MACHINE LEARNING(ML) FOR GESTURE RECOGNITION.

A Project Report

submitted by

RAHUL MEENA

in partial fulfillment of the requirements for the award of the degree of

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

JUNE 2021

CERTIFICATE

This is to certify that the report titled **Machine Learning (ML) for Gesture Recognition**, submitted by **RAHUL MEENA**, to the Indian Institute of Technology Madras, for the award of the degree of **Master of Technology**, is a bonafide record of the work done by him under my supervision. The contents of this report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Place: Chennai

Date: 17-06-2021

Prof. Anil Prabhakar
Professor
Dept. of Electrical Engineering
IIT Madras, 600 036

ACKNOWLEDGEMENTS

I am extremely grateful to my advisor Prof. Anil Prabhakar without whose constant support and guidance, this work would not have been possible. I am also grateful to him for his incredible patience and for granting me the opportunity to learn multiple things all of which have been instrumental in shaping my research interests. Further, I would also like to thank all the members working at the Enability Company who have all, in some way shape or form, made my work possible. In particular, I am grateful to Vel Murugan, and Kasthuri for their illuminating insights and to Stalin whose company made working at the lab all the more enjoyable. It would be remiss if I did not thank my parents, whose unwavering support has encouraged me to endeavour to perform good research and persevere through difficult times.

ABSTRACT

Nowadays Gesture recognition has been widely used for the persons who have certain disorders such as Cerebal palsy, Aphsia, Metachromatic leukodystrophy, Pelizaeus Merzbacher etc i.e the person who have problems in speaking and those who can not able to express what they want to say properly to let the people understand. For those persons we can make a wearable device in particularly for hand, in order to let the people understand what they want to speak by performing gestures using hand. So, we can implement machine learning algorithms on the wearable device in order to get fast and accurate predictions of gestures. The aim of this work was to learn, understand, and implement the Machine learning algorithms in particular K-nearest neighbour (aka KNN) and the fully connected feed-forward neural network model (aka Multi-layer perceptron) on the collected different gesture's (in this work we have collected data of 5 different gestures) data with the help of IMU MPU 6050 sensor mounted on the Robotic Arm by using BLE HM-10. Subsequently, did the comparison of above mentioned algorithms and can be concluded that the feed-forward neural network algorithm gave a better accuracy compared to that of K-nearest neighbour algorithm. We have achieved the accuracy with K-nearest neighbour algorithm is 65-72% and with feed-forward neural network model is 85-90%.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

KNN	K-Nearest Neighbour
NN	Neural Network
FFNN	Feed Forward Neural Network
IMU	Inertial Measurement Unit
HGR	Hand Gesture Recognition
RNN	Recurrent Neural Networks
LSTM	Long Short Term Memory
DTW	Dynamic Time Wrapping
BLE	Bluetooth Low Energy
EMG	Electromyography
VBR	Vision Based Recognition
SBR	Sensor Based Recognition
AdaGrad	Adaptive Gradient
RMSProp	Root Mean Square Propagation
Adam	Adaptive Moment Estimation
SGD	Stochastic Adaptive Descent
ReLu	Rectified Linear Unit
MSE	Mean Square Error
MAE	Mean of the Absolute Error

CHAPTER 1

Introduction

Hand gestures are precise movement of fingers and hands that represent a particular message in a non verbal communication. Gestures also enhance verbal communication through communication human intentions in specific conversations. HGR is a intuitive calculation that enables machines to identify the hand movements and perform the appropriate action. Gesture recognition is an area of active current research in computer vision and machine learning. HGR is a field in which many researchers in the academic institutions and in the industries are working on multiple applications to make interactions more easy, natural, accurate and convenient. Although extremely good development has been made these days, fast and robust hand gesture popularity stays an open trouble, on the grounds that existing techniques have not presented a sensible compromise between the performance and the efficiency.

HGR approaches can be separated into two classes that are Vision Based Recognition algorithms and Sensor-Based Recognition algorithms. The VBR algorithms perform gesture recognition from images which is captured by a camera. Although accurate classification is possible but high computational efforts may be required to extract information from images for both training and inference operation.

The SBR algorithms as the name suggests is based on various sensors such inertial measurement unit(IMU) sensors, electromyography(EMG) sensors, brain wave sensors, electrocardiograph sensors, and radar sensors. Here in this project we have used IMU MPU 6050 sensor which is owing to their low cost and low power characteristics. In addition, due to the fact IMU sensors can be at once attached to the user's body, they can obtain notably correct hand gesture data.

In some studies according to SBR, The accuracy of the gesture recognition classifier was observed. However most of these techniques Serial gesture recognition is not supported. Only individual actions can be recognized.

Recurrent neural networks(RNN), Long short term memory(LSTM), Dynamic time wrapping(DTW), K-nearest neighbour algorithm(KNN), Naive bayes, Fully connected feed forward neural network (FFNN)(aka Multi layer perceptron) with Adam optimization algorithm are frequently used to recognize hand gestures with IMU sensors. In this project we have considered two algorithms i.e KNN and FFNN to train and classify the model based on the the sensor data. We have observed from the results that FFNN is more efficient and give better accuracy i.e 85-90% as compared to that of KNN i.e 65-72%.

In this report, Chapter 2 details the theoretical background of all the components that we have used in our project, then discussed how we used them to calculate the sensor reading and lastly discussed what are the algorithms we have used and then explained it, in Chapter 3 we discussed the results we have got by using both of these algorithms i.e KNN and FFNN, then finally in the Chapter 4 we discussed the future scope of this work and then concluded the work we have done.

CHAPTER 2

Theory and Algorithms

This chapter presents about Theory of gesture recognition, How we have collected the data, What are the components we have used for data collection, What are the gestures we have considered in our project and then Lastly this chapter will focus on What are the algorithms that we have used for training and classifying the model.

2.1 Gesture Recognition

2.1.1 Background

Users interact with computers through the provided interfaces, motion or vocal. These different interactions need to be such that information retrieval is easier and Human Computer Interaction(HCI) is concerned with which the way humans interact with technology. It deals with how humans work with computers and how computer systems can be designed to best facilitate the users in achieving the goals. In future days, Human Computer Interaction will become a field with a variety of sectors that need to characterize it. Users will be able to use any type of interaction which is a potential part of HCI, Interactions can be body movements, hand movements, facial gestures and vocals.

