Shared Reality: Detecting Stealthy Attacks Against Autonomous Vehicles

Raul Quinonez
The University of Texas at Dallas
Richardson, Texas, USA
raul.quinonez.t@gmail.com

Sleiman Safaoui
The University of Texas at Dallas
Richardson, Texas, USA
sleiman.safaoui@utdallas.edu

Tyler Summers
The University of Texas at Dallas
Richardson, Texas, USA
tyler.summers@utdallas.edu

Bhavani Thuraisingham
The University of Texas at Dallas
Richardson, Texas, USA
bxt043000@utdallas.edu

Alvaro A. Cardenas
University of California Santa Cruz
Santa Cruz, California, USA
alacarde@ucsc.edu

ABSTRACT

Autonomous Vehicles (AVs), also known as self-driving cars, are becoming more prevalent in our daily lives. AVs rely on sensor information to evaluate their environment and make crucial decisions in real-time, however, new attacks can create false sensor and actuation commands. As technological advancements expand the usage of AVs to perform more complex tasks, it is imperative to secure the integrity of these devices against malicious external tampering. In this paper, we propose a security framework we call *Shared Reality*, which consists of verifying that sensors perceive the same physical reality. We implement our design on a custom hardware platform that uses the popular Robot Operating System (ROS) software. Our experiments show that AVs utilizing our proposed security framework ensured security with low overhead while performing several autonomous tasks.

CCS CONCEPTS

• **Computer systems organization** → **Sensors and actuators**; **Embedded software**.

KEYWORDS

Autonomous Vehicles (AVs), Extended Kalman Filter (EKF), Data Fusion, Sensor Redundancy, Security

ACM Reference Format:
Raul Quinonez, Sleiman Safaoui, Tyler Summers, Bhavani Thuraisingham, and Alvaro A. Cardenas. 2021. Shared Reality: Detecting Stealthy Attacks Against Autonomous Vehicles. In *Proceedings of the 2nd Workshop on CPS&IoT Security and Privacy (CPSIoTSec '21), November 15, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3462633.3483981

1 INTRODUCTION

The Society of Automotive Engineers defines six levels of automation for ground vehicles, ranging from no automation (level 0) to fully automated with no human interaction required (level 5) [32]. Currently, consumers are exposed to level 2 vehicles in the market (partial automation) with manufacturers promising level 3 (conditional automation) and beyond in the near future [27]. Many industries recognize Autonomous Vehicles (AVs) as an important component in the future of transportation and are investing heavily in AV research and development. For example, Uber and General Motors each have spent over 1 billion dollars on research and development in automation technology [8].

As AVs become more ubiquitous, we need to guarantee their security and safety against increasingly sophisticated attacks because of the physically hazardous consequences these machines will increasingly impose in the real world. For example, AVs have been involved in several incidents in which pedestrians were not recognized by the autopilot, or the system interpreted its environment incorrectly, resulting in fatalities [28, 50].

AVs rely on a wide variety of sensors [25] to evaluate their physical world including IMUs (Inertial Measurement Unit), cameras, LiDARs, RADARs, ultra-sonic sensors, and GPS. While these sensors measure different physical properties of the environment, they are susceptible to malicious tampering and transduction attacks [20]. Malicious tampering occurs when an attacker injects data into the system with the goal of causing disruptions or even hijacking the system. On the other hand, transduction attacks leverage the physical properties of sensors (e.g., accelerometers are susceptible to acoustic injection attacks [46]) to tamper their measurements. This reliance on sensor information to perform critical tasks without verification can cause AVs to become vulnerable to targeted attacks.

Researchers have demonstrated the feasibility of these types of attacks against consumer vehicles on the market. Mobileye 630 PRO and Tesla Model X (HW 2.5) autopilots can be manipulated to incorrectly recognize phantom projected images as street signs and pedestrians, thereby triggering the automatic breaking system or diverting the vehicle into the oncoming traffic lane [34]. Furthermore, Tesla's autopilot has also been demonstrated to be vulnerable to attacks targeting the activation of the windshield wipers and lane recognition system [26].

To solve these problems, there has been extensive research to avoid, detect, and prevent sensor tampering in real-time. Physics-Based Attack Detection algorithms [22] represent a promising way to detect sensor and actuator tampering attacks against autonomous vehicles. The main idea behind these techniques is to check the consistency between the control actions sent to actuators, and the received sensor measurements. Any discrepancy between expected vs. observed behavior is then tested statistically over time and if the deviations are statistically significant, then the anomaly detector raises an alarm. While Physics-Based attack detection has several benefits, they are not infallible since they can be attacked by injecting a small perturbation that is not significant at each time step, and are therefore undetectable, but that over time they cause

CPSIoTSec '21, November 15, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8487-2/21/11.
https://doi.org/10.1145/3462633.3483981

significant deviations to the operation of the system. We call these types of attacks *small but persistent* attacks.

In order to address these concerns, this paper introduces a framework called *Shared Reality*, which focuses on verifying that different sensors are perceiving the same physical world. Our solution is able to detect *slow* attacks that are able to bypass current state-of-the-art physics-based anomaly detection systems for autonomous vehicles [17, 38]. To demonstrate its effectiveness, we implement our approach in an advanced automation application called vehicle platooning (leader-follower).

Our contributions include,

- We demonstrate how *slow but persistent* attacks on individual sensors can bypass physics-based anomaly detection systems in AVs like SAVIOR [38].
- We present our *Shared Reality* concept as a computationally light anomaly detector for *small but persistent* attacks that monitors physical data across multiple sensors to detect attacks.
- We propose an augmented secure pipeline that leverages the individual strengths of SAVIOR [38] and our *Shared Reality* proposal to secure a vehicle to a wider range of attack patterns from *slow but persistent* to *fast and abrupt*.
- We demonstrate the operation of our *Shared Reality* proposal in a platoon system, using open-source hardware and software, where each vehicle detects the lane and vehicle ahead of it and adjusts its steering and throttle to maintain a constant distance.
- We analyze the performance of SAVIOR and *Shared Reality* modules to highlight the strengths of each method and the advantage of combing them.

The rest of the paper is organized as follows. Section 2 provides background information on mathematical models for ground vehicle movement, platooning, and threat modeling. In Section 3, we describe the overall design of *Shared Reality*. Section 4 discusses the implementation of *Shared Reality* in a set of platooning autonomous vehicles. Section 5 provides an evaluation of the secure platoon algorithm. Finally, section 7 concludes our work.

2 BACKGROUND

2.1 Threat Model and Assumptions

We assume an adversary that can inject false signals in one of the sensors used by AVs. For example, AVs typically use sensors like Cameras, IMUs, GPS receivers, RADARs, LiDARs, or ultrasonic sensors. Unfortunately, all of these sensors are vulnerable to transduction attacks including IMU [43, 46, 48], RADAR [55], LiDAR [12, 37, 42], ultrasonic [55], and camera [18, 37, 55] sensor measurements. The threat model in our paper is similar to the threat model in all of these previous research efforts.

In this paper, we focus on attacks against two of the most important sensors for autonomous vehicle navigation, cameras, and LiDAR. The camera is the primary sensor involved in the navigation of the vehicle and it is also used for detection and recognition tasks as well. A LiDAR consists of an infrared (IR) emitter and receiver rotating about a vertical axis. The sensor measures the time it takes to receive an emitted signal after it bounces back from an object and calculates the distance to that object. This allows the vehicle to acquire distance information about its surroundings at a specific height but in a range of directions.

Attacks against cameras and LiDAR sensors can range from physically placing stickers on the road, to driving an attack vehicle near to the target. For example, a dirty road patch can be a physical-world attack against the camera being used for lane-keeping assist [41]: in this setting, the attacker prints a malicious perturbation on asphalt, rubber, or posters, and then places it on the road, causing a vehicle to drive off lane boundaries [41]. Similarly, a vehicle in front of the target vehicle can spoof LiDAR signals causing the vehicle to perceive nonexisting obstacles or ignore existing ones [12].

Our goal is to detect *slow but persistent* attacks: where the fake sensor signal will *drift slowly over time*. Our work combines a *Shared Reality* and a physics-based attack detection system based on SAVIOR [38] (our previous work) as the combination of both can secure anomaly detection to *slow but persistent* as well as *abrupt* attacks.

We do not replace sensor fusion by our *Shared Reality*. Our goal is to use *Shared Reality alongside sensor fusion* algorithms to detect attacks and improve the vehicle's performance.

2.2 Platooning

We implement and test our *Shared Reality* proposal in the setting of vehicle platooning. Vehicle platooning, or simply platooning, is the concept of systematically interconnecting two or more vehicles. This vehicle-to-vehicle (V2V) connectivity allows for vehicles to drive at the same speed with minimal distance in between—distances so small that only autonomous vehicle technology can achieve them while still being safe. The average human driver takes 1.5 seconds to detect and react to road hazards [33]; such delays are minimized when automation is involved allowing to reduce this inter-vehicular distance. The interactions of vehicles in platoons are depicted in Fig. 1.

Another important benefit of platooning is the reduced consumption of fuel for vehicles. These savings are attributable in part to the wind drag that is offset by the leading vehicle in the platoon nexus. By following the leading vehicle closely, the following vehicles reduce the impact of wind resistance that they would otherwise have to combat and therefore consume less fuel. The fuel reduction is significant especially for larger vehicles [5, 30]; for a platoon of trucks, it improves fuel consumption by 10% on average [47]. The increase in safety and efficiency becomes more notable considering that it could be expanded to over 65% of haul transport [1]. With surface freight transport expected to increase dramatically in the coming decades, platooning can have a major impact on fuel savings for large vehicle fleets and on transport capacity [5].

