Sensor Degradation Detection Algorithm for Automated Driving Systems

March 2023 | Final Report

SAFE-D
SAFETY THROUGH DISRUPTION

Texas A&M Transportation Institute

San Diego State University
Leadership Starts Here

VIRGINIA TECH TRANSPORTATION INSTITUTE
VIRGINIA TECH

Disclaimer

The contents of this report reflect the views of the authors, who are responsible for the facts and the accuracy of the information presented herein. This document is disseminated in the interest of information exchange. The report is funded, partially or entirely, by a grant from the U.S. Department of Transportation's University Transportation Centers Program. However, the U.S. Government assumes no liability for the contents or use thereof.

TECHNICAL REPORT DOCUMENTATION PAGE

1. Report No. VTTI-00-034	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Sensor Degradation Detection Algorithm for Automated Driving Systems		5. Report Date March 2023 6. Performing Organization Code:
7. Author(s) Jonathan M. Darab Christina J. Witcher		8. Performing Organization Report No. VTTI-00-034
9. Performing Organization Name and Address: Safe-D National UTC Virginia Tech Transportation Institute 3500 Transportation Research Plaza Blacksburg, VA 24061		10. Work Unit No. 11. Contract or Grant No. 69A3551747115/Project VTTI-00-034
12. Sponsoring Agency Name and Address Office of the Secretary of Transportation (OST) U.S. Department of Transportation (US DOT)		13. Type of Report and Period Final Research Report Start: 04/2021 End: 03/2023 14. Sponsoring Agency Code

15. Supplementary Notes
This project was funded by the Safety through Disruption (Safe-D) National University Transportation Center, a grant from the U.S. Department of Transportation – Office of the Assistant Secretary for Research and Technology, University Transportation Centers Program.

16. Abstract
The project developed a sensor degradation detection algorithm for Automated Driving Systems (ADS). Weather, cyberattacks, and sensor malfunction can degrade sensor information, resulting in significant safety issues, such as leading the vehicle off the road or causing a sudden stop in the middle of an intersection. From the Virginia Tech Transportation Institute's (VTTI's) Naturalistic Driving Database (NDD), 100 events related to sensor perception were selected to establish baseline sensor performance. VTTI determined performance metrics using these events for comparison in simulation. A virtual framework was used to test degraded sensor states and the detection algorithm's response. Old Dominion University developed the GPS model and collaborated with the Global Center for Automotive Performance Simulation (GCAPS) to develop the degradation detection algorithm utilizing the DeepPOSE algorithm. GCAPS created the virtual framework, developed the LiDAR and radar sensor models, and executed the simulations. The sensor degradation detection algorithm will aid ADS vehicles in decision making by identifying degraded sensor performance. The detection algorithm achieved 70% accuracy. Additional training methods and adjustments are needed for the accuracy level required for vehicle system implementation. The process of collecting sensor data, creating sensor models, and utilizing simulation for algorithm development are major outcomes of the research.

17. Key Words Algorithms, degradation failures, detection and identification systems, driver support systems, Global Positioning System, laser radar, metrics (quantitative assessment), sensors	18. Distribution Statement No restrictions. This document is available to the public through the Safe-D National UTC website, as well as the following repositories: VTechWorks, The National Transportation Library, The Transportation Library, Volpe National Transportation Systems Center, Federal Highway Administration Research Library, and the National Technical Reports Library.

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No. of Pages 21	22. Price $0

Form DOT F 1700.7 (8-72) Reproduction of completed page authorized

Abstract

The project developed a sensor degradation detection algorithm for Automated Driving Systems (ADS). Weather, cyberattacks, and sensor malfunction can degrade sensor information, resulting in significant safety issues, such as leading the vehicle off the road or causing a sudden stop in the middle of an intersection. From the Virginia Tech Transportation Institute's (VTTI's) Naturalistic Driving Database (NDD), 100 events related to sensor perception were selected to establish baseline sensor performance. VTTI determined performance metrics using these events for comparison in simulation. A virtual framework was used to test degraded sensor states and the detection algorithm's response. Old Dominion University developed the GPS model and collaborated with the Global Center for Automotive Performance Simulation (GCAPS) to develop the degradation detection algorithm utilizing the DeepPOSE algorithm. GCAPS created the virtual framework, developed the LiDAR and radar sensor models, and executed the simulations. The sensor degradation detection algorithm will aid ADS vehicles in decision making by identifying degraded sensor performance. The detection algorithm achieved 70% accuracy. Additional training methods and adjustments are needed for the accuracy level required for vehicle system implementation. The process of collecting sensor data, creating sensor models, and utilizing simulation for algorithm development are major outcomes of the research.

Acknowledgements

The authors would like to acknowledge the entire project team for their depth of knowledge and effort put into the project: Kevin Kefauver, Michelle Chaka, Changsheng Xin, Jonathan Darab, Steven Huggins, Alex Hatchett, Stephen Young, Cong Chen, Prakhar Kumar, Christina Witcher, Kenny Custer, Peng Jiang, and Danella Zhao.

The project team would also like to acknowledge our subject matter expert reviewer, Matthew Palmer, for his knowledge and insight.

This project was co-funded by the Safety through Disruption (Safe-D) National University Transportation Center, a grant from the U.S. Department of Transportation – Office of the Assistant Secretary for Research and Technology, University Transportation Centers Program, and by the Commonwealth Cyber Initiative.

Table of Contents

List of Figures

List of Tables

Introduction

Vehicles equipped with Automated Driving Systems (ADS) rely on sensors that provide information about the surrounding environment to make control systems decisions for real-time object avoidance and path planning. All sensors have strengths and weaknesses that vary by sensor technology, as shown in Figure 1. Thus, a comprehensive ADS sensor suite should have a range of sensors to establish a robust system. Example Tier 1 sensors include perception sensors (LiDAR, radar, visual cameras, ultrasonic) and localization sensors (global navigation satellite [GPS] system, high-definition maps).

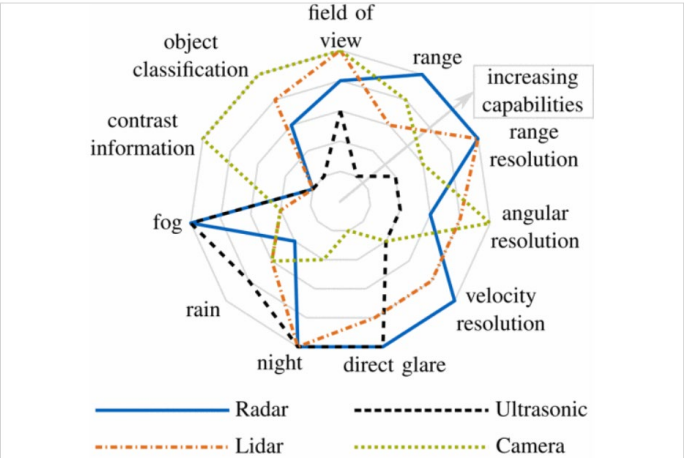

Figure 1. Radar chart. Systematic analysis of the sensor coverage of automated vehicles using phenomenological sensor models. [1]

The accuracy of the sensor information passing to the sensor fusion algorithm and then to the control algorithms greatly affects vehicle performance. Various forms of degradation can affect the sensor information, such as weather and information corruption due to cyberattack. Cyberthreats can create degraded sensor states by introducing misinformation to the sensor or system, including direct communication to the vehicle, false physical signage, and projected images in front of vehicles [2]. It has been widely demonstrated in the literature that an attacker can easily launch a GPS spoofing attack using a low-cost off-the-shelf software-defined radio such as the bladeRF. This could fool the victim vehicle into following a route crafted by the attacker. Misinformation from a sensor could also present significant safety concerns by leading the vehicle off the road or causing the vehicle to suddenly stop in middle of an intersection.

Background

The goal of this project was to explore whether algorithm-based detection can identify sensor misinformation. The misinformation can be in various forms such as degradation from weather or information corruption due to a cyberattack. The objective was to develop a sensor degradation detection algorithm through simulation, constructed and validated from physical testing and real-

world data. The intent of developing the algorithm was to improve ADS performance reliability by identifying potentially degraded sensor data. The project work was divided between three organizations. The Virginia Tech Transportation Institute (VTTI) provided real-world driving data and analysis to understand the conditions a sensor may experience (e.g., degradation factors), identified select use cases (event incident types), supported physical sensor testing on the Virginia Smart Roads, and explored overall performance considerations of the sensor degradation detection algorithm. Old Dominion University (ODU) developed a GPS model with spoofing detection and introduced cyberthreats for GPS-related information. The Global Center for Automotive Performance Simulation (GCAPS) developed test methods and models for LiDAR and radar sensors. GCAPS also developed a virtual simulation framework to evaluate the performance of the degradation detection algorithm. Both GCAPS and ODU developed the cyberthreat methodology and sensor degradation detection algorithm.

This project was divided into six subtasks:

- Task 1: Project Management
- Task 2: Analysis of Sensor Perception-related Crash and Near-crash Events
- Task 3: Virtual Framework for Simulation
- Task 4: Sensor Model Development
- Task 5: Sensor Degradation and Algorithm Development and Assessment
- Task 6: Final Report

Because the technical content was contained within Tasks 2 through 5, the remainder of this report will focus on those tasks.

Methods and Results

Task 2: Analysis of Sensor Perception-related Crash and Near-crash Events

The research leveraged real-world naturalistic driving study (NDS) data from the Second Strategic Highway Research Program (SHRP 2) NDS dataset, which includes vehicle kinematics and video data for over 5.5 million trips taken by over 3,500 participants [3]. These trips encompass approximately 32 million miles of driving data collected during thousands of hours of vehicle operation. Nearly 2,000 crash events, ranging from severe to minor, and nearly 3,000 near-crash events have been identified in the SHRP 2 dataset. T

From the NDS, the research team selected 100 events for initial simulations. From the 100 events, variations of each event were created in simulation to bring the total number of events to 1,000.

To determine the 100 events, a review of all available variables associated with crashes and near-crashes in the SHRP 2 dataset was conducted. The team identified four factors relevant to sensors perceiving the environment (weather, roadway alignment, lane of travel, and traffic density) and

analyzed the distribution of these factors for the selected incident types and crash severities. These variables are categorical, meaning they contain only a few discrete values and are not necessarily continuous or linear, which precludes many relationship-based analyses from the selection of events. There were 41,539 cases pulled from the database. Eighty percent were baseline events that did not contain crash or near-crash scenarios. These were initially provided in case they were useful for model development, which turned out to be unnecessary. Some high-level distributions of the data are shown in Table 1 and Table 2 in the Appendix.

Proceeding with the non-baseline events, the dataset was evaluated by variable combinations given the categorical data type. There were 8,960 non-baseline events and 1,078 unique combinations of the data and description variables; 566 of those combinations had only one occurrence within the dataset. There were still far more combinations than the target of 100 re-created events.