A Human Computer Interaction(HCI) has several types of interaction and one of those is called gestures. One simplest definition of a gesture is a non-verbal method of communication utilised in HCI interfaces. The high target of a gesture is to design a specific system that can identify human gestures a designedly and use these gestures to convey information for device control. Our primary factor in this project is the matching between the system and the real world which ensures that the system should use the the users movements.

2.1.2 Definition and Types of Gesture Recognition

In the previous section Gesture Recognition were defined as non-verbal motions used as a method of communication in HCI interfaces. Gestures are one of the significant aspects of HCI in both interpersonally and in the device interfaces. Another definition of gestures is physical movements or positions of human’s finger, hand, arms or full body used to convert information. The process by which gestures are formed in certain ways by a person, are made to known to a system, is the main principle of gesture recognition. Commonly used Gesture Recognition techniques are: vision-based recognition and non-vision-based recognition method. In vision based approach, algorithms are used to derive face, hand, body pose and trajectory information and environmental factors such as background illumination, hands occlusion, and skin color plays a significant role during recognition process. In non-vision based scheme, users need to wear data gloves, bands or in general wearable things and handle the cabling part of recognizing system.

2.1.3 Components and Gesture images

The hardware components which we have used in the process of collection of data for different gestures are: Arduino UNO microcontroller, IMU MPU 6050, HM-10 BLE and Robotic Arm.

Arduino UNO microcontroller is the one of the most popular Arduino boards. It consists of 14-digital I/O pins, where 6-pins can be used as PWM(pulse width modulation outputs), 6-analog inputs, a reset button, a power jack, a USB connection and more. It includes everything required to hold up the microcontroller, simply attach it to a PC with the help of a USB cable and give the supply to get started with AC-to-DC adapter or battery.

IMU MPU 6050 is a MEMS-based 6-axis motion tracking device. It has an on-chip gyroscope and accelerometer sensors along with temperature sensor. MPU 6050 is a digital device. This module is of very small in size, has a low-power consumption requirements, highly accurate, has high repeatability, high shock tolerance, it has application-specific performance programmability and low consumer prices points.

MPU 6050 can be easily interfaced with other sensors such as magnetometers and microcontrollers.

HM-10 BLE is a Bluetooth 4.0 Module that includes Bluetooth Low Energy(BL-E). This module transmits over the 2.4 Ghz ISM band like traditional Bluetooth, however Bluetooth Low Energy(BLE) uses considerably less power while still maintaining it's effective communication range. This makes Bluetooth Low Energy(BLE) a viable option for IoT communication devices.

Robotic Arm is basically the Robotic manipulators resembling human arm. They are constituted by a structure consisting of structurally robust links coupled by either rotational joints(also referred to as revolute joints) or translating joints(also referred to as prismatic joints) a robotic arm is thus a type of mechanical arm, usually programmable, with similar functions to a human arm.

So we have used the above described components as our basic building blocks to make the circuit and then by using arduino and python codes we have got the sensor data or we can say data of 5 (five) different gestures that we have make the Robotic Arm to perform.

There are some circuits(parts of the circuits required for final circuit) that we have considered to make the final circuit by final circuit I mean to say that all the above components are connected all together, that we have made it in the lab.

<u>Arduino UNO - MPU 6050 interface</u>:

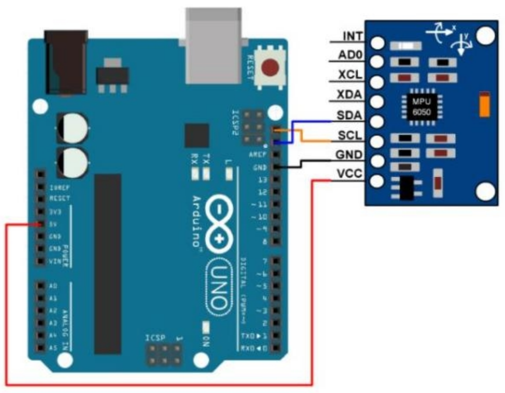


Figure 2.1: Interfacing MPU6050 Module With Arduino

<u>Arduino UNO - HM-10 interface</u>:

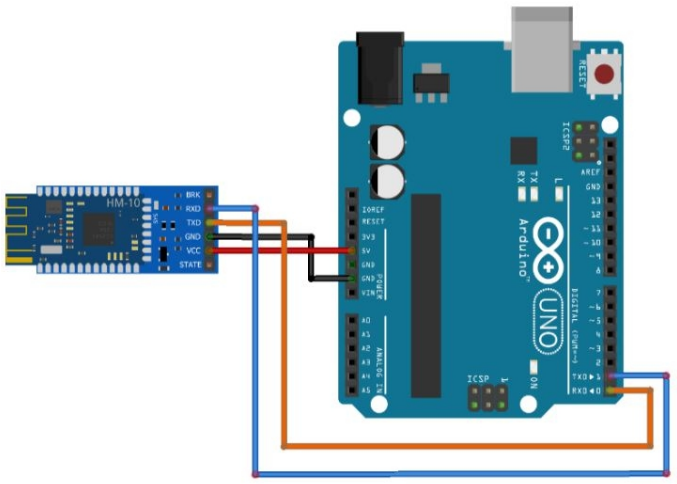


Figure 2.2: Connecting Arduino and HM-10 BLE Module

<u>Arduino UNO - Robotic Arm interface</u>:

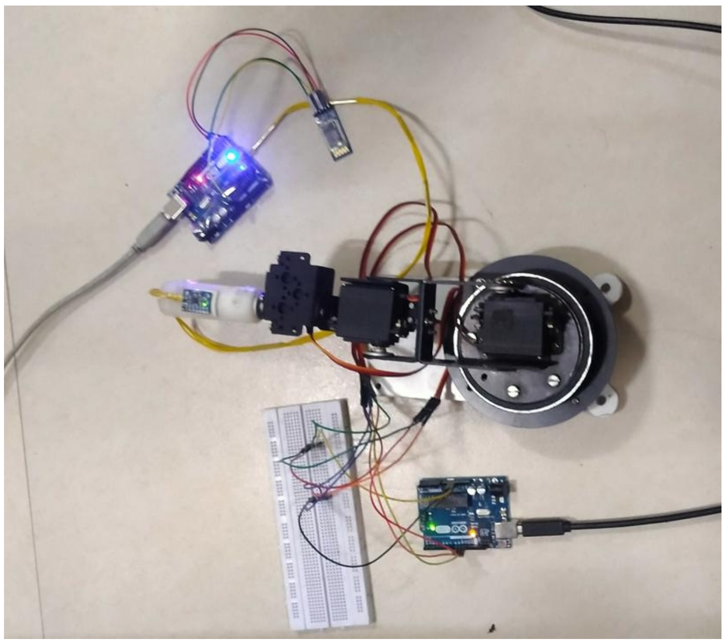


Figure 2.3: Connecting Arduino, HM-10, MPU 6050 and Robotic Arm

So, as you can see Figure 2.3 was our final circuit that we have used to get the sensor data i.e the data we were getting from MPU 6050.

As you can see a MPU 6050 is mounted on the white color part of the Robotic Arm i.e is in fact the wrist of the Robotic Arm. There are three motors in this Robotic Arm i.e for Shoulder, Elbow and for Wrist.

In our final circuit as you can see there are two Arduino UNO, one Arduino is dedicated to the Robotic and the other one is for MPU 6050 and HM-10. So, in the above circuit what we were doing was we were given some commands to the Robotic Arm like what will be the initial position of Robotic Arm, what will be the Baud Rate, for every movement of any of the three motors should be synchronous i.e be it Elbow movement, Shoulder movement or Wrist movement the time they will take will be equal for all, we were given power supply to these servo motors by the help of Arduino Code burn it on the Arduino UNO. So, by this we can say one Arduino was controlling Robotic Arm. The other Arduino were collecting the values that it was receiving from the MPU 6050 which was mounted on the Robotic Arm while the Robotic Arm were in motion and the transmitting those received data on the computer by Using Python code. Remember we were getting the data on computer for every 50ms.

In our work we have considered five(5) different gestures and for each and every gesture we made the Robotic Arm to perform the same gesture hundred(100) times, and collected the data for all the gestures.

The images of Gestures that we have taken in the consideration are given below:

Gesture 1 :

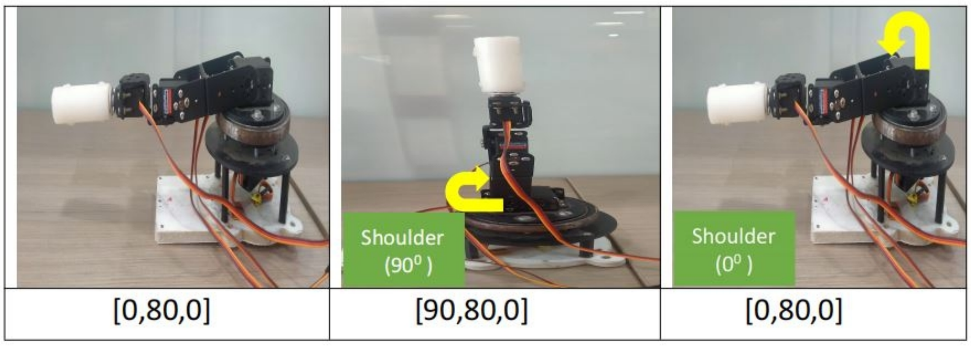


Figure 2.4: Gesture 1

So, we have assumed [0,80,0] was our initial position in each and every gesture, all elements in the bracket are in degrees. The initial position means our fingers are toward the ground i.e each of them are at rest or at Zero degree. So, Gesture 1 says move your Shoulder with 90 degree, and then move it back to the initial position.

<u>Gesture 2</u>:

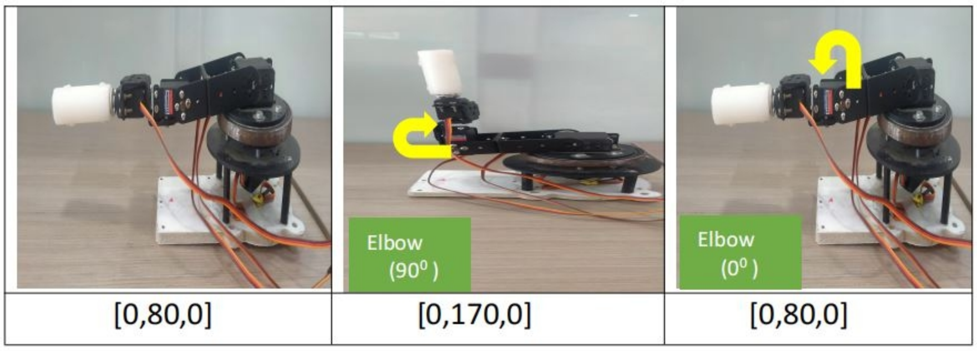


Figure 2.5: Gesture 2

In Figure 2.4, we have considered it as Gesture 2. In this Gesture we just have to move our Eblow wirh 90 degree and then get it back to our initial position.

<u>Gesture 3</u>:

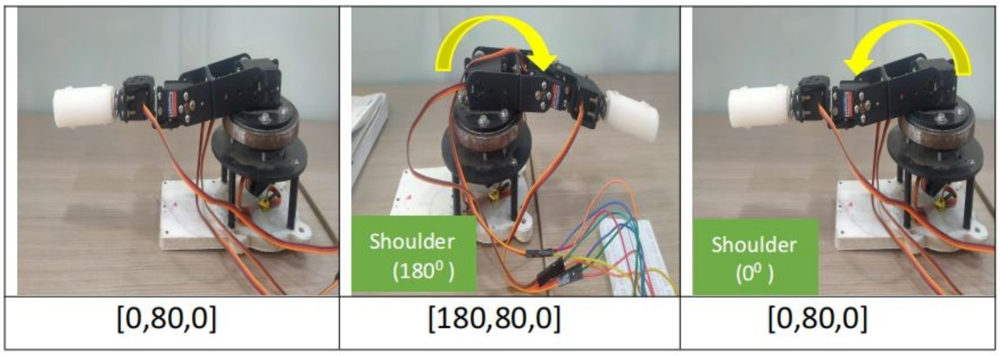


Figure 2.6: Gesture 3

In Figure 2.5, we have considered it as Gesture 3. In this Gesture we just have to move Shoulder with 180 degree and then move it back to the initial position.

