Defending Ultrasonic Sensors against Spoofing in Autonomous Vehicles

EECS 452 - Digital Signal Processing Laboratory
The University of Michigan

Team Members
Jack Brady
Leela Mukherjee
Ian Steele

Advising Faculty:
Prof. Shai Revzen
Marion Anderson
Daphne Zhou

April 23, 2025

Table of Contents

Introduction

Ultrasonic sensors are commonly used in modern vehicles, especially autonomous vehicles, for close-range distance sensing applications. Examples include lane assist, parking, and blind spot detection. They operate by emitting an ultrasonic wave and measuring the time it takes to send and receive the signal. This produces a distance measurement. However, they are susceptible to malfunction by certain materials, environments, and even potential malicious attacks. These attacks could come in the form of jamming or spoofing, which can disrupt obstacle detection or cause other erratic behavior from the sensor, which impacts the vehicle's function.

This project focuses on demonstrating the feasibility of contactless attacks on autonomous vehicles through their ultrasonic sensors and developing a defense software to identify and warn of these attacks. To achieve this, we built a hardware system replicating the interaction between an autonomous vehicle's ultrasonic sensor and a potential attacker's ultrasonic device. This setup served as a test bed for analyzing various attack vectors and their impact on sensor readings. Additionally, we designed an algorithm for STM32 hardware, combined with Python post-processing, that can detect common attack patterns and ensure the integrity of data transmitted to the vehicle's sensor systems. By addressing this vulnerability, our project aims to enhance the overall security of autonomous vehicles that use these sensors, hopefully contributing to safer and more reliable transportation systems in the future.

Background

We referred to the following articles to get a better understanding of existing solutions, background on the topics of ultrasonic sensors and attack styles, and analysis of how feasible this project would be to implement for our project. Specifically, we used *SoundFence* to define how we should approach our project, since it is a similar approach, and we looked at *A review of cyber attacks on sensors* to design the signal injection on our attack sensor.

1. SoundFence: Securing Ultrasonic Sensors in Vehicles Using Physical Layer Defense
   https://doi.org/10.1109/SECON52354.2021.9491590
   We used paper to determine how to design the hardware portion of our project, in order to recreate a system that replicates the interaction of an ultrasonic sensor in an autonomous vehicle system with that of an "attacker" ultrasonic sensor. We specifically looked at the general hardware model setup by the paper (see Page 5 and 7), the pulsing patterns used by the AV sensor and the attacker sensors (see Page 5 Table III), and the equations used to design the sensor interactions (see Page 3 B, Page 4 C, and Page 4 E).

2. A review of cyber attacks on sensors and perception systems in autonomous vehicles
   https://doi.org/10.1016/j.ject.2024.01.002
   This paper was a good overview of the specific equations attributed to each kind of sensor attack (See Sections 4.1 through 4.4), which we then used to characterize the

defense and attack signals in our hardware. There was also a section on different defense techniques, Table 4. This section covered analysis between existing and previous defensive techniques and links to other papers, but we needed to be careful when considering these techniques because most of them used some form of machine learning, which we could not implement on bare metal.

3. Analysing the Ultrasonic Sensors Security for Autonomous Vehicles and its Enhancement
   https://doi.org/10.1109/JIOT.2018.2867917
   This paper was more of a general overview of ultrasonic sensor attacks and defenses, and gave an overview of different attacks or situations that could result in inaccurate results from an ultrasonic sensor. Focusing on the following sections helped us gain a deeper understanding of multiple kinds of attacks.

4. Can You Trust Autonomous Vehicles: Contactless Attacks against Sensors of Self-driving Vehicle
   https://cyansec.com/files/articles/16DEFCON-Sensor.pdf
   We used this paper to classify and visualise the actual signals of what an attack on an ultrasonic sensor looks like, and better design what our signal attacks look like, specifically in Section 5 of this paper, named Attacking Ultrasonic Sensors. We focused on 5.1 System Model which explains time-of-flight equations and the component requirements of an ultrasonic sensor and 5.2 Jamming Attack, 5.3 Spoofing Attack, and 5.4 Acoustic Quieting which characterizes the different attacks.

