Fault Diagnosis for the Perception Software in Highly Automated Vehicles and Reference Generation Based on World Model

Yao Hu[1]

[1]*General Motors, Warren, Michigan, 48092, USA*
yao.hu@gm.com

ABSTRACT

In a highly automated vehicle (HAV), the perception system performs critical sensing tasks such as object detection, scene understanding, etc. The perception software is based on complex intelligent algorithms and subjected to different failures such as missing object detection and false classification. It is a challenge to detect and identify these faults in the run time due to lack of ground truth and performance metrics. In this study, we introduce a method of perception software diagnostics, which is a generic method applicable to any state-of-the-art object detection models. In this method, perception results are compared with references generated with different sources, including pre-determined ground truth. The inconsistency between the perception results and the references, together with other performance metrics such as spatial variance, is used by a diagnostic logic to determine the fault type. Moreover, a method to generate references from a world model is presented. With validation, it is shown that the diagnostic algorithm provides a good performance in fault detection and isolation. This study enables fault mitigation in the run time and supports patch development in the development time.

1. INTRODUCTION

A highly automated vehicle refers to a vehicle with certain types of automation features including active safety, self-driving, etc. In the Society of Automotive Engineers (SAE) international standard J3016, automation is defined in six levels (from level 0 to level 5), and the term "highly automated" is listed as a definition from German Federal Highway Research Institute (BASt). It is equivalent to SAE level 3. In other documents, "highly automated vehicles" may be defined differently. American Association of Motor Vehicle Administrators (AAMVA, 2018) defines "highly automated vehicles" as vehicles of SAE level 3, 4 and 5. In this work, we use AAMVA's definition for highly automated vehicles.

The perception system is critical in a highly automated vehicle. It performs the task of collecting information and extracting relevant knowledge of the environment, which mainly mimics the function of human vision. Even in a level 0 or 1 vehicle, it may be implemented for active safety features such as lane departure warning and forward collision warning. The perception system's inputs are the environment information captured by different sensors, including cameras, radars, light detection and ranging (LIDARs), ultra-sonic sensors, global positioning system (GPS), inertial measurement unit (IMU), etc. The outputs of the perception system are the visual cognition results such as detection results of objects, which are fed to the downstream systems for localization, path planning, decision making and vehicle control (Brummelen, O'Brien, Gruyer, & Najjaran, 2018).

The perception software is implemented as the core of the perception system to convert the raw sensor data to visual cognition results. In a highly automated vehicle, the visual cognition tasks of the perception software include road detection (for lane marks, road surface, free space, etc.) (Pendleton, Andersen, Du, Shen, Meghjani, Eng, Rus, & Ang Jr., 2017), object detection and localization (for obstacles, surrounding vehicles, pedestrians, traffic signs, traffic lights, land marks, etc.), object behavior estimation (vehicle/pedestrian tracking, prediction of merging vehicles, etc.), scene understanding (traffic, event, etc.) (Janaia, Güneya, Behla, & Geigera, 2017), and localization (simultaneous localization and mapping (SLAM), pose estimation, ego-motion estimation) (Pendleton et al., 2017; Janaia et al., 2017), etc.

The perception software uses various computer vision and machine learning algorithms, including Canny edge detection (Canny, 1986), Hough Transform, supporting vector machine (SVM) (Dalal & Triggs, 2005), Histogram of Orientation (HOG) extraction (Dalal & Triggs, 2005), convolutional neural networks (CNN) (Janaia et al., 2017), optical flow approaches (Janaia et al., 2017), Kalman filter (Pendleton et al., 2017), particle filter (Pendleton et al., 2017), Markov

chain Monte Carlo (Zhu, Yuen, Mihaylova, & Leung, 2017), Scale-invariant feature transform (SIFT) (Zhu et al., 2017), etc. In recent years, deep learning approaches such as CNN have achieved great performance boosts (Pendleton et al., 2017). It's now widely used for object detection, semantic segmentation and scene understanding tasks (Ren, He, Girshick, & Sun, 2017; Redmon & Farhadi, 2018; Liu, Anguelov, Erhan, Szegedy, Reed, Fu, & Berg, 2016; Dai, Li, He, & Sun, 2016; He, Gkioxari, Dollár, & Girshick, 2017).

Failures in the perception system attract increasing attention due to their consequences related to safety (Brummelen et al., 2018). The failures of deep learning approaches have some uniqueness. A deep neural network (DNN) is trained with a training set of data, which has a finite coverage of cases in the real world. Those cases not covered by the training set are corner cases. When a corner case happens during a test, an incorrect result may be generated by the deep neural network. Such a fault may generate a failure in perception tasks, for example, a misdetection in an object detection task. In addition, corner cases can also happen within the training set. This is because the training method involves minimizing the mean of the loss function (which is the error between the results and the ground truth) over the whole training set. It is very difficult to make each training sample generate a correct result without overfitting, given that the size of the training set is relatively large.

In addition, there are other software failure modes that are common in general, such as computational faults, data faults, interface faults, exceptions, etc. These failure modes can be categorized in functions, locations (Gray, 1990), symptoms, behaviors (Wang, Quek, Rafacz, & Patel, 2004), causes (Chu, Martinez-Guridi, Yue, & Lehner, 2006), etc. Also, failure modes can be defined in different levels, from the system level (Gray, 1990) to the processor/memory operation level (Wang et al. 2004). We categorize the perception software failure modes in the system level in terms of behaviors/symptoms. Besides the DNN corner cases, which are the computational faults, we also consider data missing, data freeze, and timing faults in this study. Section 3 discusses more details.

To assess, detect and identify perception failures, methods are proposed in mainly three directions. The first direction is failure prediction based on the input of the perception system (Zhang, Wang, Farhadi, Hebert, & Parikh, 2014; Daftry, Zeng, Bagnell, & Hebert, 2016). These solutions calculate a probability of a failure or a confidence score of the perception with a given frame of input data, without checking the perception results. The second direction is perception system validation and verification (Dreossi, Donzé, & Seshia, 2019; Pei, Cao, Yang, & Jana, 2017; Pezzementi, Tabor, Yim, Chang, Drozd, Guttendorf, Wagner, & Koopman, 2018; Pei, Cao, Yang, & Jana, 2017). These solutions detect and identify failures during the validation and verification phase of developing the perception system. The last direction is runtime diagnostics of the perception system (Ramanagopal, Anderson, Vasudevan, & Johnson-Roberson, 2018; Dokhanchi, Amor, Deshmukh, & Fainekos, 2018). The solution detects and identifies the failure when the perception system is functioning.