To further reduce the dataset, the team evaluated the unique combinations based on their variable distance, meaning the number of variables that were different between combinations. Variable distance between unique combinations is shown in the square matrix plot in Figure 2. A value of 0 indicates the combinations are identical and, because the data only contains unique combinations, values of 0 only occur on the diagonal. A value of 7 indicates the two combinations are different across all variables. Some clusters of similar events are evident in the dark blue areas of the figure resulting from the matrix sorting of unique combinations; however, there is still too much data to draw specific conclusions.

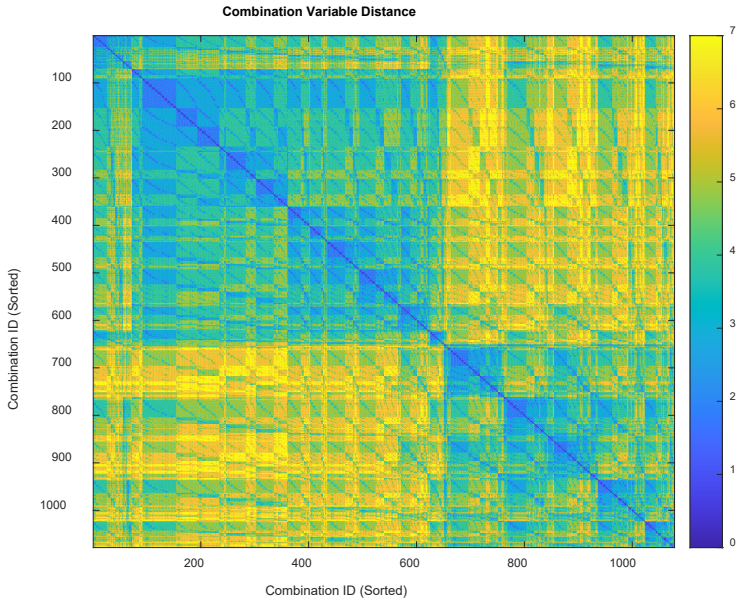

Figure 2. Square matrix plot. Variable distance between each unique combination.

The next step in reducing the combinations was to develop a method that would optimize the quality of the final selection. This was done with an algorithm to rank and eliminate combinations that are different in only one variable, which resulted in a ranked list containing 219 combinations.

The team used the top 100 combinations in this list as the final selection of events. These final combinations were matched to their actual events in the dataset. Because there were multiple events for each of these combinations, the events were evaluated on their data quality. The requirements included availability of the data channels needed for event re-creation and minimum signal dropouts. This final list of 100 events was carried forward into Task 3.

Task 3: Virtual Framework for Simulation

The virtual framework consisted of using MATLAB and IPG CarMaker together in a co-simulation environment to provide event simulation and sensor algorithm evaluation. Data groupings and flow path were identified between the two software applications to support the sensor models and algorithm.

Relative to simulation, the sensor models were integrated post simulation into the data stream. The object-level output data was collected from CarMaker's internal sensor models. These models handle the geometric constraints and sensor occlusions of the sensor. The degradation sensor model then augmented this time series data to reflect the response of the degraded sensor. The degraded sensor data was applied at a prescribed intensity and time. This data was unknown to the degradation detection algorithm and compared after the fact to assess the accuracy and performance of the algorithm. A diagram of this integration is shown in Figure 3.

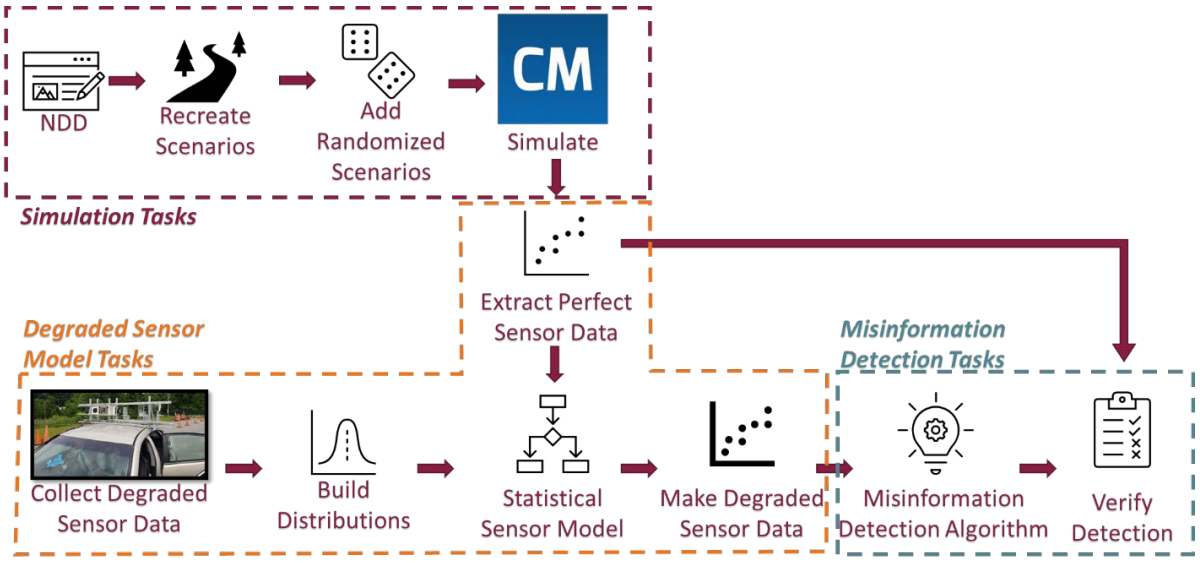

Figure 3. Diagram. Integration of simulation, sensor models, and misinformation detection tasks.

Simulation

To be used in the framework, NDD events required conversion using a GCAPS-developed method, which also allowed variations of each baseline event to be created via the OpenScenario format.

Decode: The simulation process began with reducing real-world event radar, inertial measurement unit (IMU) data, and visual data to a local coordinate digital format using a proprietary MATLAB graphical user interface (GUI).

Creation: A 3D environment of each event location was created in MathWorks RoadRunner. Environmental objects and road geometry were replicated for use in OpenScenario and CarMaker.

OpenScenario: The results from the Decode and Creation processes were used for the OpenScenario process. Combining the information from the decoding (location of actors, trajectory) and the 3D environment (lanes/junctions/exits) produces an estimate to visualize the trajectory of the ego vehicle with respect to other actors in the scene.

CarMaker Conversion: MATLAB processing script was run on the OpenScenario base event output to convert the event-based data back to X-Y data. Subject vehicle velocity and trajectory files for the controller algorithm and vehicle actor positive text files were created in this step.

The CarMaker global coordinate system (GCS) was configured to output latitude and longitude data. The actor vehicles were manually set to mirror the original event, and the input file was set to the event ID number. The randomized OpenScenario events were run through CarMaker, and the output results were algorithmically analyzed based on eliminating vehicle actor collisions, eliminating variations where the subject vehicle took the incorrect path, and selecting variations with enough speed and position variation compared to the original base event. The simulation time, sample rate, and sensor parameters were set to match the NDD and test data parameters, respectively. Then, the final mass CarMaker simulations were run, and the output data was configured and saved in MATLAB.

Task 4: Sensor Model Development

Of the multiple sensors available to the ADS, this research focused on three Tier 1 sensors: LiDAR, radar, and GPS. Cameras were omitted due to project scope. This study focused on an algorithm's ability to detect degradations in a sensor signal. Since cameras and LiDAR are both light sensing devices, there would be some duplication of effort in the limited time and resources to complete the project. Sensor models of LiDAR and radar were created from data collected on the Virginia Smart Roads. While the sensor models were empirically based, the test conditions captured physical response data (i.e., different colors, surfaces, and orientations to capture the various light effects for a LiDAR sensor). Events for sensor characterization included non-moving, continuous moving, and sudden movement targets in a variety of degraded states, including varying rain conditions and lighting conditions. The sensor model for the simulation will need to work within the time domain; thus, GCAPS incorporated time continuity into the model outputs.

The sensor models were based around the statistical distribution of responses collected from actual sensors. The distributions were relative to the parameters of the simulation (target vehicle position/pose, lane curvature, velocity, etc.) and the prescribed degradation intensity (rain). This determined the augmentation applied to the simulation signal relative to modeled performance metrics. Depending on the response, the distributions were fit in MATLAB to match an existing probability distribution, parametric equations, and/or custom fitted and smoothed distributions.

The sensors under test were focused on more advanced types of radar and LiDAR technology: the Continental ARS430 radar and the beta version of the Innoviz Pro LiDAR. The newer ARS430 operates at a higher frequency, 76 GHz, and outputs point-cloud-like data. The Innoviz is a solid-state LiDAR with no motor-driven components, which are common in older sensor models.

A differential GPS (DGPS) was used to track the test vehicles' exact position to aid in processing. This eliminated the potential variable of which object detection technique to use to extract the target data and allowed that element to be controlled within the sensor models, if desired. The known relative position of vehicles seeded the bounding box extraction of object information from the point cloud outputs. It also provided the true value to compare against sensor responses and generated error signals needed to build the metrics that describe the statistical model components.

In Figure 4, a top-down LiDAR point cloud is shown at $t = 0$ s and $t = 5$ s. The Innoviz LiDAR, being solid-state, has a fixed field of view of 72°. It contains contour lines of the roadway and vertical structures such as the fence line containing the test track and far-off buildings that may come within range. The Innoviz LiDAR detects the intensity of returned light, so the road lines stand out because of their retroreflective paint. In the figure, the green circle is the subject vehicle with its heading arrow from the DGPS; the red circle is the target vehicle from its DGPS. Having the DGPS allows the simple removal of non-target data to make the process of object detection straightforward and deterministic, without adding the inaccuracies of a perception system. The radar data is similar; the ARS430 produces a front-facing pseudo-point cloud, though it is much less sparse than a LiDAR and more susceptible to noise and variations in returned power.

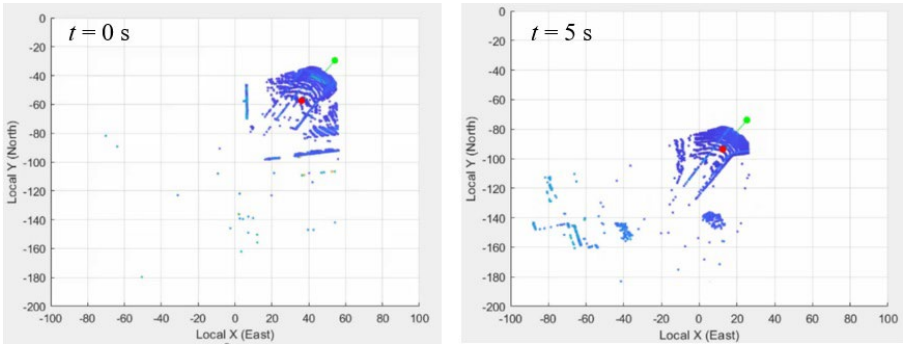

Figure 4. Scatter plots. LiDAR data examples.

