



Peter the Great St. Petersburg Polytechnic University  
Institute of Computer Sciences and Technologies  
School of Cyber-Physical Systems and Control

## Coursework

Lane change decision making for Autonomous vehicles - A Knowledge base approach

Discipline Knowledge Engineering and Knowledge Management

Student

Chintha Sai Tejeswar Reddy

Group 3540901/01701

Supervisor

Assoc. Prof, Ph.D.

Vadim A. Onufriev

13.05.2021

# Contents

<b>Introduction .....</b>	<b>4</b>
<b>Process Description .....</b>	<b>5</b>
Lane Detection Neural Network .....	5
Object Detection Neural Network.....	6
Decision Making Neural Network.....	6
IDEF0 Diagram.....	6
Objective function.....	9
<b>Model Development .....</b>	<b>10</b>
Lane Detection NN.....	11
Object Detection NN <sup>[5]</sup> .....	11
Mathematical Model.....	13
Decision Making NN .....	17
Training the custom made NN .....	17
Concept Map .....	20
<b>Methodology.....</b>	<b>20</b>
UML Diagram .....	21
Swimlane Diagram.....	21
<b>Software Implementation .....</b>	<b>25</b>
Neural Network Interface .....	26
Knowledge Base .....	28
Inconsistency Checking .....	32
Rules in KB – Inconsistency check & Inference .....	33
Automatic Knowledge Base ANN ( KBANN ).....	36
<b>Testing and Results .....</b>	<b>39</b>
<b>Conclusion and Further Developments .....</b>	<b>43</b>
<b>Appendix .....</b>	<b>46</b>
<b>References.....</b>	<b>46</b>

## List of Figures

Figure 1: IDEF0 Node A0	6
Figure 2: IDEF0 Node A0 – Decomposition	7
Figure 3: IDEF0 Node A2	7
Figure 4: IDEF0 Node A3	8
Figure 5: IDEF0 Node A4	8
Figure 6: Lane Dataset Sample images	11
Figure 7: Mask RCNN COCO output image	12
Figure 8: Mask RCNN COCO masks	12
Figure 9: Mask RCNN COCO Output with masks	12
Figure 10: Equations of Lanes	13
Figure 11: MATLAB code snippet for drawing lanes	14
Figure 12: Left lane and Right lane marked	15
Figure 13: Right lane and Immediate right lane marked	15
Figure 14: Right lane and Immediate left lane marked	15
Figure 15: Plots of different lanes in Excel	16
Figure 16: Concise plot of all lanes	16
Figure 17: Decision Making Neural Network	17
Figure 18: Training data for Decision Making NN	18
Figure 19: Code of initial build of Decision Making NN	18
Figure 20: Accuracy vs Epochs and Loss vs Epochs graphs	19
Figure 21: Concept map of the system	20
Figure 22: UML Diagram	21
Figure 23: Swimlane Diagram	22
Figure 24: Web application interface	25
Figure 25: Neural Network form created in C#	26
Figure 26: Initial build of Manual Neural Network testing interface	27
Figure 27: Latest build of Manual Neural network testing interface	27
Figure 28: URIs used in KB	28
Figure 29: Turtle code of processes in ontology	28
Figure 30: Snippet of some classes used in KB	28
Figure 31: Ontology graph of some classes in KB	29
Figure 32: Function names in Ontology	31
Figure 33: Disjoint classes in KB	32
Figure 34: Inconsistency in Ontology	32
Figure 35: OWL Rule in KB	34
Figure 36: Knowledge Base changing	38
Figure 37: Calibration Images	39
Figure 38: Camera Parameters	39
Figure 39: Lane detection Result sample 1	40
Figure 40: Lane detection Result sample 2	40
Figure 41: Object Detection result sample 1	41
Figure 42: Object Detection result sample 2	41
Figure 43: Recording User response interface sample image 1	42
Figure 44: Recording User response interface sample image 2	42

## Introduction

Knowledge engineering (KE) refers to all technical, scientific, and social aspects involved in building, maintaining, and using knowledge-based systems <sup>[1]</sup>. Knowledge engineers work to convert human expertise into what is known as a knowledge-based machine, which can mimic someone's response. These systems are used to solve complicated, high-level challenges that would otherwise need the assistance of a business specialist. Knowledge engineering has evolved because of emerging technologies such as cloud computing services, which demand vast volumes of data.

In this project a Knowledge Base is used in addition to Artificial Neural Networks to replicate a human thinking while making the decision to change lanes in an autonomous vehicle on freeways. In simple terms, Knowledge Base is a set of instructions, and/or rules and/or decisions in certain conditions which are generally written using Resource Description Framework (RDF). The Knowledge Base is written using Turtle syntax<sup>[2]</sup>. OWL<sup>[3]</sup> and SWRL<sup>[4]</sup> are the ontology languages used to develop the Knowledge Base and SPARQL is used for querying the same. A web app interface is used to view the various parts of the project which includes documentation, ontology, Neural Network training and an option to test the KBANN which makes the decision for Lane changing.

