Improving the Security and Safety of Transportation Cyber-Physical Systems

Md Mhafuzul Islam

Clemson University, MDMHAFI@CLEMSON.EDU

Follow this and additional works at: https://tigerprints.clemson.edu/all_dissertations

Recommended Citation
https://tigerprints.clemson.edu/all_dissertations/2809

This Dissertation is brought to you for free and open access by the Dissertations at TigerPrints. It has been accepted for inclusion in All Dissertations by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.
IMPROVING THE SECURITY AND SAFETY OF TRANSPORTATION CYBER-PHYSICAL SYSTEMS

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Civil Engineering

by
Md Mhafuzul Islam
May 2021

Accepted by:
Dr. Mashrur Chowdhury, Committee Chair
Dr. Amy Apon
Dr. Feng Luo
Dr. Long Cheng
The transportation system is rapidly evolving with new connected and automated vehicle (CAV) technologies that integrate CAVs with other vehicles and roadside infrastructure to form a transportation cyber-physical system (TCPS). Through connectivity, CAVs affect their environments and vice versa, increasing the size of the cyberattack surface and the risk of exploitation of security vulnerabilities by malicious actors. Thus, a greater understanding of potential CAV-TCPS cyber-attacks and of ways to prevent them is a high priority. Moreover, making the CAV navigate safely in an unexpected environment is a critical safety requirement. Considering the safety while maintaining the in-vehicle security is the focus of this study, where first, in part 1, the author explores the CAV safety through machine learning models, more specifically deep neural network, to help the vehicle to navigate safely in an unexpected environment, which is required for real-world deployment and has not been fully explored by researchers and industries. In part 2, the author developed a connected vehicle application development platform (CVDeP), such that developers can develop and validate the CAV safety and mobility applications in a controlled and real-world connected vehicle testbed. Our study shows that applications developed through the platform meet the safety requirements of connected vehicle applications.

Later, in part 3, the author explores the in-vehicle security aspect, where the author leverages the state-of-the-art cloud supported quantum computers to classify in-vehicle cyberattacks, more specifically amplitude shift attacks. The author develop the quantum-classical hybrid neural network to detect amplitude shift in-vehicle cyberattack. This study
integrates the digital infrastructure and a CAV’s in-vehicle system, where the author has shown the potential of using a combination of quantum and classical neural network to improve the cyberattack detection accuracy compared to classical neural network and quantum neural network alone.
DEDICATION

I dedicate this dissertation to my newborn daughter, Atheela Zohr Mhafuz, and my wife, Rasna Sharmin.
ACKNOWLEDGMENTS

I wish to take this opportunity to thank all the people responsible for my successful graduation from the Glenn Department of Civil Engineering at Clemson University. First, I would like to thank my advisor, Professor Dr. Mashrur Chowdhury, for his excellent supervision throughout my graduate study. His sheer brilliance, sound technical insights, immense knowledge, and constant support were highly inspiring, and encouraged me in all of my academic research and daily life. I feel fortunate to know him and to be a part of his research group.

I also would like to thank my committee member Dr. Amy Apon, Dr. Feng Luo, Dr. Long Cheng, and Dr. Hongxin Hu for their guidance and valuable support and for taking the time to review this dissertation.

Appreciation next goes out to our research group members for their valuable feedbacks and scholarly discussion. I especially thank Dr. Sakib Mahmud Khan, Dr. Mizanur Rahman, Zadid Khan, and Hongda Li for supporting me in the technical aspects of my research. I feel lucky to know these brilliant minds and be a part of this research team where collaboration is highly appreciated. I wish them the best for the future.

I would like to thank Dr. Mizanur Rahman, Dr. Sakib Mahmud Khan, and Dr. M Nasimul Hoque Chowdhury for providing focus and inspiration at the early stages of my academic career at Clemson University. Also, I want to thank my friends at Clemson (too many to mention by name). Because of them, thousands of miles away from home and family, I consider Clemson as my second home. The credit goes to a highly jovial and friendly Bangladeshi Community that I was lucky to be a part of.
I also thank the staff of the Civil Department, and in particular Kristi Baker for clearing away all the administrative issues with an ever-present smile.

Above all, I thank my better half, Rasna Sharmin, who defines me and responsible for whatever I have achieved so far in my life.
TABLE OF CONTENTS

ABSTRACT......................................................................................................................ii

DEDICATION................................................................................................................... iv

ACKNOWLEDGMENTS.................................................................................................... v

LIST OF TABLES............................................................................................................ x

LIST OF FIGURES......................................................................................................... xi

CHAPTER ONE

VISION-BASED NAVIGATION OF AUTONOMOUS VEHICLES IN ROADWAY
ENVIRONMENTS WITH UNEXPECTED HAZARDS......................................................1

Introduction................................................................................................................1
Related work ..............................................................................................................4
  DNN-based Autonomous Vehicle Driving Model .............................................4

  Autonomous Vehicle Dataset ...........................................................................6

  DNN-based Object Detection and Segmentation ............................................6

Method .....................................................................................................................8
  DNN-based hazard detection and segmentation model ...............................9

  Hazard analysis and avoidance model .........................................................10

  DNN-based autonomous vehicle driving model ......................................14

Experimental setup ..............................................................................................15
  Data Collection .................................................................................................15

  Data Preparation ..............................................................................................18

  Data Augmentation .........................................................................................20
Introduction..........................................................................................................................68
Cyberattack Detection in IoT Cloud.....................................................................................71
  Preparing dataset..................................................................................................................71
  Feature extraction using classical neural network...............................................................72
  Quantum encoding..............................................................................................................72
  Classification using quantum neural network.................................................................73
Experimental Results........................................................................................................73
Chapter Conclusions.........................................................................................................76
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1. 1</td>
<td>Dataset description</td>
<td>20</td>
</tr>
<tr>
<td>Table 2. 1</td>
<td>Summary of Latency for FCW Application Evaluation</td>
<td>59</td>
</tr>
<tr>
<td>Table 2. 2</td>
<td>Summary of Latency for Mobility Application with Five CVs</td>
<td>66</td>
</tr>
<tr>
<td>Table 3. 1</td>
<td>Hyperparameter for different neural networks</td>
<td>75</td>
</tr>
</tbody>
</table>
**LIST OF FIGURES**

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1. 1 DNN-based autonomous vehicle driving system in an unexpected hazardous environment.</td>
<td>9</td>
</tr>
<tr>
<td>Figure 1. 2 FCN-based object detection and image segmentation model used in this study.</td>
<td>10</td>
</tr>
<tr>
<td>Figure 1. 3 Heatmap for the threat value based on the longitudinal and latitudinal distance of a hazardous object using Radar sensor data.</td>
<td>13</td>
</tr>
<tr>
<td>Figure 1. 4 Heatmap for the threat value based on the location of hazard using pixel value from the segmented image.</td>
<td>14</td>
</tr>
<tr>
<td>Figure 1. 5 CNN-based end-to-end autonomous vehicle driving model used in this study.</td>
<td>15</td>
</tr>
<tr>
<td>Figure 1. 6 Roadway environment setup for an autonomous vehicle with hazardous objects.</td>
<td>16</td>
</tr>
<tr>
<td>Figure 1. 7 Camera placements in the autonomous vehicle (left, center, and right cameras).</td>
<td>17</td>
</tr>
<tr>
<td>Figure 1. 8 Example of images collected by the three cameras in the autonomous vehicle (left, center, and right camera images (from left to right)).</td>
<td>18</td>
</tr>
<tr>
<td>Figure 1. 9 Example of a normalized steering wheel angle plot from the training dataset.</td>
<td>19</td>
</tr>
<tr>
<td>Figure 1. 10 Example of an original and cropped image in the training dataset.</td>
<td>19</td>
</tr>
<tr>
<td>Figure 1. 11 Training and validation performance of the end-to-end driving model for Case 1, Case 2, and Case 3 on the training and validation dataset.</td>
<td>22</td>
</tr>
<tr>
<td>Figure 1. 12 Error measurement on the testing dataset.</td>
<td>24</td>
</tr>
<tr>
<td>Figure 1. 13 Qualitative results of ground truth, Case 1, Case 2, and Case 3 of the driving direction.</td>
<td>26</td>
</tr>
<tr>
<td>Figure 1. 14 Trajectory of the autonomous vehicle for ground truth, Case 1, Case 2, and Case 3.</td>
<td>28</td>
</tr>
<tr>
<td>Figure 2. 1 Approach for the CVDeP development, evaluation and validation</td>
<td>38</td>
</tr>
<tr>
<td>Figure 2. 2 The CVDeP architecture for an edge-centric CPS</td>
<td>40</td>
</tr>
<tr>
<td>Figure 2. 3 Implementation of application security management module, and access control and credential management module</td>
<td>44</td>
</tr>
<tr>
<td>Figure 2. 4 Implementation of application development graphical user interface</td>
<td>50</td>
</tr>
<tr>
<td>Figure 2. 5 CVDeP setup in an emulated edge-centric CPS</td>
<td>53</td>
</tr>
<tr>
<td>Figure 2. 6 Implementation of CVDeP modules in an emulated edge-centric CPS</td>
<td>54</td>
</tr>
<tr>
<td>Figure 2. 7 Evaluation of CVDeP for mobility application using application throughput and latency</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Figures (Continued)

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 3. 1</td>
<td>Cloud based In-vehicle Cyberattack Detection System</td>
<td>70</td>
</tr>
<tr>
<td>Figure 3. 2</td>
<td>Hybrid quantum-classical neural network</td>
<td>71</td>
</tr>
<tr>
<td>Figure 3. 3</td>
<td>Quantum encoded binary data</td>
<td>73</td>
</tr>
<tr>
<td>Figure 3. 4</td>
<td>Comparable classical neural network</td>
<td>74</td>
</tr>
<tr>
<td>Figure 3. 5</td>
<td>Comparable quantum only neural network</td>
<td>74</td>
</tr>
<tr>
<td>Figure 3. 6</td>
<td>Cyberattack detection accuracy for different neural networks</td>
<td>75</td>
</tr>
</tbody>
</table>
CHAPTER ONE
VISION-BASED NAVIGATION OF AUTONOMOUS VEHICLES IN ROADWAY ENVIRONMENTS WITH UNEXPECTED HAZARDS

Introduction

Vision-based navigation of autonomous vehicles primarily depends on the Deep Neural Network (DNN) based systems in which the controller obtains input from sensors/detectors, such as cameras and produces a vehicle control output, such as a steering wheel angle to navigate the vehicle safely in a roadway traffic environment. Typically, these DNN-based systems of the autonomous vehicle are trained through supervised learning; however, recent studies show that a trained DNN-based system can be compromised by perturbation or adversarial inputs. Similarly, this perturbation can be introduced into the DNN-based systems of autonomous vehicles by unexpected roadway hazards, such as debris and roadblocks. In this study, the author first introduces a roadway hazardous environment (both intentional and unintentional roadway hazards) that can compromise the DNN-based navigational system of an autonomous vehicle, and produces an incorrect steering wheel angle, which can cause crashes resulting in fatality and injury. Then, the author develops a DNN-based autonomous vehicle driving system using object detection and semantic segmentation to mitigate the adverse effect of this type of hazardous environment, which helps the autonomous vehicle to navigate safely around such hazards. The author find that our developed DNN-based autonomous vehicle driving system including hazardous object detection and semantic segmentation improves the navigational
ability of an autonomous vehicle to avoid a potential hazard by 21% compared to the traditional DNN-based autonomous vehicle driving system (Islam et al. 2019).