<u>Gesture 4</u>:

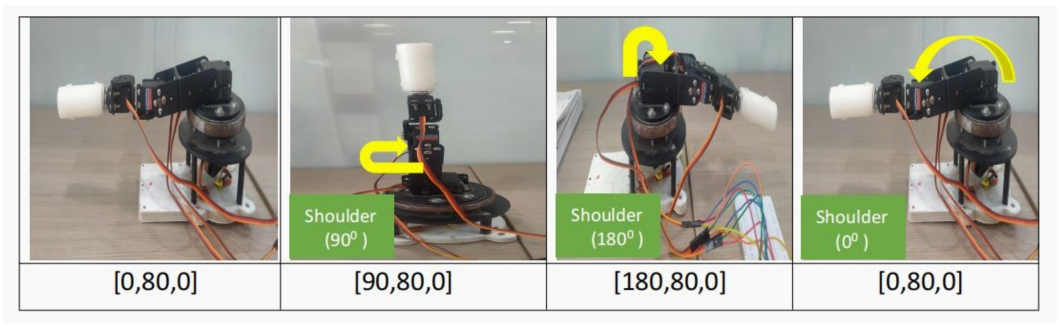


Figure 2.7: Gesture 4

In Figure 2.6, we have considered it as Gesture 4. In this Gesture we first move our Shoulder with 90, then further move that to 90 degree and finally move back to our initial position with 180 degree. Here while going from 0 to 180 degree we have to stop at 90 degree unlike Gesture 2.

<u>Gesture 5</u>:

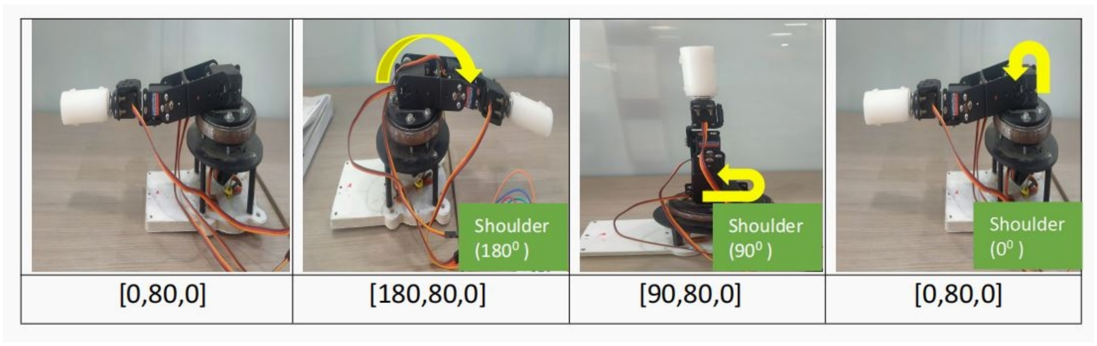


Figure 2.8: Gesture 5

In Figure 2.7, we have considered it as Gesture 5. In this Gesture from the initial position Shoulder has to go directly with 180 degree, then while getting back to the initial position it should have to Stop at Shoulder reaches 90 degree position, then finally get back to Zero degree that is our initial position.

2.2 Algorithms

As you have seen in the above section we have got the sensor data of all the 5 gestures and we saved that data. Now we need to train and classify the algorithms, in order to implement these algorithms in real-time based on the sensor data(Gesture data) we have got. So, here in our work we have considered two models i.e K-Nearest Neighbour and Fully Connected Feed Forward Neural Network(aka Multi Layer Perceptron). So let's discuss those Algorithms.

2.2.1 K-Nearest Neighbor

Introduction:

K-nearest neighbor algorithm is a algorithm that comes under the category of supervised machine learning algorithm which can be used for both classification and regression predictive problems, but it is mostly used for classification related problems in the industry. It is one of the most basic yet essential classification algorithms in the machine learning. The model representation for KNN is the entire training set i.e this algorithm has no model other than training data set that' why here no learning is required. The following three properties would define a KNN well-

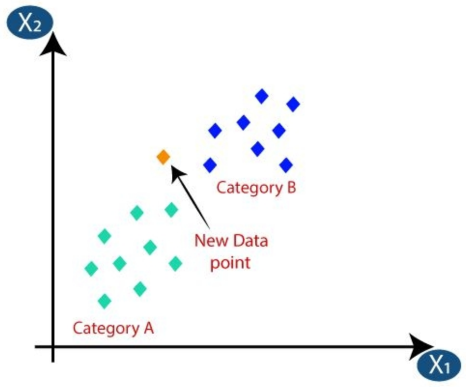
1. **Instance based learning**: Here we do not learn weights from training data to predict the data, unlike as we do in model based learning. However, here we use entire training data to predict output for new data.
2. **Lazy learning**: KNN is known as lazy learning because it does not have specialized training phase and uses all the data for training while classification.
3. **Non-parametric**: KNN is also a non-parametric learning algorithm because there is no predefined form of the mapping function.

Working of KNN Algorithm:

L-nearest neighbour algorithm uses similarity of features in order to predict the values of new query which means that the new query is going to assigned the value based of how closely it matches the value in the training set. We can understand the it's working with the help of the following example:

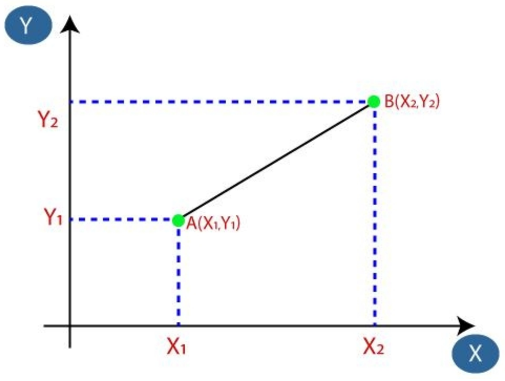
The following is an example to understand the concept of K and working of KNN algorithm.