High-Level Project Description

We demonstrated our project's functionality by first showing the interaction between a naïve ultrasonic sensor and an attacker ultrasonic sensor, showing how an attacker can corrupt an AV's sensor signal. Then, we plan to run our defense algorithm on the threatened sensor and demonstrate its ability to identify an attack.

We mounted two ultrasonic sensors so that their signals intersected each other, and we designated one as an attacker sensor and the other as a naïve sensor. This allowed us to program our naïve sensor as a typical collision detection sensor in a car with added hardware defense, and to program our attacker signal to inject a false signal into the sensing stream of the naïve sensor. We ran two independent programs for the attacker and naïve sensor on the same dev board. The user was able to pick what kind of signal to inject with the attack sensor, and on the naïve sensor, the program stored distance data and processed it to defend against an attack. The processed data was then sent through serial to another layer of Python processing on the user's PC. After this layer of processing, either a warning for random spoofing or jamming was issued, or no attack was identified.

Hardware:

- STM32 NUCLEO-L4R5ZI-P
- HC-SR04 Ultrasonic sensors (x2)
- Wooden Structure
- 3D printed sensor stands
- Breadboard
- Various jumper wires
- External button
- Junction box

Figure 1 below shows how our attack signal interaction works on a hardware level, with both ultrasonic sensors being controlled by the same STM32 NUCLEO board and the sensor signal interactions showing how the naïve signal bouncing off an object is interfered with by the attack signal.

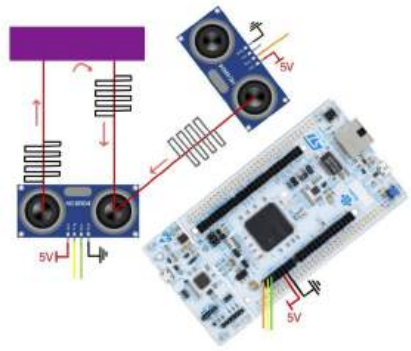


Figure 1. Diagram of signal interactions and attack signal interference.

Software:

C/C++ Processing (on STM32)

- Runs attacker and defender loops with parallel analog outputs
- Partially analyzes defender data
- Transfers defender data to external PC by serial data transfer

- Python Processing (on PC with serial data from STM32)
  - Takes transferred defender data and analyzes data further with DSP tools
  - Categorizes attack type as jamming or random spoofing based on DSP analysis

Figure 2 shows the software block diagram of our project. Inside the red dotted outline is all of the processing that occurs on the STM32, the embedded bare-metal portion of our project. The portion besides that happens over serial to the connected PC and is displayed on a live plot.

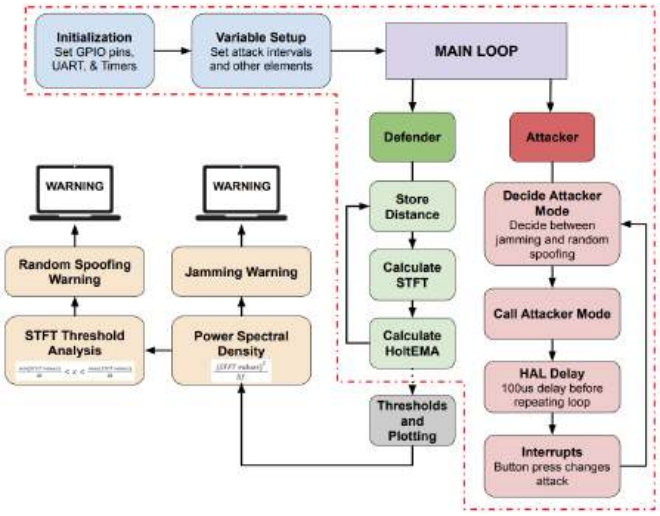


Figure 2. Block diagram of the software elements. Color broadly indicates code subroutines.

Ultrasonic Sensor Typical Operation

When an ultrasonic sensor is operating in its general state, an initial pulse is sent out (TX) and a timer is started, it bounces against an object, and the same pulse is received (RX) and the timer is stopped. The time between the pulse being sent and received is the propagation delay, and this indicates the distance between the sensor and the object.

