Reliable Echo Detection from FIUS Signals and Decision about Reliability. Error propogation calculation. Modelling of Ultrasonic signal intensity and spectra over distance and angle

Master in Engineering
Information Technology
Autonomous Intelligent Systems
Nastayeen Abdul Majid, Jenny Nadar,
Onyema Kizito Nnaemeka
1427970, 1427226, 1335745
nastayeen.abdul-majid@stud.fra-uas.de, jenny.nadar@student.fra-uas.de,
kizito.onyema@stud.fra-uas.de

***Abstract*—Autonomous driving necessitates accurate differentiation between objects and humans. Object detection is vital for driver assistance systems which rely on ultrasonic signal analysis. Our proposed system prioritizes first echo detection reliability testing and sensor system enhancement through machine learning model integration. By validating echo detection accuracy, system trustworthiness is improved. Training the MLP model on sensor data aims to enhance echo detection precision and efficiency. MLP's selection over other models is dictated by problem-specific requirements. Our approach aims to optimize performance and address challenges in autonomous driving scenarios.**

Keywords—FIUS sensor, first echo detection, fast fourier transform, machine learning, multi-layer perceptron, confusion matrix.

I. Introduction

Autonomous driving systems face a formidable challenge in distinguishing between objects and humans, particularly in pedestrian detection scenarios crucial for driver assistance systems. Current solutions heavily rely on intricate analysis of ultrasonic signals to differentiate pedestrians from other obstacles like cars. These systems integrate an ultrasonic transducer for signal emission and reception, alongside an embedded system for signal control and acquisition, and a computational unit for signal analysis and feature extraction. If the surface is abrasive with respect to the wavelength then the wave is not only reflected but also scattered and backscattered [1]. By scrutinizing backscattered ultrasonic signals, these systems extract features crucial for distinguishing pedestrians, thereby augmenting vehicle response in urban environments. Our proposed system focuses on two main objectives: enhancing echo detection reliability and improving sensor system performance through machine learning integration. Leveraging Multi-Layer Perceptron (MLP) architecture, our approach seeks to effectively address these challenges. Firstly, by developing a robust reliability test for echo detection, we aim to validate result accuracy and consistency, thereby bolstering system trustworthiness. Secondly, by training an MLP model on pertinent sensor data, we aim to refine echo detection accuracy and efficiency, facilitating more dependable identification of objects or obstacles. The choice of Multi-Layer Perceptron (MLP) over alternative machine learning models is contingent upon various factors, including the specific demands and intricacies of the problem being addressed.

II. Literature Review

A. Overview of FIUS Sensor

The FUIS Technology signal produces a versatile distance range, spanning from a minimum of 16 centimeters to a maximum of 6 meters. This extensive reach makes it a suitable choice for a wide array of applications, from proximity sensing in robotics to monitoring objects or obstacles in larger environments. The flexibility offered by the 16cm to 6m range enhances the module's utility, allowing it to cater to diverse scenarios where precise and reliable distance measurements are essential. The FUIS signal operates at a frequency of 40 kHz, utilizing ultrasonic waves within this frequency range to measure distances. The 40 kHz frequency is a common choice for ultrasonic sensors due to its balance between performance and practical considerations. This frequency provides a good compromise between accuracy and the ability to traverse various materials and environments effectively. The SRF02's use of a 40 kHz frequency ensures reliable and consistent distance measurements, making it well-suited for applications where precision and versatility are paramount, such as in robotics, automation, and object detection systems. The FUIS Sensor signal incorporates a fully timed echo system, relieving the host controller of the task of precisely timing the ultrasonic pulses. This feature streamlines the integration process by offloading the timing responsibilities to the sensor itself. With a built-in timing mechanism, the FUIS technology autonomously manages the emission of ultrasonic pulses and accurately measures the corresponding echoes, allowing the host controller to focus on other aspects of the application. The Signals generated by the in the FUIS Signal can be adjusted with different inputs, providing additional flexibility to customize and fine-tune the signals output and performance according to our project requirements. Analog inputs enable the adjustment of parameters related to sensitivity and gain, allowing the optimization of the signal responses to varying environmental conditions.

Fig. 1. FIUS sensor[8] is the measuring equipment which is a close integration of 3 main components. Fig. 2. Is ultrasonic sensor SRF02 with a mean frequency of 40 kHz and a power output of 150 mW (manufacturer's data) for sensing. It is a single transducer ultrasonic rangefinder in a tiny footprint PCB. For the above experiment setup, the FUIS sensor was used which is the main equipment for the Detection of Objects. The FUIS sensor projects directly to the Object of detection beaming signals directly on the object to detect its shape by means of ultrasound.

Embedded System Red Pitaya with a sampling frequency of 1.95 MS/s and a resolution of 14 14 bit for controlling the SRF02 and for analog digital conversion of the received signal. It helps in wireless transfer of data to laptop with UDP_Client Application (Fig. 4.), for additional processing and data collection. The captured signals from the sensor are computed, converted and extracted in a dataset using UDP software that shows the Readings and Measurement of the signals captured by the sensor at the placement of the Object. Several Projections must be made to get accurate results. For the Analysis and storing of the captured signal, a computer was used to set up the UDP software and Dataset storage.