The objective of this study is to develop solutions for runtime diagnostics of the perception system. We introduce a reference-based method to diagnose failures when they happen, instead of assessing failure prior. This type of solution is significant because of a few reasons. First, the runtime failures may cause direct consequence including safety concerns and negative user experience. From the literature, only a small portion of existing studies is targeted for runtime diagnostics with posterior assessment. Second, even with these runtime diagnostics methods, there are still various failures uncovered. For example, Ramanagopal et al. (2018) may not cover cases such as object missing in consecutive frames and simultaneous faults in stereo cameras. A reference-based method may better detect such a failure, but there is a lack of study in this direction. Due to the complexity of perception failures, it's very challenging to cover all failures with a single solution. Thus, we need a solution pool for perception diagnostics. Third, fault isolation is not discussed sufficiently in the literature. Existing studies are focused on neural network internal corner cases, while its upstream and inter-components failures may propagate and need to be taken into account. Our study is intended to reduce the gap in these three aspects.

In this work, we focus on the fault diagnostics method for the perception software of a specific perception task, which is object detection based on camera images. Section 2 introduces a popular object detection software, Faster R-CNN (region-based convolutional neural network). Section 3 introduces the fault injection methods for this study. Section 4 discusses the diagnostics algorithm. Section 5 discusses a method to generate references based on a world model. Section 6 presents the results.

2. FASTER R-CNN

In this study, we use Faster R-CNN as the target software to diagnose. Faster R-CNN (Ren et al., 2017) is a DNN model to detect objects in images. For a given image, Faster R-CNN outputs the information of each object, including the location represented by a bounding box, and probability scores of different object types. Faster R-CNN is one of the mainstream methods of object detection nowadays. Its inputs, outputs and internal variables are similar to other methods such as YOLO (You Only Look Once) (Redmon & Farhadi, 2018) and SSD (single shot multibox detector) (Liu et al., 2016), which allows the same diagnostics algorithm to be applied to any of them.

Faster R-CNN's input image first goes through the convolutional layers to generate the feature maps. Then the feature maps go through the region proposal network (RPN)

to generate proposals of objects. The proposals are also called regions of interest (RoI). In the RPN, anchors are predetermined as reference boxes of object proposals. Each anchor is centered at a predetermined position and has a predetermined size. The convolutional layers implement a box regressor and a classifier. The box regressor outputs the center coordinates, width and height for each proposal. The classifier outputs a score representing the probability of object or not object for each proposal. After the convolutional layers, duplicated and low-score proposals are filtered out.

To classify each RoI, the feature maps in each RoI are fed into the classifier based on fully connected layers. Since the RoI size is variant and the input size of the classifier is fixed, the RoI pooling layer divides each RoI into a fixed number of small pieces and extract the features of a fixed size. After RoI pooling, features of each RoI is processed by the classifier to determine the probabilities of different types.

With the understanding of Faster R-CNN, we inject faults and develop the diagnostics algorithm for the perception software.

3. FAULT INJECTION

In this study, we simulate different faults to develop the diagnostic algorithm. Even though some reports discuss the high-level failures of autonomous vehicle (such as Department of Motor Vehicles, 2019), there is a lack of comprehensive DFMEA (design failure mode and effect analysis) on perception software. Almost all the related studies in the literature are focused on the DNN corner cases. During our work of perception development, we experience additional faults of data missing, data freeze and timing. In this study, the faults injected are DNN corner cases, data missing, data freeze and timing faults.

DNN corner cases represents the DNN model limitation after training. Due to the limited size of the training sample set and the training method of stochastic gradient descent, it is possible that in certain cases the model generates wrong results. To simulate this fault, we use a pretrained model without fine tuning. The model is trained on a training set (PASCAL VOC 2007) (Everingham, Gool, Williams, Winn, & Zisserman, 2010) and tested in a different set, which is collected by our test vehicle, without adaptation. The difference between the two data sets generates the corner cases.

Data missing fault represents that the data is not presented at a certain layer of the DNN model. The root causes include communication error and memory crash. To simulate this fault, we assign the data to a matrix of all zeros at a given layer.

Data freeze is another data fault which may happen to the DNN model. It usually happens at the input layer, which causes the input frame of image not updating. The root cause may relate to the upstream sensor and communication. To simulate this fault, we assign the data of the previous frame to the current frame.

Timing fault represents that the DNN model runs slower than normal, including the whole model freeze. The root causes include memory leakage, memory lock, racing condition, etc. To simulate this fault, we insert a configurable delay during the execution of Faster R-CNN between two frames.

Based on these faults, we develop the perception diagnostics algorithm.

4. PERCEPTION DIAGNOSTICS ALGORITHM

In this section, we discuss details of the algorithm to detect and isolate of the perception faults of corner cases, data missing, data freeze and timing faults. To detect the corner cases, we develop the indicators of false positive rate and false negative rate. To detect data missing, we develop the indicator of spatial variance. To detect data freeze, we develop the indicator of temporal change. To detect the timing faults, we use the processing time of each frame as an indicator.

4.1. Pre-processing of Faster R-CNN Results

In this study, we use the Faster R-CNN pretrained with the PASCAL VOC 2007 data set. There are 21 classes of objects in total in PASCAL VOC 2007 data set (Everingham et al., 2010), including aeroplane, bicycle, background, etc. Each class is associated with an ID. For each object, Faster R-CNN generates 21 boxes and scores, where each box and score is associated with one class, respectively. A box represents the object position in current frame, and a score represents the probability of the object belonging to a class.

Some pre-processing is performed on the results of Faster R-CNN. First, we remove the results of "background" class in boxes and scores. Second, we perform non-maximum suppression (NMS) to eliminate the overlapped detections of objects. Last, we keep the detections of high confidence by filtering out the detections whose scores are lower than a threshold. We denote the detection results in one image as $\{(x^r_{1,i}, y^r_{1,i}, x^r_{2,i}, y^r_{2,i}, c^r_i)|i = 1,2, \dots, I\}$. I is the total number of objects detected in this image. i is the index of each object. $x^r_{1,i}, y^r_{1,i}, x^r_{2,i}$ and $y^r_{2,i}$ are the coordinates of the bounding box b^r_i. c^r_i is the class ID of the object associated with b^r_i.

4.2. False Positive and False Negative Rates

When a corner case happens, the observation is that an image generates incorrect perception results. An indicator of the corner case is the correctness of the perception results. To measure the correctness, we compare the perception results to certain references. The references include predetermined ground truth, results from other sensor channels, results from other vehicles, information from a centralized world model, etc. In Section 5, we will discuss generating references with

a world model. In this section, we assume the references are provided without specifying the exact sources.