According to the 2016 American automobile association report, 50,658 crashes occurred in the U.S. from the year 2011 to 2014 due to debris resulting in 9,805 injuries and 125 deaths annually (Tefft 2016). The roadway hazards, such as debris, are considered to be non-fixed and unexpected objects on the travel or driving lane of the roadway and include objects that have fallen from vehicles or have come from construction sites or littering. Given that the autonomous vehicle is considered the future of surface transportation, its ability to detect debris or hazards and then navigate safely around them is crucial for avoiding potential crashes. Recently, such a navigational task has been accomplished using Deep Neural Network (DNN). Typically, an autonomous vehicle perceives its surrounding roadway environment using sensors, and the software running in the vehicle determines the action to be taken based on the input from the sensors. Several types of sensors, such as vision-based sensors (e.g., Cameras), LIDAR, and Radar are currently available for the perception task. Due to the cost-effectiveness of the vision-based sensor compared to the other types of sensors (e.g., LIDAR and Radar), vision-based navigation becomes an attractive solution for autonomous vehicles (Bertozzi, Broggi, and Fascioli 2000)(Dagan et al. 2004)(Taterek, Kronenberger, and Handmann 2017).

The recent development of DNNs, in particular, Convolutional Neural Network (CNN)(Krizhevsky, Sutskever, and Hinton 2012), has improved vision-based navigation for autonomous vehicles significantly. After being trained and tested using a dataset collected by sensors, these CNN models are then deployed in autonomous vehicles to
navigate the vehicle safely. For example, during training, the CNN-based end-to-end driving model maps a relationship between the driving behavior of humans using roadway images collected from cameras and the steering wheel angle (Yang et al. 2018)(Bojarski et al. 2016). Thus, the performance of autonomous vehicles primarily depends on the training dataset, meaning if a hazard that the CNN model is not trained on appears on the roadway, the autonomous vehicle driving model may produce an incorrect steering wheel angle and may cause a crash. A recent study shows that the autonomous vehicle navigation system may fail to navigate safely due to several reasons, such as Radar sensor failure, camera sensor failure, and software failure (Bhavsar et al. 2017). This study addresses the situation where a well-trained driving model may fail due to unexpected hazards that may lead to unsafe navigation, and then explores the use of object detection and semantic segmentation (Yao, Fidler, and Urtasun 2012) for mitigating the navigational problem in this hazardous condition.

The remainder of the chapter is organized as follows. The related work section explores the existing studies on autonomous vehicle navigation, state-of-art DNN-based autonomous vehicle driving models, and the limitations of the traditional DNN-based model. Then the author introduces the method developed in this study for navigating an autonomous vehicle on a roadway with unexpected hazards. Furthermore, the author validate our proposed method using three case studies: (i) a model trained using a dataset that includes hazards but without considering them as separate input features; (ii) a model trained on a dataset that considers hazards as separate input features and uses a distance measurement sensor and image segmentation; (iii) a model trained on a dataset that
considers hazards as separate input features and only uses image segmentation. In the second and third case studies, the author introduces a DNN-based autonomous vehicle driving system to enhance the ability of an autonomous vehicle to navigate safely in a hazardous environment. Then the author presents the experimental setup employed in this study. After that, the author evaluates all the case scenarios and report the results obtained through our experiments, and finally, the author discusses the conclusions and suggest the areas for future work.

**Related work**

This section reviews the previous research on hazard detection, the DNN-based driving systems used in an autonomous vehicle, and the techniques for and the importance of object detection and image segmentation in addition to the limitations of using DNN in autonomous vehicles.

*DNN-based Autonomous Vehicle Driving Model*

DNN-based autonomous vehicle driving systems are rapidly evolving (Bojarski et al. 2016)(Pomerleau 1989). Not only software companies such as Waymo (Google) Uber, and Lyft are using the DNN-based systems for autonomous vehicles, but many car companies such as Tesla, Volvo, BMW, and Ford are currently working on DNN-based autonomous vehicles driving systems (Zhang et al. 2018). In such systems, sensors like cameras, LIDAR, and Radar provide input to DNN models, such as Convolutional Neural Network (CNN)(Krizhevsky, Sutskever, and Hinton 2012) or Recurrent Neural Network
(RNN) (Mnih et al. 2015), which then produce outputs such as steering wheel angle and velocity. For example, the autonomous vehicle architecture developed by NVIDIA, named DAVE-2, uses a CNN model which takes input from a camera and outputs a steering wheel commands for navigation (Bojarski et al. 2016), while Udacity autonomous vehicle driving architectures include both CNN-based (e.g., Autumn) and RNN-based (e.g., Chauffeur using CNN and RNN) (“Udacity Self Driving Car,” n.d.). This study used a CNN-based driving model similar to DAVE-2 as it is the fundamental base of DNN-based autonomous vehicle systems.

DNN-based autonomous vehicle driving systems, which are intrinsically software systems, can be error-prone and cause severe consequences if they do not function as intended. Several studies have shown the vulnerabilities of the existing DNN models (Carlini and Wagner 2017)(Papernot, McDaniel, et al. 2016)(Athalye, Carlini, and Wagner 2018)(Papernot, Mcdaniel, et al. 2016). For example, DNN-based image classification can be exploited by adding a small perturbation to an input image such that the DNN model misclassifies it as another category, a vulnerability recently confirmed by (Eykholt et al. 2017), which found that attackers can physically modify objects using a low-cost technique to cause classification errors in DNN-based vision systems. These perturbations can be introduced under widely varying distances, angles, and resolutions. For example, in (Eykholt et al. 2017) perturbations caused a DNN model to interpret a subtly modified physical stop sign as a speed limit of 45 mph sign. Similarly, the debris or roadblocks on the road can also compromise the autonomous vehicle driving system by producing incorrect steering wheel angles, potentially causing a fatal collision. These limitations
prompted this study to evaluate the impact of unexpected hazardous environments on a DNN-based autonomous vehicle driving system.

**Autonomous Vehicle Dataset**

Data are an important part of deep learning-based systems, and this study requires a dataset that supports (i) end-to-end driving systems (input: image; output: steering wheel angle), (ii) image segmentation, and (iii) hazard detection. To find an appropriate one, the author explores various existing datasets used by the autonomous vehicle community. The closest dataset is provided by Udacity, which supports end-to-end data and image segmentation, but it does not provide the ground truth for hazards in the drivable lane (“Udacity Self Driving Car,” n.d.). KTI (Geiger et al. 2013) and Cityscape (Cordts et al. 2016) datasets also do not support hazard detection as ground truth data. The dataset matching our requirements the closest is the Lost and Found dataset (Pinggera et al. 2016), which contains the image as the input, and the yaw rate (angular velocity), but not the steering wheel angle required by this study, as an output. Since existing datasets do not fully meet our needs, after careful consideration, the author created our dataset using simulation as described in the experimental setup section.

**DNN-based Object Detection and Segmentation**

Object detection and classification are core components of autonomous driving. By detecting and classifying the objects, the autonomous vehicle controller determines safe navigation for both path planning and route planning. If an autonomous vehicle is not able to detect unexpected hazards on the road, it will not be able to navigate safely, perhaps resulting in a crash. However, detecting these objects or hazards is a challenging task.
While various sensors, such as Radar and LIDAR, can be used for accurate distance and velocity measurement, these sensors are relatively costly than camera sensors (Pinggera et al. 2016). Considering these limitations, vision-based sensors, such as cameras, are being used on autonomous vehicles for the navigational task. With the recent development of DNNs, DNN-based object detection and semantic segmentation can be applied to detect these roadway hazards, making navigation of autonomous vehicles safer.

Semantic segmentation is a technology that has been widely used in the computer vision area to divide an unknown image into different parts (Guo et al. 2018), can be applied to an image containing unknown objects. This technology is effective in providing the scenario depicted by an image, allowing the DNN to capture additional information about the dataset during training. There are three major types of semantic segmentation technologies: Region-based semantic segmentation (Caesar, Uijlings, and Ferrari 2016)(Girshick et al. 2014), Fully Convolutional Network (FCN)-based semantic segmentation (Long, Shelhamer, and Darrell 2015)(Eigen and Fergus 2015)(Liu, Guo, and Lew 2017) and Weakly-Supervised semantic segmentation (Dai, He, and Sun 2015)(Papandreou et al. 2015)(Khoreva et al. 2017). The region-based semantic segmentation provides segmentation based on the results of object detection, meaning it can be developed on any CNN model. The FCN-based semantic segmentation segments each pixel of the image, meaning it does not require extracting regions of the image and, thus, can be applied to arbitrary sizes of images. The weakly supervised semantic segmentation technology, which was developed to reduce the labeling cost of a large dataset (Khoreva et al. 2017), achieves semantic segmentation by exploiting annotated
bounding boxes or image-level labels. While recent studies show that segmentation-based navigation can improve navigational performance (Eraqi, Moustafa, and Honer 2017)(Teichmann et al. 2016)(Siam et al. 2018), none considers the navigation of autonomous vehicles in hazardous environments. Thus, by leveraging these DNN-based models, the author can detect hazards and then extract their semantic information from images obtained from the camera sensor of an autonomous vehicle. The approach adopted in this study uses an FCN-based model as one such network is relatively small, yet the network yields fast results (Long, Shelhamer, and Darrell 2015). To the best of our knowledge, this is the first work that develops a DNN-based autonomous vehicle driving system focusing on unexpected roadway hazardous environments.

**Method**

In this section, the author describe our approach for developing a safer autonomous vehicle driving system in a hazardous environment. This study uses DNN-based object detection and segmentation to create a corrected image, which is subsequently used by the autonomous vehicle driving system to predict the steering wheel angle. As presented in Figure 1. 1, the author develops a DNN-based autonomous vehicle driving system, which comprises three DNN models. The first one is the DNN-based hazard detection and segmentation model, which detects the hazard and creates a segmented image. The second model is the hazard analysis and avoidance model, which fuses the segmented image with the original input image from the dashboard camera to make the autonomous vehicle driving model aware of the unexpected roadway hazards. This model then analyzes the
hazard and determines if the hazard should be ignored or considered as a threat for a potential crash using a threat factor \((T_f)\). The third model is the DNN-based autonomous vehicle driving model, which takes the fused image with hazard information and produces the steering wheel angle required to navigate the vehicle safely in an unexpected hazardous environment. The author provided a detailed description of these three models in the following subsections.

**Figure 1. 1 DNN-based autonomous vehicle driving system in an unexpected hazardous environment** (Islam et al. 2019).

*DNN-based hazard detection and segmentation model*

For hazard detection and image segmentation, this study uses an FCN, which is a DNN-based image object detection and segmentation model (Long, Shelhamer, and Darrell 2015). Figure 1.2 shows the structure of the FCN network used in our study. It takes an input of image size 400x600x3 and outputs a segmented image of the same size. The author uses a pre-trained network with a weight of VGGNet (Simonyan and Zisserman 2014), which is a deep convolutional network for large-scale image recognition, and then the
author re-trained the model with our training dataset to classify the hazard and perform image segmentation.

Figure 1. 2 FCN-based object detection and image segmentation model used in this study (Islam et al. 2019).

Hazard analysis and avoidance model

As shown in Figure 1. 1, the image captured from the center dashboard camera first goes to the hazard detection and segmentation model, which provides an output of the detected object in addition to a segmented image. Then, this output is combined with the original image in the hazard analysis and avoidance model. In this study, the author developed a hazard analysis and avoidance model based on the following equation:

\[ I = (1 - T_f) \times I_{\text{original}} + T_f \times I_{\text{segmented}} \]

where \( I \) is the image used to predict the autonomous vehicle driving model; \( I_{\text{original}} \) is the data from the center dashboard camera of the vehicle; \( I_{\text{segmented}} \) is the segmented image of \( I_{\text{original}} \) containing the hazardous object detected and segmented; and \( T_f \) is the threat value of the detected hazardous object or physical-world threat object. This threat value depends on the position of a detected object on a driving lane. If the object
is dangerous to the autonomous vehicle, it will have a high threat factor, while a negligible threat object will have a lower threat factor. This threat value depends on the longitudinal and latitudinal distance from the autonomous vehicle. Depending on the hazardous object localization technique, the author has used two procedures to determine the threat value: (ii) Procedure 1 - threat value determination using a distance measurement sensor (e.g., Radar); and (ii) Procedure 2 – threat value determination using image segmentation.