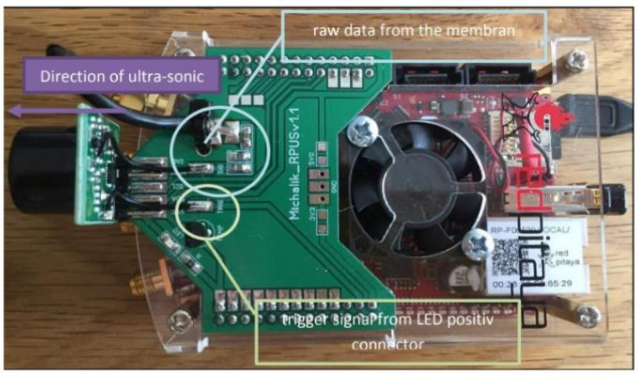

Fig. 1. FIUS sensor

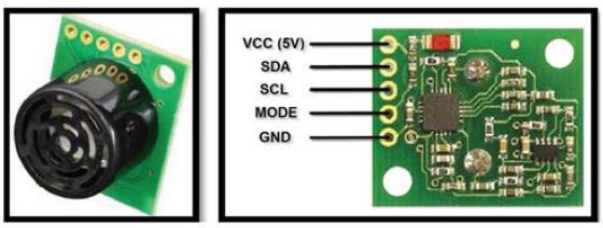

Fig. 2. Sonar sensor SRF02

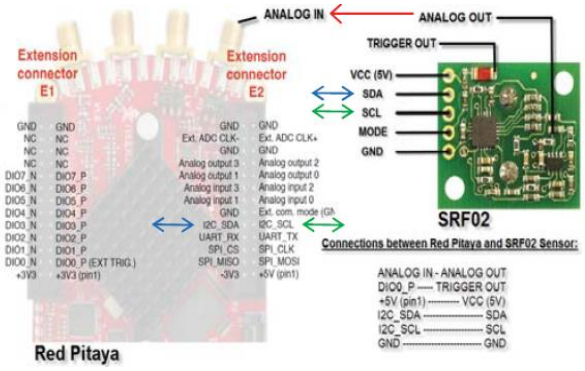

Fig. 3. Interface between Red Pitaya embedded system and ultrasonic sensor SRF02

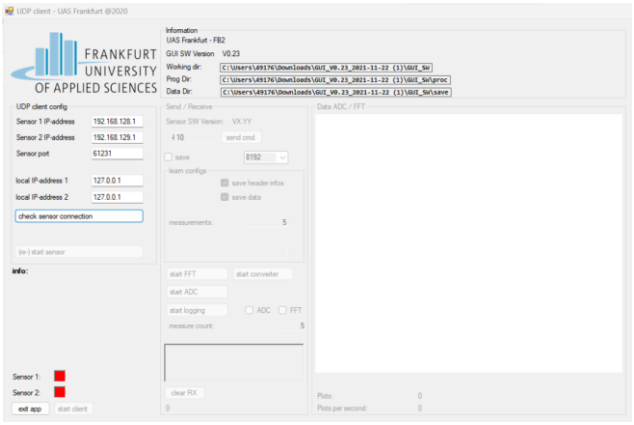

Fig. 4. UDP client application on laptop

B. *Principles of echo detection in FIUS technology*

Echo detection from FUIS technology, operate on the principle of detecting objects by means of emitting high-frequency sound waves, typically beyond the range of human hearing, and then detecting the echoes produced when these waves encounter an object. [6] Fig.5. The FUIS technology consists of a sensor with a transducer, often made of piezoelectric crystals, capable of converting electrical energy into mechanical vibrations to generate ultrasonic waves. These waves travel through the air until they encounter an object, at which point they bounce back as echoes to be received by the same transducer. The sensor sends a sound frequency of above 18 kHz in the air at the speed of 344 meter per second (at 20°C) and receives the reflected sound from the object. The key to the technical functionality of object detections using echo in FUIS, lies in measuring the time it takes for the emitted waves to travel to the object and back. Since the speed of sound in air is constant, this time measurement can be used to calculate the distance to the object. [7] Fig. 7. The distance (D) is determined using the formula:

$$D = (t \times v) / 2$$

where 't' is the time taken for the round trip of the ultrasonic waves, and 'v' is the speed of sound in the air.

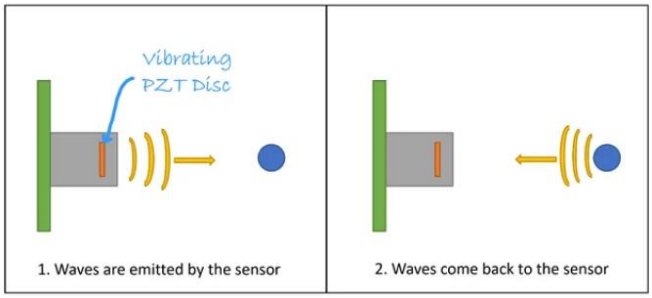

Fig. 5. Wave emission

C. Overview of Ultrasonic Distance Measurement

By processing these signals and employing signal conditioning techniques, the sensor provides accurate distance information, making it valuable for applications such as obstacle detection and avoidance in robotics.

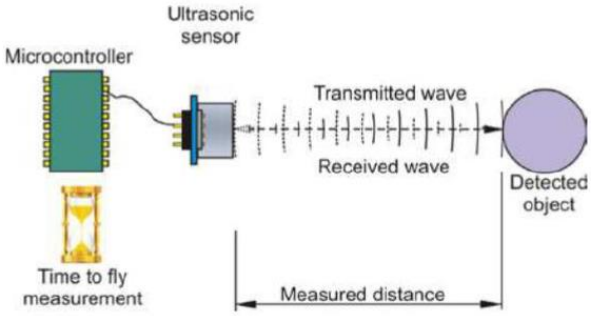

Fig. 6. Object detection theory

Ultrasonic sensors are commonly used for distance measurement in robotics, industrial automation, and automotive systems. These sensors utilize ultrasonic waves, which are sound waves with frequencies above the audible range for humans, typically in the range of 20 kHz to several MHz.

Some of the distance measurement methods using ultrasonic sensors:

1) Time-of-Flight Techniques: The time of flight monitors the distance in the time domain between two waveforms and is used to measure the speed of sound in the process material [2]. Ultrasonic sensors measure the time it takes for an ultrasonic pulse to travel from the sensor to the target object and back. This time is directly proportional to the distance between the sensor and the object.

2) Pulse-Echo Systems: The ultrasonic sensor emits a short ultrasonic pulse, and the same sensor detects the echo when the pulse reflects off an object. The time delay between the emission and reception of the pulse is used to calculate the distance. It improves the resolution and reliability of distance measurements. Techniques such as matched filtering and deconvolution have been explored to enhance the accuracy of echo detection.

3) Multiple Sensor Configurations: It is use of multiple ultrasonic sensors in an array to overcome limitations of blind spots and improve coverage. Here are some common multiple sensor configurations used in ultrasonic distance measurement.

4) Sensor arrays: Multiple ultrasonic sensors are arranged in an array, covering different directions or sections of the environment. It provides a broader field of view and helps in minimizing blind spots.

5) Triangulation: It enhances accuracy and precision in distance measurements.

6) Sensor fusion: It involves combining data from different types of sensors, such as ultrasonic sensors, cameras, radar, and lidar. Combining ultrasonic sensors with vision sensors can improve object recognition and provide additional context for distance measurements.

7) Redundant sensors: If one sensor fails or provides inaccurate data, redundant sensors can act as backups, maintaining the system's functionality. Redundancy is crucial in safety-critical applications.

8) Temperature Compensation: New innovative temperature compensation algorithms and sensor designs mitigate the impact of environmental conditions on ultrasonic distance measurement accuracy.

9) Phase-Shift techniques: Involves analyzing the phase difference between transmitted and received ultrasonic waves to determine the distance to an object. This technique is based on the principle that the phase of a wave changes as it travels through a medium or reflects off an object. By measuring the phase shift, the system can calculate the distance between the ultrasonic sensor and the object. The relationship between phase shift and distance is often linear, allowing for a straightforward conversion.

