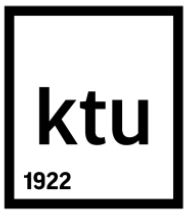


KAUNAS UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATICS

Rahul Raj Devaraja

Autonomous Interface for Robust Stimulus Applications (AISRA)

Master‘s Degree Final Project

Supervisor
prof. dr. Rytis Maskeliūnas

KAUNAS, 2018

KAUNAS UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATICS

Autonomous Interface for Robust Stimulus Applications (AISRA)

Master's Degree Final Project

Master's in Informatics (P000M106)

Supervisor
(parašas) Assoc. prof. dr. Rytis Maskeliūnas
(date)

Reviewer
(parašas) Assoc. prof. dr. Armantas Ostreika
(date)

Project made by
(parašas) Rahul Raj Devaraja
(date)

KAUNAS, 2018

KAUNAS UNIVERSITY OF TECHNOLOGY

Faculty of Informatics

(Faculty)

Rahul Raj Devaraja

(Student's name, surname)

Master's in Informatics (P000M106)

(Title and code of study programme)

"Title of Final Project"

DECLARATION OF ACADEMIC INTEGRITY

2018 June

Kaunas

I confirm that the final project of mine, **Rahul Raj Devaraja**, on the subject "**Autonomous Interface for Robust Stimulus Applications**" is written completely by myself; all the provided data and research results are correct and have been obtained honestly. None of the parts of this thesis have been plagiarized from any printed, Internet-based or otherwise recorded sources. All direct and indirect quotations from external resources are indicated in the list of references. No monetary funds (unless required by law) have been paid to anyone for any contribution to this thesis.

I fully and completely understand that any discovery of any manifestations/case/facts of dishonesty inevitably results in me incurring a penalty according to the procedure(s) effective at Kaunas University of Technology.

(name and surname filled in by hand) *(signature)*

Rahul Raj Devaraja. Autonomous Interface for Stimulus Robust Applications. *Master's Final* Degree Project/supervisor Assoc. prof. dr.Rytis Maskeliūnas Faculty of Multimedia Engineering, Kaunas University of Technology.

Study field and area (study field group): Robotics, Neural networks

Keywords: Obstacle avoidance, LDS base navigation, autonomous navigation

Kaunas, 2018. 74 pages.

SUMMARY

The work deals with the design of an autonomous interface using Neural Network that helps a mobile robot to navigate in space autonomously. This paper presents a unique design for a autonomous interface using Behavioral Cloning for the process of training the robotic-system, to teach the robot how to navigate in space and avoid obstacles. Designing of AISRA was an end to end process, this document holds the details of data accumulation, preprocessing of data, neural network architecture and inferencing. Unlike the existing camera-mounted stimulated model, AISRA design is based on the laser scan information and leveraging the power of neural networks. The analysis study is about analyzing laser scan information for input to the system augmented with Neural Network instead of the existent camera-mounted stimulated models.

In a traditional behavioral cloned system, training data set fed to the system will contain recorded decisions a human will make to turn the steering wheel to overcome the obstacles ahead. So, a study is conducted using a Robot Operating System based mobile robot Turtlebot 3 Burger, to run in a real-time environment accumulate to gather the data. Turtlebot 3 comes with a Laser Distance Sensor that when used for SLAM operations which helps us to gather Laser Distance Range data for 360-degree circumference. The system contains Keras deep learning library used for architecting the neural network and evaluating its performance. The final trained model is transferred to Nvidia's Jetson to inference the data in real time.

SANTRAUKA

Darbe nagrinėjama autonominio roboto valdymo sąsaja, naudojant neuroninį tinklą, kuris padeda mobiliam robotui autonomiškai naviguoti vidaus erdvėse. Algoritmas skirtas išmokyti robotą judėti erdvėje ir išvengti kliūčių. Ataskaita pateikiama duomenų kaupimo, duomenų išankstinio apdorojimo aprašai, neuroninių tinklų architektūros ir eksperimentų rezultatai. Skirtingai nuo tipinio modelio, AISRA dizainas paremtas lazerinio erdvės skenavimo informacija ir neuroninių tinklų galios sverto efektu. Sistema analizuoja lazerio nuskaitymo informaciją, klasifikuoja įvestį neuroniniu tinklu, taip formuojant kliūčių vengimo valdymo algoritmą. Tyrimas atliekamas naudojant robotų operacinę sistemą (ROS), naudojant "Turtlebot 3" "Burger" robotą, siekiant kaupti duomenis realiuoju laiku. Turtlebot 3 yra komplektuojamas su lazeriu atstumo jutikliu, naudojamu SLAM operacijoms ir skirtu rinkti duomenis apie 360 laipsnių perimetrą. Sistema paremta "Keras" giliojo mokymosi biblioteka. Galutinis apmokytas modelis perduodamas "Netscape" "Jetson" platformai, kurioje yra apdorojami realiu laiku.

TABLE OF CONTENTS

ABBREVIATIONS AND NOTATIONS

CNN: Convolution Neural Network
ANN: Artificial Neural Network
ROS: Robot Operating System
CSV: Comma Separated Values

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Autonomous Interface for Robust Stimulus Applications - AISRA

Autonomous behavior is rapidly increasing in every engineering discipline. Machine learning and AI is benchmarking its potential in all aspects, right from modeling a pen tip till exploring Mars and beyond. Machine Learning has done wonders in automating various functionalities that were considered complex and nearly impossible only a few decades ago.

During this period of the shift from manual process to automation, bringing intelligence to embedded devices is growing exponentially. Autonomous vehicles are one of the significant areas of research and essential inventions which gradually increased from a decade.

Today there are self-driving cars, flying drones, mobile robots, and rockets. Autonomous behavior is a term that is used to refer unmanned vehicles that can make decisions on its own decision-making ability or through a method of decision making pre-programmed into it. For example: A self-driving car will typically have three cameras- left, right and center camera. These cameras will snapshot the images of road that will be used to train the neural network. Also, image processing requires high-end computational hardware such as mutli-core processors that have GPU enabled for real-time image processing.

In the recent times, the research area of the autonomous vehicles has certainly got the mainstream boost with governments and multinational companies showing their interest to invest in the field. However, since its inception, the concept of unmanned vehicles is majorly dependent on the camera-mounted models. The currently existing camera-mounted models with no doubt have achieved to address the needs of primary input that the neural network model needs and have achieved considerable accuracy in obstacle avoidance. However, using image processing for primary input needs will always be a challenge in situations where camera fails to capture necessary images for navigation, or in dim-lit areas, or in spaces where it is required to navigate without camera input. In such circumstances, the existing models fail to obtain their purposes.

1.1. Aim:

The aim of this research is to determine the possibilities of obstacle avoidance and navigating autonomously using just laser sensors

a) To create an end to end system to model the object avoidance system which can be deployed to any robot which has an LDS sensor
b) Investigate the inference mechanism with ROS and development embedded board called Jetson

My proposal to contribute to this area of research brings me to design an interface to which a mobile robot can autonomously navigate in a space based on the laser scan information and leveraging the power of neural networks. Whereas, Lidar based system can run on simple ARMs 64 bit processor such as Raspberry Pi also.

I have used Behavioral cloning as the process of training the system to teach it how to navigate in space and avoid obstacles, designing AISRA was an end to end process, this document holds the details of data accumulation, preprocessing, neural network architecture and inferencing.

We used ROS based mobile robot Turtlebot 3 Burger, to run in a real-time environment accumulate to gather the data. Turtlebot 3 comes with an LDS scanner that when used for SLAM operations helps us to gather Laser Distance Range data for 360-degree circumference. Thanks to Keras, deep learning library which made our lives easier in architecting the neural network and evaluating its performance.

1.2. Document structure:

This paper consists of three chapters, references and appendices. The first chapter will introduce to the concepts and overall idea of the proposal. Followed by the chapter which demonstrated how the data is collected, preprocessed, rendered to neural network and explains the inference with ROS and Nvidia's Jetson

2. LITERATURE SURVEY

2.1. Behavioral Cloning:

According to the Encyclopedia of Machine Learning, Behavioral cloning [6] is a method of monitoring human cognitive skills and remapping into computer program. As the human subject performs the skill, his or her actions are recorded along with the situation that gave rise to the action. A log of these records is used as input to a learning program. [6]. The learning agent tries to reproduce with certain behavioral rules to mimic human. Behavioral cloning can be used to automate certain complex tasks where human intervention would be needed [6].

Behavioral cloning can be termed as training a system to mimic a human behavior. For E.g.: In a self-driving car a human will drive the car around for long hours to capture the images of road ahead and record the decisions taken by the driver when it comes across obstacles. This data is then pre-processed and fed into the system to train the system to make decisions on its own.

The existing models of behavioral cloning mostly uses a camera-mounted stimulated model that will help capture the input data required to train the system. The decisions taken by human driver to turn the steering wheel such as the steering angle when an obstacle is encountered will also be recorded to train the model. Once the model is trained, the network will be able to generate steering commands based of the video images from the cameras.

Autonomous Behavior: Autonomous behavior is a term in reference to self-driving cars and autonomous vehicles. A system that is truly autonomous in behavior might look feasible in theory but is indeed complicated to construct and train in with datasets that will account for all the different known situations and certain unknown situations that it will encounter during its life cycle. Autonomous behavior brings in a certain amount of conscious to a system that lets the machine make its own decisions without any human intervention [6].

Autonomous behavior relies on adequately training the system with exhibiting the data of the very behavior the system should mimic. So, it is crucial to spend adequate time to pre-process the data and randomize it as much as possible to train the system for better decision-making abilities. Exploring the data to train the model on and adding relatively few constraints will surely help in avoiding pitfalls and dead-ends but also give insights to build better models and strategies [6].

2.2 Neural Networks

An ANN is a network architecture which is stimulated as a human brain to the content and information. The critical notion of this paradigm is a complex architecture to process information[5]. This architecture is composed of a larger number of sophistically connected neurons to solve specific problems

Most conventional computing systems use an algorithmic approach to solve problems. The use of algorithmic approach means the conventional computers will follow a set of instructions to solve a problem. Conversely, if the specific set of instructions are not available, the computer systems cannot come to any conclusion in solving problems. That restricts the computers to solve problems that we already understand and know how to solve. The conventional computing systems cannot extend their problem-solving capabilities beyond the cognitive approach where the way the problem is to be solved must be known and broken down into small unambiguous steps. These instructions primarily converted into a high-level language program and then to machine-code will be fed to the systems to process. [5]

Neural networks will provide the power and ability for computing systems to tap into the unknown dimension of conventional computing which allows us to solve problems that are not entirely understood or to the problems which do not have a ready-made solution. Neural networks approach the problem and process information very similar to that of a human brain. Neural networks are computers modeled based on the human brain and nervous system. Hence, the neural networks will learn and acquire decision-making capabilities as a human does that is by reaching rational decisions based on prior experience and continuously learning and adapting to the new situations the system comes across. Investigation and similarities will help the neural networks in decision making. [5]

2.3 Architecture of Neural Networks

The architecture of neural networks can be broken down into the following essential elements:

Feed-forward networks: As the name suggests the signals are feed forward [26] in these networks. The signals in feed-forward ANN's travel unidirectionally. The signals travel only from the input layer to output layer. There are no feedbacks(loops) to the layers which means that the output of any layer will not affect the same layer. Hence, there are also called top-down or bottom-up networks. Feed-forward networks are used mainly in pattern recognition. [5]

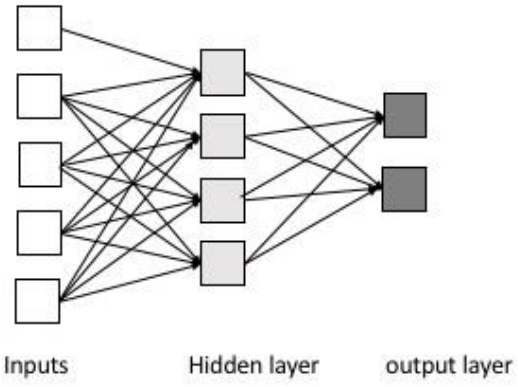


Figure 1 Example of feed-forward neural network[5]

Feedback networks: Unlike feed-forward networks, the signals can travel bi-directionally in feedback networks. Feedback network scan have signals traveling both the side by introducing loops into the network. Having loops in the networks makes them exceedingly powerful and can also get significantly complicated. The state of feedback networks is dynamic, i.e., [5]their states keep continuously changing until they reach an equilibrium point. This state remains unchanged until the input changes and requires calculation of a new equilibrium point. Feedback networks are also called interactive or recurrent networks. Recurrent networks are used to denote feedback connections in single layer organizations. [5]

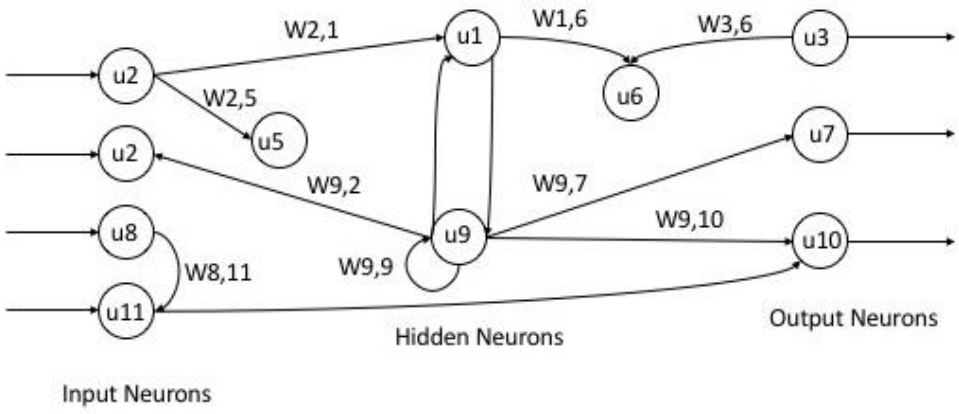


Figure 2 Example of feedback network

Network Layer: Every neural network consists of three basic units: the input unit, the hidden unit, and an output unit. So, typically a neural network will contain a layer of input units connected to a layer of hidden units, which is [5]connected to a layer of output units.

- Input neurons processes the raw input in to the model
- Hidden layer is the sum of weights between the hidden units and the input nodes
- The output layer depends on the summed weights between the nodes of the hidden layer with the nodes of the output layer

The simple network comprises of single layer of neural network, the hidden layer makes the neural network complex in architecture. The weight between the input layer and hidden layer denotes which nodes of the hidden layer is active

Neural network comprises of single layer neural network and multilayer neural network. Single layer network, in which all nodes are interconnected. Multilayer networks, the nodes are numbers based on the depth of the layer instead of global numbering

Perceptron: The most prominent work on neural networks in the 60's spread beneath the headline of 'perceptron' a term coined by Frank Rosenblatt. The perceptron [5] Figure 3 transposes to be an MCP model (neuron with weighted inputs) with some additional, fixed, pre-processing. The units labeled A1, A2, Aj, and Ap are called association units, and their task is to extract specific, localized featured from the input images. Perceptron mimic the fundamental concept behind the human visual system. They were chiefly used in pattern recognition even though their capabilities stretched much further.

In 1969 Minsky and Paper wrote a book in which they outlined the limitations of single layer Perceptrons. It revealed mathematically that single layer perceptron [5] could not do some basic pattern recognition operations like determining the parity of shape or determining whether a shape is connected or not. It was in the 80's that researchers apprehended, that given the appropriate training, multilevel perceptron can do these operations.

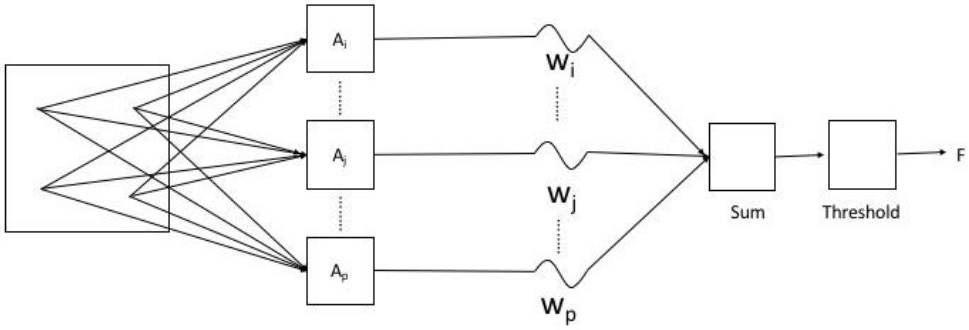


Figure 3 The perceptron

The Learning Process

The process of pattern recognition and the consequent response of the network is categorized into two general archetypes: associative mapping and regularity detection [5].

Associative Mapping is the process of analyzing particular patterns on the set of inputs passed through the input layer against the already existing patterns in the input nodes. The associative mappings can be further divided into categories

- **Auto-association**: Auto-association [5] is the mechanism where an input pattern is associated with itself and the states of input and output units coincide. The purpose of auto association is to render the pattern in the network, that is, to output a pattern when a similar match of input data is found. In the second case, the network is responsible for saving the pairs of patterns exhibiting the associations between them
- **Hetero-association**: Hetero-association [5] is related to two recall mechanisms:
    - **Nearest-neighbor recall**: Outputs of the network are related to the patterns found in the input node which is stored, which is the closest represented pattern
    - **Interpolative recall:** Output patterns has a similarity dependent interpolation of the patterns saved with respect to pattern presented. Another perspective of interpolative cell, which is a form of associative mappings responsible for classification, that is only when defined number of categories for which each input can be classified [5]

Regularity Detection is the process in which units learn to respond to distinct attributes of the input patterns. Whereas in associative mapping [5]the network collects the relationships among patterns, in the sequence of regularity detection, every response of the particular unit has a definition This workflow od learning process is much essential and required attribute for feature discovery and knowledge representation

In every layer of the network, knowledge possessed between the nodes which is transmitted are in the form of weights. [26] Modifying the knowledge which is saved in the network as a function of learning by experience implies the rule for its process of changing its weights.

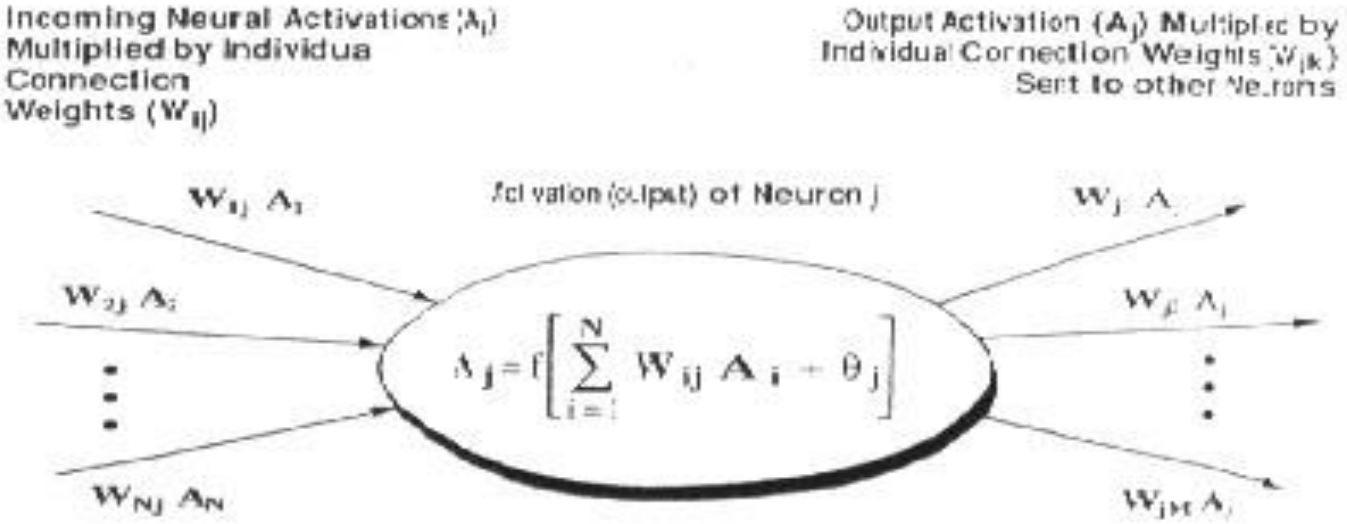


Figure 4 Regularity Detection Weight Matrix [5]

Information stored in the weight matrix W of a neural network. Learning is the determination of the weights. [5]Based on the learning performed, we can distinguish two major categories of neural networks:

- **Fixed networks** in which the weights cannot change, i.e. $\frac{dW}{dt} = 0$ In such networks, the weights are fixed a priority according to the problem to solve.
- **Adaptive networks** which can change their weights, i.e. $\frac{dW}{dt}$ $not\ equal\ to\ 0$

The classification of all learning methods used for adaptive neural networks can be narrowed down to two major categories:

Supervised learning: Supervised learning [5]incorporates an external teacher to instruct each output units what its desired response to input signals ought to be. There might be a requirement of global information during the learning process. Perspectives of supervised learning comprises of minimizing the error, reinforcement learning and stochastic learning.