Suppose we have a new data point and we need to put it in the required category. Consider the below image:



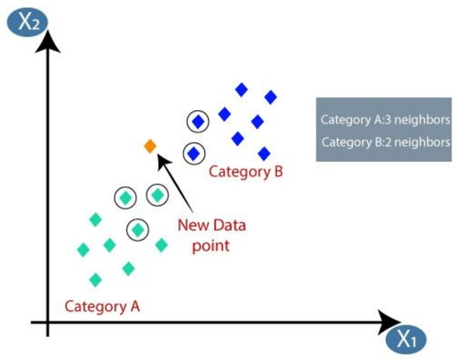
Firstly, here we will choose the number of numbers let' say k=5.

Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between the two points, which we have already familiar about. It can be calculated as:



Euclidean distance between A and B = $\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$

By, calculating the Euclidean distance we got the nearest neighbors, as we can see in the below image that in Category A, three neighbors are there and in Category B, two neighbors are there. Consider the below image:



As we can see maximum number of nearest neighbors that is 3 are from Category A. So, this new point belongs to Category A.

<u>Pseudo Code For K- nearest neighbor</u>:

```
1. Load the data.
2. Initialize K in order to choose the number of neighbors.
3. For each example in the data
3.1 Calculate the Euclidean distance between the new data point and the
Current example from the training data.
3.2 Add the distance and the index of the new point to an ordered collection.
4. Sort the ordered collection of distances and indices from smallest to
largest(in ascending order) by the distances.
5. Pick the first K entries from the sorted collection.
6. Get the labels of the selected K entries.
7. If Regression, returns the mean of the K labels.
8. If Classification, returns the mode of the K labels.
```

<u>Advantages of KNN</u>:

1. It is a very simple algorithm and very useful for non-linear data because there is no assumption about data in this algorithm.
2. It has relatively high accuracy but there are much better other supervised learning algorithms.
3. There is no need to build a model, tune several parameters or make additional assumptions.

Disadvantages of KNN:

1. Higher memory storage required as compared to other supervised learning algorithms.

2. The algorithm get significantly slower as the number of examples and independent variables increase.

2.2.2 Feed-Forward Neural Network (Multi Layer Perceptron)

For a single perceptron there is a limitation, it can only deal with the functions which are linearly separable. So, that is why we need Multi layer perceptron and it's proved that it can handle arbitrary boolean function whether linearly separable or not. So, we will need a large number of neurons in a hidden layer. We also know that the perceptron have the harsh thresholding logic so which makes the decision very unnatural.

Feed-forward:

A general version of neural network is called Feed-forward neural network. The fully connected feed forward neural network looks like the following diagram:

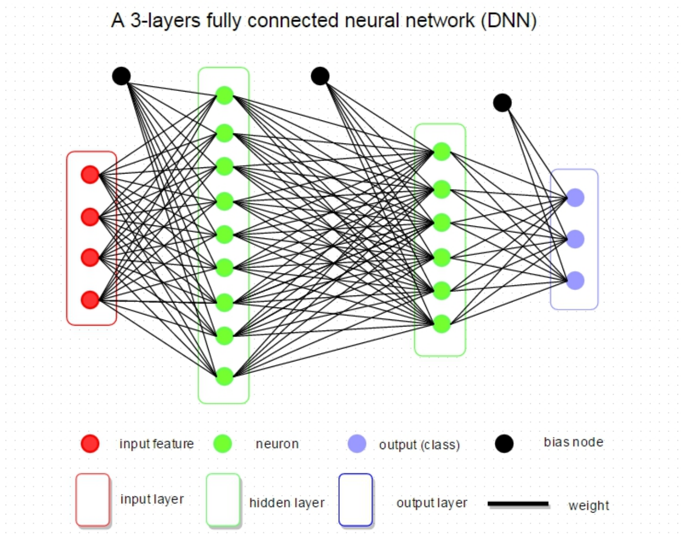


Figure 2.9: Multi Layer perceptron

A Feed-forward network or simple neural network the term you would have heard basically a collection of neurons, each of these units here are neuron. Now each of these neurons neurons or each of these layers which are concatenated vertically has specific name, the very first layer is called Input Layer. Here you will have multiple features but in our work we have 6 features i.e Acx, Acy, Acz, Gyx, Gyy, Gyz. The intermediate layers are called Hidden Layers, as you can see in Figure 2.8 number of Hidden Layers are 2 which is same as we have considered in our work but the number of units or neurons is different. In our work we have considered 12 units in the first hidden layer and 6 units in the second hidden layer. You could have multiple Hidden Layers. If the number of Hidden Layer is greater than one, then it's called deep network. The final layer where you actually get the output you are interested in is called Output Layer. Here as you can see in the Figure 2.8, the number of classes or targets in the Output Layer are 3 but in our work we have 5 classes for 5 gestures. The number of neurons in the Output Layer need not be equal to the number of neurons in the Input Layer and in general each layer might have different size. Now each of these elements here is an artificial neuron, we can even treat the Input Layer as if they were neurons but generally it's only after the input layer that we look at each of these neurons and call them artificial neurons.

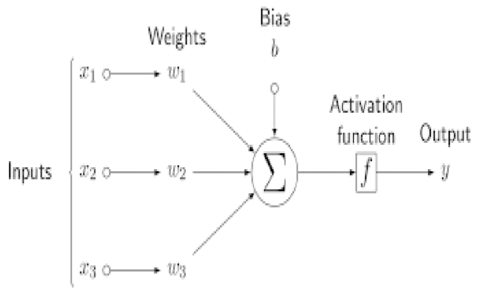


Figure 2.10: Single neuron

Each neuron in the Hidden Layer and the Output Layer can be split into two parts i.e preactivation and activation. Preactivation does aggregation(linear combination) and activation does non-linearity. Now, if I look at any neuron, it has inputs coming from all the previous entities in the input layer. So, let's assume each neuron has n-inputs plus a bias unit, here in Figure 2.9 there are 3 inputs but in our work we have already mentioned that we have 6 inputs.