10) Continuous Wave Doppler Methods: It is used in ultrasonic sensing to measure the velocity of a moving object by analyzing the Doppler shift in the frequency of continuous ultrasonic waves. The Doppler effect describes the change in frequency or wavelength of a wave in relation to an observer moving relative to its source.

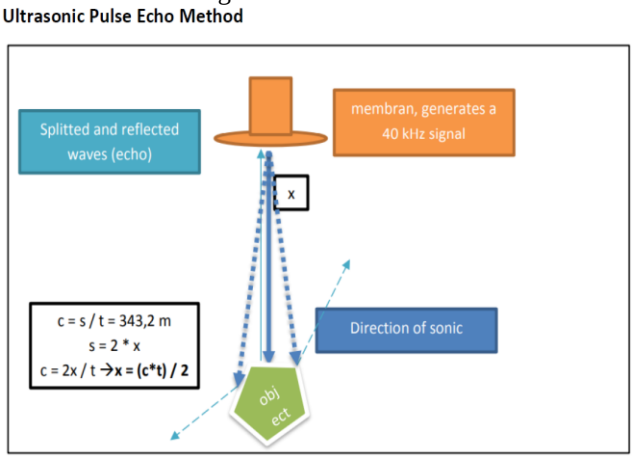

Fig. 7. Ultrasonic pulse echo method

D. Fast Fourier Transform

The Fast Fourier Transform (FFT) is a widely used algorithm for efficiently computing the Discrete Fourier Transform (DFT) of a sequence or signal. One should also note that multiplication in the time t corresponds to convolution in the frequency at domain [3]. It allows for the rapid computation of the frequency components present in a signal by converting it from the time domain to the frequency domain. It speeds up the computation of the DFT by exploiting symmetries and redundancies in the calculation. Instead of performing N^2 complex multiplications required by the straightforward DFT algorithm, the FFT reduces the complexity to N log(N) operations, making it much faster for large input sizes. The FFT algorithm works by recursively dividing the input sequence into smaller subsequences, applying DFT computations to each subsequence, and then combining the results using various mathematical operations, such as

additions and multiplications. This divide-and-conquer approach allows for significant computational savings compared to the direct computation of the DFT.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-i\,2\pi\,k\,n\,/\,N}$$

Fig. 6. Fast fourier transform formula

E. Inverse Fast Fourier Transform

The Inverse Fast Fourier Transform (IFFT) is a mathematical operation that reverses the process of the Fast Fourier Transform (FFT). It converts frequency-domain representations of signals back into the time-domain. In essence, it takes a signal's frequency components and reconstructs the original signal in the time domain. This operation is invaluable in many signal processing applications, including audio and image processing, where signals are often analysed and manipulated in the frequency domain before being transformed back to the time domain for further processing or interpretation.

F. Machine Learning Models

1) Convolutional Neural Network (CNN): CNN is a deep neural network commonly used for processing grid-like data, such as images or time-series data. It consists of convolutional layers, pooling layers, and fully connected layers. CNNs are adept at capturing spatial hierarchies in data, making them particularly effective for tasks like image classification, object detection, and image segmentation.

2) Long Short-Term Memory (LSTM): LSTM is a type of recurrent neural network (RNN) architecture designed to overcome the vanishing gradient problem in traditional RNNs. LSTMs introduce specialized memory cells and gating mechanisms that allow them to capture long-range dependencies and remember information over extended time steps. They are widely used for sequential data processing tasks such as speech recognition, language modeling, and time-series forecasting.

3) Multi-Layer Perceptron (MLP): It is a feedforward neural network consisting of multiple layers of neurons, including an input layer, one or more hidden layers, and an output layer. Each neuron in one layer is connected to every neuron in the subsequent layer. The formation of a neural network is created by an “input” layer, one or quite one “hidden” layer(s), and also the “output” layer [4]. MLPs are versatile and can be applied to various tasks, including classification, regression, and pattern recognition. They are often used as a baseline model for comparison with more complex neural network architectures.

4) Random Forest Regressor: Random Forest is an ensemble learning method that constructs a multitude of decision trees during training and outputs the average prediction of the individual trees (regressor) or the mode of the classes (classifier). Each decision tree in the forest is trained on a random subset of the training data and a random subset of the features. Random Forests are robust against overfitting and can handle high-dimensional data efficiently.

G. Confusion Matrix

A confusion matrix of size n x n associated with a classifier shows the predicted and actual classification, where n is the number of different classes [5]. A confusion matrix is a table used in classification to evaluate the performance of a machine learning model. It allows visualization of the performance of an algorithm by presenting the actual and predicted classes in a tabular format. Four main components are:

1) True Positives (TP): The number of instances correctly predicted as positive by the model. For example, if the actual class is positive (1) and the model predicts it correctly as positive, it is a true positive.

2) True Negatives (TN): The number of instances correctly predicted as negative by the model. For example, if the actual class is negative (0) and the model predicts it correctly as negative, it is a true negative.

3) False Positives (FP): These are instances where the model incorrectly predicts a positive class when the actual class is negative. For example, if the actual class is negative, but the model predicts it as positive, it is a false positive.

4) True Negatives (FN): These are instances where the model incorrectly predicts a negative class when the actual class is positive. For example, if the actual class is positive, but the model predicts it as negative, it is a false negative.

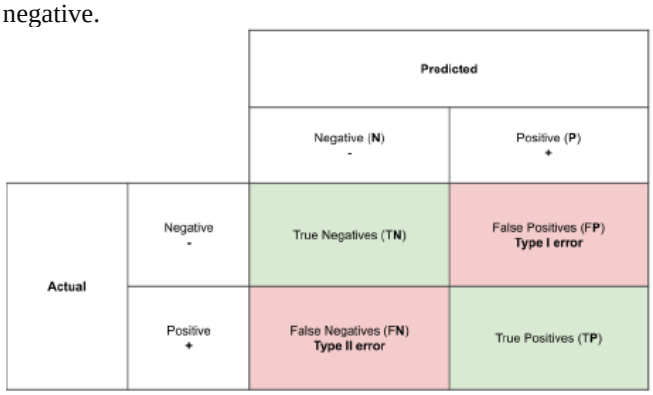

Fig. 8. Confusion matrix

III. METHODOLOGY

A. Existing System

In urban autonomous driving scenarios, distinguishing between objects and humans poses a significant challenge for vehicle systems. One approach to address this challenge is through pedestrian detection, a critical aspect of driver assistance systems. In existing systems, pedestrian detection relies on sophisticated analysis of ultrasonic signals. The primary goal is to enable vehicles to differentiate between pedestrians and cars as obstacles in their path. This is achieved through specialized signal analysis methods applied to ultrasonic signals reflected from obstacles. The system comprises an ultrasonic transducer, responsible for emitting and receiving signals, an embedded system for

transducer control and signal acquisition, and a computer for signal analysis and feature extraction. By analyzing the backscattered ultrasonic signals, the system identifies and extracts relevant features to distinguish pedestrians from other objects, thereby enhancing the vehicle's ability to respond appropriately in urban environments.