An important issue concerning supervised learning is the problem of error convergence, i.e., the minimization of error between the desired and computed unit values. The aim of the supervised learning is to minimize the error between the predicted values to the actual values using the methods of optimization. The most popular ones are Least Mean Square and Root Mean Square

Unsupervised learning: Unsupervised learning is a form of learning in which it has no reference or to the output variables. [5] So basically, it has no teacher to train its network to predict the output values. It is self-organized among itself in finding clusters and patterns. Unsupervised learning is totally a competitive learning. Usually supervised learning uses offline method of training their data, whereas unsupervised learning has [5] to be open to all the incoming data to recognize the patterns, so the training usually is made online

2.4 Multi-layer Perceptron

Perceptron [5]is a relative term coined in the 50s that indicated how the future perception of machine learning looked back then. Perceptron is a simple algorithm that performs binary classification operation which predicts if the inputs belong to a particular category or not. The perceptron endures a unique spot in the history of neural networks and artificial intelligence. Because the initial hype about its performance led to a rebuttal by Minsky and Papert, and widespread resentment that cast a cloak on neural network research for decades, a neural net winter that wholly thawed only with Geoff Hinton's research in the 2000s, the results of which have since cleared the machine-learning community.

A perceptron [5]is a linear classifier algorithm. This algorithm linearly separates the data points in to two classes with a straight line. Input is typically a feature vector x multiplied by weights w and added to a bias b: y = w * x + b. A perceptron will produce single output for the linear classifier based on the real number input by forming a linear combination of weights. Mathematically its written as shown in eq 1.

$$y = \varphi(\sum_{i=1}^{n} w_i + b) = \varphi(w^T x + b) \quad (1)$$

where w denotes the vector of weights, x is the vector of inputs, b is the bias and φ is the non-linear activation function.

The original perceptron built by Rosenblatt was a single-layer perceptron. Rosenblatt's hardware-algorithm [5]was a shallow neural network because it did not include multiple layers, which allow neural networks to model a feature hierarchy. Single-layer perceptron prevented his perceptron from performing non-linear classification, such as the X-OR as Minsky and Paper showed in their book. [5]

A multilayer perceptron (MLP) [5]is a deep learning, ANN. It is packed up with more than one perceptron. The architecture of MLP is same as a perceptron which has an input layer, the difference which makes is the stacking of hidden layers. Then comes the output layer which makes the prediction based on the input received. MLPs with one hidden layer are capable of approximating any continuous function because of its hidden layers.

A multilayer perceptron (MLP) [5] is a sub-category of feedforward neural networks. A multilayer perceptron comprises a minimum of three layers of nodes: input node, hidden node, an output node. Apart

from the input nodes, every node in the neural network uses a nonlinear activation function. A multilayer perceptron employs a supervised learning algorithm procedure called backpropagation for training the network. Linear Perceptron uses direct activation whereas multilayer perceptron has multiple layers and uses non-linear activation. A multilayer perceptron can recognize data that is non-linearly separable. Multilayer perceptron's with a single hidden layer are seldom vaguely attributed to as "vanilla" neural networks.

Layers: A multilayer perceptron typically comprises three minimum layers: an input and an output layer including one or more hidden layers. An MLP is termed a deep neural network because of these layers of nonlinearly-activating nodes. Considering multilayer perceptron's are wholly connected, every node in individual layer connects with a particular weight to each node in the subsequent layer [5].

Activation Function: When a linear activation function is included in each node of a multilayer perceptron, that is, the weighted inputs are mapped to the output of each node through a linear function. Every layer can be compressed into a two-layer input-output model with linear algebra. Certain nodes in multilayer perceptron's make use of a nonlinear activation function. [14]Nonlinear activation function was developed to illustrate the recurrence frequency of action potentials of the nodes or triggering of neural network nodes. The activation functions are two sigmoid which are described by:

$$y(v_i) = \tanh(v_i)\ and\ y(v_i) = (1 + e^{-vi})\text{-}1 \quad (2)$$

Where the first function is a hyperbolic tangent that ranges from -1 to 1, while the other is the logistic function that ranges from 0 to 1, y_i is the is the output of the i^{th} node (neuron), and v_i is the weighted sum of the input connections. [14]

A model which uses a linear function (i.e., a model with no activation function) is impactful only for a single layer and will be unable to make sense of complicated data, such as, speech, videos, and so on. An activation function is used to introduce non-linearity into a network. Thus, it enables to model a class label that varies non-linearly with independent variables. Any linear combination of inputs cannot replicate the output resulted from the non-linearity. This allows the model to learn complex mappings from the available data, and thus the network becomes a universal approximator. [14]

The most extensively utilized neural network activation functions are as follows:

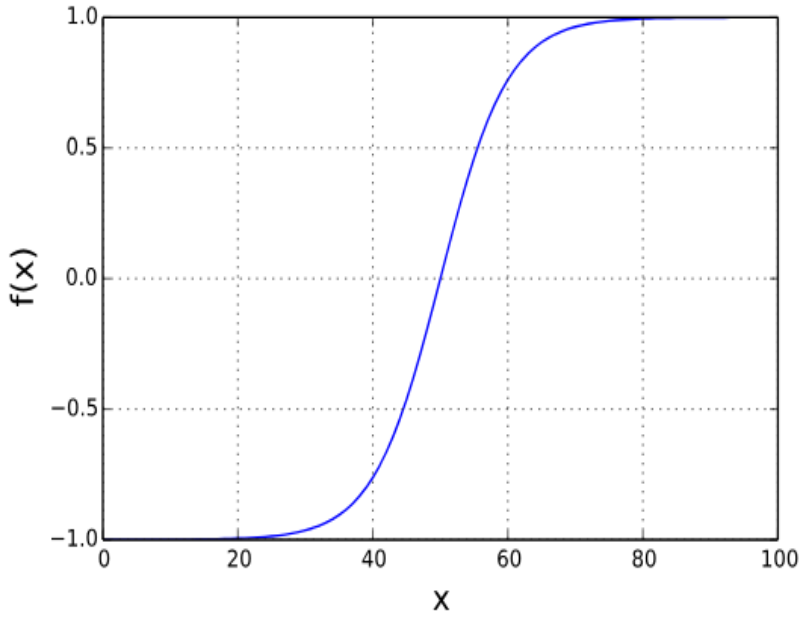
- **Tanh:** Tanh is a non-linearity activation function which maps the output to zero centered value by taking the input and converting its form ranging from negative one to positive one

$$f(x) = \tanh(x) = \frac{2}{1+e^{2x}} - 1 \quad (3)$$

Furthermore, the gradients for tanh are steeper than sigmoid, but it suffers from the vanishing gradient problem. Tanh is commonly referred to as the scaled version of the sigmoid. Generally, this equation holds: [14]

$$\tanh(x) = 2sigmoid(2x) - 1 \quad (3)$$

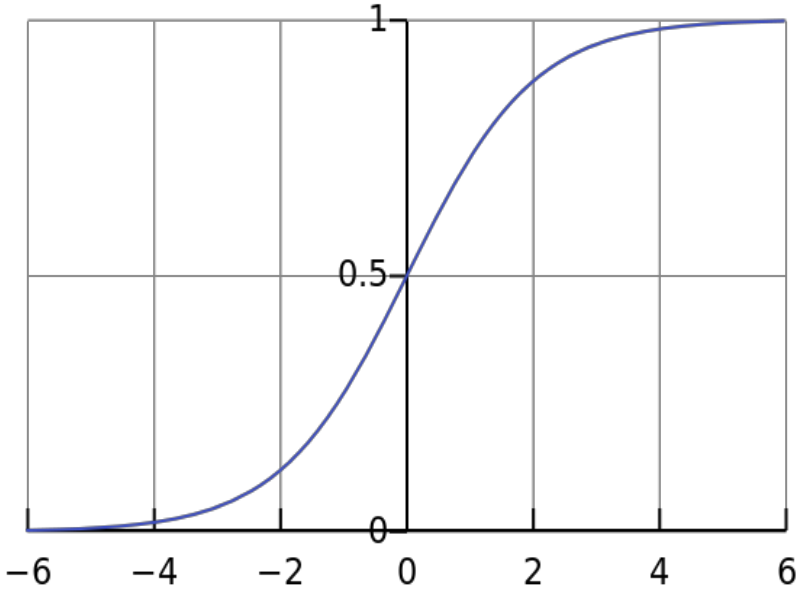
The gradient is stronger for tanh than sigmoid (derivatives are steeper). Deciding between the sigmoid or tanh will depend on your requirement of gradient strength.



- **Sigmoid:** Sigmoid activation is majorly used for multi-class classification. The sigmoid or logistic activation function maps the input values in the range (0,1), which is often their probability of belonging to a class. The output it produces is not zero-centered, which causes difficulties during optimization. It also has a low convergence rate. However, like tanh, it also suffers from the vanishing gradient problem. [14]

$$A = \frac{1}{1 + e^{-x}}$$

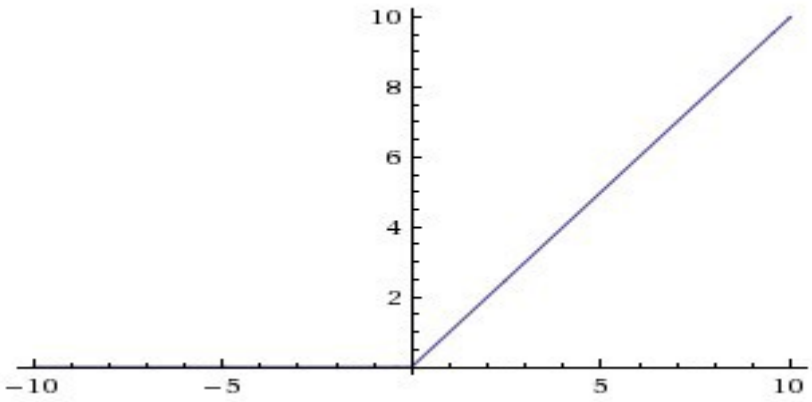
In the following graph of sigmoid function, we can notice that the gradient has almost disappeared towards the either end. This indicates that at these points the network refuses to learn further or is drastically slow (depending on use case and until gradient /computation gets hit by floating point value limits).



- **ReLU:** ReLU is extensively used in convolutional neural networks. ReLU or the rectified linear unit has the output 0 if its input is less than or equal to 0; otherwise, its output is equal to its input. It is biologically accurate, and superior to the sigmoid and tanh activation function, because it does not suffer from the vanishing gradient problem. Thus, it allows for faster and effective training of deep neural architectures. [14]

$$A(x) = Max(0, x) \tag{5}$$

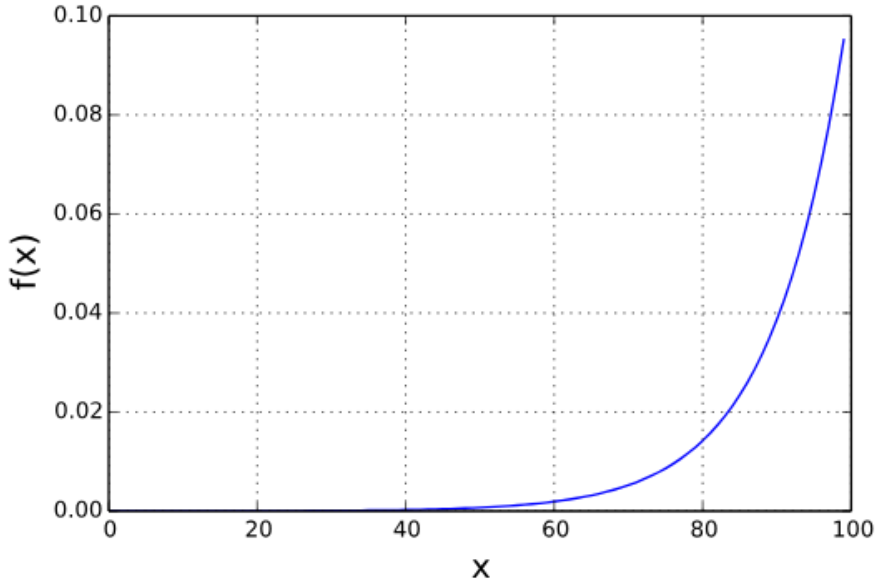
However, being non-differentiable at 0, ReLU neurons tend to become inactive for all inputs. This behavior is caused due to high learning rates and can thus reduce the model's learning capacity. Hence, this behavior is referred to as the "Dying ReLU" problem. [14]



- **SoftMax:** SoftMax function is widely used in multiple classification logistic regression models. The SoftMax function's output indicates the probabilities that any of the [14] classes are true, so it produces values in the range (0,1). It highlights the largest values and tries to suppress values which are below the maximum value, and its resulting values always sum to

$$\sigma(x_j) = e^{xj}/\sum_i e^x i \quad (6)$$

From the SoftMax activation graph, it can be comprehended that the sigmoid function value will continue to rise towards the peak score of 1 with the rise in the input values. In the graph, the values in the range of 0.9 to 0.99, which are the peak most values are meeting at the tip. [14]



2.5 Back-propagation Algorithm:

In order to train the network. The model has to correct its weights to prioritize the nodes for next recursive operation. In such way the error between the predicted value and the actual value can be determined. In other way it must calculate the weights when the respective errors decrease or increases. This approach is basically known for error weight manipulation [5]

The process of learning in perceptron occurs by changing connection weights after every piece of data is processed, based on the measure of error in the output compared to the expected result. This instance proves the supervised learning is carried out through this process of backpropagation by root mean square in the linear perceptron [5]

The error in output node [5] represented as j in the n^{th} data point (training example) by $e_j(n) = d_j(n) - y_j(n)$ where d is the target value and y is the value produced by the perceptron. The node weights are adjusted based on corrections that minimize the error in the entire output in eq 6, given by:

$$\epsilon(n) = \frac{1}{2}\sum_j e_j^2(n) \quad (6)$$

By applying gradient descent algorithm to the above equation, the change in the weight of each unit can be calculated by the following equation 7:

$$\Delta w_{ji}(n) = -n\frac{\delta\varepsilon(n)}{\delta v_j(n)} y_i(n) \tag{7}$$

Where η is the rate of learning, (the rate of learning is chosen to guarantee that the participating weights immediately focalize into a response, without any oscillations) and y_i is the output of the previous node. The derivative is calculated based on the induced local field v_j which usually keeps varying. [5]
The above derivative for any output node can be reduced to:

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = e_j(n)\phi'(v_j(n)) \tag{8}$$

Where ϕ' is the above described activation function derivative (the activation function on its own does not oscillate). Estimating the error weight derivative for a hidden node is exceedingly tricky to analyze. The hidden node error derivative can be represented by the following simplified equation: [5]

$$-\frac{\partial \mathcal{E}(n)}{\partial v_j(n)} = \phi'(v_j(n)) \sum_k -\frac{\partial \mathcal{E}(n)}{\partial v_k(n)} w_{kj}(n) \tag{9}$$

The derivative is completely dependent on the variation in weights of the k^{th} nodes. These nodes denote the nodes of the neural network model's output layer. The weights of the output node layer vary with respect to the derivation of the output layers' activation function. This variation, in turn, changes the weights of the hidden layer nodes. Hence, this algorithm renders a backpropagation of the activation function.

When each node in the neural network model is a linear perceptron, the backpropagation algorithm can be understood placidly. In such cases, the algorithm will have to compute the derivative of error weights for every node. [5]To do so, the changes in the activity of each node with respect to the error weights of those nodes has to be calculated. The algorithm computes each error derivative of the weights(EW) by first computing the EA, the rate at which the error changes as the activity level of a unit changes. For output units, the EA is merely the difference between the actual and the desired output.

To compute the error approximation for the hidden nodes of the network which is stacked before the output layer, initially we need to figure out which nodes in the hidden layer is connected to output layer. Then we multiply those weights with EA's other layers in the network by backpropagating. For the hidden network node, the EA are summed up. Propagating between the layers in the opposite direction activates the weights through the whole network. Once the EA is computed for a node, no it's the same process of computing EA for future upcoming nodes in the network. EW is the multiple of EA and the activity for the future connections ahead [5]

As stated before, the backpropagation algorithm is straightforward to comprehend for networks to learn from when all nodes involved are linear. However, when the neural network models have nonlinear nodes, an additional step will be included in the backpropagation algorithm. That is, before back-propagating, the

EA [5]must be converted into the EI (the rate at which the error changes as the total input received by a unit is changed).

To make the neural networks predict the imminent decisions that have to be taken in order to achieve the functionality it was designed for, the neural networks have to be trained with the right set of data. Finding the right set of data for training the neural networks is always a tedious job. Moreover, even the network trained with the most accurate data will fall short in predicting accurate decisions in some unforeseen situations. So, while selecting the data set to train the neural network, it is of foremost importance to pick the [5]datasets in which the weights of units can be adjusted in the most optimal way to reduce the error in weights between the actual output the system predicts and the desired output that the system is expected to predict. Therefore, the neural network model should calculate how the error weight changes with a slight increase or decrease in the weights of each unit.

By employing backpropagation algorithm to measure the error weights, it is easy to train the system whether it be either supervised or semi-supervised learning. The connection weights are varied after each dataset is processed by the model based on the measure of the difference in error of the obtained output weights and the desired output weights. Once we get the error weight of the dataset unit, we calculate the error derivative of weights obtained by neural network model. In vague words, we can term backpropagation algorithm to be the generalization of the least mean square algorithm that is used in case of the linear perceptron. [5]

2.6 Robot Operating System

ROS is made up of couple of components such as Master, Parameter Server, ROS Services, ROS Topics, ROS Bags. [2]

- **Nodes:** Robot System comprises series of nodes which constitutes the architecture of ROS, Nodes are also called as the processes that perform the various tasks. For instance, LDS Sensor, which will be a node responsible of publishing the data, cmd_vel will be a node responsible for publishing velocities. [2]
- **Master:** ROS Master provides the lookup table for all the available nodes which are active. It forms a directory-based ledger to map all the services, nodes, topics, and handing messages between the nodes. ROS mater must start initially to activate all the nodes and create handshake property between nodes and messages published between them. ROS masters are started using roscore command [2]
- **Parameter Server:** Parameter server acts as dictionary to set all the required parameters between nodes exchange. It is currently part of a Master node [2]
- **Messages:** Nodes interact between two nodes in Publisher and Subscriber mechanism. Nodes communicate between messages. ROS Messages provides the data structures for data exchange.
- **Topics:** Messages are routed via a transport system with publisher subscriber mechanism. A node talk to publishers and subscribers between them using a channel. There may be any number of publishers over a topic and any number of subscribers. Single node publisher may also have many subscribers. In general, publishers and subscribers are not aware of each other existence. The idea is to decouple the production of information from its consumption. [2]
- **Services:** Publisher and Subscribers models has a flexible message exchange paradigm, its architecture supports 1 to 1, 1 to n or just one-way communication. Thou one-way communications doesn't support request and response cycle. Certain times in robots needs a perspective to work as a webserver which accepts requests and deliver responses. For instance. To instruct robot to click a picture of yours. ROS services also works as RPC when calling the function available in the other nodes
- **Bags:** Bags serves as containers to save all the data published by the robot. These bags are further used in debugging any algorithms built on top of ROS or inside ROS with having robot to publish data every time to test the algorithms [2]

ROS Master has a ROS Computation graph. The ROS master saves all the information about the parameters servers and tracks all the topics and delivers messages between the nodes. The nodes in the master communicate their registration information [2].