To fit the models, the team used scenario-based tests to build and validate the statistical sensor models. The tests evaluated were ranging sweeps, lane slaloms, and circle track tests. The range sweeps provided dynamic longitudinal information. The lane slalom provided lateral information. The circle track provided target pose and position information. There was, of course, overlap in these designations. The baselines of these models were built around a static subject vehicle (sensing vehicle) with validation tests at speed to ensure that the responses did not deviate with speed.

The sensor models were built around the statistical distribution of data collected from actual sensors (i.e., Innoviz Pro LiDAR and Continental ARS radar). The sensor model's process for modifying the simulation data is shown in Figure 5. The sensor is broken into three main parts: preprocessing, data manipulation, and setpoint randomization.

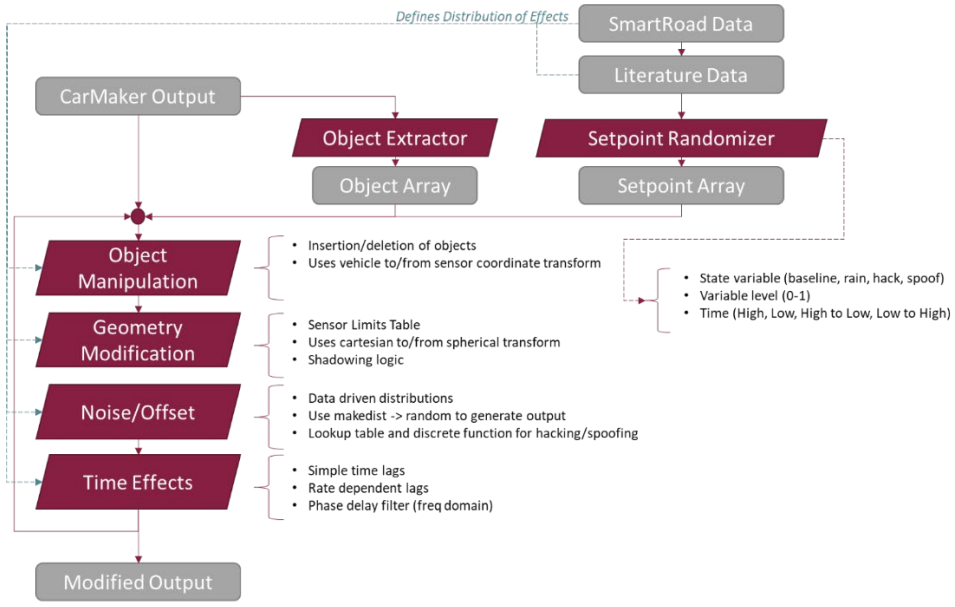

Figure 5. Flowchart. Sensor model process.

The preprocessing part of the model is an object extractor and coordinate transformer. The traffic and environment objects were collated from the CarMaker data into a more usable format for the degradation algorithm and/or the DeepPOSE localization algorithm being co-developed by ODU. DeepPOSE is a framework that equips a deep learning model to address the noise introduced in sensor readings and detect GPS spoofing attacks on multiple platforms, including mobile phones and vehicles. Each LiDAR/radar sensor's coordinate system was converted to the same reference frame to improve the usability. The core data manipulation of the model is four subprocesses:

1. Object Manipulation (Insertion/deletion)
2. Geometry Manipulation (Sensing limits, detection range, and field of view)
3. Noise/Offset (Shifting and scaling of data)
4. Time Effects (Latency, dropouts)

The continuous effects, like noise and offset, were defined statistically. The distributions defining these subprocesses are relative to the parameters of the simulation (target vehicle position/pose, lane curvature, velocity, etc.) and the prescribed degradation intensity (rain). These were built in MATLAB around existing probability distribution, parametric equations, and/or custom fitted and smoothed piecewise cumulative distribution functions. For other modeled sensor effects such as spoofing, hacking, or dropouts, the responses were parameterized to state variables according to a function/system of equations, lookup tables with interpolation, statistical distributions, or a combination of these. The setpoint randomizer determined how the CarMaker datasets were

augmented. The array for all these setpoints is unknown to the detection algorithm but saved for calculation and to compare with the algorithm output to determine the proficiency of detection.

Additional GPS data was collected to validate the GPS/IMU sensor models to provide augmented data to feed through DeepPOSE. The GPS test data was collected on two separate days with clear and overcast weather, driving across all areas of the Smart Roads (urban, highway, and rural). A standalone automotive GPS and a DGPS system were used to indicate the accuracy of the onboard GPS system to atmospheric conditions to scale the continuous degradation effects to the normalized degradation input request to the model and non-continuous shifts in position to the loss/gain of satellite reception. The data was adapted to the statistical distributions and sensitivities of simulation variables by the same methods as the LiDAR and radar data. Example plots of the GPS and IMU degradation effects on the yaw signal are below in Figure 6. The effects shown were the noise from atmospheric conditions to GPS, the potential voltage bias of the output amplifier of an IMU, and the calibration error that commonly drifts over time of an IMU. The effects on the IMU grow in error overtime because IMU sensors inherently rely on integration down to the first order signals, and the integration errors accumulate.

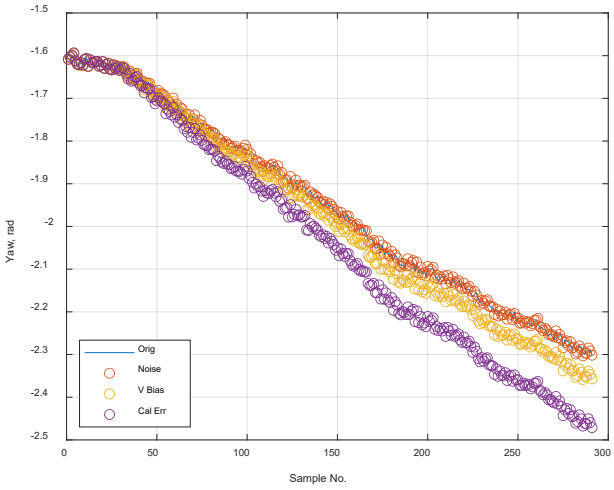

Figure 6. Scatter plot. Example yaw data of GPS signal with noise and degradation.

The team created the input degradation array to challenge the detection algorithm. The array was built to be informed by a mixture of natural rainfall parameters and enough steady state conditions to not over or under constrain the variety of data to train the algorithm. The rain intensity was normalized across the unit interval, 0 to 1, with 0 being no rain and 1 being tropical storm levels. The way this intensity changes over time, shown in Figure 12, is typically a gentle lead-in to the heart of the storm, then a large and rapid change in rainfall, and then a similarly gently lead-out.

To make this into a simple input array for the simulations, a sigmoid shape was used with randomly assigned start and end points. These points were defined by two normally distributed intensities to match the lower intensity lead-in or lead-out periods and the higher intensity peak period of the

storm. This bimodal intensity results in some events having a large transition from low to high, and vice versa, and some events having mostly constant high or low intensities. The output shape was scaled and shifted to the beginning and ending intensities. Thirty percent of the cases were overwritten to be a constant 0 intensity and 10% were overwritten to be a constant 1 intensity. This was to ensure that the training data allows the detection algorithm to be sensitive to dry conditions and discern degraded signals, even if they are constant in the evaluation time.

The event breakdown of rain and level of rain is:

- 1,155 Total events
  - 369 Events with constant conditions
    - 161 with no rain-37 with all heavy rain
  - 394 Events with increasing rain
    - 188 from no rain
  - 392 Events with decreasing rain
    - 71 from heavy rain

Task 5: Sensor Degradation and Algorithm Development and Assessment

The sensor degradation detection algorithm (1) uses the sensor inputs to determine the possible representation of vehicle positions in space and time relative to the subject vehicle and road geometry for each sensor, (2) compares the results, and (3) determines possible misinformation of the sensor. This was accomplished using input data sources along with external map data. The completed algorithm was run across the 1,000 events to create time histories of all relevant objects in the scenario. The research team applied state-of-the-art convolutional neural networks that allow precise and fast detection of traffic objects from naturalistic driving data despite varying image quality. Subject vehicle movements were estimated by fusing different sensor outputs in a Bayesian framework, taking into account different error sources in the data. The object detection results and the estimated trajectory of the subject vehicle were complemented with other data sources such as radar and map data.

The ODU team designed a deep learning system that uses motion sensors in a vehicle IMU, including accelerometer and gyroscope sensors, to estimate vehicle position and further detect GPS spoofing attacks. The deep learning system has two components: a vehicle position estimator and a GPS spoofing detector. To estimate the vehicle position from the noisy motion sensor readings, the system uses a combined convolution neural network and sequence-to-sequence neural network to transform sequences from the source domain to sequences in the target domain. In the system, the multi-dimension sensor measurement was taken as the source sequence, and the vehicle state, including the speed and direction, served as the target sequence. To detect GPS spoofing, an efficient detection algorithm was designed that detects the attack by composing the real-time trajectory (based on the vehicle speed and direction obtained from the motion sensor data) and comparing it with the trajectory reconstructed by the GPS signals.

To increase the accuracy of the trajectory reconstruction from the noisy motion sensor data, a novel map alignment scheme was conducted to align the reconstructed trajectory with an open-source street map, which can significantly reduce the accumulation of displacement error from the trajectory reconstructed from the noisy sensor data when the vehicle driving distance is large.

Baseline Noise and Degradation Noise

Baseline noise refers to any background interference, which can cause fluctuations in the sensor data measurements. The baseline noise typically has the form of stray ambient light, which affects beam diameter and LiDAR data signal strength measurements. Our simulated baseline noise, which mimics the baseline erroneous readings, has the form of various levels of spikes adding to the original data. Because the baseline noise is a natural property of LiDAR data and is different from misinformation in the object detection level that requires detection, it becomes important to minimize the background noise while increasing the accuracy in degradation pattern detection.

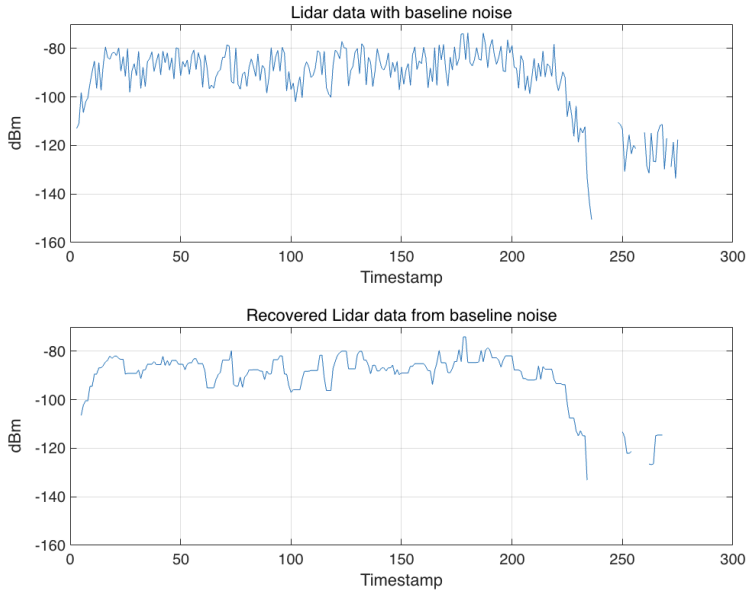

Figure 7. Line graphs. Experimental results of recovered LiDAR data from baseline noise.