Procedure 1 - threat value determination using a distance measurement sensor

According to the first procedure, the author measures the longitudinal distance \(l_x\), and latitudinal distance \(l_y\) of hazardous objects from the vehicle using a distance measurement sensor. If the vehicle is moving forward (longitudinal movement) or steering towards (latitudinal movement) the hazard the value of \(l_x\) and \(l_y\) decreases, respectively, and hence the hazard poses a higher threat of colliding with the vehicle. The author considers the hazard as a threat to the vehicle if the hazard is within the longitudinal distance, \(l_{x,max}\) and latitudinal distance, \(l_{y,max}\). In our study, the author uses the Radar sensor to measure the longitudinal distance and the latitudinal distance, and the author measures the threat value using the following equations:

\[
T = \sqrt{\left(\frac{l_{x,max} - l_x}{l_{x,max}}\right)^2 + \left(\frac{l_{y,max} - l_y}{l_{y,max}}\right)^2}.
\]

\[
T_f = \begin{cases} 
\frac{T - T_{min}}{T_{max} - T_{min}} & \text{if } l_y \leq l_{y,max} \text{ and } l_x \leq l_{x,max} \\
0 & \text{if } l_y > l_{y,max} \text{ or } l_x > l_{x,max} 
\end{cases}
\]
where, $T_f$ is the threat value corresponding to the hazardous object; $l_x$ and $l_y$ are the longitudinal distance and latitudinal distance in centimeters (cm) to the detected hazard from the vehicle, respectively; $l_{x,\text{max}}$ and $l_{y,\text{max}}$ is the maximum longitudinal distance and maximum latitudinal distance, correspondingly, to consider the hazard as a threat; and $T$ is the threat value calculated from the longitudinal and the latitudinal distance. Then, the value of $T$ is normalized using the Min-Max normalization technique to obtain a value between 0 to +1 to determine the final threat value, $T_f$ (Suarez-Alvarez et al. 2012). In our experiment, the author has selected $l_{x,\text{max}}$ as 6000cm as this is the Radar’s maximum range of finding an object in our experimental setup, and $l_{y,\text{max}}$ is selected as 370 cm, which is the standard lane width of a roadway. The author can visualize the relationship between the threat value, and longitudinal and latitudinal distance in Figure 1.3.
Figure 1. 3 Heatmap for the threat value based on the longitudinal and latitudinal distance of a hazardous object using Radar sensor data (Islam et al. 2019).

Procedure 2 – threat value determination using image segmentation

In this procedure, instead of using a Radar sensor, the author used the segmented image to calculate the threat value. In this way, the author can eliminate the use of any sensor data besides the camera video feed. After the image segmentation, the author gets the image coordinates \((x, y)\) of the hazard. As the camera is located at the center dashboard of the vehicle facing the front roadway, the author measures the relative distance of the hazardous object in the image of size \((h, w)\), from the bottom center pixel, \((h, \frac{w}{2})\) to quantify the threat. The author calculates the threat based on the location of the hazard in the segmented image using the following equation:

\[
T_f = 1 - \sqrt{\frac{(x - h)^2 + \left(y - \frac{w}{2}\right)^2}{h^2 + \left(\frac{w}{2}\right)^2}}
\]

where, \(T_f\) is the threat value corresponding to the hazardous object located in the segmented image at location \((x, y)\) pixels, where \((x, y)\) is the pixel value closest to the bottom center pixel, \((h, \frac{w}{2})\), of the image. The value of \(h\) and \(w\) indicates the height and width of the image, respectively. As the camera of the vehicle is located at the center of the vehicle facing the front roadway, the author subtracts \(h\) and \(\frac{w}{2}\) values from the \(x\) and \(y\) values, respectively, to obtain the longitudinal and latitudinal distance of the hazard relative to the front center of the vehicle. As the author described the equation above, the author calculates the threat value. The author can visualize the threat value in Figure 1. 4, where the threat value decreases as the object moves from the center bottom pixel of the image.
Figure 1. 4 Heatmap for the threat value based on the location of hazard using pixel value from the segmented image (Islam et al. 2019).

**DNN-based autonomous vehicle driving model**

In our study, the author have implemented an autonomous vehicle driving model similar to DAVE-2, an end-to-end autonomous vehicle driving model (Bojarski et al. 2016). As shown in Figure 1. 5, the network receives an input image of 400x600x3 pixels and produces a steering wheel angle as an output. This network includes one lambda layer, one normalization layer, five convolution layers (Conv2D), and four fully connected (FC) layers. The author has used a 5x5 kernel (i.e., filters) and 2x2 stride (i.e., the increment of kernel movement) in the first 3 Conv2D layers, and a 1x1 stride and a 3x3 kernel in the last two Conv2D layers. The entire network contains 7,970,619 trainable parameters.
Figure 1. 5 CNN-based end-to-end autonomous vehicle driving model used in this study (Islam et al. 2019).

The author trains our driving model of an autonomous vehicle from the output of the hazard analysis and avoidance model followed by the deployment to test the performance. After the training, our trained autonomous vehicle driving model is aware of hazardous objects on the roadway and produces a steering wheel angle to navigate safely around the hazard.

**Experimental setup**

In the experimental setup, the author describes the data collection method, data preparation, and data augmentation; and finally, the author trains and validates the DNN-based autonomous vehicle driving model. The steps of our experiment setup are as follows:

*Data Collection*

For this study, the author has used the robotics simulation platform Webots (Cyberbotics Ltd. 2013) to create the roadway environment with hazardous objects and to
collect the data including the driving attributes of the camera image, timestamp, location, vehicle speed, and steering wheel angle. The following subsections describe the collection procedure of the dataset.

Roadway Environment Setup

The roadway built in the simulation consists of two lanes in each direction and 1663m in length with 16 curves (having 45 degrees to 90 degrees radius of curvature) and two intersections as shown in Figure 1. 6. Six additional non-autonomous vehicles are placed randomly on the roadway. The hazardous debris, which includes five objects: rocks, wooden boxes, oil barrels, wooden pallets, and sections of pipe are created in Webots (Cyberbotics Ltd. 2013) and placed randomly on the roadway as shown in Figure 1. 6.

![Roadway environment](image1.png) ![Hazards in the roadway](image2.png)

**Figure 1. 6 Roadway environment setup for an autonomous vehicle with hazardous objects** (Islam et al. 2019).
Autonomous Vehicle Setup

For collecting the data, the autonomous vehicle is equipped with three dashboard cameras, a front, left and right camera (as shown in Figure 1.7), and a Radar sensor. The data collected using these cameras are used to train the end-to-end autonomous vehicle driving model. For example, as seen in Figure 1.8, the images collected by the left and right camera differ from the center camera. After training, the autonomous vehicle uses only a single front camera to navigate through the roadway, similar to the DAVE-2 system (Bojarski et al. 2016). In our developed driving model, the author used the Delphi ESR Radar sensor, which is commercially used in the existing autonomous vehicles (AutonomouStuff 2013). The author used the medium-range mode configurations (horizontal field of view of 90 degrees and a maximum range of 6000 cm) of the Radar sensor in our autonomous vehicle (“Webots Documentation: Radar Sensors” n.d.). The author has also equipped the vehicle with three other Radar sensors in three directions (left, right, and back side) for monitoring the near-by traffic condition and vehicles. These Radars sensors are also configured in the medium range mode.

Figure 1.7 Camera placements in the autonomous vehicle (left, center, and right cameras) (Islam et al. 2019).
Data Preparation

After collecting the data, the author prepares the image dataset for training the end-to-end driving model by normalizing and resizing. As shown in Figure 1. 9, the steering wheel angle output is normalized between the values of -0.5 and +0.5, where a positive value indicates the steering to the right, and a negative value represents steering to the left using linear transformation following this equation:

$$
\theta_{normalized} = -0.5 + \max\left(0, \min\left(1.0, \frac{\theta_{raw} - \theta_{min}}{\theta_{max} - \theta_{min}}\right)\right)
$$

where, $\theta_{normalized}$ is the normalized steering angle between -0.5 and +0.5; $\theta_{raw}$ is the actual steering wheel angle (in radians) measured from the vehicle; $\theta_{max}$ and $\theta_{min}$ are the maximum and minimum steering wheel angle, respectively. The author also normalize the input images for training, which is necessary to improve the DNN model performance (Zha et al. 2015). Normalization is also done on the input images. The red, green, and blue (RGB) channel values of the input images are normalized between the values of -1.0 and +1.0, and their top 200 pixels are cropped using a Lambda layer (as shown Figure 1. 10) as the top portion of the image is not necessary to predict the steering wheel angle and
doing so does not impact the steering wheel angle output of the driving model. For all data collected, the author used an online image annotation tool, LabelMe (Russell et al. 2008), for labeling the hazardous object and segmented image. Using this tool, the author has created the ground truth data for training the image segmentation model for detecting and segmenting the hazards in an image.

![Graph showing normalized steering wheel angle output](image1.png)

**Figure 1. 9** Example of a normalized steering wheel angle plot from the training dataset (Islam et al. 2019).

![Original and cropped image examples](image2.png)

(a) Original image  
(b) Cropped image

**Figure 1. 10** Example of an original and cropped image in the training dataset (Islam et al. 2019).
Data Augmentation

To obtain satisfactory performance from the driving model, it is necessary to train the model on multiple training datasets. Using the techniques of data augmentation, the author created additional data from the existing data through affine transformation (Tian et al. 2017), specifically random rotation, random brightness change, and horizontal flipping of the images, to double the size of the dataset as shown in Table 1.1. From our first simulation, the author has collected 1390 images in total, and the author has split the image dataset in training (i.e., 1112 images) and validation dataset (i.e., 278 images) as shown in column 2 of Table 1.1. Then the author has doubled the dataset size (i.e., 2780 images) using data augmentation as presented in column 3 of Table 1.1. Among these 2780 images, 2224 images are used for training, and the remaining 556 images are used for validation. Among the 2224 images used for training, 468 images contained hazards. Furthermore, the author has collected 104 images from a second simulation where all the images contained hazards. These 104 images are used to evaluate or test the driving model performance.

Table 1.1 Dataset description

<table>
<thead>
<tr>
<th>Dataset type</th>
<th>Collected dataset size</th>
<th>Dataset size after data augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dataset size</td>
<td>1390</td>
<td>2780</td>
</tr>
<tr>
<td>Training dataset size</td>
<td>1112</td>
<td>2224</td>
</tr>
<tr>
<td>Validation dataset size</td>
<td>278</td>
<td>556</td>
</tr>
<tr>
<td>Testing dataset size (all containing hazard)</td>
<td>52</td>
<td>104</td>
</tr>
</tbody>
</table>
Model Training and Validation

After the development of the end-to-end autonomous vehicle driving model, the author trains it using the augmented dataset. This dataset is divided into two, 80% in a training set (2224 images as per Table 1. 1) and the remaining 20% in a validation set (556 images as per Table 1). The author then trains three models for our evaluation:

**Case 1:** A model trained on a dataset that includes hazards but without considering them as a separate input feature.

**Case 2:** A model trained on a dataset that considers hazards as separate input features and uses a distance measurement sensor and image segmentation. In this case, the threat value is determined using a distance measurement sensor (Radar in our case), following Procedure 1 as described in the method section.

**Case 3:** A model trained on a dataset that considers hazards as separate input features and uses image segmentation. In this case, the threat value is determined using the image segmentation, following Procedure 2 as described in the method section.

For the training of the autonomous vehicle driving model, the author used the Adam optimizer that can change the learning rate dynamically (Konur 2015). The mean square error based loss function, a dropout rate of 0.5 in the last four FC layers, and L2 regularization are used to reduce overfitting and under-fitting and to minimize training error (Baldi and Sadowski 2013). The author used model checkpoints to stop the training when the validation loss is not decreasing over time (“Keras,” n.d.). Figure 1. 11 shows the performance of the model training for Case 1, where the training is stopped after 14 epochs.
because the model does not exhibit much improvement after 11 epochs. The author observes no overfitting or under-fitting during the training. In Case 2, the model stopped training after 16 epochs, and in Case 3, the model stopped the training after 15 epochs.