2.7 Existing workflows

2.7.1 Navigation Stack in ROS

ROS is a moderately complicated piece of middleware, at its heart, navigation stack of ROS is a system that enables the robot to navigate around the place without crashing into the obstacles.

The position of the robot in the space is called pose, which is represented by coordinates of the map by its orientation and position. The initial erudition navigation stack performs is the creation of a global cost map. [2] Global cost map indicates how good or bad the robot is positioned away from the collision on the map.

The global path is one of the paths the robot has to follow on its map, but it moves along with the local planner. The local planner assists the global path in balancing to avoid local obstacles. The global path supported by the local planner, which receives inputs that are not on the map based on the sensor inputs will balance avoiding local obstacles [2].

2.7.1.1 ROS Path finding algorithms:

Navigation in the mobile robots has to undergo sequence of maneuvers to move from the initial position to the goal position without crashing obstacles. There are two algorithms to find the optimal path planning included in ROS navigation stack [23]

1) Graph based navigation
2) Occupancy grid-based navigation

Graph based navigation: Navigation takes place in the form of graphs; each graph will have possible directions to move around. All the places are defined as vertices of the graph. So, during moving around places in the space, graphs basically take the traversing between the edges of the graph. Selecting the points to navigate in space depends on the weights between the edges of the graphs. Finding the shortest path between those graph vertices specifies the initial position and the destination position. [23]

There are various types of approaches in calculating trajectory, path planning, and path following. Search of optimal minimal path depends on the sum of weights of all edges in the graph from the robots initial position to the goal position. The standard optimal path finding algorithms based on graphs in robotics are A*, D* or Dijkstra algorithm. [23]

Occupancy Grid based Navigation: This method of navigation divides the space into map pixels or map grid. Once the map grid is taken into consideration, an algorithm will mark the cells which are free or occupied with an entity or an obstacle. Further, the map is marked with robot's initial position and goal position. So, the path planning will be trajectory that is based on finding the shortest distance without falling on any occupied cell.

Occupancy Grid in ROS: In ROS it is possible to move the robot around a path when the map is optained.so then the map is converted into occupancy grid. For example, using a slam gmapping tool to create the occupancy grid. There is an inbuilt path finding package in ROS called move_base which holds the move_base node which is responsible for navigation

This move_base node will be subscribed to the node which publishes the required velocities called cmd_vel. Once the robot receives the information about the velocities, it also further publishes the transformation information between robot's initial position.

move_base node creates an occupancy grid cost map. So, every cell in this grid map will be marked with an obstacle or an empty space in it and the cost is defined between these cells to the obstacles around it depending on the distance between them. Move_base makes use of two types of maps for finding the path, local map for determining the current executing motions and the global cost maps for long distance trajectory-based planning.

2.7.2 End-to-End Learning for Self-Driving Cars

Collected the data samples from images which maps the image pixels into steering commands. Architecture from Nvidia stated that using Convolution Neural network for this approach will be extremely powerful, with minimal training on the samples from cameras mounted to the car and watch humans drive in traffic, local roads, on highways and even on the roads with no line markings for reference.
The system learns Internal representation by convolution operations by detecting features only with steering angle as training signal [10]

2.7.2.1 Convolutional Neural Networks to Process Visual Data

CNNs have transformed the pattern recognition [24] process. Before the widespread adoption of CNNs, most pattern recognition tasks were accomplished using a primary stage of handcrafted feature extraction followed by a secondary stage classifier. The critical breakthrough of Convolution neural networks is that features are now learned by itself during the learning process. The CNN is benchmarked for image classification task by its internal architecture and the powerful combinations of Feature extractions and pooling. By using the convolution kernels to scan an entire image, relatively few parameters need to be learned compared to the total number of operations. [10]
The adoption of CNNs has exploded in recent years because of two significant developments:

- Large, labeled data sets such as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) are now broadly available for training and validation.
- CNN learning algorithms are implemented on massively parallel graphics processing units (GPUs) that tremendously accelerates the learning and inference ability. [10]

2.7.2.2 Data Collection

The data was acquired by taking a car on a drive with cameras mounted to it. In 2016, a team of engineers in Nvidia used Ford Focus with three cameras mounted on. The system was designed as such there is no dependencies about the vehicle vendor or manufacturer. With careful attention the drivers drive around 72 hours on various environment conditions. [10]

2.7.2.3 Neural Network Architecture

The overall design of the Neural network was focused on minimizing the error between the steering angles which is output of network. The neural network architecture, which has 9 layers, including normalization and five convolution layers, and three FC Layers. Input of CNN is an image which is the spilt into YUV planes and passed to the network [10]

The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process [24]. Performing normalization in the network allows the normalization scheme to be altered with the network architecture, and to be accelerated via GPU processing. (Reference) The convolution layers are designed to map the kernels filters to handle feature extraction which are chosen based on series of tests and experiments that vary based on the layer configurations. The configuration uses 2x2 strides for first three layers to perform its convolution operation and 5x5 kernel and 3x3 for last two layers of convolutions. [10]

The five convolution layers are followed by three FC layers followed by its output layer which predicts the inverse radians of turning angles. The FC are designed to function as a controller setting for steering angles It is possible to make a clean break between which parts of the architecture are dedicated for feature extraction and which will be served as a controller [10]

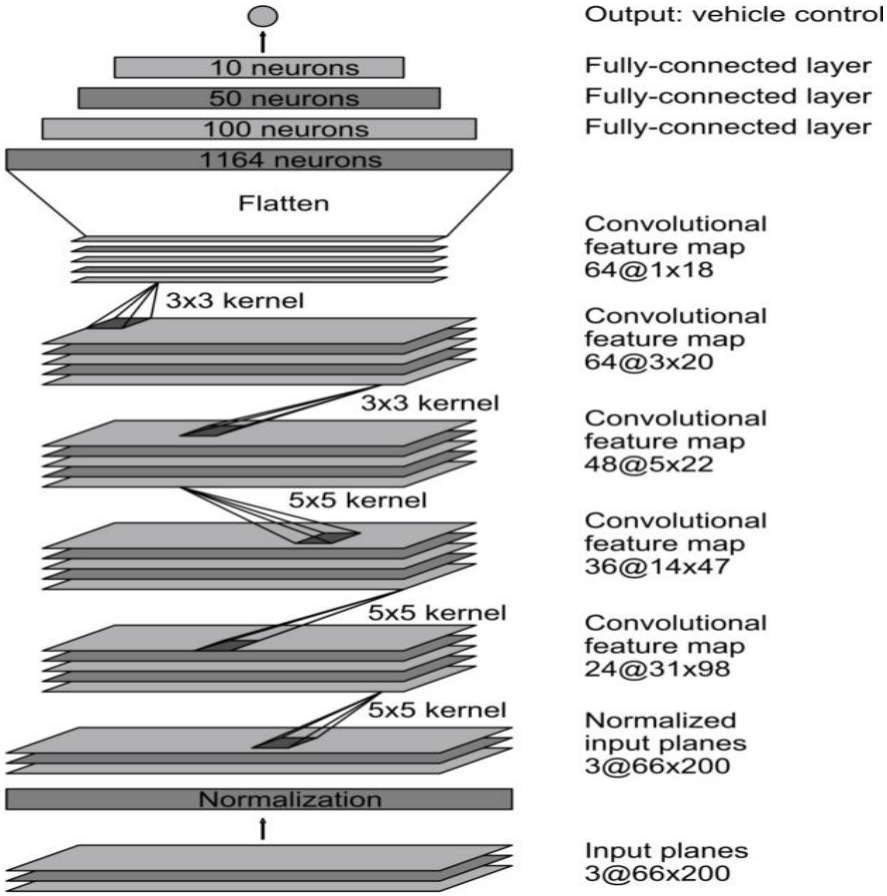


Figure 5 CNN for Nvidia's Self driving Car

2.7.2.4 Training process

1. **Data Selection:** The first process of the training will be selecting the frames in the video which would support the training of neural network. The collected data is labelled with the type of the road the care is driven in, weather condition and the activities of the person driving the car. Like when the person is crossing the road or changing the line. For instance, to train the car to drive in the lane, the frames are selected from the video at 10 FPS in which the driver stays in lanes and rest are discarded. To eliminate the bias towards driving straight, the training data includes a higher proportion of frames that represent road curves. [10]
2. **Augmentation:** Once we have a final set of frames for training process. The dataset is augmented by adding shifts and rotations to make the network learn how to recover from poor position to its orientation. The magnitude of the vectors which is created by these perturbations is chosen randomly from normal distribution. The standard deviation is twice as the human drivers who drove the car.
3. **Simulation:** The simulator inputs the frames from the camera facing towards the road, which is connected to human driven data acquisition process. During this data acquisition process cameras onboard vehicle generate frames mapping to the steering commands of the vehicle. These test videos are time-synchronized with the recorded steering commands generated by the human driver. [10]
4. **Evaluation:** The neural network is evaluated in two various ways. One using the simulator and second on the road test. In the environment for simulating the network, the car in the simulator is driven for couple of hours or many laps as possible. [10]The test data depends on the various lighting conditions and weather conditions and includes highways, local roads, inside residential areas [10]
5. **Simulation Tests:** Estimate the percentage of the time the network could drive the car according to equation 10 by counting the simulated human interventions [10] that occur when the simulated vehicle departs from the centerline by more than one meter. For instance, in real time driving with the human intervention, would need six seconds; time to take over the control of dragged car towards off road and bring it back to center of the road and start cruising again. According to equation 10. Estimating the error percentage is taking a count of these scenarios and multiply six seconds and divide it by total number of stimulated test and subtract it by 1. To map the inverse steering ratios

$$anatomy = (1 - \frac{\#\ of\ interventions[seconds]}{time\ elapsed[seconds]}).100 \quad (10)$$

Thus, if we had 10 interventions in 600 seconds, we would have an autonomy value of

$$(1 - \frac{10.6}{600}).100 = 90\%$$

2.7.2.5 Visualization of CNN State

The figures below show the activations of two kernel maps of two different input samples. An unpaved road from a forest may have too much of noise due to its proportionality and uneven features on roads. So, the network has no useful information to learn from this image. The demonstration that CNN learned to detect useful information from the roads which it has seen during training on its own. We never trained on the outlines of the roads explicitly [10]

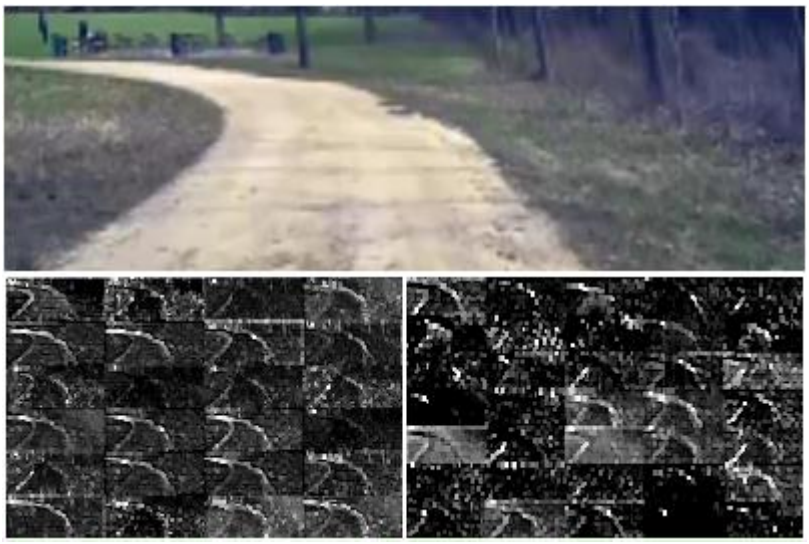


Figure 6 CNN's Visualization of an unpaved road. Top: Subset of the camera image to the CNN. Bottom left: Activation of the first layer feature maps. Bottom right: Activation of the second layer feature maps

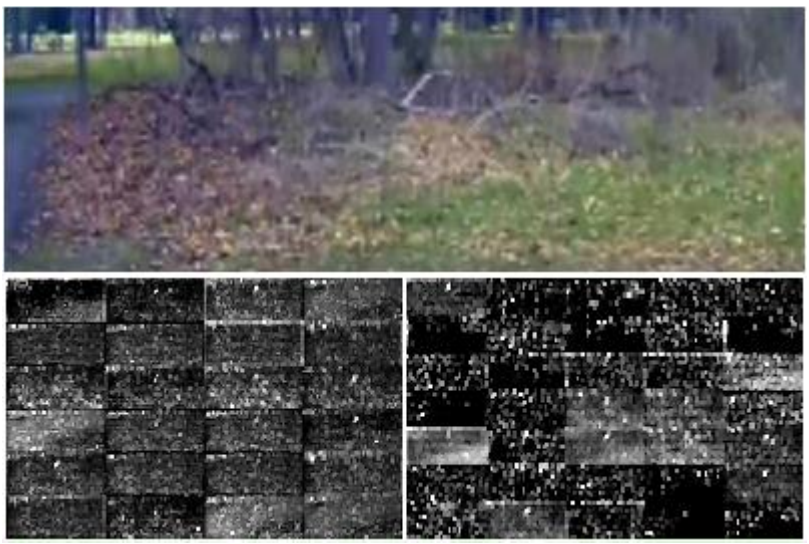


Figure 7 CNN's Visualization of image with no road. The activations of the first two feature maps appear to contain mostly noise, i.e. the CNN doesn't recognize any useful features in this image

2.7.2.6 Conclusion

NVIDIA demonstrated the CNNs are able to learn the entire task of lane and road following without manual decomposition into road or lane marking detection, semantic abstraction, path planning, and control. CNN is able to extract the features from the visible roads and learn to predict the traffic signs, cross roads and unpaved roads. The model learns to understands geometry of the roads with respect to the edge line of the road and the curve of the road without need of explicit mapping to its training labels. The future work is to enhance and improve the efficiency of this model. [10]

2.7.3 A Deep Learning Solution Towards Model-less Obstacles Avoidance

Obstacle avoidance is the known problem in mobile robots. Navigating in autonomous environments will be hard unless the robot is mapped to its environment. Initial approaches were designed to construct the maps based on global maps and cost maps. In the era of machine learning, algorithms have made significant improvement in cognitive tasks such as visual recognition and pattern matching. Neural networks are designed to mimic the architecture of brain to control the environment and decision making. Taking the advantages of deep learning into account, we take obstacle avoidance as an example to show the effectiveness of a hierarchical structure defuses CNN with decision making. The network accepts the depth images as input and generates control commands as class probabilities. This approach had a significant impact while avoiding the obstacles during navigating. [11]

2.7.3.1 Neural Network Architecture

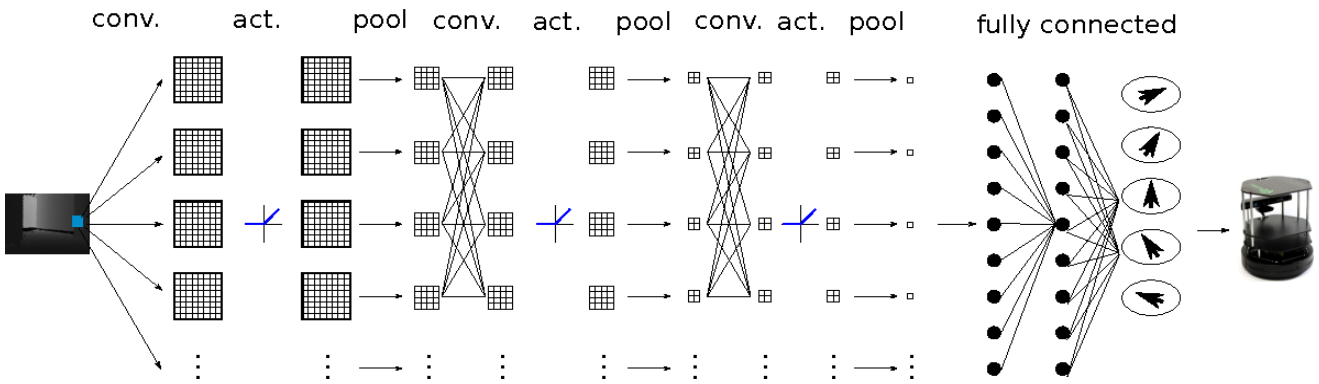


Figure 8 The proposed model which combines CNN with fully connected Neural Network for Robot control

CNN is a type of hierarchical neural architecture which is benchmarked for visual processing and image analysis. Feature extraction is the important functionality of CNN's. By backpropagating the error difference and update those weights. The framework allows to learn a multistage feature hierarchy. Each stage of the network has a feature extraction layer which has a convolution, non-linear activation and pooling layer. [11]

$$y_i j_k = (w_i . x) j_k + b$$

Convolution: Image filtering mimics the operation on kernel filtering or convolution operation. It takes the summed weights of the pixels in the image and forms the 3D grid of its representation.

Non-Linear Activation: After kernel filtering, there must be an activation function to trigger the node the network. This is inspired by the nervous system in the human brain. where synapse is responsible for this

process stimulated by neurons. Synapse function is mimicked by the activation functions such as Tanh, sigmoid, SoftMax, recent additions to process which stood out was ReLU. [11]

$$f(x) = max(0: x)$$

A neuron employing the rectifier is also called a rectified linear unit (ReLU) [18]. Due to its piece-wise linear property, the rectifier executes faster than the previous two non-linear functions for activation.

Pooling: The operations takes the maximum and average of the values in images patches in the operational stride values. There are various pooling methods, ranging from Max Pooling, Mean Pooling and much more. This pooling mainly focuses on eliminating noise and improving robustness of the network [11]

Stride: The stride [18] parameter exists in the convolution layer as well as pooling layer. It means the step over pixels of convolution by patch-by-patch scanning. When stride s > 1, the output feature maps is down-sampled by a factor of s. By introducing the stride parameter, the parameter size of the whole network is reduced. [11]

2.7.3.2 Sample Results and Evaluation

In all trails the robot does not collide with obstacles. However, we consider this does not reflect the true performance of the system sufficiently. To evaluate the performance of the system, we study the similarity of the robot decision and human decision under the same situation.