The top of Figure 7 illustrates the frontal LiDAR signal strength data with baseline noise as irregular spikes at various levels. The baseline noise removal algorithm is a preprocessing step of the degradation detection algorithm that removes baseline noise before the analysis of the degradation data is started. The baseline noise removal step is implemented using a sliding window method that calculates the neighborhood data point's intrinsic statistical information and analyzes the discontinuation of the spike data point. The optimal sliding window size is 5, based on the experiments. The output of the baseline noise removal method is shown in the bottom of Figure 7.

Deep Learning-based Degradation Detection Algorithms

The training dataset is a structure array with two fields: Data and Labels. The Data field contains the object level detection data for each actor within the scenario that is reduced from the many subject vehicle sensors down to a single detected object by a priority filter. This filter prioritizes

by frontal sensor detections and then by strength of signal in the object data output. This limits the number of signal switches that could confound the detection algorithm.

The Data field is a 2D matrix in which each row is a simulated driving event sampled at 10 Hz. The row index represents the event index, and the column index represents the timestamp. The baseline noise has been put in the top half of the data, and the degradation sensor data has been put in the bottom half of the data, arranged such that each baseline noise data and degradation noise data from the same event have the same interval.

To analyze the sensor signal feature on multiple levels to maximumly extract the feature of data, time-frequency representations of the sensor signals were created, also called scalograms. The scalogram is the absolute value of the sensor signal's continuous wavelet transform (CWT) coefficients. The first step for creating the scalograms is to precompute the CWT filter bank of the sensor signal. The analytic Morse (3,60) wavelet is used, which is a default in the structure of the GoogLeNet network backbone described in the next section. To match the simulation data's sampling time and frequency, a signal length of 290 and a signal frequency of 10 were used. To examine one of them, the filter bank was created from the first 290 signals to take the CWT and get the coefficients of the signal to then create the scalogram. The CWT filter bank provides the CWT of the sensor signals. The scalograms have been constructed based on the wavelet coefficients. Then, the scalograms were saved as 2D RGB images and fed into the deep learning neural networks. An example is shown in Figure 8.

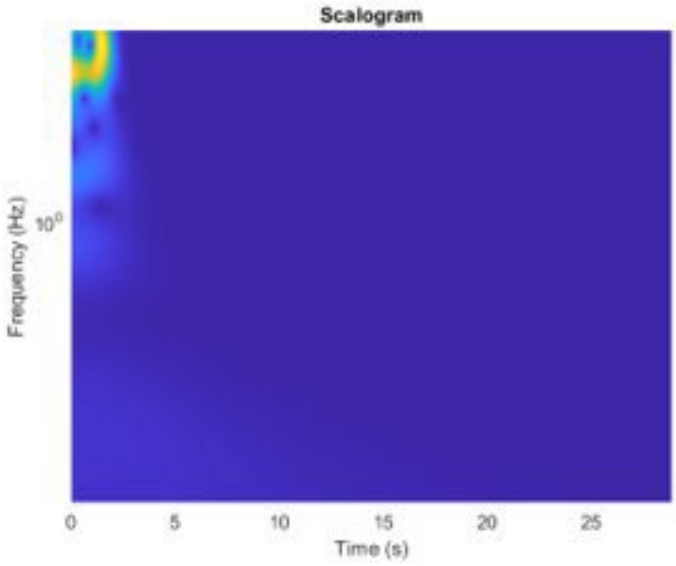

Figure 8. RGB scalogram. The conflict happened in the beginning of the 30-second event.

The vehicle conflict data is concentrated in the beginning of this event due to the nature of the original naturalistic data and randomization process. The low frequency content in this region is also reduced by the filtering processes to remove the slow and steady state content expected in the data that can confound the detection algorithm. The data has been randomly divided into two

groups. Eighty percent of the dataset was assigned to the training set and 20% to the validation set. This ratio is common for training with this type of data as it provides a bit more data to train the neural networks with a limited dataset of simulated scenarios.

The network backbone is GoogLeNet [4]. It takes 2D RGB images as input, and the input size is 224 x 224 x 3. The GoogLeNet is designed to classify more than 1,000 categories for images. The model is pretrained to its original purpose, so the network needed to be retrained for the proposed sensor signal with two types (baseline noise and degradation noise).

The early stages of the network try to extract some low-level image features, including color and edges. The later stages are expected to extract more intrinsic features of the sensor signals, which can be more representative for separating the baseline noise and degradation noise of sensor data.

The more detailed layer graph of the GoogLeNet used is listed in Figure 13. The dropout layer was added to prevent overfitting, which can randomly change the input element's value to become 0 based on a 0.5 probability, and the final dropout layer's probability was set to 0.6. The last two layers of the convolutional neural network are the important part for extracting the image feature for classification. The last two layers combine all the information extracted in the previous layers to become the probability of a certain class label, as well as the loss values. Because this research only concerned the two types of sensor signal, which is different from the original GoogLeNet designed for 1,000 categories, the last two layers were replaced for the current task. The original fully connected layer was replaced with a new fully connected layer that has the two filters. This also allowed increasing the learning rate in the fully connected layer for a faster learning speed.

The target of the training process is to minimize the loss function iteratively between prediction and ground truth. Inside each iteration time, the gradient of the loss function leads to the change of the descent algorithm weights. The parameter tuning inside the training process can make a big impact on the performance of the neural network. For example, tuning the number of epochs is critical. If the epochs are too small, the network may have a problem with underfitting, which means the model cannot provide sufficient information to predict the output. If the epochs are too large, then the model can have a problem with overfitting, which means the model is too good for the training data but lacks the generalization for adapting to new testing data. After many experiments, 20 epochs were chosen as optimal. The mini batch size is 10, which corresponds to the subset number of training data inside each iteration. The initial learning rate was set as 0.0001, which represents the initial step size toward the direction of negative gradient in the loss function.

The training set chosen is:

Four state variable inputs for the models

- Baseline noise
- Rain
- Emulating two hacking/spoofing effects

Four activation levels for each of those variables

- On throughout
- Off throughout
- On to off
- Off to on

Cross entropy was chosen as the loss function. The convergence process is shown in Figure 9. As indicated in the figure, the training accuracy shown at the top keeps increasing as the iterations increase, and the loss shown at the bottom keeps decreasing during the process. After 20 epochs and 280 iterations, the training finished in 23 minutes with a single CPU. The learning rate of 0.0001 was the most optimal value based on the results.

The accuracy of the network is 70.37% when using the given validation dataset. In this instance, only two categories of data were used: baseline noise sensor data and degradation sensor data. In the future, improvements may be made using new categories for new types of essential patterns inside the sensor data.

The proposed degradation signals have very subtle differences compared to baseline signal data. Both signals have irregular patterns compared to common periodical signals. Thus, the sensor data requires the networks to have a stronger learning capability to best capture the different patterns inside the degradation data. To further improve the performance of the current algorithm, a better data cleaning method that refines the sparse data to have more informative features inside the training dataset can be very useful based on our experiments.

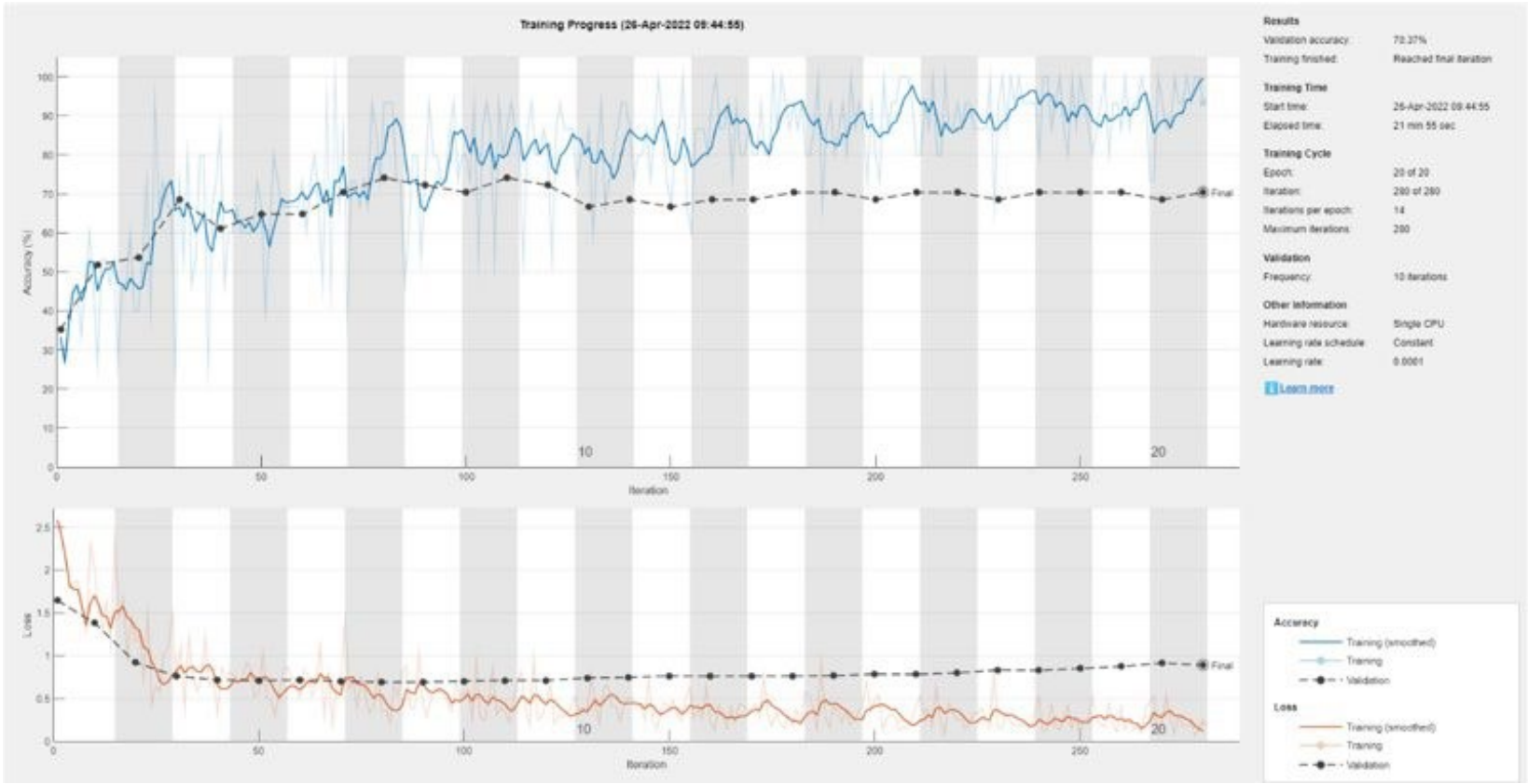

Figure 9. Line graphs. Training process of the proposed network.

