Automotive Sensor for Object Recognition using RedPitaya and Raspberry Pi

M.Eng IT
Automation Lab and Autonomous
Intelligent System
Sardar Eyaan Ahmed : 1273852
sadarahm@stud.fra-uas.de
Aamir Muhammad : 1272918
amuhamma@stud.fra-uas.de

Abstract— **The documentation enlightens a brief overview over Object Differentiation using Ultra sonic sonar sensor. The concept of machine learning was the key behind implementing project. Extracting the key features for the object and implementing the chosen machine learning algorithm on the data. The chosen supervised machine learning algorithms for this project are SVM and Bayes Classifiers. The implementation was done using both algorithms (SVM and Bayes) and a detailed comparison is discussed between the two classifiers on the achieved results. Furthermore, two major devices Raspberry Pi and Red-Pitaya plays a vital role in implementation of Object Differentiation project. The final output of the project is displayed using Red Pitaya and Raspberry Pi. Python was used as the programming language as the use of Raspberry Pi was mandatory. The final output can differentiate between three different objects mainly Wall, Human and Car, no other object can be detected by the sensor as the algorithm only defined for these three objects. The final output is displayed using Red-pitaya with the help of LED's mounted on it, moreover the Object Differentiation/Detection was made plug and play meaning as we boot up the Raspberry Pi and Red-pitaya the object differentiation will going to start automatically.**

Keywords— Machine Learning, Supervised Learning, Feature Extraction, State Vector Machine (SVM), Bayes Classifier, Raspberry Pi, Red-pitaya, Ultra Sonic sensor, Object Detection, Wall, Human, Car.

I. INTRODUCTION

The aim of our project in Artificial Intelligence System module is to use Ultrasonic Sensor for object recognition using *Red Pitaya* and *Raspberry Pi*.

Our project work can be split into distinct section which we will discuss extensively in the following parts of this documents. To introduce briefly regarding our project work the two systems *Red Pitaya* and *Raspberry Pi* is connected either through Wi-Fi or ethernet cable and with the help of *SSH/SCPI* commands *[1]* both the system are configured to communicate with each other which we will talk about in detail in section *(SCPI Commands)* of our document. Reflected data signals from the objects like Car, Human and Wall is analyzed and processed with the help of *SSH* and Python code to generate a plot of the signals which we will discuss in Sections (*Graph Pattern for Wall, Graph Pattern for Human, Graph Pattern for Car*).

After analyzing the reflected signals from the objects, the data had been stored in the form of *XLSX* files making use of Python code. A data set of around 150 thousand has been collected for each of the object for training, testing and classification.

Before object recognition different features namely *Energy sum, RMS, Standard Deviation, Abs* etc. *[2]* has been extracted from the data sets which is explained in section *(Features Extraction).*

After Feature Extraction two Supervised neural networks techniques namely *SVM (Support Vector Machine)* and *Naïve Bayes* method is used for the object recognition and classification which is described in section *(Classification).* Also, a comparison between the two neural network is mentioned in section *(Comparison Between SVM and Bayes Classifier).*

To make the whole project plug and play programming in python and shell scripting is done that allows robust and reliable starting, capturing and saving signals from the attached ultrasonic sensor of *Red Pitaya* automatically after powering it. The result of the project is shown in section *(Results).*

Every object is recognized by processing the reflected signal from the object and lighting corresponding *LED* for that object. The General Architecture for the project is displayed below.

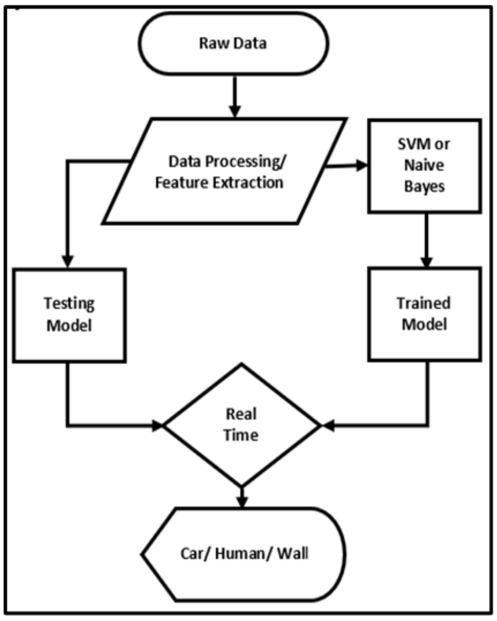


Figure 1 : General Architecture of Model

II. THEORATICAL BACKGROUND

In this section we will give a brief overview about theoretical background of the project. For instance, we will present the specification of the hardware operated, programming language applied, and the underlying network algorithm incorporated.

A. *Red Pitaya*

The Red Pitaya is a microchip *STEM Lab* board which has a wide bandwidth of the *ADC* and *DAC*, that is the reason Red Pitaya is used extensively as radio receiver and transmitter and in other radio frequency applications. Enough functionalities are available for signal processing in this microchip board. We make use of an automotive ultrasonic sensor which is mounted on the Red Pitaya. The board is used as the platform to process signals from the sensor as well as classify the signal if trained through network algorithms. The board is illustrated in Fig 2 below:

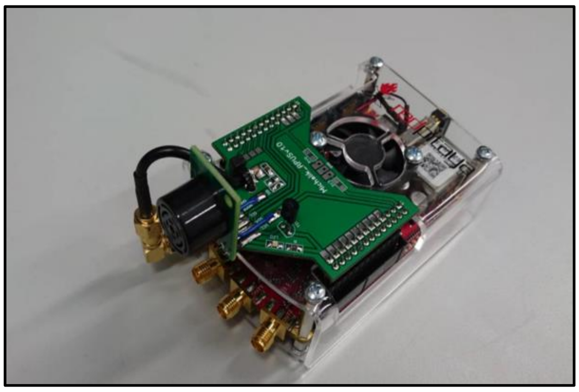


Figure 2 : Red Pitaya board with attached Ultrasonic sensor

1) *Specification*

The Red Pitaya has enough capability to handle and process ultrasonic signal through the Ultrasonic sensor attached above it. Some important information could be listed as following:

- Processor: processor dual core arm cortex (a) dual core arm cortex a9.
- RF inputs/outputs sampling rate: *125 MS/s.*
- RF inputs/ outputs *ADC* resolution: *10* bits.
- Bandwidth: *40MHz.*
- Frequency Range: *0-50MHz.*

In addition, the information below is the characteristic of the ultrasonic sensor. Further information about the specification is described thoroughly in *[10]*.

- Nominal Frequency: *40kHz.*
- Detectable range: *0.2 – 4 m.*
- Sensitivity: *-63+/- db.*

B. *SRF02*

The *SRF02* is a single transducer ultrasonic rangefinder mounted on a small *PCB*. It features both *I2C* and a Serial interface. The serial interface is a standard *TTL* level *UART* format at 9600 baud,1 start, 2 stop and no parity bits, and may be connected directly to the serial ports on any microcontroller.

New commands in the *SRF02* include the ability to send an ultrasonic burst on its own without a reception cycle, and the ability to perform a reception cycle without the preceding burst. This has been as requested feature on our sonar's and the *SRF02* is the first to see its implementation. Because the *SRF02* uses a single transducer for both transmission and reception, the minimum range is higher than our other dual transducer rangers. The minimum measurement range varies from around 17-18cm (7 inches) on a warm day down to around 15-16cm (6 inches) on a cool day. Like all our rangefinders, the *SRF02* can measure in µSec, cm or inches. Details regarding *SRF02* are available in *[3]*.