Firstly, the model uses the soft max classifier to predict the drive classes. Total sample of 1104 depth images were used to train the network which were entirely based on the indoor driving maneuvers. These images were categorized into five driving controls which had a uniform distribution of training samples and test samples according to the paper, training took place on 750 images and was tested on 354 images. The result is shown in figure. We could see that the overall accuracy of the test set is 80.2% [18]. The class accuracy is 79.76%, i.e. the mean accuracy of each class. Furthermore, regarding mis-classification, there is quite low chance for our system to generate totally opposite decision, e.g. to mis-classify “left” as “right”. A large portion of misclassifications could be mis-classifying a “turn-half-left” to a “turn-full-left” or “go-straightforward”. The results prove the effectiveness of the model’s confidence score, in terms of error distributions. [11]

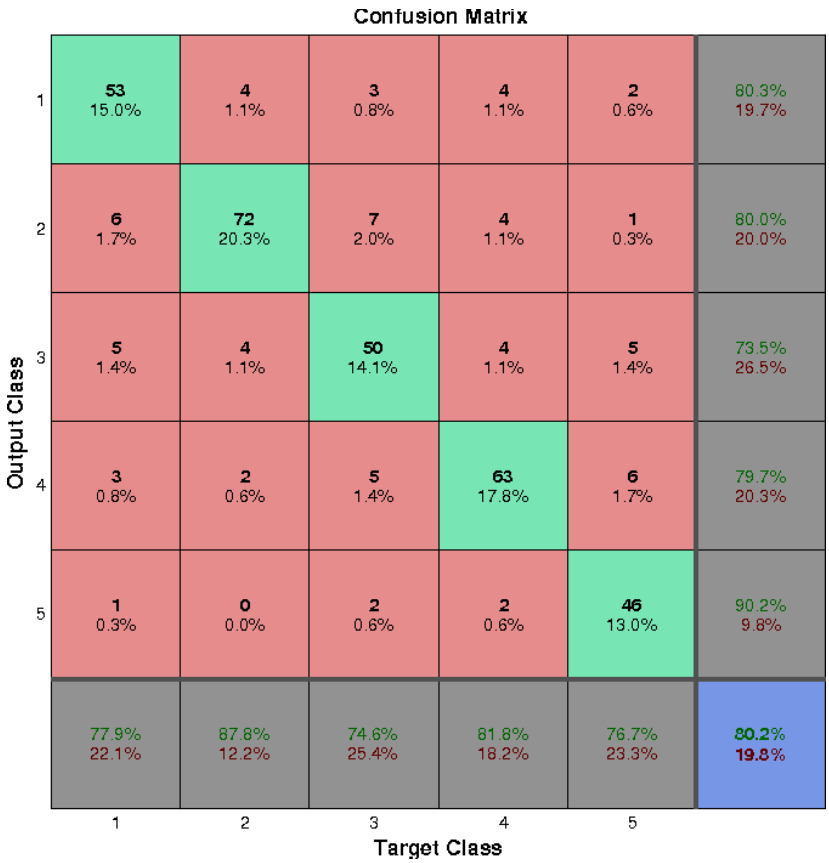


Figure 9 Confusion Matrix on the Test Set. The green-to-red color-map indicates the accuracy of inference. Note that the outcome is equivalent to the five labelled classification problems

.

3 PROTOTYPE DEVELOPMENT APPROACH

Our approach towards designing this interface consists of two parts. The first part consists of data preprocessing and training, which gathers the sensor reading and process it to a neural network. The second is the inference, process of integrating the intelligence into a real time environment.

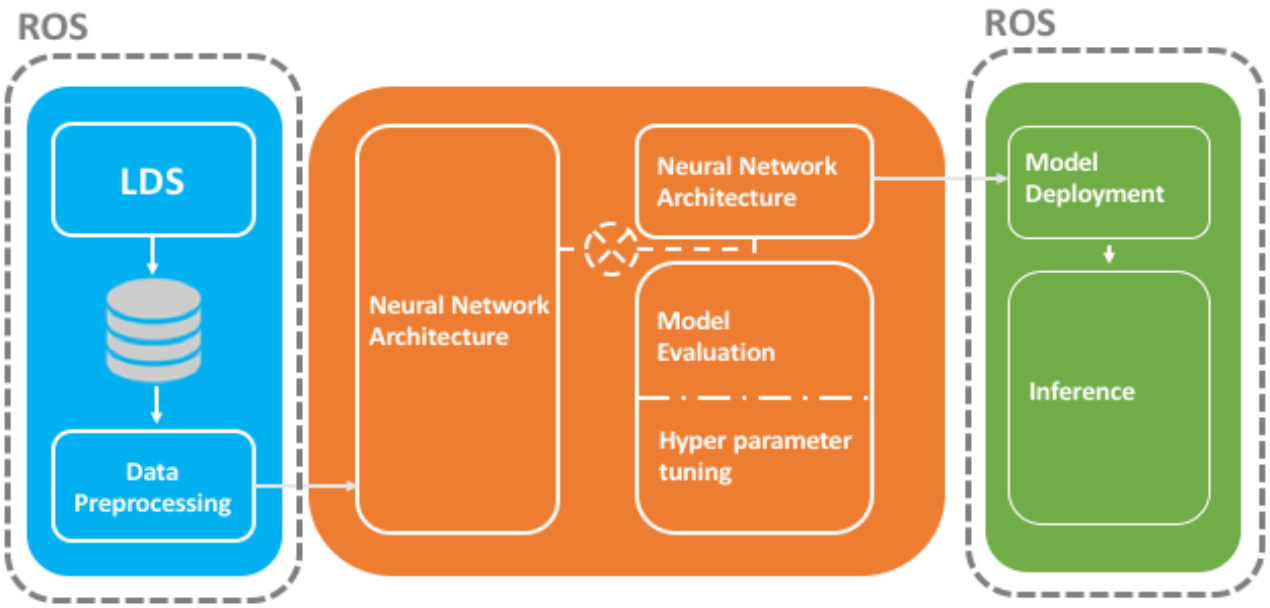


Figure 10 Architecture of AISRA

3.2 Overview of AISRA architecture

The architecture of AISRA makes use of the Multilayer Perceptron Neural network:

- perceptron arranged in layers: the input layer, hidden layer and the output layer.
- Each perceptron in one layer is connected to every perceptron on the next layer. Hence information is invariably "fed forward" from one layer to the next.

3.2.1 Use cases for AISRA

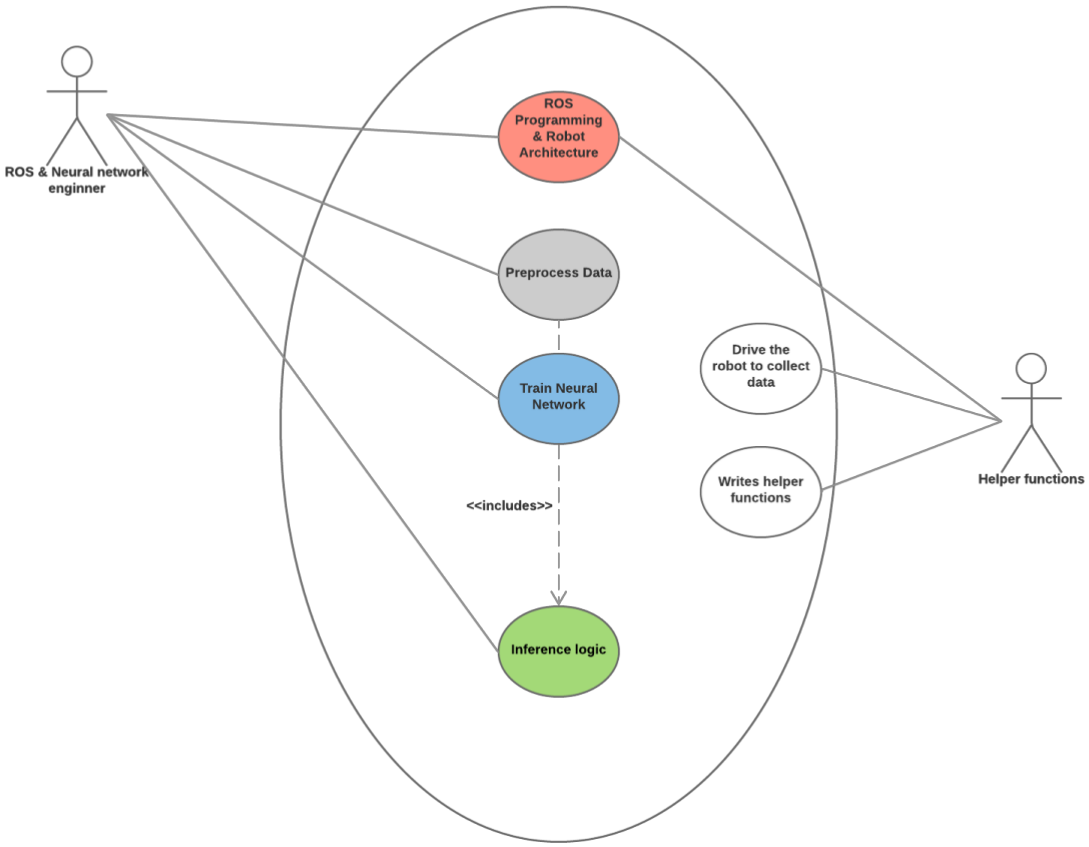


Figure 11 Use case diagram of AISRA

AISRA is an end to end process in making a robot navigate itself, there are two actors on screen

- A person who holds the experience in ROS and Machine learning profile

Roles:

  - Preprocess Data
  - Train the neural network
  - Inference logic

- A person who is skilled in writing helper functions

Roles:

  - Drive the robot around to collect the dataset
  - Write helper functions to filter dataset
  - Write helper functions in ROS Nodes

3.2.2 Robot Operating System (ROS)

ROS is complex piece of middleware which embedded between the operating system and the kernel. Name may specify it as an operating system, but it is mainly built for cluster-based methodology which includes hardware configuration, device control, manipulation of kernel scripts, message passing between user defined process states. Running ROS is a graph-based process which denotes all the active nodes and those nodes are interconnected between them using topics active. Even ROS is not an RTOS but we can map ROS to Realtime code leveraging the power of publisher and subscriber mechanism

Software in the ROS Ecosystem can be separated into three groups:

- language-and platform-independent tools used for building and distributing ROS-based software.
- ROS client library implementations such as roscpp, rospy, and roslisp.
- packages containing application-related code which uses one or more ROS client libraries.

3.2.3 Input Sequence

The input layer of AISRA comprises three major components:

- LDS: Laser Distance Sensor is used to procure input data from the environment to be fed into the system.
- Database: The raw data procured from LDS is exported into CSV files based on timestamps of publishing. This data is saved for future processing.
- Data preprocessing: The raw data is then preprocessed using various ROS functionalities.

3.2.4 Middleware Sequence

The middle layers have no connections with the external world and are thus called the hidden layer. The hidden layer contains of three components:

- Neural network architecture: System specific neural network architecture that comprises of 360 input nodes, 194 hidden nodes and 9 output modes.
- Model Evaluation: The model configured was evaluated using K-Fold evaluation metric.
- Hyper parameter tuning: Hyper-parameters are parameters that are not directly learned within estimators. Hyperparameters must be set explicitly before training and tuned to train the system to predict probabilities in the cases that were not already present in the training dataset.

3.2.5 Inference Sequence

The output layer performs logical operations based the outputs from hidden layer so that the whole network classifies the input layers into specified units. By varying the number of nodes in the hidden layer, the number of layers, and the number of input and output nodes, one can classification of points in arbitrary dimension into an arbitrary number of groups. Hence feed-forward networks are commonly used for classification.

The two main functionalities happening in the models output layer are:

- Activation function: The function used in the model for activation of output nodes is SoftMax. SoftMax activation is used to show the highest probability among the 9 nodes to the given instance of the row data.
- Inference: We need high-end configured systems to initially train the neural network models. From this trained model we get the neural network model architecture and model weights. Once we have the pre-trained model, we can download the model summary and model weights to deploy the model on Nvidia Jetson predict computational results.

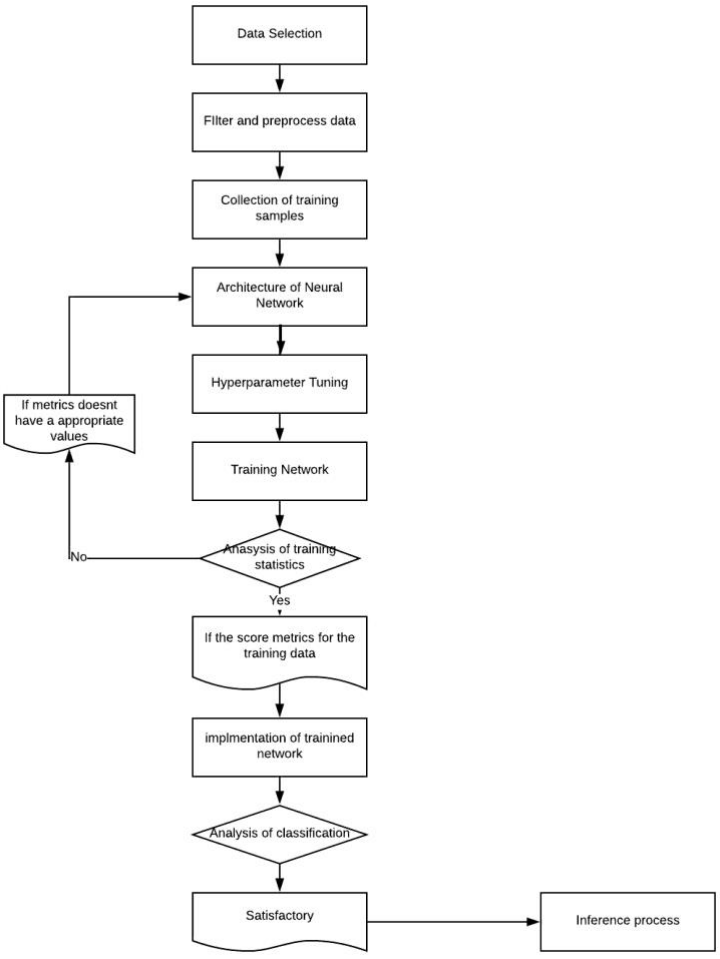


Figure 12 Flowchart AISRA

3.3 Data Preprocessing

We used Turtlebot 3 from Robotics for data collection and inferencing

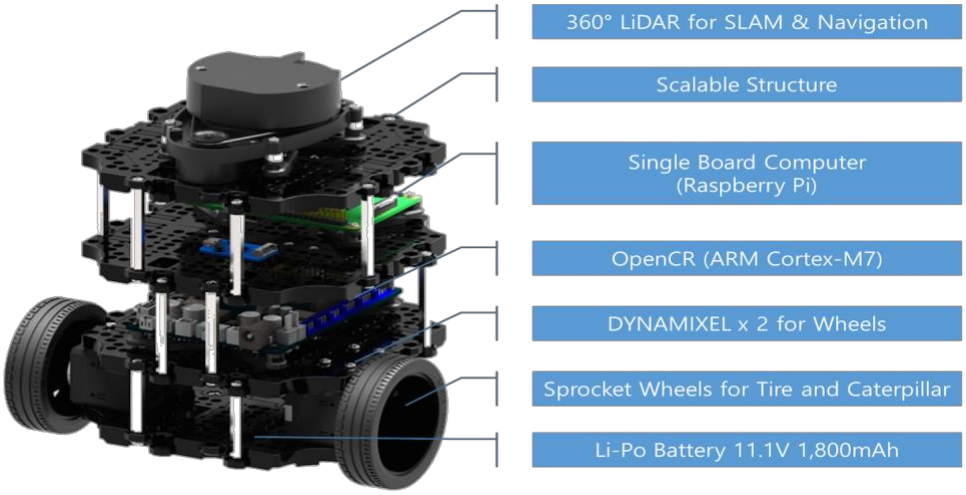


Figure 13 Turtlebot 3 Burger

We took our robot on a drive in various environments, such as a pathway, house and a room with full of obstacles. Map images as shown in figure 14 and figure 15. The robot was mounted with an LDS sensor and data was recorded overtime.

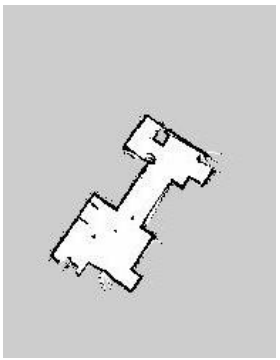


Figure 14 Map of Room with obstacles

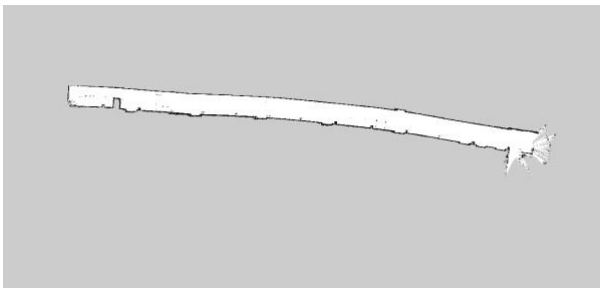


Figure 15 Map of Longest Floor

A bag [19] is a file format in ROS for storing ROS messages data. Bags (so named because of them. bag extension) have an important role in ROS, and a variety of tools have been written to allow you to store, process, analyze, [2] and visualize them. ROS Bags serves as a container to record all data which are published by various sensors and feedback signals. ROS provides a mechanism to record all the data with its internal component called ROS Bags.

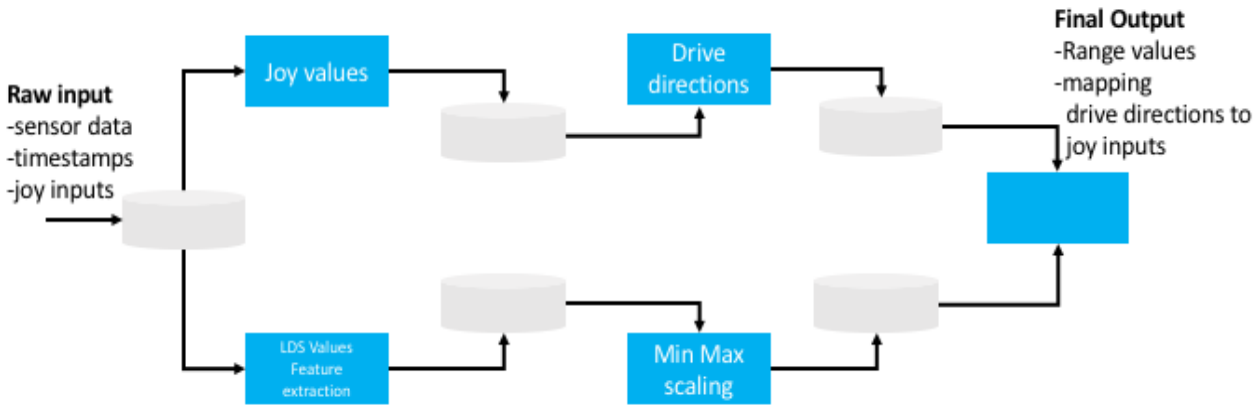


Figure 16 Data processing Phase

ROS Bags works as a tool to record and replay to debug various algorithms, since every time you see the same data during playing the bags which helps to isolate and fix the errors. ROS bags has graph of publishing messages which is similar to one when it was recorded. Though it may face the synchronize issues, to handle this ROS bags provides the attribute called stimulated clocks to publish data along with the timestamp. [2]

The bag file [2] format is very efficient for both recording and playback, as messages are stored in the same representation used in the network transport layer of ROS. All the data during the data accumulation process is collected as rosbags containers.

Raw Inputs: To procure the raw data required for training the neural network model, we played the ros bags to export joystick values and LDS Sensor data to a CSV files based on timestamps of publishing and saved the recorded data as individual files for future data processing. Both the datasets obtained from joystick values and LDS sensors have unique timestamps which served us to merge the datasets into a single output file after data processing and feature extraction.

3.3.1 Preprocessing Raw Data inputs

ROS provides an open source package called ps3joy that can work with PS3 Controllers and is suitable for teleoperation of robots. This ROS Package returns the Joy axis values and button trigger values for every action taken.

```
devaraja@aisra:~$ rostopic echo -n1 joy
header:
  seq: 1888
  stamp:
    secs: 1526266142
    nsecs: 557739920
  frame_id: ''
axes: [-1.0, -0.0, -0.0, -0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0630381628870964
, 0.0, 0.0, 0.0, 0.0, 0.0, -0.0, -0.0, -0.0, -0.0]
buttons: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
---
devaraja@aisra:~$
```

Figure 17 Terminal screenshot of Joy Values

Process Drive Directions:

Figure 18 Joy Stick axis movements

The exported values from Joy controller is saved in form of CSV files according to the timestamp values. Every stream of data received from the Joy controller is parsed in the format of JSON which returns axis values and triggered button values.

For the intent of processing drive directions, we extract the values from the 16 and 17 axis movement ranging its value from 0 to 1. Once we have the extracted values from the returned object, we categorize the movement into nine different drive states. That is, up, down, left, right, up-left, up-right, bottom-left, bottom-right and stop.