The scope of this project remains to assist the driver to take a decision in a situation where there is a need for lane changing while driving on freeway. The output of the decision can be used to control the Car to perform the action of Lane changing in case of Level-3 or more autonomous vehicles, but this is beyond the scope of this project.

This report contains Process Description followed by system model, solution, a crisp explanation of software implementation followed by results and in the end Conclusion and Further developments.

## Process Description

In general, any Lane change assist system, the Automated Vehicle ( called as Ego Vehicle from now) understands the environment through various sensors fitted on to it. A wide range of cameras can be used depending on the type of post processing of the input. For example, a Monocular Camera with a fisheye lens is one of the best options to detect lane markings and transform the image to Bird Eye view, a stereo camera is a good choice of selection when depths of the objects are key parameters while making the lane change decision. In this project a recorded video is used to test the software.

The decision-making system contains three neural networks which remain to be the heart of the whole system. These neural networks are powered by a knowledge base to make better decisions while lane changing. A pictorial representation of functions of the system are shown in the IDEF0 diagram later in the report. The system takes in a '.mp4' file as an input and the video is extracted framewise, which means every frame of the video is sent as an image to the Neural networks to detect lanes and objects.

Three Neural Networks used in this project are:

1. Lane Detection Neural Network
2. Object Detection ( mask RCNN COCO – pre trained )<sup>[5]</sup>
3. Decision Making Neural Network ( created for this project )

### Lane Detection Neural Network

This neural network is used to detect the lanes in the image and are plotted onto the image. Although there are many methods of detecting lines using Canny Edge detection, Hough transform and OpenCV library, a neural network is used in the project to detect the lanes because the neural network can be trained unlike the former and they are close to human behavioural learning.

## Object Detection Neural Network

The object detection process is done using Mask RCNN (COCO) pre trained Neural Network. This ANN can detect various range of objects out of which some are person, traffic light, cars, trucks, bicycles, horse etc. As the system requires very few types of object classes the ANN is used to detect only Cars, Trucks and Motorcycles in the test video and send their positions to Mathematical Model to check their presence on a particular lane.

## Decision Making Neural Network

This is a NN which is designed especially for lane change decision making. It takes ten inputs in the form of Binary digits which are the input neurons. The NN gives three different outputs, which are the three directions of steering namely right, left, and straight ( do not steer ). This NN is trained for different types of conditions which lie in the scope of this project.