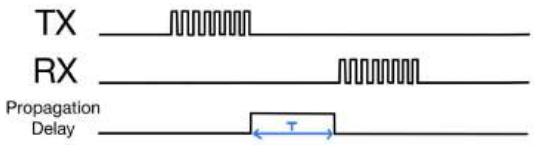


Figure 3. Propagation delay diagram, distance = propagation time × speed of sound.

Ultrasonic Sensor Attacker Algorithms

The first attack we focused on is called random spoofing, which is when a signal with a randomized period is injected into the sensing stream of the naïve sensor by the attacker sensor. This is what our attacker loop injects when “Random Spoofing” is indicated by our user.

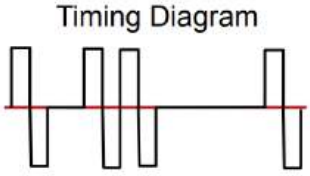


Figure 3. Timing of random spoofing, showing short attack bursts sent at random intervals.

The second attack we looked at for the scope of this project is jamming. This is when a continuous signal with a high amplitude is injected into the sensing stream of the naïve sensor. The signal equation is indicated below. This is what our attacker loop injects through the attacker sensor when “Jamming” is indicated by our user.

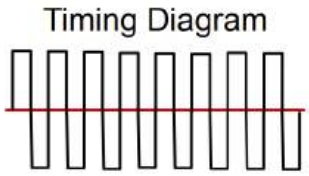


Figure 3. Timing diagram of jamming, a high-amplitude high-frequency interference signal.

STM32 Defense Processing

The first layer of defense on the ultrasonic sensor processing is taking the plain signal from the propagation delay on the naïve sensor and calculating the Holt exponential moving average (EMA) and short-time Fourier transform (STFT) of it. Holt EMA is used to smooth and track a slowly varying mean and its drift, and reduce noise with minimal phase lag. The STFT of a signal is used to track time varying spectra and to extract features for DSP analysis. We used this to pre-process data for the Python plotting algorithm, but we also used it to track major elements that change with time in the spectrogram of the received naïve signal.

Python Defense Processing

The second and last layer of defense takes the STFT data that is sent from the STM32 board to the user’s PC. Various thresholds are applied, including drop distance, STFT magnitudes over various frequencies, distance variance, and a “spectral power” measurement over a very recent window of distance values. If these thresholds are exceeded, it indicates the distance data received had an unnatural signal injected into it. Typically, this manifests itself as unusually fast changes, which exhibit high frequency components, or unreasonably still distance measurements from the effects of jamming after the drop.

Technical Issues

Environmental and Object Factors

We found that the material, shape, and motion of objects sensed by our defender affected the quality of results. Objects like a hand which didn’t cover the full FOV of the sensor would sometimes return overflow distance results, making our algorithm falsely think there was an attack. Shape and absorptive quality of the object also determined whether we got good distance data or not. Occasionally, vibrations on the surface we were testing on would affect distance data and we think that other vibrations at high frequencies could also prevent us from getting good data.

Threshold determination
When we were applying thresholds in our analysis of the distance time and frequency data, we found that different environments caused the hard-coded thresholds to fail. This was the tradeoff we made because we ran out of time, but future attempts to recreate this project should make a note of this. Uncalibrated thresholds will return junk attack classification.

Holt EMA application
We chose to plot the Holt EMA but use the raw, unsmoothed data in our analysis. This was because the EMA led to much smoother jumps during attacks, so it would have made analysis more difficult. If future attempts were to use this averaged distance data they should be sure to tread carefully or reconsider their choice.

Testing Our System

In our hardware testing, we primarily used Arduino programs with the built-in trigPulse and echoPulse functions, just to confirm interaction of the signals. We only did this on a repeatable level after we designed the mounting system on a piece of wood. Before that, it was very messy and unreliable.

Our software testing, aside from the interrupt bugs and the plotting errors, consisted mostly of looking for ways to detect the attack signals. We had a full range of distance-time and distance-frequency metrics to choose from, and after looking at how each attack affected these metrics in meaningful and distinct ways, we settled on a mix of time and frequency thresholds to tune. We found that the frequency characteristics of the single drop found in jamming were not different enough from the repeated drops in random spoofing, so we stuck with distance variance, range, and derivative range over a short time window. For spoofing detection, we used distance variance, derivative range, and frequency power over the same window.