B. Proposed System

In our proposed system, we prioritize the development of a robust echo detection reliability test and the enhancement of the sensor system through integration with a machine learning model. Our approach involves leveraging Multi-Layer Perceptron (MLP) architecture to address these challenges effectively. By developing a comprehensive reliability test, we aim to validate the accuracy and consistency of echo detection results, thereby enhancing the trustworthiness of the system. By training the MLP model on relevant data acquired from the sensor system, we can improve the accuracy and efficiency of echo detection, enabling more reliable and precise identification of objects or obstacles.

The decision to choose Multi-Layer Perceptron (MLP) over other machine learning models depends on various factors and the specific requirements of the problem at hand.

1) Non-Liner mapping: MLPs can learn and approximate non-linear mappings between input and output variables. This makes them particularly effective in tasks where the underlying relationships in the data are complex and not easily captured by linear models.

2) Feature Learning: MLPs are capable of automatically learning relevant features from raw data, reducing the need for manual feature engineering. This can be advantageous in scenarios where the input data has high dimensionality or contains complex patterns that are difficult to extract manually.

3) Scalability: MLPs can be scaled up by increasing the number of layers and neurons, allowing for the development of deeper and more complex architectures. This scalability enables MLPs to handle large-scale datasets and capture intricate relationships in the data.

4) Performance: MLPs can achieve competitive performance on a variety of tasks, especially when trained on large and diverse datasets. With appropriate hyperparameter tuning and regularization techniques, MLPs can often achieve state-of-the-art results in many applications.

IV. IMPLEMENTATION AND EXECUTION

Detecting the first reflection in ultrasonic sensor data using machine learning typically involves training a model to recognize the signature patterns associated with the first reflected signal.

A. Data Collection

Gathering a dataset containing ultrasonic sensor readings in different scenarios where the first reflection needs to be detected. This includes instances with and without reflections, varying distances, and different environmental conditions. In Machine Learning laboratory, the FIUS sensor is mounted to the top over a long-black metal stand. The object to be measured is placed over white-short stand directly under the FIUS sensor. In other, to properly project the beams from the FUIS sensor on the desired object of capture, a firm stand which grips the sensor is placed above the object of capture at a certain height. From the grip of the sensor on a stand certain vibrational noises are avoided while capturing the object shape. Objects desired to be captured are placed on a roller or moveable table to reach an exact position of capture within the range of the signal beams from the sensor. Objects must be within the range of the sensor beam. Markers are required and used in the Experiment to appropriately map out the exact ranges of the sensors beam. This helps to prevent objects getting into the range of the signal beam during capture. To ensure the accurate placement of the objects, certain distances must be met. Measuring Tapes are of great utilization in the experiment to obtain accurate distances and length during the experiment. During the Experiment certain shapes and density of objects are to be captured. These Objects could be hard or soft. By Projection of the FUIS sensor beam, the signal indicates a change when a new Object enters its range of beams, which can be analyzed after capture. At the cause of the experiment, thousands and repetitive captures are required to properly have enough data required to train a Model for Object Identification.

1) Detection of hard-object: In Fig. 8. we placed a hard-box under sensor at distance of 1 meter and 15 cm by using a measuring scale. We noted the difference in distances which is manually measured with that shown on UDP_client application and collected 1000 ADC data for 5 times. Ultrasonic Sensor SRF02 with a mean frequency of 40 kHz and a power output of 150 mW (manufacturer's data) for sensing.

2) Detection of human: A person stood under the sensor at distance of 1 meter and 15 cm by using a measuring scale (Fig. 9.). Pedestrian width is lognormally distributed, so is the joint distribution of width and height[6]. We noted the difference in distances which is manually measured with that shown on UDP_client application and collected 1000 ADC data for 5 times.

Fig. 9. Detection of hard object

Fig. 10. Detection of human

B. Data Preprocessing

```
import pandas as pd
import numpy as np
from scipy.fft import fft
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras import layers, models
from scipy.signal import find_peaks
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score, f1_score
from tensorflow.keras import models, layers, regularizers,  callbacks
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

Fig. 11. Python libraries implemented

1) Python Libraries

a) pandas: Python library for data manipulation and analysis, providing data structures and functions to work with structured data.

b) numPy: Fundamental package for numerical computing in Python, offering powerful array operations and mathematical functions.

c) scipy.fft: Partof the SciPy library, providing functions for computing Fast Fourier Transforms (FFT) and related operations.

d) sklearn.model_selection: Function from the scikit-learn library for splitting datasets into training and testing subsets for machine learning model evaluation.

e) tensorflow: Open-source machine learning framework developed by Google, enabling the creation and training of deep learning models with ease.

f) scipy.signal: A function that detects peaks in a 1-dimensional array or time series data.

g) sklearn.metrics.confusion_matrix: A function that computes a confusion matrix to evaluate the performance of a classification model.

h) sklearn.metrics.recall: A function that computes the recall (sensitivity) score, which measures the ability of a classifier to correctly identify positive instances.

i) F1_score: A function that computes the F1 score, which is the harmonic mean of precision and recall, providing a balanced measure of a classifier's performance in binary classification tasks.

j) tensorflow.keras.models: Importing modules for defining and training neural network models using TensorFlow's Keras API.

k) tensorflow.keras.layers: Importing modules for building layers of neural networks, such as dense (fully connected), convolutional, and recurrent layers.

l) tensorflow.keras.regulizers: Importing modules for applying regularization techniques to neural network models, such as L1 and L2 regularization.

m)tensorflows.keras.regulizers: Importing modules for defining callbacks during model training, such as early stopping and learning rate adjustment.

n) matplotlib.pyplot: Importing modules for creating visualizations, such as plots and charts, using Matplotlib's Pyplot interface.

o) seaborn: Importing modules for statistical data visualization, providing a high-level interface for drawing attractive and informative statistical graphics.

p) os: Importing modules for interacting with the operating system, enabling tasks such as file and directory manipulation.

2) Variable Declaration for echo detection analysis