Activation Functions:

These are the functions which take in some input and provides some output depending on the speciality of the function. For example a common activation function is sigmoid function. The sigmoid function squeezes the given input between 0 and 1. However, in our work we have used Relu function for the Hidden Layers and Softmax function for the Output Layer.

Relu function is a piecewise linear function that will output the input directly if it is positive otherwise, it will output zero.

Softmax function that converts vector of numbers into a vector of probabilities, where the probabilities of each values are proportional to the relative scale of each value in the vector.

The formal way of looking what happens at a single node:

$$\text{Output of node} = \mathrm{y}(\mathrm{W}^{\mathrm{T}} * \mathrm{X} + \mathrm{b})$$

Where y= Activation function, W= Weights matrix, X= Input matrix, b= Bias term, W^T= Transpose of Weight matrix.

Loss function or Cost function:

In supervised machine learning algorithms, we want to minimize the error for each training example during the learning process. This is done using some optimization strategies like Gradient Descent, Stochastic Gradient Descent, AdaGrad, RMSProp, Adam etc and this error we get from the Loss function. Loss function is the average loss over the entire training data set. The optimization strategies aim at minimizing the cost function. Various type of Loss functions are available such as Mean square Error(MSE), Mean of the Absolute Error(MAE), Huber Loss etc. We have used Huber Loss function in our work.

Huber Loss combines the best properties of MSE and MAE. It is quadratic for smaller errors and is linear(and similarly for it's gradient). It is identified by it's delta parameter.

$$\mathrm{L}_{\delta} = \begin{cases} \frac{1}{2}(y - f(x))^2, \ if \ |y - f(x)| \le \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, \ Otherwise \end{cases}$$

Huber Loss is more robust to outliers than MSE. It is used in Robust Regression, M-estimation, and Additive modelling.

<u>Optimization Algorithm</u>:

Gradient descent is an optimization algorithm that follows the negative gradient of a Loss function in order to find the minimum of the function. A limitation of Gradient descent is that a single step size(learning rate) for all the input variables. Improved versions of Gradient descent like AdaGrad and RMSProp update the algorithm to use different step sizes for all of the input variables but may result in step size that rapidly decreases to very small values.

The Adaptive Moment Estimation(Adam) algorithm is an improved version of Gradient descent and the combination of AdaGrad and RMSProp that automatically adapts step size for all input variables for the Loss function and further smooths the search process by using an exponentially decreasing moving average of the gradient to make the update variables. We have used Adam algorithm in our work and following the algorithm shown as:

<u>Adam Optimizer (Learning Algorithm)</u>:

```
1. Initialize $m_0 \leftarrow 0, v_0 \leftarrow 0, t \leftarrow 0$
while $\theta_t$ not converged do
2. $m_t = \beta_1 * m_{t-1} + (1-\beta_1) * g_t$
3. $v_t = \beta_2 * v_{t-1} + (1-\beta_2) * g_t{}^2$
4. $\widehat{m}_t = \frac{m_t}{(1-\beta_1^t)}$
5. $\hat{v}_t = \frac{v_t}{(1-\beta_2^t)}$
6. $\theta_t = \theta_{t-1} - \alpha * \left(\frac{\widehat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}\right)$
end while
return $\theta_t$
```

Adam basically a combination of AdaGrad and RMSProp. The purpose of these two algorithms combined are AdaGrad should not overshoot the global minima that is the purpose of the RMSProp and the purpose of AdaGrad is the step size should adjust itself accordingly, when it is near and far from the minima. Adagrad is a Adaptive gradient optimizer which uses the first order derivative or the normal derivative g_t. In case of RMSProp it uses the square derivative g_t^2. The above algorithm you see is taken from Adam research paper.

m_t is the exponential moving averages of the gradient.

v_t is the exponential moving averages of the squared gradient.

The default value of β_1 and β_2 is recommended by 0.9 and 0.999 respectively.

The value ϵ and α is recommended as 10^{-8} and 0.0001 respectively.

The meaning of Step 2 is at any give time t, we are giving more weightage to the recent momentum and less weightage to the derivatives of momentum before that, Same logic goes for Step 3 i.e here we are giving more weightage to v_{t-1} and less weightage to the squared derivatives of the momentum before that.

The main concept of Step 2 and Step 3 lies in kind of smoothing or kind of taken into consideration like what is the next step to be taken. Once we have computed m_t and v_t , then we compute $\widehat{m}_t$ and $\hat{\mathrm{v}}_t$. Now these two terms are known as bias adjustment terms.

We do this bias adjustment in Step 4 and Step 5 because we have seen in the Step 1 only m_0 and v_0 is initialized to zero at time t=0. Hence m_0 and v_0 are biased towards zero, in order to control these bias terms we use the refine term $\widehat{m}_t$ and $\hat{\mathrm{v}}_t$. So, $\widehat{m}_t$ and $\hat{\mathrm{v}}_t$ reduces the bias by using Step 4 and Step 5 respectively. Step 5 is nothing but normal weight updating step. So, in this step $\hat{\mathrm{v}}_t$ is there in the denominator, v_t is the RMSProp term means we are taking square derivation into consideration. So, when $\hat{\mathrm{v}}_t$ is large then this entire $\left(\frac{\widehat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}\right)$ value will be less provided we have taken m_t constant, then there will be a lesser shift in the denominator, subsequently α will get multiplied with that the weight will get adjusted.

Adam optimizer optimizes the function fast and in many research papers it is observed that this optimizer works very well.

CHAPTER 3

Simulation Results

In this Chapter we will focus on the results which we have got by using both the algorithms which we have discussed in section 2.2 on the same collected data i.e the data which we have got by performing the different gestures on the Robotic Arm by using the IMU MPU6050. So, in this Chapter we will create two sections i.e section 3.1 and section 3.2. Section 3.1 will focus on the results which we've got by using the KNN and Section 3.2 will focus on the results which we've got by using the Neural Network model.