![Training and validation performance graphs for Case 1, Case 2, and Case 3](image_url)

**Figure 1.** Training and validation performance of the end-to-end driving model for Case 1, Case 2, and Case 3 on the training and validation dataset (Islam et al. 2019).

**Analysis results**

After training and validating the model using the dataset from the first simulation, the author evaluates the trained end-to-end autonomous vehicle driving model using the test dataset of 104 images (as depicted in Table 1. 1). The author created this dataset of 104
images from a second simulation where all debris are placed in the middle of the driving lane, and the author measured the predicted steering wheel angle for each test image. In this second simulation, first, the author creates the ground truth by manually driving the vehicle on the roadway. Then the author deploys the trained end-to-end autonomous vehicle driving model for Case 1, Case 2, and Case 3. The author then analyzes the performance of the model for each case using the following quantitative measures: root means square error (RMSE) and mean absolute error (MAE), and a qualitative measure through visualization.

**Quantitative Results of Model Performance**

The quantitative results include the RMSE and the MAE, which are measured by comparing the predicted steering wheel angle with the actual steering wheel angle (i.e., ground truth data). The author defines the RMSE and MAE as follows:

\[
RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (G_i - P_i)^2}
\]

\[
MAE = \frac{1}{N} \sum_{i=1}^{N} |G_i - P_i|
\]

where N is the total number of images in the testing dataset; and \(G_i\) and \(P_i\) are the ground truth and predicted steering wheel angle, respectively, for the \(i^{th}\) image of the testing dataset. As shown in Figure 1.12, both the RMSE and MAE are higher for Case 1 than Case 2 and Case 3. A lower RMSE and MAE indicate that the predicted steering wheel
angle is closely following the actual steering wheel angle or ground truth data related to steering wheel angle.

![Graph showing error measurements for different cases.](image)

**Figure 1. 12 Error measurement on the testing dataset** (Islam et al. 2019).

The author measured the steering wheel angle prediction accuracy and improvement of Case 2 and Case 3, over Case 1. By comparing $RMSE$ of Case 2 and Case 3 with $RMSE_{case1}$, the author calculates the steering wheel angle prediction improvement based on the equation below:

$$\text{Percentage of improvement} = \left( \frac{|RMSE - RMSE_{case1}|}{RMSE_{case1}} \times 100 \right) \%$$

Based on our experiment, the author found a 21% improvement in the steering wheel angle prediction of Case 2 over Case 1, and 18% improvement in the steering wheel angle prediction of Case 3 over Case 1.
angle prediction of Case 3 over Case 1. The results suggest that both Case 2 and Case 3 improve autonomous vehicle navigation to avoid an unexpected hazard on the roadway.

**Qualitative Results for Driving Direction**

Figure 1. 13 shows the qualitative results of our study on the autonomous vehicle driving direction. To obtain the qualitative measurement, the author transforms the steering wheel angle (-0.5 to +0.5) into a driving direction angle (-25 degrees to +25 degrees) using a linear transformation. In Webots, the steering wheel angle follows the Ackermann geometry, representing a linear relationship between steering wheel angle and driving direction (Cyberbotics Ltd. 2013)(Mitchell, Staniforth, and Scott 2006). The prediction accuracy can be presented qualitatively by observing the driving direction angle or angle of movement of the autonomous vehicle. For example, Figure 1. 13 shows that the continuous steering wheel output of data from the time step of 64000 milliseconds (ms) to 72000ms window for ground truth, Case 1, Case 2, and Case 3. In the presence of a hazard on the roadway, the autonomous vehicle driving model is producing the output for maneuvering the autonomous vehicle. According to Figure 1. 13, the autonomous vehicle is moving towards the left for each case. For example, in Case 1, at time step 66000ms, the predicted driving direction is +5.2 degrees, causing the car to move closer to the hazard (represented here as a box) compared to Case 2 and Case 3. However, in Case 2 and Case 3, the predicted driving direction is +11.7 degree and +9.28 degree, respectively, which is a value closer to the ground truth than in Case 1. Overall, the qualitative results indicate better accuracy prediction for Case 2 and Case 3 than for Case 1.
Figure 1. 13 Qualitative results of ground truth, Case 1, Case 2, and Case 3 of the driving direction (Islam et al. 2019).

Quantitative Results for Driving Direction

Following the Frenet coordinate system, the author has performed quantitative analyses of hazard avoidance. In a Frenet coordinate system, the longitudinal movement and latitudinal movement are represented in the x-axis and y-axis, respectively (Houenou et al. 2013). Instead of following the Frenet coordinate system for the performance evaluation, the author plotted the time step in the x-axis and latitudinal movement in the y-
axis (see Figure 1.14) to show the deviation of latitudinal movement of an autonomous vehicle and how the vehicle avoids a hazardous object for different cases (as described in the ‘Model Training and Validation’ subsection) over the time. The author analyzed the trajectory of the autonomous vehicle and calculate the RMSE between the vehicle trajectory of each case and the ground truth. In Figure 1.14, the author presents the autonomous vehicle trajectories for all three cases from the time step 62000ms to 72000ms to show how accurately the vehicle following the ground truth trajectory data for each case to avoid the hazardous object. For Case 2, the vehicle trajectory produced from the autonomous vehicle driving systems is closely following the ground truth vehicle trajectory compared to Case 1 and Case 3. However, in all cases, i.e., Case 1, Case 2, and Case 3, the vehicle can avoid the hazard (Figure 1.14). In Case 1, the RMSE value was 0.52. On the other hand, the RMSE values for Case 2 and Case 3 are 0.07 and 0.23, respectively. The author performed a statistical significance test (pairwise t-test) between the ground truth and each case separately at a 95% confidence interval. The author finds that Case 1 is significantly different from the ground truth at a 95% confidence interval. However, Case 2, which uses both image segmentation and a distance measurement sensor, and Case 3, which only uses the segmented image, are not significantly different from the ground truth. Thus, based on the statistical analyses of Case 2 and Case 3, the author achieves the same level of performance using image segmentation, and not using any additional distance measurement sensor, i.e., Radar.
Figure 1. 14 Trajectory of the autonomous vehicle for ground truth, Case 1, Case 2, and Case 3 (Islam et al. 2019).

Chapter Conclusions

Detecting unexpected hazards on a roadway is a crucial task for the safe operation of an autonomous vehicle. In this work, the author developed and evaluated a DNN-based driving system for autonomous vehicles in an unexpected hazardous roadway environment. First, the author detects the hazard, and then using semantic segmentation, the author extracts the hazard information and perform data fusion to improve the navigation of an autonomous vehicle. This study makes the following contributions to the current body of research: (i) the author evaluates the effect of the hazardous roadway environment on the DNN-based driving system of an autonomous vehicle; (ii) the author develops a DNN-based driving system for autonomous driving that can address an unexpected hazardous
roadway environment and can navigate the autonomous vehicle safely through this environment. More specifically, the author explored the object detection and semantic segmentation based deep learning models to address an unsafe navigational problem; (iii) the author contributes a new dataset that can be used by the autonomous vehicle community to improve the driving model in unexpected hazardous roadway environment. Based on the analysis result, the author concludes that our method improved the safety of the autonomous vehicle by 21% in terms of avoiding hazards, compared to a vision-based navigation system of autonomous vehicles having no hazard detection and segmentation as separate input features. Future work will include fusing the temporal and spatial information into the DNN-based model, potentially further improving the safety of autonomous vehicles operating in an unexpected hazardous roadway environment.
CHAPTER TWO

DEVELOPMENT AND PERFORMANCE EVALUATION OF A CONNECTED VEHICLE APPLICATION DEVELOPMENT PLATFORM (CVDEP)

Introduction

Connected vehicle (CV) application developers need a development platform to build, test, and debug real-world CV applications, such as safety, mobility, and environmental applications, in edge-centric cyber-physical systems. Our objective is to develop and evaluate a scalable and secure CV application development platform (CVDeP) that enables application developers to build, test, and debug CV applications in real-time while meeting the functional requirements of any CV applications. The author evaluated the efficacy of the CVDeP using two types of CV applications (one safety and one mobility application) and validated them through field experiments at the South Carolina Connected Vehicle Testbed (SC-CVT). Our analyses show that the CVDeP satisfies the functional requirements in terms of latency and throughput of a CV application while maintaining the scalability and security of the platform and applications (Islam et al. 2020).

The emerging connected vehicle (CV) environment consists of different components, such as vehicle onboard units (OBUs), and roadside units (RSUs), which are capable of exchanging data with each other as well as communicating with personal devices (e.g., cell phone), sensors (e.g., camera sensors), and traffic management centers (TMCs) (Sotelo et al. 2012). With integrated computing and control capabilities, these connected physical components communicate with each other to form a cyber-physical
The architecture reference for cooperative and intelligent transportation (ARC-IT), which has been developed with the sponsorship of the U.S. Department of Transportation (USDOT), has listed the functional requirements and provided the implementation guidelines of over a hundred CV applications for safety, mobility, and environmental benefits (ARC-IT 2019). For example, “vehicle data for traffic operations (2)” is a CV application, which uses CV data obtained from vehicle OBUs to support roadway traffic operations. To develop such CV applications for an edge-centric CPS, developers need a dedicated platform where they can build, test, and debug CV applications. The operational data environment (ODE) system, which is being developed by Intelligent Transportation Systems Joint Program Office (ITS JPO n.d.), is a real-time data collection and distribution software system that collects, processes, and distributes data to different components of the CV environment, such as CVs themselves, personal mobile devices, infrastructure components (e.g., traffic signal) and sensors (e.g., camera and environmental sensor). Although a user can stream CV data through the ODE platform in real-time for developing a CV application, it does not provide a platform for the application developers to build, test, and debug CV applications. Thus, it is critical to develop an application development platform and evaluate the platform in terms of latency and throughput to satisfy the temporal and spatial requirements of CV applications (Du et al. 2018).

Considering a large-scale deployment of connected vehicle CPS, the concept of edge computing is introduced as the underlying computing approach (Rayamajhi et al. 2017). Edge computing has the potential benefits for enabling reduced communication
latency and increased scalability. Such benefits are a result of bringing resources, such as storage, and computational resources, closer to the edge (Lopez et al. 2015) (Grethe authors et al. 2017). In an edge-centric CPS, the resources for communication, computation, control, and storage are placed at different edge layers (e.g., a mobile edge as a vehicle, a fixed edge as a roadside infrastructure, and a system edge as a backend server or TMC) in a CV environment (Rayamajhi et al. 2017). Therefore, a CV application can be divided into sub-applications where sub-applications of a CV application run in different edge layers depending on the requirements of an application.

Major challenges for developing a CV application development platform for an edge-centric CPS are to (a) collect, process, and distribute data while running multiple CV applications concurrently in real-time in different edge layers; and (b) provide the scalability and security of the platform and applications. The objective of this study is to develop and evaluate a scalable and secure CV application development platform that handles real-time data from CVs in an edge-centric CPS and can satisfy the requirements imposed by CV applications. This platform, which the author calls ‘connected vehicle application development platform (CVDeP)’ has been designed to hide the underlying low-level software, hardware, and associated details. An application development graphical user interface provides the application developers easy and secure access to the edge devices. The access control and credential management module in the application development platform prevents unwanted access to the edge devices, which provides platform security. In addition, the application security module prevents malicious operations or activities propagated through an application in an edge-centric CPS. In this
study, a policy-based security system is utilized to provide application security against cyberattacks. However, developing security policies for detecting different types of cyberattacks and identifying related countermeasures are not the focus of this study.