There are several use-cases of platooning currently under development, like package delivery and supply chain (UPS is testing truck platooning technologies on highways [9]), next-generation combat vehicles to reduce exposure of soldiers in unsecured corridors, driver-assisted buses with shorter following distances to meet the demand for transportation in congested areas such as the Lincoln tunnel between New York and New Jersey, and platooning for timber transport on back-country roads, as the forest industry suffers from driver shortages in moving cut timber to mills [9].

Platooning in its simplest form can be decoupled into two subtasks: 1) detect and follow a path, and 2) detect and maintain a constant distance to the vehicle ahead. We thus divide the actuation between these two tasks. The path detection task determines where the vehicle is relative to a path and controls the steering of the vehicle, while the distance control task measures the distance between vehicles and adjusts the speed.

2.3 Physical Model of Vehicle

Physics-based attack detection systems need accurate models of the behavior of a physical phenomena [22]. While extracting models of physical phenomena is complicated and sometimes requires advanced system identification tools, the physical behavior of ground

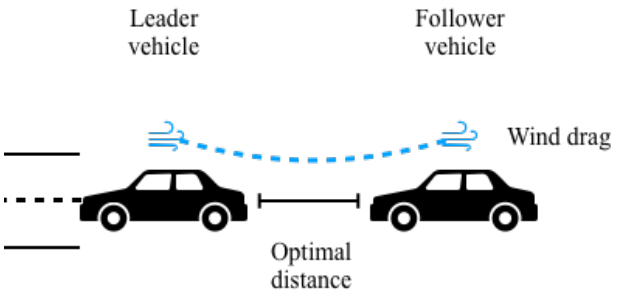

Figure 1: Vehicle Platooning. Communication and sensing between vehicles allow them to drive closely together minimizing fuel consumption and road space usage.

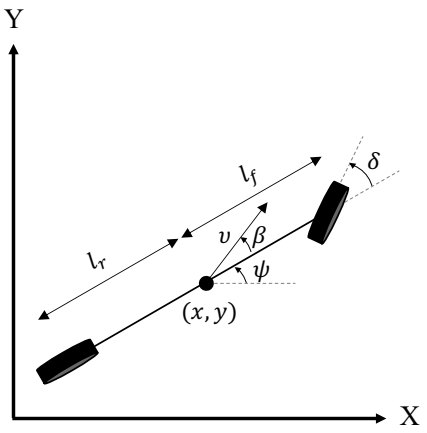

Figure 2: Bicycle model for ground AV.

vehicles is a fairly well-understood process. In particular, we use the well-known bicycle model that captures the behavior of a four-wheel vehicle by the following non-linear equations [39]:

$$\begin{cases} \dot{x} = v\cos(\psi+\beta) \\ \dot{y} = v\sin(\psi+\beta) \\ \dot{\psi} = \frac{v}{l_r}\sin(\beta) \\ \dot{v} = a \\ \beta = \tan^{-1}(\frac{l_r}{l_f+l_r}\tan(\delta)) \end{cases} \quad (1)$$

where (x, y) is the position of the vehicle's center of mass, ψ is the orientation of the vehicle relative to the x-axis, v is the vehicle's speed, β is the direction relative to the vehicle's heading, δ is the steering angle relative to the vehicle's heading, a is the acceleration, and l_r and l_f are the distances from the center of the vehicle to the rear and front wheels respectively ($l_r + l_f$ is the wheelbase). These parameters are illustrated in Fig. 2.

The control inputs are the steering angle δ and the acceleration a. The values that we need to estimate from the various sensors include the speed v, orientation (yaw angle) ψ, and position pair x, y. Note that we adopt a frame that moves along the line. The y-axis always points in the direction of the curve (tangent to it) at a constant offset from the AV, and the x-axis is perpendicular to that (normal to the curve).

2.4 Physics-Based Attack Detection

Classical security mechanisms such as software security, memory protection, authentication, or cryptography are not enough to protect sensors against physical and transduction attacks [20]. In order to identify these new attacks, there is growing interest in Physics-Based Attack Detection (PBAD) [22].

PBAD consists of two steps: the first step is performed off-line and extracts physical invariants of the system to create a model that captures the expected correlations between sensors (also known as sensor fusion) and between actuators and sensors (i.e., between the inputs and the outputs to the system). For autonomous vehicles, these physical invariants correspond to Equation (1). The second step is an online anomaly detection algorithm that compares predictions with observed states and raises an alarm when the accumulated discrepancy between predicted and observed states exceeds a threshold. PBAD has been explored in water control systems [4, 23], state estimation in the power grid [19, 31], chemical processes [6, 13], and in particular, autonomous vehicles [17, 38].

One of the key weaknesses of PBAD is that it is vulnerable to stealthy attacks [49] (what we call *slow but persistent* attacks in this paper). A fundamental reason for the existence of stealthy attacks is that any control of a physical system would not need sensors if we knew exactly the physical evolution of the process given the control commands (this is called open-loop control). Meanwhile, almost all control algorithms run in "closed-loop" because model uncertainties and perturbations prevent us from knowing exactly the evolution of a physical process. This uncertainty allows malicious users to create attacks that behave seemingly like the physical process under control, but create a small deviation that over time can be catastrophic. Unfortunately, none of the prior PBAD efforts on autonomous vehicles has considered a robust solution to detect these slow but persistent attacks [17, 38]. In this paper we show how combining a PBAD mechanism with our proposed Shared Reality algorithms, can detect these slow but persistent attacks and a wide variety of other attacks.

3 SHARED REALITY DESIGN

While our concept of a shared reality can be applied to a variety of settings, we focus in this paper on a particular instantiation of our idea by focusing on how the camera and LiDAR sensors in a vehicle produce shared information about the environment in a platooning scenario.

Intuitively, our *Shared Reality* proposal ensures that all cross-coupled data remains roughly the same over time. This data corresponds to the same physical object if an attack is initiated on one sensor (e.g. the front vehicle is depicted to drift to the right in the camera) then the second sensor will not detect this (the LiDAR should notice this inconsistency), causing the error measurement across the cross-coupled data to increase. Even if the drift is small, thanks to the *slow but persistent attacks*, this error will accumulate over time if only one sensor is under attack.

We also note that the choice of cross-coupled data is a design choice. For our platoon system, we chose vertical and lateral offsets because they are specific to the application and persistent. However, the choice can be generalized to any other cross-coupled physical data. A few potential examples include: lane marks detected by multiple front-facing cameras, estimated vehicle speed from encoders and IMU or a SLAM (simultaneous localization and mapping) algorithm, detected vehicles via LiDAR and RADAR.

Our design is illustrated in Fig. 3. The main idea is to use two independent physics-based anomaly detection algorithms for each sensor (camera and LiDAR), and in parallel, compute a metric in

the shared reality node that denotes the similarity and consistency between the perception of each sensor. We implement SAVIOR [38] as a reference physics-based attack detection system and adapt it to LiDAR data (the original work only considered the camera sensor).

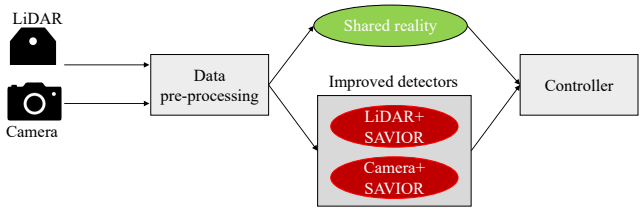

Figure 3: We use an anomaly detector on both LiDAR and camera data to detect malicious input on either sensor. If one of the sensors is compromised, the system will ignore it and navigate using the remaining untampered sensors.

We now describe how we measure this consistency between sensors in a platooning scenario. Notice that the distance between a reference vehicle and the leading vehicle (the vehicle in front) and the lateral offset of the leader relative to the reference vehicle can be estimated by a camera and a LiDAR. These measurements are the basis of the *Shared Reality* module since they describe common physical data between the two sensors that should be closely related. These variables are represented by $\delta_l, \delta_c, l_l, l_c$ which corresponds to the vertical distance measured by the LiDAR and camera, and the lateral offset measured by the LiDAR and camera respectively (see Fig 4). Using that information, we define two error measurements

$$\Delta_t = \delta_l - \delta_c \tag{2}$$

$$\Delta_l = l_l - l_c \tag{3}$$

which denote the difference in vertical distance and lateral offset measured by the two sensors respectively.

The *Shared Reality* module monitors how Δ_t and Δ_l change to indicate the presence of an attack. As the vehicle moves, the width may change. However, since we operate on a flat surface, the height of the front vehicle is invariant to those changes. Since the left and right sides of the front vehicle vary with rotation, we define the height as ${}^i h = abs({}^iUC_y - {}^iLC_y)$ where ${}^iUC_y$ is the height of the center of the top side (${}^iUC = ({}^iUL + {}^iUR)/2$) and ${}^iLC_y$ is the height of the center of the bottom side (${}^iLC = ({}^iLL + {}^iLR)/2$). We also know the height of the vehicle ($side_{meters}$). We can thus calculate the distance between the camera and the front vehicle as $\delta_c = \frac{side_{meters}}{{}^i h} f m_y$. The distance is regulated to a desired distance δ_{des} through a proportional controller and the velocity control is calculated as $ctrl_{vel} = K_{pv}(\delta_{des} - \delta_c)$. $ctrl_{vel}$ is used in the bicycle model as the acceleration and thus controls velocity. Note that leader detection can also be done with cameras using neural networks. The vision-based distance can be estimated in several ways. These include triangulation using multiple cameras or exploiting known vehicle width and height data.

