internet of Things Journal, vol. 5, no. 2, pp. 829-846, April.

Levinson J. & Thrun S. 2010. Robust vehicle localization in urban environments using probabilistic maps**.** 2010 IEEE International Conference on Robotics and Automation, May, pp. 4372–4378.

Levinson, J. Montemerlo, M. & Thrun, S. 2007. Map-Based Precision Vehicle Localization in Urban Environments. Conference: Robotics: Science and Systems III, June 27-30, , Georgia Institute of Technology, Atlanta, Georgia, USA.

Lu F. & Milios, E. 1997. Robot pose estimation in unknown environments by matching 2d range scans**.** Journal of Intelligent and Robotic Systems, vol. 18, no. 3, pp. 249–275.

Modsching M. Kramer R. & Hagen K. 2006. Field trial on GPS accuracy in a medium-size city: the influence of built-up. 3rd Workshop on Positioning, Navigation and Communication, vol. 2006, pp. 209–218.

Nagiub, M. & Farag, W. 2013. Automatic selection of compiler options using genetic techniques for embedded software design. IEEE 14th Inter. Symposium on Comp. Intelligence and Informatics (CINTI), Budapest, Hungary, Nov. 19.

Piché, R. 2016. Online tests of Kalman filter consistency. *Intern. Journal of Adaptive Control and Signal Processing*. 30(1):115–124.

Sander, J. Xu, X. Ester, M. & Kriegel, H-P. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. Proc. of the 2nd Inter. Conf. on Knowledge Discovery and Data Mining, pp. 226–231, August.

Sefati, M. Daum, M. Sondermann, B. Kreisk, K.D. & Kampker, A. 2017. Improving vehicle localization using semantic and pole-like landmarks. 2017 IEEE Intelligent Vehicles Symposium, June, pp. 13–19.

Schaefer, A. Büscher, D. Vertens, J. Luft, L. & Burgard, W. 2019. Long-Term Urban Vehicle Localization Using Pole Landmarks Extracted from 3-D Lidar Scans. European Conference on Mobile Robots (ECMR), Czech Republic, 4-6 Sept.

Schubert, R. Richter, E. & Wanielik, G. 2008. Comparison and Evaluation of Advanced Motion Models for Vehicle Tracking. 11th Inter. Conf. on Information Fusion, Cologne, Germany, July.

Suhr, J.K. Jang, J. Min, D. & Jung, H.G. 2017. Sensor Fusion-Based Low-Cost Vehicle Localization System for Complex Urban Environments. IEEE Trans. on Intelligent Transportation Systems, Vol. 18(5), May.

Thrun, S. 2002. Particle Filters in Robotics. Proceedings of 18th Annual Conf. on Uncertainty in AI (UAI), Edmonton, Canada.

Ubuntu Linux 2020. https://www.ubuntu.com/, accessed on 11th March.

Wan, E.A. & Van Der Merwe, R. 2000. The unscented Kalman filter for nonlinear estimation. IEEE Adaptive Sys. for Signal Processing, Comm., and Control Symposium, Alberta, Canada, Oct.

Weng, L. Yang, M. Guo, L. Wang, B. & Wang, C. 2018. Pole-based realtime localization for autonomous driving in congested urban scenarios**.** 2018 IEEE International Conference on Real-time Computing and Robotics, August, pp. 96–101.

Woo, A. Fidan, B. & Melek, W.W. 2019. Localization for Autonomous Driving**.** Handbook of Position Location: Theory, Practice, and Advances, 2nd Ed., Wiley**.**

Yurtsever, E. Lambert, Carballo, J. & Takeda, A. K. 2020. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access*, vol. 8, pp. 58443-58469. doi:10.1109/ACCESS.2020.2983149.

Zarchan, P. & Musoff, H. 2013. Fundamentals of Kalman Filtering: A Practical Approach. American Institute of Aeronautics and Astronautics, Incorporated, 4th Ed., ISBN 978-1-62410-276-9.

Zhao, S. & Huang, B. 2017. On Initialization of the Kalman Filter. 6th Inter. Symposium on Adv. Control of Ind. Processes (AdCONIP), Taipei, Taiwan, May 28-31.