CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# A sensor fusion filter structure based on RBFNN aided Kalman filter in target positioning

Master's thesis in Embedded Electronic System Design

MING XU WENQIAN HAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2020

Master's thesis 2020

# A sensor fusion filter structure based on RBFNN aided Kalman filter in target positioning

MING XU WENQIAN HAN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2020

A sensor fusion filter structure based on RBFNN aided Kalman filter in target positioning
MING XU
WENQIAN HAN

© MING XU, 2020.
© WENQIAN HAN, 2020.

Supervisor : Lena Peterson, Department of Computer Science and Engineering
Advisor : Ted Henriksson, China Euro Vehicle Technology AB
Examiner: Per Larsson-Edefors, Department of Computer Science and Engineering

Master's Thesis 2020
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2020

A sensor fusion filter structure based on RBFNN aided Kalman filter in target positioning

MING XU
WENQIAN HAN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

Nowadays multiple sensors are mounted in one vehicle to obtain reliable data useful for environment perception, Kalman-filter-based multisensor data fusion is commonly adopted in vehicles for target positioning to provide active safety features to the end user. Kalman filter is the optimal solution to numerous data prediction problems as long as the noise is Gaussian. However, non-Gaussian noise or un-modeled noise contained in filter signals can seriously degrade the filter performance. Without a priori knowledge of noise in the system, tuning the parameters of Kalman filter can be difficult.

To improve the accuracy of filter estimates, a sensor fusion system that integrates a Kalman filter with a radial basis function neural network (RBFNN) is presented in the thesis. Both extended Kalman filter (EKF) and converted measurement Kalman filter (CMKF) are implemented to verify the univesality of the system. RBFNN is chosen due to its universal function approximation, simple structure and faster learning speed. An incremental constructive method is adopted to design the RBFNN in the project. Filter status (time interval between measurements and estimation results) as well as host information (steering angle and acceleration) are sent as input of RBFNN. The training target is the difference between the conventional filter output and the ground truth from the DAQ system. After training, the neural network is able to compensate for the estimation error of the Kalman filter.

In the project, Normalized Root Mean Square Error (NRMSE), which measures the difference between the filter's estimates and actual ground truth data, is used as the evaluation criteria for fusion filter performance. After applying RBFNN to the fusion filter system, the NRMSE of the CMKF is reduced by more than 30%. and the NRMSE of the EKF is improved by more than 20%. The promising results proves that a fusion filter combining neural network and conventional Kalman filter can achieve a better performance than the stand-alone conventional Kalman filter. The proposed neural network is a universal tool to compensate for the estimate error of different Kalman filter types. Since the camera was not applied in the test because of hardware failure, some features of sensor fusion could not be verified. For further improvement, plenty of training sets in more complex scenario should be collected for network training to make the fusion filter reliable on the real road. The association algorithm can be updated to achieve multiple object tracking.

Keywords: Kalman filter, active safety, EKF, CMKF, sensor fusion, radial basis function neural network, target positioning

# Acknowledgements

The thesis work was supported by CEVT (China Euro Vehicle Technology AB), we would first like to thank our supervisor Ted Henriksson at CEVT for his support in sensor device installation, data collection and final evaluation on the test track. His willingness to give his time so generously has been very much appreciated.

Sarah Wills Carlsson (Module team director - Feature Design), thanks for providing us with the opportunity to conduct the thesis at CEVT, and all your support for project resources.

We would also like to express our gratitude to our supervisor Lena Peterson at Chalmers for her patient guidance on research work and academic writing.

Ming Xu and Wenqian Han, Gothenburg, November, 2020

# Abbreviations

**ADAS** - Advanced Driving Asistance Systems
**BPVS** - Backpropagation Training with Variable Stepsize
**CAN** - Controller Area Network
**CMKF** - Converted Measurement Kalman Filter
**DAQ** - Data Acquisition Systems
**EKF** - Extended Kalman Filter
**ESR** - Electronically Scanning Radar
**FCW** - Forward Collision Warning
**FLOP** - Floating-point Operation
**GRNN** - Generalized Regression Neural Network
**GPS** - Global Positioning System
**IMU** - Inertial Measurement Unit
**LMS** - Least Mean Square
**LIN** - Local Interconnect Network
**LDW** - Lane Departure Warning
**MSE** - Mean Squared Error
**MLP** - Multi-Layer Perceptron
**MCU** - Microcontroller Unit
**NRMSE** - Normalized Root Mean Square Error
**NNDA** - Nearest Neighbor Data Association
**OLS** - Orthogonal Least Squares
**PCW** - Pedestrian Collision Warning
**PDA** - Probabilistic Data Association
**RBFNN** - Radial Basis Function Neural Network
**RMSE** - Root Mean Square Error
**RTK** - Real-time Kinematic
**SLFNs** - Single-hidden-layer Feedforward Networks

# Contents

# List of Figures

# List of Tables

# 1
# Introduction

Nowadays, advanced driving assist systems [1] (ADAS) are in demand by society. Plenty of sensors like lidar, camera, radar are mounted in a vehicle for environment perception. To overcome the respective shortcomings of each sensor type when being used individually, sensor fusion [2], which combines data derived from disparate sources to obtain result of less uncertainty, should be applied to obtain more accurate environmental information.

## 1.1 Background

The Kalman filter [3] is the most widely used algorithm for sensor fusion, as well as an effective algorithm to estimate the state of a linear dynamical system, which performs optimally with Gaussian distribution noise. It works in a two-step process. The Kalman filter can be formulated as [4]:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{w}_k \tag{1.1}$$

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{v}_k \tag{1.2}$$

where (1.1) is the motion model, (1.2) is the measurement model, $\mathbf{A}$ is the state transition matrix, $\mathbf{H}$ is the observation matrix. $\mathbf{x}_k \in R^n$ is the state vector at time $k$, $\mathbf{z}_k \in R^m$ is the measurements of sensors, and $\mathbf{w}_k$, $\mathbf{v}_k$ are the prediction noise component and measurement noise component, Gaussian distribution is assumed for both components, and the coviarance matrices are represented as:

$$E[w_j, w_i^T] = \mathbf{Q}, \tag{1.3}$$

$$E[v_j, v_i^T] = \mathbf{R}, \tag{1.4}$$

However, non-Gaussian noise may affect the performance of filter in reality. Furthermore, tuning Kalman filter to get suitable parameters like measurement noise covariance $\mathbf{R}$ and process noise covariance $\mathbf{Q}$, is necessary for filter implementation. In a car's navigation system, a Kalman filter can be used to smoothen the minimum position-error estimation to get the best signal to noise ratios. Optimal tuning parameters will contribute to better performance of Kalman filter. Until now, tuning

these parameters has been a challenge and not well-defined [5]. Usually, with some prior knowledge of the process noise, multiple experiments will be carried out to decide the filter parameters [4].

To dynamically estimate **Q** and **R** for a better estimation performance, Akhlaghi et al. in [6] proposed an adaptive extended Kalman filter approach, in which an innovation-based method is used to adjust **Q** and a residual-based method is used to adjust **R**.

In addition to focusing on tuning **Q** and **R**, artificial neural network, which is inspired by the biological neural networks of animal brains to learn to perform some task through analyzing training examples, can be applied to improve the performance of the filter. A method proposed by Ning et al. [7] identified that an optimal Radial Basis Function Neural Network(RBFNN) can be used to mitigate the influence of the errors in both the motion model and measurement model.

## 1.2 Related work

In recent years, a large number of scholars have investigated to formulate a neural network based methodology to enhance the performance of Kalman filter. In Korniyenko et al.'s research [4], two neural network architectures, Generalized Regression Neural Network (GRNN) and regular Radial Basis Neural Networks (RBNN) were analyzed and applied to the parameter tuning problem. When their performance measures and computational efficiency were evaluated, RBNN was shown to be superior to GRNN. A radial basis function neural networks (RBFNN) aided Adaptive Extended Kalman Filter was presented by Pesce et al. [8] to compensate for the prediction of EKF in order to deliver a better estimation.

## 1.3 Objective

In this project, a sensor fusion system that combines a Kalman filter and RBFNN is proposed to be employed in a vehicle tracking system with camera and radar, in which a trained RBFNN is used to compensate for the error of the output of the Kalman filter in order to achieve higher accuracy when tracking the target vehicle. The Converted Measurement Kalman Filter(CMKF) and Extended Kalman Filter(EKF) are applied in the fusion system respectively to evaluate the improvement compared to the stand-alone Kalman filter.

The new approach with a neural network is implemented in Matlab and evaluated in the test car. The performance is compared to that of a stand-alone Kalman filter. Improvement should be proved in the comparison test, which means the mean squared error (MSE) of the estimation should diminish comparing to a stand-alone Kalman filter.

## 1.4 Limitations

Multiple object detection is not achieved in the project. Because it requires much more effort to consider the data association [9] since we have to decide which one of these multiple sensor responses is caused by the specific target.

Another limitation is that environmental variation may affect the performance of the sensors. The test should be conducted in the proving ground without harsh conditions or extreme weather such as heavy fog or snow because they can cause bad sensor outputs.

## 1.5 Outline

The thesis is divided into seven chapters. Chapter 2 introduces the basic theory of Kalman filter and its variants CMKF and EKF as well as their applications in the thesis project, a general knowledge of neural network and the RBFNN adopted in our design. Chapter 3 introduces the hardware used in the project including radar, camera, data acquisition system and CANoe. Chapter 4 describes the design and implementation of the fusion filter architecture, the neural network design and parameters determinations as well as the sensor fusion method. Chapter 5 lists the test results and result analysis. Chapter 6 discusses the applicability of the neural network and the shortcomings of the design due to radar instability and incomprehensive training examples. Chapter 7 reviews the overall fusion system and possible future developments, moreover, the hardware usage of the design is analyzed.

# 2

# Theory

In this chapter, the Kalman filter and its two variants applied in the project are introduced. In addition to that, the fundamental knowledge of neural network, network training and RBFNN are described in detail.

## 2.1 Kalman filter

The Kalman filter, as introduced in section 1.1, works in a recursive two-step process, as shown in Figure 2.1.

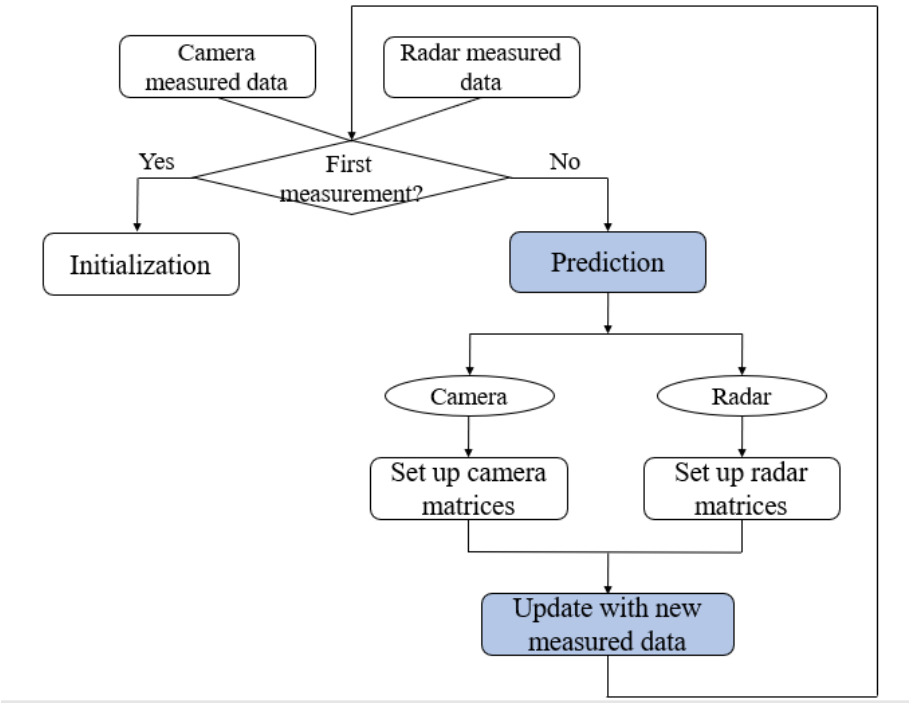



**Figure 2.1:** Kalman filter for sensor fusion: two-step process

The measured data coming from the camera or radar is the input state vector of the Kalman filter. In the predict step, the new state vector is predicted based on the previous value, and also, an uncertainty caused by process noises in the system is predicted. The predicted state vector and predicted covariance is defined as:

$$\mathbf{x}'_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{v}_{k-1} \tag{2.1}$$

$$\mathbf{P}'_k = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^{\mathbf{T}} + \mathbf{Q}_{k-1} \tag{2.2}$$

where $\mathbf{x}'_k$ is the predicted value of iteration $k$, $\mathbf{A}$ is the transition matrix of the motion model, $\mathbf{v}_{k-1}$ is the process noise, and $\mathbf{Q}_{k-1}$ is the process noise covariance of iteration $k-1$.

In the update step, the predicted values are corrected by the actual measurements. The difference $\mathbf{y}_k$ between the predicted value and measured value, known as the innovation in (2.3), as well as the total error $\mathbf{S}_k$ in (2.4) (prediction error plus measurement error), are calculated as the intermediate variables in the following equations. Kalman gain $\mathbf{K}_k$ in (2.5) is also introduced to compute how much correction should be taken from observation and update the state estimate. These are the equations:

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}\mathbf{x}'_k \tag{2.3}$$

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}'_k\mathbf{H}^{\mathbf{T}} + \mathbf{R}_k \tag{2.4}$$

$$\mathbf{K}_k = \mathbf{P}'_k\mathbf{H}^{\mathbf{T}}\mathbf{S}_k^{-1} \tag{2.5}$$

Here $\mathbf{z}_k$ denotes the measured value, $\mathbf{R}_k$ is the measurement noise covariance. Then the updates are made using the Kalman gain:

$$\mathbf{x}_k = \mathbf{x}'_k + \mathbf{K}_k\mathbf{y}_k \tag{2.6}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}'_k \tag{2.7}$$

The updated values are the final predictions of the current iteration and will be fed into the next iteration of predict step [10].