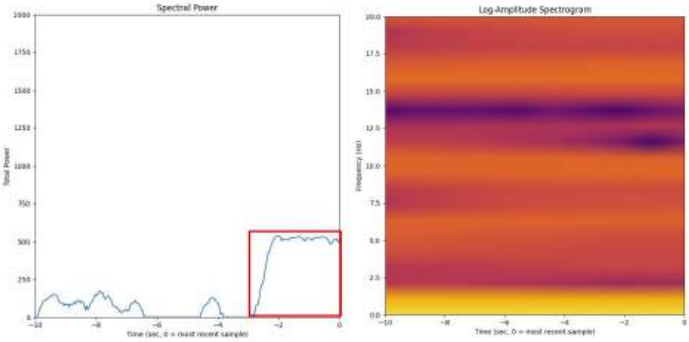
We tried to use scalable, relative thresholds but we were limited by time and the differences in testing environments and object distances. We settled on using hard-coded thresholds for most elements, although we did see that the derivative threshold for spoofing was about double that of jamming because spoofing involved spikes both up and down, while jamming saw a steep drop down only.

Results and Analysis

When testing our system, we were able to identify when our system was being attacked, as well as when it was in a state of no attack. We were able to do this due to the criterion described above, in addition to tweaking our bounds due to environmental impacts. Below, we have visualized the two main ways we identified attacks–spectral power density and STFT analysis. These main differentiating factors are what led us to successfully identify an attack, and tell the difference between random spoofing and jamming.

Random Spoofing Attack

When a naive sensor received a signal that was injected by random spoofing, the spectral power jumped up and stayed high throughout the duration of the spoofing. This high-frequency activity can be seen in the coloration and striation of the spectrogram, which allowed us to set thresholds for what was considered “too high” of a frequency activity to be due to normal operation.

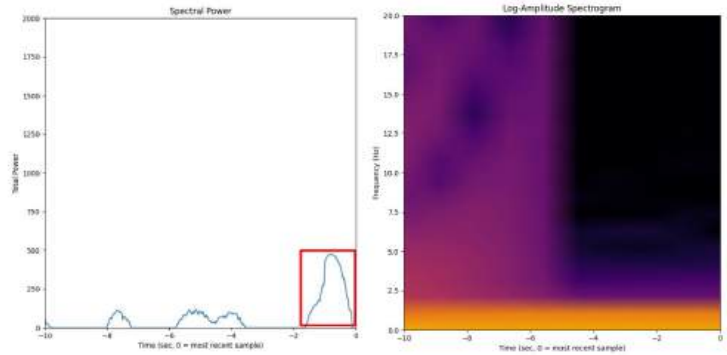


Spectral power jumps up and stays high - lots of freq. content

STFT shows higher values, including lines on certain freqs.

Jamming Attack

When a naive sensor received a signal that was injected by jamming, the spectral power jumped up and then dropped very low. This drastic cutoff can be seen in the high frequency activity coloration in the spectrogram.



Spectral power jumps, then drops incredibly low - very quiet signal

STFT shows a sharp increase in certain higher freq., then only DC

Future Work

With a better ultrasonic sensor that has better online documentation and more time, we firmly believe we could have achieved much better results. With these things, we could have dug deeper into the internals of the sensor, looking at the raw analog data gathered by the sensor, and been able to adjust more on the naïve defender side.

We could have adjusted the sensing period of the defender sensor, allowing us to reduce or avoid the periodic effects of jamming, which could have gotten us closer to fully cleaning the attacked signal on the naive sensor.

Due to the sensitivity and range limitation of the sensor we were using on our demo, we had to make adjustments to see the signal interactions we were prompting, such as setting specific thresholds in reference to the acoustic environment we were operating in. In the future, it would benefit our understanding of the demo and sensor interaction to make our system more robust and adaptive to multiple unique environments, not just the singular one we run our experiments in.