In order to keep track of the history of discrepancies over time, we select the CUSUM algorithm inside of *Shared Reality* to keep track of the vertical distance and the lateral offset.

3.1 CUSUM Algorithm

One of the key components in our anomaly detection pipeline is CUmulative SUM control chart (CUSUM): an algorithm for monitoring and change detection. During each iteration at state k, the residual

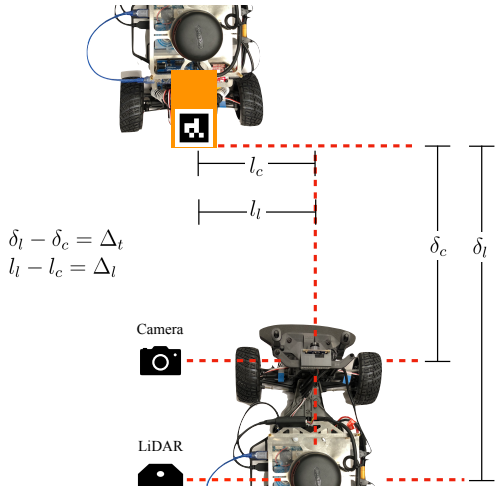

Figure 4: The follower vehicle detects the plate (orange) using the camera and LiDAR and extracts the vertical distances δ_l, δ_c and lateral offsets l_l, l_c of the leader vehicle relative to itself.

change $r_l(k)$ and $r_t(k)$ is calculated from the absolute value of the lateral and vertical offset. This interaction is described as follows:

$$r_l(k) = |\Delta_l| \tag{4}$$

$$r_t(k) = |\delta_t| \tag{5}$$

where both residuals are independent of each other. Each residual is then evaluated by the CUSUM algorithm as described on the following equation:

$$S_l(k+1) = (S_l(k) + r_l(k) - b_l)^+ \tag{6}$$

$$S_t(k+1) = (S_t(k) + r_t(k) - b_t)^+ \tag{7}$$

where $(x)^+$ denotes $\max(0, x)$, and $S_l(k+1)$ and $S_t(k+1)$ represent the total sum of discrepancies over time that will be used for the next iteration. These values are calculated from the previous accumulation $S_l(k)$ and $S_t(k)$ and the addition of its respective residuals minus biases. The biases b_l and b_t are calculated in order to avoid false alarms when the system is not under attack. Finally, each sum raises an alarm when either the lateral or vertical sum increases over a predefined threshold. In particular, we follow the CUSUM parameter (biases and thresholds) tuning recommendations from Giraldo et al. [22].

3.2 SAVIOR

SAVIOR [38] is a non-linear physics-based anomaly detection system that uses the Extended Kalman Filter (EKF). The EFK generates a prediction of the system state that accounts for noise on states and outputs. The predicted states are compared with the actual system states at the future iteration. If the aggregated difference (via the CUSUM statistic) is above a threshold, an alert is raised and the sensor is considered compromised. The underlying logic is that the predicted and observed states should match up to some threshold which may vary with the size of the noise. If the CUSUM threshold is exceeded, then the behavior of the system does not match our understanding of the physical evolution of the system. Due to the noisy nature of the sensors, the threshold for triggering an attack

is often relatively high which leaves the system vulnerable to *slow but persistent* attacks.

Extended Kalman Filter. In order to estimate the state of the vehicle, we utilize an Extended Kalman Filter (EKF) algorithm [40] to predict the future states of the camera and LiDAR sensors and compare them against measured inputs to detect anomalies. This is required primarily for the SAVIOR algorithm. The EKF algorithm uses the known system dynamics, accounts for process noise (noise on the states) and observation noise (noise on the outputs), and predicts the future system state. We run two separate instances of an EKF algorithm on each sensor (camera and LiDAR). This allows us to secure individual sensors and keep track of malicious tampering against them (this secures individual sensors to abrupt and moderate changes). The behavior of the system can be calculated using the following equation:

$$x_{k+1} = f(x_k, u_k) \tag{8}$$

where at time k, $f(x_k, u_k)$ approximates the behavior of the system using non-linear equations (1) with x_k representing the previous states of the system and u_k the current input.

The EKF algorithm is divided into two steps: prediction and update. In the prediction step, the algorithm takes into account the previous measurements to generate a calculated estimate. This value is then improved on the update step by calculating the Kalman gain and updating the estimated value. The prediction equations are:

$$x_k^- = Ax_{k-1} + Bu_{k-1} \tag{9}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{10}$$

Variables x_k^- and P_k^- describe the a priori predicted estimation state and predicted estimated co-variance. These values are then used in the update stage to improve the estimated state. $A \in \mathbb{R}^{n\times n}$ is the system dynamics matrix, $B \in \mathbb{R}^{n\times m}$ is the input matrix, and $H \in \mathbb{R}^{p\times n}$ is the output matrix. $x_k \in \mathbb{R}^n$ is the states of the system, while $u_k \in \mathbb{R}^m$ is the input. Specifically, x_k (vector) is the current vehicle state while y_k is a vector of observed outputs through the sensors as a result of applying the input u_{k-1}. The update stage consists of the following equations:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{11}$$

$$x_k = x_k^- + K_k(y_k - Hx_k^-) \tag{12}$$

$$P_k = (I - K_k H)P_k^- \tag{13}$$

K_k calculates the Kalman gain aim to minimize the error covariance of the a posteriori estimate x_k by calculating the actual contributions of the a priori estimate and the measurements while P_k states the error covariance of the a posteriori estimated state.

3.2.1 SAVIOR for Camera Data. We followed the implementations described in [38]. This allowed us to have an anomaly detector based on the camera information. The main two pieces of information that are extracted from the camera sensor are the line angle and line lateral position offset. This information is then fed into the EKF algorithm to predict the upcoming state. The historical behavior is analyzed using a CUSUM algorithm that keeps track of deviations over time.

3.2.2 SAVIOR for LiDAR Data. The LiDAR detects the location and orientation of the leading vehicle. Since the leading vehicle is following the same path, its lateral offset and orientation are thus indicative of the line's lateral location and orientation albeit at a future point in time. This is the same type of information that the camera generates but with a small offset. Hence, we can implement an instance of SAVIOR for LiDAR data as well.

We note that since the leading vehicle is executing commands based on a "future route", it is not our intention to compare the measurement that we obtain from the camera against the ones that we obtain from the LiDAR but to keep track of LiDAR data as discrepancies to avoid abrupt tampering against that sensor. The main goal is to enable anomaly detectors on both sensors to detect sudden and abrupt changes to the sensor data that might be indicative of a targeted attack.

3.3 Shared Reality Vs Sensor Fusion

It is important to recognize the assumptions and objects of these algorithms. The goal of sensor fusion is to improve the quality of detection (end result) beyond what an individual sensor can deliver. When fusing data about the same object, each sensor has its own noise and confidence level. However, we know from Bayesian statistics that the combined result will be no worse than either individual result. This highlights a key idea: *sensor fusion is not concerned with anomaly detection and flagging attacks*; sensor fusion operates under the assumption of *safe but uncertain data* that may be outside of their operating limits. Furthermore, the output of sensor fusion is an "improved" estimate of features or decisions.

On the other hand for our *Shared Reality* module, we make two assumptions about the data: 1) sensors operate in their respective regions of operation, and 2) sensors may be either safe or under attack. These two assumptions imply that in general, sensors provide truthful (yet noisy) data when safe; any anomalies or unexpected deviations and biases with sensor data are indicative of attacks. Thus, we do not consider issues such as corrupt sensor data due to their operation limits such as incorrect camera detection (or the lack of detection) when exposure suddenly changes. The goal or output of the *Shared Reality* module is a Boolean check on whether a sensor is safe or compromised. Thus, we rely on the overlapping field of view between multiple sensors to compare the properties of physical objects and cross-validate them.

While sensor fusion assumes safe but uncertain data either in or out of the region of operation, *Shared Reality* assumes that sensors are either safe or under attack but within their region of operation. Sensor fusion outputs a combined result to improve detection quality while *Shared Reality* outputs a Boolean check on the safety of a sensor and tracks the changes in the difference between data from different sensors instead of combining it. Furthermore, *Shared Reality* is designed to be computationally lightweight whereas sensor fusion can have a larger overhead depending on the implementation.

As such, *Shared Reality* does not replace sensor fusion nor is it a special case of sensor fusion. Ideally, AVs would be equipped with both *Shared Reality* and sensor fusion modules to respectively flag attacks and improve the quality of sensor estimates across the safe sensors. Realizing any overlaps in the execution of both modules can be leveraged to reduce the computation overhead by combining computational steps or sharing data between the two modules.

4 IMPLEMENTATION

To test our proposed pipeline to secure a platoon, we develop an RC car platform we call COMO, which is a variant of the Berkeley BARC project [2]. COMO is a low-cost testing platform with a camera, a 2D LiDAR, an IMU, and wheel encoders. It uses the low-level control architecture of the BARC project but has a different build and high-level control components. Each COMO car runs Ubuntu MATE 16.04 with ROS 1 (Robot Operating System) Kinetic

Kame on an Odroid XU4 [3] compute board. As we explained in Section 2, we only use the camera, LiDAR, and wheel encoders. The camera and LiDAR data process as well as the platform will be discussed in more detail.