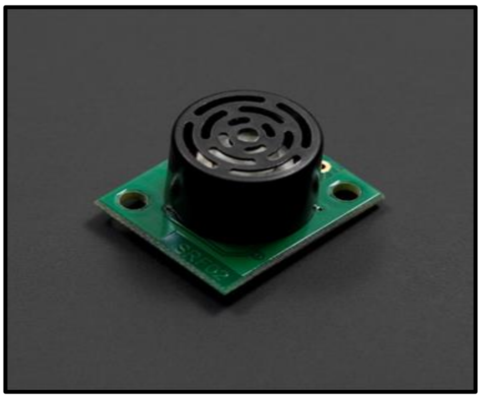


Figure 3 : Ultrasonic Sensor SRF02

1) *Specification*

Some of the features for the Ultrasonic Sensor *SRF02* is given below.

- Range: *16cm* to *6m.*
- Power: *5v*, *4mA* Typ.
- Frequency: *40KHz.*
- Size: *24mm x 20mm x 17mm* height.
- Analogue Gain: Automatic *64* step gain control.
- Connection Modes: 1 - Standard *I2C* Bus 2 - Serial Bus (connects to *16* devices to any µP or *UART* serial port).
- Fully Automatic Tuning: No calibration, just power up and go.
- Timing: Fully timed echo, freeing host controller of task.
- Units: Range reported in µSec, mm or inches.
- Light Weight: *4.6gm.*

C. SCPI Commands

STEM lab board can be controlled remotely over *LAN* or wireless interface using *MATLAB*, *LabVIEW*, *Scilab* or Python via Red Pitaya *SCPI (Standard Commands for Programmable Instrumentation)* list of commands. *SCPI* interface/environment is commonly used to control *T&M* instruments for development, research or test automation purposes. *SCPI* uses a set of *SCPI* commands that are recognized by the instruments to enable specific actions to be taken (e.g.: acquiring data from fast analog inputs, generating signals and controlling another periphery of the Red Pitaya *STEM lab* platform).

The *SCPI* commands are extremely useful when complex signal analysis is required where *SW* environment such as Python provides powerful data analysis code and *SCPI* commands provides simple access to raw data acquired on *STEM lab* board. Details regarding *SCPI* commands are available in *[4]*. SCPI commands can be used with different *IDEs* or software namely *MATLAB*, *Python*, *LabVIEW* and *SciLab*.

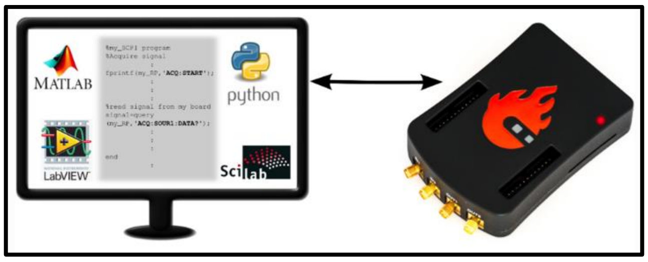

```
%my_SCPI program
%Acquire signal
        :
fprintf(my_RP,'ACQ:START');
        :
        :
        :
%read signal from my board
signal=query
(my_RP,'ACQ:SOUR1:DATA?');
        :
        :
        :
end
        :
```

Figure 4 : SCPI commands compatibility

D. Machine Learning

Machine learning is a branch of Artificial Intelligence *(AI)* that enables computer to learn and improve automatically after it has been trained on an initial set of data. So basically, through experience with past data computers are making decision regarding the data received in future that means no need for Humans to program explicitly each time.

The main intention of machine learning is to permit computers to learn automatically without the Human support and modify actions accordingly. In the past step by step algorithms were designed for computers to solve the specific problems at hand telling the computer exactly how to solve this problem. So, on computer's part no learning was needed. Humans are the only one to analyze and learn. This approach is only possible for small and simple tasks for more advanced and complicated tasks for us Humans it will be very difficult to manually create the needed algorithms. So rather than Human programmers it will be more practical to let computers create and modify the algorithms itself each time it gets new information.

1) Types of Machine Learning

We have many different types of machine learning approaches. We will not go into much details but broadly we can classify it into three Supervised, Unsupervised and reinforcement machine learning. We will not discuss here about reinforcement machine learning. We will give a brief overview of about the remaining two approaches of machine learning. Then in each of the two categories we have different kind of algorithms which is shown in the diagram below. Our main task is to implement *Support Vector Machine* and *Naïve Bayes* machine learning algorithm.

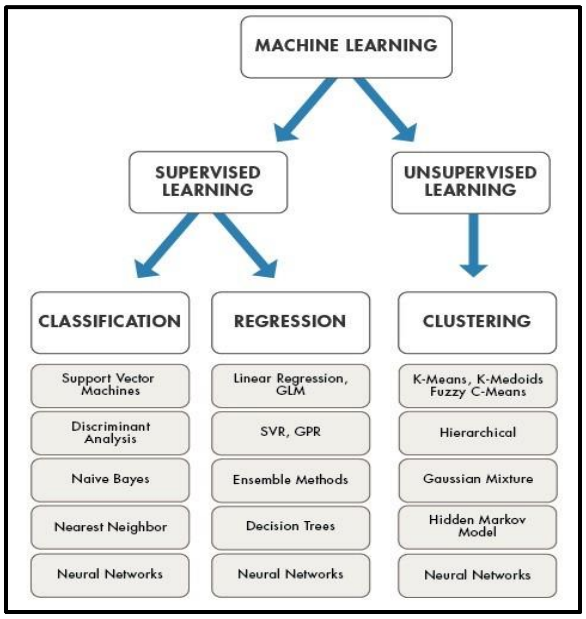


Figure 5 : Types of Machine Learning Algorithm

a) Unsupervised learning

In this type the computer is trained with unlabeled data. No labels are given to the learning algorithm, leaving it on its own to find structure in its input. So here basically we don't need an expert to help learn the computer instead in this kind of learning working with data most of the times unsupervised learning algorithm let the experts know about the different kind of patterns present in data. These algorithms use different kind of methods and procedures on the input data to discover patterns, mine for rules, and group and summarize the data points which help in deriving meaningful insights and describe the data better to the users.

b) Supervised learning

In supervised machine learning algorithm, a Human expert teaches the computer with the training data if we get this kind of input data the answer must be this output. And after a set of trained data the computer must be able to predict for the future patterns. So supervised learning is a kind of spoon feeding and the objective is to learn a general rule that maps inputs into outputs. Supervised learning algorithms are further divided into Classification and Regression. In our project we will make use of two classification algorithms i.e.

Support Vector Machine *(SVM)* and *Naïve Bayes* algorithm. In this kind of algorithms labelled data is used.

E. Support Vector Machine Learning Algorithm

Support vector machine *(SVM)* is a machine learning algorithm which is selected by many professionals as it delivers significant accuracy with less computational power. Support Vector Machine can be used for both regression and classification tasks but, it is extensively used for classification purposes. Even in our project we will use *SVM* for classifying and distinguishing between Wall, Human and Car. And after testing the *SVM* algorithm on our data set we found out that it gives an accuracy of 1.