The algorithm accuracy during training achieved approximately 70%. To improve accuracy, larger data sets are required. To compare to the NDD data, simulation data was "played back" and evaluated at each simulation time step with the data up to that time step. When the algorithm determined that a degradation was occurring, the playback clock was stopped to determine the "reaction time" of the algorithm. After iterating through 44 events in this way, the algorithm was shown to perform poorly (Figure 10) in this evaluation method. The remaining events were not evaluated since the early results already indicated that improvement was needed in the algorithm detection rate. The improvement in the accuracy of the algorithm will contribute to improving the reaction time. Further investigation is needed to determine other paths than this playback evaluation for improving the algorithm reaction time.

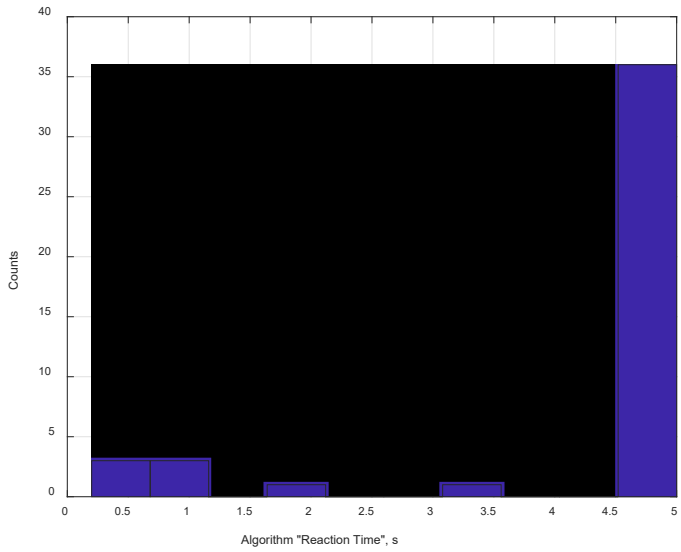

Figure 10. Bar graph. Algorithm reaction time from first 44 events.

DeepPOSE algorithm

DeepPOSE is a deep learning framework proposed to (a) address the noise introduced in sensor readings and (b) detect GPS spoofing attacks on mobile platforms. It contains a vehicle position estimator, which uses a convolutional and recurrent neural network (RNN) to reduce the noise and recover a vehicle's real-time trajectory from multiple sensor inputs, and an offline map, which projects a reconstructed trajectory in a real-world road map to eliminate the accumulation of errors on the trajectory estimation effectively. The reconstructed trajectory from sensors is then used to detect the GPS spoofing attack.

DeepPOSE takes sequential IMU sensor data as input, then produces estimated vehicle positions sequentially to profile the relative movements of the target vehicle. The benefits of using DeepPOSE to reconstruct the vehicle trajectory instead of using a self-correcting filter are manifold. First, a self-correcting filter like Kalman requires a good understanding of the system being estimated, including the system dynamics and measurement noise. Estimating the necessary parameters for a complex vehicle system can be challenging. Furthermore, to accurately estimate a vehicle's state with a Kalman filter, multiple sensors must often be used in combination. However, fusing sensor data from different sources can be complex and challenging, as different sensors may have different noise characteristics and biases. While DeepPOSE is proposed as a unified framework to process noisy data from multiple sources, it can effectively estimate the vehicle position without considering the sensors' different noise characteristics and biases.

Model Inputs and Outputs

There were K different sensors provided by CarMaker. Sensor k has the sampling frequency f_k. Let d_k represent the number of axes for each sensor (e.g., measurements along x, y, and z axes). The GPS coordinates C are sampled at the frequency f_c.

Unlike other deep learning models that require each data input to be paired with a target or label, the sequence-to-sequence model used in our design requires pairing a series of inputs with a series of target speeds and directions in the time domain. After all sensor measurements are aligned with the correct targets, the raw measurements were split into small sequences with the same number of the time unit, n_τ. A 3D matrix, $\mathcal{X}^k$, was used to represent an input sequence generated by the sensor k. The depth of the input is n_τ, which is also the number of encoders/decoders in the sequence-to-sequence model. The height of input is d_k, which depends on the number of axes of sensor k. The width of the input is $\omega \cdot l_k$, where ω is the size of the sliding window to control the number of sensor measurements aligned to one single target. When multiple sensors are considered, the inputs are stacked into a $n_\tau \times \sum_k d_k \times \omega\, l_k$ matrix $\boldsymbol{\mathcal{X}}$.

A unified symbol, $\boldsymbol{\mathcal{Y}}$, denotes the output vector corresponding to each input vector $\boldsymbol{\mathcal{X}}$. When the model is used to estimate the vehicle speed, $\boldsymbol{\mathcal{Y}} = \boldsymbol{v}$, where $\boldsymbol{v} = \{v_t\}, t \in \{1, \ldots, n_\tau\}$.

Similarly, $Y = \theta$ when the target is the vehicle direction. For the sake of consistency, all vectors are denoted by bold $\boldsymbol{\mathcal{Y}}$, and an instant target value at time *t* is denoted by $\mathcal{Y}_t$.

Sequence-to-Sequence Modeling

. Unlike a single or stacked RNN, which operates on a sequence and feeds its own outputs for subsequent cells, most Sequence-to-sequence learning (seq2seq) models are encoder-decoder models composed of a set of two RNNs. The first RNN, encoder, trains the input data, and then passes the last state of its recurrent layer as an initial state to the first recurrent layer of the decoder. Figure 14 illustrates the overall design of our sequence-to-sequence model used by DeepPOSE.

Based on the sequence-to-sequence model, the encoder long short-term memory (LSTM) processes the input sequence $\boldsymbol{\mathcal{X}}$ of n_τ elements and passes the internal state (hidden state) representation to the next encoder until it reaches the last encoder. Then, the first decoder receives the hidden state generated by the last encoder and adds one initial input $\mathcal{Y}_0$ to generate its target output $\mathcal{Y}_1$ and updates the hidden states. The second decoder takes the new hidden state and $\mathcal{Y}_1$ as inputs to generate a new output $\mathcal{Y}_2$, and so forth. Eventually, the target sequence $\boldsymbol{\mathcal{Y}}$ is obtained, which is ($\mathcal{Y}_1$, ..., $\mathcal{Y}_{n_\tau}$).

In the training process, $Dec(\cdot)$ was used to denote the decoder output of our model and $< \boldsymbol{\mathcal{X}}, \boldsymbol{\mathcal{Y}} >$ to denote the training samples and labels. The loss function for training the model, $\mathcal{L}$, is given as:

$$\mathcal{L} = \sum_{t}^{n_\tau} l(\, Dec(\boldsymbol{\mathcal{X}}_t), \mathcal{Y}_t)) + \sum_{j} \lambda_j P_j$$

The second term in the loss function is the regularizations' function, and λ_j controls the importance of penalty or regularization terms. In the design, an L2 regularization is used in the training process, and the weight decay is set to 0.01.

Performance of DeepPOSE in Speed and Direction Estimation

The team selected 560 out of 700 trips (with average duration of about 40 seconds) to form training data to optimize the model learning process, and the remaining 140 trips were used to perform the stepwise analysis for degradation detection. The maximum vehicle speed in the dataset is 36.39 m/s, and the range of vehicle heading falls into the range of [-25, 25] degrees. Based on this, the vehicle instant speed and direction were converted into 50 categorical values (encodings) before entering them into the model. The DeepPOSE algorithm will take a sequence of IMU inputs and predict a sequence of categorical outputs at the same length. The predicted categorial values will be converted back to the vehicle states (i.e., speed and direction) to construct the vehicle trajectory. With a larger encoding number, a more precise estimation of vehicle speed and direction can be achieved. However, it cannot maintain the same accuracy due to the insufficient training set.

The following configurations were tested to obtain the speed and direction estimation:

- **Types of input IMU sensor**: accelerometer only, gyroscope only, accelerometer and gyroscope
- **Sequence length**: 5 seconds, 10 seconds, and 15 seconds
- **Number of hidden states**: 512, 768, and 1,024
- **Depths of ConvLSTM encoder**: two layers of Conv2D and three layers of Conv2D

Accelerometer combined with gyroscope inputs were optimal from the sensor inputs. The optimal sequence length is 5 seconds in the current dataset. Empirical results suggest that a longer sequence will increase the learning capability but decrease the number of sequences for training the model. When the sequence length is 5 seconds and the distance between two adjacent sequences is 3 seconds, 560 trips can produce 4,482 sequences for training and 1,121 for validation. When the sequence length increases to 10 and 15 seconds, the number of training/validation pairs decreases to 3,636/910 and 3,101/776, respectively. To enter enough training samples into DeepPOSE, the sequence length was kept to 5 seconds. In terms of the depth of the model, two layers of Conv2D were used instead of three layers to avoid possible overfitting. Therefore, each ConvLSTM encoder has two convolution layers with 64 filters, and the kernel size is 3 x 3. For the same reason, the hidden state in LSTM is set to 768 instead of 1,024 to reduce the model complexity.

Intensive experiences are conducted with various hyperparameters including the selection of length of sequence, regularizations, number of hidden layers, etc. In addition, to test the model performance with the configurations mentioned previously (type of IMU sensor inputs, sequence length, number of hidden states, depth of ConvLSTM encoder, etc.), weight decay is applied from 0 to 1, with a step of 0.05, to leverage the overfitting and fine-tune the model. Among the various results, the best accuracies for speed and direction estimation are shown in Figure 11. The results show an overfitting in the training process, which means the validation accuracies for both speed and direction estimation (about 72%) are much lower than the training accuracy (near 99%) when the model is set with simplest structure, i.e., with a single ConvLSTM layer and 768 hidden states. Overfitting usually occurs when the model has a relatively large number of learning parameters

instead of generalizing the examples, which is 5,000 training pairs. Therefore, the model is not performed at the optimal performance on the testing dataset. In the future, improvements may be made using more events simulated from CarMaker to better generalize the model.

