Individual Project

Supervised By Prof. Dr. Andreas Pech

Human Presence Detection with Ultrasonic Proximity Sensor and Machine Learning

Mashnunul Huq
1384042
mashnunul.huq@stud.fra-uas.de

***Abstract*: This paper represents a machine learning algorithmic approach to solve the problem of human presence detection from ultrasonic sensor data. The Ultrasonic Proximity sensor uses high-frequency sound waves and eventually gives analog to Digital conversion (ADC) data signal for a time frame to detect objects and measure distance. It is a conundrum to detect a human presence from the received signal. Also, Convolutional Neural Network (CNN) was invented for image classification and principally used for person detection from an image. This paper proves that, with the right processing of ADC time signal data, CNN can detect human presence from the ultrasonic proximity sensor data to great extent.**

Keywords—Convolutional Neural Network (CNN), Mobile Network Version 2 (MobiNetV2), Power Spectral Density (PSD), Fast Fourier Transform (FFT), Signal Processing, Red Pitaya, Ultrasonic Sensor.

I. INTRODUCTION

Ultrasonic sensors are a type of non-contact sensors that utilize sound waves to measure distance, presence, or level of objects at some distance. Emitting high-frequency sound waves and receiving the deflected waves from the object allows the sensor to calculate the distance between sensor and the object. Ultrasonic sensors are commonly used in a variety of applications including vehicle parking assistance, object's features measurement, fault detection, organ imaging in medical science, RFID sensing, Fire extinguishing and various fields of robotics. They can detect objects in a wide range of materials, including metal, plastic, and liquids, making them versatile and useful in many different settings. One of the key advantages of ultrasonic sensors is their ability to operate in harsh environments. Ultrasonic sensor follows rudimentary principle of sound propagation and reflection within ultrasonic frequency range (20kHz to several Hundred kilohertz). It can function promisingly in conditions where light intensity is low and dark. To determine the presence of an object or human, one can use imaging tools like cameras or thermal cameras which are highly accurate with AI tools but expensive in terms of power and cost. To detect an object or human presence one can also arrange some ultrasonic sensors, mounted parallel, and then place an object within the detection area of these ultrasonic sensors (Stiawan, Kusumadjati, Aminah, Djamal, & Viridi, 2019). Ultrasonic sensor provides accurate distance measurement of an object. The transmitter of the sensor transmits ultrasonic waves, and when the waves hit an object, part of their energy reflected back to receiver of the sensor as echo signal. The distance to the object can be calculated through the speed of the ultrasonic sound waves (which is approximately 343 meters per second or 1125 feet per second at room temperature) in the media and the angle with a calculation approximately multiplying the time taken by the speed of sound to receive and dividing it by 2 since the wave has to travel to the object and back. Figure 1. demonstrates the working principle of the ultrasonic sensor. Ultrasonic proximity sensors can measure up to 10m target distance approximately. The inclination surface of the object as the waves hit may cause deflected part of it away from the sensor's receiver and then the sensing accuracy decreases. In this project I2C proximity sensor was used which is Inter-Integrated Circuit type proximity sensor using 2 wire communication system (SDA-data line and SCL-clock line). This is ideal for embedded systems and micro controllers.

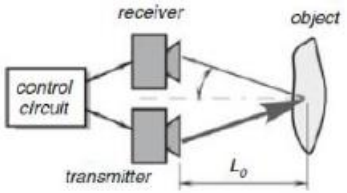

Figure 1: Ultrasonic Sensor's Working Principle

The required microcontroller for signal related project is Redpitaya. It is an open-source platform which integrates a wide range of functionalities related to signal processing into a single compact device. Red Pitaya combines a dual-channel oscilloscope, a function generator, a spectrum analyzer, and various other features on a programmable hardware platform featured with an ARM Cortex-A9 processor. It allows web applications for instant signal monitoring and processing and also gives C and Python programming functions or API for signal processing integration into different programming applications. Red Pitaya platform also includes various peripherals, such as high-speed analog-to-digital converters (ADCs) and digital-to-analog converters (DACs), as well as GPIO, Ethernet, USB, and HDMI interfaces. The combination of two ultrasonic sensors and Red Pitaya have been used to acquire raw data from the environment, which is then pre-processed and tuned in order to fetch them in a CNN model.