The correctness of the results can be represented by false positive and false negative rates. False positive rate represents how much of the detection is not presented in the references. False negative rate represents how much of the references is missed in the detection. In this work, an assumption is that references are treated as ground truth. In reality, references may be incorrect if they are not determined by the ground truth. An incorrect reference may generate a wrong diagnostic result of false negative or positive. To improve the accuracy of references requires technologies such as using temporal information, fusion of object information from different sensors and vehicles, etc., which are not discussed in this work.

For a given frame, we denote the references as $\{(x^l_{1,j}, y^l_{1,j}, x^l_{2,j}, y^l_{2,j}, c^l_j) | j = 1,2,\dots,J\}$. J is the total number of references in the current frame. j is the index of each reference. $b^l_j = (x^l_{1,j}, y^l_{1,j}, x^l_{2,j}, y^l_{2,j})$ is the bounding box of reference j. c^l_j is the class ID of reference j. When calculating the false positive and negative rates, we use the intersection-over-union (IoU) rate to determine the correlation of two bounding boxes, b^r_i from the detection results and b^l_j from the references. Figure 1 shows an example of the IoU of two bounding boxes.

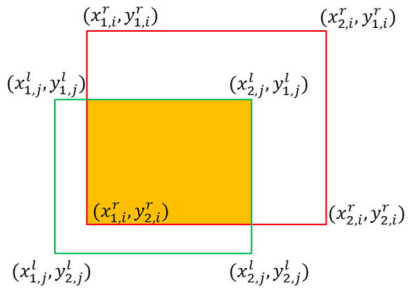


Figure 1. An example of the IoU of two bounding boxes, $b^r_i\ (x^r_{1,i}, y^r_{1,i}, x^r_{2,i}, y^r_{2,i})$ and $b^l_j\ (x^l_{1,j}, y^l_{1,j}, x^l_{2,j}, y^l_{2,j})$. The IoU is the rate of the intersection (the yellow area) divided by the union (the area covered by green and red boxes together).

We define the correlation of bounding boxes b^r_i and b^l_j as:

$$b\_cor_{i,j} = \frac{\text{the intersection area of } b^r_i \text{ and } b^l_j}{\text{the union area of } b^r_i \text{ and } b^l_j} \qquad (1)$$

Next, we determine the correlation between a class from the detection results and a class from the references. We use a look-up table to determine the correlation between two classes c^r_i and c^l_j. Table 1 shows an example of such a look-up table.

In Table 1, each row is associated with a class in detection results. Each column is associated with a class in the references. In general, the set of reference classes may not be the same as the set of classes of perception results. The values in the look-up table are pre-determined and configurable. For example, the correlation between a car and a car is 1; the correlation between a car and a truck is 0.9. Such a table provides the correlation $c\_cor_{i,j}$ between two given classes c^r_i and c^l_j.

Table 1. An example of the look-up table to determine class correlation.

c^r_i \ c^l_j	Car	Truck	Adult	Child	Bicycle	…
Car	1	0.9	0	0	0	
Bicycle	0	0	0	0	1	
Motorbike	0	0	0	0	0.5	
Person	0	0	1	1	0	
Boat	0	0	0	0	0	
…						

Now we calculate the false positive rate and false negative rate for each frame. The detection rate of reference j is:

$$cor_j = \max_i (b\_cor_{i,j} \cdot c\_cor_{i,j}) \qquad (2)$$

where $b\_cor_{i,j}$ is the correlation between b^r_i and b^l_j, and $c\_cor_{i,j}$ is the correlation between c^r_i and c^l_j.

Then, the rate of missed detection of object j is $1-cor_j$. The false negative rate of current frame is defined as the average rate of missed detection of all references, which is calculated by:

$$FN = \frac{1}{J}\sum_{j=1}^{J}(1 - cor_j) \qquad (3)$$

Similarly, the accuracy of a given detection i is:

$$cor_i = \max_j (b\_cor_{i,j} \cdot c\_cor_{i,j}) \qquad (4)$$

The false positive rate of current frame is the average false-detection rate of all detections:

$$FP = \frac{1}{I}\sum_{i=1}^{I}(1 - cor_i) \qquad (5)$$

Compared to mAP (mean average precision) used by many other studies, *FN* and *FP* use direct IoU values rather than comparing IoU with fixed thresholds. They provide a more precise measurement on the bounding box error. With the false positive and negative rates, we determine if a failure happens due to corner cases. Details are discussed in Subsection 4.6.

4.3. Processing Time

In addition to failures due to corner cases, for the timing related failures, we use the processing time of each frame as an indicator. We define the processing time t_k as the time used for Faster R-CNN to process frame k. Denote $T_{1,k}$ as the timestamp when frame k is fed to the input layer of Faster R-

CNN. Denote $T_{2,k}$ as the timestamp when the results of frame k is generated at the output layer of Faster R-CNN. The processing time is calculated by:

$$t_k = T_{2,k} - T_{1,k} \qquad (6)$$

This indicator is used to determine if a timing related fault happens. Details are discussed in Subsection 4.6.

4.4. Temporal Change

Another type of failure that we detect and isolate is the data freeze. We define an indicator, temporal change, to monitor the changes in data in Faster R-CNN between two consecutive frames.

For a given frame k, denote $d_{p,k}$ as the data at layer p of Faster R-CNN when finishing the processing of frame k. Notice that $d_{0,k}$ represents the input image of frame k. We can reshape $d_{p,k}$ into a vector. The temporal change is defined as the mean square error (MSE) of all the elements in $d_{p,k} - d_{p,k-1}$:

$$\Delta_{p,k} = \mathrm{MSE}(d_{p,k} - d_{p,k-1}) \qquad (7)$$

In Subsection 4.6, we check $\Delta_{p,k}$ to determine if a failure of data freeze happens.

4.5. Spatial Variance

The last type of failure to detect and isolate is the data missing. We define an indicator, spatial variance, to monitor the validity of data in Faster R-CNN.

For each layer before RPN, $d_{p,k}$ is a 3D matrix. The three dimensions are in the order of channel, vertical and horizontal dimensions. $d_{p,k}$ of ROI pooling layer has four dimensions: proposal, channel, vertical and horizontal, and is converted into a 3D matrix by stacking the proposal dimension in the channel dimension. For the rest of layers, $d_{p,k}$ is a 2D matrix, whose two dimensions are proposal and channel dimensions. We extend one dimension of size 1 to make $d_{p,k}$ a 3D matrix.