ROS follows a publish-and-subscribe architecture for inter-process communication in which modules publish data to defined topics (message exchange busses) to which other modules can subscribe. While this approach is aimed at meeting real-time constraints, it undermines security and data integrity due to the lack of publish-and-subscribe policies. Any module can publish on any topic and the modules are not aware of which module is publishing data. This allows for the scenario in which a malicious module is able to publish erroneous data to a crucial topic and disrupt or control the system. Our aim is to demonstrate the dangers of such a scenario and protect the system against undetected sensor tampering. We note that ROS 2 features authentication methods to improve security, however, authentication will not prevent the physical and transduction attacks we described earlier (e.g., a dirty patch to fool the camera or a car nearby sending laser signals to fool the LiDAR).

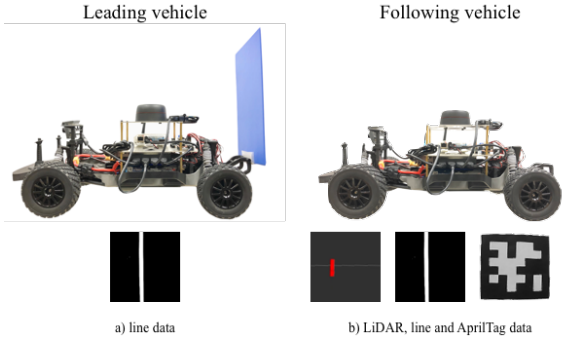

Figure 5: Physical platooning implementation.

The camera and LiDAR processing algorithms are implemented using ROS on COMO and interface directly with the motor control topic for specifying steering and throttle. The camera focuses on following the line and detecting the AprilTag (for vehicle identification and distance estimation) and the LiDAR focuses on detecting the leading vehicle to estimate distances and relative orientation. Yet, both, the camera and LiDAR, end up capturing aspects of the same reality hence the name *Shared Reality*. In our setting, we have the AprilTag and vertical plat attached to the back of the front AV. As depicted in Fig. 5, the leading AV detects the line and uses the line data to navigate while the following AV detects the plate via the LiDAR and the line and AprilTag via the camera.

4.1 Path Detection

Having a path to follow is critical for any navigation algorithm. This path is often a lane, or more abstractly, a center curve to follow. To detect the path in an image, we use the following pipeline: 1) import an image, 2) mask or crop the image, 3) apply a geometric transformation to generate a top-down view, 4) convert to a binary image, 5) detect contours, 6) weigh the contours, and 7) select the main contour for further processing.

The first step before image processing is camera calibration to obtain the camera intrinsic (focal length and optical center) and extrinsic (relative rotation and translation between the camera body frame and the ground plane) parameters. The intrinsic parameters are characteristic of the camera and need to be calculated only once. The extrinsic parameters need to be updated whenever the camera is moved (relative to the vehicle body). This establishes the following mapping between a point ${}^{w}p := [{}^{w}x, {}^{w}y, {}^{w}z]^T$ in the world frame and the pixel coordinates ${}^{i}p := [{}^{i}x, {}^{i}y]^T$:

$$\begin{bmatrix} {}^{i}x \\ {}^{i}y \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} fm_x & 0 & u_0 \\ 0 & fm_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{A \text{ (Intrinsic)}} \underbrace{\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix}}_{H \text{ (Extrinsic)}} \begin{bmatrix} {}^{w}x \\ {}^{w}y \\ {}^{w}z \\ 1 \end{bmatrix} \quad (14)$$

where f is the focal length in meters, m_x, m_y are scaling factors to express the focal distance in pixels, (u_0, v_0) is the optical center or principle point, $R \in SO(3)$ (special orthogonal group of 3×3 matrices) is the rotation matrix and $T \in \mathbb{R}^3$ is the translation matrix between the camera frame and the world frame.

There are several ways to obtain these matrices. Intuitively, the idea is to solve (14) by matching the detected corners of a chessboard of known size to the expected pattern (PnP algorithm) [29].

With a binary image, we can apply a contour detection algorithm based on [45]. Each contour has a number of sides (edges) and is characterized through the arc parameter, area, and moments which involve simple pixel counting procedures. Moments m_{jk} of a contour S are generated using:

$$m_{jk} = \Sigma_{{}^{i}x, {}^{i}y \in S} \, ({}^{i}x)^j . ({}^{i}y)^k \quad (15)$$

where $({}^{i}x, {}^{i}y)$ are the pixel coordinate pairs in a contour. We can then determine the centroid of the contour as:

$${}^{i}\overline{x} = \frac{m_{10}}{m_{00}}, \quad {}^{i}\overline{y} = \frac{m_{02}}{m_{00}}. \quad (16)$$

We generate that data for all contours and score each contour based on how closely it matches the expected characteristics of the contour for the curve. Among the criteria are the area and parameter being larger than minimum values determined experimentally, the number of sides it has, having points on the top and bottom sides of the image, and finally the closeness of the centroid $({}^{i}\overline{x}, {}^{i}\overline{y})$ to the centroid of the previous curve contour (continuity of the curve). The curve with the highest score is selected as the main contour and chosen as the target path curve.

4.2 Path Following

The offset in meters of the line from the center of the vehicle is l_{line}. This value is used with a proportional controller and regulated to zero. The control is generated as $ctrl_{steer} = K_{ps} l_{line}$ where K_{ps} is a proportional control gain. $ctrl_{steer}$ is the input δ in the bicycle model. Note that for the leader vehicle (with no vehicles in front of it), the velocity is not regulated. It is merely set to a constant force signal (open-loop control). Fig. 6 shows our experimental testbed, where the second vehicle tracks the position of the first one using the LiDAR and the camera.

4.3 Additional Sensor Processing

The platoon can operate as such: all vehicles use a camera to follow a curve; the first vehicle uses an open-loop control for velocity (simply sets the acceleration to $a = a_{des}$, where a_{des} is a constant desired value chosen experimentally and does not monitor the speed); and all other vehicles estimate the distance between themselves and the leading vehicle and maintain a constant value by varying the acceleration input as described before. In this case, a simple attack

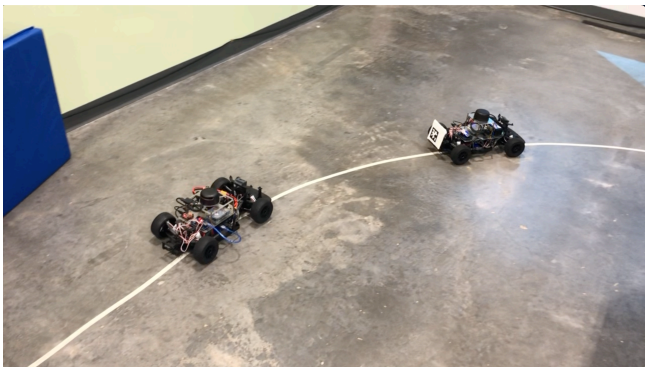

Figure 6: Both vehicles follow the line marked with the vehicle behind maintaining a constant distance.

on the camera could render the platoon useless and potentially cause costly accidents. To mitigate this, we need additional data from the camera and other sensors (2D LiDAR in our case).

We use the camera to estimate the lateral offset between the vehicle and the leading vehicle in a similar way to the vertical distance. The lateral offset is calculated as $l_c = \frac{side_{meters}}{{}^{i}l_{offset}} fm_x$ where ${}^{i}l_{offset}$ is the pixel offset of the center of the AprilTag ${}^{i}tag_{center} = abs({}^{i}UC + {}^{i}LC)$ and the image center along the horizontal axis: ${}^{i}l_{offset} = 320 - {}^{i}tag_{center,x}$ (320 is the pixel median of the image width).

Additionally, we use the LiDAR to find similar estimates of lateral offset l_l and vertical distance δ_l between the two vehicles. Since the LiDAR generates a 360-degree scan at one height only, we have to make the leading vehicle distinguishable. To that end, we attach a 0.15x0.45 meter plate to the back of the leading vehicles as seen in Fig. 5. The plate appears in the scan of the LiDAR as a cluster of neighboring points (typically 3 to 11 points depending on distance). To detect that cluster, we use the following pipeline: 1) mask LiDAR scan, 2) cluster points, and 3) select the appropriate cluster. Since the leader will only be within a certain angular and

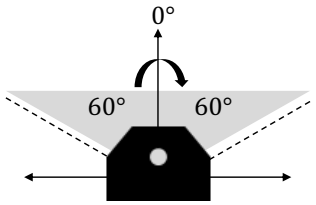

Figure 7: The LiDAR starts at position 90-degrees and moves clockwise. We are only using 120 degrees which is divided equally from the starting position to the left and right sides.

vertical distance from the follower, we mask the LiDAR scan to include a 120-degree angle centered about the vehicle's heading as indicated in Fig. 7 and a range from 0.1 to 2 meters. We then pass through the remaining points in a counter-clockwise manner and identify clusters by adding a point to the cluster if its distance to the previous point is less than 0.1 meters (to account for the naturally separated points and LiDAR error). A new cluster is identified if that threshold is exceeded. We can then calculate the location of each cluster by averaging a cluster's C points' (${}^{l}p \in C$) coordinates in the LiDAR frame:

$$ {}^{l}\overline{x} = \frac{\Sigma_{{}^{l}p \in C}({}^{l}p_x)}{N}; \quad {}^{l}\overline{y} = \frac{\Sigma_{{}^{l}p \in C}({}^{l}p_y)}{N} \tag{17} $$

where N is the number of points in that cluster C. The orientation of the plate is determined through a similar approach as that described before: the points are combined into a matrix for which the covariance matrix is obtained. The eigenvector with the largest eigenvalue is the vector representing the orientation of the plate. The angle is found as the angle between that vector and the horizontal axis.