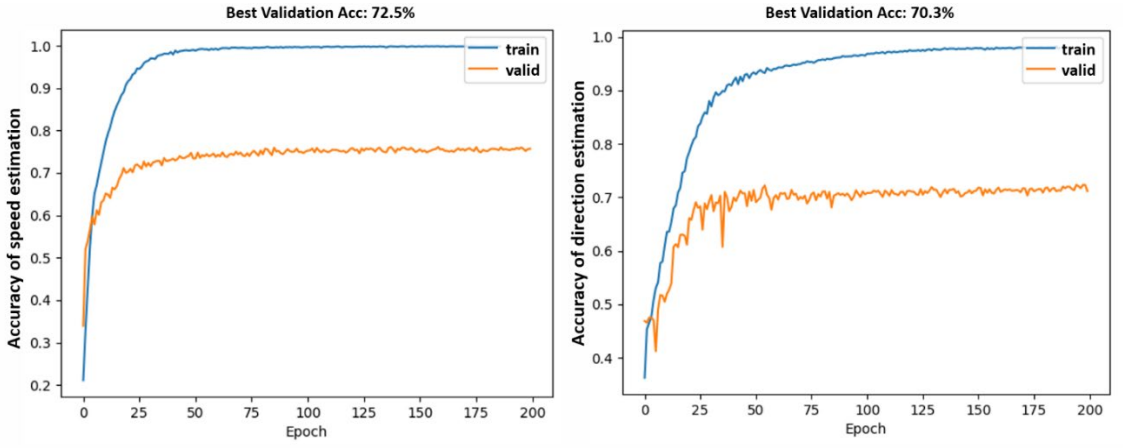

Figure 11. Line graphs. Accuracy of DeepPOSE from training events and validation events.

Performance of DeepPOSE in GPS Degradation Detection

To assess the performance of the designed framework in detecting such GPS degradation scenarios, we first use DeepPOSE to reconstruct the vehicle trajectories from IMU sensors, which are robust to bad weather and GPS spoofing attacks. The reconstructed trajectory generated from IMU is then compared with the vehicle trajectory extracted from the coordinates reported by the GPS module, which is subject to various degradation scenarios. If the difference between the reconstructed trajectory and the reported GPS trajectory is larger than the threshold, the GPS signal is then considered degraded. The experiment was run on 140 testing trips in the dataset that the model had not seen and with various levels of rain intensity. Figure 15 includes an example of the necessary information to perform degradation detection with DeepPOSE. There are nine subfigures generated through the assessment. Subplots are numbered as <*column,row*>, where subplot <1,1> refers to the plot in the top left corner, subplot <3,3> refers to the plot in the bottom right corner. Subplots <1,1> and <1,2> show the vehicle speed and instant direction extracted from CarMaker (ground truth), GPS, and DeepPOSE. Subplot <1,3> plots the rain (degradation) intensity. Plots in the second column indicate the errors introduced by GPS and DeepPOSE, compared to the ground truth values produced by CarMaker at each timestamp. Plots in the third column show the differences between GPS and DeepPOSE, which are critical to determining if the GPS signal is under degradation.

Given that DeepPOSE is not working at optimal performance, the GPS error introduced by rain intensity is similar to or even less than the error estimated by DeepPOSE, which makes it hard to use DeepPOSE to identify the starting time of the GPS degradation by measuring the Euclidean distance error between GPS and DeepPOSE.

Although the distance error of DeepPOSE is similar to or higher than the degradation caused by rain, our model is vigilant to a special type of GPS degradation, GPS random walk. In the random walk degradation, the GPS signal drifts randomly but is still close to the real location. Therefore, the bearing difference between GPS and DeepPOSE can be used to detect such degradation.

Figure 16 demonstrates the GPS random variation degradation, where the vehicle instant course calculated from the GPS signals fluctuates significantly. To eliminate the false alarm caused by the noise introduced by DeepPOSE, an observation window was set to detect if the random variation persists after a period of time. If the variation exists, this will confirm the degradation.

Among the 140 testing trips, 20 trips were selected in which GPS performed a random walk when the rain started, and the DeepPOSE algorithm successfully detected 16 of them, with the detection rate of 80% when the observation window is set to 5 seconds. When the observation window reduces to 3 seconds, the detection rate remains the same, but the false alarm rate increases.

Further investigation is needed to improve the performance of DeepPOSE, which can help the system reduce the reaction time for the GPS random walk degradation and other types of GPS degradation caused by weather.

GPS and LiDAR Data Comparison

Because the GPS data format is different from the LiDAR data, the two data streams are compared in a different hidden level, the action level, instead of in the raw data. After extracting the raw data to become action symbolic data in the different time window, the two data streams can be compared and output the timeframe when the two data streams have misinformation compared to each other. In the first step, the "car offset" data was extracted based on XODR files, then the derivative of the offset data stream was calculated. Big drift inside the derivative data can indicate the change point of the vehicle's action, so the change point detection algorithm was applied to detect and output the action change timestamps for both LiDAR and GPS data for comparison. The change point detection algorithm's time-division variable keeps iterating automatically until the loss function inside each division reaches the minimum. The results show that the change point detection algorithm can effectively and accurately output the timestamp based on the vehicle's lane line offset data acquired from LiDAR when the ego vehicle changes the action.

Assessment

As part of this task, the performance of the algorithm was compared with reaction times recorded during crash and near-crash events in the SHRP 2 NDS. Figure 17 shows the time, in seconds, between the Subject Reaction Start time and the Impact Proximity Time for all events in the SHRP 2 NDS, where:

- Subject Reaction Start is "[the] timestamp, in milliseconds, after the start of the file, when the driver is first seen to recognize and begin to react to the safety critical incidents occurring. Defined as the first change in facial expression to one of the alarm or surprise

or the first movement of a body part in a way that indicates awareness and and/or the start of an evasive maneuver, whichever occurs first." [3]

- Impact Proximity Time is "[the] timestamp, in milliseconds after the start of the file, when the subject vehicle and another object of conflict first make impact. In the case of a near-crash, [this is] the timestamp when the subject vehicle and other object of conflict are at their closest distance to each other." [3]

For most crash events, driver reaction started less than 1 second before impact. However, for near-crashes, there was more time between the start of the driver's reaction and the Impact Proximity Time. For events with rain present (Figure 18), the distribution is similar to non-rain events.

This analysis indicates that the detection algorithm needs to identify degradation from weather, malfunction, or cyberattack within the same time span, less than 1 second.

Discussion

The accuracy of the algorithm in detecting the degraded state of the sensors was the main topic of discussion. For the GCAPS-created algorithm, several tuning parameter sets were attempted to improve the accuracy but had little effect on the results. From this, the source of the inaccuracy was discussed relative to the algorithm chosen and the training sets. It was decided that the training set limitations are more than likely contributing to the poor accuracy. The study was limited in the number of events that could be replicated and simulated, and this directly impacts the size of training data. Increased data in the training set can also improve the model's ability to discern some of the subtler changes in response from lightly degraded events. Relative to the ODU DeepPOSE algorithm, this limitation also produces a low accuracy for detecting degraded states. The limited dataset was determined to influence accuracy somewhat, but another contributor was the initial design of the algorithm. The algorithm was initially scoped to also detect larger spoofing errors and other cybersecurity misinformation, not just the subtler influences from weather or noisy sensors. This provided a greater challenge to the performance of the algorithm.

Conclusions and Recommendations

The team successfully created a method to train a sensor degradation detection algorithm through simulation. The method developed specifically for characterizing and implementing degraded sensors into simulation has applications to many other ADS development activities. Some refinement is needed to improve the generation of training data into a less time-intensive creation process. Also, a linear instead of binary classifier in the neural net may be more realistic for detecting the continuous variable degradation intensity, with a binary classifier then applied to that output if desired, along with the estimated degradation intensity.

Additional Products

Education and Workforce Development Products

Education and Workforce Development Activity or Product	Date Expected	Completed?
Students onboarded	4/1/2021	Completed
Establish monthly multi-disciplinary project coordination meetings	4/15/2021	Completed
Sample sensor data and model documentation	5/15/2022	Completed
Present webinar	8/19/2022	Completed
Archive report to VTechWorks	5/31/2022	In Progress

Technology Transfer Products

Technology Transfer Activity or Product	Date Expected	Completed?
Present webinar	8/19/2022	Completed
Archive report to VTechWorks	6/30/22	In Progress
Present at Road Safety on Five Continents (RS5C) Conference	10/10/2022	In Progress

Data Products

Processed Sensor Data: https://safed.vtti.vt.edu/projects/

References

1. Ponn, T., F. Müller, and F. Diermeyer. Systematic Analysis of the Sensor Coverage of Automated Vehicles Using Phenomenological Sensor Models. In *2019 IEEE Intelligent Vehicles Symposium, IV 2019, Paris, France, June 9-12, 2019*, pp. 1000-1006. [ieeexplore.ieee.org/document/8813794]

2. Nassi, B., et al. Phantom of the ADAS: Securing Advanced Driver-Assistance Systems from Split-second Phantom Attacks. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. October 2020, pp. 293–308.

3. Hankey, J.M., M.A. Perez, and J.A. McClafferty. Description of the SHRP 2 Naturalistic Database and the Crash, Near-Crash, and Baseline Data Sets. April 2016. [vtechworks.lib.vt.edu/handle/10919/70850]

4. Szegedy, C., et al. Going Deeper with Convolutions. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

5. Moore, J.N., and R.C. Riley. Comparison of Temporal Rainfall Distributions for Near Probable Maximum Precipitation Storm Events for Dam Design. National Water Management Center. [www.nrcs.usda.gov/Internet/FSE_DOCUMENTS/stelprdb1044900.pdf]

Appendix. Supplemental Visualizations

Table 1. SHRP 2 Selection Distribution of Event Description Variables

	Value	Count	Percent
EVENTSEVERITY	Additional Baseline	12581	30.3%
	Balanced-Sample Baseline	19998	48.1%
	Crash	1855	4.5%
	Crash-Relevant	42	0.1%
	Near-Crash	6923	16.7%
	Non-Subject Conflict	140	0.3%
CRASHSEVERITY	I - Most Severe	113	0.3%
	II - Police-reportable Crash	184	0.4%
	III - Minor Crash	777	1.9%
	IV - Low-risk Tire Strike	799	1.9%
	Not a Crash	7087	17.1%
	N/A	32579	78.4%
INCIDENTTYPE	N/A	39182	94.3%
	Pedalcyclist-related	67	0.2%
	Pedestrian-related	169	0.4%
	Road departure (end)	137	0.3%
	Road departure (left or right)	1259	3.0%
	Turn across path	327	0.8%
	Turn into path (same direction)	398	1.0%