## IDEF0 Diagram

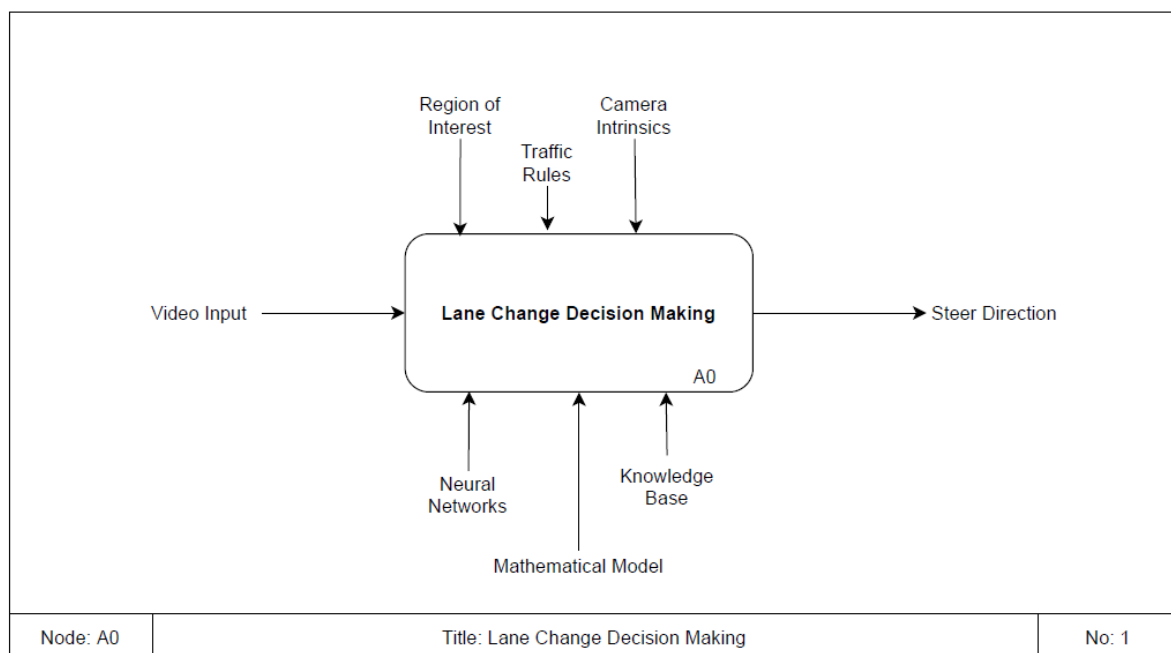


Figure 1: IDEF0 Node A0

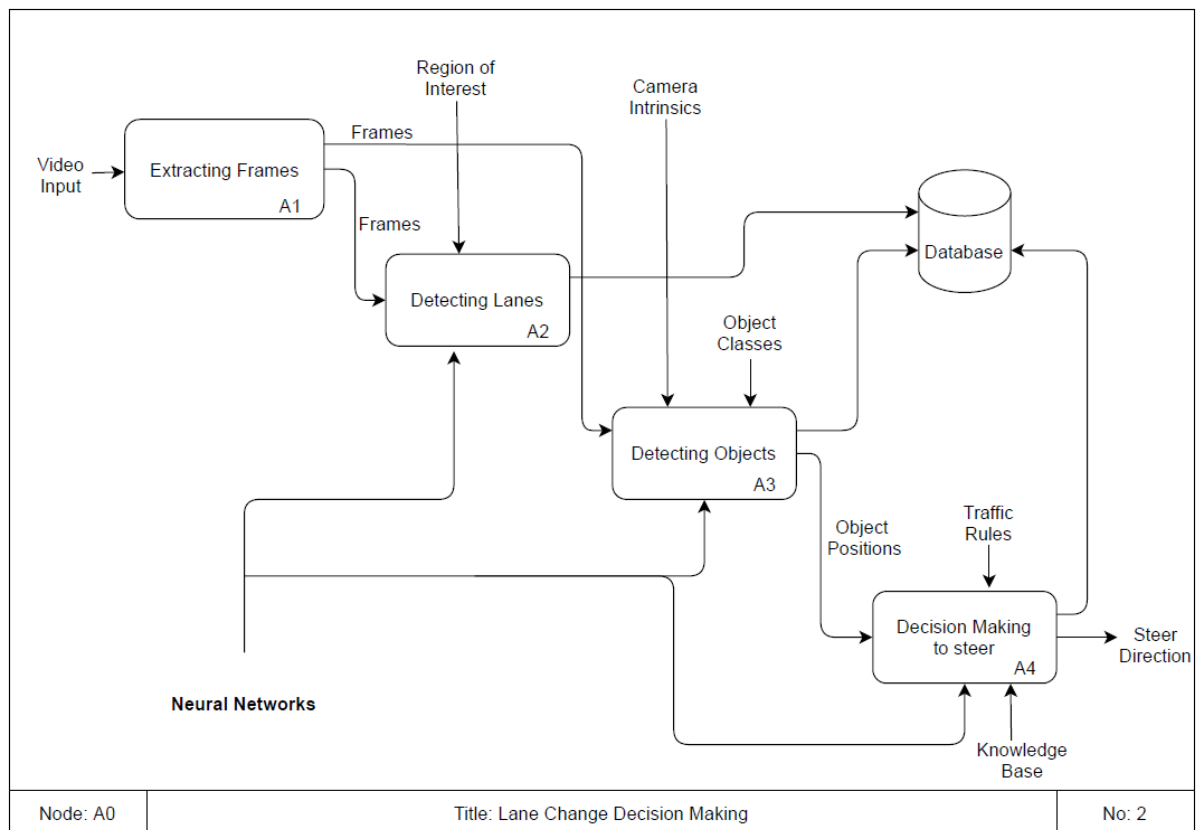


Figure 2: IDEF0 Node A0 – Decomposition

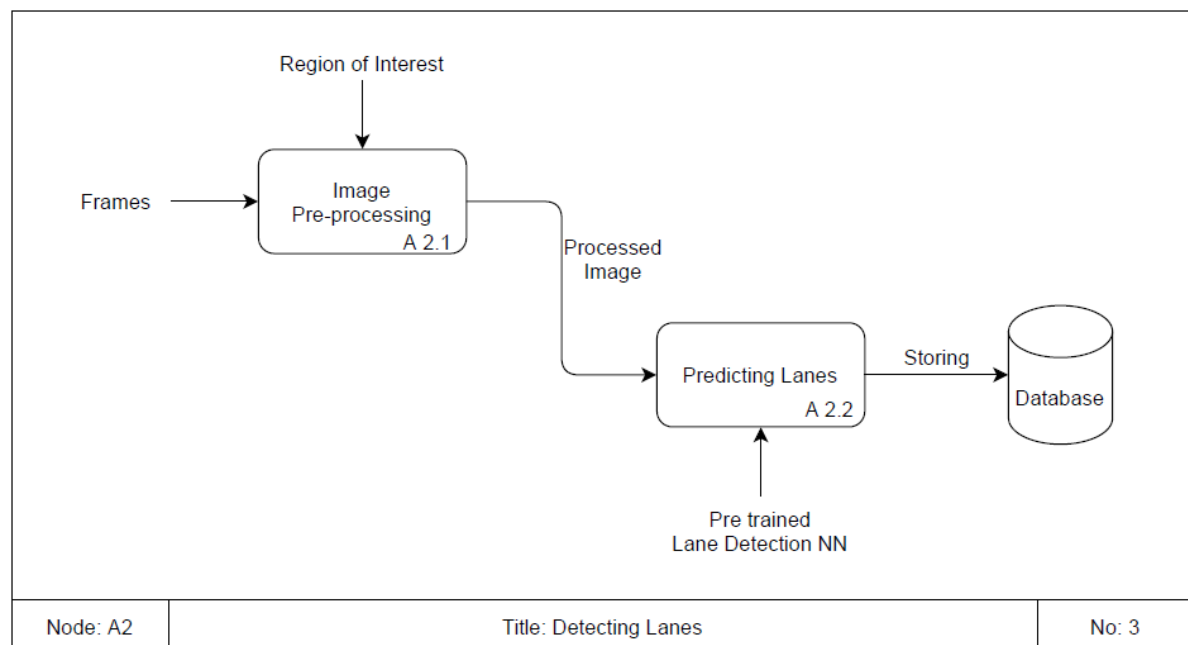


Figure 3: IDEF0 Node A2

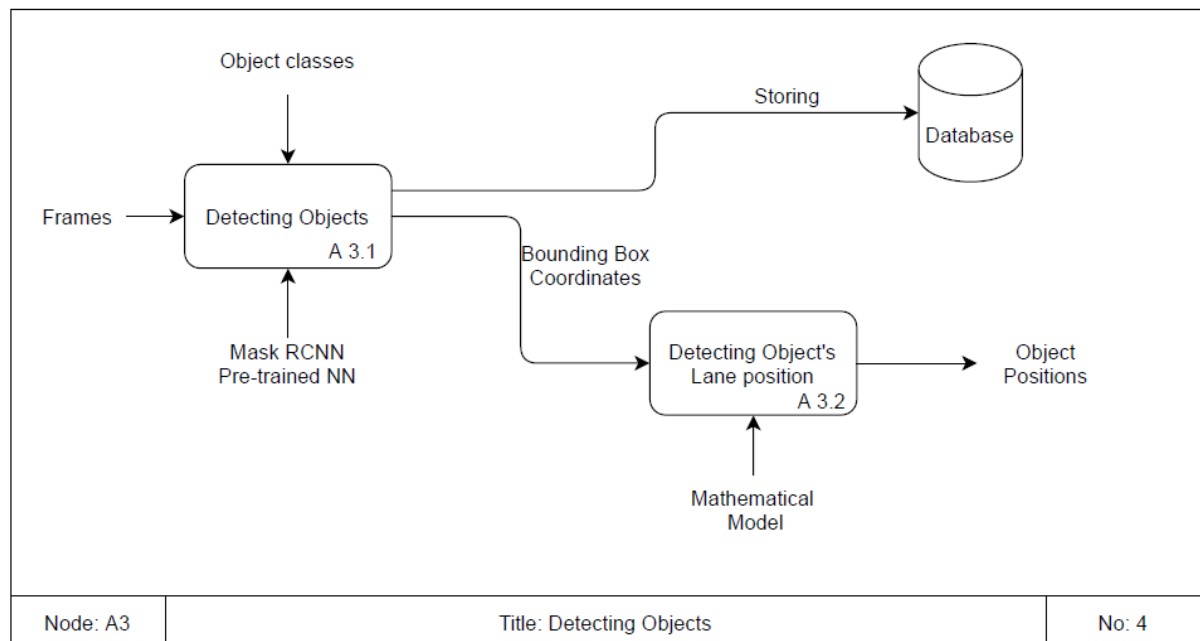


Figure 4: IDEF0 Node A3

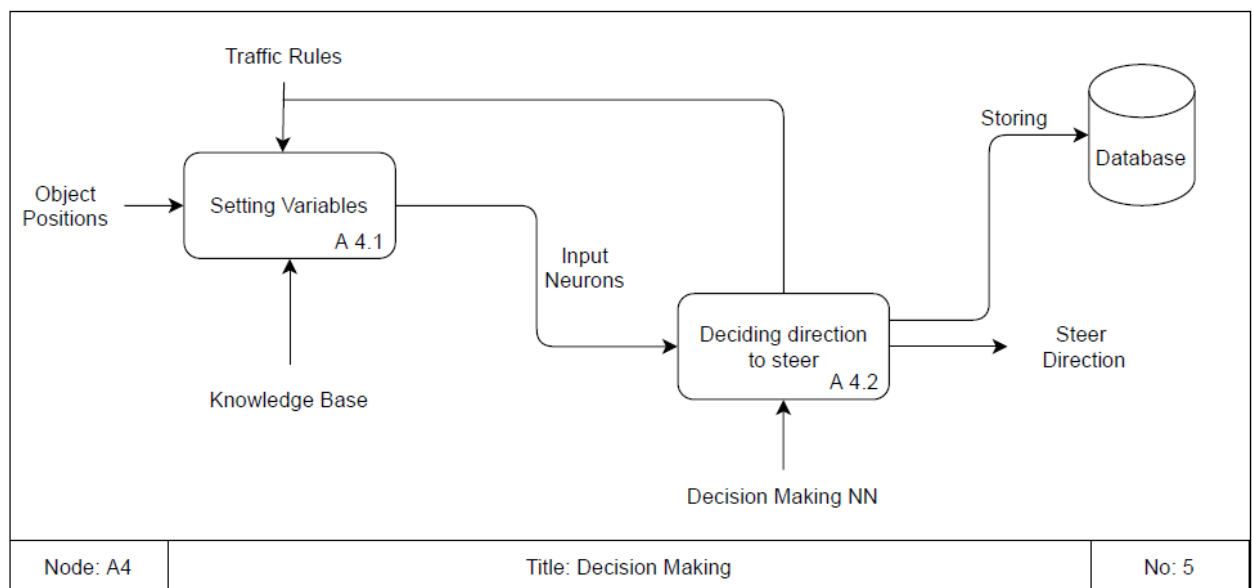


Figure 5: IDEF0 Node A4



## Objective function

*Minimize*

$$\varphi_{err} = \varphi_d - \varphi_a$$

Where:

$\varphi_{err}$  = Error

$\varphi_d$  = Desired Steering angle - obtained from the system

$\varphi_a$  = Actual Steering angle - angle achieved after manoeuvre

And

$\varphi \in (0, \pi)$  in radians

or  $\varphi \in (0, 180^\circ)$  in degrees

This is the actual objective of the project, which is to reduce the error between the desired steering angle and actual steering angle that is achieved by the system after executing the lane change manoeuvre. The system's goodness can be