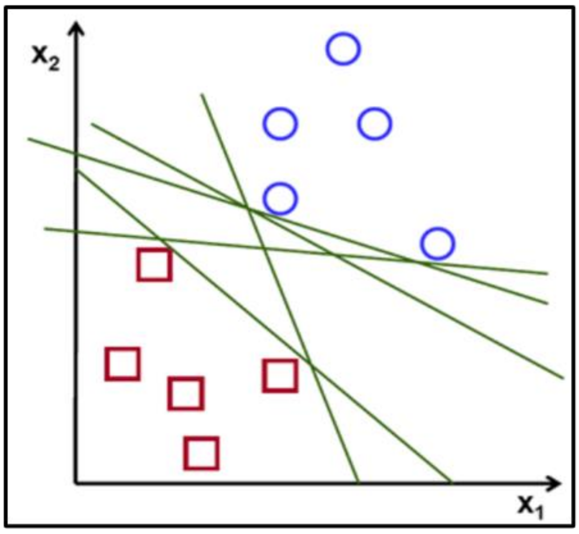


Figure 6 : Different Types of Hyperplanes

Support vector machine algorithm try to find the hyperplane in an n-dimensional space that distinctly classifies the data points *[5].*

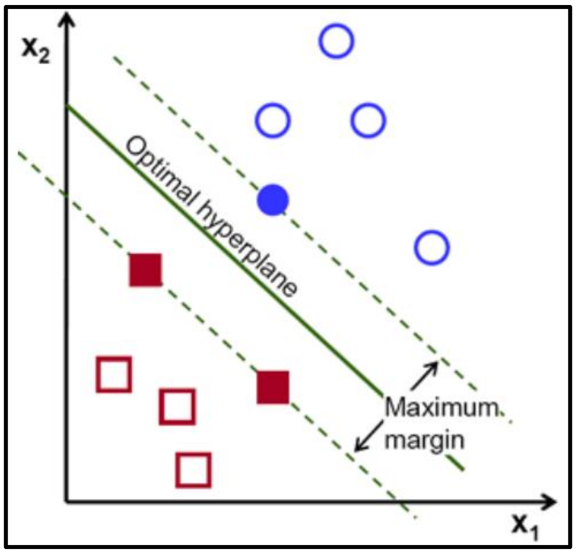


Figure 7 : Optimal Hyperplane

There are many possible hyperplanes that could be chosen. Our task is to look for a plane that has the maximum margin, i.e. the maximum distance between data points of both classes. Maximizing the margin distance provides some support so that future data points can be classified with more certainty.

kernel parameters are used to select the type of hyperplane that will separate the data. Using 'linear' will use a linear hyperplane (a line in the case of *2D* data). 'rbf' and 'poly' uses a nonlinear hyper-plane. In our project we will use *linear kernel*. For more complicated data sets we have to look for other kernels instead of linear in order to work better.

A *linear kernel* can be used as normal dot product any two given observations. The product between two vectors is the sum of the multiplication of each pair of input values [6].

$$K(x, xi) = sum(x * xi) \qquad (1)$$

F. Naive Bayes Machine Learning Algorithm

It is a classification technique based on *Bayes' Theorem*. In simple words a Naive Bayes Algorithm assumes that the presence of a particular feature in a class is not related to any other feature of the class *[7].*

Naive Bayes algorithm is easy to assemble and exceptionally useful for very large data sets. *Naïve Bayes* classification perform well in highly complex classification even though it is so simple.

This lets us examine the probability of an event based on the prior knowledge of any event that related to the former event. Mathematical equation is given below *[8].*

$$P(A/B) = \frac{P(B/A)P(A)}{P(B)} \qquad (2)$$

Above equation gives the basic representation of the *Bayes' theorem.* Here *A* and *B* are two events and

- *P(A|B)*: the conditional probability that event *A* occurs, given that *B* has occurred. This is also known as the posterior probability.
- *P(A)* and *P(B)*: probability of *A* and *B* without regard of each other.
- *P(B|A)*: the conditional probability that event *B* occurs, given that *A* has occurred.

Take a simple machine learning problem, where we need to learn our model from a given set of attributes (in training examples) and then form a hypothesis or a relation to a response variable. Then we use this relation to predict a response, given attributes of a new instance. *Using the Bayes' theorem, it's possible to build a classifier that predicts the probability of the response variable belonging to some class, given a new set of attributes.*

Consider the previous equation again. Now, assume that *A* is the response variable and *B* is the input attribute. So according to the equation, we have

- *P(A|B):* conditional probability of response variable belonging to a particular value, given the input attributes. This is also known as the posterior probability.
- *P(A):* The prior probability of the response variable.
- *P(B):* The probability of training data or the evidence.
- *P(B|A):* This is known as the likelihood of the training data.

Therefore, the above equation can be rewritten as

$$posterior = \frac{prior \times likehood}{evidence} \qquad (3)$$

This is clearly unrealistic in most practical learning domains. For example, if there are 30 Boolean attributes, then we will need to estimate more than 3 billion parameters.

The complexity of the above Bayesian classifier needs to be reduced, for it to be practical. The *Naive Bayes* algorithm does that by making an assumption of conditional independence over the training dataset. This drastically reduces the complexity of above-mentioned problem.

The assumption of conditional independence states that, given random variables X, Y and Z, we say X is conditionally independent of Y given Z, if and only if the probability distribution governing X is independent of the value of Y given Z.

In other words, *X* and *Y* are conditionally independent given *Z* if and only if, given knowledge that *Z* occurs, knowledge of whether *X* occurs provides no information on the likelihood of *Y* occurring, and knowledge of whether *Y* occurs provides no information on the likelihood of *X* occurring.

This assumption makes the Bayes algorithm, naive. Given, n different attribute values, the likelihood now can be written as

$$P(X_1 \dots X_n/Y) = \prod_{i=1}^{n} P\,(X_i/Y) \qquad (4)$$

Here, *X* represents the attributes or features, and *Y* is the response variable. Now, *P(X|Y)* becomes equal to the products of, probability distribution of each attribute *X* given *Y.*

1) Maximizing a Posteriori

What we are interested in, is finding the posterior probability or *P(Y|X).* Now, for multiple values of *Y,* we will need to calculate this expression for each of them.

Given a new instance *Xnew,* we need to calculate the probability that *Y* will take on any given value, given the observed attribute values of *Xnew* and given the distributions *P(Y)* and *P(X|Y)* estimated from the training data.

So, how will we predict the class of the response variable, based on the different values we attain for *P(Y|X).* We simply take the most probable or maximum of these values. Therefore, this procedure is also known as maximizing posteriori.

2) Maximizing Likelihood

If we assume that the response variable is uniformly distributed, that it is equally likely to get any response, then we can further simplify the algorithm. With this assumption the priori or *P(Y)* becomes a constant value, which is 1/categories of the response.

As, the priori and evidence are now independent of the response variable, these can be removed from the equation. Therefore, the maximizing the posteriori is reduced to maximizing the likelihood problem.

3) Feature Distribution

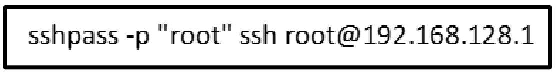
As seen above, we need to estimate the distribution of the response variable from training set or assume uniform distribution. Similarly, to estimate the parameters for a feature's distribution, one must assume a distribution or generate nonparametric models for the features from the training set. Such assumptions are known as event models. The variations in these assumptions generates different algorithms for different purposes. For continuous distributions, the *Gaussian naive Bayes* is the algorithm of choice. For discrete features, multinomial and Bernoulli distributions as popular. In our project we are using Gaussian naïve Bayes classification technique which we will see in detail with python code.

III. METHODOLOGY

A detailed step by step procedure for the object differentiation using ultra sonic sensor, *Raspberry pi* and *Red Pitaya* is discussed in this section. An operating system was installed on *Raspberry Pi* first to have a working platform on it. Also installing necessary python libraries like *NumPy, SCPI, Pyplot, tsfresh* etc.