Once we have all the drive directions for all the data, we create a data column in the dataset which has the equal joy stroke values to its instance of axis values. This column serves as Y axis for our future data file or our supervised learning model.

3.3.2 Processing LDS Scanned Data

The system needs to sense its environment in more the depth manner to understand the geometry of its environment, looking out to some distance to find obstacles and build a map that is useful for performing tasks it is designed for. LDS Sensors are designed to measure distances between the robot to the obstacle or object passing by. These lasers may range from 10 meters to 3000 meters based on the purposed of usage. LDS are mainly used for localization in the space and perform navigation without colliding with objects

LDS Sensor provides the 360-degree circumference of its environment that collects the data for robot to localize itself in the MAP and navigate around the environment. The Simultaneous Localization and Mapping, or SLAM, [16]is a technique to draw a map by estimating current location in an arbitrary space. [16]

- The LDS-01 is used for TurtleBot3 Burger, Waffle and Waffle Pi models.
- It supports USB interface(USB2LDS) and is easy to install on a PC.
- It supports UART interface for embedded board.

Machine Learning models use LDS as their primary sensor for mapping, localization, and obstacle avoidance. 360 Laser Distance Sensor LDS-01 is a cost-effective way to overcome all these shortages without compromising the performance of the laser sensor. [16]

Table 1 Hardware Specification of LDS Sensor

Operating supply voltage	5V DC ±5%
Light source	Semiconductor Laser Diode(λ=785nm)
LASER safety	IEC60825-1 Class 1
Current consumption	400mA or less (Rush current 1A)
Detection distance	120mm ~ 3,500mm
Interface	3.3V USART (230,400 bps) 42bytes per 6 degrees, Full Duplex option
Ambient Light Resistance	10,000 lux or less
Sampling Rate	1.8kHz
Dimensions	69.5(W) X 95.5(D) X 39.5(H)mm

Table 2 Performance of LDS Sensor

Distance Range	120 ~ 3,500mm
Distance Accuracy (120mm ~ 499mm)	±15mm
Distance Accuracy (500mm ~ 3,500mm)	±5.0%
Distance Precision (120mm ~ 499mm)	±10mm
Distance Precision (500mm ~ 3,500mm)	±3.5%
Scan Rate	300±10 rpm
Angular Range	360°
Angular Resolution	1°

The LDS sensor will scan the environment is it placed in all 360 degrees. Each scan from LDS sensor returns 360 values specifying its range of distance in each degree. We extract just the required columns from the dataset which holds just the ranges of data which serves as a feature for supervised model. . [16]

Figure 19 Screenshot of RAW LDS Values

3.3.3 Final Dataset

As of now, there are two data blocks available to us which have to be merged to get the full-fledged dataset for the model to be trained. Timestamps of publishing the data will act as a key to merge both of the datasets, thanks to python package pandas for this handy tool to perform this operation.

Table 3 View of Joy axis data table

Time stamps	axis_0	axis_1	axis_2	axis_3	axis_4	axis_5	axis_6	axis_7	…	axis_19
20180518040434	1	0	0	0	0	0	0	0	…	0
20180518040435	0	1	0	0	0	0	0	0	…	0
20180518040436	1	-0.9	0	0	0	0	0	0	…	0
20180518040437	0.2	1	0	0	0	0	0	0	…	0
20180518040438	1	1	0	0	0	0	0	0	…	0

Table 4 View of LDS data table

timestamps	range_0	range_1	range_2	range_3	range_4	range_5		range_360
20180518040434	2.50500011	2.50099993	2.53399992	2.44700003	2.546	2.47600007	..	2.56699991
20180518040435	2.49799991	2.53600001	2.50500011	2.47799993	2.48000002	2.55800009	..	2.50500011
20180518040436	2.50099993	2.49000001	2.53999996	2.454	2.53200006	2.49300003	…	2.47600007
20180518040437	2.52999997	2.53999996	2.51799989	2.52200007	2.43799996	2.56699991	…	2.50900006
20180518040438	2.48600006	2.47799993	2.38400006	2.50900006	2.4289999	2.51600003	…	2.53999996

After completion of merging of the data from two data blocks, timestamps values and axis values that are merged from joy drive dataset is dropped from the final dataset. So final dataset holds the range values and the joy strokes which has all the dependent and independent columns for our neural network.

Table 5 View of preprocessed data table

range_0	range_1	range_2	…	range_360	Joy strokes
2.50500011	2.50099993	2.53399992	…	2.47600007	Up
2.50500011	2.50099993	2.53399992	…	2.47600007	Up
2.50500011	2.50099993	2.53399992	…	2.47600007	Left
2.50500011	2.50099993	2.53399992	…	2.47600007	Down
2.50500011	2.50099993	2.53399992	…	2.47600007	up

We have collected over 365000 rows of LDS data instances, during the mapping process.

Final Dataset for model training

Following is the brief outlook on the data sample from the final dataset and its subsequent number of collected data rows with respect to the data sample's joy strokes.

Table 6 Category based data collection

Directions	Samples count
Up	85517
Left	81297
Right	68276
Up-Right	5173
Up-Left	2174
Down	1313
Bottom-Right	849
Bottom-Left	538

I went ahead further and divided the dataset into X and Y values. X values will hold the range features, and Y values will hold the classes of joy strokes.

3.3.4 Stratified sampling

Until now we had a random sampling method when dividing the dataset into a train and split sets. However, in our classification problem, we have 8 different classes, and random sampling method will work if we have the large dataset for this. However, the dataset which is gathered is a relatively smaller one compared to the distribution of data to 8 different classes. If the training data is not prepared correctly, we may run into high bias.

So, we decided to collect the data evenly from every different class and form a dataset for training. So, we have data from all the drive directions to train the neural network. Therefore, it learns every class to its best accuracy.

For this method, we use Stratified sampling method from sci-kit learn library.

```
from sklearn.model_selection import StratifiedShuffleSplit
spilt_data = StratifiedShuffleSplit(test_size=0.25, random_state=42)
for train_index, test_index in spilt_data.split(x_dataset, y_dataset):
        x_train,x_test = x_dataset.iloc[train_index],x_dataset.iloc[test_index]
_       y_train, _y_test = y_dataset.iloc[train_index], y_dataset.iloc[test_index]
```

Stratified shuffle split provides the test and training indices for our dataset. Stratified Shuffle split is an object that concatenates two methods in its process. Stratified KFold and Shuffle Split

Stratified sampling takes few parameters:

- test size: we define the test size of the data sampling process. Whereas in our purpose, we have separated over 25% of overall dataset for the testing purpose.
- train size: we define the training data set size here, in our case as we declared the testing size the rest will be allocated for training data.

Now we are ready to do stratified sampling based on the classes. We get all the training and testing variables which we can process for training and testing of neural network

3.3.5 Encode the output variable

When modeling the multi-classification architecture using neural networks, it is always a good practice to reshape the output attribute from a vector that contains the values of each class into a matrix that contains the Boolean values of those classes. This process is called One Hot encoding.

ML algorithms prefer to work with numerical values rather than text attributes because with text attributes we cannot compute its median. However, the primary issue with numerical representation is the that ML algorithm will assume that two nearby values are more similar than two distant values.

A standard solution to overcome this shortage is to create one binary attribute per category: one attribute is equal to 1 when it belongs to a particular category (and 0 otherwise), another attribute equal to 1 when it belongs to another particular category (and 0 otherwise), and so on. This categorization is referred to as one-hot encoding because only one attribute will be equal to 1 (hot), while the other will be 0 (cold).

Scikit-Learn provides an encoder called 'OneHotEncoder' which helps in converting integer categorical values into one-hot vectors. The output of this encoder will be a SciPy sparse matrix, instead of a NumPy array. This is particularly of great help when we have categorical attributes with thousands of categories. Both transformations (from text categories to integer categories, then from integer categories to one-hot vectors) in one shot can be applied using the LabelBinarizer class.

```
encoded = LabelEncoder()
encoded_y_train = encoded.fit_transform(y_train)
encoded_y_test = encoded.fit_transform(y_test)

y_train = np_utils.to_categorical(encoded_y_train,9)
y_test = np_utils.to_categorical(encoded_y_test,9)
```

3.3.6 Feature scaling of LDS Scan data

Feature scaling is one of the most significant transformations applied in data transformation step. It is rare that Machine Learning algorithms perform well when the input, numerical attributes have very different scales. So, it is at most necessary to get all attributes of the same scale.

It is crucial to fit the scalers to the training data only and not to the full dataset. There are two ways to get all attributes on the same scale: min-max scaling and standardization. [12]

Min-max scaling (or **normalization**) is a simple method used to scale attributes. All the values are shifted and re-scaled to range from 0 to 1. Scikit-Learn provides a transformer called MinMaxScaler to perform this transformation.

- MinMax scaling logic: subtract the min value and divide by max minus min.

Standardization is different from min-max scaling because it does not bind values to a specific range. So, it is much less affected by outliers. However, not binding values to a range might be a problem in case of algorithms that expect an input value ranging from 0 to 1. Scikit-Learn provides a transformer called StandardScaler for standardization.

- Standardization logic: subtract the mean value and divide by the variance so that the resulting distribution has unit variance.

Most ML algorithms expect an input data in range 0 to 1. This is the reason Machine Learning model learns fast when the values range from 0 to 1. Hence, I have used MinMaxScaler from Scikit-Learn to process this step.

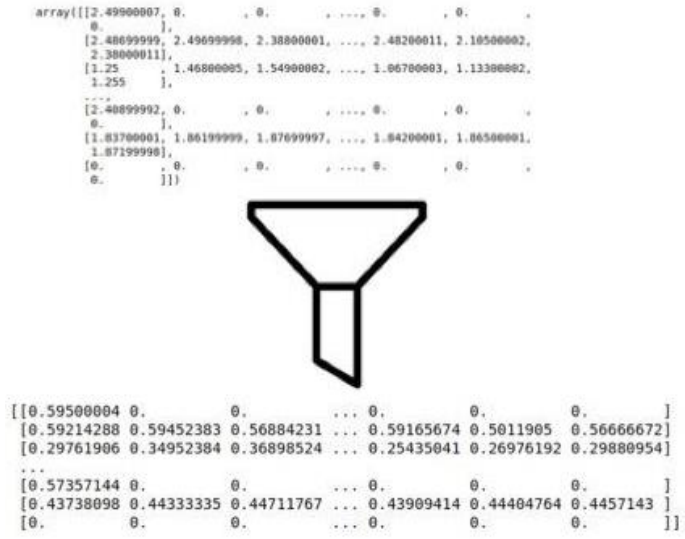


Figure 20 Min Max Scaling

All the features extracted in the earlier step goes through to MinMaxScaler. So, each value will range from 0 to 1.

```
In [8]: from sklearn.preprocessing import MinMaxScaler
        scaler = MinMaxScaler(feature_range=(0,1))
        x_train = scaler.fit_transform(_x_train)
        x_test = scaler.fit_transform(_x_test)
        print(x_train)
```

```
[[0.59500004 0.         0.         ... 0.         0.         0.        ]
 [0.59214288 0.59452383 0.56884231 ... 0.59165674 0.5011905  0.56666672]
 [0.29761906 0.34952384 0.36898524 ... 0.25435041 0.26976192 0.29880954]
 ...
 [0.57357144 0.         0.         ... 0.         0.         0.        ]
 [0.43738098 0.44333335 0.44711767 ... 0.43909414 0.44404764 0.4457143 ]
 [0.         0.         0.         ... 0.         0.         0.        ]]
```

Figure 21 Minmax Scaling code Snippet

3.4 Neural Network Architecture

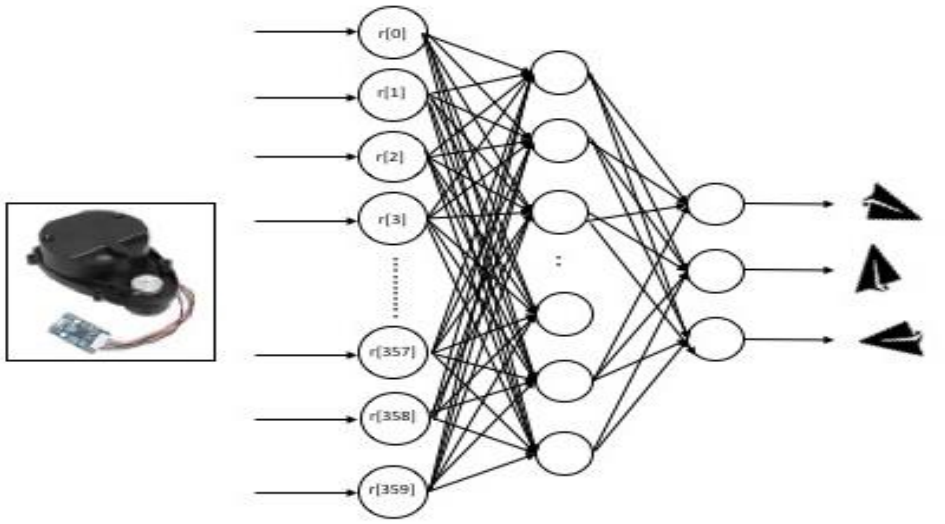


Figure 22 Architecture of Neural Network

The neural network architecture of AISRA can be broken down into three fundamental layers: the input layer, the hidden layer, and the output layer. The nodes within each layer are independent of one another. However, every node in one layer is connected to every other node in the subsequent layer. This means that every node in the input layer is connected to every node in the hidden layer and every node in the hidden layer is connected to the output layer. The data is processed in one stage and forwarded to the next, which means there are no loops in the system. The signals always traverse unidirectionally making this a feed-forward system.

The neural network receives its input from the LDS Sensor which will render 360 inputs to the 360 input nodes of the system. Input nodes will hold the current detail range information about the 360-degree circumference of the space scanned by LDS Sensor. The input layer is connected to the hidden layer which is activated by rectified linear function ('ReLU'). The model is evaluated using K-Fold evaluation metric and hyper parameter tuning. The output layer uses the SoftMax activation function which outputs the probabilities for nine different categories available.

Neural networks produce multiple outputs in multiclass classification problems. However, they do not have the ability to produce exact outputs; they can only produce continuous results. We would apply some additional steps to transform continuous results to exact classification results. Categorical cross entropy is a loss function that minimizes the loss between two different instances of a data row, so it gets

nearer to the prediction level. We use categorical cross entropy with SoftMax activation function, so it predicts every probability of every class.

3.4.1 Training Process:

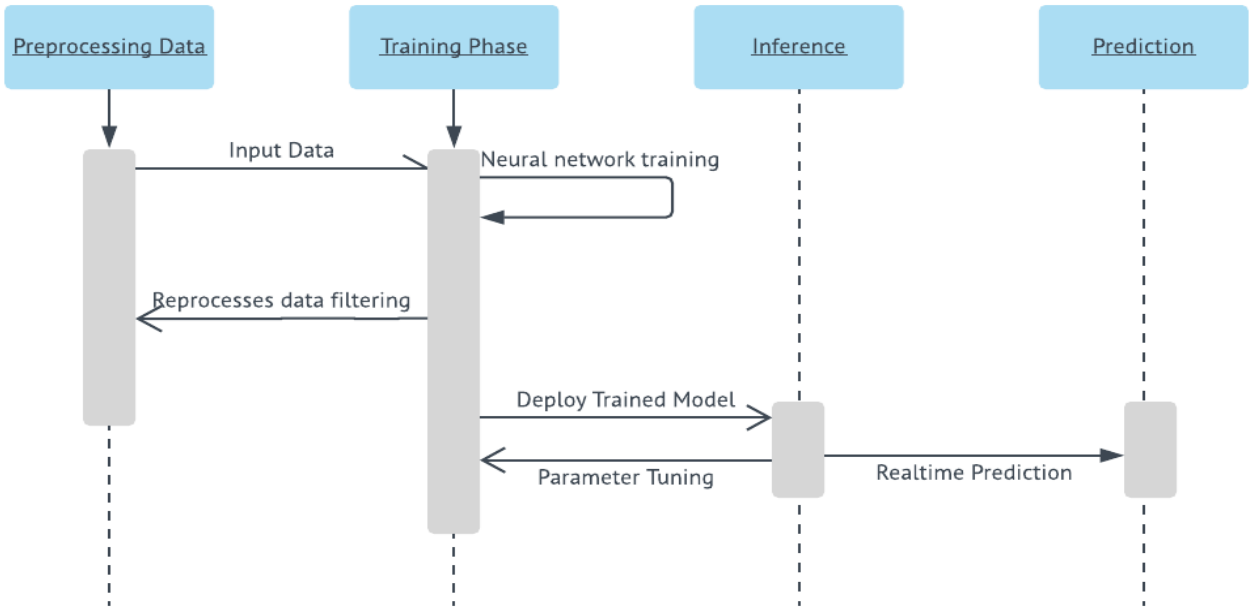


Figure 23 Sequence diagram of training process

This process constitutes of four main objects

- **Pre-processing :** This object just focuses on the preparation and manipulations of data.
- **Training phase:** Neural network training and configuring of parameters, this object constitutes under goes series of evaluations and sends the trained model to inference object, if in case the trained model doesn't work as expected. The model is again trained by tweaking the data manipulation by calling the pre-processing object.
- **Inference Phase:** it is responsible for Realtime predictions from the Realtime input from the sensors.
- **Prediction Phase**: This phase has a scaled and pre-processed inputs which holds LDS sensor data which streams the input data

3.4.1.1 System Setup:

Keras is a High-Level API written in python and built on top of TensorFlow or Theano. It was developed to test the ANN rapidly and deployment for testing. Keras supports two versions of python .i.e. 2.7 and 3.5 and later. Because of the architecture of TensorFlow and Theano, Keras also runs on CPU and GPU enabled system for training and inference. Keras was developed and maintained by François Chollet, [8]a Google engineer using four guiding principles:

- **Modularity:** Keras API provides sequence modelling and functional modelling of the neural network do it can run as a graph or a standalone sequence stack. Model can be even further manipulated using concatinating models and much more
- **Minimalism:** The library provides just enough to achieve an outcome, no frills and maximizing readability.
- **Extensibility:** New components are designedly easy to add and use within the framework, intended for researchers to trial and explore new ideas.
- **Python:** Uses native Python. No separate model files with custom file formats. [8]

Theano Library: Theano is base library for neural networks which is written in python, which can be used to architecture deep learning models or create wrappers for other libraries to simplify the application building process

Theano is much mathematical flavored coding in python. It basically translates the structures into a native code for numpy, dynamic libraries as BLAS and native C++ which can run on both GPU and CPU clusters. [15]

The syntax of Theano is much of symbolic representation, where the mathematical expression is much defined in more abstract way, compiled and later made used to calculate. Theano is much preferred for complex neural networks or very deep neural network more than three to four hidden layers. It is considered as one of the standards for Deep learning research and development [15]

Keras is a wrapper on Theano and hides its code completely. It provides an easy accessible API to create neural network models. It completely isolates the wrapper from another most popular computational library called TensorFlow [15]

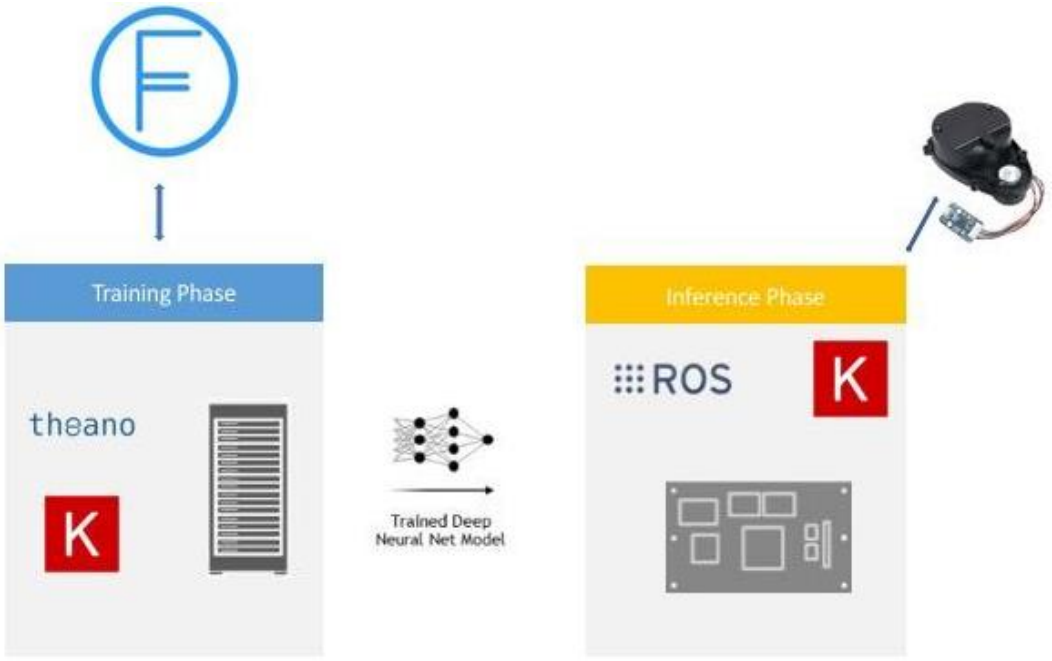


Figure 24 Process flow of neural network training

Training Process takes place in the Floyd hub cloud, due to the architecture of Theano and large dataset. Training takes places in computational Nvidia's GPU cluster. Once the data is trained the model files are transferred to Jetson TX2.