evaluated based on the error obtained after executing the lane change. However, the system which is developed can not be evaluated based on the above Objective function as the current model can only make decisions and executing the lane change manoeuvre is yet to be developed with additional hardware and software packages that include sensor fusion and path planning algorithms. The equation below indicates the dependencies of current decision-making system.

*"Lane Change Decision" depends on {LD, LI, LS, VL, RS, VR, LA, VA, LL, RL}*

Where :

LD – Lanes Decreasing

LI – Lanes Increasing

LS – Left Safety Distance

VL – Vehicle Left Ahead

RS – Right Safety Distance

VR – Vehicle Right Ahead

AS – Ahead Safety Distance

VA – Vehicle Ahead

LL – Left Lane Edge

RL – Right Lane Edge

The decision of changing the lane is dependent on the binary values of the above mentioned ten Input variables. The decision includes three types of outputs

- Steer Right
- Do not Steer
- Steer Left

*Table 1: Decision making dependencies and corresponding output*

LD	LI	LS	VL	RS	VR	AS	VA	LL	RL	SR	DS	SL
0	0	0	0	0	0	0	1	0	1			1
0	0	0	0	0	0	0	1	1	0	1		
0	0	0	0	0	0	1	1	0	0		1	

The table above shows a few types of how the decision is made based on the decision-making function's dependencies. As this is not an exact mathematical function, these conditions are fed to the decision-making neural network as training data. The decision of the NN can be further processed to execute lane change manoeuvre through additional hardware and software as mentioned earlier.

## Model Development

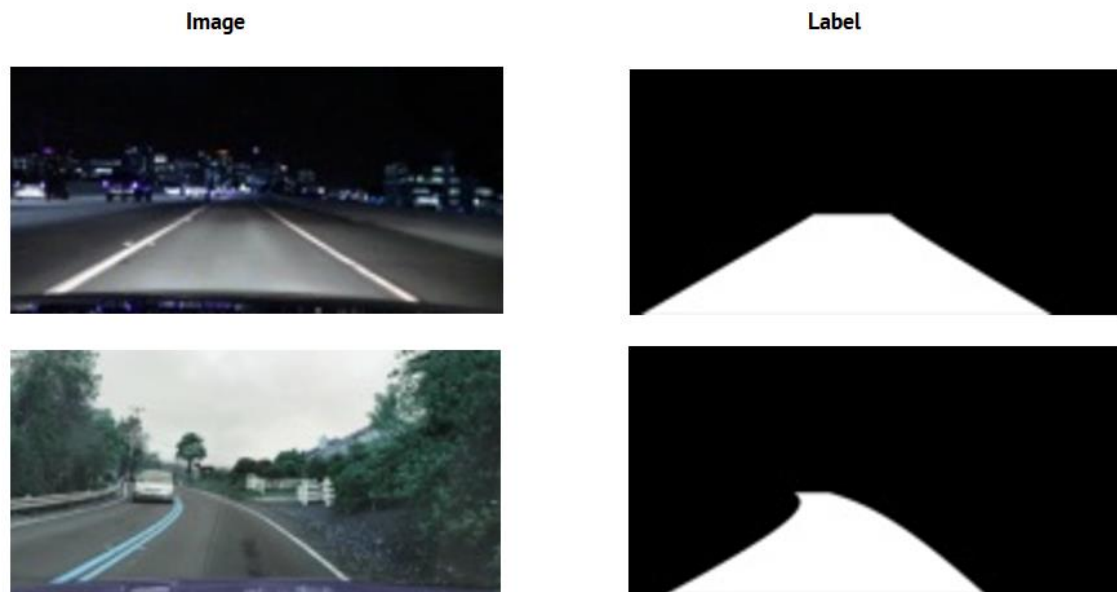
The model of the system contains five major blocks as mentioned below:

- Video Processing
- Lane Detection
- Object Detection
- Mathematical Model
- Decision Making Model

Video Processing block is a software unit that uses camera to record the video and convert the video into frames.