The author conducted experiments to evaluate the efficacy of the CVDeP using a safety application (i.e., “forward collision warning (FCW)” (ARC-IT 2019)) and a mobility application (i.e., vehicle data for traffic operations (ARC-IT 2019)). These applications were developed and evaluated in an emulated environment and later validated in a real-world edge-centric South Carolina Connected Vehicle Testbed (SC-CVT), which is located at Clemson, South Carolina. The FCW application was selected for our experiment, as it is a fundamental application for vehicle-to-vehicle (V2V) safety (8). Similarly, the vehicle data for traffic operations application was selected, because this application supports many other vehicle-to-infrastructure (V2I) safety and mobility applications, such as cooperative adaptive cruise control, incident detection, and implementation of localized roadway traffic operational strategies (e.g., altering signal timing based on traffic flows, freeway speed harmonization, and optimization of ramp metering rates) (ARC-IT 2019). In addition, the efficacy of the CVDeP was presented using two communication-related measures of effectiveness, which are latency and throughput.

**Contribution of the study**

The primary contribution of this study is the development of an architecture for an edge-centric CV application development platform (Islam et al. 2020). In this study, the author systemically developed the architecture of the CVDeP and evaluated and validated
the CVDeP through experiments. In the “Conceptual Development and Implementation of CVDeP” subsection of the “Connected Vehicle Application Development Platform (CVDeP)” section, the author presented the architecture of the application development platform and defined each module of this architecture. The architecture of the development platform supports modular development so that any user can easily include additional modules (e.g., adding an energy optimization module at mobile and fixed edge levels for an eco-driving application) into the development platform if and when needed. Furthermore, the author published the source code of the CVDeP on GitHub, an open-source platform, so that any external users can use it and contribute to expanding its utility of CVDeP by adding more modules (Islam 2019). The CVDeP open-source software will be maintained through a git version-control system.

Related work

To develop the CVDeP that uses real-time CV data, the author reviewed existing work related to the CV applications development requirements, and developer access control and application security.

CV Application Development Requirements

CV applications are bounded by temporal and spatial requirements for providing the desired services (Karagiannis et al. 2011). If CV data are not received within the temporal and spatial threshold as required by a CV application, CV data will not have any efficacy for real-time applications. The Michigan connected vehicle testbed ‘Proof of
concept test report’ categorized CV data by time and spatial contexts (U.S. Department of Transportation (US DOT) 2010), meaning that timestamp information and location information should be included in the CV data.

Application developers may require two kinds of data depending on the application, namely real-time disaggregated data and aggregated data. For example, applications such as incident detection applications require real-time disaggregated data for running and testing of algorithms (Du et al. 2018), thus making it necessary for the platform to provide such data. On the other hand, applications, such as those that provide queue warning after every 5 minutes (Balke, Charara, and Sunkari 2014) may not require the disaggregated data, but aggregated data is sufficient. A CV environment is considered to be one of the largest distributed networks of the near future (Qian and Moayeri 2008). As the size of the network grows (e.g., number of vehicles, sensors, and roadside infrastructures), the demand for data will also increase (Baker et al. 2016). Thus, a platform for the CV application developers needs to be designed in such a way so that it can handle a high demand of data without compromising the quality of service (in terms of temporal and spatial requirements). Thus, in providing the data to the users, the CVDeP needs to meet the application requirement in terms of latency and throughput and must be capable of handling the scalability issue related to the increasing number of connected vehicles, sensors, and roadside infrastructures.

Access Control and Application Security

Security is one of the major concerns in deploying CV applications because of the safety-critical aspect of connected transportation systems (Zarki et al. 2002)(Raw, Kumar,
and Singh 2013). The USDOT partnered with the automotive industry and industry security experts to design and develop a state-of-the-art security framework and presented a security concept called ‘security credential management system (SCMS)’ to provide privacy and integrity to a CV system as well as provide CV application security. The data shared between applications and edge devices need to be secured and the author needs to maintain data confidentiality, integrity, and availability (ARC-IT 2019). One way to protect the data from unwanted user access is to authenticate user information before sharing and streaming data. In SCMS, fixed edges (e.g., a communication device (e.g., RSU) along with a computing device (e.g., general-purpose processor)) will provide a certificate to a CV application, which can be used by the application for exchanging messages (Whyte et al. 2013)(Ahmed-Zaid, F., Bai, F., Bai, S., Basnayake, C., Bellur, B., Brovold, S., Brown, G., Caminiti, L., Cunningham, D., Elzein, H., Hong, K., Ivan, J., Jiang, D., Kenney, J., Krishnan, H., Lovell, J., Maile, M., Masselink, D., McGlohon, E., Mudson, P., Popov et al. 2011). A registration authority (RA) and a certificate authority (CA) were considered for providing the certificates. While an RA verifies the user request and checks the digital signature, a CA issues a new digital certificate or renews a certificate. In our study, the author adopted a security module for access control and credential management following the SCMS. The application security management is adopted based on security policies developed by (Islam et al. 2018). In this study, the author considered the access control and credential management and application security, however, network security is not part of this study.
Connected vehicle application development platform (CVDeP)

Figure 2. 1 presented the conceptual development and implementation, and the evaluation and validation of the CVDeP. In an edge-centric CPS, the CVDeP architecture is developed including an application management platform and an application development graphical user interface for CV application development. The application management platform contains three modules: (i) control platform module; (ii) communication module; and (iii) data warehouse module. The application development graphical interface contains a graphical user interface through which an application developer can develop and deploy any CV application in the edge devices. The control platform module includes four sub-modules in total: (i) access control and credential management; (ii) application security management; (iii) data collection and distribution; and (iv) data broadcasting and receiving. The author evaluated and validated the CVDeP using selected safety and mobility applications in two stages: (i) evaluation in an emulated environment; and (ii) field validation in a real-world edge-centric SC-CVT. The safety application is evaluated using communication and computational latency metrics. On the other hand, the mobile application is evaluated using communication and computational latency along with data transmission throughput (to test the scalability of the platform). Later, the author explained the experimental set-up in the emulated and real-world environment and CV applications for the evaluation of the CVDeP. In the following sections, the author presented the study approach in detail for developing and evaluating the CVDeP.
Figure 2.1 Approach for the CVDeP development, evaluation and validation (Islam et al. 2020).
In an edge-centric CPS, the physical proximity of devices to the data source reduces the wireless communication latency, and a layered architecture increases the scalability (Mashrur Chowdhury et al. 2018). The edge-centric CPS as shown in Figure 2 consists of three edge layers: (i) mobile edge (e.g., on-board sensors and computing device inside a vehicle); (ii) fixed edge (e.g., roadside transportation data infrastructure); and (iii) system edge (e.g., backend server at TMC) (Rayamajhi et al. 2017). This hierarchical cyber-physical system architecture can address complexity and scale issues of CV systems. Participating CVs in our system will act as mobile edges and are equipped with a low latency communication device. Although the author considered DSRC in our study, any low latency communication technology, such as 5G and LTE for Vehicles (LTE-V) can be incorporated in our development platform. A fixed edge includes a general-purpose processor (i.e., application development device) and a dedicated short-range communication (DSRC)-based RSU. A fixed edge can communicate with mobile edges using DSRC and communicate with the system edge using optical fiber or Wi-Fi. A fixed edge can be extended to support a video camera and other sensing devices, such as weather sensors and GPS. A system edge can be a single endpoint in a cloud server. Fixed edges are connected to a system edge through a long-range communication option, such as optical fiber or LTE/Wi-Fi. Mobile edges (edge layer 1) can exchange data with fixed edges (edge layer 2) and system edges (edge layer 3) using DSRC and LTE/Wi-Fi communication, respectively, as shown in Figure 2. 2.
In an edge-centric CPS for CVs, each component generates different types of data.

For example, an OBU installed in a vehicle (i.e., mobile edge) broadcasts basic safety messages (BSMs), which contain a vehicle’s information, such as location, speed, direction, acceleration, and braking status (Kenney 2011). A fixed edge collects data from the OBUs within its communication range, and acts as a primary gateway to transfer data from CVs to the transportation infrastructures (e.g., system edge, which could represent a TMC). For developing a CV application, developers need to interact with all of the edge layers. Edge layers can be accessed through an application development graphical user interface, which provides a way for a CV application developer to interact with the different edges. Figure 2.2 illustrates the architecture of the CVDeP for an edge-centric CPS, which
comprises of application management platform and application development graphical user interface.

Application Management Platform

The application management platform is responsible for the selection of an appropriate communication medium for an application, and data collection, storage, broadcasting, and distribution, while providing the security of the platform by enabling secured access to the edge layers and security of the CV applications. As presented in Figure 2. 2, application developers interact with the application management platform through an application development graphical user interface. The application management platform is a part of each edge layer of the edge-centric CPS. The application management platform is made up of the following modules: (i) control platform module; (ii) data warehouse module; and (iii) communication module. The following subsections describe the conceptual development and implementation of each of the modules in detail.

*Conceptual development of control platform module*

The control platform module of the system edge (edge layer 3) supports three types of sub-modules: (i) access control and credential management; (ii) application security management; and (iii) data collection and distribution. On the other hand, the control platform module of the fixed edge (edge layer 2) supports four types of sub-modules: (i) access control and credential management; (ii) application security management; (iii) data collection and distribution; and (iv) data broadcasting and receiving. However, the control platform module of mobile edge (edge layer 1) includes: (i) access control and credential management; (ii) application security management; and (iii) data broadcasting and
In an edge-centric CPS, edge devices continuously exchange data between different edges. The data broadcasting and receiving module in the mobile edges and fixed edges handles the continuous data exchange between other mobile edges and fixed edges. This module continuously broadcasts and receives messages that can be used to develop CV applications through application development graphical user interface. On the other hand, the data collection and distribution module in fixed edges and system edges are responsible to gather and distribute data to and from mobile edges, fixed edges, and system edge in real-time. After the access control and credential management modules are activated, an authenticated application developer can access, gather and visualize real-time streaming data generated from different edges of an edge-centric CPS. In addition, the application security management module is responsible for monitoring the data flow and securing the application using security policies.

Implementation of control platform module

The control platform module contains the following sub-modules, and what sub-modules are included in each layer varies by whether the edge device is a mobile, fixed or system edge. Implementation overviews of these sub-modules are as follows:

- **Access control and credential management.** The access control and credential management sub-module ensures that only authorized users have access to CVDeP services. A CV application developer is authenticated via a login interface before giving access to the edge-centric CPS testbed components. Permission-based access control is implemented by providing access rights to application-specific data and services (e.g.,
access to the BSMs, access to sensors data, access to the data warehouse) like an android application system where permission are written in a manifest file prior to developers develop an Android application (Felt et al. 2012). On the other hand, the credential management system (CMS) is implemented based on the public key infrastructure (PKI), which takes care of public key exchange that is needed for encrypting and authenticating data using a digital signature. A digital signature is used to verify the authenticity of a message. The CMS is built in such a way that the functionalities of SCMS presented by the USDOT are replicated (Whyte et al. 2013)(Brecht et al. 2018). The author followed the assumptions of the National Highway Traffic Safety Administration (NHTSA) supported connected vehicle pilot program where V2V messages are digitally signed with a digital signature, but not encrypted, and V2I messages are both signed and encrypted (Weil 2017).

- **Application security management.** In order to provide security for any applications, a data consumer and a data producer must be authenticated and complete certificate exchange (data flow 1 (DF1) and (data flow 2 (DF2)) to send any producer generated data and receive any verified producer generated data, respectively (as shown in Figure 2.3). The access control and credential management module is used to authenticate and exchange certificates to secure access (as described in the “access control and credential management” module) to any edge devices. As presented in (Fernandes et al. 2016), the author implemented a flow policy-based application security in the application security management module, which contains trusted API and quarantine submodules. In our study, the author implemented the flow policies using ‘<source, sink>’ tracking
(Fernandes et al. 2016) in which the source is the producer of the data and sink is the intended consumer of that data. The trusted API submodule removes any sensitive information (e.g., drivers identify and vehicle ID of a mobile edge) from the producer generated data (data flow 3 (DF3)). The quarantine submodule will remove any unexpected or malicious data flows between a producer and a consumer that is not listed in the flow policies. Flow policies can be pre-defined or can be changed by an administrator (e.g., a certificate authority) dynamically. Finally, verified data from a producer is passed to its intended consumer (data flow 4 (DF4)).