A state vector including 2D position and velocity component in Cartesian coordinates is defined as

$$\mathbf{x} = \begin{pmatrix} p_x \\ p_y \\ v_x \\ v_y \end{pmatrix} \tag{2.8}$$

In this project, a constant-velocity motion model is assumed to describe linear and smooth target motion. It manages to represent the behavior of the host car quite well; therefore, the next position is

$$\begin{pmatrix} p'_x \\ p'_y \end{pmatrix} = \begin{pmatrix} p_x + v_x \Delta t \\ p_y + v_y \Delta t \end{pmatrix} \tag{2.9}$$

where $\Delta t$ is the discrete-time step of the sensor measurement. The corresponding transition matrix $\mathbf{A}$ will be:

$$\mathbf{A} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.10}$$

The camera measurements, $\mathbf{z}$, only contains a position component. When calculating the innovation, $\mathbf{y}$, the measured value, and the predicted value are compared, so we need to design the measurement matrix $\mathbf{H}$ to get rid of the velocity component [11]:

$$\begin{pmatrix} p_x \\ p_y \end{pmatrix} = \mathbf{H} \begin{pmatrix} p'_x \\ p'_y \\ v'_x \\ v'_y \end{pmatrix}, \qquad \text{where } \mathbf{H} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \tag{2.11}$$

In reality, the motion model is usually described in Cartesian coordinates; however, the radar measurements are reported in polar coordinates; therefore, it brings in a non-linear problem when we perform a coordinate system conversion. Since the measurement function, $h$, is no longer linear, of which the output is not Gaussian distribution anymore. Because basic Kalman filter can not solve such a problem, non-linear Kalman filter is investigated and implemented. One option is using an extended Kalman filter (EKF) for the fusion, another is converted measurement Kalman filter (CMKF).

### 2.1.1 Extended Kalman filter

The radar measures radial velocity and range. One way to perform the coordinate conversion is to map the state vector $\mathbf{x}$ into polar coordinates to get the observation function, $h$, which is a non-linear function denoted by the distance $\rho$ from the origin to the target car, and the bearing angle $\phi$ as shown in Figure 2.2.

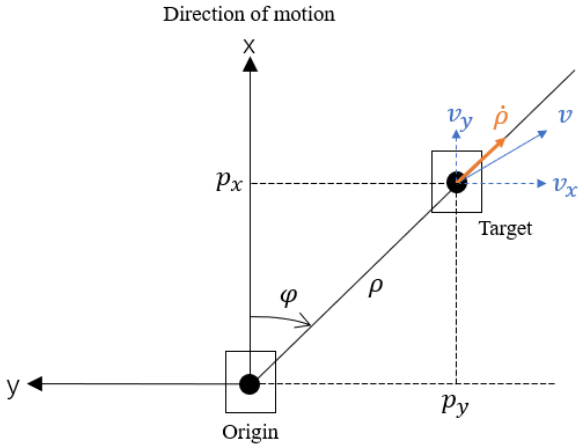



**Figure 2.2:** Target position measurement by radar in polar coordinates

$$h(\mathbf{x}') = \begin{pmatrix} \rho \\ \phi \\ \dot{\rho} \end{pmatrix} = \begin{pmatrix} \sqrt{p_x'^2 + p_y'^2} \\ \arctan(p_y'/p_x') \\ \frac{p_x' v_x' + p_y' v_y'}{\sqrt{p_x'^2 + p_y'^2}} \end{pmatrix} \tag{2.12}$$

EKF is one of the most classic non-linear filtering algorithms. For practical nonlinear problems, EKF usually gives good estimation accuracy. The real-time computational complexity of EKF is also modest. The non-linear system can be indicated as below.

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k) + \mathbf{w}_k \tag{2.13}$$

$$\mathbf{z}_k = h(\mathbf{x}_k) + \mathbf{v}_k \tag{2.14}$$

where $f$, $h$ are the non-linear transition function and observation function.

The basic idea of EKF is to linearize the system by computing a local linear approximation using the first-order Taylor series expansion about an operating point [12] after using the observation function, $h$, to map the state vector in Cartesian coordinates, mathematically expressed as

$$h(\mathbf{x}) = h(\mu) + \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}}(\mathbf{x} - \mu) \tag{2.15}$$

The partial derivation turns into the Jacobian matrix when expanded to **n** dimensions. So the equations of the predict step becomes:

$$\mathbf{x}'_k = f(\mathbf{x}_{k-1}) + \mathbf{v_{k-1}} \tag{2.16}$$

$$\mathbf{P}'_k = \mathbf{A}\mathbf{P}_{k-1}\mathbf{A}^\mathbf{T} + \mathbf{Q}_{k-1}, \qquad \mathbf{A} = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \tag{2.17}$$

And the update steps are:

$$\mathbf{y}_k = \mathbf{z}_k - h(\mathbf{x}'_k) \tag{2.18}$$

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}'_k\mathbf{H}^\mathbf{T} + \mathbf{R}_k, \qquad \mathbf{H} = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} \tag{2.19}$$

$$\mathbf{K}_k = \mathbf{P}'_k\mathbf{H}^\mathbf{T}\mathbf{S}_k^{-1} \tag{2.20}$$

$$\mathbf{x}_k = \mathbf{x}'_k + \mathbf{K}_k\mathbf{y}_k \tag{2.21}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}'_k \tag{2.22}$$

The approximation in the EKF algorithm yields linearization errors which may lead to filtering divergence [13]. CMKF, converts the polar measurements to Cartesian coordinates and applies standard linear Kalman filter.

### 2.1.2 Converted measurement Kalman filter

The measured range and measured bearing angle in polar coordinates shown in Figure 2.2 are defined as

$$\rho_m = \rho + \tilde{\rho} \tag{2.23}$$

$$\phi_m = \phi + \tilde{\phi} \tag{2.24}$$

where $\rho$ and $\phi$ are the true range and bearing angle, $\tilde{\rho}$ and $\tilde{\phi}$ are uncorrelated zero-mean Gaussian noises with deviations $\sigma_\rho^2$ and $\sigma_\phi^2$.

With the conventional conversion,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \rho\cos\phi \\ \rho\sin\phi \end{pmatrix} \tag{2.25}$$

the error between converted measurement and true position in Cartesian coordinates is:

$$\begin{pmatrix}\tilde{x}\\ \tilde{y}\end{pmatrix} = \begin{pmatrix}x_m - x\\ y_m - y\end{pmatrix} = \begin{pmatrix}(\rho+\tilde{\rho})\cos(\phi+\tilde{\phi}) - \rho\cos\phi\\ (\rho+\tilde{\rho})\sin(\phi+\tilde{\phi}) - \rho\sin\phi\end{pmatrix} \tag{2.26}$$

where $\tilde{x}$ and $\tilde{y}$ are white noise with mean $\xi$.

$$\xi = E\begin{pmatrix}\tilde{x}\\ \tilde{y}\end{pmatrix} = \begin{pmatrix}(e^{-\sigma_\phi^2/2} - 1)\rho\cos\phi\\ (e^{-\sigma_\phi^2/2} - 1)\rho\sin\phi\end{pmatrix} \tag{2.27}$$

Then we calculate debiased converted measurement with (2.28), the covariance matrix of which can be derived; thus we can get the linearized measurement model, and the problem can be solved [14].

$$\begin{pmatrix}x_d\\ y_d\end{pmatrix} = \begin{pmatrix}x_m\\ y_m\end{pmatrix} - \xi \tag{2.28}$$

## 2.2 Data Association Algorithm

In multiple object tacking, data association is required to assign measurements to tracks. The method of data association for multi-sensor has been studied in [15]. Several data association algorithms are listed:

- Nearest Neighbor Data Association (NNDA) [15]: The nearest neighbor data association is the simplest method. The nearest neighbor measurement in the target tracking is the measurement closest to the predicted target position, which is the only factor in NNDA to decide if it originated from the target.

- Probabilistic Data Association (PDA) [15]: In the PDA method, the probabilities of association between the measurements and the tracked target are calculated in each step. Rather than selecting the best among the measurements, all the related measurements will be incorporated.

## 2.3 Neural Network

Neural networks are algorithms that try to mimic the human brain. The technique is included in machine learning, in which a computer learns to perform some tasks by analyzing training sets. It was first proposed in 1944 and was widely used in the 80s and early 90s. In recent years, neural networks have become the state-of-the-art technique for many applications. The information transmitting process of neurons in the brain is illustrated in Figure 2.3. Dendrilles of a neuron cell collects inputs from other nodes. After processing the inputs, it transmits outputs through axon terminals, and the outputs of current neurons become the input of others. Similarly, a neural network consists of simulated processing neurons; each of them is considered a node connected to other nodes.

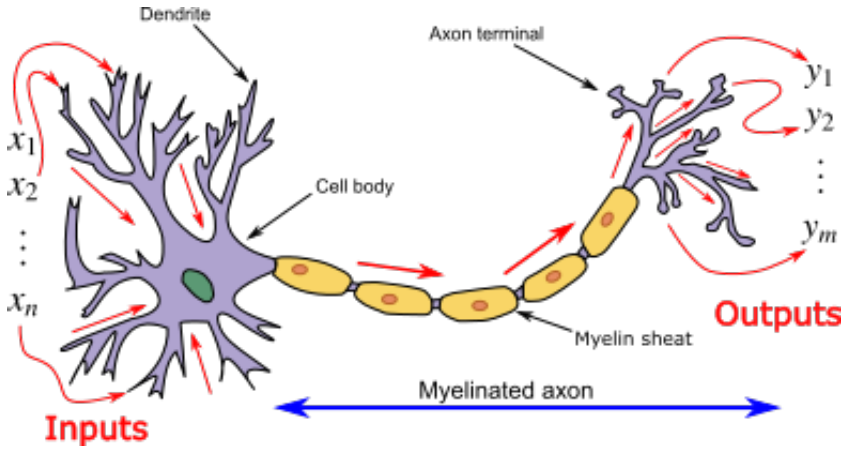



**Figure 2.3:** Transmitting process of neuron in the brain [16]

### 2.3.1 Feedforward Neural Network

Today's neural nets are mostly organized into layers of nodes, as shown in Fig 2.4, the input, output, and weight vector of each layer of the network are denoted accordingly by $\mathbf{x_{NN}}$, $\mathbf{y_{NN}}$ and $\omega_{\mathbf{ij}}$. Each layer's output is simultaneously the input to the subsequent layer. The simplest type of neural networks used in the project is a feedforward neural network. The data moves in one direction through the network, from input to output layers through in between hidden layers. There are no loops or cycles in the network, unlike recurrent networks. External training data is fed to the input layer. After the data passes through hidden layers with some complex computations, the final result is produced by the output layer.

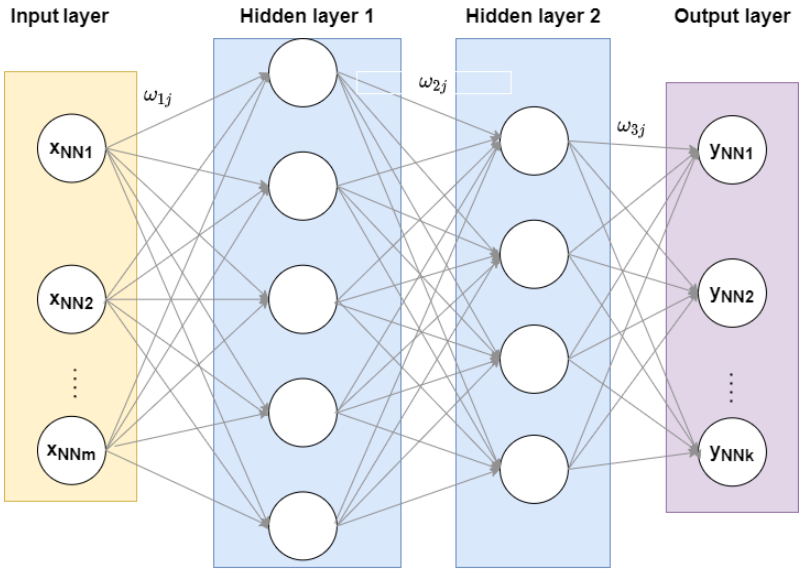



**Figure 2.4:** Layered organization of an example neural network

### 2.3.2 Forward Propagation

Forward propagation refers to the process of calculation of the intermediate variables and outputs of a neural network. With the input data fed in the forward direction, the subsequent hidden layer receives and processes the data then passes the result to the successive layers. For each incoming connection, the $j$-th node of current layer will assign a "weight" $\omega_{\mathbf{i}}$, which represents the relative significance to inputs as illustrated in Figure 2.5.