Figure 2: Redpitaya board with mounted I2C Ultrasonic Proximity Sensor

For the decision-making unit a computer loaded with TensorFlow library is used as decision server. TensorFlow is an open-source machine learning library developed by the Google Brain team which provides a comprehensive platform for building and deploying machine learning models including neural networks for image and speech recognition, natural language processing and more. TensorFlow supports both CPU and GPU acceleration, enabling efficient training and inference on various hardware architectures and comes up with TensorFlow-lite for small processors like arm64. It is flexible to work with high-level APIs for quick model prototyping or dive into lower-level operations for fine-tuning and customization of data.

CNN is a concept of deep neural networks specifically designed for image processing and classification. TensorFlow's CNN capabilities enable the creation of intricate neural network architectures with convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification along with dropout layers for learning parameters reduction. TensorFlow framework facilitates the training of CNNs on large datasets, allowing models to learn hierarchical representations of visual features for image detection. TensorFlow's flexibility and scalability make it a admired choice for a wide range of computer vision tasks, from image classification to object detection.

MobileNetV2 is a lightweight and efficient CNN architecture designed for small processors and edge devices with resource constraints. Developed by Google, MobileNetV2 focuses on achieving high accuracy with a significantly reduced computational cost and time. It employs Depthwise Separable Convolutions, which separate spatial and channel-wise convolutions, reducing the number of parameters for computation (10 to 30% reduction). MobileNetV2 is well-suited for real-time applications on devices with limited processing power, making it ideal for mobile applications, embedded systems, and the Internet of Things (IoT). Despite its efficiency, MobileNetV2 maintains competitive performance, making it a compelling choice for deploying deep learning models on resource-constrained platforms. (Sandler, Howard, Zhu, Zhmoginov, & Chen, 2018)

II. METHODS

A. *ADC Data Acquisition*

With the help of inbuilt redpitaya library for C programming “rp.h” a UDP client software was prebuilt before starting this project by other teams. This software calls for the I2C functionality for getting data from the FIUAS project built Ultrasonic proximity sensor. The sensor works around 2.5volt threshold voltage and upon receiving ultrasonic sound creates a value according to positive or negative to 2.5V as the sensor has no negative value storing capability with float values. At last the signal in whole is deducted from the positive threshold voltage and gets an Alternative Current like signal shape crossing the zero axis.

Then this data stored in a buffer memory is then processed finding where the object has been detected by analyzing the signal within redpitaya. Then the data is manipulated and the object (where data value is different than a threshold) signal is shifted to be in the middle of the whole data. Then this data was collected using the prebuilt UDP connection program.`

B. *Data Preprocessing*

Before training these data into CNN one needs to convert data into image. Form a time series data image can be meant for two types. Either the ADC signal image which is on time scale or the frequency scale image. From the ADC signal a machine learning algorithm cannot get significant features of a signal. For that reason, most of the signal processing machine learning algorithms works on the frequency domain data. A benefit from the frequency domain data is that a particular object will always emit a range of frequency. For example, the emission from a wall and the emission from a human body will always differ and there will be significant amount of this frequency range in deflected signal also.

To get this frequency domain features a better approach is to work with the Power Spectral Density with a particular colour map designated for the intensity of data. It is a measurement to show how the power of a signal is distributed across different frequencies. In other words, it provides a way to analyze the frequency content of a signal and understand how much power is associated with each frequency component. PSD is used to analyze the frequency components of a signal. It helps identify dominant frequencies and understand the overall frequency distribution. As most of the noise in real life effects the amplitude of the signal, a power spectral density is immune to noise data received by the sensor while tracking the deflected signal. It also helps in distinguishing between signal and noise components in various applications. With frequency distribution color map machine can learn easily the difference between human presence and object presence only.

In course of achieving PSD there are many theories like FFT square or Wiener-Khinchin theorem. In this project Wiener-Khinchin theorem approach is used. The Descrete Fourier Transform is done by the formula of,

$$X_k = \sum_{n=0}^{N-1} x_n . e^{\frac{-2\pi i}{N} kn};$$ X_k = is the complex DFT coefficient at frequency kk,

X_n =is the input sequence, N = is the length of the sequence, i = is the imaginary unit.