Once the model is configured and final model would look as illustrated under model summary

Model Summary

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 184)	66424
dense_2 (Dense)	(None, 184)	34040
dense_3 (Dense)	(None, 3)	1665

Total params: 102,129
Trainable params: 102,129
Non-trainable params: 0

3.4.2 Hyperparameter Tuning

While the system learns the model parameters through training, hyperparameters must be set explicitly before training. Hyper-parameters are parameters that are not directly learned within estimators. In scikit-learn, they are passed as arguments to the constructor of the estimator classes. Scikit-Learn implements a set of sensible default hyperparameters for all models, but these are not guaranteed to be optimal for a problem. The best hyperparameters [28] are usually impossible to determine ahead of time, and tuning a model is where machine learning turns from science to trial-and-error based engineering.

It is inferred to search the hyper-parameter [28] space for the best cross-validation score. Any parameter provided when constructing an estimator can be optimized. A search consists of:

- an estimator (regressor or classifier such as sklearn.svm.SVC()).
- a parameter spaces.
- a method for searching or sampling candidates.
- a cross-validation scheme.
- a score functions.

Important functions used for searching are:

- Parameter Grid [28] generates all the combinations of a hyperparameter grid.
- sklearn.model_selection.train_test_split[28] is a utility function to split the data into a development set usable for fitting a GridSearchCV instance and an evaluation set for its final evaluation.
- sklearn.metrics.make_scorer[28] makes a scorer from a performance metric or loss function.

Scikit-learn uses the wrappers to provide by Keras [7] models to make use of sci-kit functionality for feature adoptions. Keras classifiers and Keras regressors are two mainly used wrappers with Scikit-learn. To make use of these classifiers and regressors the model has to be defined in the way of writing a function which holds the parameters for model training. The constructor for the KerasClassifier class can take default arguments that are passed on to the calls to model, such as the number of epochs and the batch size. The following wrapper is used to implement the Scikit-Learn classifier interface. [7]

keras.wrappers.scikit_learn.KerasClassifier(build_fn=None,**sk_params)

Grid search is a model hyperparameter optimization technique [20]. The grid search provided by the scikit-learn class, GridSearchCV exhaustively generates candidates from a grid of parameter values specified with the param_grid parameter. It performs an exhaustive search over specified parameter values for an estimator. When constructing this class, render a dictionary of hyperparameters to evaluate in the param_grid argument. This is a map of the model parameter name and an array of values to try. [7]

By default, accuracy is the parameter which has to be on focused and optimized. But other scores can be mentioned in the attributes of the GridSearchCV class. Also, by default GridSearch will use one thread the couple of available cores in the processors, specifying n_jobs are equal to -1 will activate all the cores of the processor to run simultaneously [7]

Grid Search will form a grid which will have all the parameters which will be trained on. It forms all the combinations of available parameters and run one after one. Cross validation will have a number of splits in which the case will be evaluated on.

The return values of gridsearch are the best parameters and their accuracy scores. Gridsearch fits the model function and observes the best parameters and returns a dictionary with all the values

The batch size defines the subsets of the entire dataset exposed to the network. It is also an optimization of training function. The number of Epochs defines the numbers of times the dataset is iterated during the training process.

- **Optimizer tuning**: An optimizer is one of the two arguments required for compiling a Keras model. An optimizer is either instantiated before passing it to model.compile() or called by its name. In the second case, the default parameters for the optimizer will be used. [7]

  For example:

  model.compile(loss='mean_squared_error', optimizer='sgd')

  keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)

- **Batch size tuning**: Batch size defines the number of samples that will propagate through the network. It generally requires less memory. Since the train network uses less number of samples, the overall training procedure requires less memory. It is especially important in the cases dataset does not fit in memory. Typically, the networks train faster with mini-batches. That is because of the update weights after each propagation. The smaller the batch, the less accurate estimate of the gradient. Stochastic is just a mini-batch with batch size equal to 1. Gradient changes its direction even more often than a mini-batch. [7]

- **Initializers (Neural network weight's) tuning**: Initializations define the way to set the initial random weights of Keras layers. The keyword arguments used for passing initializers to layers will depend on the layer. It usually is kernel initializer and bias_initializer. Small random values were used to initialize Neural network weights. Tune the selection of network weight initialization by evaluating all of the available techniques. [20]

  For example:

  model.add(Dense(64,kernel_initializer='random_uniform', bias_initializer='zeros'))

The same weight initialization method is used on each layer. Ideally, it may be better to use different weight initialization schemes according to the activation function used on each layer. [7]

- **Epoch tuning**: An "epoch" describes the number of times the algorithm sees the entire data set. So, each time the algorithm has seen all samples in the dataset, an epoch has completed. One epoch is one forward pass and one backward pass of all the training examples. Updating the weights with a single pass or one epoch is not enough. We use a limited dataset and to optimize the learning and the graph using Gradient Descent which is an iterative process. One epoch leads to underfitting of the curve in the graph. As the number of epochs [20] increases, the number of times the weights are changed in the neural network increases, and the curve goes from underfitting to optimal to overfitting curve.

Once we have the model with all the required architecture we wrap the neural network into Keras classifier to support Scikit Learn GridSearchCV module [12]

3.4.2.1 Hyper parameter test cases

We have run the test cases to tune the hyper parameter for the following configurations.

Case 1: For 200 EPOCHS

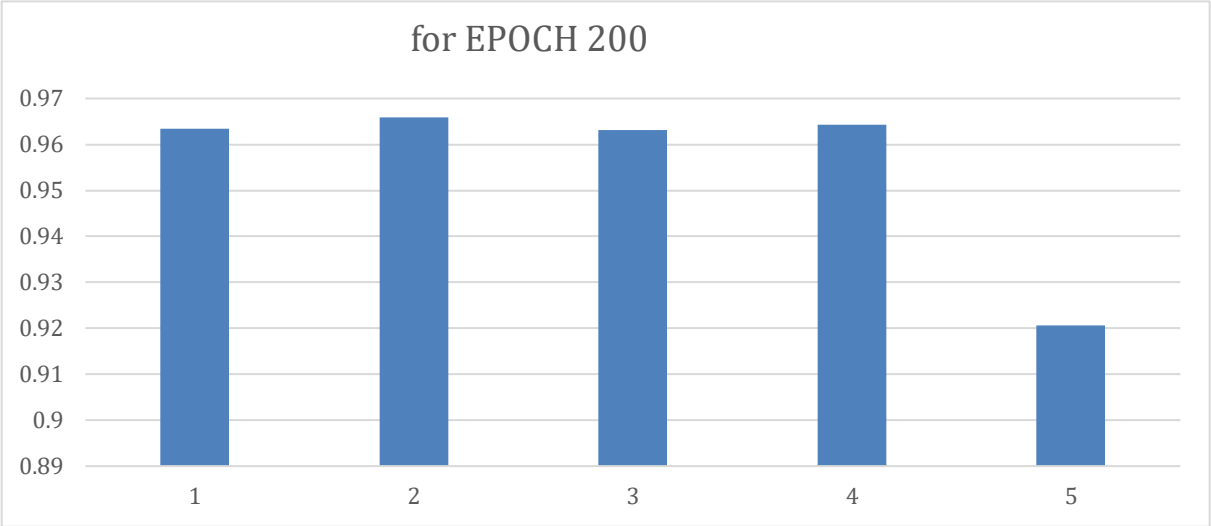


Table 7 Case 1: For 200 EPOCHS

	Epoch	Kernels	Optimizer	Batch size	Accuracy	Loss
1	200	glorot_uniform	adam	32	0.963467	0.00266
2	200	glorot_uniform	rms_prop	64	0.961867	0.002407
3	200	glorot_uniform	adam	64	0.963067	0.002927
4	200	normal	rms_prop	128	0.964267	0.002175
5	200	uniform	rms_prop	128	0.920667	0.014611

Looking at the table 7 we can see that all the optimizers did their best job in minimizing the loss. But I found that adam optimizers bench marked this process for 200 epochs with 96% of accuracy with 0.002927 of loss.

Case 2: For 150 EPOCHS

In this case much we find much better performance with all the instances, especially for adam optimizer most of the test variables suits to perform more then 96% all the time. But when considering even loss into accountability adam optimizer throws up lower loss then compared to other instances when paired up with glorot_uniform with 64 batch sizes.

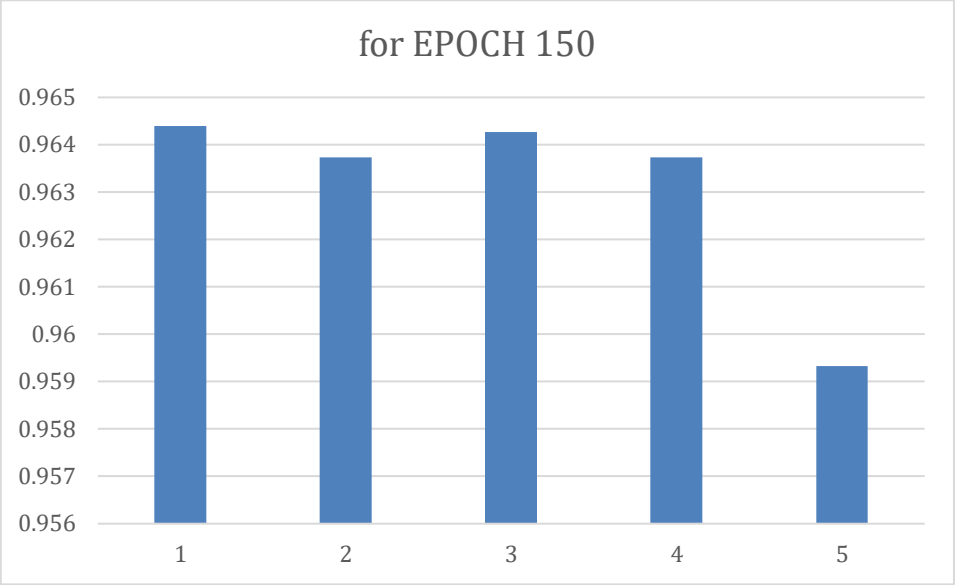


Table 8 Case 2: For 150 EPOCHS

	Epoch	Kernels	Optimizer	Batch size	Accuracy	Loss
1	150	normal	rms_prop	32	0.9644	0.003116
2	150	normal	adam	32	0.963733	0.002853
3	150	glorot_uniform	adam	64	0.964267	0.00262
4	150	uniform	adam	64	0.963733	0.002473
5	150	glorot_uniform	rms_prop	128	0.959333	0.001969

Case 3: For 100 EPOCHS

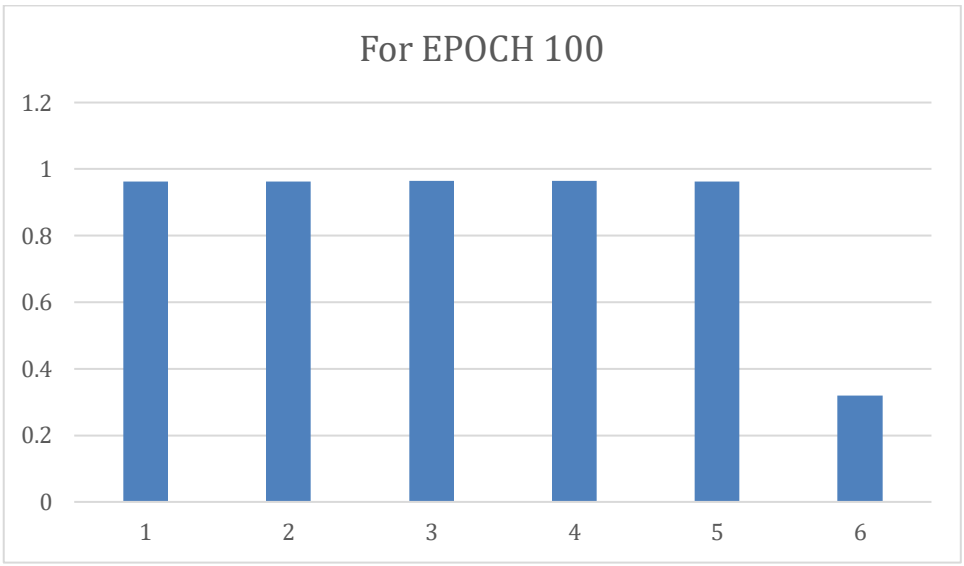


Table 9 Case 3: For 100 EPOCHS

	Epochs	Kernels	Optimizer	Batch size	Accuracy	Loss
1	100	glorot_uniform	adam	32	0.963467	0.003523
2	100	uniform	adam	32	0.963333	0.002739
3	100	glorot_uniform	adam	64	0.963867	0.00264
4	100	normal	adam	64	0.963867	0.00262
5	100	glorot_uniform	rms_prop	128	0.961733	0.003104
6	100	uniform	rms_prop	128	0.319733	0.4111163

In this test case we found out that the rms_prop does perform very low compared to other cases. The accuracy of the model falls down to 30% where still adam optimizers benchmarks even this test case with 96%

Selections of best hyper parameters based on the Grid CV Results

After analysis the complete test cases for various epoch conditions, we found that following hyper parameters will train the neural network with best accuracy and minimize the error to the lowest value.

Table 10 Selection of best hyperparameters

Variables	Hyper Parameters
EPOCHS	200
KERNELS	GLOROT UNIFORM
OPTIMIZER	ADAM

3.4.3 Model Evaluation:

Cross validation is a model of evaluation that is better than residuals. The actual problem with these residuals is that they don't give an insight of how the model would work on unseen data. The learner will be asked to make the new predictions based on the model which is trained. To supersede this problem, is that not to use the whole dataset for training purpose. The better approach is to divide the dataset into training and tests sets. So, when the training set after the spilt function undergoes the network training. So now the model will be evaluated for the test dataset to get the accuracy of the model which is trained. . [23]

1. We should train the model on a large portion of the dataset. Otherwise we'll fail to read and recognize the underlying trend in the data. This will eventually result in a higher bias
2. We also need a good ratio of testing data points. As we have seen above, less amount of data points can lead to a variance error while testing the effectiveness of the model

3. We should iterate on the training and testing process multiple times. We should change the train and test dataset distribution. This helps in validating the model effectiveness properly

For above configuration the model was evaluated using K-Fold evaluation metrics with 89% accuracy. K-fold cross validation is one way to evaluate the model with the test data. The data set is divided into k subsets, and the epochs are run k times. The process of this method is to divide the data into k subsets and epochs run on total k times specified during the KFold initialization process. This approach will give you the option to choose test size and how trails you average over. [23]

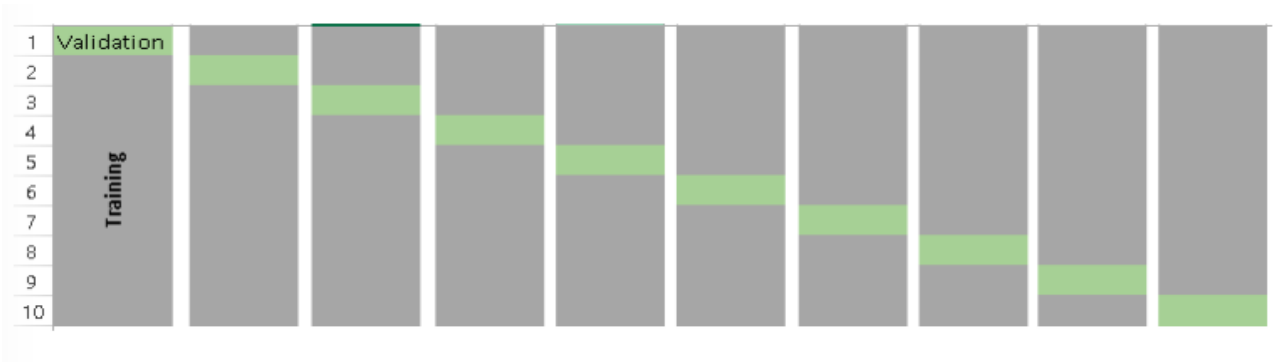


Figure 25 K-Fold for 10 splits

```
In [*]: def build_network():
            model = Sequential()
            model.add(Dense(output_dim=184,init='uniform',activation="relu",input_dim=360))
            model.add(Dense(output_dim=184,init='uniform',activation='relu'))
            model.add(Dense(output_dim=9,init="uniform",activation="softmax"))
            model.compile(loss="categorical_crossentropy",optimizer='adam',metrics = ['accuracy'])
            return model

        model = KerasClassifier(build_fn=build_network,epochs=100,batch_size=50)
        accuracies = cross_val_score(estimator=model,X=x_train,y=y_train, cv=10,n_jobs=-1)
```

Figure 26 Code snippet of KFold validation

Each time, the datasets have to be divided for training and evaluating the model. Among the available k subsets, k-1 subsets will be used for training the model whereas the remaining one set will be used for evaluating the model. Later the average of the training subsets and the evaluation subset will be analyzed to compute its efficiency. The benefit of the aforementioned approach is that the division of data as training and test set does not cause any data loss because each data object will be treated as training set k-1 times and as a test set once.

```
In [14]: accuracies
Out[14]: array([0.90044114, 0.89512733, 0.89812493, 0.89812493, 0.89932818,
                0.88960192, 0.89682141, 0.89932818, 0.89872656, 0.89411411])

In [15]: accuracies.mean()
Out[15]: 0.8969738698209891

In [16]: accuracies.std()
Out[16]: 0.0030787372480128453
```

Figure 27 KFold Accuracies

Model evaluation is a process of running the model on test to check the performance accuracy and how predicts on unseen data.