The spatial variance of $d_{p,k}$ is defined as the variance over the second and third dimensions. Denote $d_{p,k}(l,n,m)$ as an element in $d_{p,k}$. The spatial mean is calculated by:

$$\mu_{p,k,l} = \underset{m,n}{\mathrm{mean}}\{d_{p,k}(l,n,m)\} \qquad (8)$$

Then, the spatial variance is calculated by:

$$\sigma^2_{p,k,l} = \underset{m,n}{\mathrm{mean}}\left\{\left(d_{p,k}(l,n,m) - \mu_{p,k,l}\right)^2\right\} \qquad (9)$$

$$\sigma^2_{p,k} = \underset{l}{\mathrm{mean}}\{\sigma^2_{p,k,l}\} \qquad (10)$$

With this indicator, we determine whether the data $d_{p,k}$ is invalid. If $d_{p,k}$ is missing, then it is filled with default value, which is shown as an image of pure color such as black. In this case, $\sigma^2_{p,k}$ equals to 0. Spatial variance can also detect partial data fault. In our tests, images partially blocked by black patches have higher or lower spatial variance values compared to the normal images, depending on the patch size. In the next subsection, we discuss the details of using $\sigma^2_{p,k}$ for data missing fault diagnostics.

4.6. Fault Diagnostic Logic

We develop a logic to generate the diagnostic decisions based on indicators *FP*, *FN*, t_k, $\Delta_{p,k}$, and $\sigma^2_{p,k}$. The diagnostic decision includes the color codes for all four types of faults: corner case, data missing, data freeze and timing fault. Color codes red represents that a severe fault is detected. Green represents that no obvious fault is detected for a given type. Yellow represents that a low severity fault is detected, or the severity is unknown and needs further inspection. Notice that both low severity and unknown severity are coded as yellow, but they are two different severities.

The diagnostic program runs in parallel with the perception program. In each cycle of processing an image k, we time the execution of perception program and calculate the indicator t_k. The variables at each layer of the neural network is buffered to calculate other indicators. After generating the indicators, we follow the diagnostic logic shown in Figure 2. we first initialize the diagnostic decision of current frame k. In this step, we assign unknown severity to all four types of faults. Next, we check t_k against thresholds θ_0, θ_1 and θ_2. The thresholds satisfy $\theta_0<\theta_2<\theta_1$. Depending on the value of t_k, we assign the corresponding severity to the timing fault.

Next, we determine the data missing fault. If the spatial variance equal to 0 at a certain layer, we assign red to the data missing fault; otherwise, we assign green to the data missing fault. In the practical implementation, we compare $\sigma^2_{p,k}$ with a very small threshold to determine whether it is 0. Then, we check data freeze fault based on the temporal change $\Delta_{p,k}$. If the temporal change equals to 0 at certain layer, we assign red to the data freeze fault; otherwise, we assign green to this fault.

Last, we determine the severity of the corner case fault based on the false positive and negative rates. The thresholds satisfy $\theta_3<\theta_7<\theta_5$ and $\theta_4<\theta_8<\theta_6$. If both *FN* and *FP* are smaller than the corresponding thresholds θ_5 and θ_6, we assign green to the corner case fault; if either *FN* or *FP* is greater than thresholds θ_5 or θ_6 respectively, we assign red to the corner case fault; otherwise, if either *FN* or *FP* is greater than thresholds θ_7 or θ_8 respectively, we assign yellow to the corner case fault to indicate low severity; otherwise, the severity of corner case fault is unknown.

If a data missing fault is determined, we skip checking data freeze and corner case, since the data is invalid. Also, if a data freeze fault is determined, we skip checking corner case, since the data is invalid.

The thresholds are calibrated heuristically based on test samples. With the fault diagnostic logic, we determine the severity of each type of fault in perception software. These severities are output as the diagnostic decision for each

frame. Section 6 discusses the diagnostics results under different faults.

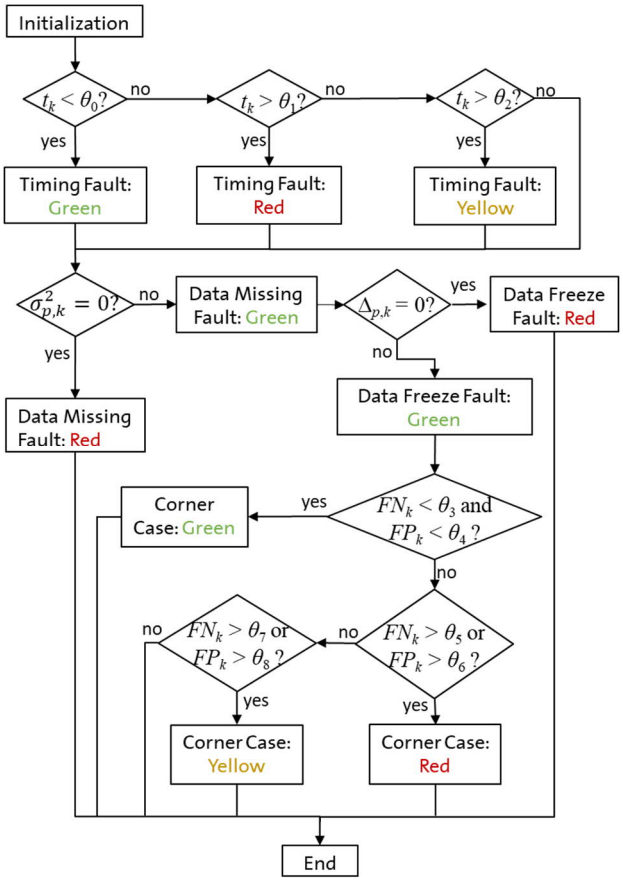


Figure 2. Fault diagnostic logic for perception software.

5. REFERENCE GENERATION BASED ON WORLD MODEL

In Section 4, we discuss the details of the perception diagnostic algorithms. To detect the corner case fault, we need references of the detected objects to calculate the false positive and negative rates. In this section, we present a method to generate such references based on a world model.