Similar to the camera curve-detection weighing algorithm, we deploy a weighting algorithm for all clusters primarily influenced by the closeness of the cluster centers to the previous center. The winner is selected as the cluster representing the front vehicle plate and the values $l_l = {}^{l}\overline{x}$ and $\delta_l = {}^{l}\overline{y}$ are populated.

That l_c and δ_c are comparable to l_l and δ_l respectively. The lateral offsets are expected to be similar and the vertical distances are only expected to differ by the camera-LiDAR separation of about 0.21 meters. Experimentally, however, these values might not match as exactly as expected due to limitations in resolution and computational power. Nonetheless, that is accounted for in the secure platoon design.

Our secure platoon design combines two main components: securing sensor with SAVIOR [38] and securing cross-coupled data in *Shared Reality*. As an overview, SAVIOR helps secure individual sensors by detecting abrupt and moderate changes to the sensor data, but it fails at detecting slow *"realistic"* attacks. The *Shared Reality* module compares cross-coupled sensor data, i.e., similar data belonging to physical objects detected by multiple sensors. Thus, even with slow attacks on a sensor, the attack could still be detected as long as it is not coordinated with attacks on *all* the sensors and cross-coupled data that are involved in the shared reality.

4.4 Secure Pipeline

We now implement the design we originally presented in Fig. 3. We use raw LiDAR and camera data in the form of a distance scan and image matrix respectively. The data is passed to the data preprocessing module which performs all the operations on data to be used by the anomaly detectors. At its output, we get the estimates for the vertical and lateral distances between the two successive vehicles, information about the orientation of the line, and the lateral offset of the follower car relative to it. The inter-vehicular information is passed to the *Shared Reality* and the SAVIOR-based "improved detectors" modules. The latter also receives estimates of the vehicle's (follower vehicle) position relative to the line (line information) and the speed estimated by wheel encoders. The two modules operate as discussed in the previous subsections.

At the output of each module, we have boolean decisions on whether or not the module has detected an attack. The *Shared Reality* module flags an attack when the cross-coupled (shared) information between the sensors passes a certain threshold. The improved detectors module flags an attack when the data of one or both of the sensors is inconsistent with the prediction and passes a certain threshold. The modules inform which of the sensors is under attack as well. The last step is the controller which takes the boolean attack flags and selects the appropriate control sequence.

The controller is designed with many factors in mind. For example, suppose that a system has five sensors all of which have cross-coupled components checked by *Shared Reality*. If one sensor is flagged as under attack, the controller may ignore data coming from the flagged sensor, continue normal navigation, but initiate

an attack confirmation sequence to reaffirm the attack status. If two sensors are flagged, the controller may do the same thing but also alert a human operator or an AV security company that the system might be under attack. However, if three of the five sensors are flagged, the controller may direct the vehicle to the nearest safe emergency parking space and completely shut down the vehicle. Thus, the controller is designed based on the available sensors, the sensors, and data involved in *Shared Reality*, and the level of safety desired.

In our case, we have only two sensors for this demonstration. Since the controller decision-making is not our main focus, we designed a simple controller decision-making sequence: if either one of the sensors is under attack, halt the system immediately; if no attacks are detected, continue normal platoon operation.

As we stated before, since the *Shared Reality* module checks for slow drifting attacks while the SAVIOR modules check for medium to abrupt changes, either one or both modules might issue an attack flag. In either case, the existence of at least one flag indicates an attack. This secures the AVs to a wide range of attacks.

Earlier we discussed sensor fusion and explained the differences between that and our *Shared Reality* module. In light of this section, it should be clear that the two methods have different assumptions, operations, and goals. Unlike sensor fusion which uses statistical tools to combine "safe" (but uncertain/noisy) data to improve overall detection quality, *Shared Reality* uses CUSUM modules to analyze the change in the error between cross-coupled data to detect attacks.

Realizing the difference between sensor fusion and *Shared Reality* allows for the integration of the two tools. For example, the sensor fusion algorithm could take the flags from *Shared Reality* to ignore sensors under attack. This would improve the fusion quality and protect it. Further, we speculate that sensor fusion might be used to indicate whether a sensor is outside of its region of operation to prevent false-positive alerts in the *Shared Reality* module. These are topics of future work.

5 EVALUATION

For our experiments, we place two COMO cars on a circular track. Both AVs run the line-following algorithm. The first vehicle is considered the leading vehicle and the second vehicle is considered the follower vehicle. The follower vehicle additionally runs a version of the platooning algorithm with the secure platoon pipeline for anomaly detection. While the vehicles are in charge of following the line, the second vehicle's platooning algorithm is also in charge of keeping track of the leading vehicle and also maintaining a distance of approximately 60 cm at all times. We proceeded to attack both sensors, the camera, and the LiDAR, by injecting false data individually.

5.1 Attacks

We create a malicious node in ROS that publishes bad data at rates higher than the legitimate sensor. This malicious node can represent a software attack or a physical attack. Fig. 8 depicts how our malicious module injects data in ROS.

In order to attack the camera, we first recorded footage of the compromised path. Since the vehicle is running the anomaly detector SAVIOR, abrupt and moderate changes will be detected. In order to craft this attack, we recorded small movements of the line and the AprilTag towards the left of the screen. Since these changes are gradual and small, they could be caused by normal behavior and therefore SAVIOR will not be able to recognize these changes as malicious (e.g., during experimentation the vehicle started turning to the left accordingly to account for the changes). Fig. 9 shows how the vehicle is interpreting the attack. From image a) to c), the line and leading vehicle are gradually moving towards the left side of the screen. Despite our improvements to the camera by adding the AprilTag detector, an attacker can inject images of the line and the AprilTag and still remain undetected.

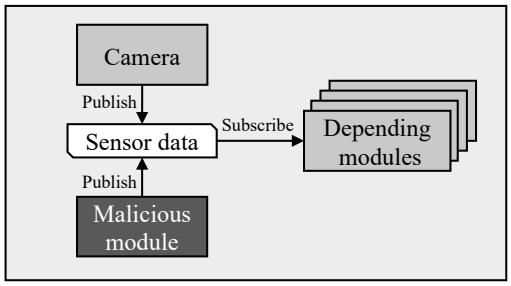

Figure 8: A malicious node can publish data to any topic in the system once an attacker has gained access. Underlying modules that depend on critical data will act upon malicious tampered data due to a lack of verification.

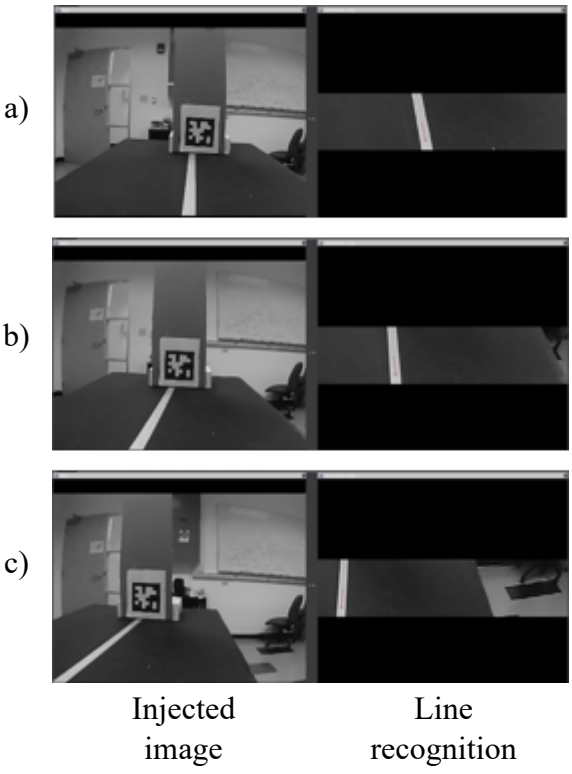

Figure 9: The injected image is shown on the left column and the line being recognized is shown on the right column. a) shows the initial injected image and how the system is recognizing it. b) and c) show slow movement towards the left side of the screen. The line detected shows gradual movement that will not be detected by previous approaches

Similarly, the LiDAR sensor is also vulnerable to the same type of attack. Since we implemented an anomaly detector using the LiDAR data, abrupt and moderate changes will be detected as well but subtle consistent movements will not. Fig. 10 shows how we were able to inject LiDAR data by moving the leading vehicle gradually to the right while the line remained in the middle. This attack is also not detected by the previous implementation due to the fact that changes are very minimal.

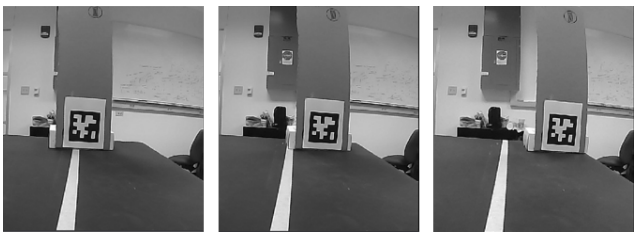

Figure 10: LiDAR attack. From a) to c), the line remains in the middle while the leading vehicle is moving gradually to the right. Since the movement is not abrupt or moderate, it will not be detected by our LiDAR anomaly detector alone.