Table 2. SHRP 2 Selection Distribution of Event Data Variables

	Value	Count	Percent
WEATHER	Fog	189	0.5%
	Mist/Light Rain	1534	3.7%
	N/A	265	0.6%
	No Adverse Conditions	37516	90.3%
	Rain and Fog	25	0.1%
	Raining	2010	4.8%
ALIGNMENT	Curve left	2796	6.7%
	Curve right	3105	7.5%
	Other	2	0.005%
	Straight	35635	85.8%
	Unknown	1	0.0%
LANE	1	24299	58.5%
	2	9322	22.4%
	3	2576	6.2%
	4	632	1.5%
	5	140	0.3%
	6	12	0.03%
	7	3	0.01%
	Acceleration lane	308	0.7%
	Center 2-way turn lane	123	0.3%
	Deceleration lane	360	0.9%
	Dedicated left turn lane	1188	2.9%
	Dedicated right turn lane	855	2.1%
	N/A	1721	4.1%
DENSITY	A1: Free flow, no lead traffic	14702	35.4%
	A2: Free flow, leading traffic present	11385	27.4%
	B: Flow with some restrictions	11539	27.8%
	C: Stable flow, maneuverability and speed are more restricted	2577	6.2%
	D Unstable flow - temporary restrictions substantially slow driver	872	2.1%
	N/A	464	1.1%

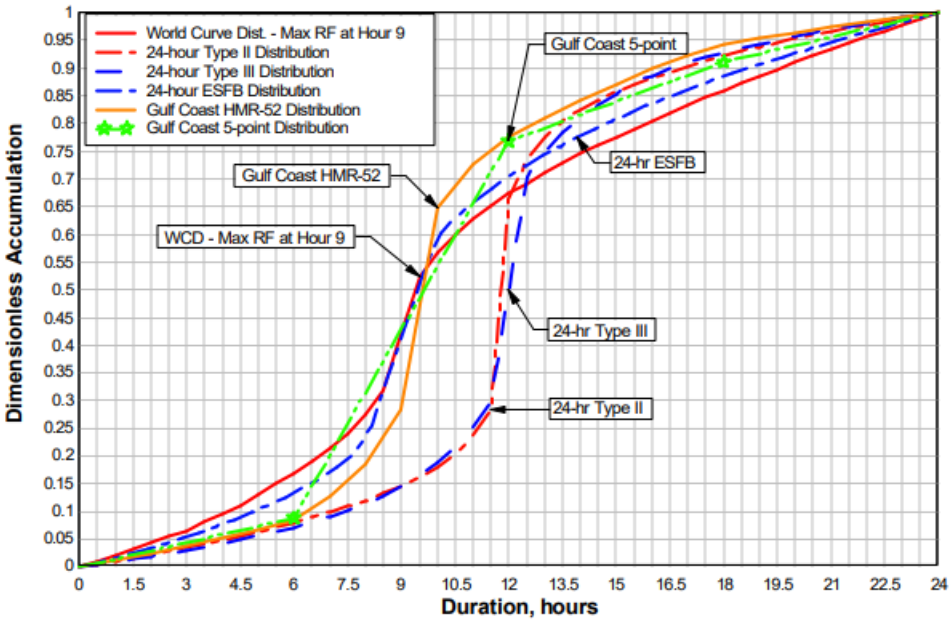

Figure 12. Line graph. Storm rainfall model comparison [5].

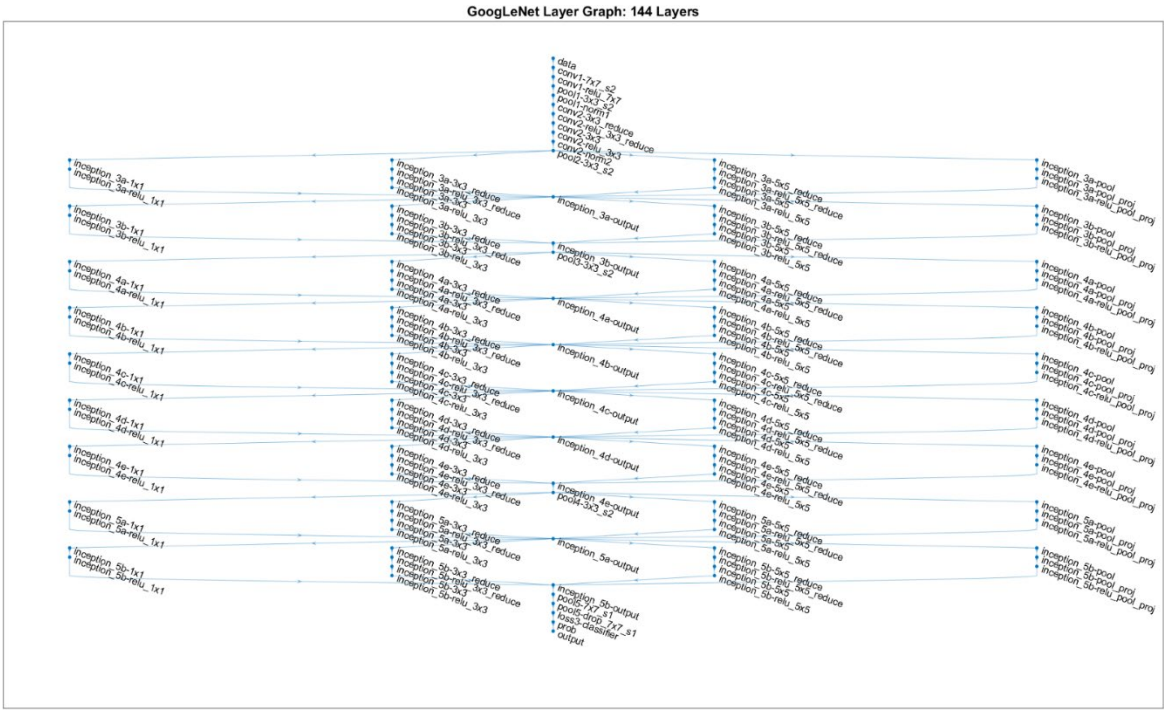

Figure 13. Layer graph. Layer graph of the GoogLeNet.

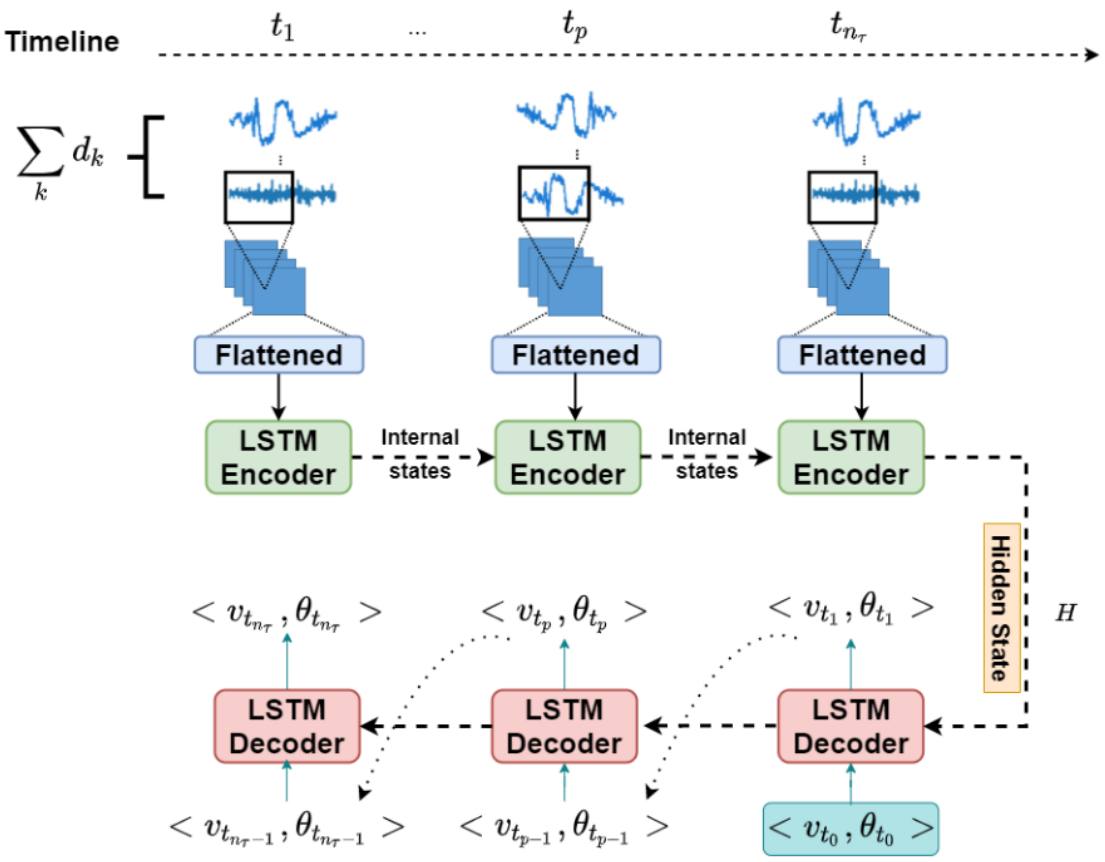

Figure 14. Flowchart. Sequence-to-sequence model in DeepPOSE.