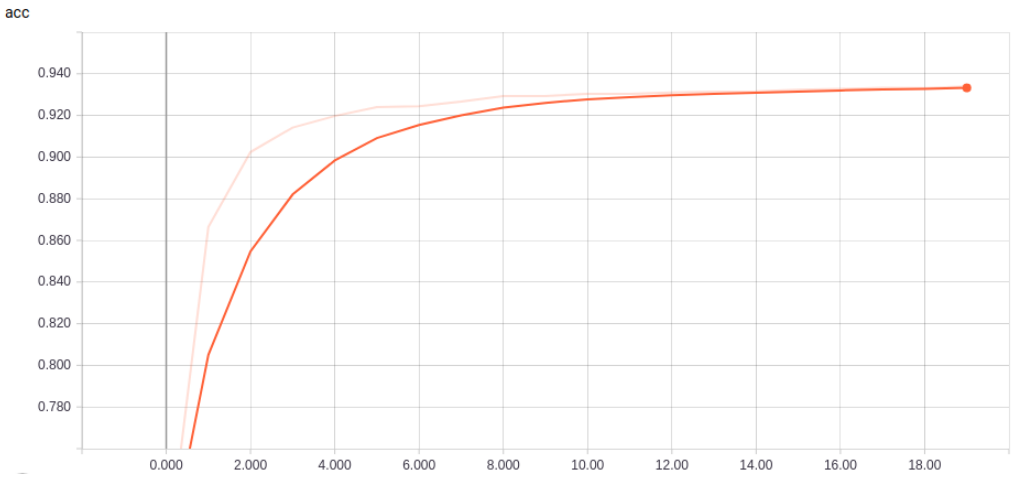


Figure 28 Accuracy on Trained Model

Model was evaluated on a quick run with 20 epochs which never converged at any point when tested on 100 epochs. Model was successfully evaluated on the test data at 0.93 accuracy as shown in above Figure 27 and Figure 28 illustrates the loss 0.19 Loss

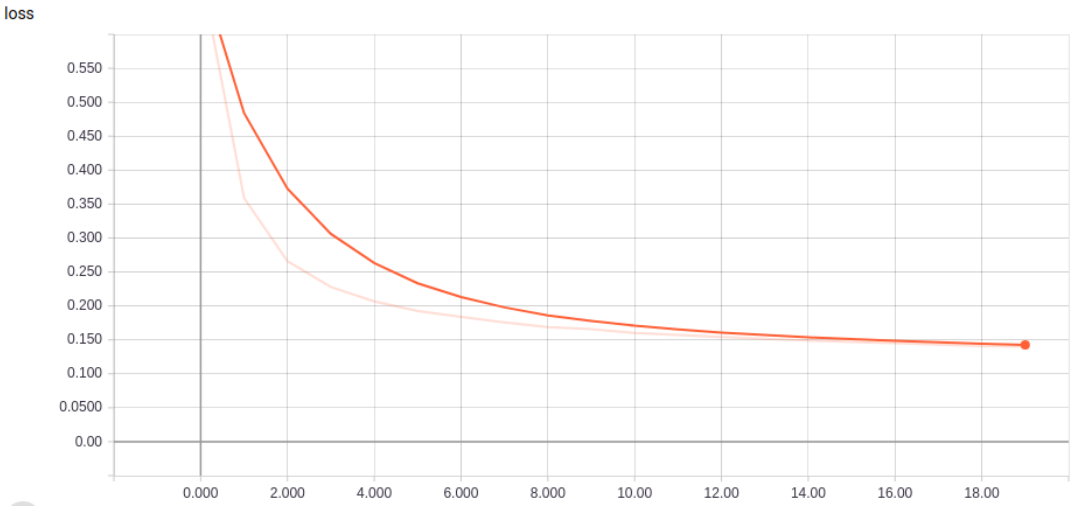


Figure 29 Loss function on model trained

Model validation on the training data:

Keras has an inbuilt validation function during the call of fit method where we can separate an amount of data for validation on training set. This validation set will be used in the to evaluate the model performance during the training phase.

Validation accuracy

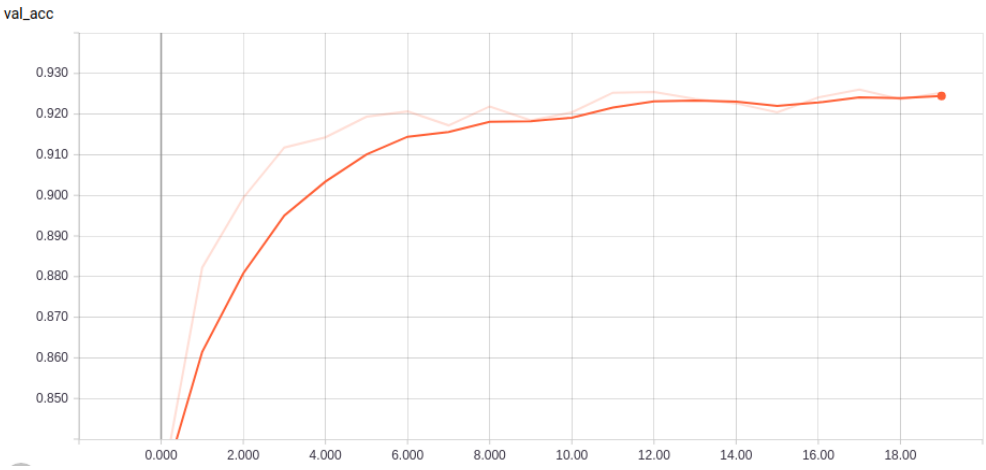


Figure 30 Validation Accuracy

Model has validated over 93.06 % of performance accuracy on validation spilt data and 16% loss

Validation Loss

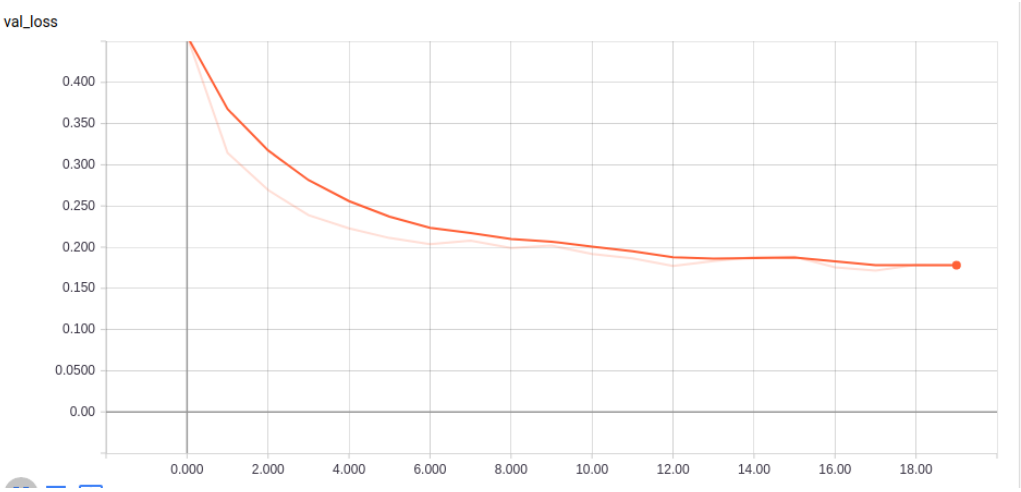


Figure 31 Validation Loss

3.4.4 Confusion matrix

The confusion matrix is a well-known technique to understand and evaluate the classification model based on the visual mappings between the predicted classes and the actual classes. For instance, the performance of the neural network may benchmark the classification of dogs and cats but may literally be bamboozled when the classification takes places between dogs and wolves.

Analysis of overall dataset to predict the confusion matrix to study the false positives and true negatives.

Table 11 Dataset Samples based on categories

Directions	Samples count
Up	85517
Left	81297
Right	68276
Up-Right	5173
Up-Left	2174
Down	1313
Bottom-Right	849
Bottom-Left	538

Confusion metrics

To begin with, we sampled the results using SoftMax classifier for decision making; we gathered over 245000 samples of lidar data into 8 categories which individually represent the drive direction after dropping the standby joystick stroke which held the rest of the data.

	Bottom-left	**Bottom-Right**	**Down**	**Left**	**Right**	**Up**	**Up-Left**	**Up-Right**
Bottom-left	100	0	6	19	4	6	0	0
Bottom-right	0	155	7	3	28	17	0	2
Down	6	5	291	6	17	3	0	0
Left	37	5	7	19190	137	904	43	1
Right	0	28	3	138	16078	650	2	170
Up	7	17	7	1639	787	18790	79	170
Up-Left	0	0	0	138	38	98	267	3
Up-Right	0	2	0	42	607	121	2	519

Table 12 Confusion matrix values

Based on the data collection samples and the training phase parameter tuning resulted in 90% of accuracy. Though the prediction angles would perform as expected to move the robot forward and turn right and left, there is still an amount of miss classification and a low chance of generating totally opposite direction and slight turning error precisions.

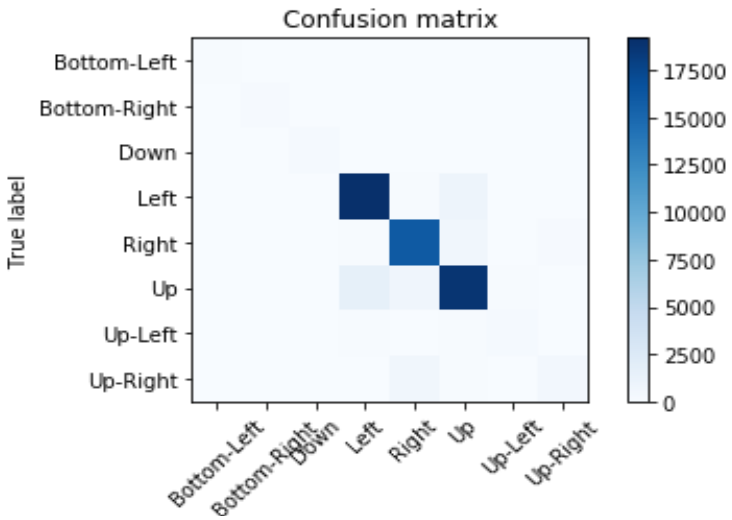


Figure 32 Confusion Matrix for entire dataset

In our purpose, due to the imbalanced dataset for directions. We further planned to drop the other directions and mainly focused on having the data which can drive the robot around.

Data Filtering.

We filtered all the unwanted directions to make the model more robust and avoid bias conditions.

k-fold validation results

```
In [18]: data_filters = dataset[dataset.joy_stroke != ('Standby')]
         data_filters_1 = data_filters[data_filters.joy_stroke != ('Up-Right')]
         data_filters_2 = data_filters_1[data_filters.joy_stroke != ('Up-Left')]
         data_filters_3 = data_filters_2[data_filters_2.joy_stroke != ('Down')]
         data_filters_4 = data_filters_3[data_filters_3.joy_stroke != ('Bottom-Left')]
         data_filters_5 = data_filters_4[data_filters_4.joy_stroke != ('Bottom-Right')]
```

Figure 33 Data Filtering

Now after filtering, our dataset holds following fields to drive the robot with less biased conditions with just going forward taking a left turn and a right turn.

Table 13 Filter values for confusion matrix

Directions	Samples count
Up	85517
Left	81297
Right	68276

Confusion Matrix

Table 14 Confusion matrix values for filtered data

Left	19205	136	983
Right	142	16410	517
UP	1516	953	18911
	Left	**Right**	**Up**

After re-filtering the dataset, we found that still there is bit confusion between the robot to take precisive directions and confuses sometimes between taking a left and going forward**.** We can still mutually avoid this by hard cording the logics on the robot, which will be not the focus on this research.

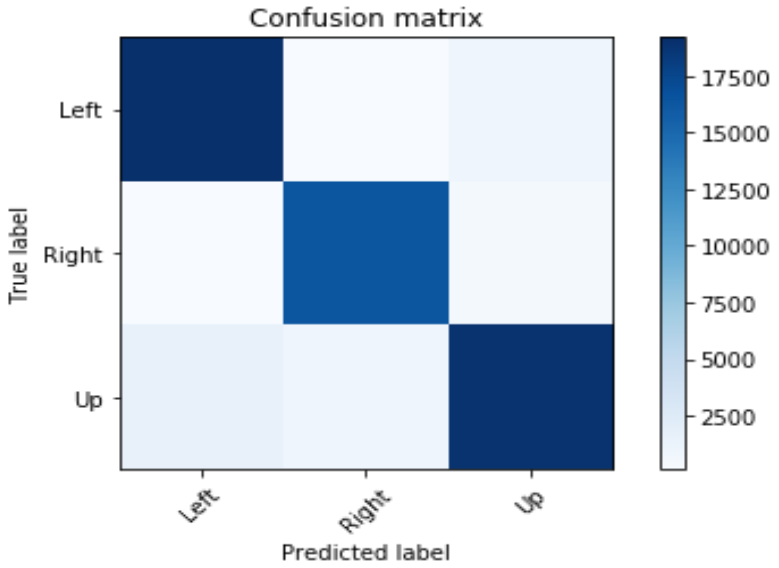
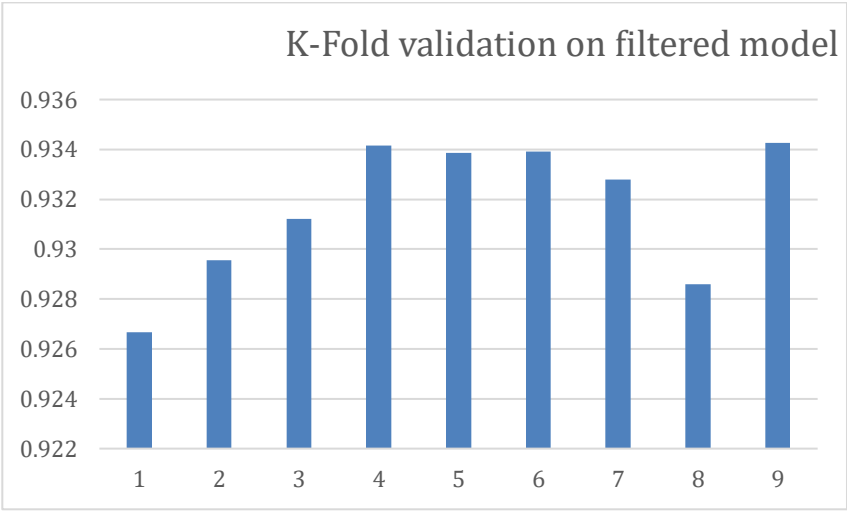


Figure 34 Confusion Matrix for filtered data

3.4.5 K-Fold validation results



We ran a K-fold validation analysis on the data which was filter to just check its model performance it was significantly good with a standard deviation of 0.93 and mean value of 0.0026.

3.5 Inference:

Learning is associated with parameter estimation and is not explicitly thought of as an inference problem. Usually, an inference is more like making some prediction. For example, in linear regression, given some features and some learned parameters, we want to predict some real-valued variable. Learning the values of the latent variables for a specific example is inference. Fitting the "hyperparameters" of the model for all the examples is learning.

Inference can't happen without training. So, the model is first trained on Floyd Hub cloud. The trained model is transferred to a client machine for real-time driving classification based on the LDS Sensor readings.

Deployment:

NVIDIA DIGITS is used to interactively train network models on annotated datasets in the cloud or PC, while TensorRT, inference optimizer that delivers low latency and high- throughput for deep learning applications. Jetson are used to deploy runtime inference in the field. TensorRT uses graph optimizations and half-precision FP16 support to more than double DNN inferencing. Together, DIGITS and TensorRT form an effective workflow for developing and deploying deep neural networks capable of implementing advanced AI and perception. [13]

We deployed the trained model to Nvidia's Jetson TX2, which is especially designed for embedded systems intelligence. Nvidia's Jetson has 256 CUDA Cores which helps inferencing on the fly in parallel. Jetson is loaded with Ubuntu 16.04 LTS with Robot operating system and kernel is modified to meet the needs of turtlebot.

Figure 35 Nvidia Jetson

3.5.1 Sequence for Inference process

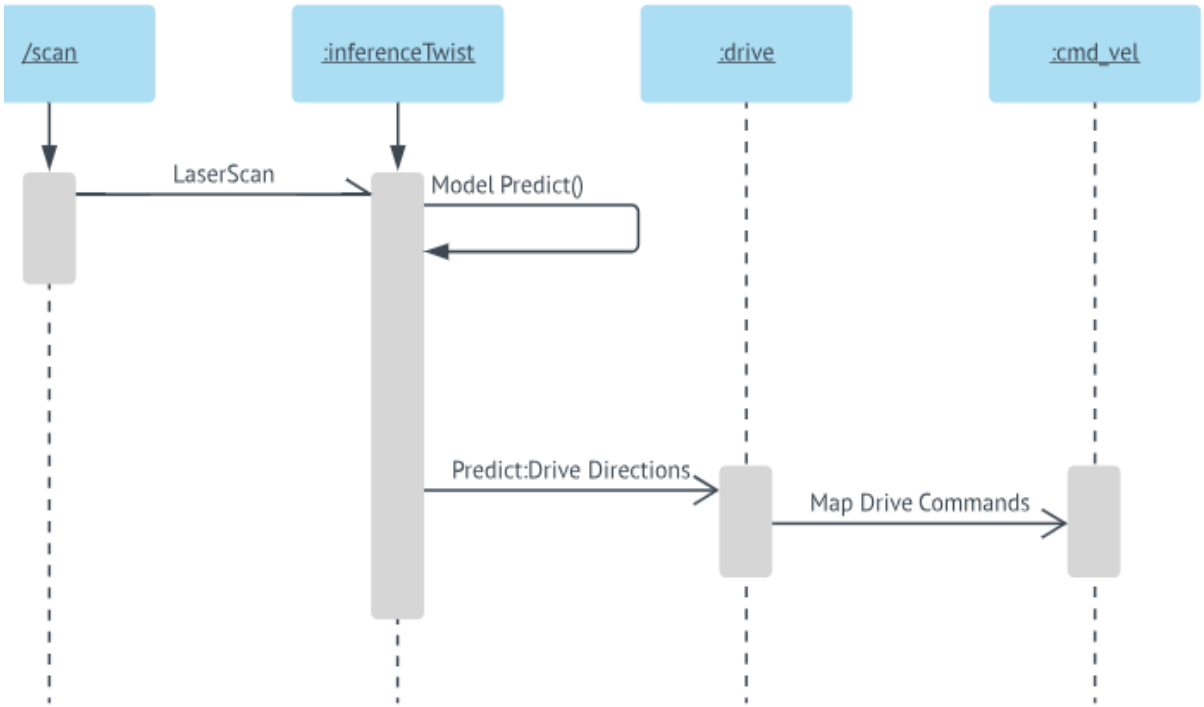


Figure 36 Sequence Diagram for inference process

The process of inferencing is represented sequentially in the above UML diagram.
The first object in the sequential diagram is the *scan* object that has an activation box where the execution begins. Next, we have the *inference Twist* object that has the ModelPredict () method that is looped with itself to make decisions on to predict the drive directions. The drive directions will do a synchronous call to the subsequent *drive* object that will map the drive commands the node cmd_vel.

Consider the scenario where the neural network model comes across an obstacle such as a wall. The best course of action will be to turn right or left depending of the space and the available node information. At first, the model will *scan* for the data using the LDS Laser Scanner that will fetch the information about all the 360-degree mapping to the obstacle. Upon detecting an obstacle the *inference Twist* node will process that data and use *ModelPredict()* to make decisions based on the knowledge gained by training datasets. The function is looped to itself to map the data with relevant directions so that the robot can predict accurate drive directions that will be passed as a synchronous signal to the *cmd_vel node*. The node *cmd_vel* will subscribe to *inferenceTwist* that will convert the high-level messages to lower machine level velocities. The system will take the linear velocity of 1 m/s and angular velocity of 2 radians/degree that will be translated to machine code for the robot to drive around in the space.

ROS neural network architecture licenses decoupled operation. In such systems, the names are the chief medians through which more extensive and exceedingly complicated networks can be constructed. These names possess a significant role in the architecture: nodes, topics, services, and parameters all have names. All ROS client libraries support command-line remapping of names. This which means a compiled program can be reconfigured at runtime to operate in a different Computation Graph topology. [22]

For instance, to measure a LDS Sensor laser rangefinder, start the lds_node driver, that communicates with the laser and publishes sensor_msgs/LaserScan messages on the scan topic. Write a node using laser_filters that subscribes to the messages on the scan topic to process the scanned data. The filter will automatically begin to receive messages from the laser after subscription.