```
#****************** Variable defination ********************

'# Directory containing the files'
directory = "C:/Users/49176/Desktop/AIS& ML/Datasets"
velocity_of_sound = 343 # meters per second (for air at roo
time_delay_microseconds = 2  # microseconds
data= None
'# Define window size and overlap'
window_size = 64  # microseconds
overlap = 0  # no overlap for now
# Initialize variables for hits and fails
hits = 0
fails = 0
false_positives = 0
true_negatives = 0
'# Initialize confusion matrix'
conf_matrix = [[0, 0], [0, 0]]  # [[True Positive, False Ne
```

Fig. 12. Global variable declaration

a) Directory: Directory path containing the files for analysis.

b) Velocity of sound: Speed of sound in meters per second, typically for air at room temperature.

c) Time delay in microseconds: Time delay in microseconds, often used in signal processing calculations.

d) Data: Placeholder variable for storing data, typically used for loading datasets.

e) Window size: Size of the window in microseconds used for signal analysis.

f) Overlap: Amount of overlap between consecutive windows, often used in signal processing techniques.

g) Hits, fails, false positives, true negatives: Variables for counting hits, fails, false positives, and true negatives in echo detection analysis.

h) Confusion matrix: Matrix for tracking the performance of the echo detection algorithm, with entries for true positives, false negatives, false positives, and true negatives.

Next, the function calculates the distance of the maximum peak detected in a signal based on the position of the peak and the velocity of sound. It multiplies half of the position of the maximum peak by the velocity of sound and returns the calculated distance.

```
# Define a function to calculate distance based on frequency
def calculate_distance(position_of_max_peak,velocity_of_sound):
    # Calculate the distance of the maximum peak
    distance_of_max_peak = 0.5 * position_of_max_peak * velocity_of_sound
    return distance_of_max_peak
```

Fig. 13. Maximum distance calculation

```
def apply_window_and_fft(row):
    # Apply your window function here
    windowed_row = row * np.ones_like(row)  # Example: Using rectangle window
    # windowed_row = row * np.hamming(len(row))
    fft_data = np.fft.fft(windowed_row)
    # Find the index of the maximum magnitude component in the FFT data
    max_magnitude_index = np.argmax(np.abs(fft_data))
    # Calculate the corresponding frequency
    sampling_rate = 125000000  # Example: Sampling rate (adjust according to your data)
    max_frequency = max_magnitude_index * sampling_rate / len(fft_data)
    # Construct a frequency domain signal with only the component corresponding to the maximum frequency
    max_freq_signal = np.zeros_like(fft_data)
    max_freq_signal[max_magnitude_index] = fft_data[max_magnitude_index]
    # Perform inverse FFT on the maximum frequency component
    ifft_result = np.fft.ifft(max_freq_signal)
    # Get only the real part of the inverse FFT result
    ifft_real_part = np.real(ifft_result)
    # Find peaks in the real part of the inverse FFT result
    peaks_indices, _ = find_peaks(ifft_real_part)
    # Get the peak values
    peaks_values = ifft_real_part[peaks_indices]
    # Check if peaks_values is empty
    if len(peaks_values) == 0:
        # Handle the case where no peaks are found
        return 0
    # Find the index of the maximum peak value
    max_peak_index = np.argmax(peaks_values)
    # Get the maximum peak value and its index
    max_peak_value = peaks_values[max_peak_index]
    max_peak_index_absolute = peaks_indices[max_peak_index]
    # Calculate the position of the maximum peak in terms of time delay
    position_of_max_peak = max_peak_index_absolute * (2 * time_delay_microseconds) / len(ifft_real_part)
    # Calculate the distance corresponding to the maximum frequency
    max_distance = calculate_distance(position_of_max_peak, velocity_of_sound)
    return position_of_max_peak
```

Fig. 14. Apply window size and FFT

The function *apply_winow_and_fft* is designed to process a given signal for echo detection purposes, employing several signal processing techniques to extract relevant information. Initially, the function applies a window function to the input signal, which helps mitigate spectral leakage and enhances the resolution of frequency components. Subsequently, the Fast Fourier Transform (FFT) is computed on the windowed signal, converting it from the time domain to the frequency domain. The function then identifies the index of the maximum magnitude component in the FFT data, indicating the dominant frequency or peak within the signal. Using this index and the sampling rate, the corresponding frequency of the peak is calculated. Next, an inverse FFT is performed to isolate the component associated with the maximum frequency, facilitating further analysis. Peaks are detected in the real part of the inverse FFT result, and the function determines the position of the maximum peak, representing the time delay. Finally, the distance corresponding to the maximum frequency is computed based on the time delay and the velocity of sound, utilizing a helper function *calculate_distance*. Overall, *apply_winow_and_fft* serves as a comprehensive signal processing module for echo detection applications, extracting key features from the input signal to facilitate accurate distance estimation and object recognition.

C. Data Labeling

The *process_file* function (Fig. 14.) is responsible for processing a given file containing data relevant to echo

detection. It begins by extracting information from the file name, such as the manual distance measurement *manual_distance_dML*, which is essential for comparison with calculated distances. The function then reads the data from the file, selecting relevant columns for analysis. For each row of data, it applies the *apply_winow_and_fft* function to perform windowing and FFT, extracting the maximum distance detected from the signal. Subsequently, it computes the deviation between the calculated distance and the manual distance measurement, extracting features for further analysis. These features, along with their corresponding labels based on deviation thresholds, are appended to feature matrix X and label vector y respectively. Overall, *process_file* function orchestrates the extraction of features and labels from the file data, facilitating subsequent training and evaluation of echo detection models.

```
def process_file(file_path):

    #Apply feature extraction to the data and create feature matrix X and la
    file_name = os.path.basename(file_path)
    file_name_without_extension = os.path.splitext(file_name)[0]
    parts = file_name_without_extension.split("_")
    for part in parts:
        if "cm" in part:
            manual_distance_dML = float(part.replace("cm", "")) / 100  # Con
            break
        elif "m" in part:
            manual_distance_dML = float(part.replace("m", ""))  # ALready in
            break
    if manual_distance_dML is not None:
        print("Distance for", file_name, ":", manual_distance_dML, "meters")
    else:
        print("Distance information not found in file name for", file_name)