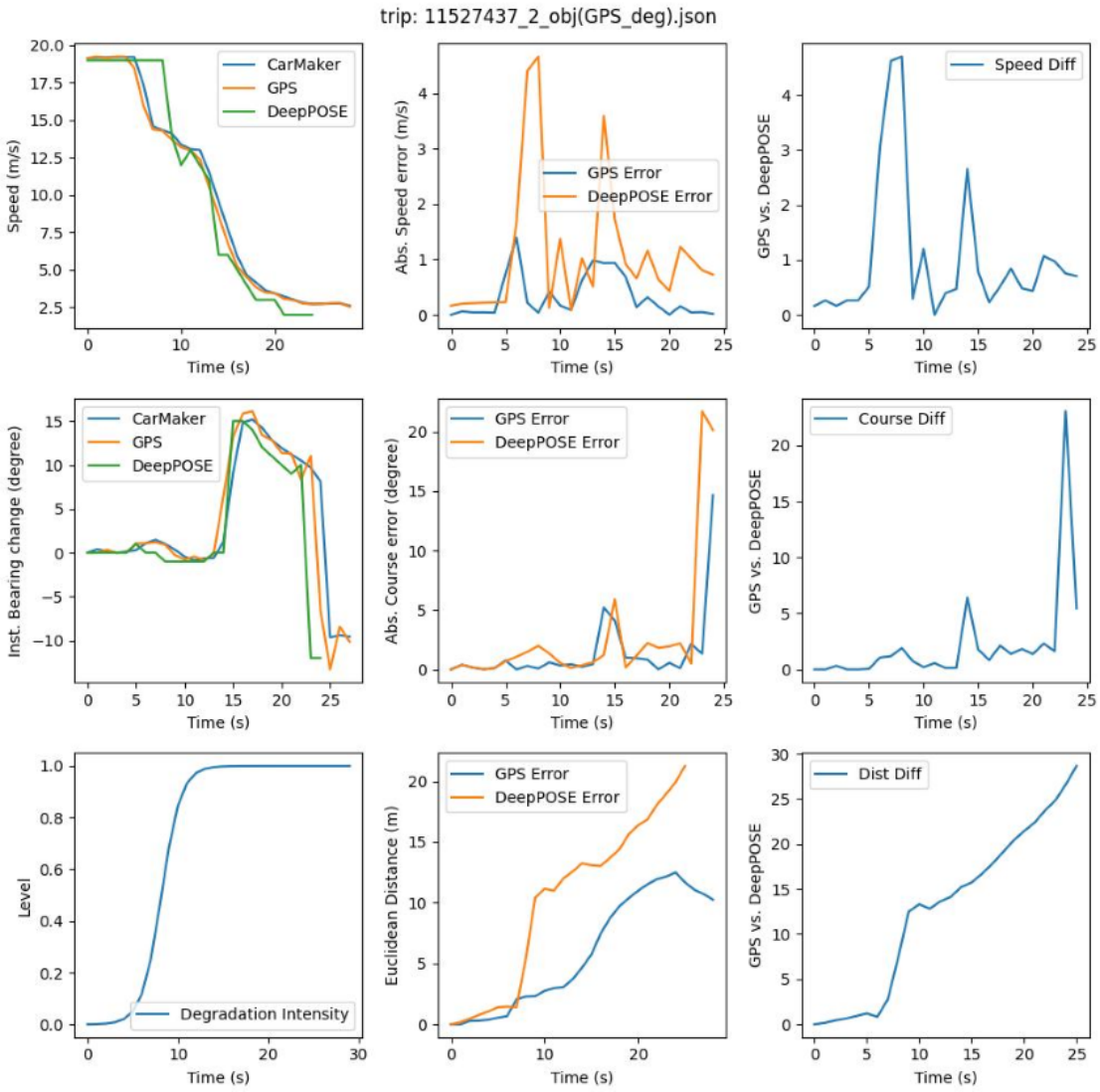

Figure 15. Line graphs. DeepPOSE results for rain effect.

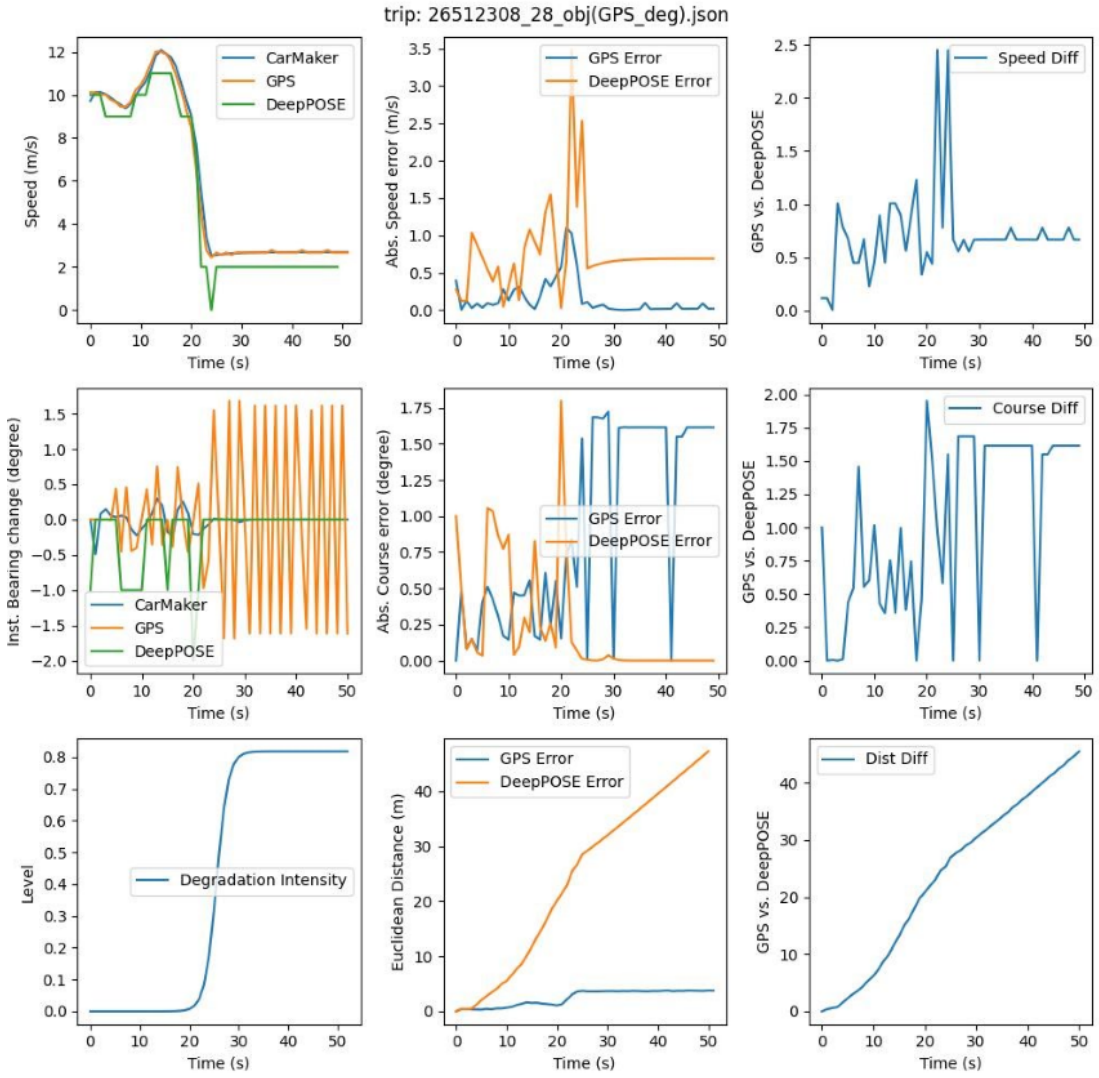

Figure 16. Line graphs. DeepPOSE results for GPS random variation.

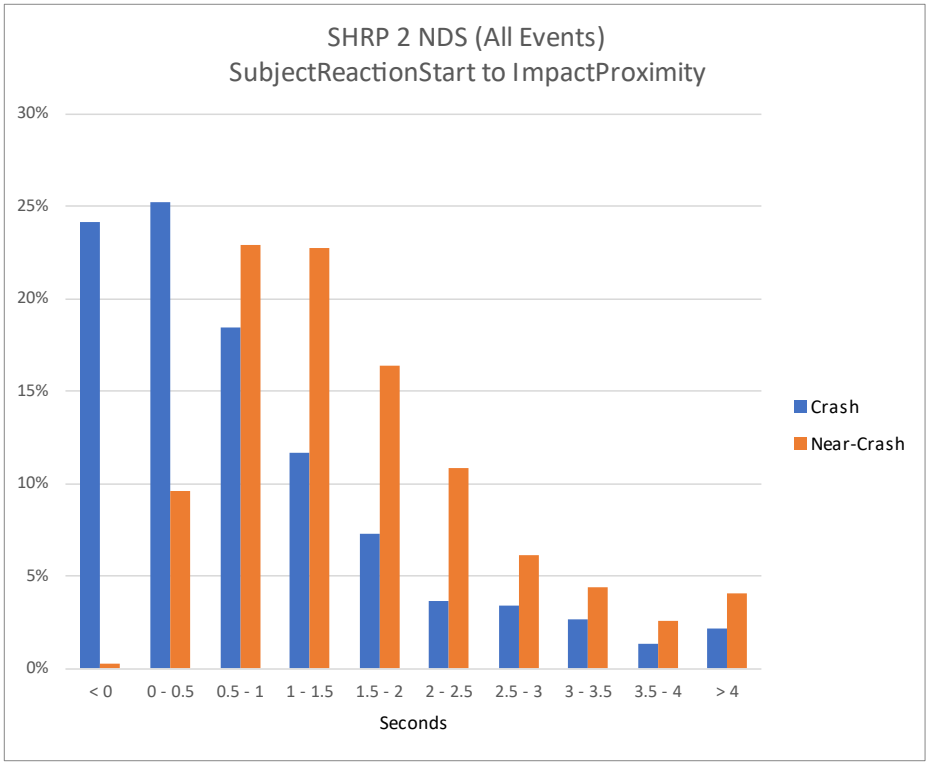

Figure 17. Bar graph. Seconds between Subject Reaction Start and Impact Proximity Time for all crash and near-crash events in the SHRP 2 NDS.

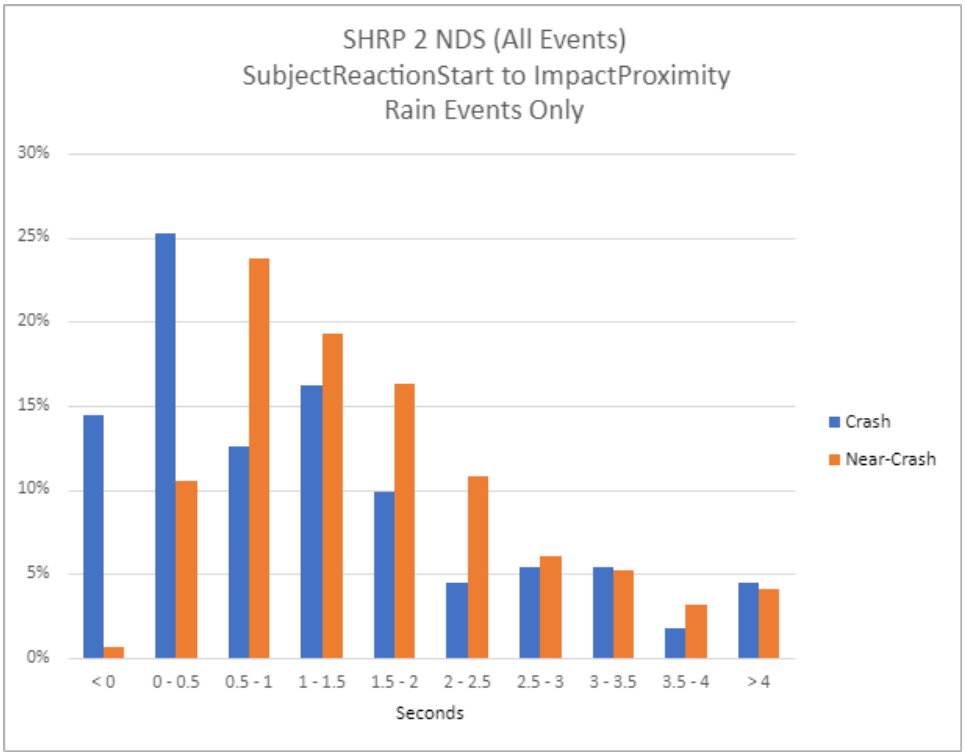

Figure 18. Bar graph. Seconds between Subject Reaction Start and Impact Proximity Time for crash and near-crash events in the SHRP 2 NDS with rain present.