The main reason why these attacks are not detected by SAVIOR is the fact that the changes are done very slowly which indicates that its residual is smaller than the expected value that is subtracted from the CUSUM algorithm (recall that the expected value is subtracted from each sensor to prevent the algorithm from increasing during normal execution). Attacks identified by SAVIOR required a sudden change that would produce a residual that was bigger than the expected subtracted value and the accumulation eventually resulted in an alarm being raised. In order to defend against these types of attacks in which the attacker knows the underlying cumulative algorithm and he or she is able to constantly inject data that will cause a deviation not detected by previous work, we leveraged sensor redundancy (camera and LiDAR) and implemented a solution in our *Shared Reality* module.

5.2 Shared Reality Comparison

While the LiDAR and the camera sensors are both used for different purposes in the vehicle with the camera used for navigation and the LiDAR used for collision avoidance, we were able to leverage this sensor redundancy to improve the detection of these types of attacks. This is due to an added protection in our *Shared Reality* module that synchronizes the camera and the LiDAR to ensure that both sensors are experiencing the same reality. If an attacker is able to insert malicious inputs into the camera while making sure of staying within the threshold boundaries (slow attacks), SAVIOR alone will not be able to detect it. But by using LiDAR data, we were able to ensure that even small deviations that will normally not be detected by a single sensor, can be noted and alarm can be raised. Attacks on each specific sensor, the camera or the LiDAR, are able to be detected by our *Shared Reality* module.

Fig. 11 depicts the comparison between SAVIOR and the *Shared Reality* module with an attack targeting the camera. The y-axis represents the residual percentage needed to raise an alarm with respect to the detector (e.g., one equals an attack has been detected) and the x-axis represents time. For this experiment, we set our *Shared Reality* module to detect anomalies when the camera and the LiDAR lateral position differ by more than 10cm. Since the changes

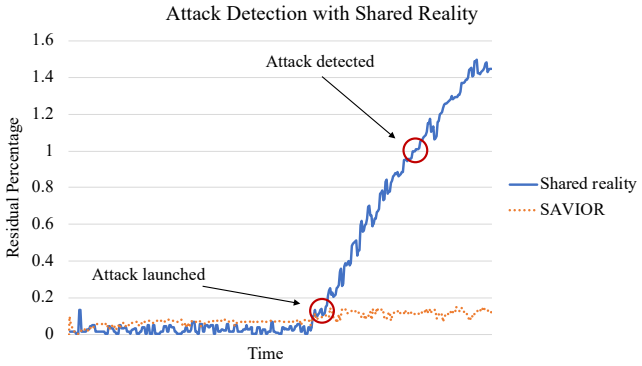

Figure 11: The *Shared Reality* module is able to detect slow attacks that mimic normal behavior while they still remain undetected by SAVIOR.

produced by our slow attack are minimal, SAVIOR's residuals do not go beyond 20%. On the other hand, once the attack was launched, our *Shared Reality* module started accumulating residual until they reached 100% (meeting our threshold of 10 cm).

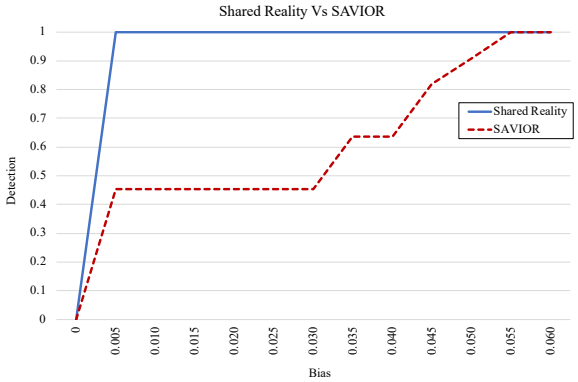

Figure 12: Bias attack detection between *Shared Reality* and SAVIOR. As the bias is increased, SAVIOR is able to detect more attacks while *Shared Reality* is detecting all attacks at several intensity intervals.

We also evaluate *Shared Reality* and SAVIOR in terms of granularity. For this experiment, we inject a small bias deviation to the vehicle position ranging from 0.005 to 0.06 meters. For each bias injected, we perform 11 experiments with 11 distinct thresholds ranging from 50 to 150 percent variations with 100 percent being the normal threshold of the system (e.g., the threshold selected under normal behavior that does not raise false alarms). When the bias is zero, both systems do not detect any attacks as the system is under normal behavior (the detection of an attack under zero bias would be a false positive). Once we start injecting a bias of 0.005 m, *Shared Reality* detects attacks for all 11 threshold variations (0.5 to 1.5). At the same bias, SAVIOR is only able to detect an attack in 45.45% of the 11 variations. While *Shared Reality* is able to identify all attacks under all injected biases, SAVIOR gradually increases its detection as the bias grows. Both systems are able to identify all

attacks when the bias is greater than or equal to 0.055 m. Fig. 12 depicts the results.

5.3 Performance

We tested the efficiency of both SAVIOR and *Shared Reality* at different attack frequencies in order to find the point in which SAVIOR would start identifying attacks. In order to do this, we injected an image into the camera sensor with slight pixel movements every frame. Fig. 13 shows the results of moving an image from 5 pixels to 100 pixels per frame to the left side of the screen. The time shown in seconds reveals that *Shared Reality* takes 6.0676 seconds to detect movement at 5 pixels while it is reduced to 0.17 seconds at 100 pixels. This is due to the fact that the faster the change causes discrepancies to accumulate faster. SAVIOR was able to detect attacks starting at 70 pixels per frame in 0.04 seconds and stayed relatively close to detecting 100-pixel changes at 0.07 seconds. While the *Shared Reality* module continued decreasing its time to detect the attack, once SAVIOR started detecting the attacks, it was much faster at identifying them. This states that while the addition of a *Shared Reality* module improves the detection of "small consistent attacks", it is not a replacement for the presence of an anomaly detector but a complementary tool.

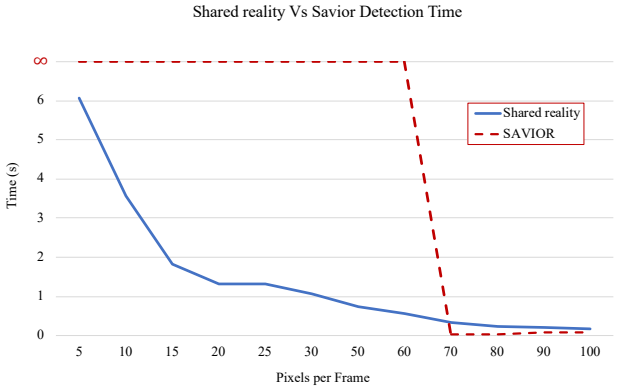

Figure 13: The injected image was moved to the right side at different pixel rates. *Shared Reality* starts detecting attacks at 5 pixels in 6.04 s and gradually decreases detection time to 0.17 s at 100 pixels. SAVIOR starts detecting attacks at 70 pixels in 0.04 s with similar detection time as pixels increases.

We also evaluated the overhead introduced to the system by *Shared Reality* in terms of CPU utilization. Since ROS operates modules independently from each other, we opted to record utilization information from the OS point of view. To achieve this, we recorded processor utilization for each module (including threads) while the vehicle executed the platoon algorithm. We collected about 750 measurements of CPU utilization for all modules. This information is then aggregated using a Python script that computes the average of all available measurements per module including their threads. Table 1 summarizes the CPU Utilization for the modules involved in the implementation.

It is a well-known fact that camera processing is among the most computationally expensive tasks. This is primarily due to the size of the data being processed. Even for our small 640x480 image, the matrix representing the image is 307200 pixels each of which is

Module	CPU utilization
camera_preprocessing	14.6716%
LiDAR_pre-processing	11.2368%
lidar_collision_avoidance	10.9354%
process_line	9.05788%
img_preprocessing	6.48217%
shared_reality	4.96333%
Camera+SAVIOR	3.11916%
LiDAR+SAVIOR	3.06290%
arduino_node	2.53407%
elp_cam_bridge	1.17409%
line_follower	1.10546%
lidar_follower	1.00095%
controller_low_level	0.98185%
platoon	0.86881%
ros_launch	0.43253%
rosmaster	0.31747%
rosout	0.29065%

Table 1: CPU utilization of modules inside of the ground vehicle. *Shared Reality* adds a 4.963% of CPU utilization to the system while both implementations of SAVIOR remain at around 3.1%.

an 8-bit value (2.4576 million bits). Compared to that, the LiDAR consists of 360 data points where each point is a float (32 bits) totaling only 11,520 bits. This results in the camera data being over 200 times larger than the LiDAR data. Table 1 reaffirms that result. The 'camera_preprocessing' and 'img_preprocessing' modules consist of nodes that perform various image processing tasks before subsequent nodes can extract useful information. These tasks include identifying regions of interest, cropping them, and performing a spatial transformation to obtain the top-down image of the line. These modules use up over 20% of the CPU utilization. Processing the line image ('process_line') performs the necessary image processing on the cropped image containing the line to extract the path information (relative position and orientation between the line and the car). Even with a smaller cropped image, the module still required about 9% CPU utilization. *Shared Reality* adds on CPU utilization of 4.963%. This is slightly higher than SAVIOR. Both implementations of SAVIOR (camera and LiDAR) utilize about 3.1% of the CPU. The modules with the highest CPU utilization are the modules in charge of the camera and LiDAR pre-processing. These modules provide the heavy computations that will be used later on in the anomaly detector. The process_line module and image_processing modules perform computations related to the line following algorithm. The arduino_node module is in charge of sending the actuator commands to the wheels. The elp_cam_bridge module receives data from the camera and makes it available to the system. The line and LiDAR follower modules are instances that keep track of the movements of the line and the leading vehicle respectively and calculates the offset between the vehicle and the center of the line. The controller_low_level module serves as a message interface between the computer board and the microcontroller that communicates with the actuators. The platoon module is in charge of following the leading vehicle and adding or reducing speed if the leading vehicle is closer than the defined threshold. Finally, the ros_launch, rosmaster, and rosout modules perform operations related to ROS such as launching several modules simultaneously, providing communication between modules, and providing an interface to log information about the system.