    data = pd.read_excel(file_path)
    data = data.iloc[:, 16:]
    for index, row in data.iterrows():
        # Apply windowing and FFT function to each row
        max_distance = apply_window_and_fft(row)
        max_distance = float(round(max_distance, 2))
    for index, row in data.iterrows():
        max_distance, deviation = extract_features(row,manual_distance_dML)
        X.append([max_distance, deviation])
        # Labeling based on deviation threshold
        if deviation < 0.01:
            y.append(1)  # Hit
        else:
            y.append(0)  # Fail
    return X, y
```

Fig. 15. Error finding and labeling

D. Model Training

The dataset is prepared for training and testing a Multi-Layer Perceptron (MLP) model for echo detection. The features (X) and labels (y) extracted from the data are converted into NumPy arrays. The data is then split into training and testing sets using the *train_test_split* function, with 80% of the data allocated for training and 20% for testing, ensuring randomization with a fixed seed (random_state=42) for reproducibility.

Next, an MLP model is defined using the Keras Sequential API, consisting of several dense (fully connected) layers. The input layer has 128 neurons, followed by additional hidden layers with 64 and 32 neurons, respectively, each using ReLU activation functions. The output layer consists of a single neuron with a sigmoid activation function, suitable for binary classification tasks.

The model is compiled using the Adam optimizer and binary cross-entropy loss function, while accuracy is selected as the metric to monitor during training. Additionally, a learning rate scheduler (*lr_scheduler*) is defined to adjust the learning rate dynamically during training, reducing it by a factor of 0.5 if no improvement in the validation loss is observed after three epochs.

Finally, the model is trained on the training data (X_train and y_train) for 20 epochs with a batch size of 32. The training process is verbose=1, displaying progress updates during each epoch, and the learning rate scheduler is applied as a callback to adaptively adjust the learning rate

```
X = np.array(X)
y = np.array(y)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
model = models.Sequential([
    layers.Dense(128, activation='relu', input_shape=(2,)),  # Increased neurons in the firs
    layers.Dense(64, activation='relu'),                     # Additional hidden layer with
    layers.Dense(32, activation='relu'),                     # Original hidden layer
    layers.Dense(1, activation='sigmoid')
])
lr_scheduler = callbacks.ReduceLROnPlateau(factor=0.5, patience=3, verbose=1)
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=1, callbacks=[lr_scheduler])
```

Fig. 16. relu activation

V. Critical Review Of Results And Validation

The trained MLP model is evaluated using the testing data (X_test and y_test). The *evaluate* function computes the test loss and accuracy of the model, providing insights into its performance on unseen data. Additionally, the model makes predictions on the testing data using the *predict* function, and these predictions are converted into binary form based on a threshold of 0.5, classifying each instance as either 0 or 1.

Subsequently, performance metrics such as recall and F1 score are computed using the *recall_score* and *f1_score* functions, respectively, to assess the model's ability to correctly identify positive instances (hits) while accounting for class imbalance. Moreover, a confusion matrix is generated using the *confusion_matrix* function, which provides a comprehensive summary of the model's classification results, including true positives, false positives, true negatives, and false negatives.

To visualize the confusion matrix, a heatmap is created using Matplotlib and Seaborn libraries. The heatmap displays the confusion matrix with annotations, facilitating the interpretation of model performance. Each axis of the heatmap corresponds to the actual and predicted classes, while the color intensity represents the number of instances classified into each category. Overall, this analysis offers a concise summary of the MLP model's performance in echo

detection, aiding in understanding its strengths and weaknesses.

```
test_loss, test_acc = model.evaluate(X_test, y_test)
predictions = model.predict(X_test)
binary_predictions = (predictions > 0.5).astype(int)
recall = recall_score(y_test, binary_predictions, zero_division=0)
f1 = f1_score(y_test, binary_predictions, zero_division=0)
conf_matrix = confusion_matrix(y_test, binary_predictions)
labels = ['True Negative', 'False Positive', 'False Negative', 'True Positive']
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=labels, yticklabels=labels)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

Fig. 17. Confusion matrix code

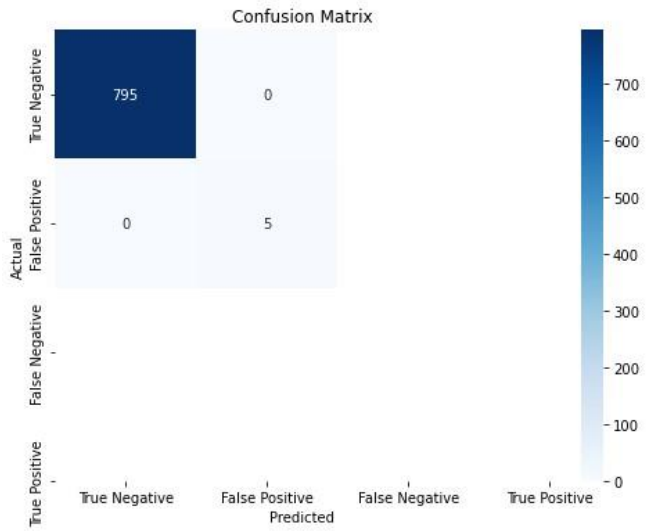

Fig. 18. Output of confusion matrix

```
Test accuracy: 1.0
25/25 [==============
Recall: 1.0
F1 Score: 1.0
Confusion Matrix:
[[795   0]
 [  0   5]]
```

Fig. 19. Output of accuracy, recall, f1 score

The provided evaluation metrics and confusion matrix indicate the performance of a classification model on a binary classification task. The recall, also known as the true positive rate, indicates the proportion of actual positive instances (hits) that were correctly predicted by the model. A recall score of 1.0 suggests that the model correctly identified all positive instances in the dataset, indicating perfect performance in capturing all true positives. The F1 score is the harmonic mean of precision and recall, providing a single metric that balances both precision and recall. A perfect F1 score of 1.0 indicates optimal performance, where the model achieves both high precision and high recall. The confusion matrix is a tabular representation of the model's predictions compared to the ground truth labels. In this case, the confusion matrix shows a total of 800 instances, where 795 instances were correctly classified as the negative class (true negatives) and 5 instances were correctly classified as the positive class (true positives). There are no false positives or false negatives, indicating that the model made no errors in its predictions. Overall, the evaluation metrics and confusion matrix suggest that the model achieved perfect performance, correctly classifying all instances in the dataset.

A. Model Optimization

To improve the performance of our neural network model, we used the following strategies:

1) Improvement of decision speed and measurement: Stochastic Gradient Descent (SGD) is a optimization algorithm used to minimize the loss function and update the parameters of a model iteratively. It operates by randomly selecting a subset of training data (a mini-batch) at each iteration and computing the gradient of the loss function with respect to the parameters based on that mini-batch. The parameters are then updated in the opposite direction of the gradient to reduce the loss.

2) Adam optimization: It is adaptive optimization algorithm that combines the advantages of both AdaGrad and RMSProp. It maintains separate adaptive learning rates for each parameter by keeping track of both the first and second moments of the gradients.

3) Increase Model Complexity: Added more layers or increase the number of neurons in existing layers. This allows the model to learn more complex patterns in the data.

4) Regularization: Applied dropout regularization to prevent overfitting. Dropout randomly drops a fraction of neurons during training, forcing the model to learn more robust features.