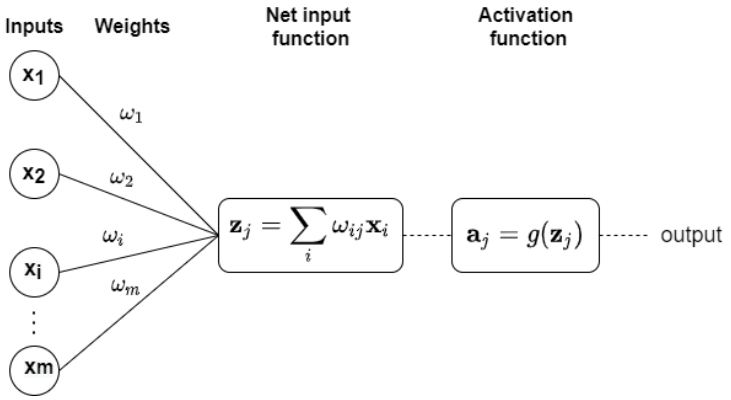



**Figure 2.5:** Example forward propagation process on a neural network node, where $\omega_{\mathbf{1}}, \omega_{\mathbf{2}}, \omega_{\mathbf{3}}$ and $\omega_m$ are the weights assigned to corresponding input

Then all products of each input and corresponding weight are summed, producing the net input denoted by $\mathbf{z}_j$, shown in (2.29). The net input will be passed through the node's activation function $g(\mathbf{z}_j)$, which determines whether the signal should be considered and delivered to the following nodes through the network and to what extent. The activation process is expressed as (2.30), where $\mathbf{a}_j$ is the activation of the current node and input of the following nodes. If the signal passes through, the

node has been activated [17].

$$\mathbf{z}_j = \sum_i \omega_{ij} \mathbf{x}_i \tag{2.29}$$

$$\mathbf{a}_j = g(\mathbf{z}_j) \tag{2.30}$$

### 2.3.3 Network Training

Learning is the process where the network adapts to perform the task better by observing the training samples. A set of training examples $(\mathbf{x}_\mu, \mathbf{y}_\mu)$ with $\mathbf{x}_\mu \in \mathbb{R}^m$ and $\mathbf{y}_\mu \in \mathbb{R}^d$ are required, where $\mathbf{x}_\mu$ is a training input vector of the network and $\mathbf{y}_\mu$ is a corresponding actual output vector, supposing there are $n$ examples. Training can be considered as a process or an algorithm to adjust, and decide the parameters of the network so that its output is approximately the same as the actual output $\mathbf{y}_\mu$. An example cost function in (2.31) is defined to represent the errors between the network output and actual output corresponding to the training input from training sets.

$$E(\mathbf{W}) = \frac{1}{2n} \sum_{\mu=1}^{n} \|(\mathbf{y}_\mu - \mathbf{y}_{\mathbf{NN}})\|^2 \tag{2.31}$$

Here, the notation $\|\mathbf{v}\|$ means the length function for vector $\mathbf{v}$, so the quadratic cost function $E$ is a non-negative value which becomes small or minimal when $\mathbf{y}_{\mathbf{NN}}$ is approximately equal to the actual output for all training input. In order to minimize the observed error, the weights are continually adjusted until the learning is complete when applying additional observations does not reduce the error rate during training.

#### Gradient Descent

Gradient descent is an iterative optimization algorithm for finding the parameters of the network that minimizes the cost function. Before the network training starts, all of its parameters are randomly set, which can be considered a randomly chosen starting point of the cost function. The gradient descent algorithm tries to find the local minimum value of the cost function by changing the parameters of the network iteratively so that the error decreases towards the opposite direction of current gradient. For each iteration of gradient descent, the corresponding error $E$ moves towards the direction in which the cost function decreases most rapidly by a small distance $\eta$, the update rule for the parameters of the network in terms of matrices is shown in (2.32).

$$\mathbf{W}^{\tau+1} = \mathbf{W}^{\tau} - \eta \nabla E(\mathbf{W}^{\tau}) \tag{2.32}$$

where $\tau$ is the current number of iteration, $\eta$ is the learning rate, and $\nabla E$ is the gradient. Adjusting the learning rate $\eta$ ensures that the gradient descent algorithm converges in a reasonable time. If $\eta$ is too small, it will be slow for gradient descent to converge. If $\eta$ is too large, gradient descent can overshoot the minimum of the error surface, which may cause it to fail to converge or even diverge.

By applying the update rule to make a move over and over again, the error $E$ keeps decreasing until it reaches the minimum. In practice, gradient descent often works well to find the global minimum of the cost function [18].

### Backpropagation training

Consider a network with differentiable activation functions, the activations of the output nodes are also differentiable functions of the inputs and weights. The values of the weights can be decided by evaluating the derivatives of the error with respect to the weights when minimizing the cost function. Backpropagation is a supervised training algorithm to tackle this problem. It computes the derivatives by propagating the errors from the output layer backwards to hidden layers. The derivatives are then applied to the adjust the weights. It works in two distinct stages [19]:

1. Derivative evaluation
   For each training input $\mathbf{x}_{\mu}$, forward propagation is applied through the network to calculate the activations of hidden layers and output nodes. By applying the chain rule to the partial derivatives, we have:

$$\frac{\partial E}{\partial \omega_{\mathbf{j}}} = \frac{\partial E}{\partial \mathbf{a_j}} \frac{\partial \mathbf{a_j}}{\partial \omega_{\mathbf{j}}} \tag{2.33}$$

   and $\delta$ is introduced to denote the partial derivative of the error of the cost function to the activation of current layer:

$$\delta = \frac{\partial E}{\partial \mathbf{a_j}} \tag{2.34}$$

   First $\delta$ for the output layer is computed:

$$\delta = g'(\mathbf{z}) \frac{\partial E}{\partial \mathbf{y_{NNj}}} \tag{2.35}$$

By propagating $\delta$ backwards from nodes higher up in the network, the value of $\delta_{\mathbf{j}}$ for hidden layer can be obtained as

$$\delta = g'(\mathbf{z}) \sum_i \omega_{ij} \delta_{\mathbf{k}} \tag{2.36}$$

Then after recursively applying the above equation from the output layers, the derivatives of the cost function is given by:

$$\frac{\partial E}{\partial \omega_{\mathbf{j}}} = \mathbf{a}_{j-1} \delta_{\mathbf{j}} \tag{2.37}$$

2. The derivative derived in the last stage will be used to adjust the weights. It can be applied to the gradient descent algorithm or cost functions other than simple sum-of-squares. For weight adjustment using derivatives, there are also plenty of optimization schemes other than gradient descent [19].

### 2.3.4 RBFNN

The radial basis function network is a neural network that uses radial basis functions as activation functions. Radial basis function networks can be used in function approximation, classification, time series prediction, and system control. It is distinguished from the classic Multi-Layer Perceptron (MLP) due to its simple structure, universal function approximation, and faster learning speed.

#### RBFNN Structure

The block diagram of an RBF neural network with one hidden layer is illustrated in Figure 2.6. The neural network consists of three layers, the input layer with a set of inputs and the output layer with a set of outputs. A hidden layer is located between the input and output layer, in which each neuron is implemented with a non-linear radial basis function. Each neuron in the hidden layer has its own center. It computes the distance between each input $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_m})$ and its center and outputs some nonlinear function of that distance. Consequently, each node in the hidden layer produces an output depending on a radially symmetric activation function. When the inputs are near the center of the node, the largest output is usually obtained.

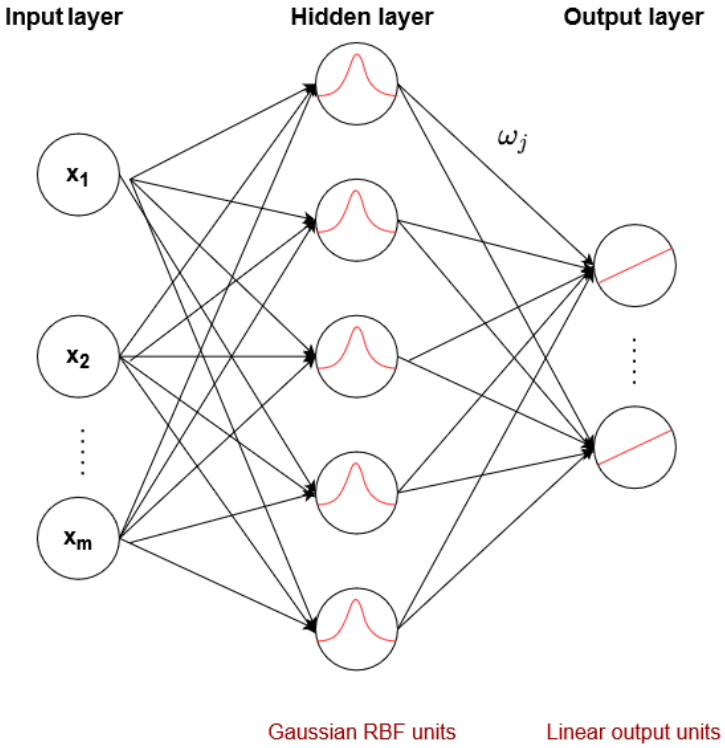



**Figure 2.6:** RBF network block diagram

Regarding each neuron in the hidden layer, normally a Gaussian function is selected among the radial functions as the activation function [7], the activation of the $i$th node is provided as:

$$\phi_i(\mathbf{x}) = e^{-\frac{1}{2\sigma_i{}^2}(\mathbf{x}-\mathbf{c}_i)^2} \tag{2.38}$$

where $i$ is the index of the neurons in the hidden layer, the width parameter $\sigma_i$ gives an impact on the shape of the Gaussian function, and it will be user-defined, $c_i$ is the center vector and $\mathbf{x} - \mathbf{c}_i$ is usually taken to be Euclidean distance.
Then the output of the neural network can be represented as a linear combination of all basis functions:

$$\mathbf{y}_{\mathbf{NN}_j} = \sum_{i=1}^{h} \omega_{ij}\phi_i(\mathbf{x}) \qquad j = 1, 2, ..., n \tag{2.39}$$

where $\mathbf{y}_{\mathbf{NN}_j}$ is the output of $j$ th node in the output layer, $h$ is the number of hidden neurons and $\omega_{ij}$ is the weight.

### RBFNN Training

To design an RBFNN, a list of parameters represented in table 2.1 should be determined

**Table 2.1:** RBFNN Parameters

| Symbol | Description |
|---|---|
| f | basis function type |
| $n$ | number of basis function(hidden neurons) |
| $c$ | basis function centers |
| $\sigma$ | basis function width |
| $w$ | hidden neuron weights |

Normally, the function type $f$ and hidden neurons $n$ are set at the beginning, and the other parameters should be obtained from learning.

The learning algorithm of RBFNN is discussed in [20], where the training methods are categorized into three schemes including one-phase, two-phase and three-phase learning scheme.

1. **One-phase learning**

   In this method, the basic function centers $c_j$ are sub-sampled from the input data, and the width $\sigma_j$ of all hidden neurons are set to the same value in advance, which makes the weights between hidden layers and outputs to be the only parameter that should be trained.

   The pseudo inverse solution [20] is widely applied to compute the weights. The output equation of RBFNN can be represented by matrix.

$$\mathbf{Y_{NN}} = \mathbf{\Phi W} \tag{2.40}$$

   where $\mathbf{\Phi} = (\phi_i(\mathbf{x}))$ and $\mathbf{Y_{NN}} = (\mathbf{y_{NN}}_j)$. Accordingly, given the matrices $\mathbf{\Phi}$ and $\mathbf{Y_{NN}}$, by minimizing the error function as (2.41), the weight matrix $\mathbf{W}$ of the output layer can be found by (2.42) or gradient descent optimization:

$$E(\mathbf{W}) = \|\mathbf{\Phi W} - \mathbf{Y_{NN}}\|^2 \tag{2.41}$$

$$\mathbf{W} = \mathbf{\Phi}^{+}\mathbf{Y_{NN}} \tag{2.42}$$

   where the pseudo inverse matrix $\mathbf{\Phi}^{+}$ can be computed by $(\mathbf{\Phi^T \Phi})^{-1}\mathbf{\Phi^T}$.

**2. Two-phase learning**

This method consists of two steps since two layers of the network are trained respectively.

Firstly, basic function centers $c_j$ and width $\sigma_j$ are obtained. There are many algorithms available to compute both parameters, two of them are introduced here.

Secondly, after determining $c_j$ and $\sigma_j$, the weights of the output can be calculated with the Least Mean Square (LMS) [21] algorithm.

- **RBF centers determination**
  Unsupervised training, when the training sets are given with little or no idea what the results should look like, requires the network to derive structure from data. Clustering and vector quantization algorithms are normally used to get a representative set of centers among a larger data set to minimize quantization error.

  A popular clustering method which minimizes the clustering error is the k-means clustering [22]. The aim is to partition the m-dimension observations $\mathbf{x}$ into $K$ clusters ($K$ neurons in the hidden layer) so that each observation belongs to the cluster with nearest center vector $\mathbf{c_j}$. The partition of the input space is known as Voronoi diagram [23] in which center vector $\mathbf{c_j}$ serves as a prototype vector of the cluster, each cluster $S_j$ is defined by

$$S_j = \{\mathbf{x} \in \mathbb{R}^m \mid \|\mathbf{x} - \mathbf{c_j}\| = \min_{j=1,...,K} \|\mathbf{x} - \mathbf{c_j}\|\} \tag{2.43}$$

  K-means algorithm starts with a set of randomly selected center vectors as initial seed prototypes, then it performs an iterative process: each time the input space is partitioned according to current cluster vectors, the quantization error specified in (2.44) becomes smaller as the prototype $\mathbf{c_j}$ of each cluster approaches the corresponding center of gravity of training sets.

$$E(\mathbf{c_1}, ..., \mathbf{c_K}) = \sum_{j=1}^{K} \sum_{\mathbf{x}^\mu \in S_j} \|\mathbf{x}^\mu - \mathbf{c_j}\|^2 \tag{2.44}$$

  The prototype $\mathbf{c_j}$ is optimized according to the learning rule . The learning rule in (2.45) can be described as updating each center of the cluster

with the average of all observations in the corresponding cluster.

$$\mathbf{c_j} = \frac{1}{S_j} \sum_{\mathbf{x}^\mu \in S_j} \mathbf{x}^\mu \tag{2.45}$$

The process stops when the centers have stabilized, that is when the value of the centers are equal in two consecutive learning process because the clustering has been successful.