A. Enabling SSH Connection between Red Pitaya and Raspberry Pi

The first part was to enable a Secure Shell (*SSH*) connection between Red Pitaya and Raspberry Pi. *SSH* is a cryptographic network protocol for operating network services securely over an unsecured network *[9].* Any network can be secured with *SSH* connection. A shell script was written in python language to enable the *SSH* connection between *Red Pitaya* and *Raspberry Pi.* Following script was written for enabling *SSH* connection between the two devices.



```
sshpass -p "root" ssh root@192.168.128.1
```

Where "192.168.128.1" is the static IP address of *Red Pitaya.* After running the above-mentioned script on terminal after that password will be required to remotely access the command window of *Red Pitaya*. The password is *"root".* After the successful *SSH* connection the *Red Pitaya* triggers the ultra-sonic sensor mounted on *Red Pitaya*. Which will start giving random values on terminal. The *SSH* connection script runs on its on whenever the *Raspberry Pi* boot up and connected to *Red Pitaya* this will be discuss in *(Plug and Play)* section of this document.

B. *SCPI script and SCPI Commands for data acquisition*

SCPI command as described stands for “Standard Commands for Programmable Instruments”. *SCPI* commands was used to send commands from *Raspberry Pi* to control *Red Pitaya* after enabling a connection between the two devices. A *Red Pitaya SCPI* python file was used as a library for *SCPI* access to Red Pitaya over an *IP* network. Following *SCPI* commands were sent from *Raspberry Pi* to *Red Pitaya* for data acquisition:

- *ACQ*: *DEC* 64: Setting Decimation to ‘64’ for approximately 16000 samples
- *ACQ*: *TRIG EXT_PE*: Setting Trigger to *EXT_PE* to receive a proper signal using *Red Pitaya*
- *ACQ*: *TRIG*: *DLY* 8192: Setting Trigger delay in samples (was set to 8192 using hit and try operation, as delay was best suit at 8192 samples)
- *ACQ*: *START*: Start the data acquisition
- *ACQ*: *TRIG*: *STAT*? Status of Trigger (For a perfect signal status should be equal to ‘*TD*’)
- *ACQ*: *SOUR1*: *DATA*? Start the data acquisition from source 1 of *Red Pitaya* as ultra-sonic sensor is at source 1

Using these *SCPI* commands a collection of data was possible and using that data further for Feature Extraction and classification. Data acquisition was done for three objects “Wall”, “Human” and “Car”. The *Red Pitaya* has an internal link with the ultra-sonic sensor which triggers on as *SCPI* commands are send to *SCPI* server on *Red Pitaya*. To enable *SCPI* Server following script was used using the *IP* address of *Red Pitaya*

```
rp = scpi.scpi('192.168.128.1')
```

Every time when the code shown in (*Index*) is executed using *SCPI* commands to trigger the *SCPI* server and sensor the *Red Pitaya* responses with a sample of almost *16384*. All those samples are stored in a excel file and plotted as well, will discuss about this in the next section.

C. *Saving and Plotting the real time data from Red Pitaya*

As soon as data acquisition starts on *Raspberry Pi* terminal from *Red Pitaya's* ultra-sonic sensor. The next phase is data collection to collect the data for Wall, Human and Car. After that saving the real time data for each object in an excel (.csv) file. Equally plotting the real time data coming from ultra-sonic sensor attached with *Red Pitaya*. The python code/script for both plotting and saving data is shown in *(Index)* section.

The data collection was done for all three objects Wall, Human and Car one by one and for each object an excel file was created with almost for each object *16000* samples were taken almost *150* times. Which means collected almost of *2400000* samples for each object. The graph for each object at any sample instant is shown below:

1) *Graph Pattern for Wall*

Graph plot for Wall is shown below. *Red Pitaya* was placed approximately 1 meter away from the Wall. Using the data acquisition script, the ultra-sonic sensor is triggered through *Red Pitaya* and then it starts to through the waves to the object, Wall in this case and then receive the reflected waves back from the object. The data was taken with minimum interruption of noise signal. The data was plotted and saved in an excel file as discussed above:

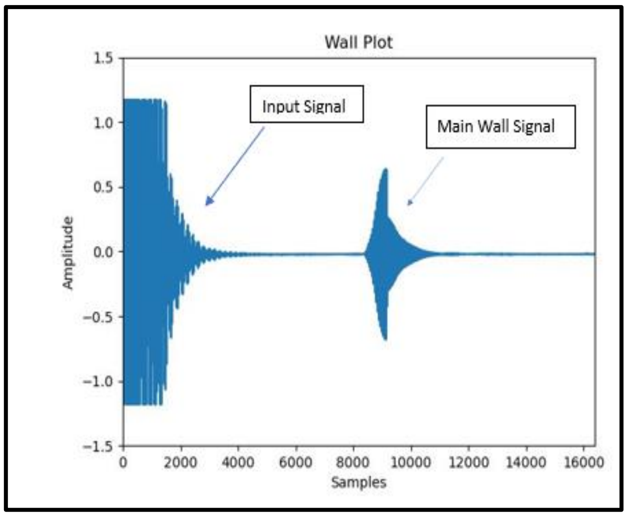


Figure 8 : Graph Pattern for Wall

2) *Graph Pattern for Human*

At a similar distance of 1 meter away from Human, *Red Pitaya* was placed and saved the data in an excel file for Human and following plot was achieved:

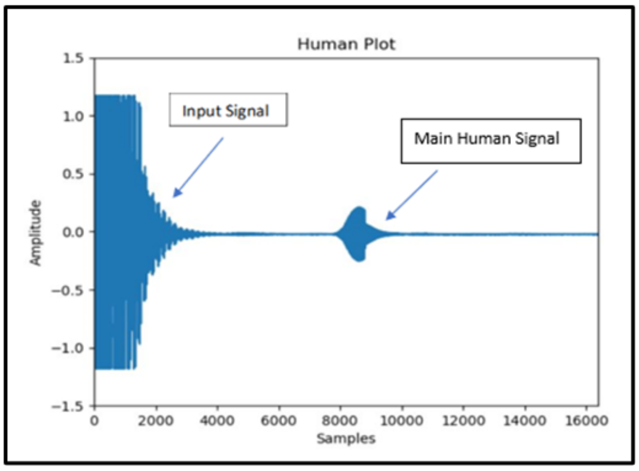


Figure 9 : Graph Pattern for Human

3) *Graph Pattern for Car*

Similarly, the graph reading for Car was also taken by placing *Red Pitaya* at 1 meter away from the Car. As the data was taken outside so there was other surrounding effect of noise as well like "air". Tried to collect data with least effect of noise. The graph for Car front is shown below

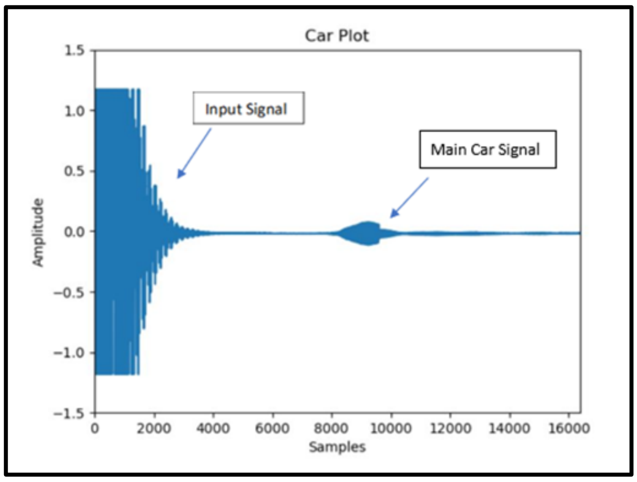


Figure 10 : Graph pattern for Car

D. Features Extraction

After the data collection and saving each object data into an excel file. Next part is to extract key features for each object using the acquired data. Feature Extraction is necessary part before applying machine learning algorithms. Feature Extraction is done to extract the main features from the data which gives the most important and most significant information. "*tsfresh*" library specially build for python Feature Extraction was imported feature extraction. As it already had all the scripts build by default for different features, which makes the work easier as no work required to write the complete code for each feature. Just importing tsfresh library and calling the script for desired feature extraction. Following features were extracted for each object:

1) Absolute Energy of the signal

Absolute Energy of the signal is the sum over the squared values of time signal. The mathematical equation for Absolute Energy over "x" can be given as:

$$E = \sum_{i=0,1\ldots,n} x_i^2 \qquad (5)$$

The parameter "x" is the time series signal to calculate the feature of. This will return the value of this feature i.e. *Absolute Energy* of the signal. The return type for this feature will be "*float*". The *Feature Extraction* script used to calculate this feature is:

```
tsfresh.feature_extraction.feature_calculators.abs
_energy(x)
```

The script was just used in our *Feature Extraction code* to extract the *Absolute Energy* of the real time signal. The basic code run behind the execution if this script is

```
@set_property("fctype", "simple")
def abs_energy(x):
if not isinstance(x, (np.ndarray, pd.Series)):
x = np.asarray(x)
return np.dot(x, x)
```

The graphical plot of *Absolute Energy* for each object Wall, Car and Human is shown below:

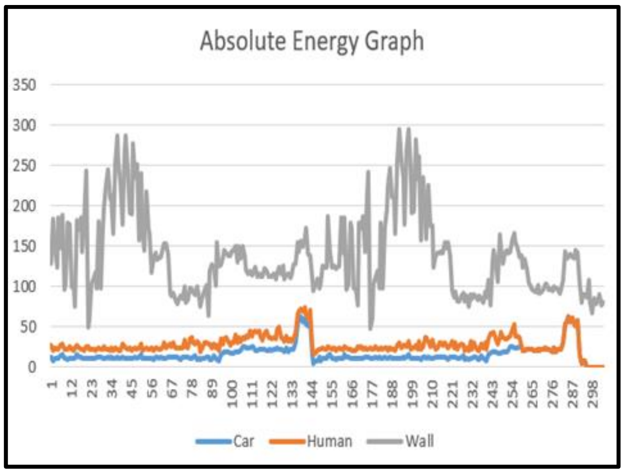


Figure 11 : Absolute Energy Graph for (Car, Human and Wall)

2) Absolute Sum

Absolute Sum calculates the sum of change in time signal over "x". The mathematical relation can be given as:

$$S = \lim_{n\to\infty} \sum_{i=1}^{n} x_i \qquad (6)$$

The parameter "x" is time series signal for which the feature is being calculated. The return value type for this feature will be a "float" type. The used script for *Absolute Sum* of consecutive change is:

```
tsfresh.feature_extraction.feature_calculators.sum
_values(x)
```

Every time code executes this script is used is:

```
@set_property("fctype", "simple")
@set_property("minimal", True)
def sum_values(x):
if len(x) == 0:
return 0
return np.sum(x)
```

The graphical plot of *Absolute Sum* for each object Wall, Car and Human is shown below

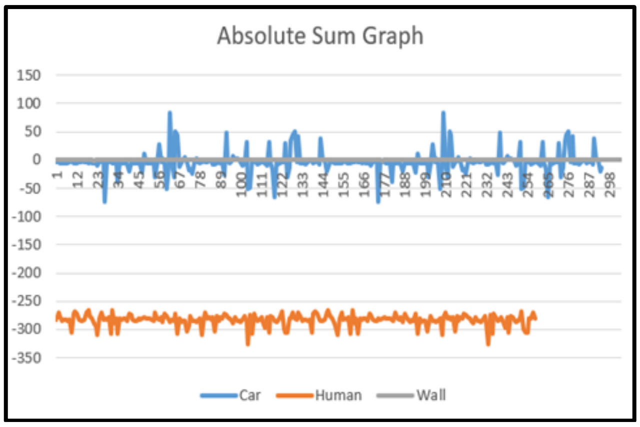


Figure 12 : Absolute Sum Graph for (Car, Wall and Human)

3) *Standard Deviation of the signal*

The *Standard Deviation* is a measure of the amount of variation or dispersion of a set of values. A low *Standard Deviation* indicates that the values tend to be close to the mean (also called the expected value) of the set, while a high *Standard Deviation* indicates that the values are spread out over a wider range *[11].*

The third extracted feature is the *Standard Deviation* of the time signal. The mathematical expression for *Standard Deviation* is given by:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}} \qquad (7)$$

Where "σ" is the *Standard Deviation*. "x_i" is the time series signal. "$\bar{x}$" denotes the total mean of the time signal. And "n" denotes the total number of samples. The following python script was used calculating for *Standard Deviation* using *tsfresh* library:

```
tsfresh. feature_extraction.feature_calculators.standard
_deviation(x)
```

The code executed every time this script is used:

```
@set_property("fctype", "simple")
@set_property("minimal", True)
def standard_deviation(x):
return np.std(x)
```

The graphical plot of *Standard Deviation* for each object Wall, Car and Human is shown below:

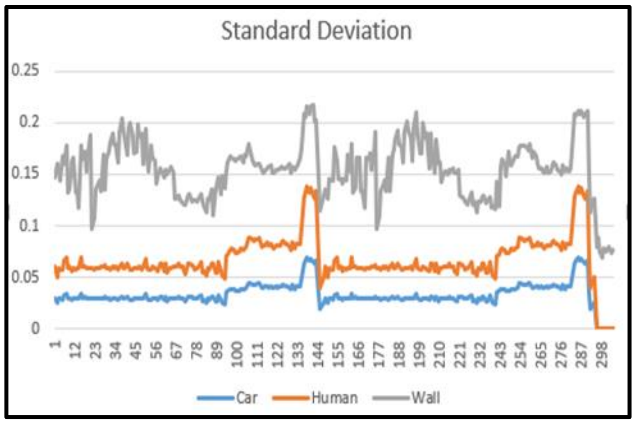


Figure 13 : Standard Deviation Graph for (Car, Human and Wall)

For all the script code "*NumPy (np)"* is also used. So now to extract all these features from the respective, excel file of each object (Wall, Human and Car). Python code was compiled and executed to extract the above three features. After extracting the feature, a new excel file will be automatically generated containing a list of above described extracted features. The code for Feature Extraction will be listed in the index section of this paper.

E. *Classification*

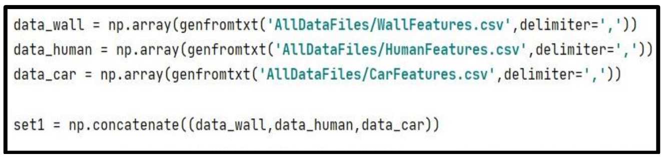
In this part discussion would be focused towards the machine learning part of the project. The classification is done between three objects Wall, Human and Car. Two different "*Supervised Machine Learning*" algorithms are used for the classification of fore mentioned objects. The Classifiers used are:

1. *SVM Classifier (Support Vector Machine)*
2. *Naive Bayes Classifier*

To produce an accurate model for classification, the data obtained after *Feature Extraction* for each object is surpassed further for applying Machine Learning concept of training, testing and predicting. To train the model 80% of the data from *Feature Extraction* was used and the rest 20% data was used for testing. Two separate python algorithms file for *SVM* and *Bayes* classifiers were created. For each classifier training and prediction of data was done. Let's discuss about how both the classifiers are working on feature extracted data in detail.