## Lane Detection NN

Lane detection block is supported by a retrainable neural network to detect the lanes. This NN is trained with a set of 12,763 images along with their labels. To increase the robustness of the detection system, these images are shuffled, resized, blurred, and rotated so that the NN can detect the lanes at various conditions. Samples of the dataset which is used to train the NN are displayed below.



*Figure 6: Lane Dataset Sample images*

## Object Detection NN<sup>[5]</sup>

The Object Detection unit contains a pre-trained NN namely Mask RCNN COCO, which is an open source NN used to detect various classes of objects and is trained on a set of more than 67,000 images of all the classes. This NN is used to detect cars, trucks and motorcycles and get their positions on the image. Pictures below show how this NN detects the objects in an Image. We can use the model with few lines of code shown below.

```
seg = instance_segmentation()
seg.load_model("mask_rcnn_coco.h5")
target_classes = seg.select_target_classes(car=True, truck=True)
```

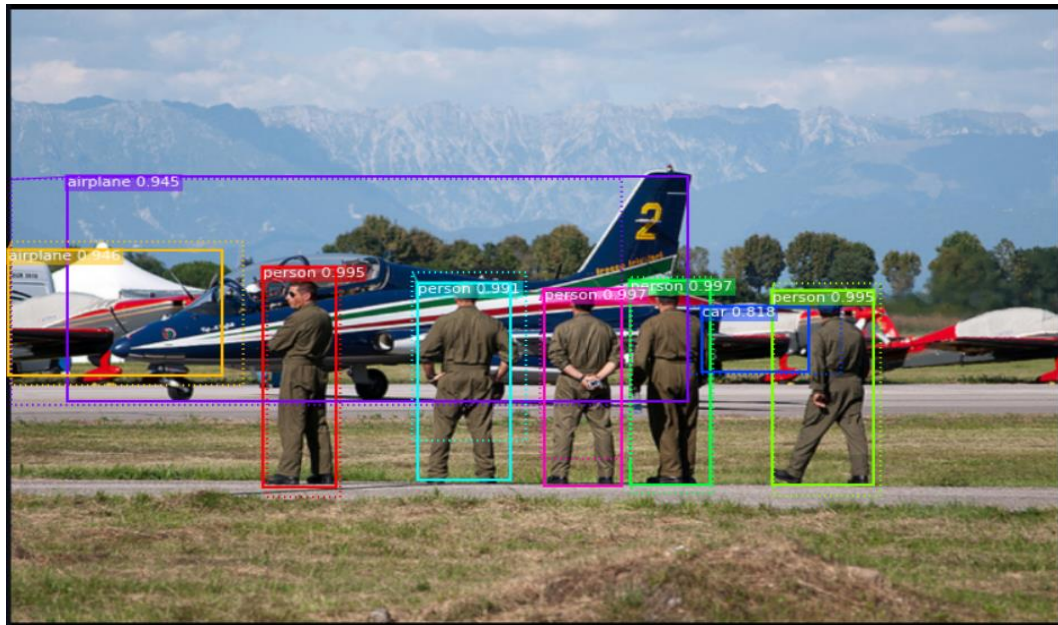


Figure 7: Mask RCNN COCO output image

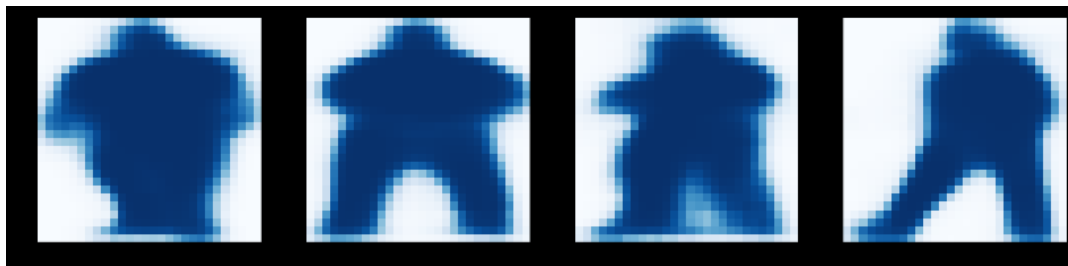


Figure 8: Mask RCNN COCO masks



Figure 9: Mask RCNN COCO Output with masks

## Mathematical Model

Mathematical model is fed with object positions by the Object detection NN, and the equations of the lanes are stored in the model so that the depth of the object is calculated with the object positions. The camera which is used for recording the video is calibrated with MATLAB camera calibrator tool as well as OpenCV in python to get the focal length of the camera lens. The depth each object is obtained with formulas shown below. Here,  $f$  = focal length of the camera (in pixels) obtained through deduction method and the depth is calculated using the basic principles of optics by knowing the actual height of the object and its height in the image.

```
if vals == 8:
    h_m = 4 #avg height of truck 4.5m

if vals == 3:
    h_m = 1.8 #avg height of car 1.6m (swift)

f = 600
#f = 1130
h_p = seg_vals[0]['rois'][car][2]-seg_vals[0]['rois'][car][0]

depth = h_m*f/h_p
depth = np.round(depth,decimals=2)
```