This is also the Fast Fourier Transform (FFT), Now Power Spectral Density (PSD) is calculated as the Fourier transform of the auto correlation function R(τ):

$$S(f) = \int_{-\infty}^{\infty} R(\tau) e^{-j2\pi f\tau} d\tau$$

For the discrete signal PSD is calculated with following equation at a frequency f_k,

$\hat{S}(f_k) = \int_{-\infty}^{\infty} \frac{1}{N} \left| \sum_{n=0}^{N-1} x[n] e^{-j2\pi f_k n} \right|^2$; N = is the length of the signal, f_k = is the frequency corresponding to the k-th value of the DFT, j = is the imaginary unit.

Also, the Auto Correlation Function is calculated for a discrete signal using the following equation,

$R[k] = \sum_{n=0}^{N-1} x[n].x[n-k]$; k = lag and N = length of the signal

After calculating the PSD value of the fixed length windowed signal with colour map of 'magma' the picture will have the whole signal frequency range with the intensity (marked as golden) of the dominant frequency data.

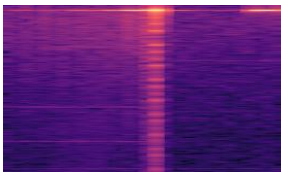

Figure 3: PSD of signal with 512 freq domain data of a signal with presence of Human

As per received signal criteria the first 512 data of the frequency domain is most significant in case of object or human detection. These pictorial representations are converted to numpy arrays and fed as features to the CNN architecture.

C. *Machine Learning*

The PSD pictures are fed to the neural network. Here there are two options. First making a CNN machine using tensor flow from scratch or use already built up and trained known networks like ResNet, LeNet-5, AlexNet, MobileNet etc. For this project two approaches were persued. First built a CNN with a (256, 256, 1) input fed to a Convolutional 2D layer followed by a Max Pooling layer and a Dropout layer till the flattening of the image and after Flatten layer a Dense layer followed by a Dropout layer is used before using Sigmoid activation function to loose unwanted overfitting and resource usage.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 32)      320

 max_pooling2d (MaxPooling2D  (None, 127, 127, 32)     0
 )

 conv2d_1 (Conv2D)           (None, 125, 125, 64)      18496

 max_pooling2d_1 (MaxPooling  (None, 62, 62, 64)       0
 2D)

 dropout (Dropout)           (None, 62, 62, 64)        0

 conv2d_2 (Conv2D)           (None, 60, 60, 128)       73856

 max_pooling2d_2 (MaxPooling  (None, 30, 30, 128)      0
 2D)

 conv2d_3 (Conv2D)           (None, 28, 28, 256)       295168

 max_pooling2d_3 (MaxPooling  (None, 14, 14, 256)      0
 2D)

 flatten (Flatten)           (None, 50176)             0

 dense (Dense)               (None, 64)                3211328

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 1)                 65

=================================================================
Total params: 3,599,233
Trainable params: 3,599,233
Non-trainable params: 0
_________________________________________________________________
```

Figure 4: CNN Architecture used.

In this CNN the picture data is also saved as numpy arrays and fed easily to the input CNN as gray images.

Next Mobilenet version 2 is trained with the picture data converted to (224, 224, 3). As the Depthwise Sperable layer requires colored image, instead of gray images the colored image is fed to the input layer of mobile network. The network is made of 16 blocks of Convolutional 2D layer followed by Batch Normalization and ReLU activation layer and Depthwise Convolution layer.

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3       0         []
                             )]

 Conv1 (Conv2D)              (None, 112, 112, 32       864       ['input_1[0][0]']
                             )

 bn_Conv1 (BatchNormalization)  (None, 112, 112, 32    128       ['Conv1[0][0]']
                             )

 Conv1_relu (ReLU)           (None, 112, 112, 32       0         ['bn_Conv1[0][0]']
                             )

 expanded_conv_depthwise (Depth  (None, 112, 112, 32   288       ['Conv1_relu[0][0]']
 wiseConv2D)                 )

 ..........

 Conv_1 (Conv2D)             (None, 7, 7, 1280)        409600    ['block_16_project_BN[0][0]']

 Conv_1_bn (BatchNormalization)  (None, 7, 7, 1280)    5120      ['Conv_1[0][0]']

 out_relu (ReLU)             (None, 7, 7, 1280)        0         ['Conv_1_bn[0][0]']

 global_average_pooling2d (Glob  (None, 1280)          0         ['out_relu[0][0]']
 alAveragePooling2D)

 dense (Dense)               (None, 2)                 2562      ['global_average_pooling2d[0][0]'
                                                                 ]

=================================================================
Total params: 2,260,546
Trainable params: 2,226,434
Non-trainable params: 34,112
_________________________________________________________________
```