- **Widths Calculation**

  The first alternative to decide the widths is to set the widths to a constant for all radial basis functions, discussed in [24]. The widths are fixed as:

$$\sigma = \frac{d_{max}}{\sqrt{2K}} \tag{2.46}$$

  where $d_{max}$ is the maximum distance between any pair of centers. If the data are uniformly distributed in the input space, the centers distribution will also be uniform, this method could be near optimal.

  Another choice is to assign a specific width to each Gaussian kernel. When the centers are widely separated from each other, a larger width should be assigned, on the contrary, when the centers are closer, a smaller width should be assigned. According to Voronoi partition, the inputs can be divided into clusters associated to each center, by computing the standard deviation $\sigma_j^c$ of the distance between inputs in a cluster and the corresponding center, the width of each Gaussian kernel is estimated independently. This method takes the distribution variation of the data into account thus provides better adaptability to the data.

  A widths-computation algorithm based on an exhaustive search was studied and presented in [25]. It guarantees a natural overlap between Gaussian kernels and exploits the network's generalization ability. After computing $\sigma_j^c$, a width scaling factor $s$ is introduced so the widths are defined as:

$$\sigma_j = s\sigma_j^c \tag{2.47}$$

The width scaling factor $s$ can affect the smoothness of the approximation function. The choice of the optimal scaling factor depends on the function to approximate, the dimension of the input set and the data distribution [26].

For every $s_l$ in a width factor set $S$, the MSE(Mean Square Error) is evaluated. The optimal width factor $s_{opt}$ corresponds to the smallest MSE.

- **Weights Determination**

  The output weights of RBF network $\omega_{\mathbf{j}}$ are updated by the Least Mean Square (LMS) algorithm as

$$\omega_{\mathbf{j}}{}^{\tau+1} = \omega_{\mathbf{j}}{}^{\tau} + \eta(\mathbf{y}^{\mu} - \hat{\mathbf{y}})\frac{1}{S_j}\sum_{\mathbf{x}^{\mu}\in S_j}\mathbf{x}^{\mu}, \text{ for } j = 1, 2, ..., K \tag{2.48}$$

  where $\eta$ is the learning rate, $\hat{\mathbf{y}}$ is the network's output at the current time step.

**3. Three-phase learning**

After two-phase learning, a third training phase is proposed in [20], where a backpropagation method is applied to adapt these parameters (including basis function centers $\mathbf{c}$ , function width $\sigma$ and output weights $\mathbf{w}$) simultaneously.

The error function of the network's outputs is defined as a differentiable sum-of-squares function in (2.49):

$$E = \frac{1}{2}\sum_{\mu=1}^{M}\sum_{\mathbf{x}^{\mu}}(\mathbf{y}^{\mu} - \mathbf{y}_{\mathbf{NN}})^2 \tag{2.49}$$

For the radial basis function network with differentiable activation functions, the minimal error is achieved when the derivatives of the parameters' center vector $\mathbf{c_j}$, width $\sigma_{\mathbf{j}}$ and output weights $\omega_{\mathbf{j}}$ of the network vanish. Gradient descent is widely used to solve this problem. Given a parameter set, denoted by $\mathbf{U} = (\mathbf{c_j}, \sigma_{\mathbf{j}}, \omega_{\mathbf{j}})$, for each iteration of gradient descent, the corresponding error $E$ is moving towards the direction in which the error function decreases most rapidly by a small distance $\eta$ as in (2.50).

$$\mathbf{U}^{(\tau+1)} = \mathbf{U}^{(\tau)} - \eta\nabla E(\mathbf{U}^{(\tau)}) \tag{2.50}$$

Consequently, the learning rules for three RBF network parameters' center vector $\mathbf{c_j}$, width $\sigma_\mathbf{j}$ and output weights $\omega_\mathbf{j}$ can be derived.

The backpropagation training with variable stepsize (BPVS) presented by Magoulas et al. in [27] provides a computationally efficient and robust method for three-phase learning. The stepsize tuning procedure is proposed by the following theorem.

**Theorem 2.3.1** *If $\eta_0$ is an arbitrarily assigned positive number, consider the sequence $\eta_m = \eta_0/2^{m-1}$, m = 1, 2, ... . Then the sequence of weight vectors $(\mathbf{U}^{(\tau)})_{\tau=0}^{\infty}$ defined by*

$$\mathbf{U}^{(\tau+1)} = \mathbf{U}^{(\tau)} - \eta_{m_\tau} \nabla E(\mathbf{U}^{(\tau)}),\ \tau = 0, 1, 2, ... \tag{2.51}$$

*where $m_\tau$ is the smallest positive integer for which:*

$$E(\mathbf{U}^{(\tau)} - \eta_{m_\tau} \nabla E(\mathbf{U}^{(\tau)})) - E(\mathbf{U}^{(\tau)}) \leq -\frac{1}{2} \eta_{m_\tau} \|\nabla E(\mathbf{U}^{(\tau)})\|^2 \tag{2.52}$$

*converges to $\mathbf{U}^*$ which minimizes the error function E locally.*

The stepsize adaptation procedure is used to ensure that the subsequent parameter set updates do not overshoot the minimum of the error surface. In case that the stepsize is too small, a specific stepsize lower bound can be selected by the desired accuracy in acquiring the convergence parameter set $\mathbf{U}^*$. If the stepsize is smaller than the lower bound, BPVS will increase the stepsize by doubling it.

# 3

# Hardware

The hardware components used in the project are introduced in this chapter, including radar and camera as sensors on vehicle, data acquisition system for accurate measurement of the target as well as network interface hardware VN8911 together with CANoe for network communication analysis.

## 3.1 Radar

Delphi's multimode ESR2.5 (Electronically Scanning Radar) in Figure 3.1 is used for object detection and measurements of range and range-rate in the project. It combines wide coverage at mid-range and high-resolution long-range coverage to supply two measurement modes simultaneously with a single radar and updates the measurement every 50 ms. Figure 3.2 illustrates the scanning coverage of ESR2.5. Wide mid-range covers the range of 60 meters with more than 90 degrees in azimuth field of view. The coverage detects vehicles cutting in from adjacent lanes and identifies vehicles and pedestrians across the equipped vehicle. Long-range covers the range of 175 meters with more than 20 degrees in azimuth field of view. Long-range coverage measures the accurate range and speed data with powerful object discrimination, which can identify up to 64 targets in the vehicle's path [28].

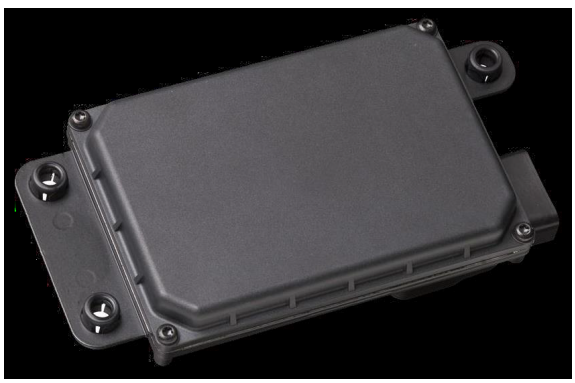


**Figure 3.1:** Delphi ESR2.5 External View [28]

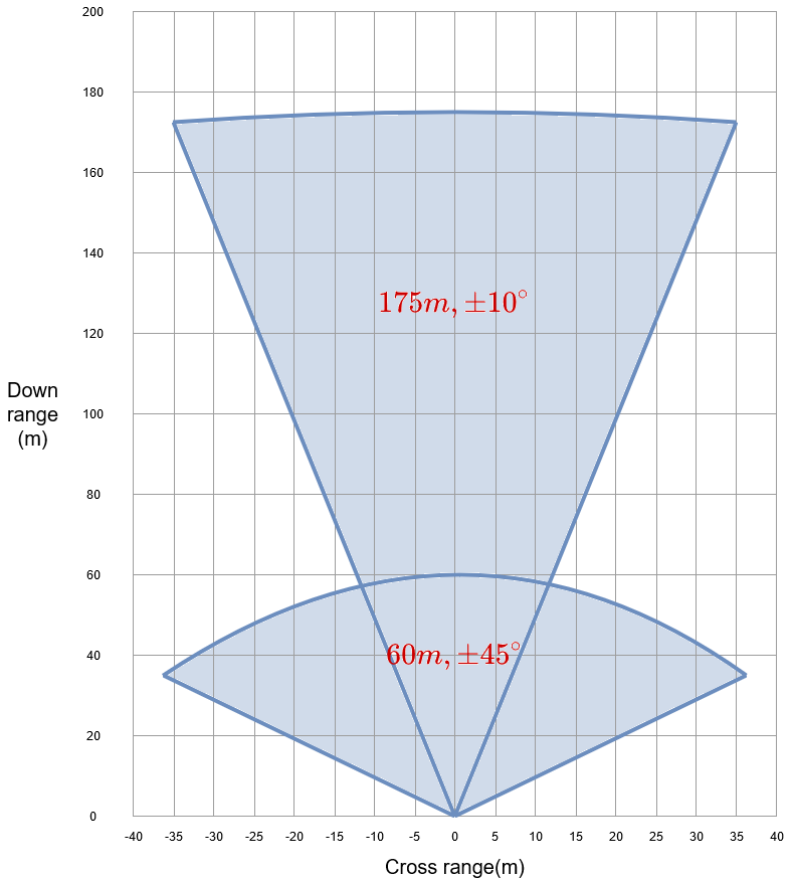



**Figure 3.2:** ESR2.5 scanning coverage in range and azimuth field of view

The ESR connector is illustrated in Figure 3.3 and important ESR pins are defined in table 3.1. Pin 7 and 8 are connected to vehicle CAN bus to get the vehicle velocity, yaw rate and steering wheel angle. Target information detected by radar such as range and relative velocity are transmitted through pin 9 and 18.

**Table 3.1:** Important pin definition of ESR2.5

| Pin number | Pin definition |
|---|---|
| 1 | Battery |
| 3 | CAN GND |
| 7 | Vehicle CAN low |
| 8 | Vehicle CAN high |
| 9 | Private CAN low |
| 10 | Ignition |
| 18 | Private CAN high |

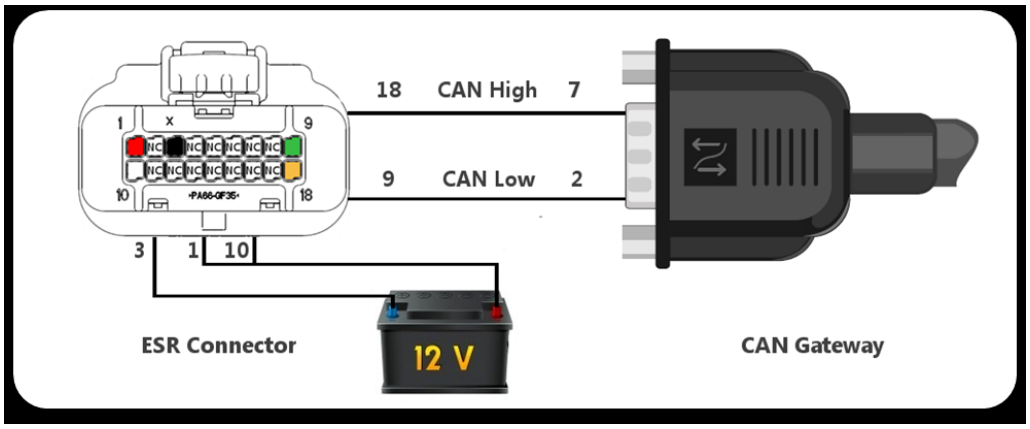



**Figure 3.3:** Pin connection of ESR interface [28]

**Figure 3.4:** Radar module: ESR2.5 mounted on the test vehicle

With the mid-range and long-range coverage radar, the ESR is able to track up to 64 targets in total. For an object entering the detection zone, the track will be confirmed within 150 ms. After the object leaves the detection zone, the track will be dropped within 250 ms. The ESR keeps track of the range, range rate, angle, measurement status and other important information of all 64 tracks of targets. For each track, moving flags are set by the ESR when corresponding targets are moving. In the test session, the radar module ESR2.5 was mounted in the front of the test vehicle, shown in Figure 3.4.

The measurement data by ESR's system enables high functionality, including adaptive cruise control, forward collision warning, brake support, and headway alert [28].

## 3.2 Camera

Mobileye ME630 collision prevention system consists of a camera/buzzer unit as well as a display and control unit, shown in Figure 3.5.

Camera/buzzer unit front view

Camera/buzzer unit back view


Display and control unit

**Figure 3.5:** ME630 system components [29]

The camera/buzzer unit identifies lane markings, vehicles, and pedestrians that are ahead of the equipped vehicle, processes the video data for potentially dangerous situations, and instructs the system display unit to display prospective alert [29]. The display and control unit provides visual alerts, system status indications and contains control buttons for system control and configuration.

The collision prevention system is enabled with the following functions:

- Forward Collision Warning (FCW) alerts you to the danger of an impending collision with the vehicle ahead.
- Pedestrian Collision Warning (PCW) alerts you to the danger of an impending collision with a Pedestrian ahead.
- Lane Departure Warning (LDW) to alert you when you are about to unintentionally swerve outside of the lane you are driving in.
- Headway Monitoring and Warning monitors the driving distance from the vehicle in front of you (headway) and alerts you when the headway is less than a pre-defined threshold measured in seconds [29].

## 3.3 Data Acquisition (DAQ) Systems

Data Acquisition (DAQ) Systems are required to obtain accurate measurements of the target. In the project, the DAQ system from Dewesoft company is selected, which consists of four elements.