5) Batch Normalization: Added batch normalization layers after each hidden layer to normalize the activations, which can accelerate training and improve convergence.

6) Learning Rate Scheduling: Used learning rate schedules such as ReduceLROnPlateau or exponential decay to adaptively adjust the learning rate during training. This can help the model converge faster and avoid getting stuck in local minima.

7) Optimization Algorithm: Experiment with different optimizers such as RMSprop or SGD with momentum, in addition to Adam. Each optimizer has different properties and may perform better on certain types of data or architectures.

8) Activation Functions: Different activation functions in hidden layers, such as Leaky ReLU or ELU, to mitigate the vanishing gradient problem and encourage faster convergence.

9) Initialization: Using different weight initialization schemes such as Glorot uniform or He uniform initialization. Proper initialization can help alleviate the vanishing or exploding gradient problem and improve training stability.

10) Model Ensemble: Train multiple neural network models with different initializations or architectures and combine their predictions using techniques like averaging or

stacking. Ensemble methods often lead to improved performance by leveraging the diversity of individual models.

11) Hyperparameter Tuning: Systematically search for optimal hyperparameters using techniques like grid search, random search, or Bayesian optimization. Tune parameters such as the learning rate, dropout rate, batch size, and number of epochs.

12) Early Stopping: Implement early stopping to prevent overfitting and find the optimal number of training epochs. Monitor the validation loss during training and stop training when it starts to increase consistently.

B. Deployment

GUI development for signal visualization and settings adjustment. A graphical user interface (GUI) was designed and implemented to visualize signals and results in real-time. The GUI provides interactive features for adjusting settings such as the length and position of the time window relative to the first echo. Additionally, the GUI allows users to monitor signal quality and make informed decisions based on visual feedback. The enhanced GUI provides users with a seamless and intuitive interface for interacting with the sensor system, facilitating efficient data analysis and system optimization.

This code snippet illustrates the creation of a custom Tkinter application using the cutomtkinter library, enhancing the appearance and functionality of the user interface. The App class inherits from customtkinter.CTk, providing customized features for the application's appearance and behavior. Within the App class, initialization methods configure the application window, layout, and various widgets, including labels, buttons, and option menus. Widgets are organized using grid layout management for proper alignment and resizing. Event handling methods are defined to respond to user interactions, such as button clicks or option menu selections. Default values for appearance mode and UI scaling are set within the class. *Event handling methods, such as open_input_dialog_event, change_appearance_mode_event, change_scaling_event, sidebar_button_event are defined to handle specific user actions triggered by button clicks or menu selections. These methods manage tasks such as opening input dialogs, changing appearance modes, adjusting UI scaling, and responding to sidebar button clicks.*

```
import customtkinter

customtkinter.set_appearance_mode("System")
customtkinter.set_default_color_theme("blue")


1 usage
class App(customtkinter.CTk):
    def __init__(self):
        super().__init__()

        # configure window
        self.title("Peak Echo Measurement")
        self.geometry(f"{1100}x{580}")

        # configure grid layout (4x4)
        self.grid_columnconfigure( index: 1, weight=1)
        self.grid_columnconfigure( index: (2, 3), weight=0)
        self.grid_rowconfigure( index: (0, 1, 2), weight=1)

        # create sidebar frame with widgets
        self.sidebar_frame = customtkinter.CTkFrame(
            self, width=140, corner_radius=0)
        self.sidebar_frame.grid(
            row=0, column=0, rowspan=4, sticky="nsew")
        self.sidebar_frame.grid_rowconfigure( index: 4, weight=1)
        self.logo_label = customtkinter.CTkLabel(
            self.sidebar_frame, text="Echo Detection",
            font=customtkinter.CTkFont(size=20, weight="bold"))
        self.logo_label.grid(row=0, column=0, padx=20, pady=(20, 10))
        self.sidebar_button_1 = customtkinter.CTkButton(
            self.sidebar_frame, command=self.sidebar_button_event,
            text="Select File")
        self.sidebar_button_1.grid(
            row=1, column=0, padx=20, pady=10)
        self.appearance_mode_label = customtkinter.CTkLabel(
            self.sidebar_frame, text="Appearance Mode:", anchor="w")
        self.appearance_mode_label.grid(
            row=5, column=0, padx=20, pady=(10, 0))
        self.appearance_mode_optionemenu = customtkinter.CTkOptionMenu(
            self.sidebar_frame,values=["Light", "Dark", "System"],
```

Fig. 20. GUI code

```
        self.appearance_mode_optionemenu = customtkinter.CTkOptionMenu(
            self.sidebar_frame,values=["Light", "Dark", "System"],
            command=self.change_appearance_mode_event)
        self.appearance_mode_optionemenu.grid(
            row=6, column=0, padx=20, pady=(10, 10))
        self.scaling_label = customtkinter.CTkLabel(
            self.sidebar_frame, text="UI Scaling:", anchor="w")
        self.scaling_label.grid(row=7, column=0, padx=20, pady=(10, 0))
        self.scaling_optionemenu = customtkinter.CTkOptionMenu(
            self.sidebar_frame, values=["80%", "90%", "100%", "110%", "120%"],
            command=self.change_scaling_event)
        self.scaling_optionemenu.grid(row=8, column=0, padx=20, pady=(10, 20))

        # set default values
        self.appearance_mode_optionemenu.set("Dark")
        self.scaling_optionemenu.set("100%")

    def open_input_dialog_event(self):
        dialog = customtkinter.CTkInputDialog(
            text="Type in a number:", title="CTkInputDialog")
        print("CTkInputDialog:", dialog.get_input())

    1 usage
    def change_appearance_mode_event(self, new_appearance_mode: str):
        customtkinter.set_appearance_mode(new_appearance_mode)

    1 usage
    def change_scaling_event(self, new_scaling: str):
        new_scaling_float = int(new_scaling.replace( _old: "%",  _new: "")) / 100
        customtkinter.set_widget_scaling(new_scaling_float)

    1 usage
    def sidebar_button_event(self):
        print("sidebar_button click")


if __name__ == "__main__":
    app = App()
    app.mainloop()
```