Figure 2. 3 Implementation of application security management module, and access control and credential management module (Islam et al. 2020).
• **Data collection and distribution.** The data collection and distribution sub-module is the core part of the fixed and system edges of the CVDeP. The author selected Kafka (Kreps, Narkhede, and Rao 2011) as a broker-based data collection and distribution system because of the following efficacies: (i) high throughput; (ii) low latency; (ii) reliability of data delivery and (iv) scalability. In a publish-subscribe based broker-system, such as Kafka, Message queueing telemetry transport (MQTT) and WebSphere, data producers (e.g., mobile edges, fixed edges, connected vehicle applications) produce and publish data to the broker, whereas the data consumers (e.g., fixed edge, connected vehicle applications) subscribe and consume the data available at the broker. By tagging individual data elements with a label based on a topic, producers (e.g., a connected vehicle) can produce data on a particular topic, and consumers (e.g., a CV application) can subscribe and consume the data of that topic. The broker receives data from producers and immediately makes the data available for consumers to consume. As a result, producers and consumers can generate and consume data, respectively, asynchronously and independently reducing the latency and improving reliability.

• **Data broadcasting and receiving.** The data broadcasting and receiving sub-module is developed for mobile edges and fixed edges, where it is responsible for broadcasting BSMs and receiving BSMs from other mobile edges and fixed edges. In our implementation, each mobile edge broadcasts BSMs at a default rate of 10Hz and each BSM contains necessary attributes for safety applications (e.g., position, speed, and direction) (Kenney 2011)(Park and Kim 2012). Additionally, each fixed edge broadcasts
safety warnings (e.g., intersection safety warning) at a rate of 10Hz, which are generated for V2I applications. In addition, each mobile edge and each fixed edge receives BSMs from all other mobile edges and fixed edges within their corresponding communication range.

Conceptual development of data warehouse module
The data warehouse module stores the data generated from different edge devices, roadside sensors, and applications deployed in the fixed and system edge layers. It is a distributed storage system that resides in fixed edges and system edges. The purpose of the data warehouse module is to store and provide the necessary historical data that is needed by the CV application developers and/or edge layers. As a mobile edge is limited by computation power and storage size, the author did not include a data warehouse module in mobile edges. In fixed edges and system edges, the structure of the data warehouse module is such that it can support and store both structured (e.g., GPS data) and unstructured data (e.g., text and images). A structured data has a strict tabular format whose column size and attributes of each entity are defined. Examples of structured data include any data that can be stored in delimited formats, spreadsheets, or SQL tables, whose columns are defined. A semi-structured data includes data whose fields are defined but organized hierarchically. Examples include data stored in extensible markup language (XML) or JavaScript object notation (JSON) formats. Unstructured data, such as pictures, videos, and textual data, do not have any structural organization associated with the data itself.
**Implementation of data warehouse module**

In our implementation, to support structured, semi-structured, as well as unstructured data, the author used MySQL for structured data in a tabular format, and MongoDB for semi-structured and unstructured data in JSON format. The structured, semi-structured, and unstructured data together produces a huge amount of data in terms of volume. Realistically, CV applications do not need to access the raw data in their original format. Thus, a big data engineering infrastructure can be employed to reduce and compress raw data for further direct access by CV applications. In our case, the author used Clemson University’s Cypress cluster for this purpose. Cypress is a Hadoop-based big data cluster and has both Hadoop Distributed File System (“Hadoop,” n.d.) for large-scale data storage and Apache Spark for big data processing (Zaharia et al. 2010).

**Conceptual development of communication module**

The communication module decides the best available communication medium based on the communication latency requirement of an application. Developers will provide the requirements of an application to the communication module through the application development graphical user interface, and then the communication module creates an abstraction layer to characterize communication network attributes of the available communication networks. For example, the communication module could select DSRC, 5G or LTE-V, or any low latency communication medium, from the available communication mediums to satisfy the requirement of safety applications. While the application is running in an edge device, the CVDeP will provide communication metadata (e.g., available communication mediums, such as DSRC, 5G, LTE, LTE-V, and Wi-Fi, and their average, maximum, and minimum transmission latency and throughput) for
evaluating the performance of the application. The decision for selecting a wireless communication medium, by the communication module, will be completed based on the characteristics of available communication mediums and the application requirements set by the application developers.

**Implementation of communication module**

The communication module manages the underlying communication network connectivity in an edge-centric CPS. The communication network services are implemented in the network layer of each edge device to manage the connectivity using the available communication mediums to connect with other edge devices. In our communication module implementation, the discovery or searching of communication mediums and their network characteristics are measured asynchronously. The communication module selects a medium to use for transmitting and receiving data based on the application requirements. The author added a metadata support layer in the communication module to provide metadata to the application developers that can support them to develop their applications. Through this metadata layer, developers will be able to observe the communication attributes, such as signal strength, bandwidth utilization, and data loss. A script running in the CVDeP provides communication attributes to the developers through the application development graphical user interface, and developers can evaluate the performance of an application through these attributes.

**Application Development Graphical User Interface**

Application developers can access the underlying edge devices of the edge-centric CPS using a graphical user interface and can develop and deploy any CV application
directly on the edge-centric CPS. Based on access control rights to the available services (e.g., communication services and data storage service) of the platform and the requirements of a CV application, an application developer can access different types of data (e.g., real-time and historical data) from each layer through an application development graphical user interface. Using this application development graphical user interface, application developers can also request any specific data for a specific application purpose. For example, developers can request historical data from the data warehouse module to predict future roadway traffic conditions. Application development graphical user interface will provide an interactive platform to the developers to build their applications and test these applications by requesting real-time data from both mobile and fixed edges, and historical data from the data warehouse module from both fixed and system edges.
As shown in Figure 2. 4, the application development graphical user interface is divided into four blocks: (i) applications development services block (using this block a developer can connect to the edge devices through an authentication procedure using the accessibility details, such as username and password. After the authentication procedure, developers will be provided with a list of available edge devices (e.g., location, number, and type of edge devices), services (e.g., available communication mediums and their characteristics), and sensors (e.g., GPS, camera) of each edge device.); (ii) applications development block (inside this block, an application developer can implement an application in an edge device using Python or C++); (iii) Applications development tools (using this block, an application developer can develop, deploy, test, and debug an
application in edge devices); and (iv) Applications output and performance measurement block (after deploying an application, developers can observe and save the output and performance data of an application through this block). The application development graphical user interface is developed as a desktop application in C# (C sharp) as illustrated in Figure 2. 4. Currently, the software has been developed for the Windows operating systems as a proof-of-concept.

Experimental setup

This section describes the experimental set-up in an emulated environment as well as a real-world environment to evaluate the efficacy of the CVDeP.

Experimental Setup in Emulated Environment

A developer can develop and evaluate the performance of the developed CV applications in the emulated environment. In this environment, the developer will have dedicated hardware to emulate the real-world edge-centric CPS. As shown in Figure 2. 5, a developer can emulate mobile edges using hardware setup #1 and #2 and fixed edges using hardware setup #3, where system edges are set-up in a dedicated server at Clemson University. Each hardware setup (#1, #2, and #3) consists of one DSRC unit to send and receive the DSRC messages, and a computing device for computation. Hardware setup #1 is used for developing the safety application whereas hardware setup #2 is used for emulating other mobile edges for the safety application. For mobility and environmental applications, only hardware setup #2 can be used for emulating mobile edges. Hardware
setup #3 is used for creating any number of fixed edges where the location of fixed edges is defined by a developer through the application development graphical user interface. A dedicated server located at Clemson University is used for creating system edge instances. In this emulated edge-centric CPS, mobile edges and fixed edges communicate with each other using DSRC, and fixed edges and system edges communicate using the Clemson University communication network, which includes an optical fiber and Wi-Fi connections. In addition, developers can configure the number of edges in each layer as required by an application. To generate the movement data of mobile edges, the movement of the mobile edges is exported from the ‘Simulation of Urban Mobility (SUMO) (“DLR - Institute of Transportation Systems - SUMO – Simulation of Urban MObility” 2017)’, which is a microscopic traffic simulator software, as a SUMO trace file. Using this SUMO trace file, developers can create any roadway environment, and generate any number of emulated vehicles and their corresponding BSMs. A program running in mobile edges reads that trace file and generates BSMs for each vehicle. Then, these BSMs are broadcasted using DSRC to each vehicle. Fixed edges will receive BSMs from mobile edges within their corresponding communication ranges. Developers can access the edges through the CVDeP application development graphical user interface to develop and evaluate the performance of the developed CV application.
The author implemented all the modules of the CVDeP in each layer, as shown in Figure 2.6. Hardware setup #1 and #2 represent the edge layer 1, Hardware setup #3 represents the edge layer 2, and the Server setup represents the edge layer 3 of an edge-centric CPS. The implemented modules of the CVDeP are (i) control platform module, which consists of access control and credential management, application security management, data collection and distribution, and data broadcasting and receiving; (ii) communication module; and (iii) data warehouse module. The control platform module resides in a computing device and is implemented in each hardware setup. However, the data broadcasting and receiving sub-module of the control platform module resides in a computing device, which is a part of each mobile and fixed edges. For the data warehouse module, the author used an external hard disk drive (HDD) for storing data in the fixed edges, and cloud storage for storing data in the system edge. In our case, an application
developer interacts with each hardware through the Clemson University communication network to develop, debug, and test a CV application.

**Figure 2.6** Implementation of CVDeP modules in an emulated edge-centric CPS (Islam et al. 2020).

**Experimental Setup in SC-CVT**

The SC-CVT has three fixed edges, which are deployed along the Perimeter Road in Clemson, South Carolina, and one system edge is deployed as the backend server (Mashrur Chowdhury et al. 2018). The backend server is located at Clemson University and connected to the Clemson University network. Two of the fixed edges are connected...
to the Clemson University network with an optical fiber link and one fixed edge is connected to the Clemson University network with a Wi-Fi link. Each fixed edge has its DSRC radio to communicate with mobile edges. Each mobile edge (primarily OBU on vehicles) is equipped with wireless communication devices. In our case, the author used DSRC-enabled OBU, although, any low latency communication mediums, such as 5G or LTE-V can be used. As per our definition of a mobile edge, a connected vehicle will act as a mobile edge and a vehicle owner will own a commercially available low latency communication device (e.g., DSRC, 5G, or LTE-V enabled communication device) along with a computing device for running an application at the vehicle level. Also, a vehicle owner can install these communication and computing devices to create a mobile edge.

**Evaluation and validation of CVDeP**

For our experiments, the author developed a forward collision warning (FCW) as a safety application and vehicle data for traffic operations as a mobility application (ARC-IT 2019) using the CVDeP. Then, to prove the efficacy of the CVDeP, the FCW and vehicle data for traffic operations applications are evaluated in an emulated environment and the real-world SC-CVT (Mashrur Chowdhury et al. 2018).