3.1 Results by using KNN

Amount of data belongs to each gesture:

As we know that we have considered the number of gestures are five which we have discussed in Chapter 2. So, first we see what amount of data we've got for five different gestures i.e the data we were collecting while performing each of the five gestures 100 times. The variation you see in the amount of data for different gestures because some of the gestures take less time to complete a single instance. The bar plot you can see below shows the amount of data belongs to each gesture:

```
Data generated for each gesture
<matplotlib.axes._subplots.AxesSubplot at 0x7f96b2c80f90>
```

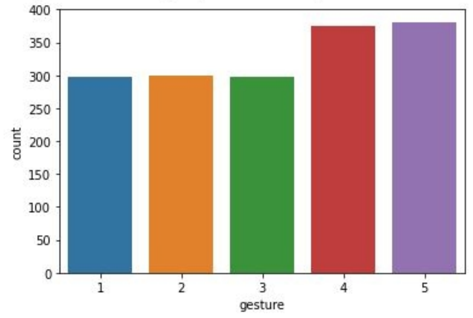
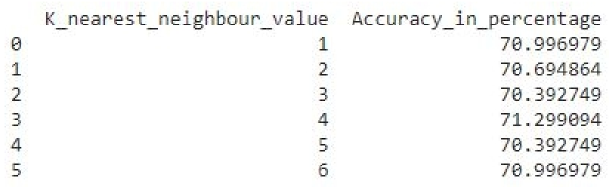


Figure 3.1: Amount of data in five gestures

Test Accuracy:

As we know from Chapter 2 that the model representation for KNN is the entire training data set. In other word we can say that KNN has no model other than storing the entire data set, So there is no learning required. KNN makes the predictions using the training data set directly.

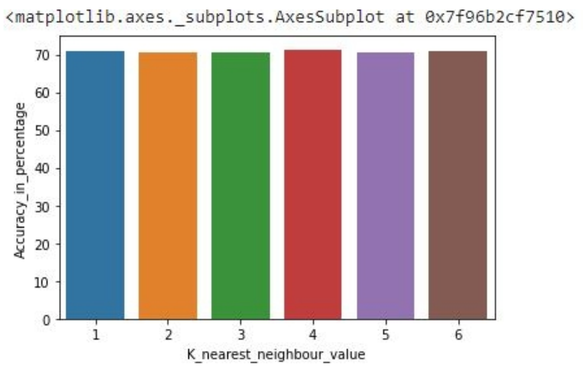
In our work we have considered K values from 1 to 6 i.e we varied the neighbor values from one neighbor to six neighbors and then we evaluated the corresponding accuracy for all K values. So, table you see below is the accuracy which we have got for all K values are:



```
   K_nearest_neighbour_value  Accuracy_in_percentage
0                          1               70.996979
1                          2               70.694864
2                          3               70.392749
3                          4               71.299094
4                          5               70.392749
5                          6               70.996979
```

Figure 3.2: Accuracy for different K values in tabular form

The bar plot of accuracy in percentage corresponding to each K is given below:



```
<matplotlib.axes._subplots.AxesSubplot at 0x7f96b2cf7510>
```

Figure 3.3: Accuracy for different K values in plotted form

So, from KNN algorithm we have got the highest test accuracy which is **71.29 %** that too for K=4, which is not so good accuracy that is why we need to explore some other model which gives better accuracy compared to that of KNN. To improve the accuracy we have considered Fully Connected Feed-Forward Neural Network model aka Multi Layer Perceptron with Adam Learning algorithm, which we have discussed in Chapter 2.

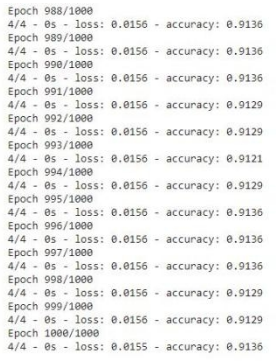
3.2 Results by using Feed-Forward Neural Network

Here in this model unlike KNN we have to train the model first by using the training data and then we used testing data to make predictions. So, here the model is Fully Connected Feed-Forward Neural Network model and we have trained this model by using Adam optimization algorithm and for evaluation of loss we have used Huber Loss which we have described in the Chapter 2. Therefore here we will have two accuracy i.e training accuracy and test accuracy. Training accuracy we have while training the model and it shows how good you have trained your model and this training accuracy we have got because of training data whereas Test accuracy or we can say it as real time accuracy is the accuracy which we have got while predicting the model i.e how good the model predicted the class value based on the new data i.e when data is from test data set. Since we know only training data is known to the model not the testing data, that is why this testing accuracy is also known as real time accuracy.

Training Accuracy:

Here for training we have considered 1000 epochs and 350 batch size. Thousand epochs means the model goes through the data 1000 times or we can call epochs as iterations as well. Batch size is a hyper parameter of the learning algorithm that controls the number of training samples to work through before the model's internal parameters are updated.

The below table shows fewer epochs(final epochs) that is at last at what accuracy or how good our model is trained:



```
Epoch 988/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9136
Epoch 989/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9136
Epoch 990/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9136
Epoch 991/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9129
Epoch 992/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9129
Epoch 993/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9121
Epoch 994/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9129
Epoch 995/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9136
Epoch 996/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9136
Epoch 997/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9136
Epoch 998/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9129
Epoch 999/1000
4/4 - 0s - loss: 0.0156 - accuracy: 0.9129
Epoch 1000/1000
4/4 - 0s - loss: 0.0155 - accuracy: 0.9136
```

Figure 3.4: Train accuracy for final epochs

We have also plotted the graph of the model accuracy or we can training accuracy, where you have to multiply by 100 to know the accuracy in percentage. The model accuracy plot is shown below:

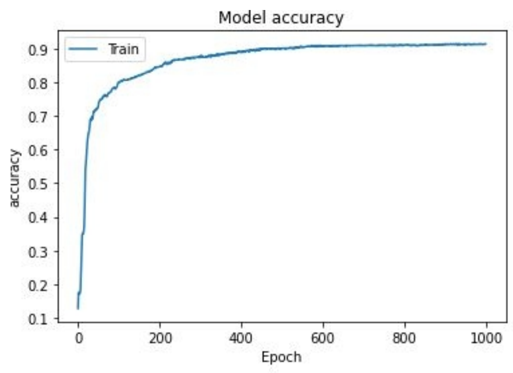


Figure 3.5: Model accuracy while training the model

We have also plotted the graph for the model loss as well, here also you have to multiply by 100 to the loss which is given in decimals in the plot to know the model loss in percentage, the plot is shown below:

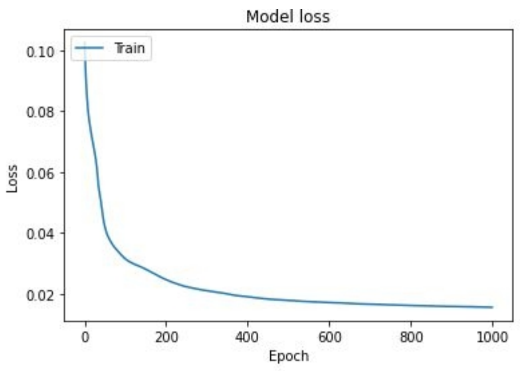
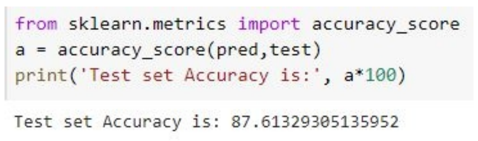


Figure 3.6: Model loss while training the model

<u>Testing Accuracy or Real time Accuracy</u>:

The accuracy which we have got by giving the new set of values that is the values from the testing data set to the model in order to check how good the model works in the real time. Here the test accuracy which we have got from the model is given below:

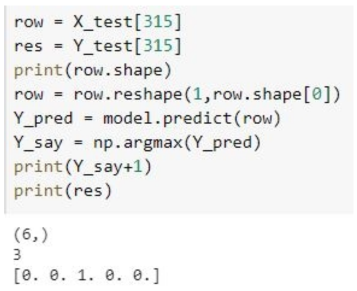


```
from sklearn.metrics import accuracy_score
a = accuracy_score(pred,test)
print('Test set Accuracy is:', a*100)
```

```
Test set Accuracy is: 87.61329305135952
```

Figure 3.7: Real time accuracy

You can also see how good this model predict by the given result which we have got for a single test data value. The picture of the prediction you can see below:



```
row = X_test[315]
res = Y_test[315]
print(row.shape)
row = row.reshape(1,row.shape[0])
Y_pred = model.predict(row)
Y_say = np.argmax(Y_pred)
print(Y_say+1)
print(res)
```

```
(6,)
3
[0. 0. 1. 0. 0.]
```

Figure 3.8: Sample of prediction.

As you can see in the Figure 3.8 it is shown that our actual value is which is 3 is equal to the predicted value. Here, we have used One-Hot Encoder to standardize the class value which initial was in words i.e one, two ,three etc. As you can see from the above diagram as well that the encoded value of three is [0. 0. 1. 0. 0.] by using this hot encoder.

So, as we have seen from this Feed-Forward Neural Network Model we have got a better accuracy i.e **87.613%**, the accuracy I mean testing accuracy, which is for single testing data set but the overall accuracy varies from **85-90%** which is better compared to that of KNN. You can observe our training accuracy as well that is **91.36%** which is good.

So, overall we can say that as expected our Neural network model is far better as compared to that of KNN and one should choose or use Neural Network model to get good real time accuracy.

CHAPTER 4

Future Work and Conclusion

This chapter discusses what one can do in future having that much of results i.e the result which we have got and finally concludes the work which we have done here.

Future Work:

Much of this work has focused, on the simulation part that is here we have trained the model based on the data we have got from sensor for different gestures. Now, one can proceed further from this work by implementing the algorithms which we have discussed in our work on the hardware device in order to take advantage in real time problems. The real time problem for instance it can helpful for the people who can not able to communicate via voice to other person. For those persons one can make the device by using of Arduino, BLE, MPU 6050, some wearable type hardware in which one would be able to fit all these hardware and then simply burn the algorithm on the Arduino that we have discussed in our work, we found out that Neural Network model gives far better accuracy compared to that of KNN. So, one can rule out KNN if someone want to makes device which aims for predictions in real time.

Conclusion:

The Data was generated by the robotic arm by placing a sensor MPU6050 over the device. Initially, Data cleaning process was carried out in order to find out NULL values and removed if any, in the collected sensor data. In order to visualize the data t-SNE was implemented. So, the main objective was to find out the best algorithm which would give us the better accuracy. The entire above mentioned algorithm is coded in python. The some of the gestures are somewhat similar to other, by taking all the gestures as different from one another we can even improve our accuracy. In this work we have seen the Neural Network model with Adam optimizer gives us the better accuracy to predict the above classification problem and it's accuracy which we have got is 85-90%.

REFERENCES

1. **Diedrick P. Kingma, Jimmy Ba**. Adam: A Method For Stochastic Optimization.
2. **Rafiqul Zaman Khan, Noor Adnan Ibraheem**. COMPARATIVE STUDY OF HAND GESTURE RECOGNITION SYSTEM.
3. **Paulo Trigueiros, Fernando Ribeiro, Luis Paulo Reis**. A comparision of machine learning algorithms applied to hand gesture recognition.
4. **Sungtae Shin, Han UI Yoon, Byungseok Yoo.** Hand Gesture Recognition Using EGaIn- Silicone Soft Sensors.
5. **Monu Verma, Ayushi Gupta, Santosh K Vipparthi.** One For All: An End to End Compact Solution For Hand Gesture Recognition.
6. **Hao Tang, Hong Liu, Wei Xiao, Nicu Sebe**. Fast and Robust Dynamic Hand Gesture Recognition via Key Frames Extraction and Feature Fusion.
7. **Minwoo Kim, Jaechan Cho, Seongjoo Lee, Yunho Jung**. IMU Sensor-Based Hand Gesture Recognition for Human-Machine Interface.
8. **Yen-Cheng Chu, Yun-Jie Jhang, Tsung-Ming Tai, Wen-Jyi Hwang**. Recognition of Hand Gesture Sequences by Accelerometers and Gyroscopes.
9. **Gupta, H.P., Chudgar, H.S.,Mukherjee, S., Dutta, T., Sharma, K.** A continuous hand gestures recognition technique for human-machine interaction using accelerometer and gyroscope sensors.