- Sensors device

  Based on high accuracy 100 Hz GPS receivers and Inertial Navigation Systems (IMU - inertial measurement units), The Real-time Kinematics (RTK) [30] can provide the most precise position-based measurement with positioning accuracy down to 2 cm.


**Figure 3.6:** GPS receiver & IMU [31]

- Signal conditioners
  Signals from the sensors are processed by the signal conditioners and are subsequently sent to the A/D subsystem.


**Figure 3.7:** Sirius: signal conditioner [32]

- Analog-to-Digital Converter
  A/D converters within a data acquisition system can convert conditioned analog signals into a stream of digital data; finally the data can be processed for display, storage, and analysis.

- Computer with DAQ software
  The DAQ software will handle signal logging and analysis.

  The setup of the system includes GPS antennas, WiFi antenna for communication between cars and antenna for synchronization, shown in Figure 3.8.

**Figure 3.8:** IMU setup for testing

The system logs the longitude and latitude distance between host and target cars, as illustrated in Figure 3.9.

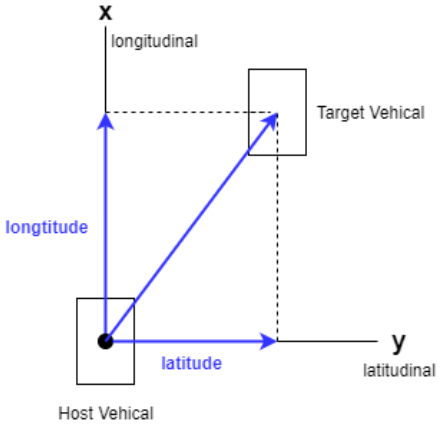



**Figure 3.9:** Longitude and latitude illustration

## 3.4 CANoe

The CANoe [33] is used for the analysis of network communication. The real-time network interface hardware VN8911 [34] by Vector in Figure 3.10, 3.11 enables high-performance applications in combination with CANoe. Together they can be applied in system simulations or bypassing applications with Simulink, remaining bus simulations, gateway implementations, test executions, or data monitoring.


**Figure 3.10:** Network interface hardware VN8911 frontside [35]


**Figure 3.11:** Network interface hardware VN8911 backside [36]

VN8911 provides up to 8 channels for CAN/LIN/J1708/K-Line bus access by a Plug-in module with individual bus transceivers. In the project, ESR bus is connected to ch2 and camera Mobileye bus is connected to ch3.

# 4

# Design

In this chapter, the architecture of the fusion filter combining a Kalman filter and RBFNN is introduced. In addition, an RBFNN is designed for function approximation to compensate for the estimate error of the Kalman filter. Furthermore, the sequential sensor fusion method is introduced to produce the estimate with measurements data from both camera and radar.

## 4.1 Fusion filter architecture

In the actual tracking process, with the mismodeling or environmental perturbations, neither the dynamic noise nor the measurement noise can be white. The performance of Kalman filter can be easily affected by the colored noise.

The Kalman filter works in two-step process as mentioned in Section 2.1 . In the predict step, based on the previous states, the Kalman filter uses the motion model to produce the estimate of current states. In the update step, the result will be updated according to both estimates in first step and present measurements using weighted average.

The dynamic model adopted in our project is a constant velocity model, which assumes that the velocity during a sampling interval is constant. This model has been widely used in many applications due to its simplicity and effectiveness, which can be represented as:

$$\mathbf{x}'_k = \mathbf{A}\mathbf{x}_{k-1} + \mathbf{w}_{k-1} + \Delta\mathbf{x}^{md}_{k-1} \tag{4.1}$$

where the $\Delta\mathbf{x}^{md}_{k-1}$ is added to the original motion model, as the mismodeling and environmental perturbations are taken into account.

The formula shows that the mismodelling noise will also affect the estimation accuracy in addition to the system dynamic noise. These noises also exist in the measurement signal. Therefore, a more accurate output of the fusion filter can be derived from [37]:

$$\mathbf{x}_k^* = \mathbf{x}_k + \Delta\mathbf{x}_k^{mod} + \Delta\mathbf{x}_k^{noise} \tag{4.2}$$

where $\Delta\mathbf{x}_k^{mod}$ is a result of mismodelling and $\Delta\mathbf{x}_k^{noise}$ represents the effect of some uncertain noises. The $\Delta\mathbf{x}_k^{mod}$ and $\Delta\mathbf{x}_k^{noise}$ should be compensated to improve the estimation accuracy. Then a neural network is proposed to compensate for these errors.

According to the prediction model in the Kalman filter (4.1), the mis-modeling error leads to the estimation error of relative position in $\mathbf{x}_k$. The action of steering and acceleration are ignored in the motion model to decrease the computational complexity. However, as an input, it also affects the status of motion. Furthermore, the Kalman gain $\mathbf{K}$, which is calculated based on measurement noise covariance $\mathbf{R}$ and motion noise covariance $\mathbf{Q}$, also has a direct impact on filtering accuracy. The aim of the neural network is to minimize the error of the fusion filter by making the output of the neural network approach the error of the Kalman filter. The error can be represented as

$$\mathbf{x}_k^{err} = \Delta\mathbf{x}_k^{mod} + \Delta\mathbf{x}_k^{noise}$$

The training structure of the neural network is shown in Figure 4.1:

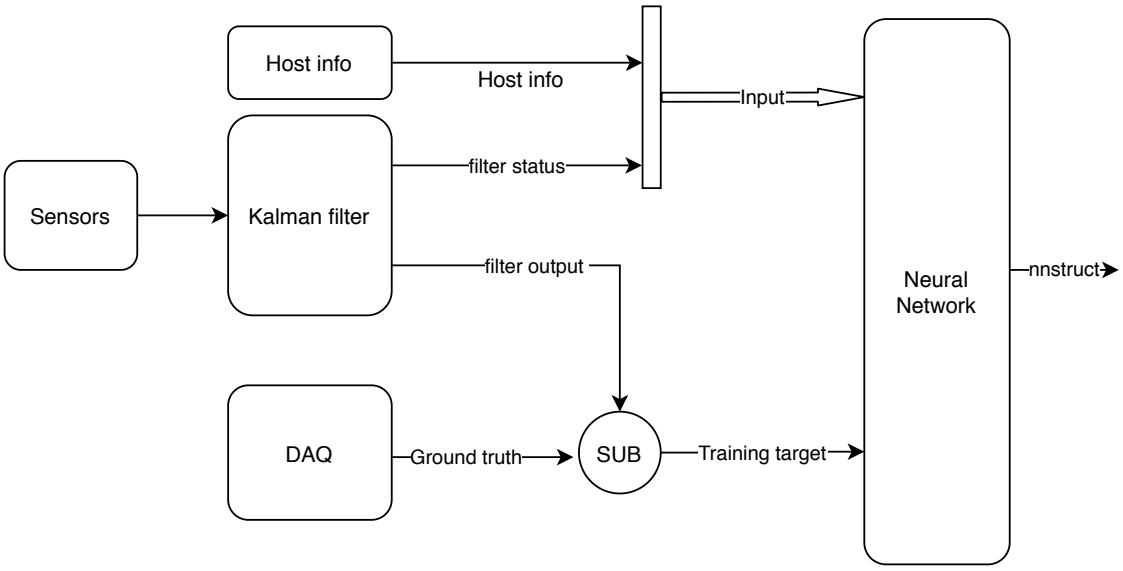



**Figure 4.1:** Neural network training structure

where host information (steering angle, acceleration) and filter status (time interval between two measurements and estimation results) are sent to RBFNN as input, while the training target is the difference between the conventional filter output

and the ground truth from the DAQ system. With these training sets, the parameters of the RBFNN can be obtained and stored in a data structure, namely *nnstruct*.

After the training phase, the RBFNN is deployed to compensate for the conventional Kalman filter's estimation error. The architecture of the new fusion filter is shown in Figure 4.2,

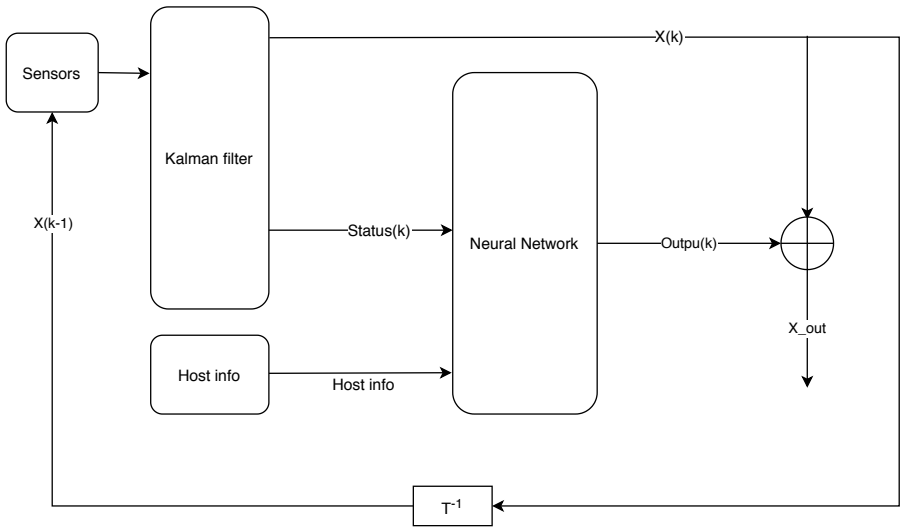



**Figure 4.2:** Fusion filter architecture

where the sensors including camera and radar send the measurements data to the Kalman filter. With the input, including host information and fusion status, the neural network outputs the filter estimation error, which is used to compensate for the output of the Kalman filter. At last, the filter delivers the compensated result $\mathbf{x}_k^{out}$. According to previous equations, the result can be represented as:

$$\mathbf{x}_k^{out} = \mathbf{x}_k + \mathbf{x}_k^{err} \tag{4.3}$$

The model is created with Simulink, which is shown in Figure 4.3.

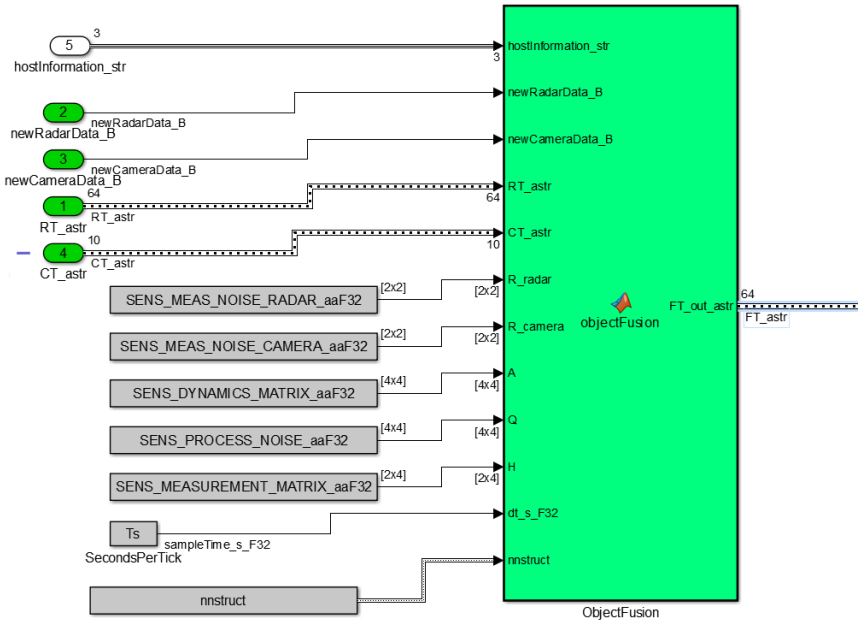



**Figure 4.3:** Simulink: Fusion model

## 4.2 Neural network design

With the capability of universal approximation, the radial basis function neural network (RBFNN) has become a popular network architecture recently. It has the advantage of a simple structure and fast training when applied to function approximation [38]. The neural network designed for the project is shown in figure 4.4, which shows the structure of the RBFNN [39].

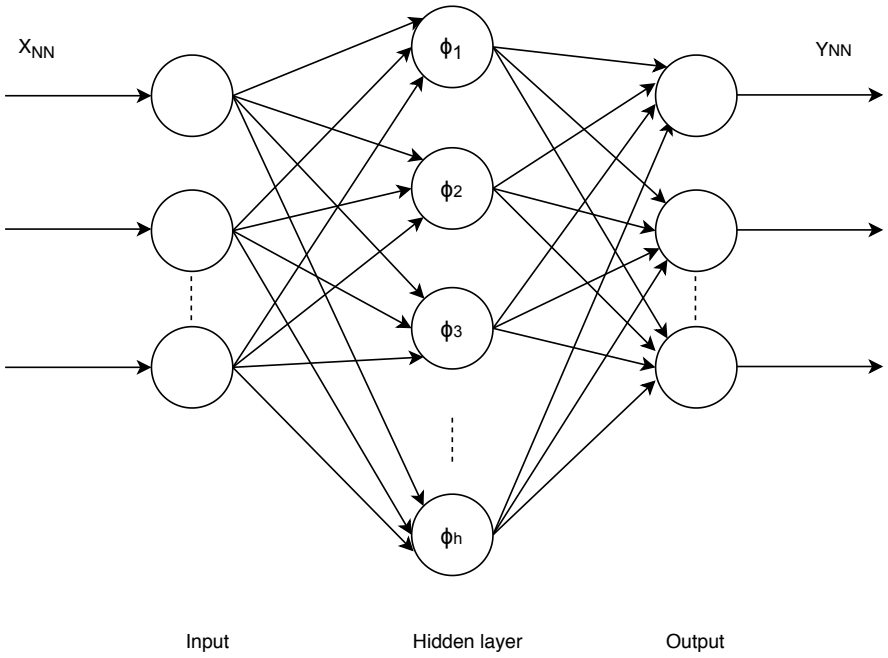



**Figure 4.4:** The architecture of RBFNN

where the input vector of the neural network can be represented as $\mathbf{x}_{NN} = [\mathbf{k}, \mathbf{y}, \mathbf{i}_{host}] \in \mathbf{R}^m$, the output vector can be represented as $\mathbf{y}_{NN} = [\mathbf{x}_{err}] \in \mathbf{R}^n$.