Figure 5: MobileNetV2 Architecture

The difference between these two CNN architectures is because of depthwise flattening of layers mobilenet will train 2,226,434 parameters and CNN will train 3,599,233 parameters. As a result, CNN will use more computational resources and time to train upon data.

D. *Choosing Right Model:*

The two selected Neural Networks were trained with the same amount of data. The CNN though trained on gray scale image gave validation accuracy of 99.89%. whereas the MobileNetV2 trained on colored image gave the validation accuracy of 95.94%. Bothe of the accuracy is above the benchmark of 91% on validation data. But the amount of time consumed by the mobilenet model is 18.5 minutes (1109s approx.) and by the CNN model is 33.6 minutes(2015.5s approx.). Because of higher accuracy CNN model architecture was selected for the decision-making unit.

```
Epoch 1/10
258/258 - 200s - loss: 0.5659 - accuracy: 0.6741 - val_loss: 0.2581 - val_accuracy: 0.8956 - 200s/epoch - 775ms/step
Epoch 2/10
258/258 - 202s - loss: 0.2610 - accuracy: 0.8936 - val_loss: 0.0934 - val_accuracy: 0.9796 - 202s/epoch - 784ms/step
Epoch 3/10
258/258 - 200s - loss: 0.1205 - accuracy: 0.9702 - val_loss: 0.0272 - val_accuracy: 0.9960 - 200s/epoch - 776ms/step
Epoch 4/10
258/258 - 201s - loss: 0.0757 - accuracy: 0.9852 - val_loss: 0.0174 - val_accuracy: 0.9978 - 201s/epoch - 778ms/step
Epoch 5/10
258/258 - 202s - loss: 0.0663 - accuracy: 0.9859 - val_loss: 0.0158 - val_accuracy: 0.9960 - 202s/epoch - 781ms/step
Epoch 6/10
258/258 - 202s - loss: 0.0613 - accuracy: 0.9865 - val_loss: 0.0126 - val_accuracy: 0.9956 - 202s/epoch - 783ms/step
Epoch 7/10
258/258 - 201s - loss: 0.0420 - accuracy: 0.9890 - val_loss: 0.0119 - val_accuracy: 0.9978 - 201s/epoch - 780ms/step
Epoch 8/10
258/258 - 202s - loss: 0.0335 - accuracy: 0.9919 - val_loss: 0.0098 - val_accuracy: 0.9978 - 202s/epoch - 783ms/step
Epoch 9/10
258/258 - 202s - loss: 0.0416 - accuracy: 0.9908 - val_loss: 0.0112 - val_accuracy: 0.9978 - 202s/epoch - 783ms/step
Epoch 10/10
258/258 - 203s - loss: 0.0300 - accuracy: 0.9941 - val_loss: 0.0087 - val_accuracy: 0.9989 - 203s/epoch - 787ms/step
2015.5018231868744
```