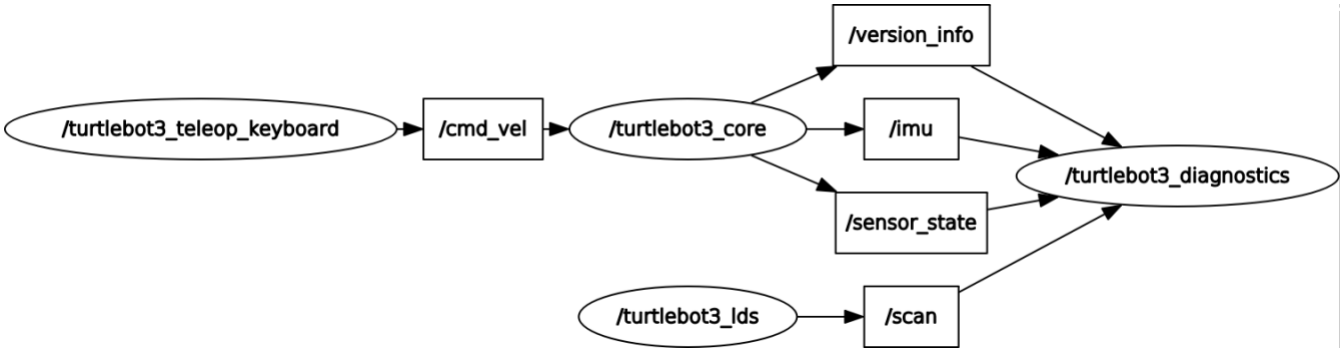


Figure 37 ROS Graph for turtlebot launch

Observe how these pair nodes happen to be decoupled. The turtlebot3_lds node always publishes scans notwithstanding to the fact if any node is subscribed to it or not. Whereas, the filter nodes always are subscribed to scans notwithstanding to the fact whether or not any node is publishing it. These two nodes can be started, killed, and restarted, in no specific order, without producing indefinite erroneousness conditions.

Subsequently, the addition of an extra laser to the robot is comparatively less complicated. To do so, we have to reconfigure the system and remap the names that are used. Restart the turtlebot3_lds and remap the scan topic to base_scan and also the filter node to base_scan. After starting, the lds_node and filter node will stop hearing messages on the scan topic but instead, communicate of the base_scan topic. Later we can ignite different lds_node for the latest laser rangefinder.

Pipeline: Multiple estimators are chained into one using a pipeline. A pipeline is advantageous as there usually happens to be a fixed series of moves in processing the data, for instance, feature selection, normalization, and classification. Pipeline works for two goals here:

- Convenience and encapsulation: to fit a complete sequence of estimators, the methods fit and predict have to be called just once on the data.

- Joint parameter selection: A pipeline is a chain of estimators this also allows us to grid search across parameters of every estimator within the pipeline at once.

Pipelines help avoid leaking statistics from the test data into the trained model in cross-validation, by ensuring that the same samples are used to train the transformers and predictors.
Every estimator inside the pipeline, besides the last estimator, has to be transformers (that is, the pipeline should always have a transform method) whereas, the last estimator can be any type (transformer, classifier, etc.,).

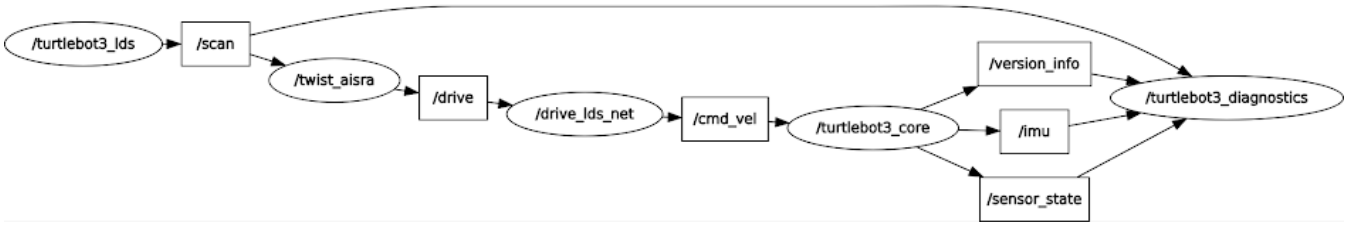


Figure 38 ROS Launch for inference for turtlebot

In the above figure 38, Under inference node twist-aisra the data is filtered and preprocessed into a scikit-learn's Pipeline which applies MinMax scaling to the data and reshape it to the Keras model's architecture and published to drive topic.

The Probabilities from the classifier are further published to other node to map the probabilities to the drive commands. Once the key mapping in drive_lds_net is done, the drive velocities and angular ratios are published to cmd_vel to drive the robot around to avoid obstacles.

3.5.2 Performance measures

As the end to end model was training to deploy and inference on low level ARM based system. It is really important to measure its performance attributes in real time data. Nvidia Jetson TX2 comes inbuilt with 256 CUDA cores and its architecture supports inferencing with GPU and CPU together.

Jetson TX2 embeds GPU and CPU in a single cluster, CPU has a dual core Denver 2 processor and 4 cores of ARM cortex A57. So along with the GPU and 6 cores of CPU, the efficiency is similar to the full-scale desktop performance. Depending on the application requirements and power factor, one can take the cores on and turn it off.

There are five available modes in nvpmodel as proposed by Nvidia

Table 15 NVP Modes of Nvidia Jetson TX2

MODE	MODE NAME	DENVER 2	FREQUENCY	ARM A57	FREQUENCY	GPU FREQUENCY
0	Max-N	2	2.0 GHz	4	2.0 GHz	1.30 GHz
1	Max-Q	0		4	1.2 GHz	0.85 GHz
2	Max-P Core-All	2	1.4 GHz	4	1.4 GHz	1.12 GHz
3	Max-P ARM	0		4	2.0 GHz	1.12 GHz
4	Max-P-Denver	2	2.0GHz	0		1.12GHz

1) **Max Q**: Max Q uses all the cores of the ARM A57 at the optimal clock speed, it sets the power profile to 7.5W. So, it uses half the power of available in Jetson TX2.
2) **Max P**: It uses only cores of ARM A57, but with faster clock rates. The power profiles set up to 15W
3) **Max N**: it uses all the cores including ARM A57 and Nvidia Denver, Max N architecture strikes the clock field to the maximum clock speeds.

The test comparison between Dell workspace system vs Jetson tx2 for 10 instances of a dataset from the LDS AISRA prediction pipeline. During the testcases, there were iterations ran over available various modes of nvpmodel available on the same dataset.

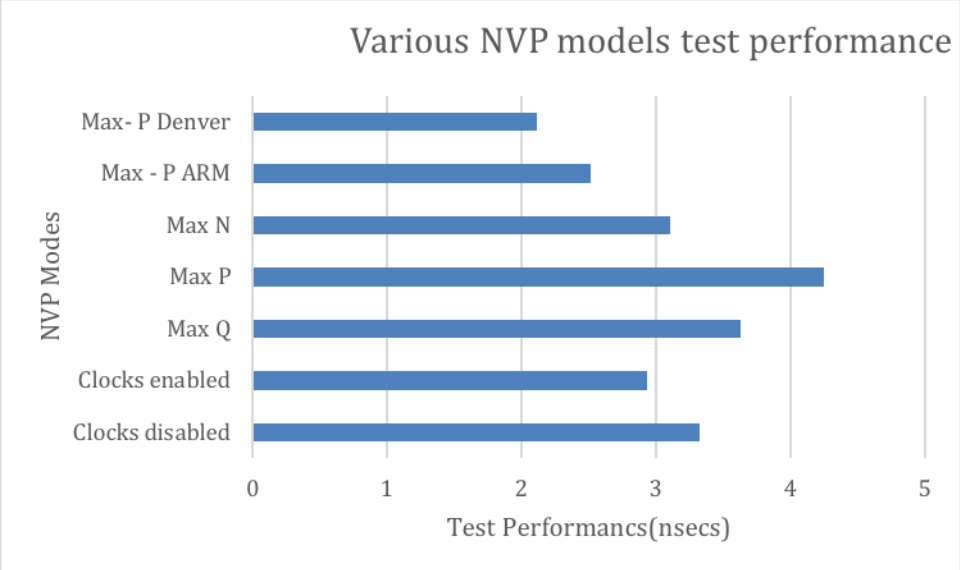


Figure 39 NVP Mode test performance

Table 16 Performance Metrics

Mode Name	Test performance
Clocks disabled	3.326122586021
Clocks enabled	2.93640089035
Max Q	3.6240298748
Max P	4.24526786804
Max N	3.10444998741
Max P ARM	2.51240801811
Max p- Denver	2.1092903033

According to the test results, Max -P Denver outperforms all the other combinations with executing 10 queries at 2.1 seconds where each incoming data can be predicted at 0.210 seconds

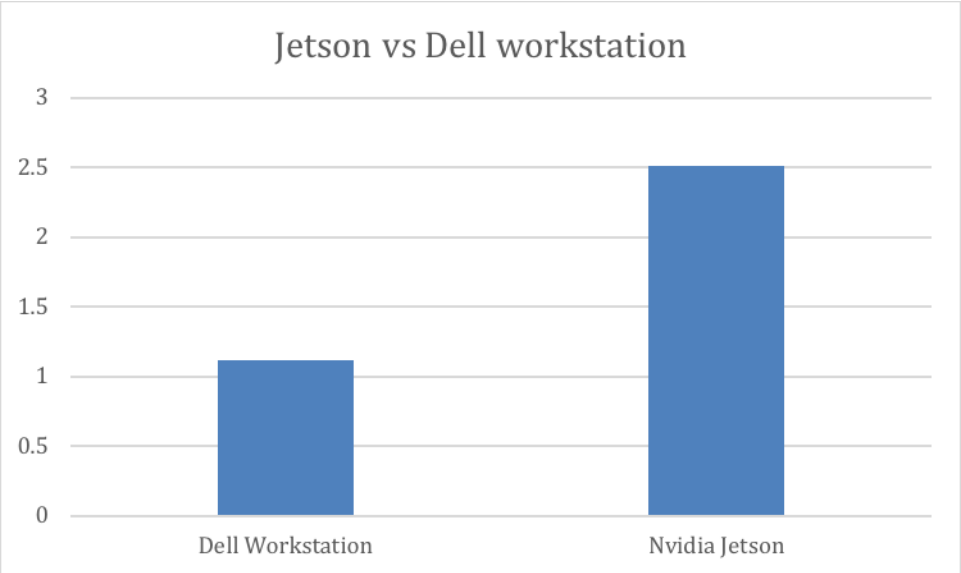


Figure 40 Dell vs Jetson TX2 Inference comparison

The same dataset was used to test inference prediction on dell and also Jetson, where dell workstation has i7 processor which is pretty nominal to execute the inference at 1.11 seconds for 10 data samples. But Jetson gives out 2.5 seconds for 10 instances which is better inference rate for an ARM processor

CONCLUSION

In this paper, we proposed the design of an autonomous interface using Neural Network that helps a mobile robot to navigate in space autonomously using obstacle avoidance approach in indoor environments. The system is based on a complex MLP architecture and accomplished desired results in real-world demonstrations. The thoughtfully conducted experiments and detailed analysis prove that our system could successfully manage obstacle avoidance.

The comparisons between resultant robot decisions and human decisions for obstacle avoidance exhibited high similarity. Nonetheless, there are still some limitations, when the ranges of different LDS scanners scale more than the trained ranges. This trained model can be various robots which support our minimal, but still robotic applications [11]. Moreover, a discrete classification which may render the angular velocity and linear velocity may not be precise enough for a continuous state space of the decisions.

For further investigation on the subject:

- A reinforcement learning irrespective of the space to learn autonomous navigation and obstacle avoidance will be developed.
- A CNN architecture will be concatenated to an existing model which can be adopted from the existing literature "A Deep-Network Solution Towards Model-less Obstacle Avoidance" [17] by Lei Tai. Such an add-on will make the model more robust and works with more accuracy.

FUTURE SCOPE

- Exploration can be automated based on the model to navigate autonomously just by infusing aisra into any robot irrespective to its geometry
- Process of SLAM can be executed without human intervention by setting the map coordinates.

REFERENCE

1. Gulli, A. and Pal, S. (2017). Deep Learning with Keras. 1st ed. Birmingham: Packt Publishing, pp.11-106.ISBN:978-1-787122-842-2

2. Quigley, Morgan, Gerkey, Brain and D Smart, William, 2017, Programming Robots with Ros. 2. : ShroffPpublishers & distr. ISBN: 978-93-5213-279-9

3. Geron, Aurelien, 2017, Hands-on machine learning with Scikit-Learn and TensorFlow. 2. Beijing [etc.] : O'Reilly. ISBN: 978-935213-521-9

4. Cross Validation, 2018. Cs.cmu.edu [online],[accessed 2018-05-21] Accessed on https://www.cs.cmu.edu/~schneide/tut5/node42.html

5. Stergiou, Chistos and Siganos DIMITRIOS, 2018, Neural Networks. Doc.ic.ac.uk [online]. 2018. [Accessed 21-May-2018].Available from: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

6. Sammut, Claude, 2018, Encyclopedia of Machine Learning and Data Mining | Claude Sammut |Springer. Springer.com [online]. 2018. [Accessed 21May2018]. Available from: https://www.springer.com/in/book/9781489976857

7. BROWNLEE, JASON, 2018, How to Grid Search Hyperparameters for Deep Learning Models in Python With Keras. Machine Learning Mastery [online]. 2018. [Accessed 19 May 2018]. Available https://machinelearningmastery.com/grid-search-hyperparameters-deep-learning-models-python-keras/

8. CHOLLET, FRANÇOIS, 2018, keras-team/keras. GitHub [online]. 2018. [Accessed 15 May 2018]. Available from: https://github.com/fchollet/keras

9. ACADEMY, ROBOT, 2018, Robot Ignite Academy. Robotigniteacademy.com[online]. 2018. [Accessed 20 May 2018]. Available from: https://www.robotigniteacademy.com/en/course/ros-navigation-in-5-days/details/

10. ZIEBA, KAROL, DEL TESTA, DAVIDE, DWORAKOWSKI, DANIEL and FIRNER, BERNHARD, 2016, End to End Learning for Self-Driving Cars. arXiv:1604.07316. 2016.

11. Lei Tai, Shaohua Li, Ming Liu. "A deep-network solution towards model-less obstacle avoidance", 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016

12. PEDREGOSA, FABIAN, VAROQUAUX, GAËL, GRAMFORT, ALEXANDRE, MICHEL, VINCENT, THIRION, BERTRAND, GRISEL, OLIVIER, BLONDEL, MATHIEU, PRETTENHOFER, PETER, WEISS, RON, DUBOURG, VINCENT, VANDERPLAS, JAKE, PASSOS, ALEXANDRE, COURNAPEAU, DAVID, BRUCHER, MATTHIEU, PERROT, MATTHIEU and DUCHESNAY, ÉDOUARD, 2018, Scikit-learn: Machine Learning in Python. Jmlr.csail.mit.edu [online]. 2018. [Accessed 20 May 2018]. Available from: http://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html

13. dusty-nv/jetson-inference, 2018. Nvidia Jetson Inference [online] Accessed on [18 May 2018], Available from: https://github.com/dusty-nv/jetson-inference

14. SHARMA, SAGAR, 2018, Activation Functions: Neural Networks – Towards Data Science. Towards Data Science [online]. 2018. [Accessed 20 May 2018]. Available from: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6

15. Welcome — Theano 1.0.0 documentation, 2018. Deeplearning.net [online], 2018 [Accessed on 18 May 2018] Available from : http://www.deeplearning.net/software/theano/

16. ROBOTIS, 2018. ROBOTIS e-Manual [online],2018[Accessed on 19 May 2018] Available from: http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_lds_01/#appendix-lds01

17. News of Robotics and Manipulation Lab [online].2018[Accessed on 13 May 2018] Available from: http://ram-lab.com

18. Tuning the hyper-parameters of an estimator documentation[online].2018[Accessed on 21 May 2018] http://scikit-learn.org/stable/modules/grid_search.html

19. Marcelo Becker, Carolina Meirelles Dantas, Weber Perdigão Macedo, "Obstacle avoidance procedure for mobile robots",2005,ABSM,Symposium Series in Mechatronics Vol 2 pp 250-257

20. Jurgen Schmidhuber ,"Deep learning in neural networks-overview", 2014 The Swiss AI Lab IDSIA, University of Lugano & SUPSI

21. K. Wyrobek, E. Berger, H.F.M. Van der Loos, K. Salisbury, "Towards a Personal Robotics Development Platform: Rationale and Design of an Intrinsically Safe Personal Robot," 2008 IEEE ICRA

22. Husarian Robotics [online] 2018 [Accessed on 28 may 2018] https://husarion.com/tutorials/ros-tutorials/7-path-planning/#7-path-planning-requirements-regarding-robot

23. James Ng. "An Analysis of Mobile Robot Navigation algorithms in unknown environments ", 2010, Seek wisdom school of electrical, electronic and computer engineering

24. Aishwarya A. Panchpor, Sam Shue, James M. Conrad, "A survey of methods for mobile robot localization and mapping in dynamic indoor environments", 2018,Signal Processing And Communication Engineering Systems (SPACES), pp. 138-144, 2018

25. Model performance - Jetson Hacks, [online] 2018[aAcessed on 1 june 2018] https://www.jetsonhacks.com/2017/03/24/caffe-deep-learning-framework-nvidia-jetson-tx2/

APPENDIX

1) Hyper Parameter tuning

EPOCH	KERNEL	OPTIMIZER	BATCH SIZE	Accuracy	Loss
200	glorot_uniform	rms_prop	32	0.907067	0.00588
200	glorot_uniform	adam	32	0.963467	0.00266
200	normal	rms_prop	32	0.907067	0.00588
200	normal	adam	32	0.9632	0.002286
200	uniform	rms_prop	32	0.907067	0.00588
200	uniform	adam	32	0.9636	0.002673
200	glorot_uniform	rms_prop	64	0.965867	0.002407
200	glorot_uniform	adam	64	0.9628	0.00204
200	normal	rms_prop	64	0.9276	0.006299
200	normal	adam	64	0.964	0.001987
200	uniform	rms_prop	64	0.6176	0.413529
200	uniform	adam	64	0.963067	0.002927
200	glorot_uniform	rms_prop	128	0.942933	0.012438
200	normal	rms_prop	128	0.964267	0.002175
200	uniform	rms_prop	128	0.920667	0.014611
150	glorot_uniform	rms_prop	32	0.964133	0.003172
150	glorot_uniform	adam	32	0.907067	0.00528
150	normal	rms_prop	32	0.9644	0.003116
150	normal	adam	32	0.963733	0.002853
150	uniform	rms_prop	32	0.907067	0.00528
150	uniform	adam	32	0.963467	0.002494
150	glorot_uniform	rms_prop	64	0.9424	0.02235
150	glorot_uniform	adam	64	0.964267	0.00262
150	normal	rms_prop	64	0.913867	0.006369
150	normal	adam	64	0.964133	0.002778
150	uniform	rms_prop	64	0.914	0.006532
150	uniform	adam	64	0.963733	0.002473
150	glorot_uniform	rms_prop	128	0.959333	0.001969
150	normal	rms_prop	128	0.918133	0.013147
150	uniform	rms_prop	128	0.323333	0.412008
100	glorot_uniform	rms_prop	32	0.907067	0.00528
100	glorot_uniform	adam	32	0.963467	0.003523
100	uniform	rms_prop	32	0.907067	0.00528
100	uniform	adam	32	0.963333	0.002739
100	normal	rms_prop	32	0.907067	0.00528
100	normal	adam	32	0.964133	0.003172
100	glorot_uniform	rms_prop	64	0.954533	0.005555
100	glorot_uniform	adam	64	0.963867	0.00264

100	normal	rms_prop	64	0.6312	0.423161
100	normal	adam	64	0.963867	0.00262
100	uniform	rms_prop	64	0.9132	0.009488
100	uniform	adam	64	0.9648	0.000864
100	glorot_uniform	rms_prop	128	0.961733	0.003104
100	normal	rms_prop	128	0.924533	0.009635
100	uniform	rms_prop	128	0.319733	0.411163