**Safety Application**

For our experiment related to safety application, the author selected forward collision warning (FCW) that considers two vehicles moving in the same direction on the same lane in an uncongested urban traffic condition. The FCW application is based on the
study by Xiang et.al. (Xiang, Qin, and Xiang 2014), where the FCW application uses the vehicle kinematics (VK) model for generating collision warnings using DSRC communication. Based on the VK model, the FCW application generates rear-end collision warnings when two vehicles are closer than a defined safe distance. In our study, the following equation is used for implementing an FCW application as suggested by Xiang et.al. (Xiang, Qin, and Xiang 2014).

\[ D_w = \frac{(V_o - V_t)^2}{2 \cdot a} + d \]

where \( D_w \) is the distance threshold for collision warning; \( V_o \) is the preceding vehicle’s speed; and \( V_t \) is the follower vehicle’s speed. The follower vehicle is the vehicle where the FCW application is intended to run; \( d \) is calculated by adding half of the length of the preceding vehicle with the half of the length of the following vehicle, and \( a \) is set to 11.2 ft/s\(^2\) (American Association of State Highway and Transportation Officials (AASHTO) 2011). Given the emulated environment within the CVDeP platform, as shown in Figure 2.5, it is possible to generate complex urban scenarios and develop and evaluate appropriate FCW application corresponding to such scenarios. Using a complex urban scenario, an application developer can develop an FCW application considering different safety constraints within that environment.

**Evaluation Scenarios**

The author created two evaluation scenarios for evaluating the CVDeP as a safety application development platform.
• **Scenario 1:** The preceding vehicle (hardware setup #2 in Figure 2.5), and the follower vehicle (hardware setup #1 in Figure 2.5) is moving in the same direction on the same lane at 20 mph and 30 mph, respectively.

• **Scenario 2:** The preceding vehicle and follower vehicle both are moving at 30 mph and the preceding vehicle stops suddenly.

In both scenarios, the FCW application is deployed in the follower vehicle, and forward-collision warnings are generated based on the comparison between calculated safety distance and the distance between two vehicles using real-time GPS data. To evaluate the performance of the application, the author considered data delivery latency as a measure of effectiveness. In this context, latency is the duration between the time when a BSM is generated by a mobile edge and the time when the application produces an FCW message in the follower vehicle. Here, latency includes network latency, computational latency, and communication medium selection latency.

**Evaluation in Emulated Environment**

The author evaluated the FCW application, using the experimental setup as described in the previous section. The application is developed using the CVDeP, and then the application is tested using two evaluation scenarios. Table 2.1 provides a summary of latency recorded from both evaluation scenarios. For the evaluation of the FCW application in the emulated environment, the author analyzed the BSMs of 200 seconds observation period containing 4000 BSMs from two mobile edges to calculate the maximum, minimum, and average latency. A connected vehicle broadcasts BSMs and receives BSMs from other connected vehicles within its communication range. A CV safety application’s
critical latency requirement represents the maximum acceptable time from generating BSMs by a preceding vehicle to generating an FCW message by a follower vehicle within the preceding vehicle’s communication range. If an FCW message is received by the driver of the follower vehicle within this safety-critical latency requirement, the driver can take action to avoid a collision after receiving a forward collision warning (Qing Xu et al. 2003). In our case, the author selected 200 ms as a maximum safety-critical latency requirement (Mashrur Chowdhury et al. 2018) in which a driver can decelerate at a deceleration rate of 11.2 ft/s² (American Association of State Highway and Transportation Officials (AASHTO) 2011), and avoid the forward collision if the warning message was delivered within 200 ms. Therefore, the maximum end-to-end latency requirement is considered as 200 ms, which will ensure the driver to stop the vehicle in case of a forward collision scenario. In our emulated experimental environment, the author found that the average latency is 18 ms for both evaluation scenario 1 and scenario 2. However, the recorded maximum latencies were 97 milliseconds (ms) and 79 ms, for scenarios 1 and 2, respectively, which are below the safety-critical latency requirement for connected vehicles (i.e., 200ms (Dey et al. 2016)). For the evaluation of the FCW application in the emulated environment, the author analyzed the data of 200s containing 4000 BSMs from two mobile edges to calculate the maximum, minimum, and average latency. In Table 2.1, the author presented the end-to-end latency, which includes communication network latency, computational latency, and communication medium selection latency. The computational latency for running the application is 1.5 ms, which is the same for both evaluation scenarios. In addition, these FCW messages are sent to the mobile edge using the best
available communication medium as decided by the communication module, which takes about 0.5 ms on average to make such a determination. During this communication medium selection process, all communication mediums (LTE, Wi-Fi, and DSRC) were running simultaneously, and the communication module was monitoring these mediums asynchronously and selects the best communication medium for a connected vehicle application following the heterogeneous wireless networking concept for CVs (Dey et al. 2016).

Table 2. 1 Summary of Latency for FCW Application Evaluation

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulated environment</td>
<td>Maximum</td>
<td>97 ms</td>
<td>79 ms</td>
<td>≤ 200 ms</td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>18 ms</td>
<td>18 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>4 ms</td>
<td>4 ms</td>
<td></td>
</tr>
<tr>
<td>SC-CVT</td>
<td>Maximum</td>
<td>115 ms</td>
<td>107 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Average</td>
<td>65 ms</td>
<td>51 ms</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Minimum</td>
<td>4 ms</td>
<td>5 ms</td>
<td></td>
</tr>
</tbody>
</table>

Note: *Scenario #1: The preceding vehicle and follower vehicle are moving in the same direction on the same lane at 20 mph and 30 mph, respectively; and ** Scenario #2:
The preceding vehicle and follower vehicle both are moving at 30 mph and the preceding vehicle stops suddenly.

Field Validation in SC-CVT

For our field evaluation of the FCW application in SC-CVT, the author followed similar speed for the corresponding vehicles for both evaluation scenarios and measured the end-to-end latency for the FCW application. Table 2. 1 provides a summary of end-to-end latency recorded for both evaluation scenarios in the field experiments and an emulated environment. Similar to the evaluation in an emulated environment, the author analyzed the data sample of 200s containing 4000 BSMs from two mobile edges to calculate the maximum, minimum, and average latency. The average end-to-end latency measured is 65 ms and 51 ms for scenarios 1 and 2, respectively. The maximum end-to-end latency recorded for the test is 115 ms and 107 ms for scenarios 1 and 2, respectively, which is below the safety-critical latency requirement (i.e., 200 ms (Dey et al. 2016)). In our field experiment, the author observed a higher latency than the latency measured in the emulated experimental setup because of the surrounding environmental effect or wireless communication propagation loss. In Table 2. 1, the author presented the end-to-end latency which includes the network latency, computational latency, and communication medium selection latency. In both cases (scenarios 1 and 2), the author can validate that the application developed using the CVDeP was able to satisfy the application’s safety-critical latency requirement ($\leq$200ms) in the field experiments.
**Mobility Application**

The author evaluated the CVDeP using vehicle data for traffic operations applications. This application collects CVs’ data (e.g., BSMs) to support traffic operations, such as incident detection and localized traffic operational strategies (ARC-IT 2019). This application is divided into two sub-applications: (i) sub-application 1: collect real-time traffic data from mobile edges; (ii) sub-application 2: collect real-time traffic data from fixed edges. The sub-application 1 runs in each fixed edge and sub-application 2 runs in the system edge.

The author evaluated the scalability of the CVDeP to ensure the CV application requirements are met in terms of latency and throughput. Here, the latency is the time difference between the time of data generation at the edge-centric SC-CVT and the time when the data is received by the users (e.g., CV applications). Data delivery latency requirements for any mobility and environmental applications must be satisfied in order to provide mobility and environmental services. As the CVDeP aims to support different mobility and environmental applications, the author considered 1000 ms as the maximum latency threshold to deliver the data from edge devices to the data consumers (e.g. CV applications) following the recommendations from (Ahmed-Zaid et al. 2011). This 1000 ms will enable the near real-time operation of mobility applications, such as queue warning and traffic rerouting applications. However, if the latency recommendations changes in the future for any CV applications, the CVDeP can still be utilized by selecting appropriate underlying technologies for different communication and computing devices to meet any new requirements. The CVDeP provides a general architecture, which is independent of
specific technologies. Our experiments demonstrate the efficacies of the CVDeP as an application development platform using selected communication and computing technologies. Also, the author needed to ensure a high throughput (i.e., the data transfer rate), which means the high use of the allocated bandwidth. Our platform already satisfied the spatial requirement of the application, as mobile edges will be within the communication range of fixed edges.

Evaluation Scenarios

The author created two different scenarios for evaluating our application development platform by varying the number of fixed edges and the number of mobile edges.

- **Scenario 1**: One system edge and one fixed edge with varying numbers of mobile edges (5, 10, 20, 30, 50, 100, 150, and 200).

- **Scenario 2**: One system edge, varying number of fixed edges (1, 2, and 3), and 200 mobile edges (CVs) for each fixed edge.

For evaluation scenario 2, based on a fixed edge’s communication range, the maximum number of CVs on Perimeter road approaching the intersection is 200 vehicles/hour/lane during a congested traffic condition. For the evaluation in the emulated environment, the author used SUMO to generate the movement data of mobile edges and calibrated the traffic network so that traffic volume data from SUMO simulation matches, within a tolerance level of 5%, with the field-collected data. For both scenarios, the author evaluated the scalability of the application development platform in terms of data delivery latency and throughput.
Evaluation in Emulated Environment

The author implemented a data collection and distribution system (a broker-based system) that is required for the real-time application development platform. The author evaluated the scalability of the CVDeP considering a data collection and distribution system, which is a broker-based system. In addition, the author compared the recorded end-to-end latency with the latency requirement for the selected CV mobility application. As shown in Figure 2. Evaluation of CVDeP for mobility application using application throughput and latency with the increasing number of mobile edge and fixed edge, the throughput of the broker-based system is linearly increasing and reaches a maximum at 5.2 Mbits/s and 8.4 Mbits/sec, respectively. Higher throughput ensures reliable and scalable services. The broker-based system (e.g., Kafka (“Apache Kafka” n.d.) for this experiment) uses an asynchronous mode that can collect and distribute data in memory and send them in batches in a single shot (Kreps, Narkhede, and Rao 2011). Because of this asynchronous mode and sending data in batch, the broker-based system can provide the required throughput. The broker-based system can adapt the throughput requirement by the application as the number of mobile edges and fixed edge increases and thus can handle more data as needed.
Figure 2. 7 Evaluation of CVDeP for mobility application using application throughput and latency (Islam et al. 2020).

The author observed that the CVDeP data collection and distribution system can maintain a lower latency with the increasing number of mobile edges and fixed edges. The increment of latency with the broker-based method is negligible for both scenarios (scenarios 1 and 2). The reason is that the broker-based system uses an intelligent ‘sendfile’ method with zero-copy optimization (i.e., sending the data directly to the consumer without any buffering or copying into memory) (“Apache Kafka” n.d.). Thus, the broker-based system can maintain a lower message delivery latency irrespective of the number of
producers and consumers thus ensuring scalability. In our experiment, the author used the default configuration of a Kafka broker-based system (Kreps, Narkhede, and Rao 2011). However, the configuration (e.g., topic partitions, replication number, and the number of brokers) of Kafka's broker-based system can be configured easily to reduce the latency if the latency is higher than the CV application threshold. In addition, by adding additional data management brokers, as presented by (Du et al. 2018), the CVDeP can be scaled up to receive and share data from additional connected data sources (e.g., personal handheld devices, news media, and weather stations, traffic operators).