Figure 6: Validation Accuracy of CNN Model

```
Epoch 1/10
2475/2475 - 176s - loss: 0.1318 - accuracy: 0.9457 - val_loss: 0.2016 - val_accuracy: 0.9229 - 176s/epoch - 71ms/step
Epoch 2/10
2475/2475 - 103s - loss: 0.0667 - accuracy: 0.9754 - val_loss: 0.0801 - val_accuracy: 0.9700 - 103s/epoch - 41ms/step
Epoch 3/10
2475/2475 - 103s - loss: 0.0468 - accuracy: 0.9818 - val_loss: 0.0450 - val_accuracy: 0.9865 - 103s/epoch - 42ms/step
Epoch 4/10
2475/2475 - 103s - loss: 0.0362 - accuracy: 0.9869 - val_loss: 0.2633 - val_accuracy: 0.9041 - 103s/epoch - 42ms/step
Epoch 5/10
2475/2475 - 103s - loss: 0.0278 - accuracy: 0.9907 - val_loss: 0.4168 - val_accuracy: 0.8899 - 103s/epoch - 42ms/step
Epoch 6/10
2475/2475 - 104s - loss: 0.0223 - accuracy: 0.9926 - val_loss: 0.0247 - val_accuracy: 0.9894 - 104s/epoch - 42ms/step
Epoch 7/10
2475/2475 - 104s - loss: 0.0162 - accuracy: 0.9940 - val_loss: 0.0344 - val_accuracy: 0.9894 - 104s/epoch - 42ms/step
Epoch 8/10
2475/2475 - 104s - loss: 0.0148 - accuracy: 0.9947 - val_loss: 0.0787 - val_accuracy: 0.9717 - 104s/epoch - 42ms/step
Epoch 9/10
2475/2475 - 104s - loss: 0.0110 - accuracy: 0.9962 - val_loss: 0.0320 - val_accuracy: 0.9888 - 104s/epoch - 42ms/step
Epoch 10/10
2475/2475 - 104s - loss: 0.0098 - accuracy: 0.9968 - val_loss: 0.1602 - val_accuracy: 0.9594 - 104s/epoch - 42ms/step
1108.4135429859161
```

Figure 7: Validation Accuracy of MobileNetV2 model

Both machines are saved and stored for future research purposes.

E. Sending Decision Directly to Redpitaya

Now the operating system of Redpitaya is built on linux kernal Debian Jessi. In that case the infamous ZeroMQ library for User Datagram Protocol (UDP) connection set up was not possible as the library installation is out dated for this linux distribution. The previous UDP connection socket format is used for communicating between the Redpitaya and decision unit PC. In the Redpitaya C programming ARPANET (Advanced Research Projects Agency Network) library is used for UDP socket build up. In the decision unit Python programming is used and that requires no additional library installation. Rather socket and struct libraries belonging to the OS framework of Python is used for the UDP connection making. The decision making program after successful connection build up with the Redpitaya system asks for data by sending message format "-a 1". Here this a is defined inside the redpitaya iic.c program to start the sensor the send ADC data to the decision making PC. Then the decision making program process the ADC signal data and creates a PSD image and predict the image. After prediction it sends "-b 1" message to the Redpitaya for turning on the light and "-c 1" message to turn off the light. If the previous decision was turn on the light as human presence detected then the decision making system will wait 1 minute (60s) to fetch another set of current ADC data to make next decision as logically a person can away from the place for for moments and the light should not be turned off. But if the previous decision is turn off the light then the decision making unit will fetch the next set of data within 20 seconds and send the decision accordingly. The decision making and corresponded message sending requires 153 mili seconds which is less than the working benchmark of 1 second. In case of turning on the light LED light number 5 and 6 present on the Redpitaya Board is used. First 1- 4 LEDs are used by the Redpitaya C programming for connection, messaging, object detection and signal manipulation indications and 7 – 9 LEDs are used by Redpitaya os, limited LEDs (5 and 6) were unoccupied and used for this project's successful detection indication.
Previously coded Redpitaya c programming algorithms were changed in switch case to add the message "b" and "c" under the udp_server.c program's function named "parsed_udp_message". The functions later to work on how to indicate the turn on and turn off lights are defined in the iic.c program's function named as iic_led_turn_on and iic_led_turn_off which current holds command for turning on LED 5 and 6 and turning off these LEDs respectedly.

Figure 8: LED 5-6 turned on as decision made as Human Presence detected.

Figure 9: LED 5-6 turned off as decision made as No Human Presence detected.