Fig. 21. GUI code

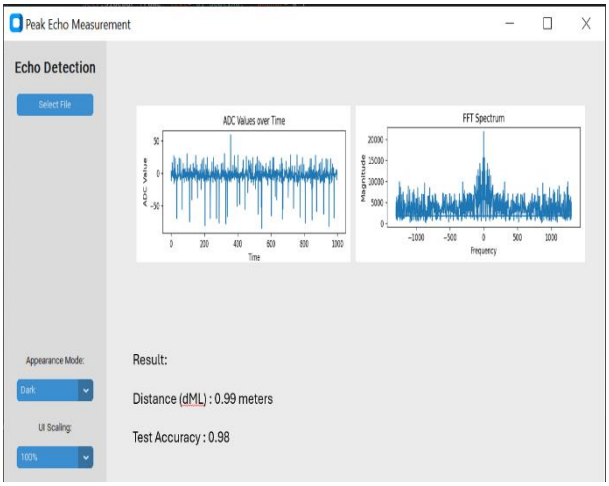

Fig. 22. Interactive GUI

VI. Challenges

The reliability of detecting objects accurately with the FUIS technology based on the FUIS signals can be affected by certain factors considering that it's mechanism and principle relies on the accurate delivering of sound waves within the object of detection from the FUIS system. Extreme temperatures and humidity levels can affect the speed of sound in the air, leading to inaccuracies in distance measurements. Compensation mechanisms or additional sensors may be required to account for these variations. Changes in air density due to factors like air flow or the presence of particulate matter can impact the propagation of ultrasonic waves in FUIS technology, affecting the accuracy of distance measurements. Small or irregularly shaped objects may not reflect sufficient ultrasonic waves back to the FUIS sensor, resulting in detection challenges. Additionally, the sensor might have difficulty distinguishing between closely spaced objects. The composition and surface characteristics of objects can affect the reflection of ultrasonic wave during experiments. Absorbent or soft materials may absorb the waves, while highly reflective surfaces might cause multiple reflections and interfere with accurate measurements. The beam width of the FUIS sensors can affect the detection area and accuracy. Understanding the beam pattern of the FUIS sensor is crucial for proper sensor placement and range coverage. Improper mounting or misalignment of the sensor can lead to inaccurate readings. Careful consideration of the sensor's field of view and the positioning of obstacles is essential to ensure reliable detection of objects. The Distance at which Objects are placed could affect the accurate detection of the Objects considering that FUIS sensors have certain range and distance at which their beams can reach.

VII. Summary

The objective of reliability of determining the distance from first echo position has been met by using MLP model. After model training, the test data is correctly classified as hit or fail based on error which is less than 1 cm. By prioritizing reliability testing, the system ensures the accuracy and consistency of echo detection. This validation process enhances the trustworthiness of the system's performance, as it verifies the reliability of the echo detection mechanism. Integrating a machine learning model, specifically a Multi-Layer Perceptron (MLP), with the sensor system allows for improved performance in echo detection. By training the MLP model on relevant sensor data, the system enhances the precision and efficiency of echo detection. The model learns patterns and features from the data, enabling more accurate identification of objects or humans in the environment. The choice of MLP over other machine learning models is driven by its suitability for the problem at hand. MLPs are known for their capability to handle complex nonlinear relationships in data, making them well-suited for tasks like echo detection in autonomous driving scenarios. Through the proposed approach, the system optimizes performance by improving the accuracy and efficiency of echo detection. By addressing the challenges inherent in autonomous driving scenarios, such as accurate object differentiation and reliable detection mechanisms, the system enhances overall safety and effectiveness in urban environments.

VIII. Outlook

Increasing the diversity and volume of training data, as well as employing data augmentation techniques, can help improve the generalization capabilities of the machine learning model. This includes collecting data from various urban environments and scenarios to capture a wider range of possible conditions. Integrating data from multiple sensor modalities, such as radar, LiDAR, and cameras, through sensor fusion techniques can provide a more comprehensive understanding of the environment. Combining information from different sensors can improve object detection accuracy and robustness, especially in challenging scenarios. Optimizing algorithms and models for real-time performance is crucial for practical deployment in autonomous driving systems. This involves reducing computational complexity, optimizing code efficiency, and leveraging hardware accelerators (e.g., GPUs or TPUs) to ensure timely processing of sensor data. Implementing continuous validation and testing procedures, including simulation-based testing and real-world validation, ensures that the system remains reliable and effective across various conditions and environments. Regular updates and improvements based on feedback from validation tests can enhance system performance over time. Enhancing the user interface and human-machine interaction aspects of autonomous driving systems can improve user experience and acceptance. Providing clear feedback, intuitive controls, and effective communication of system capabilities and limitations can enhance user trust and confidence in the technology.

Acknowledgment

We would like to express our gratitude to Prof. Dr. Andreas Pech and Prof. Dr. Peter Nauth for giving us the chance to work on this subject under their supervision and for giving us the right direction to finish the required project and this report. The lectures on "Autonomous Intelligence System" were very beneficial in providing the background knowledge and motivation for writing the report for the

seminar. We also want to express our gratitude for Prof. Julian Umansky important contribution in providing all the tools necessary for the task to be completed successfully. If we do not express our gratitude to the writers of the references and other works cited in this work, we will be in breach of our responsibility.

REFERENCES

[1] P. M. Nauth, A. H. Pech, R.Michalik, “Research on a new smart pedestrian detection sensor for vehicles,” Frankfurt University of Applied Sciences, Germany..

[2] A. L. Bowler and M. P. Pound, “A review of ultrasonic sensing amd machine learning methods to monitor industrial processes,” in Ultrasonics, University of Nottingham, UK

[3] E. O. Brigham, R.E. Morrow, “The fast fourier transform,”

[4] A. Rana and A. Rawat, “Application of Multi Layer (Perceptron) Artificial Neural Network in the Diagnosis System: A systematic review ” , IEEE, Uttaranchal University, Dehradun

[5] S. Visa, B. Ramsay, and A. Ralescu, “Confusion matrix based feature selection,” conference paper, January 2011… P. Dollar, C.Wojek, B. Schiele, “Pedestrian Detection: An evaluation of the state of the art,” Frankfurt University of Applied Sciences, Germany.

[6] “Ultrasonic Sensors 101: How They Work, and How to Simulate Them,” OnScale, Aug. 01, 2019. https://onscale.com/ultrasonic-sensors-101-how-they-work-and-how-to-simulate-them/ (accessed Dec. 29, 2023).

[7] N. Latha, B. Murthy, and K. B. Kumar, “Distance Sensing with Ultrasonic Sensor and Arduino,” undefined, 2016, Available: https://www.semanticscholar.org/paper/Distance-Sensing-with-Ultrasonic-Sensor-and-Arduino-Latha Murthy/91382c35d32683d4ec5a68283aba9d465def41d8

[8] Introduction FIUS Sonar Sensor by Prof. Dr. Nauth / Prof. Dr. Pech, Frankfurt University Of Applied Sciences

[9] A. H. Pech, P. M. Nauth and R. Michalik, “A new Approach for Pedestrian Detection in Vehicles by Ultrasonic Signal Analysis,” IEEE EUROCON 2019 -18th International Conference on Smart Technologies, 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8861933.