6 RELATED WORK

Attacks on vehicle sensors are becoming increasingly sophisticated. Researchers concerned with LiDAR spoofing attacks have demonstrated how this sensor can be tampered with to cause false positive detection of a front vehicle. The authors of CARLO demonstrated how LiDAR front-vehicle detection with machine learning can be attacked with a small number of spoofed points in introduced their solution to detect spoofing attacks using vehicle occlusion patterns and the projection of LiDAR data onto different planes [44]. RADARs have also been attacked by injecting false data to make objects appear closer than they actually are [14]. This is achieved by capturing the RADAR signals and selecting a delay before sending a frequency with the desired distance chosen by the attacker. Some of these attacks do not require specialized hardware as demonstrated attacks on camera and LiDAR sensors using commodity hardware to perform binding, jamming, replay, and spoofing attacks showed its feasibility [36]. Attacks have also been considered for platooning vehicles as [24] exposed potential vulnerabilities with LiDAR and RADAR sensors where malicious information can be shared with a targeted vehicle. Other approaches have focused on the underlying machine learning algorithms as [11] showed that vulnerabilities of LiDARs can be exploited even in settings in which AVs use machine-learning object detection systems are present and [16] demonstrated evasion attacks performed against Convoluted Neural Networks (CNN) classifying camera images to predict the steering angle.

Other works exposed sensor vulnerabilities related to RADARs, cameras, and ultrasonic sensors requiring contact-less interaction with the vehicle. These attacks have demonstrated that it is possible to blind and deceive AVs [54]. There have been approaches that focused on analyzing sensor data being detected by the vehicle. [53] proposed a framework for identifying cyber-attacks and faulty sensor readings by combining a CNN and a Kalman Filter anomaly detector that observe incoming data from multiple sensors and make real-time decisions on whether the data is anomalous or not.

From this literature review, it is clear that securing sensors to attacks is not an easy endeavor and is still actively pursued. Even well-trained machine learning algorithms can still be attacked, sometimes with simple and physically impossible data [44], but it would remain undetected. Our proposed shared reality will limit the effectiveness of these sensor attacks. *Shared reality* is related to sensor fusion and there has been extensive research on sensor fusion [7, 10, 15, 21, 35, 51, 52, 56]. The goal of sensor fusion is to improve the quality of sensor measurements. Sensor fusion, however, is not intended to detect malicious sensor tampering. In our case, instead of attempting to improve an estimate from different sensor sources, our goal is to create an independent estimate of the physical state of the world with each sensor, and then test if both sensors perceive the same physical world.

7 CONCLUSION

In this work we improved upon the EKF-based camera anomaly detector SAVIOR [38] by: 1) applying SAVIOR to 2D LiDAR data and thus securing multiple sensors to abrupt and medium changes, 2) introducing the light-weight *Shared Reality* module to monitor cross-coupled data across multiple sensors corresponding to the same physical object which enables the detection of small drifting attacks on a subset of the sensors, and 3) combining the SAVIOR and *Shared Reality* modules into the secure pipeline which ends with a control decision module that can be modified depending on the sensors and data involved in the *Shared Reality* module. We demonstrate the flaws of using SAVIOR alone and the improved attack detection range with the secure pipeline highlighting the strengths of each module. In the end, the AVs are secured against medium and abrupt changes as well as the more powerful attack pattern where the attacker knows of the SAVIOR anomaly detection mechanisms and avoids it by injecting a small offset that accumulates over time to create a significant deviation. While SAVIOR improved the detection of malicious input into the system, this work improved the detection of a new type of attack by leveraging sensor redundancy and introducing a module for reality checking that ensures that participating sensors make attacks much harder. Our design is more resilient than previous work because an attacker now has to take into account how other sensors would react if drastic or small changes are introduced into the system. We tested our implementation on a vehicle platooning algorithm and showed that sensor redundancy improves the security of the AVs with an increased overhead of 4.96333%.

8 DISCUSSION AND FUTURE DIRECTIONS

Our proposed method uses SAVIOR modules to secure sensors to medium and abrupt changes and the *Shared Reality* module to secure the AV to slow drifting changes by cross-coupling data across multiple sensors. Nonetheless, the proposed secure pipeline is not perfect. There is still room for improvement based on the following:

- The number of sensors and data cross-coupled through *Shared Reality* is important. With just two sensors and two distinct features (lateral and vertical inter-vehicle distance), as we have, the AV's secure pipeline control decision module must take escalated measures immediately. Having more sensors and more cross-coupled physical features further secures the platform.
- A slow attack on *all* sensors and that affect *all cross-coupled data* in the *same way* would thwart the *Shared Reality* module. However, having to affect all cross-coupled data in the same way forces the attacker to not only craft the attack to be a small drifting one, but also craft it so that most cross-coupled features across most sensors remain within the *Shared Reality* thresholds. This increases the difficulty of launching successful and undetected attacks.
- Incorporation of *Shared Reality* with sensor fusion: it would be interesting to merge aspects of both modules to ensure sensor fusion sensors are secure and confirm that *Shared Reality* sensors are operating properly.

We finalize by emphasizing three important features of our pipeline: 1) the *Shared Reality* module is computationally lightweight, 2) the pipeline and particularly the *Shared Reality* module serve as frameworks that can be expanded or reduced depending on the available hardware and operation setting, and 3) even with three sensors, multiple cross-coupled features can be used raising the difficulty required for an attack.

ACKNOWLEDGMENTS

This research was sponsored by the Army Research Office and was accomplished under Grant Number W911NF-20-1-0253. This research is also supported by NSF CNS-1929410, CNS-1931573, and AFRL FA8750-19-2-0010. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein

REFERENCES

[1] Platooning trucks to cut cost and improve efficiency (2018), https://www.energy.gov/eere/articles/platooning-trucks-cut-cost-and-improve-efficiency

[2] Mpc-berkeley/barc. https://github.com/MPC-Berkeley/barc (2020)

[3] Odroid-xu4 (Aug 2020), https://wiki.odroid.com/odroid-xu4/odroid-xu4

[4] Ahmed, C.M., Murguia, C., Ruths, J.: Model-based attack detection scheme for smart water distribution networks. In: Asia Conference on Computer and Communications Security (AsiaCCS). pp. 101–113. ACM (2017)

[5] Alam, A., Besselink, B., Turri, V., Martensson, J., Johansson, K.H.: Heavy-duty vehicle platooning for sustainable freight transportation: A cooperative method to enhance safety and efficiency. IEEE Control Systems Magazine **35**(6), 34–56 (2015)

[6] Aoudi, W., Iturbe, M., Almgren, M.: Truth will out: Departure-based process-level detection of stealthy attacks on control systems. In: Conference on Computer and Communications Security (CCS). pp. 817–831. ACM (2018)

[7] Asvadi, A., Garrote, L., Premebida, C., Peixoto, P., Nunes, U.J.: Multimodal vehicle detection: fusing 3d-lidar and color camera data. Pattern Recognition Letters **115**, 20–29 (2018)

[8] Bergen, M.: Uber has spent more than $1 billion on driverless cars (April 2019), https://www.bloomberg.com/news/articles/2019-04-11/uber-has-spent-more-than-1-billion-on-driverless-cars

[9] Bishop, R.: U.s. states are allowing automated follower truck platooning while the swedes may lead in europe (May 2020), https://www.forbes.com/sites/richardbishop1/2020/05/02/us-states-are-allowing-automated-follower-truck-platooning-while-the-swedes-may-lead-in-europe/#38106e60d7e8

[10] Blasch, E., Bossé, E., Lambert, D.: High-Level Information Fusion Management and System Design. Artech House, Inc., USA, 1st edn. (2012)

[11] Cao, Y., Xiao, C., Cyr, B., Zhou, Y., Park, W., Rampazzi, S., Chen, Q.A., Fu, K., Mao, Z.M.: Adversarial sensor attack on lidar-based perception in autonomous driving. Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security (Nov 2019). https://doi.org/10.1145/3319535.3339815, http://dx.doi.org/10.1145/3319535.3339815

[12] Cao, Y., Xiao, C., Cyr, B., Zhou, Y., Park, W., Rampazzi, S., Chen, Q.A., Fu, K., Mao, Z.M.: Adversarial Sensor Attack on LiDAR-based Perception in Autonomous Driving. In: Conference on Computer and Communications Security (CCS) (2019)

[13] Cardenas, A.A., Amin, S., Lin, Z.S., Huang, Y.L., Huang, C.Y., Sastry, S.: Attacks against process control systems: risk assessment, detection, and response. In: Asia Conference on Computer and Communications Security (AsiaCCS). pp. 355–366 (2011)