The world model in this study refers to a digital representation of the real-world environment. It generates a collection of data to represent the information of objects in the environment at any given time. The information of objects include class, location, pose, size, etc. A detailed world model may provide the information such as shape and surface texture of each object. Such a world model can generate synthesized sensor data such as images, which can be references for sensor diagnostics. A coarse world model may provide only the class, location, pose and size of each object, which is still sufficient to generate references of object detection results. The sources of information fed into the world model include pre-determined information of static objects, HD (high definition) map, perception data and results from other vehicles, information reported by objects themselves, information collected in history, information collected by dedicated agents, etc. Information from all these sources are aggregated and transferred using technologies such as V2V (vehicle to vehicle) and V2X (vehicle to infrastructure).

In this study, we build a world model based on our test dataset to provide the object information. For each object j in a given frame k, the world model provides the object class c_j^l, size (height h_j^l, width w_j^l and length l_j^l), position ($x_j^{l,w}$, $y_j^{l,w}$ and $z_j^{l,w}$) and pose (roll $\varphi_j^{l,w}$, pitch $\theta_j^{l,w}$ and yaw $\psi_j^{l,w}$). We label the test dataset to provide object position and pose with respect to a fixed world frame.

We generate bounding boxes references by converting the object information from the world model into bounding boxes in each frame. Since the camera field of view is relevant to the host vehicle, we use the host vehicle information to generate references. For frame k, we use the vehicle position (x^h, y^h and z^h) and pose (roll φ^h, pitch θ^h and yaw ψ^h), which are collected by the test vehicle during driving.

First, we convert the object position and pose into the coordinates and angles with respect to the coordinate system of the host vehicle. We convert the Euler angles into rotation matrices to represent the poses of host vehicle and each object (Mallick, 2016). The rotation matrix R_j^w is determined by $\varphi_j^{l,w}$, $\theta_j^{l,w}$ and $\psi_j^{l,w}$ to represent the pose of object j in the world frame. The rotation matrix R_h^w is determined by φ^h, θ^h and ψ^h to represent the pose of the host vehicle in the world frame.

The coordinates of the object with respect to the vehicle coordinate system is:

$$\begin{bmatrix} x_j^v \\ y_j^v \\ z_j^v \end{bmatrix} = (R_h^w)^{-1} \begin{bmatrix} x_j^{l,w} - x^h \\ y_j^{l,w} - y^h \\ z_j^{l,w} - z^h \end{bmatrix} \qquad (11)$$

The pose of the object with respect to the vehicle coordinate system, represented by a rotation matrix, is:

$$R_j^v = (R_h^w)^{-1} R_j^w \qquad (12)$$

Notice that we ignore the pitch and roll of each object with respect to the world and vehicle coordinate systems in this study, since they are very small in our test.

Second, we determine the object corner points positions with respect to the vehicle coordinate system. An object is represented by a 3-D box in space, which is represented by its eight corner points. To determine the coordinates of the eight corner points, the object coordinate system is defined by assigning the origin to the center point of the object. The corner points of the 3-D box are presented by the coordinates with respect to the object coordinate system:

$$B_j = \begin{bmatrix} \frac{-l_j^l}{2} & \frac{-l_j^l}{2} & \frac{l_j^l}{2} & \frac{l_j^l}{2} & \frac{-l_j^l}{2} & \frac{-l_j^l}{2} & \frac{l_j^l}{2} & \frac{l_j^l}{2} \\ \frac{w_j^l}{2} & -\frac{w_j^l}{2} & -\frac{w_j^l}{2} & \frac{w_j^l}{2} & \frac{w_j^l}{2} & -\frac{w_j^l}{2} & -\frac{w_j^l}{2} & \frac{w_j^l}{2} \\ \frac{-h_j^l}{2} & -\frac{h_j^l}{2} & -\frac{h_j^l}{2} & -\frac{h_j^l}{2} & \frac{h_j^l}{2} & \frac{h_j^l}{2} & \frac{h_j^l}{2} & \frac{h_j^l}{2} \end{bmatrix} \quad (15)$$

We convert coordinates of the corner points into the coordinates with respect to the coordinate system of vehicle:

$$\begin{bmatrix} B_j^v \\ 1 \end{bmatrix} = \begin{bmatrix} R_j^v & \begin{matrix} x_j^v \\ y_j^v \\ z_j^v \end{matrix} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \begin{bmatrix} B_j \\ 1 \end{bmatrix} \quad (16)$$

In this equation, each corner point is rotated by R_j^v and translated by $[x_j^v \quad y_j^v \quad z_j^v]^T$.

Third, we determine the object corner points positions with respect to the coordinate system of the camera. The test vehicle calibration provides the extrinsic camera calibration, which includes in translation vector t_v^{cam} and rotation matrix R_v^{cam} from the vehicle frame to the camera frame. Then, the object corner points in the camera coordinate system are:

$$\begin{bmatrix} B_j^{cam} \\ 1 \end{bmatrix} = \begin{bmatrix} R_v^{cam} & t_v^{cam} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} B_j^v \\ 1 \end{bmatrix} \quad (13)$$

In this equation, each corner point is rotated by R_v^{cam} and translated by t_v^{cam}.

Next, we use the method from OpenCV (2019) to convert each corner point $B_{j,k}^{cam} = [\, x_{j,k}^{cam}, y_{j,k}^{cam}, z_{j,k}^{cam} \,]^T$ into the coordinates $[u_{j,k}, v_{j,k}]^T$ in the image coordinate system. The test vehicle calibration provides the intrinsic parameters of the camera, including the focal lengths f_x and f_y, the principle point coordinates c_x and c_y, the radial distortion coefficients k_1, k_2 and k_3, and the tangential distortion coefficients p_1 and p_2. Refer to OpenCV (2019) for details of the intrinsic transform.

With the intrinsic transform, we determine u_j representing the horizontal coordinates of corner points in the image frame and v_j representing the vertical coordinates of corner points in the image frame. They are bounded by the image size.

Last, we generate bounding box references from the coordinates in the image frame:

$$x_{1,j}^l = \min\{u_j\},\ x_{2,j}^l = \max\{u_j\},$$
$$y_{1,j}^l = \min\{v_j\},\ y_{2,j}^l = \max\{v_j\}. \quad (19)$$

With these steps, we convert the object information from the world model into the references for perception diagnostics. In the next section, we discuss the results of perception diagnostics.

6. RESULTS

In previous sections, we introduce the perception diagnostic algorithm and the method of generating references from the world model. In this section, we discuss the results under different fault cases. As presented in Section 3, the fault cases included are DNN corner cases, data missing, data freeze and timing faults.