1. *SVM Classifier (Support Vector Machine)*

As already discuss about *SVM* in the introduction section. It is a type supervised machine learning algorithm. *SVM* Classifier was used in this project for object classification. To work with *SVM* approach. The data set obtained after *Feature Extraction* was divide among "Training" and "Testing". Using the trained model then it can classify between different objects. But let's first start from beginning how *SVM* algorithm was scripted. Firstly, concatenating all the data for each object Wall, Human and Car together in a single string variable. So, to have a complete set of data at one place which makes easier to train or test the model. The below attached snippet shows how a data was concatenated:

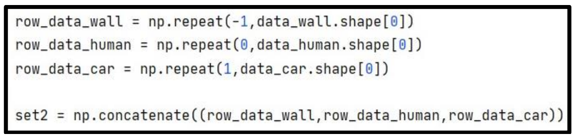


```
data_wall = np.array(genfromtxt('AllDataFiles/WallFeatures.csv',delimiter=','))
data_human = np.array(genfromtxt('AllDataFiles/HumanFeatures.csv',delimiter=','))
data_car = np.array(genfromtxt('AllDataFiles/CarFeatures.csv',delimiter=','))

set1 = np.concatenate((data_wall,data_human,data_car))
```

Figure 14 : Concatenation of All the Data

The next thing is to assign the keywords or labels for each object for object differentiation. For each object a different keyword was assigned for "*Wall data* = -1", "*Human data* = 0", "*Car data* = 1". The attached snippet below shows the python scripts for labeling each data for each object uniquely.



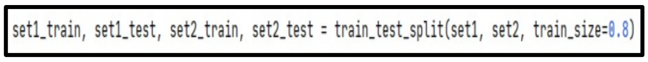
```
row_data_wall = np.repeat(-1,data_wall.shape[0])
row_data_human = np.repeat(0,data_human.shape[0])
row_data_car = np.repeat(1,data_car.shape[0])

set2 = np.concatenate((row_data_wall,row_data_human,row_data_car))
```

Figure 15 : Labels Assignment

Shape [0] means to perform row operation on data. Like from above figure for Wall it will assign label "-1" for all the Wall data. Further, respectively labelling "0" and "1" assignment for other Human and Car. In a similar way the row labelled data for each object was concatenated together as original data in a single string variable.

Now the step comes where we need to divide our model specifically for training and testing. Splitting up each set of concatenated data for training and testing. As quoted earlier 80% data for training and 20% for testing. The below snippet shows the script for it:

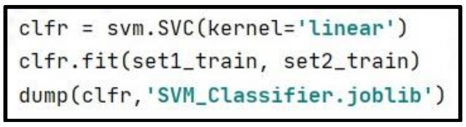


```
set1_train, set1_test, set2_train, set2_test = train_test_split(set1, set2, train_size=0.8)
```

Figure 16 : Splitting of Trained and Test Data

a) SVM Training

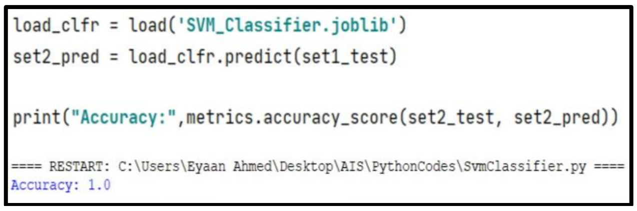
As the data is split up into training and testing portions. Let's see how is the *SVM* Training part working in the designed algorithm. For *SVM* training of data as described above in introduction section, *SVM* has many tuning parameters but for the scope of this project the tuning parameter taken into consideration is "*Linear Kernel Tuning Parameter"* for classification. To use this parameter below is the python script is attached:



```
clfr = svm.SVC(kernel='linear')
clfr.fit(set1_train, set2_train)
dump(clfr,'SVM_Classifier.joblib')
```

Figure 17 : Kernel Used

In fig. 17 Machine Learning Training is processed, Training through *SVM* Linear kernel model. The *fit()* method is used for data training to best match the curvature of given data points. After that the trained model is saved in an (*SVM_Classsifier.jobLib*). Now we have already created a of trained model to predict or classify objects for real time classification. To check the *SVM* model Accuracy a new predicted model is compared with the already test model. The Accuracy for our *SVM* model is "1" means the model is 100% accurate for our set of data. The snipped attached below shows the script and Accuracy for *SVM* Classifier.



```
load_clfr = load('SVM_Classifier.joblib')
set2_pred = load_clfr.predict(set1_test)

print("Accuracy:",metrics.accuracy_score(set2_test, set2_pred))

==== RESTART: C:\Users\Eyaan Ahmed\Desktop\AIS\PythonCodes\SvmClassifier.py ====
Accuracy: 1.0
```

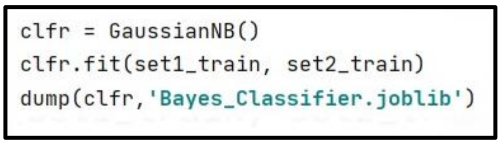
Figure 18 : SVM Model Accuracy

2) Naive Bayes Classifier

Naive Bayes is another type of Supervised Machine Learning Classifier. Another approach for object classification used in this project is using *Naive Classifier*. Let's discuss about the algorithm for Naive Classifier. The initial python script implemented is same as *SVM* classifier. That is the startup scripts for data acquisition from files shown in *Figure 14, 15, 16* is same for *Naive Bayes* as well. Coming to the training part is different as it uses Probabilistic *Gaussian Distribution* Function in *Naive Bayes*.

a) Naive Bayes

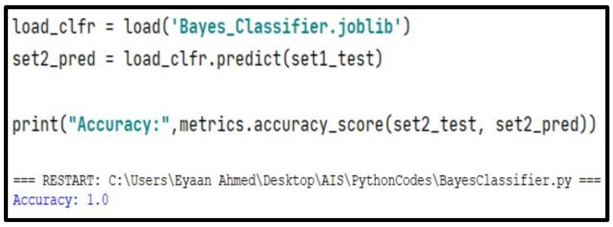
As soon as the training and testing data is splitted the next step is to train the model using *Naive Bayes Classifier*. In this project used Probabilistic *Gaussian Distribution* for training the data. The python script for this is attaches:



```
clfr = GaussianNB()
clfr.fit(set1_train, set2_train)
dump(clfr,'Bayes_Classifier.joblib')
```

Figure 19 : Naive Bayes Gaussian Distribution Training

Similarly, after training the data to check the accuracy for *Naive Bayes* Classifier, same approach as for *SVM* is used. It is found that *Naive Bayes* Classifier is also well trained as from the below figure it can be seen the accuracy is "1". Meaning that the model is 100% accurate for our defined set of data.



```
load_clfr = load('Bayes_Classifier.joblib')
set2_pred = load_clfr.predict(set1_test)

print("Accuracy:",metrics.accuracy_score(set2_test, set2_pred))

=== RESTART: C:\Users\Eyaan Ahmed\Desktop\AIS\PythonCodes\BayesClassifier.py ===
Accuracy: 1.0
```