[14] Chauhan, R., Gerdes, R.M., Heaslip, K.: Demonstration of a false-data injection attack against an fmcw radar. In: Proc. Embedded Security Cars (ESCAR). p. 135 (2014)

[15] Chen, X., Ma, H., Wan, J., Li, B., Xia, T.: Multi-view 3d object detection network for autonomous driving. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 1907–1915 (2017)

[16] Chernikova, A., Oprea, A., Nita-Rotaru, C., Kim, B.: Are self-driving cars secure? evasion attacks against deep neural networks for steering angle prediction (2019)

[17] Choi, H., Lee, W.C., Aafer, Y., Fei, F., Tu, Z., Zhang, X., Xu, D., Xinyan, X.: Detecting attacks against robotic vehicles: A control invariant approach. In: Conference on Computer and Communications Security (CCS). pp. 801–816. ACM (2018)

[18] Davidson, D., Wu, H., Jellinek, R., Ristenpart, T., Singh, V.: Controlling UAVs with sensor input spoofing attacks. In: Workshop on Offensive Technologies (WOOT). pp. 221–231. USENIX Association (2016)

[19] Etigowni, S., Tian, D.J., Hernandez, G., Zonouz, S., Butler, K.: Cpac: securing critical infrastructure with cyber-physical access control. In: Annual Computer Security Applications Conference (ACSAC). pp. 139–152. ACM (2016)

[20] Fu, K., Xu, W.: Risks of trusting the physics of sensors. Communications of the ACM **61**(2), 20–23 (2018)

[21] Gao, H., Cheng, B., Wang, J., Li, K., Zhao, J., Li, D.: Object classification using cnn-based fusion of vision and lidar in autonomous vehicle environment. IEEE Transactions on Industrial Informatics **14**(9), 4224–4231 (2018)

[22] Giraldo, J., Urbina, D., Cardenas, A., Valente, J., Faisal, M., Ruths, J., Tippenhauer, N.O., Sandberg, H., Candell, R.: A survey of physics-based attack detection in cyber-physical systems. ACM Computing Surveys (CSUR) **51**(4), 1–36 (2018)

[23] Hadžiosmanović, D., Sommer, R., Zambon, E., Hartel, P.H.: Through the eye of the PLC: semantic security monitoring for industrial processes. In: Annual Computer Security Applications Conference (ACSAC). pp. 126–135. ACM (2014)

[24] Jagielski, M., Jones, N., Lin, C.W., Nita-Rotaru, C., Shiraishi, S.: Threat detection for collaborative adaptive cruise control in connected cars. In: Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks. pp. 184–189 (2018)

[25] Kato, S., Takeuchi, E., Ishiguro, Y., Ninomiya, Y., Takeda, K., Hamada, T.: An open approach to autonomous vehicles. IEEE Micro **35**(6), 60–68 (2015)

[26] Keen, T.: Experimental security research of tesla autopilot. Tencent Keen Security Lab (2019), https://keenlab.tencent.com/en/whitepapers/Experimental_Security_Research_of_Tesla_Autopilot.pdf

[27] Kyle Hyatt, C.P.: Self-driving cars: A level-by-level explainer of autonomous vehicles (March 2018), https://www.cnet.com/roadshow/news/self-driving-car-guide-autonomous-explanation/

[28] Laris, M.: Tesla running on 'autopilot' repeatedly veered toward the spot where apple engineer later crashed and died, federal investigators say (Feb 2020), https://www.washingtonpost.com/transportation/2020/02/11/telsa-running-autopilot-repeatedly-veered-toward-spot-where-apple-engineer-later-crashed-died-federal-investigators-say/

[29] Lepetit, V., Moreno-Noguer, F., Fua, P.: Epnp: An accurate o (n) solution to the pnp problem. International journal of computer vision **81**(2), 155 (2009)

[30] Liang, K.Y., Mårtensson, J., Johansson, K.H.: Heavy-duty vehicle platoon formation for fuel efficiency. IEEE Transactions on Intelligent Transportation Systems **17**(4), 1051–1061 (2015)

[31] Liu, Y., Ning, P., Reiter, M.K.: False data injection attacks against state estimation in electric power grids. In: Conference on Computer and Communications Security (CCS). pp. 21–32. ACM (2009)

[32] Lynberg, M.: Automated vehicles for safety (Nov 2018), https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

[33] Matheson, R.: Study measures how fast humans react to road hazards (Aug 2019), http://news.mit.edu/2019/how-fast-humans-react-car-hazards-0807

[34] Nassi, B., Nassi, D., Ben-Netanel, R., Mirsky, Y., Drokin, O., Elovici, Y.: Phantom of the adas: Phantom attacks on driver-assistance systems. Cryptology ePrint Archive, Report 2020/085 (2020), https://eprint.iacr.org/2020/085

[35] Ouyang, Z., Liu, Y., Zhang, C., Niu, J.: A cgans-based scene reconstruction model using lidar point cloud. In: 2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC). pp. 1107–1114. IEEE (2017)

[36] Petit, J., Stottelaar, B., Feiri, M.: Remote attacks on automated vehicles sensors : Experiments on camera and lidar (2015)

[37] Petit, J., Stottelaar, B., Feiri, M., Kargl, F.: Remote attacks on automated vehicles sensors: Experiments on camera and lidar. Black Hat Europe 11 (2015)

[38] Quinonez, R., Giraldo, J., Salazar, L., Bauman, E., Cardenas, A., Zhiqiang, L.: SAVIOR: Securing autonomous vehicles with robust physical invariants. In: 29th USENIX Security Symposium (USENIX Security 20). USENIX Association, Boston, MA (aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/quinonez

[39] Rajamani, R.: Vehicle dynamics and control. Springer Science & Business Media (2011)

[40] Romaniuk, S., Gosiewski, Z.: Kalman filter realization for orientation and position estimation on dedicated processor. acta mechanica et automatica **8**(2), 88–94 (2014)

[41] Sato, T., Shen, J., Wang, N., Jia, Y.J., Lin, X., Chen, Q.A.: Security of deep learning based lane keeping system under physical-world adversarial attack. arXiv preprint arXiv:2003.01782 (2020)

[42] Shin, H., Kim, D., Kwon, Y., Kim, Y.: Illusion and dazzle: Adversarial optical channel exploits against lidars for automotive applications. In: International Conference on Cryptographic Hardware and Embedded Systems (CHES). pp. 445–467. Springer (2017)

[43] Son, Y.M., Shin, H.C., Kim, D.K., Park, Y.S., Noh, J.H., Choi, K.B., Choi, J.W., Kim, Y.D.: Rocking drones with intentional sound noise on gyroscopic sensors. In: USENIX Security Symposium (USENIX Security). USENIX Association (2015)

[44] Sun, J., Cao, Y., Chen, Q.A., Mao, Z.M.: Towards robust lidar-based perception in autonomous driving: General black-box adversarial sensor attack and countermeasures. arXiv preprint arXiv:2006.16974 (2020)

[45] Suzuki, S., et al.: Topological structural analysis of digitized binary images by border following. Computer vision, graphics, and image processing **30**(1), 32–46 (1985)

[46] Trippel, T., Weisse, O., Xu, W., Honeyman, P., Fu, K.: Walnut: Waging doubt on the integrity of mems accelerometers with acoustic injection attacks. In: European Symposium on Security and Privacy (EuroS&P). pp. 3–18. IEEE (2017)

[47] Tsugawa, S., Jeschke, S., Shladover, S.E.: A review of truck platooning projects for energy savings. IEEE Transactions on Intelligent Vehicles **1**(1), 68–77 (2016)

[48] Tu, Y., Lin, Z., Lee, I., Hei, X.: Injected and delivered: fabricating implicit control over actuation systems by spoofing inertial sensors. In: USENIX Security Symposium (USENIX Security). pp. 1545–1562. USENIX Association (2018)

[49] Urbina, D.I., Giraldo, J.A., Cardenas, A.A., Tippenhauer, N.O., Valente, J., Faisal, M., Ruths, J., Candell, R., Sandberg, H.: Limiting the impact of stealthy attacks on industrial control systems. In: Conference on Computer and Communications Security (CCS). pp. 1092–1105. ACM (2016)

[50] Wakabayashi, D.: Self-driving uber car kills pedestrian in arizona, where robots roam (March 2018), https://www.nytimes.com/2018/03/19/technology/uber-driverless-fatality.html

[51] Wang, Z., Wu, Y., Niu, Q.: Multi-sensor fusion in automated driving: A survey. IEEE Access (2019)

[52] Wu, T.E., Tsai, C.C., Guo, J.I.: Lidar/camera sensor fusion technology for pedestrian detection. In: 2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC). pp. 1675–1678. IEEE (2017)

[53] Wyk, F., Wang, Y., Khojandi, A., Masoud, N.: Real-time sensor anomaly detection and identification in automated vehicles (08 2018). https://doi.org/10.13140/RG.2.2.32141.38888

[54] Yan, C.: Can you trust autonomous vehicles : Contactless attacks against sensors of self-driving vehicle (2016)

[55] Yan, C., Xu, W., Liu, J.: Can you trust autonomous vehicles: Contactless attacks against sensors of self-driving vehicle. DEF CON **24** (2016)

[56] Zhong, Z., Liu, S., Mathew, M., Dubey, A.: Camera radar fusion for increased reliability in adas applications. Electronic Imaging **2018**(17), 258–1 (2018)