Due to the lack of similar work in the literature, we didn't find a common benchmark to evaluate a perception diagnostic method. In the two studies on runtime diagnostics of perception, Ramanagopal et al. (2018) used models trained with customized dataset (Sim200k) as the target systems to diagnose, while Dokhanchi et al. (2018) used a different model without providing statistic result over a dataset. The literature also shows a variety of definitions on error of object detection. Ramanagopal et al. (2018) used a fixed threshold on IoU, while Pezzementi et al. (2018) used flexible thresholds based on false positive rate.

In this study, all the tests are done in an environment with a GPU of Nvidia Quadro K2100M. The software is based on Faster R-CNN with a ZF model pretrained on VOC 2007 trainval and released on https://github.com/rbgirshick/py-faster-rcnn. Test data is collected from an instrumented test vehicle. 50 image frames are sampled from the video recorded by the front camera.

6.1. DNN Corner Cases

This subsection presents the results under DNN corner cases. Figure 3 shows the false negative and positive rates of each frame of the test data. Frame 4 generates one of the cases of false negative. Figure 4 shows the perception results, references from the world model and the top view of the world model.

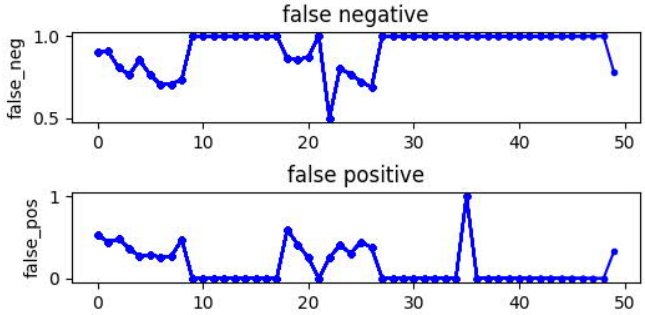


Figure 3. False negative and positive rates of each frame of the test data. The horizontal axis is the frame index.

There are six objects in the environment based on the world model. Five of them are in the field of view (FOV) of the camera, and only one car is detected by the perception software. The false negative rate is 0.8531. Based on the value of this indicator, we generate a red decision of corner case fault type. Meanwhile, the false positive rate is 0.2655

since the car detection is correct with a slight misalignment to the reference.

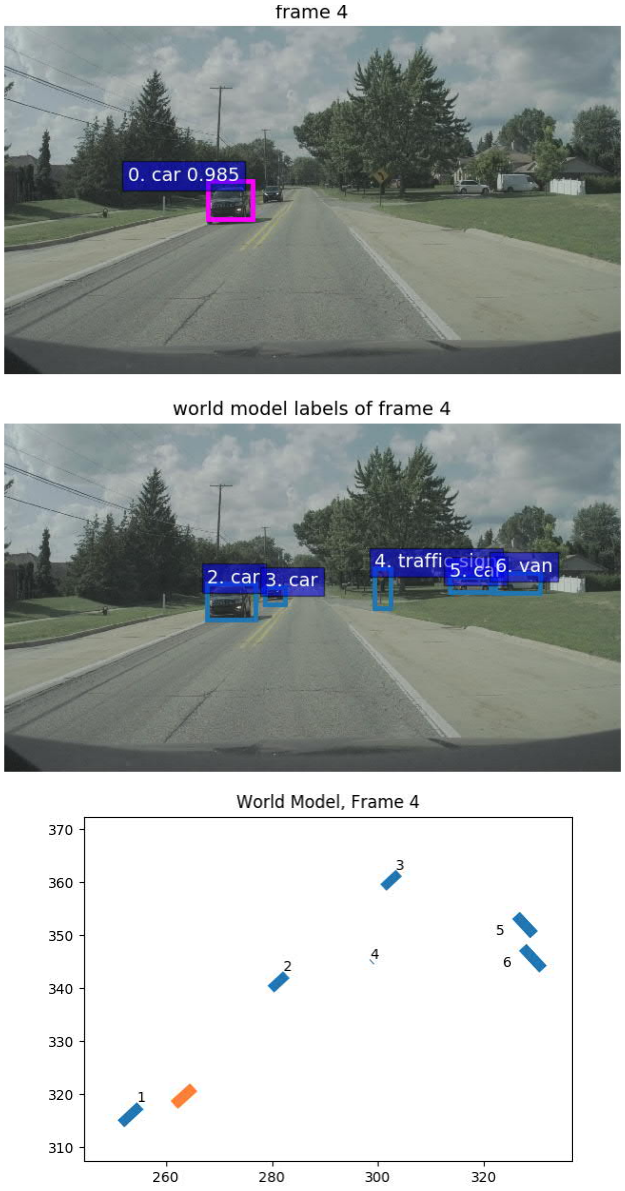


Figure 4. The perception results, references from the world model and the top view of the world model of frame 4.

Another corner case is identified in frame 35. The results are shown in Figure 5. With the assumption discussed in Subsection 4.2, we treat references as ground truth. In this frame, a potted plant is detected, which is not in the references. The false positive rate is 1, and generates a red decision of corner case fault type. The false negative rate is 1, since traffic signs are not detected by the perception software.