III. Result

Before choosing the Neural network First the validation accuracy was tested on both CNN and MobileNetV2 model. The CNN with 3,599,233 parameters and MobileNetV2 with 2,226,434 trained parameters gave the accuracy of 99.89 % and 95.94% accuracy on the validation set. According to this result CNN model was selected but when different set of mixed data as testing set fed to the CNN model the accuracy reduced to 86.26%. The confusion matrix of the testing data is following:

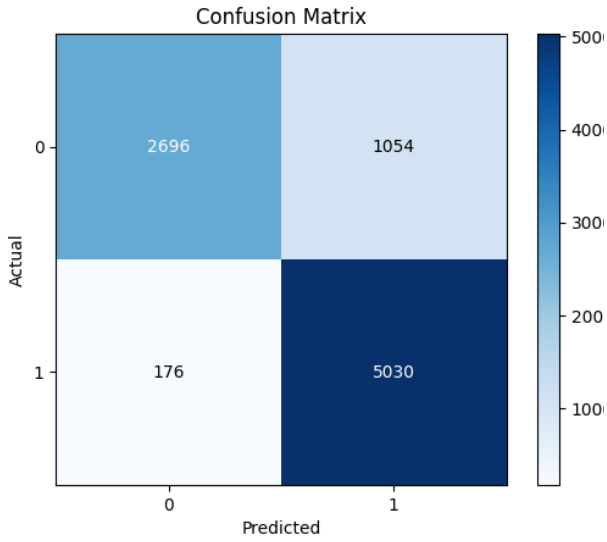

Figure 10: Confusion matrix of CNN model prediction on Testing Dataset.

The confusion matrix shows 0 as no human presence and 1 as human presence detected. False no Human Presence detection is only 176 and false Human Presence detection is 1054 which is a huge number of false presence detection. As a result the lights will be turned on because of false presence detected will be greater. This has happened because while training the machine 5 human presence possibility situation was considered. Person sitting in the middle, Person sitting in the right edge of sensor range, Person sitting in the left edge of sensor range, Person sitting back of sensor edge, Person standing in the range of proximity sensor. For this reason 6000 data as person presence were fed to the model and only 5000 data were fed as no person present in the proximity sensor range were fed. So, the person presence detection is relatively accurate than no person presence detection. In terms of mathematical analysis:

$$\text{Precision} = \frac{True\ Positive}{True\ Positive+False\ Positive} = \frac{5030}{5030+176} = 96.62\%$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive+False\ Negative} = \frac{5030}{5030+1054} = 82.67\%$$

$$\text{Specificity} = \frac{True\ Negative}{True\ Negative+False\ Positive} = \frac{2696}{2696+176} = 93.87\%$$

$$\text{F1 Score} = 2 \times \frac{Precision \times Recall}{Precision+Recall} = \frac{2\times(0.9662\times0.8267)}{0.9662+0.8267} = 0.891$$

It is evident that Precision and Specificity are in acceptable range, but Recall is below 85% benchmark. Then the system was tested in the real life scenario for 35 decision data.

Here is the accuracy result table for Practical Human Presence detection:

Actual State	No. Wrong Decision	No. Accurate Decision
Person Present	1	9
No Person Present	3	7
Person Standing	0	5
Only Object Present	6	4

From the table it is evitable that, the system shows much error in case of empty place or with an object. Because of bad recall value, most of the time the system detects the PSD as a presence of person.

IV. Discussion

From the result further research on the PSD value and the system must be carried out for better accuracy in case of non-human presence. A new machine learning approach with more data on empty place or with colored approach has to be considered as a path of higher accuracy.

Another place of improvement can be ADC signal generation. The amount of noise can be a problem with the ADC signal PSD value. One mitigation technique can be using the Kalman Filter on the generated ADC signal. This will reduce spikes and static noises in the signal making it more robust for calculation.

One problem with the UDP connection remains with the python program is that the first message with which the I2C sensor should be activated is hidden from the code present in the given Redpitaya C programming. For this reason, every starting of a new connection of the decision making server with the Redpitaya board, the old UDP client software has to be opened and manually restart of sensor has to be commanded. This area needs to be improved in future in a way which requires no manual restarting of the sensor using old UDP connection.

V. References

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). MobileNetV2: Inverted Residuals and Linear Bottlenecks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), DOI: 10.1109/CVPR.2018.00744*.

Stiawan, R., Kusumadjati, A., Aminah, N., Djamal, M., & Viridi, S. (2019). An Ultrasonic Sensor System for Vehicle Detection Application. *Journal of Physics: Conference Series, 1204(), 012017–. doi:10.1088/1742-6596/1204/1/012017*.