Figure 20 : Naive Bayes Model Accuracy

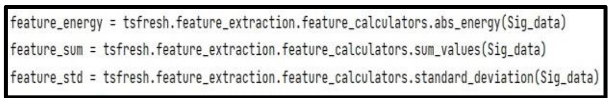
When a real time classification is to be done. First, we need to trigger *SCPI* server on *Red-Pitaya* to start data acquisition. Another important point is that the information of signal for each object is received after "*3500*" samples before that the signal is our input signal. So, classification on signal starts after *3500* sample as the object information starts after those samples.

```
Sig_data = np.array(signal_data[3500:])
```

Furthermore, from both above-mentioned Classification mechanisms. At a time only one classifier could be loaded. Either the *SVM* or *Bayes* Classifier. As shown in below figure only Bayes Classifier is loaded:

```
load_clfr = load('Bayes_Classifier.joblib')
```

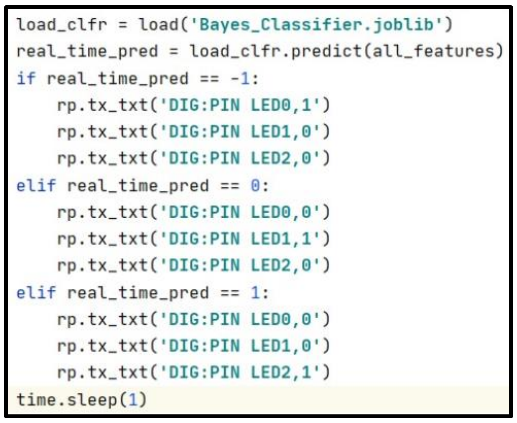
Using these already trained model, real time classification for Wall, Human and Car was done using *Red Pitaya* and ultra-sonic sensor. On a real time, classification, the signals *features extraction* on real time is again computed using *Red-Pitaya* and ultra-sonic sensor. The features computed features are *Absolute Energy*, *Absolute Sum* and *standard deviation*.



```
feature_energy = tsfresh.feature_extraction.feature_calculators.abs_energy(Sig_data)
feature_sum = tsfresh.feature_extraction.feature_calculators.sum_values(Sig_data)
feature_std = tsfresh.feature_extraction.feature_calculators.standard_deviation(Sig_data)
```

Figure 21 : Extracted Features

After the feature computation or extracting. The next phase is to checks for object detection. For which real time data is predicted with the already trained model (*SVM* or *Bayes*). Then based on which the output is displayed on *Red-Pitaya* using the *LEDs* mounted on it. For each object specific *LED* was assigned as described before. For "Wall" led "0" is assigned, for "Human" led "1" is assigned and for "Car" led "2" is assigned. Following code shows the final classification part and the assigned *LEDs* to each object:



```
load_clfr = load('Bayes_Classifier.joblib')
real_time_pred = load_clfr.predict(all_features)
if real_time_pred == -1:
    rp.tx_txt('DIG:PIN LED0,1')
    rp.tx_txt('DIG:PIN LED1,0')
    rp.tx_txt('DIG:PIN LED2,0')
elif real_time_pred == 0:
    rp.tx_txt('DIG:PIN LED0,0')
    rp.tx_txt('DIG:PIN LED1,1')
    rp.tx_txt('DIG:PIN LED2,0')
elif real_time_pred == 1:
    rp.tx_txt('DIG:PIN LED0,0')
    rp.tx_txt('DIG:PIN LED1,0')
    rp.tx_txt('DIG:PIN LED2,1')
time.sleep(1)
```

Figure 22 : Classification

F. Plug and Play

The plug and play feature enable real time classification to be started as soon as the *Red-Pitaya* and *Raspberry Pi* is boot up. All the python scripts placed inside the *Raspberry Pi*. Created a folder in *Raspberry Pi* and all the required python files and libraries are added inside the folder.

After that created one shell script file with following script

```
./ssh.sh & python3 Classification.py
```

"ssh.sh" is the script which triggers the *Red-Pitaya* ultra-sonic sensor. "Classification.py" is our main file which contains the required part to compute *feature extraction* of real time data and applying prediction on real time data using already trained model.

To make a model *Plug and Play* there are many options available. But for the scope of this project *"Systemd"* files are used to start any script during *Raspberry Pi* boot up. "*Systemd*" provides a standard process for controlling what programs run when a *Linux* system boots up. Following necessary steps were taken for plug and play feature.

To make our model *Plug and Play* the already shell script that was created needed to be first made executable. This can be done using:

```
/chmod +x code_script.sh
```

Where *"code_script.sh"* is the filename. Now using this executable script in system files to run during boot up. Create a unit file using following script:

```
sudo nano /lib/systemd/system/startup.service
```

In the unit file added the following information to make the script execute at boot.

```
[Unit]
Description=My Startup Service
After=network.target

[Service]
Type=idle
ExecStart=/bin/bash /home/pi/Desktop/PythonCodes/code
_script.sh

[Install]
WantedBy=multi-user.target
```

Once the *"Systemd"* file is changed or updated, the need is to reload the system configuration.

```
sudo systemctl daemon-reload
```

Now once the system file is updated one can check able to start, stop, restart and check the service status of

```
sudo systemctl start startup.service
sudo systemctl stop startup.service
sudo systemctl restart startup.service
systemctl status startup.service
```

The last step is to *"Enable"* the service to start automatically at boot up. This was how *Plug and Play* feature enabled.

```
sudo systemctl enable startup.service
```

IV. Comparison Between SVM and Bayes Classifier

Based on achieved result it is shown *figure 18* and *figure 20*. That both the classifiers are providing almost an accuracy of "1". But when a real time classification is done *Naive Bayes* provides better results compared to *SVM*. This is stated because the object classification speed for *Naive Bayes* was faster compared to *SVM*. But on general no further difference based on model accuracy was found.

V. Results

The final computed result when a real time classification is done using *Red Pitaya* ultra-sonic sensor. In the below the level of accuracy for each object is defined. With an accuracy level of "1" meaning highly accurate. The results showed that object classification for "Wall" and "Human" is working with full accuracy. The real time classification classifies Wall and Human with 100% accuracy. The accuracy for these two objects was tested with both the Classification models i.e. (*SVM* and *Naive Bayes*). Contrary to this the object classification for "Car" had a negative result sometimes. It could be said that the classification rate may be between 40%. Sometimes the classifiers define Car as Human. This was the result tested using both the classifiers. The Car is sometimes classified as Human because the data set for Car and Human shows a lot of similarities this can be also seen from the graphs shown in section *C* and section *D* of this document

Object	Classification Accuracy Rate
Wall	0.977 ~ 1.000
Human	0.964 ~ 1.000
Car	0.54 ~ 0.6

VI. References

[1] https://redpitaya.readthedocs.io/en/latest/appsFeatures/remoteControl/SCPI_commands.html

[2] https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html

[3] https://wiki.dfrobot.com/SRF02_Ultrasonic_sensor__SKU_SEN0005_

[4] https://redpitaya.readthedocs.io/en/latest/developerGuide/scpi/scpi.html

[5] https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[6] https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python

[7] https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/

[8] https://towardsdatascience.com/naive-bayes-in-machine-learning-f49cc8f831b4

[9] T. Ylonen; C. Lonvick (January 2006). The Secure Shell (SSH) Protocol Architecture. Network Working Group of the IETF. doi:10.17487/RFC4251. RFC 4251