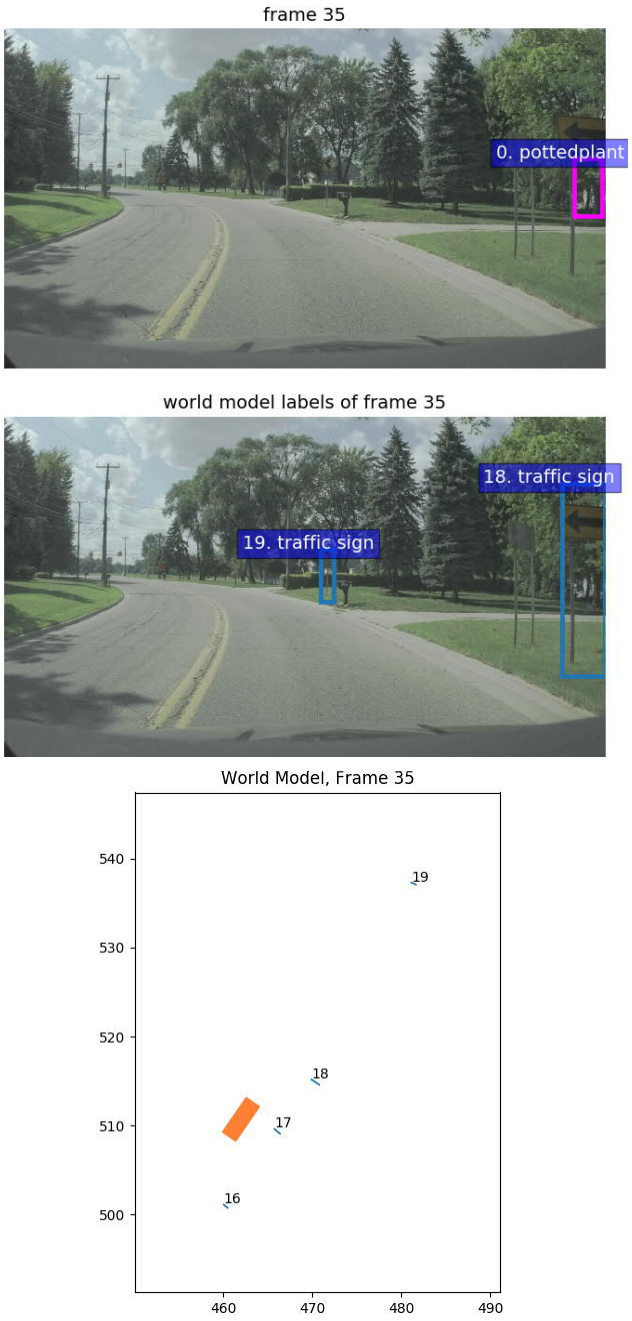


Figure 5. The perception results, references from the world model and the top view of the world model of frame 35.

6.2. Data Missing

This subsection presents the results under data missing fault. We simulate data missing by feeding a zero matrix as input to the DNN for frame 6. Thus, it doesn't generate any perception results, as shown in Figure 6.

The spatial variance of this input frame is 8.4207e-09. This indicates the input is invalid. The final decision is a red for the data missing fault. For corner cases and data freeze faults, the decision is yellow, since their evaluations are not applicable.

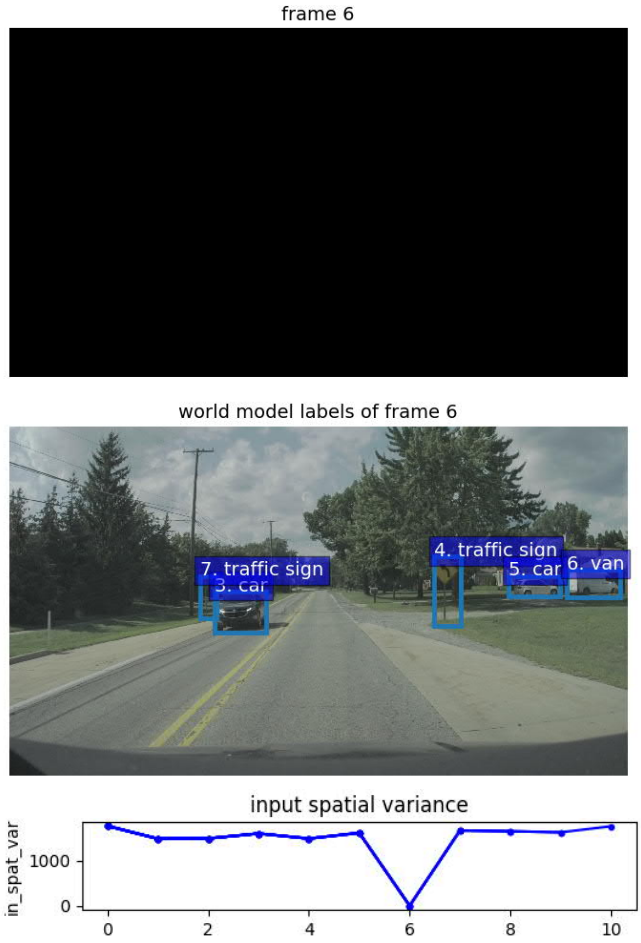


Figure 6. The perception results and references from the world model under a data missing fault. The horizontal axis of the input spatial variance chart is the frame index.

6.3. Data Freeze

This subsection discusses the results under data freeze fault. We simulate the fault by keeping the input frames not updating from frame 10 to frame 30. Thus, the input images from frame 11 to 30 are exactly the same as frame 10. The results are shown in Figure 7.

The results show that since frame 11, the temporal changes are 0 for input and output layers. Based on these indicators, the final decision is a red for the data freeze fault for these frames. The decision of the data missing fault is green, since the spatial variance is normal. The decision of the corner case fault is yellow, since the evaluation is inapplicable.

Frame 0 has relatively high temporal changes, since the input and output of frame 0 is compared with zero matrices instead of the previous frame.

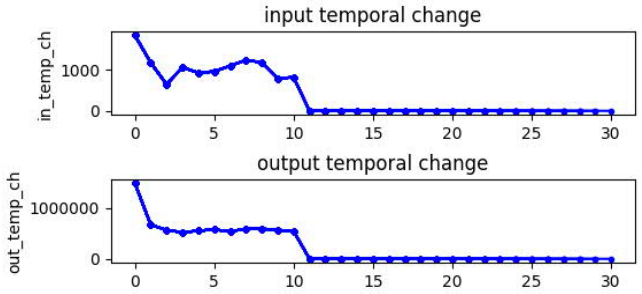


Figure 7. The temporal changes of input and output layers when a data freeze fault is injected from frame 10 to 30. The horizontal axis of these two curves is the frame index.

6.4. Timing Faults

This subsection discusses the results under timing faults. We inject a time delay when processing frame 10. The results are shown in Figure 8.

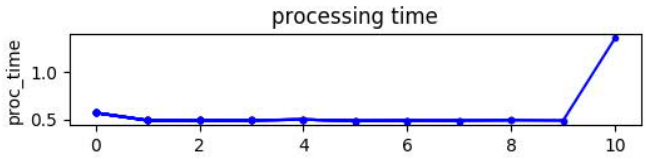


Figure 8. Processing time of each frame. A delay is injected at frame 10. The horizontal axis is the frame index.

The processing time of each frame from 1 to 9 is about 0.49 second, which indicates the regular time to process each frame. The processing time of frame 0 is 0.58 second. Some overhead time is used for initializing the memory. Frame 10 has a longer processing time than normal, which is 1.38 second. Based on this indicator, a red decision of timing fault is generated.

7. CONCLUSION

With the tests discussed in previous subsections, the perception diagnostics algorithm detects and identifies the correct fault types, and generates the corresponding diagnostic decision based on the values of indicators. The perception diagnostics algorithm can be used in vehicle for fault mitigation. It can also be used offline for perception software development, verification and validation.

The future work of perception diagnostics will be focused on diagnosis based on other references such as temporal information, scene information, pre-determined images, etc. They will cover the scenarios when the world model is not available.