After the depths are found out the object's coordinates are used to find out the presence of the object in a particular lane. The snippet below shows the equations of the lane lines on an image.

```
def eqn_lt(x):
    return 100*(0.000000451634970*x**5-0.000068466154009*x**4+0.003961508965860*x**3-0.109350788939381*x*x+1.468743169695631*x-2.241943129953762)

def eqn_rt(x):
    return 1000*(-0.000000052704060*x**5+0.000007989736221*x**4-0.000462292823840*x**3+0.012760815498220*x*x-0.171438735311443*x+1.663153300843994)

def eqn_ilt(x):
    return 1000*(0.000000128368409*x**5-0.000019460165463*x**4+0.001125981458659*x**3-0.031080823468170*x*x+0.417331197141561*x-1.823357541371203)

def eqn_irt(x):
    return 1000*(-0.000000164792945*x**5+0.000024981987411*x**4-0.001445478697383*x**3+0.039900007122568*x*x-0.536117174978811*x+3.829616024874668)
```

Figure 10: Equations of Lanes

Where,

eqn\_lt – Equation of current lane's left line

eqn\_rt – Equation of current lanes' right line

eqn\_ilt – Equation of adjacent lane's left line

eqn\_irt – Equation of adjacent lane's right line

These line equations are obtained by using MATLAB lane detection application which is developed especially for this project. This function in MATLAB is used to draw lane lines on the image and get the pixels of the plotted lines. This function takes in an Image and returns the boundaries of lanes as well as an Image with lines plotted on it.

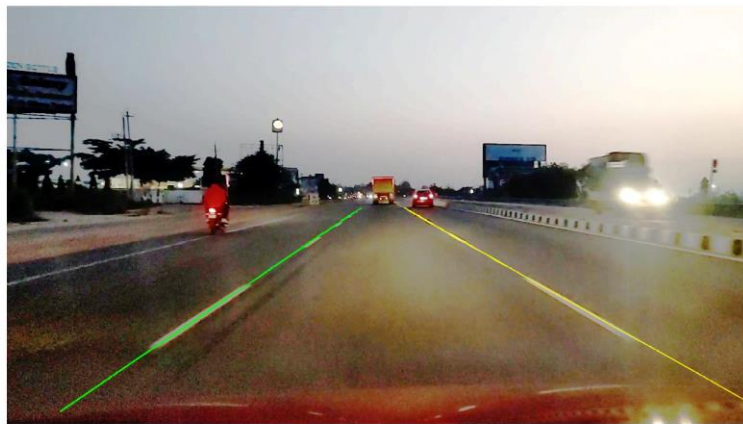
```
birdsEyeBW = segmentLaneMarkerRidge(rgb2gray(birdsEyeImage), ...  
    birdsEye, approxBoundaryWidth);  
  
[imageX, imageY] = find(birdsEyeBW);  
xyBoundaryPoints = imageToVehicle(birdsEye, [imageY, imageX]);  
  
boundaries = findParabolicLaneBoundaries(xyBoundaryPoints, approxBoundaryWidth);  
  
XPoints = 3:50;  
  
lanesI = insertLaneBoundary(I, boundaries(1), sensor, XPoints);  
lanesI = insertLaneBoundary(lanesI, boundaries(2), sensor, XPoints, 'Color', 'green');
```

*Figure 11: MATLAB code snippet for drawing lanes*

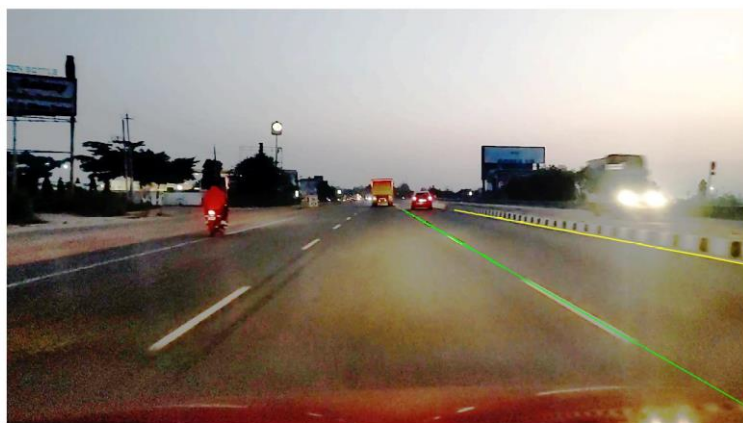
Once the boundaries of the lane lines are obtained, they are plotted with depth value on X-axis and pixel value on Y-axis to fit the equation of the line. Below are the plots of how the lane line equations are obtained in Excel using polynomial line fit. First a method of fitting the line with a second-degree polynomial has resulted in very poor results and to compensate for that a five-degree polynomial looked to fit with a correlation coefficient of 0.99 but still was unable to perform well. Hence, the significant digits of the equation are increased to 8 instead of 4 as the polynomial has the highest degree of 5 which would impact a lot with an infinitesimal change in coefficient. Ideally, theory demands to have 32 significant digits for accurate values, but we are limited with 8 digits due to limitation of the software. The pictures below show how the lane lines