In a compact form, the function of the neural network with inputs $\mathbf{x}_{NN} \in \mathbf{R}^m$ and outputs $\mathbf{y}_{NN} \in \mathbf{R}^n$ can be expressed as:

$$\mathbf{y}_{NN} = \mathbb{W}^T \mathbf{\Phi}(\mathbf{x}_{NN}) \tag{4.4}$$

where $\mathbb{W}$ is the weight matrix.

### 4.2.1 Data collection

To collect data, two car driving tests are conducted in the thesis work. The first one collects data for neural network training. The second one collects test data for evaluation. The DAQ systems were equipped to collect the ground truth data. CANoe is used to collect sensor data in the running car. By connecting CAN bus to CANoe as shown in Figure 4.5, it analyzes the bus communications and sends measured data to Matlab.

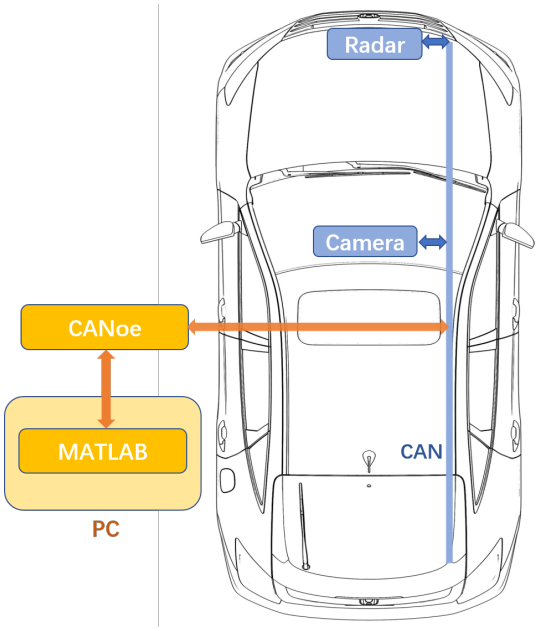



**Figure 4.5:** Test setup with CANoe

Both tests consist of same test scenarios. But the data collected in the first test are used for network training, whereas the data collected in the second test are used to evaluate the performance of the fusion system. The first scenario is that target car runs at a constant speed or parks on the way, the host car follows the target at a variable speed. The second scenario is that target car runs at a variable speed, the host car follows the target at a variable speed. The host vehicle and the target vehicle run on a test track which includes straight line and curve, operate according to the test scenarios with acceleration and turning. Speed is limited with a range of 0 to 60 $\mathrm{km\,h^{-1}}$, and the maximum absolute acceleration is limited to 2.5 $\mathrm{m\,s^{-2}}$.

### 4.2.2 Parameters determinations

- **Data pre-processing**

  Both the inputs and target of the network are scaled to normalize the training set with a mean of 0 and a standard deviation of 1.

  (4.5) represents the normalization algorithm,

$$x_i^{nor} = (x_i - x_{mean}))/x_{std} \tag{4.5}$$

  where $x_i$ is the single row vector of the input data sets and $x_i^{nor}$ is the normalized $x_i$. The $x_{mean}$ and $x_{std}$ represents the mean and standard deviation of the data sets respectively, which will also be used to normalize the input data in

the application after training is completed.

- **Incremental constructive method**

  The RBFNN used in the project is designed with an incremental constructive method proposed in [40].
  The Orthogonal Least Squares(OLS) [41] algorithm is used to select the new center in each iteration. The method is proved to be able to make the single-hidden-layer feedforward networks (SLFNs) with radial basis function (RBF) hidden nodes a universal approximator even without tuning all the parameters(tuning all the parameters might lead to high complication and inefficiency). After randomly selecting the hidden neurons, it only adjusts the weights linking the output layer and the hidden layer to achieve the network function.

  In the application, a maximum number of radial basis functions (hidden neurons) and target accuracy are set for the learning process. According to [41], the learning algorithm can be summarized as

  – Step1. Compute
  for $1 <= i <= N$, compute

$$\begin{aligned} w_l^{(i)} &= p_i \\ g_l^{(i)} &= (w_l^{(i)})^T d/((w_l^{(i)})^T w_l^{(i)}) \\ [err]_l^{(i)} &= (g_l^{(i)})^2 (w_l^{(i)})^T w_l^{(i)}/d^T d \end{aligned} \tag{4.6}$$

  where $N$ is the number of training sets, $p$ is the input vector.

  – Step2. Find the index of the most significant error, add a new neuron.
  Find the index $i_l$ when

$$[err]_l^{(i_l)} = \max\{[err]_l^{(i)}, ic <= i <= N\}$$

  and the vector in input data sets with index $i_l$ will set as center of the new neuron.

$$\text{centers} = [\text{centers}, X(i)]$$

  – Step3. Adjust the weights and bias accordingly and calculate the mean square error (MSE) of the network. Check the MSE, exit if MSE is less than the target

$$a1 = \Phi(\text{centers}, p, \theta),$$

where $\Phi$ is the Gaussian function, $w2$, $b2$ then can be calculated by $a1$ and target $t$, since

$$t = w2 * a1 + b2 = [w2, b2] * [a1; 1]$$

so the $w2$ and $b2$ can be obtained from

$$[w2, b2] = t/[a1; 1]$$

the output of neural network $a2$ can be represented as

$$a2 = w2 * a1 + b2$$

– Step4. At the $k$-th iteration where $k >= 2$, for $1 <= i <= N$, $i \neq i_l$, compute

$$\begin{aligned} \alpha_{jk}^{(i)} &= w_j^T p_i/(w_j^T w_j), 1 <= j < k \\ w_k^{(i)} &= p_i - \sum_{j=1}^{k-1} \alpha_{jk}^{(i)} w_j \\ g_k^{(i)} &= (w_k^{(i)})^T d/((w_k^{(i)})^T w_k^{(i)}) \\ [err]_k^{(i)} &= (g_k^{(i)})^2 (w_k^{(i)})^T w_k^{(i)}/d^T d \end{aligned} \tag{4.7}$$

Goto step2.

The algorithm was implemented in Matlab. Based on our training sets, the training process cost around 1 hour with the laptop. Then the parameters of Radial basis function center, width, weights, bias, hidden neurons were determined.

## 4.3 Data association and Gating

Although multiple object tracking is beyond the scope of this project, data association is still required since some fake targets(non-vehicle object) are detected by the radar sensor in the test environment. For that reason, the simplest method, Nearest Neighbor Data Association is adopted in the fusion filter.

Before the data association, gating, in which gates are created around the possible measurements to filter the unlikely pairings, is adopted to check the measurements

and remove the unlikely ones, which then reduce the computing load for data association. In the project, the gate value is set as: longitude 9.5 meters, latitude 1.0 meters, which means the measurements with more than 9.5 meters of difference in longitude or more than 1.0 meters of difference in latitude from the predicted target distance are ignored.

## 4.4 Sequential sensor fusion

In the project, measured data from each sensor are processed independently, and then the update step is executed sequentially to produce the state estimate [42]. For each observation set of camera or radar, a Kalman filter is applied to generate the state estimate. Observation matrix $\mathbf{H}$ or function $h$ is set up based on the feature of corresponding sensor measured data, and the gain matrix $\mathbf{K}$ is computed accordingly. The set of measured data from two sensors at time $k$ is:

$$\mathbf{Z}(k) = \{\mathbf{z_1}(k), \mathbf{z_2}(k)\} \tag{4.8}$$

Based on the asynchronous observations, it generates sequential update to the state estimate. If the measurements from different sensors arrive at the same time, updates for the two sensors are implemented one after another in a sequential order with the same prediction as illustrated below.

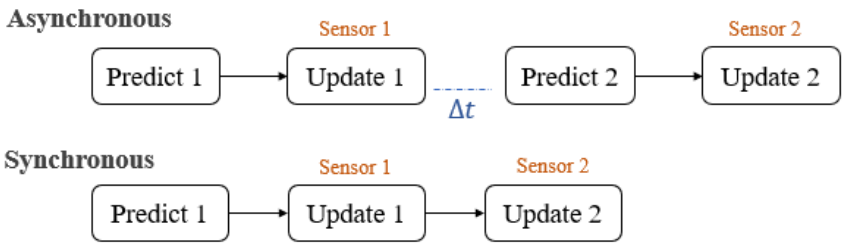



**Figure 4.6:** Sequential sensor fusion under asynchronous and synchronous circumstances

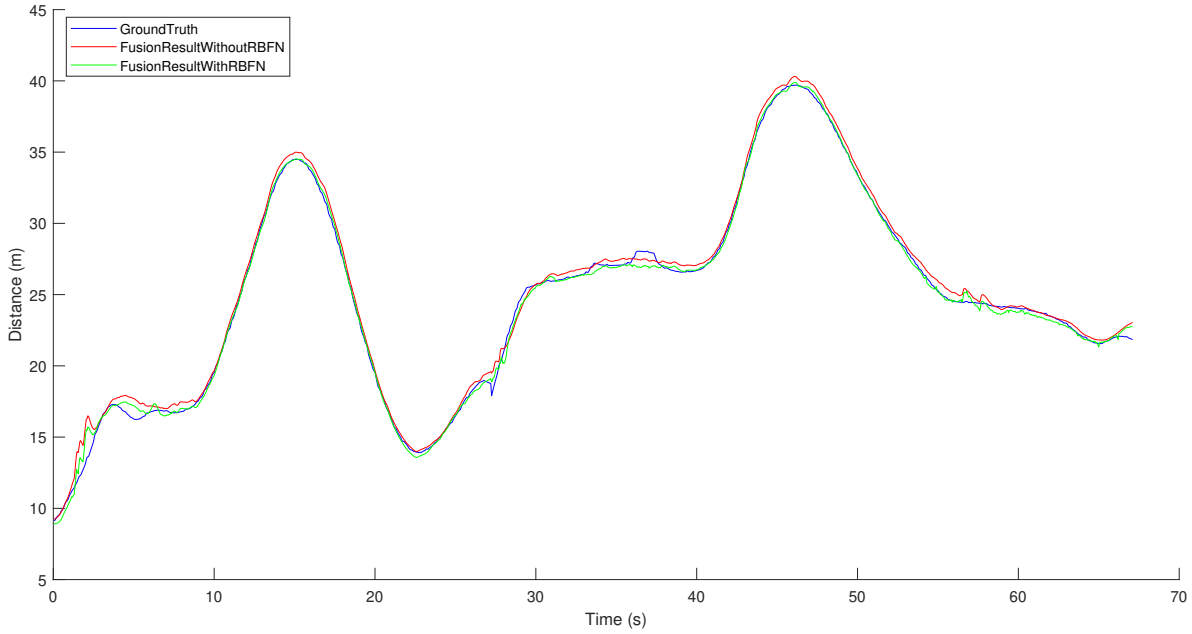
# 5
# Results

To evaluate the performance of the RBFNN aided Kalman filter, experiments are carried out with both EKF and CMKF, the performance are compared respectively to prove the improvement after integrating the RBFNN.
In our test, the camera was not applied since the hardware was broken. All the test results are based on the radar sensor only. In addition, the effect of missing camera data will be discussed in the next section.

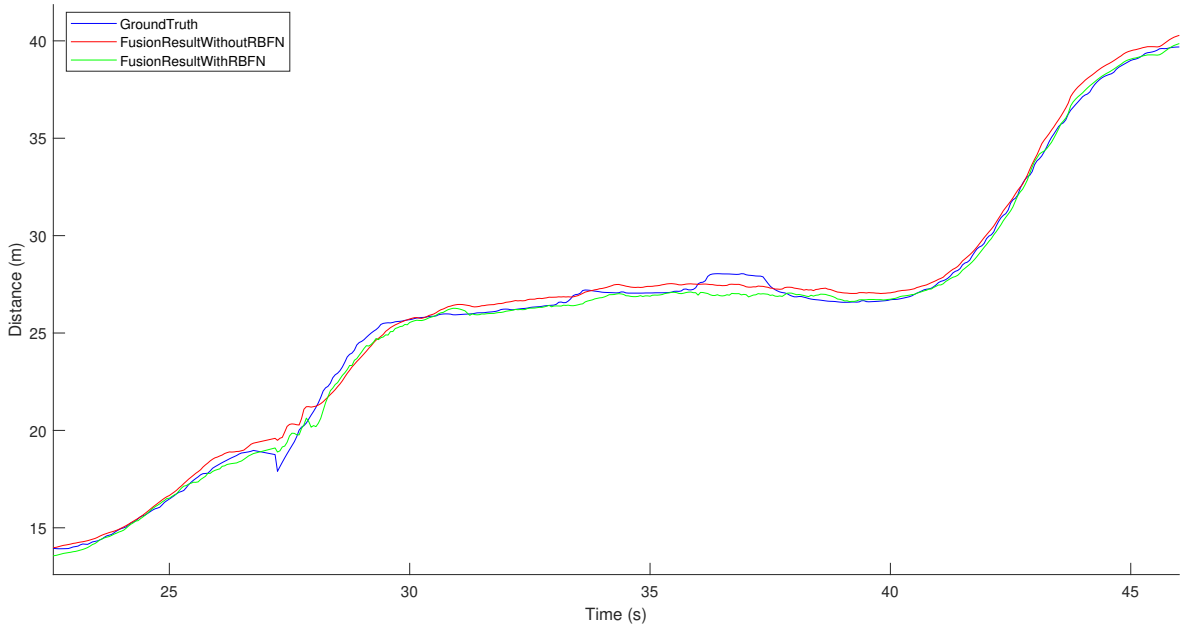
## 5.1 CMKF with RBFN

In this section, the performance of the RBFNN aided CMKF is compared with that of a stand-alone CMKF. During the test, the target car ran randomly on the test track, followed by the host car with a randomly varied distance. The fusion filter was applied to estimate the target car's position. The ground truth data was obtained from the DAQ system. The output of the CMKF and the output after being compensated by the neural network were recorded.