REFERENCES

American Association of Motor Vehicle Administrators (AAMVA) (2018). *Jurisdictional guidelines for the safe*

testing and deployment of highly automated vehicles. Arlington, VA: American Association of Motor Vehicle Administrators.

Brummelen, J. V., O'Brien, M., Gruyer, D., & Najjaran, H. (2018). Autonomous vehicle perception: The technology of today and tomorrow. *Transportation Research Part C: Emerging Technologies*, vol. 89, pp. 384-406. doi:10.1016/j.trc.2018.02.012

Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8 (6), pp. 679-698. doi:10.1109/TPAMI.1986.4767851

Chu, T. L., Martinez-Guridi, G., Yue, M., & Lehner, J. (2006). A review of software-induced failure experience. *American Nuclear Society 5th Int. Meeting on Nuclear Plant Instrumentation Control and Human Machine Interface Technology*, November 12-16, Albuquerque, NM.

Daftry, S., Zeng, S., Bagnell, J. A., & Hebert, M. (2016). Introspective perception: Learning to predict failures in vision systems. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 9-14, Daejeon, South Korea. doi:10.1109/IROS.2016.7759279

Dai, J., Li, Y., He, K., & Sun, J. (2016). R-FCN: Object detection via region-based fully convolutional networks. In Lee D., Sugiyama M., Luxburg U., Guyon I., & Garnett R. (Eds), Advances in neural information processing systems 29 (pp. 379-387). Red Hook, NY: Curran Associates, Inc.

Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, June 20-25, San Diego, CA. doi:10.1109/CVPR.2005.177

Department of Motor Vehicles (DMV) (2019). *2019 Autonomous Vehicle Disengagement Reports.* https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/disengagement-reports/

Dokhanchi, A., Amor, H. B., Deshmukh, J. V., & Fainekos, G. (2018). Evaluating perception systems for autonomous vehicles using quality temporal logic. In Colombo C., Leucker M. (Eds.), *Runtime verification* (pp. 409-416). Cham: Springer.

Dreossi, T., Donzé, A., & Seshia, S. A. (2019). Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, vol. 63, pp. 1031-1053. doi:10.1007/s10817-018-09509-5

Everingham, M., Gool, L. V., Williams, C. K. I., Winn, J., & Zisserman, A. (2010). The PASCAL visual object classes (VOC) challenge. *International Journal of Computer Vision*, vol. 88, pp. 303–338. doi:10.1007/s11263-009-0275-4

Gray, J. (1990). A census of Tandem system availability between 1985 and 1990. *IEEE Transactions on Reliability*, vol. 39 (4), pp. 409-418. doi:10.1109/24.58719

He, K., Gkioxari, G., Dollár, P., & Girshick, R. (2017). Mask R-CNN. *2017 IEEE International Conference on Computer Vision (ICCV)*, October 22-29, Venice, Italy. doi:10.1109/ICCV.2017.322

Janaia, J., Güneya, F., Behla, A., & Geigera, A. (2017). Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *arXiv preprint arXiv:1704.05519*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. C. (2016). SSD: Single shot multibox detector. In Leibe B., Matas J., Sebe N., & Welling M. (Eds.), *Computer vision - ECCV 2016* (pp. 21-37). Cham: Springer. doi:10.1007/978-3-319-46448-0_2

Mallick, S. (2016). Rotation matrix to euler angles. https://www.learnopencv.com/rotation-matrix-to-euler-angles.

OpenCV (2019). calib3d. Camera calibration and 3D reconstruction. In *The OpenCV reference manual, Release 2.4.13.7*. OpenCV.

Pei, K., Cao, Y., Yang, J., & Jana, S. (2017). DeepXplore: Automated whitebox testing of deep learning systems. *Communications of the ACM*, vol. 62 (11), pp. 137-145. doi:10.1145/3361566

Pei, K., Cao, Y., Yang, J., & Jana, S. (2017). Towards practical verification of machine learning: The case of computer vision systems. *arXiv preprint arXiv:1712.01785v3*.

Pendleton, S. D., Andersen, H., Du, X., Shen, X., Meghjani, M., Eng, Y. H., Rus, D., & Ang Jr., M. H. (2017). Perception, planning, control, and coordination for autonomous vehicles. *Machines*, vol. 5 (1), 6. doi:10.3390/machines5010006

Pezzementi, Z., Tabor, T., Yim, S., Chang, J. K., Drozd, B., Guttendorf, D., Wagner, M., & Koopman, P. (2018). Putting image manipulations in context: Robustness testing for safe perception. *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, August 6-8, Philadelphia, PA. doi:10.1109/SSRR.2018.8468619

Ramanagopal, M. S., Anderson, C., Vasudevan, R., & Johnson-Roberson, M. (2018). Failing to learn: Autonomously identifying perception failures for self-driving cars. IEEE Robotics and Automation Letters, vol. 3 (4), pp. 3860-3867. doi:10.1109/LRA.2018.2857402

Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*.

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39 (6), pp. 1137-1149. doi:10.1109/TPAMI.2016.2577031

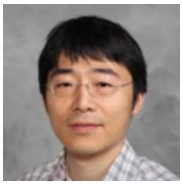
Wang, N. J., Quek, J., Rafacz, T. M., & Patel, S. J. (2004). Characterizing the effects of transient faults on a high-

performance processor pipeline. *International Conference on Dependable Systems and Networks*, June 28 - July 1, Florence, Italy. doi:10.1109/DSN.2004.1311877

Zhang, P., Wang, J., Farhadi, A., Hebert, M., & Parikh, D. (2014). Predicting failures of vision systems. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, June 23-28, Columbus, OH. doi:10.1109/CVPR.2014.456

Zhu, H., Yuen, K., Mihaylova, L., & Leung, H. (2017). Overview of environment perception for intelligent vehicles. *IEEE Transactions on Intelligent Transportation Systems*, vol. 18 (10), pp. 2584-2601. doi:10.1109/TITS.2017.2658662

BIOGRAPHIES



Yao Hu received a B.Sc. in electrical information science and technology from Fudan University, Shanghai, China, in 2004. He received a M.Eng. in system and circuit from Shanghai Institute of Technical Physics, Chinese Academy of Sciences, Shanghai, China, in 2007. He received a Ph.D. in electrical engineering from University of Kentucky, Lexington, KY, in 2013. He has been working at General Motors Global R&D Center, Warren, MI, since 2016, and currently holds the senior researcher position in the vehicle system research lab. His research interests include vehicle health management, perception, autonomous driving, deep learning and data analytics.