Field Validation in SC-CVT

The author evaluated the CVDeP in SC-CVT using five mobile edges (e.g., CVs) in the field experiment. Table 2. 2 Summary of Latency for Mobility Application with Five CVs. Table 2. 2 shows the summary of end-to-end latency when the author developed the application in the CVDeP emulated environment and SC-CVT. The author observed higher latency (maximum, average, and minimum) in the field than in the emulated environment. In the field experiment, the data exchange using DSRC technology between the mobile edges and fixed edges were affected by the environmental inferences, such as trees, roadway slope, and curvature. This causes a higher variation in latency in the field than in the emulated environment. However, the latency observed in the field was still far below the application latency requirement (≤1000ms) for the selected mobility application.
Table 2. Summary of Latency for Mobility Application with Five CVs

<table>
<thead>
<tr>
<th>Evaluation Parameter</th>
<th>End-to-End Latency</th>
<th>Latency requirements for Mobility Application (U.S. Department of Transportation (US DOT) 2010)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Evaluation in Emulated Environment</td>
<td>Validation in SC-CVT</td>
</tr>
<tr>
<td>Maximum</td>
<td>115 ms</td>
<td>267 ms</td>
</tr>
<tr>
<td>Average</td>
<td>65 ms</td>
<td>69 ms</td>
</tr>
<tr>
<td>Minimum</td>
<td>4 ms</td>
<td>6 ms</td>
</tr>
</tbody>
</table>

Chapter Conclusions

CV technology holds the promise of improving the traffic safety and efficiency of roadway traffic operations. In order to materialize CV benefits, the active participation of CV researchers and developers is necessary. This can be hindered due to the lack of real-world application development platforms that use real-world and real-time data to support the CV application development process including testing and debugging. Our research related to the CV application development platform contributes directly by developing a CV application development platform, CVDeP, for an edge-centric CPS. Using the CVDeP, the CV application developers can interact with real-world edge devices, and develop, test, and debug CV safety and mobility applications. From our experiments, it was revealed that the applications developed using the CVDeP were able to satisfy the CV
safety and mobility application latency requirements and maintain the required throughput both for an increasing number of mobile edges and fixed edges. The author showed that the forward collision warning (FCW) application (a safety application) developed using the CVDeP can satisfy the safety-critical latency requirement (under 200 milliseconds for an FCW application). Also, the vehicle data for traffic operations application (a mobility application) developed using the CVDeP with a broker-based system shows about 400 milliseconds of latency with three fixed edges and 600 mobile edges, which is much lower than the latency requirement (under 1000 milliseconds) of mobility applications. This proves the scalability of our CVDeP while satisfying the latency requirement of CV applications for an edge-centric CPS. The author published the source code of the CVDeP is released on the Github platform.

As the CVDeP is being refined further, our follow-up studies of CVDeP includes (i) evaluation of the fault tolerance and resiliency of the platform; (ii) evaluation of multiple applications running simultaneously in multiple system edges, and merging information from diverse data sources of a large roadway network (i.e., data residing at local or city/county level, regional or state level, and/or national level); (iii) incorporation of data from other traditional data sources (e.g., traffic signals, video detectors or loop detectors) and non-traditional data sources (e.g., news media, weather sensors, social networking sites); and (iv) strategy identification to make the system more secure by incorporating different security threat detection and protection mechanisms against different malicious activities including cyber-attacks.
CHAPTER THREE

HYBRID QUANTUM-CLASSICAL NEURAL NETWORK FOR CLOUD-SUPPORTED IN-VEHICLE CYBERATTACK DETECTION

Introduction

A classical computer works with ones and zeros, whereas a quantum computer uses ones, zeros, and superpositions of ones and zeros, which enables quantum computers to perform a vast number of calculations simultaneously compared to classical computers. In a cloud-supported Internet-of-Things (IoT) environment, running a machine learning application in quantum computers is often difficult, due to the existing limitations of the current quantum devices. However, with the combination of quantum-classical neural networks (NN), complex and high-dimensional features can be extracted by the classical NN to a reduced but more informative feature space to be processed by the existing quantum computers. In this study, the author develops a hybrid quantum-classical NN to detect an amplitude shift cyber-attack on an in-vehicle control area network (CAN) dataset. The author shows that using the hybrid quantum-classical NN, it is possible to achieve an attack detection accuracy of 90%, which is higher than a comparable classical NN (61%) alone or quantum NN alone (62%).

The decoherence and mechanical errors in quantum computers can make it harder for the existing quantum computers to learn the underlying data pattern, affecting the performance (Kulkarni, Kulkarni, and Pant 2020). With the recent advancement of near-term quantum processors, it is possible to use a combination of classical and quantum
computers to reduce errors. In a hybrid quantum-classical setup some computations are performed in quantum computers and some computations are performed in classical computers. Such a setup can be used in a cloud-based internet-of-things (IoT) environment, where a control area network (CAN) bus is connected to the cloud using a CAN logger attached to the OBD-II port of a vehicle. The CAN logger provides CAN bus data to the cloud to run multiple IoT applications in the cloud while meeting the delay requirements (e.g., data upload and download delay) of the vehicle’s operation (Figure 3. 1) (Nkenyereye and Jang 2017)(Deng et al. 2020). In this chapter, the hybrid quantum-classical cyberattack detection application will run in the cloud to detect a cyberattack on the in-vehicle CAN bus. The author considers an amplitude shift cyberattack, where an attacker can compromise an electronic control unit (ECU) locally or remotely and can perform an amplitude shift attack on the in-vehicle CAN bus (M. Chowdhury, Islam, and Khan 2019). As the amplitude shift attack changes the data field of a CAN frame randomly, the complex nature of the attack makes it difficult to detect this kind of attack. Studies showed that CAN bus used in existing vehicles do not have sufficient security features (Khan et al. 2020) (Song, Woo, and Kim 2020), and the security can be improved using machine learning techniques. The study by Song et al. shows an accuracy of 99% in detecting denial of service (DoS) attacks. However, their method will not work in the case of an amplitude shift attack, where the amplitude of a feature is shifted (up or down) randomly. The recent study by Khan et al. shows a detection accuracy of 87% on detecting amplitude shift attack using a deep neural network (Khan et al. 2020). To improve the attack detection accuracy, the author combines a quantum machine learning method, more specifically a quantum
neural network, with a classical neural network. By leveraging the advantages of the near-term quantum computers, the study by Farhi & Neven presented a general quantum neural network architecture that was able to classify a handwritten digit dataset (i.e., MNIST) (Broughton et al. 2020)(Farhi and Neven 2018). However, using such a quantum-only approach yields a lower classification accuracy. A more recent study shows the use of a hybrid quantum-classical neural network (NN) approach can achieve a higher classification accuracy (Mari et al. 2020). However, this approach has not been applied in a cloud-based in-vehicle cyberattack detection system. Using a cloud-based hybrid quantum-classical NN, the author overcomes the existing limitations of quantum computers, and develop a quantum computing application for in-vehicle cyberattack detection.

Figure 3. 1 Cloud based In-vehicle Cyberattack Detection System
Cyberattack Detection in IoT Cloud

In our hybrid quantum-classical NN (Figure 3. 2), first, the author preprocess the in-vehicle CAN bus dataset and construct a CAN image dataset (Song, Woo, and Kim 2020). Then the author performs feature extraction using classical convolution neural network (CNN), convert the output from the classical CNN into quantum data, and use the quantum data into a quantum NN to detect an in-vehicle cyberattack.

![Figure 3. 2 Hybrid quantum-classical neural network.](image)

Preparing dataset

The author constructs a $13 \times 13$ CAN image from 13 consecutive CAN frames, where each row represents a single CAN frame and each column represents a data feature. The author considers a $13 \times 13$ CAN image as the author has 13 data features in our dataset (Khan et al. 2020). The constructed CAN image dataset is represented by $D = \{(x_i, y_i)\}_{i=1}^N$, where $x_i$ is a $13 \times 13$ CAN image, with a label $y_i \in \{0, 1\}$ representing no attack and attack image. $N$ is the number of total samples in $D$. The author divides the total samples of, $N=6000$ into 80% training dataset (i.e., 4800 samples) and 20% testing dataset (i.e.,1200 samples).
Feature extraction using classical neural network

As presented in (Song, Woo, and Kim 2020), the author also uses a CNN for extracting the features from a $13 \times 13$ CAN image and produce a $4 \times 4$ reduced image. The feature extraction from CNN can be represented as follows:

$$L_{4x4} = L_{n-1} \circ L_{n-2} \circ L_{n-3} \cdots L_{n1} \circ L_{n0}$$

$$L_i: x_{i-1} \rightarrow x_i = \varphi(W_i x_{i-1} + b_i)$$

Where, $L_{4x4}$ is the output of a CNN, $L_i$ is the $i^{th}$ layer of the CNN; $x_{i-1}$ and $x_i$ are the input and output vectors of $L_i$; $W_i$ is the weight, $b_i$ is a bias vector and $\varphi$ is a nonlinear function. Hyperparameters, such as number of layers $(n)$, $W_i$ and $b_i$ are optimized during the training phase on the training dataset for accurate classification.

Quantum encoding

To perform quantum operations (e.g., unitary operations, such as rotation, and phase flip of qubits) on each $L_{4x4}$ image, the author needs to convert the classical data into quantum data. The author performs quantum basis encoding as follows:

$$|\varphi\rangle = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} |b_i\rangle$$

Where, $|\varphi\rangle$ is an encoded quantum basis, where the basis $|0\rangle$ represents normal data and $|+1\rangle$ represents attack data, $b_i$ is a binary value for each data point of a $L_{4x4}$ image produced using binary thresholding with a value of 0.5. Figure 3.3 shows the output after the data is encoded into quantum binary image data.
Figure 3.3 Quantum encoded binary data.

Classification using quantum neural network

With quantum encoded data, the author trains the parameterized quantum NN (Broughton et al. 2020). The parameterized quantum NN performs unitary operations, such as rotation, phase flip, on qubits and can be represented as follows:

\[ Q = Q_{d-1} \circ Q_{d-2} \cdots Q_1 \circ Q_0 \]

\[ Q_i: |\varphi\rangle \rightarrow y = U(w)|\varphi\rangle \]

Where \( Q \) is a binary output with \{0,1\}, where 0 and 1 represent no attack and attack detected, respectively; \( Q \) has \( d \) number of layers, \( U(w) \) is a unitary operation on \( |\varphi\rangle \) with a weight \( w \), and \( y \) is the output after performing the unitary operation \( U(w) \).

Experimental Results

The author compares the performance of the hybrid quantum-classical NN (Figure 3.2) with the or classical NN alone (Figure 3.4) and quantum NN alone (Figure 3.5). For
a fair comparison, the author selected the equivalent hyperparameters for each type of NN (Table 3. 1).

![Figure 3. 4 Compareable classical neural network.](image)

![Figure 3. 5 Compareable quantum only neural network](image)

Figure 3. 6 shows the attack detection accuracy on the training dataset and testing dataset. For both the training and testing dataset the hybrid quantum-classical NN shows 93% and 90% accuracy, respectively, whereas the quantum-only NN (Farhi and Neven 2018) shows 84%, and 62% accuracy on the training dataset and testing dataset, respectively. With the classical NN (Song, Woo, and Kim 2020), the attack detection accuracy is 59% and 61% on the training and testing dataset, respectively. Here, the classical NN-based feature extractor was able to extract the features and the quantum NN was able to perform more accurate attack detection. The feature map extracted from the classical NNs, CNN in this case, allowed the parameterized quantum NN to explore the neighboring features in an exponentially large linear space, potentially allowing our
hybrid-classical NN to capture the patterns in the dataset (i.e., statistical distributions) more efficiently than classical NN alone and quantum NN alone.

**Table 3.1 Hyperparameter for different neural networks**

<table>
<thead>
<tr>
<th>Hyperparameters</th>
<th>Hybrid Quantum-Classical NN</th>
<th>Quantum-only NN</th>
<th>Classical NN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of qubits</td>
<td>6</td>
<td>6</td>
<td>N/A</td>
</tr>
<tr>
<td>Number of epochs</td>
<td>20</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Number of layers</td>
<td>3</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>Batch size</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Total trainable parameters</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

**Figure 3.6 Cyberattack detection accuracy for different neural networks.**

- **Training dataset**
- **Testing dataset**
Chapter Conclusions

In a cloud-supported IoT environment, a hybrid-classical neural network performs better in detecting an in-vehicle cyberattack compared to a quantum neural network, and a classical neural network, as a hybrid-quantum neural network, can capture the complex pattern of a cyberattack more efficiently.
REFERENCES


Bojarski, Mariusz, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp,


Dey, Kakan Chandra, Anjan Rayamajhi, Mashrur Chowdhury, Parth Bhavsar, and James