Figure 5.1 shows the performance of the designed fusion filter and CMKF in estimating longitudinal distance from the host to the target with full view and a zoom-in view.




(a) Full view

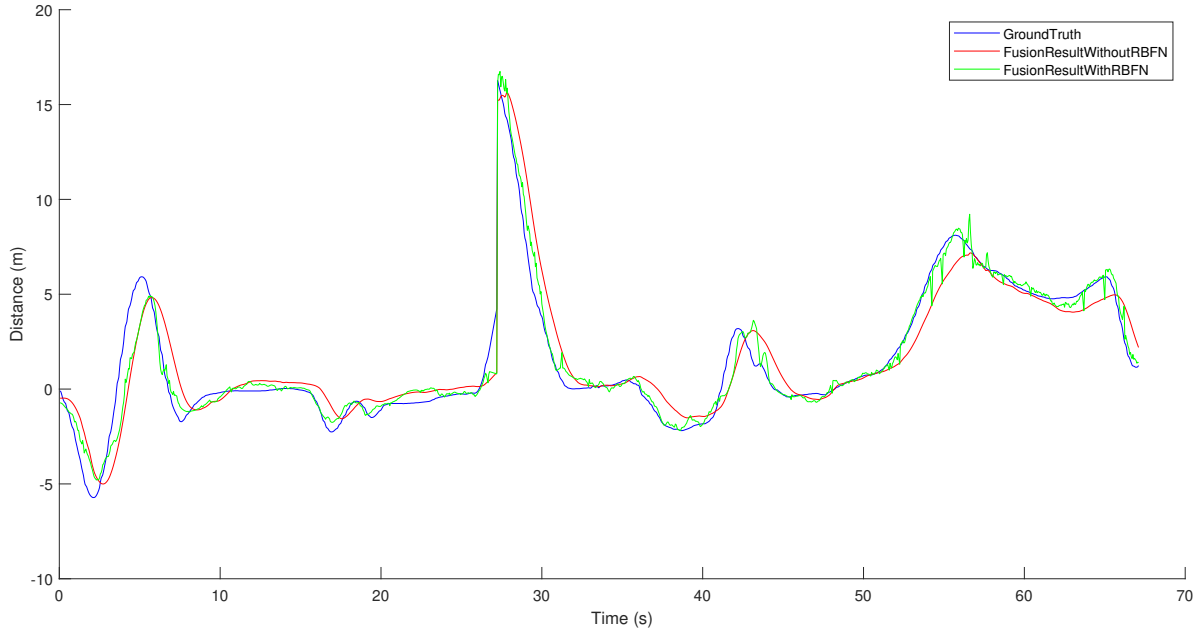



(b) Zoom-in view

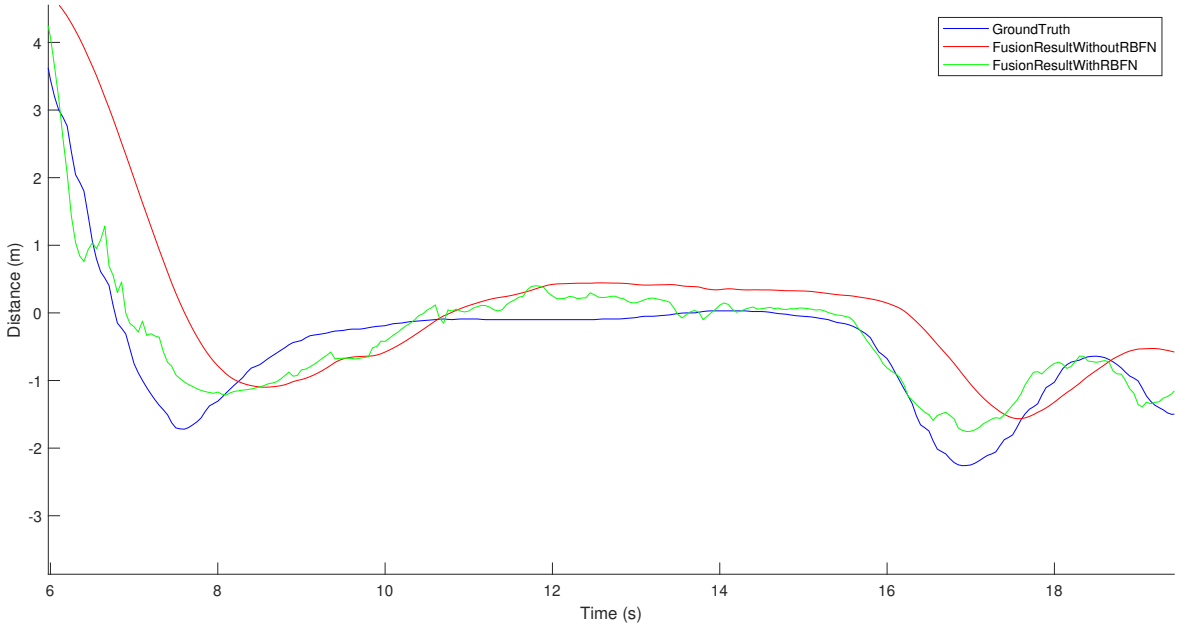
**Figure 5.1:** Comparison results between different models: Distance(longitude). The green line represents the fusion filter (RBFNN+CMKF), whereas the red line represents the CMKF. The blue line represents the ground truth, which is measured by the DAQ system.

Comparison of the performance of CMKF and CMKF+RBFNN in latitudinal dis-

tance estimation with full view and a zoom-in view are shown in Figure 5.2.




(a) Full view




(b) Zoom-in view

**Figure 5.2:** Comparison results between different models: Distance(latitude)

It can be observed that the compensated output of the designed fusion filter is closer to the actual distance most of the time.

We use Root Mean Square Error (RMSE) and Normalized Root Mean Square Error (NRMSE) as the evaluation criteria. They can be calculated by

$$RMSE(\hat{x}) = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \hat{x}_i)^2}{n}}$$
$$NRMSE(\hat{x}) = \frac{RMSE(\hat{x})}{x_{max} - x_{min}} \tag{5.1}$$

where $x$ is the ground truth value and $\hat{x}$ is the estimate generated by the filter.

Table 5.1 outlines a statistical analysis of the position (longitude) estimation of the CMKF and RBFNN aided CMKF in contrast to the actual relative position of the target car. These results suggest a decrease of 34.7% in NRMSE.

**Table 5.1:** Statistical analysis of Fig 5.1

| | RMSE | NRMSE |
|---|---|---|
| CMKF | 0.5381 | 0.0173 |
| RBFNN aided CMKF | 0.3504 | 0.0113 |

Table 5.2 shows a statistical analysis of the position(latitude) estimation. These results suggest a decrease of 42.9% in NRMSE.

**Table 5.2:** Statistical analysis of Fig 5.2

| | RMSE | NRMSE |
|---|---|---|
| CMKF | 1.2503 | 0.0606 |
| RBFNN aided CMKF | 0.7448 | 0.0346 |

To investigate the appropriate width $\sigma$ of the radial basis function, a series of parameters are evaluated with the same number of hidden neurons(that is 200). Moreover, RMSE and NRMSE are listed in Table 5.3 and Table 5.4.

**Table 5.3:** Statistical analysis: longitude

| | RMSE | NRMSE |
|---|---|---|
| CMKF | 0.5381 | 0.0173 |
| RBFNN aided CMKF(width:1.3876) | 0.3743 | 0.0121 |
| RBFNN aided CMKF(width:1.0407) | 0.3504 | 0.0113 |
| RBFNN aided CMKF(width:0.8326) | 0.3489 | 0.0113 |
| RBFNN aided CMKF(width:0.6938) | 0.3375 | 0.0110 |
| RBFNN aided CMKF(width:0.5947) | 0.3672 | 0.0119 |

In Table 5.3, for longitude distance measurement, both RMSE and NRMSE decrease as the width of radial basis function decreases from 1.3879 to 0.6938, the minimum

RMSE is 0.3375, and the minimum NRMSE is 0.0110, with the function width of 0.6938.

**Table 5.4:** Statistical analysis: latitude

| | RMSE | NRMSE |
|---|---|---|
| CMKF | 1.2503 | 0.0606 |
| RBFNN aided CMKF(width:1.3876)) | 0.9754 | 0.0447 |
| RBFNN aided CMKF(width:1.0407) | 0.7448 | 0.0346 |
| RBFNN aided CMKF(width:0.8326) | 0.6782 | 0.0314 |
| RBFNN aided CMKF(width:0.6938) | 0.5945 | 0.0268 |
| RBFNN aided CMKF(width:0.5947) | 0.5100 | 0.0228 |

In Table 5.4, for latitude distance measurement, RMSE and NRSE decrease as the width of the radial basis function decrease from 1.3879 to 0.5947.

Finally, the width 0.06938 was selected in the design since the longitude distance is of more importance.

## 5.2 EKF with RBFN

In this section, the performances of the proposed fusion filter(RBFNN aided EKF) are compared with that of the EKF. The output of the EKF and the output after being compensated by the neural network were recorded along with the measurements of DAQ system.

Figure 5.3 illustrates the estimated longitude distance from both the EKF and the RBFNN aided EKF to investigate the performance improvement.

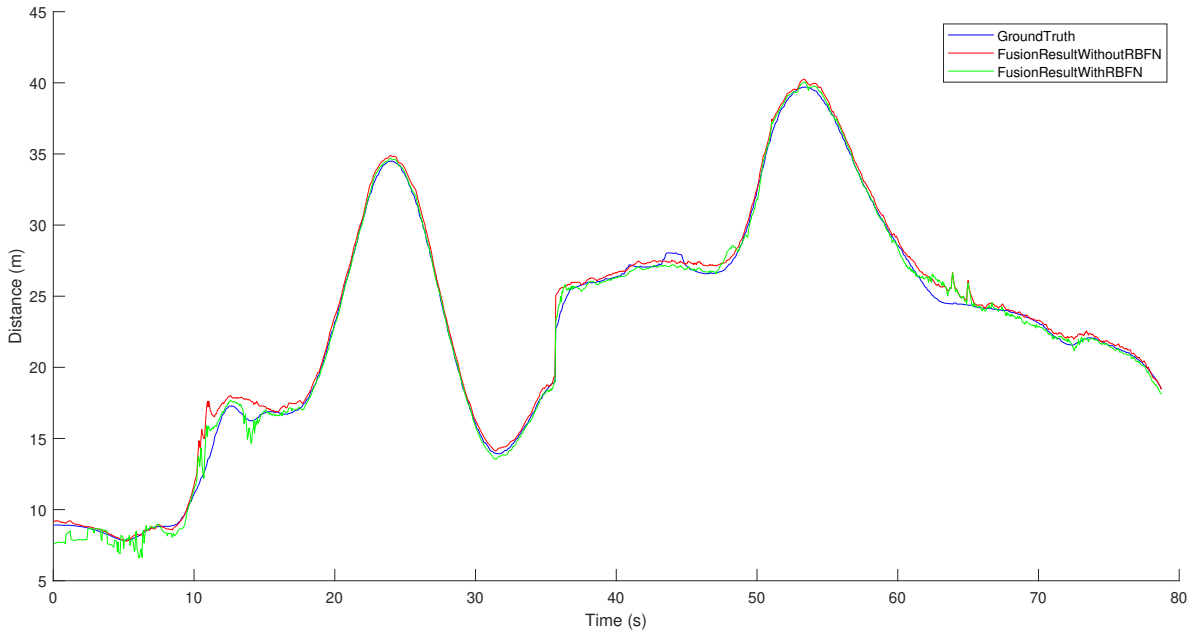



**Figure 5.3:** Comparison results between different models: Distance(longitude) The green line represents the RBFNN aided EKF, whereas the red line, the blue line represents the EKF and ground truth, respectively.

Comparison of EKF and RBFNN aided EKF in latitude distance estimation is shown in Figure 5.4.