[10] https://www.redpitaya.com/f145/specifications

[11] https://en.wikipedia.org/wiki/Standard_deviation

VII. Index

1) *SignalSave Code*

After enabling *SSH* connection we will use the following code to save the signal plots.

```
import redpitaya_scpi as scpi
import matplotlib.pyplot as plt
import numpy as np
import time

rp = scpi.scpi('192.168.128.1')
while 1:
   rp.tx_txt('ACQ:DEC 64')
   rp.tx_txt('ACQ:TRIG EXT_PE')
   rp.tx_txt('ACQ:TRIG:DLY 8192')
   rp.tx_txt('ACQ:START')
   while 1:
      rp.tx_txt('ACQ:TRIG:STAT?')
      if rp.rx_txt() == 'TD':
         break
   rp.tx_txt('ACQ:SOUR1:DATA?')
   str = rp.rx_txt()[1:-1]
   signal_data = np.fromstring(str,dtype=float,sep=',')
   plt.plot(signal_data)
   plt.show()
   signal_array = np.array([signal_data])
   new_file = open("Wall1.csv", "a")
   np.savetxt(new_file,signal_array,fmt='%3.8f',delimiter=',')
   new_file.close()
   time.sleep(3)
   plt.close()
```

2) *SignalFeatures Code*

```
import numpy as np
from numpy import genfromtxt
import tsfresh

data_wall = np.array(genfromtxt('AllDataFiles/Wall.csv',delimiter=','))
for x in range(data_wall.shape[0]):
Sig_data_wall = np.array([data_wall[x][3500:]])
feature_energy = tsfresh.feature_extraction.feature_calculators.abs_energy(Sig_data_wall[0])
feature_sum = tsfresh.feature_extraction.feature_calculators.sum_values(Sig_data_wall[0])
feature_std = tsfresh.feature_extraction.feature_calculators.standard_deviation(Sig_data_wall[0])
all_features = np.array([[feature_energy, feature_sum, feature_std]])
new_file = open("AllDataFiles/WallFeatures.csv", "a")
np.savetxt(new_file,all_features,fmt='%3.8f',delimiter=',')
new_file.close()

data_human = np.array(genfromtxt('AllDataFiles/Human.csv',delimiter=','))
for x in range(data_human.shape[0]):
Sig_data_human = np.array([data_human[x][3500:]])
feature_energy = tsfresh.feature_extraction.feature_calculators.abs_energy(Sig_data_human[0])
feature_sum = tsfresh.feature_extraction.feature_calculators.sum_values(Sig_data_human[0])
feature_std = tsfresh.feature_extraction.feature_calculators.standard_deviation(Sig_data_human[0])
all_features = np.array([[feature_energy, feature_sum, feature_std]])
new_file = open("AllDataFiles/HumanFeatures.csv", "a")
np.savetxt(new_file,all_features,fmt='%3.8f',delimiter=',')
new_file.close()

data_car = np.array(genfromtxt('AllDataFiles/Car.csv',delimiter=','))
for x in range(data_car.shape[0]):
Sig_data_car = np.array([data_car[x][3500:]])
feature_energy = tsfresh.feature_extraction.feature_calculators.abs_energy(Sig_data_car[0])
feature_sum = tsfresh.feature_extraction.feature_calculators.sum_values(Sig_data_car[0])
feature_std = tsfresh.feature_extraction.feature_calculators.standard_deviation(Sig_data_car[0])
all_features = np.array([[feature_energy, feature_sum, feature_std]])
new_file = open("AllDataFiles/CarFeatures.csv", "a")
np.savetxt(new_file, all_features, fmt='%3.8f', delimiter=',')
new_file.close()
```

3) Classification Code

```
import redpitaya_scpi as scpi
import numpy as np
import tsfresh
from joblib import load
import time

rp = scpi.scpi('192.168.128.1')
while 1:
   rp.tx_txt('ACQ:DEC 64')
   rp.tx_txt('ACQ:TRIG EXT_PE')
   rp.tx_txt('ACQ:TRIG:DLY 8192')
   rp.tx_txt('ACQ:START')
   while 1:
      rp.tx_txt('ACQ:TRIG:STAT?')
      if rp.rx_txt() == 'TD':
         break
   rp.tx_txt('ACQ:SOUR1:DATA?')
   str = rp.rx_txt()[1:-1]
   signal_data = np.fromstring(str,dtype=float,sep=',')
   Sig_data = np.array(signal_data[3500:])
   feature_energy = tsfresh.feature_extraction.feature_calculators.abs_energy(Sig_data)
   feature_sum = tsfresh.feature_extraction.feature_calculators.sum_values(Sig_data)
   feature_std = tsfresh.feature_extraction.feature_calculators.standard_deviation(Sig_data)
   all_features = np.array([[feature_energy, feature_sum, feature_std]])
   load_clfr = load('Bayes_Classifier.joblib')
   real_time_pred = load_clfr.predict(all_features)
   if real_time_pred == -1:
      rp.tx_txt('DIG:PIN LED0,1')
      rp.tx_txt('DIG:PIN LED1,0')
      rp.tx_txt('DIG:PIN LED2,0')
   elif real_time_pred == 0:
      rp.tx_txt('DIG:PIN LED0,0')
      rp.tx_txt('DIG:PIN LED1,1')
      rp.tx_txt('DIG:PIN LED2,0')
   elif real_time_pred == 1:
      rp.tx_txt('DIG:PIN LED0,0')
      rp.tx_txt('DIG:PIN LED1,0')
      rp.tx_txt('DIG:PIN LED2,1')
   time.sleep(1)
```

4) SVM Classifier Code

```
import numpy as np
from numpy import genfromtxt
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn import metrics
from joblib import dump, load

data_wall = np.array(genfromtxt('AllDataFiles/WallFeatures.csv',delimiter=','))
data_human = np.array(genfromtxt('AllDataFiles/HumanFeatures.csv',delimiter=','))
data_car = np.array(genfromtxt('AllDataFiles/CarFeatures.csv',delimiter=','))

set1 = np.concatenate((data_wall,data_human,data_car))

row_data_wall = np.repeat(-1,data_wall.shape[0])
row_data_human = np.repeat(0,data_human.shape[0])
row_data_car = np.repeat(1,data_car.shape[0])

set2 = np.concatenate((row_data_wall,row_data_human,row_data_car))

set1_train, set1_test, set2_train, set2_test = train_test_split(set1, set2, train_size=0.8)

clfr = svm.SVC(kernel='linear')
clfr.fit(set1_train, set2_train)
dump(clfr,'SVM_Classifier.joblib')

load_clfr = load('SVM_Classifier.joblib')
set2_pred = load_clfr.predict(set1_test)

print("Accuracy:",metrics.accuracy_score(set2_test, set2_pred))
```

5) *BayesClassifier Code*

```
import numpy as np
from numpy import genfromtxt
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn import metrics
from joblib import dump, load

data_wall = np.array(genfromtxt('AllDataFiles/WallFeatures.csv',delimiter=','))
data_human = np.array(genfromtxt('AllDataFiles/HumanFeatures.csv',delimiter=','))
data_car = np.array(genfromtxt('AllDataFiles/CarFeatures.csv',delimiter=','))

set1 = np.concatenate((data_wall,data_human,data_car))

row_data_wall = np.repeat(-1,data_wall.shape[0])
row_data_human = np.repeat(0,data_human.shape[0])
row_data_car = np.repeat(1,data_car.shape[0])

set2 = np.concatenate((row_data_wall,row_data_human,row_data_car))

set1_train, set1_test, set2_train, set2_test = train_test_split(set1, set2, train_size=0.8)

clfr = GaussianNB()
clfr.fit(set1_train, set2_train)
dump(clfr,'Bayes_Classifier.joblib')

load_clfr = load('Bayes_Classifier.joblib')
set2_pred = load_clfr.predict(set1_test)

print("Accuracy:",metrics.accuracy_score(set2_test, set2_pred))
```