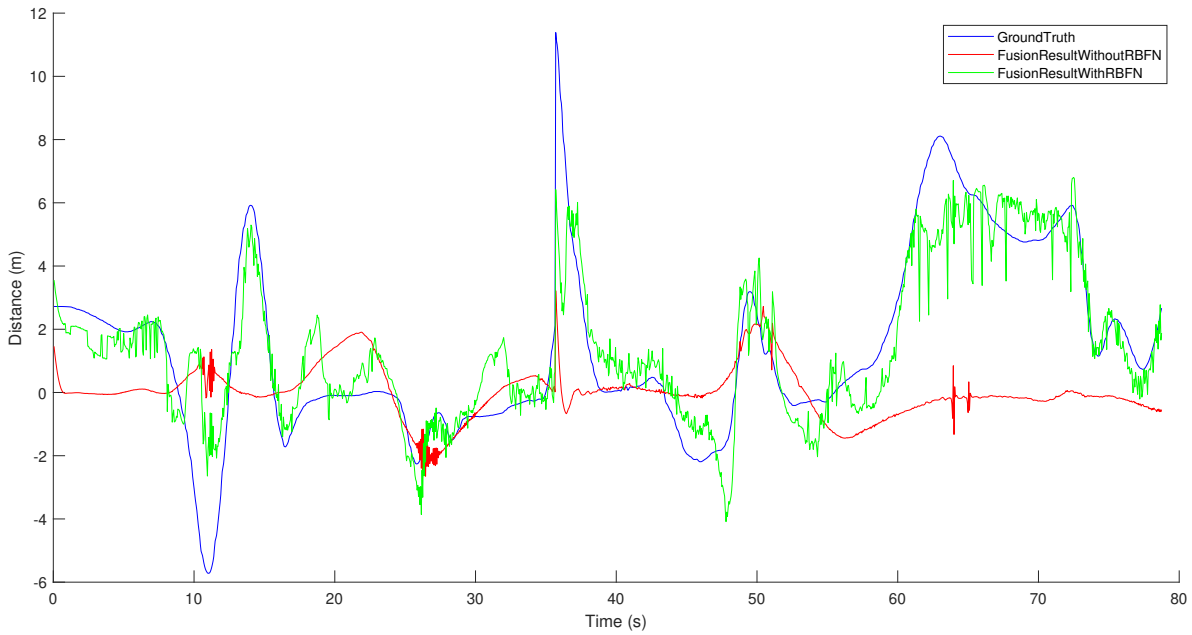



**Figure 5.4:** Comparison results between different models: Distance(latitude)

Table 5.5 summarizes the estimation precision based on RMSE and NRMSE for the longitude distance estimation.

**Table 5.5:** Statistical analysis of Fig 5.3

| | RMSE | NRMSE |
|---|---|---|
| EKF | 0.6070 | 0.0187 |
| RBFNN aided EKF | 0.4823 | 0.0144 |

The result suggests a decrease of 23.0% in NRMSE when combining the RBFNN with the EKF.

Moreover, table 5.6 demonstrates a statistical analysis of the position(longitude distance) estimation. These results suggest a decrease of 77.5% in NRMSE.

**Table 5.6:** Statistical analysis of Fig 5.4

| | RMSE | NRMSE |
|---|---|---|
| EKF | 3.2085 | 0.5468 |
| RBFNN aided EKF | 1.3426 | 0.1233 |

# 6
# Discussion

Lack of camera measurements might affect the overall performance of the filter. However, the purpose of the design is to prove the effectiveness of the sensor fusion filter structure by comparing the stand alone Kalman filter with the RBFNN aided Kalman filter. Besides, the camera's accuracy of distance measurement is much lower compared to the radar, especially in the longitude direction, which indicates the performance will not be significantly decreased without camera.

In addition to distance measurement, the primary function of the camera is object identification. Without the camera, it required much more effort to associate the radar measurements with the target position. Since apart from the target car, any other objects appearing in the radar range might be measured and sent to the CAN bus.

The promising results of the experiment in the previous section indicate that a fusion filter combining neural network and conventional Kalman filter can achieve a better performance than the stand-alone conventional Kalman filter. In the experiments, the CMKF is well pre-tuned for the test, and the EKF is configured with some initial parameters without tuning.
The experiment shows that the neural network can be a universal tool to compensate for different Kalman filter types even if the filter is not well-tuned.

Contrary to the design in [4] [7], in which the neural network may be trained online to dynamically update the parameters of the Kalman filter or provide the position estimation, our neural network is trained offline before being deployed to the application. It significantly saves computing resource and makes it easier to execute in real time on an embedded platform which may have limited computing capacity.

Because of the instability of the radar sensor, the radar track for the target car might change at times. It might lead to an unsteady state of Kalman filter for a while, during which the estimate error of the Kalman filter might be considerable. Since improving the original fusion model is not in the scope of our design, the signals in the short unsteady period are removed from the data sets before training the RBFNN. Also, the unsteady period is removed from the test result.

The RBFNN consists of 200 hidden neurons, nine inputs, and two outputs. The floating-point operation (FLOP) amount is always used to measure the complexity of the neural network. From[43], the direct method to calculate an RBF function

may cost $O(N^2)$ storage usage and $O(N^3)$ operation usage, which is 40k storage and 8M FLOPs. However, the domain decomposition method proposed in [43] can reduce the cost of solving RBF equations dramatically, only $O(N)$ storage and $O(N \log N)$ FLOPs are required for the calculation, which equals 200 storage and 1060 FLOPs. The microcontroller(MCU) used in the current project at CEVT has a 2 x 160 MHz e200Z4 Dual-core 32-bit CPU with a single-precision floating-point unit, which makes it possible to deploy the fusion filter with a trained RBFNN to the MCU. However, to provide a more accurate analysis of resource usage, the Matlab model should be converted to C++ code, which then is compiled to execute on the MCU so that we can monitor the hardware usage.

With current training sets, it took 1 hour to train the neural network on a laptop. The training time might increase with more training sets obtained in some more complex test environment. In this project, the training sets limit the generalizability of the results since all the tests are performed in the proving ground. More measurement data on real roads should be collected for RBFNN training, and the experiment should be carried out in more complex scenarios for practical use.

# 7
# Conclusion

This paper presents a sensor fusion filter architecture with RBFNN aided Kalman filter to position the target car in real time with the sensors, camera and radar. The algorithm designed aims at further implementation in a MCU platform in which the RBFNN is trained offline and deployed with the fusion filter. With an RBFNN, the fusion filter is proved to achieve higher performance. The experiment was carried out with both CMKF and EKF. The test results demonstrate that the proposed fusion filter architecture performs better to position the target car on the track than the stand-alone Kalman filter, whether the Kalman filter is well-tuned or not.

Future developments aim at updating the association algorithm to achieve multiple object tracking; also, the RBFNN can be tuned further to decrease the neurons number. Furthermore, plenty of training sets in more complex scenario should be collected for network training to make the fusion filter reliable on the real road.

# Bibliography


[1] R. Kala, "4 - advanced driver assistance systems," in *On-Road Intelligent Vehicles*, R. Kala, Ed. Butterworth-Heinemann, 2016, pp. 59 – 82. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128037294000040

[2] D. L. Hall and J. Llinas, "An introduction to multisensor data fusion," *Proceedings of the IEEE*, vol. 85, no. 1, pp. 6–23, 1997.

[3] W. van Drongelen, "Chapter 19 - kalman filter," in *Signal Processing for Neuroscientists (Second Edition)*, second edition ed., W. van Drongelen, Ed. Academic Press, 2018, pp. 361 – 374. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128104828000199

[4] O. V. Korniyenko, M. S. Sharawi, and D. N. Aloi, "Neural network based approach for tuning kalman filter," in *2005 IEEE International Conference on Electro Information Technology*, May 2005, pp. 1–5.

[5] A. Gelen and A. Atasoy, "A new method for kalman filter tuning," 09 2018.

[6] S. Akhlaghi, N. Zhou, and Z. Huang, "Adaptive adjustment of noise covariance in kalman filter for dynamic state estimation," in *2017 IEEE Power Energy Society General Meeting*, July 2017, pp. 1–5.

[7] N. Yipeng, J. Wang, H. Han, X. Tan, and T. Liu, "An optimal radial basis function neural network enhanced adaptive robust kalman filter for gnss/ins integrated systems in complex urban areas," *Sensors*, vol. 18, p. 3091, 09 2018.

[8] V. Pesce, S. Silvestrini, and M. Lavagna, "Radial basis function neural network aided adaptive extended kalman filter for spacecraft relative navigation," *Aerospace Science and Technology*, 11 2019.

[9] K. L. Bell, T. L. Corwin, L. D. Stone, and R. L. Streit, "4. classical multiple target tracking," in *Bayesian Multiple Target Tracking (2nd Edition)*. Artech House, 2014, pp. 107–108. [Online]. Available: https://app.knovel.com/hotlink/khtml/id:kt011LSZ72/bayesian-multiple-target/classical-multiple-target

[10] H. S. Chadha, "Kalman filter interview," 2018, available at https://towardsdatascience.com/kalman-filter-interview-bdc39f3e6cf3,.

[11] R. Serrano, "Extended kalman filters for dummies," 2018, available at https://medium.com/@serrano_223/extended-kalman-filters-for-dummies-4168c68e2117,.

[12] Q. Li, R. Li, K. Ji, and W. Dai, "Kalman filter and its application," in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, 2015, pp. 74–77.

[13] N. Y. Ko and T. G. Kim, "Comparison of kalman filter and particle filter used for localization of an underwater vehicle," pp. 350–352, 2012.

[14] D. Lerro and Y. Bar-Shalom, "Tracking with consistent converted measurements vs the ekf," *IFAC Proceedings Volumes*, vol. 26, no. 2, Part 5, pp. 365 – 368, 1993, 12th Triennal Wold Congress of the International Federation of Automatic control. Volume 5 Associated Technologies and Recent Developments, Sydney, Australia, 18-23 July. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1474667017482887

[15] Z. Kowalczuk, M. Sankowski, and P. Uruski, *Navigational-Radar Tracking of a Maritime Target in Clutter*. Dordrecht: Springer Netherlands, 2002, pp. 917–922. [Online]. Available: https://doi.org/10.1007/978-94-010-0556-2_51

[16] P. L. Vu-Quoc, "Neuron3," September 2018. [Online]. Available: https://commons.wikimedia.org/wiki/File:Neuron3.png

[17] "A beginner's guide to neural networks and deep learning." [Online]. Available: https://wiki.pathmind.com/neural-network

[18] M. A. Nielsen, "Neural networks and deep learning," 2018. [Online]. Available: http://neuralnetworksanddeeplearning.com/

[19] C. M. Bishop, *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995.

[20] F. Schwenker, H. Kestler, and G. Palm, "Three learning phases for radial-basis-function networks." *Neural networks : the official journal of the International Neural Network Society*, vol. 14, pp. 439–58, 06 2001.

[21] Li Jun and T. Duckett, "Some practical aspects on incremental training of rbf network for robot behavior learning," in *2008 7th World Congress on Intelligent Control and Automation*, 2008, pp. 2001–2006.

[22] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern Recognition*, vol. 36, no. 2, pp. 451 – 461, 2003, biometrics. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320302000602

[23] D. Reddy and P. K. Jana, "Initialization for k-means clustering using voronoi diagram," *Procedia Technology*, vol. 4, pp. 395 – 400, 2012, 2nd International Conference on Computer, Communication, Control and Information Technology( C3IT-2012) on February 25 - 26, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2212017312003404

[24] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

[25] V.-M. Benoudjit, N., "On the kernel widths in radial-basis function networks," *Neural Processing Letters*, vol. 18, p. 139–154, 2003. [Online]. Available: https://doi.org/10.1023/A:1026289910256

[26] N. Benoudjit, C. Archambeau, A. Lendasse, J. Lee, and M. Verleysen, "Width optimization of the gaussian kernels in radial basis function networks." 01 2002, pp. 425–432.

[27] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis, "Effective backpropagation training with variable stepsize," *Neural Networks*, vol. 10, no. 1, pp. 69 – 82, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608096000524

[28] "Delphi esr startup guide," April 2015.

[29] "Mobileye me630 user manual."

[30] "Dewesoft™- rtk users manual version: 1.0.1." [Online]. Available: https://d36j349d8rqm96.cloudfront.net/3/6/RTK_manual.pdf
[31] "Dewesoft™-general catalog," 2020. [Online]. Available: https://d36j349d8rqm96.cloudfront.net/3/218/DEWESoft_Product_Catalog_EN_latest.pdf
[32] "Dewesoft™-sirius technical reference manual." [Online]. Available: https://d36j349d8rqm96.cloudfront.net/3/6/Dewesoft-SIRIUS-Manual-EN.pdf
[33] "Canoe product information," June 2020.
[34] "Vn8900 interface family manual, version 6.4."
[35] "Base unit vn8911 frontside." [Online]. Available: https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/vn89xx/#c11482
[36] "Base unit vn8911 backside." [Online]. Available: https://www.vector.com/int/en/products/products-a-z/hardware/network-interfaces/vn89xx/#c11482
[37] X. Gao, X. Zhong, D. You, and S. Katayama, "Kalman filtering compensated by radial basis function neural network for seam tracking of laser welding," *IEEE Transactions on Control Systems Technology*, vol. 21, no. 5, pp. 1916–1923, 2013.
[38] Y. Wu, H. Wang, B. Zhang, and K.-L. Du, "Using radial basis function networks for function approximation and classification," *ISRN Applied Mathematics*, vol. 2012, 03 2012.
[39] Y. Zhang, Q. Du, S. Yu, and J. Pan, "Rbf neural network based on fuzzy evolution kalman filtering and application in mine safety monitoring," in *2009 Ninth International Conference on Hybrid Intelligent Systems*, vol. 1, 2009, pp. 467–470.
[40] Y. W. Wong, K. P. Seng, and L. Ang, "Radial basis function neural network with incremental learning for face recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 41, no. 4, pp. 940–949, 2011.
[41] M. L. Kothari, S. Madnani, and R. Segal, "Orthogonal least squares learning algorithm based radial basis function (rbf) network adaptive power system stabilizer," in *1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, vol. 1, 1997, pp. 542–547 vol.1.
[42] H. Durrant-Whyte, "Multi sensor data fusion." Australian Centre for Field Robotics The University of Sydney NSW 2006, 02 2001, pp. 77–81.
[43] R. Beatson, J. Cherrie, and C. Mouat, "Fast fitting of radial basis functions: Methods based on preconditioned gmres iteration," *Advances in Computational Mathematics*, vol. 11, pp. 253–270, 11 1999.