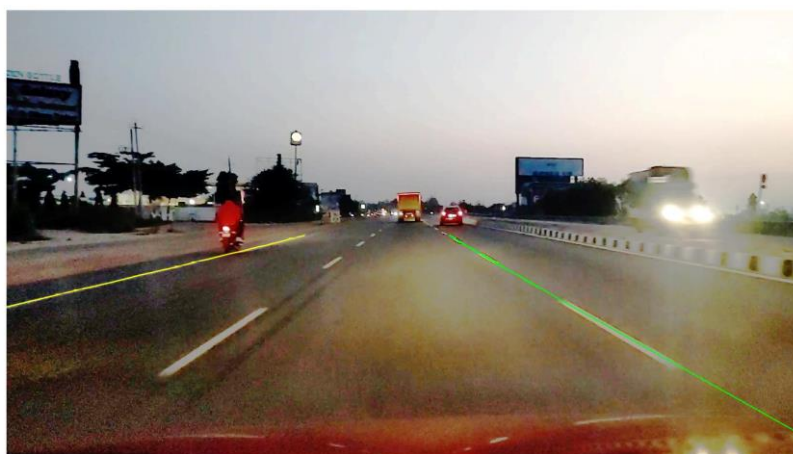
are drawn in the picture and these lines are boundaries for corresponding lanes which are the lane equations used in this project.



*Figure 12: Left lane and Right lane marked*



*Figure 13: Right lane and Immediate right lane marked*



*Figure 14: Right lane and Immediate left lane marked*

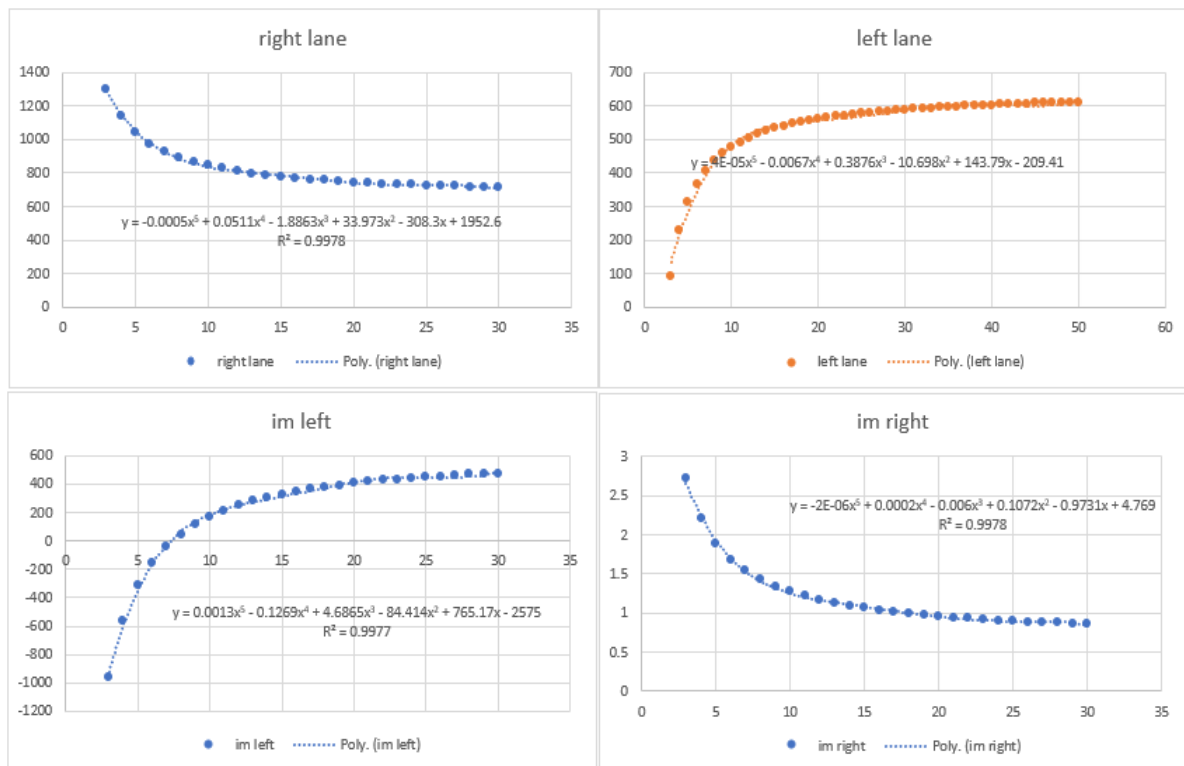


Figure 15: Plots of different lanes in Excel

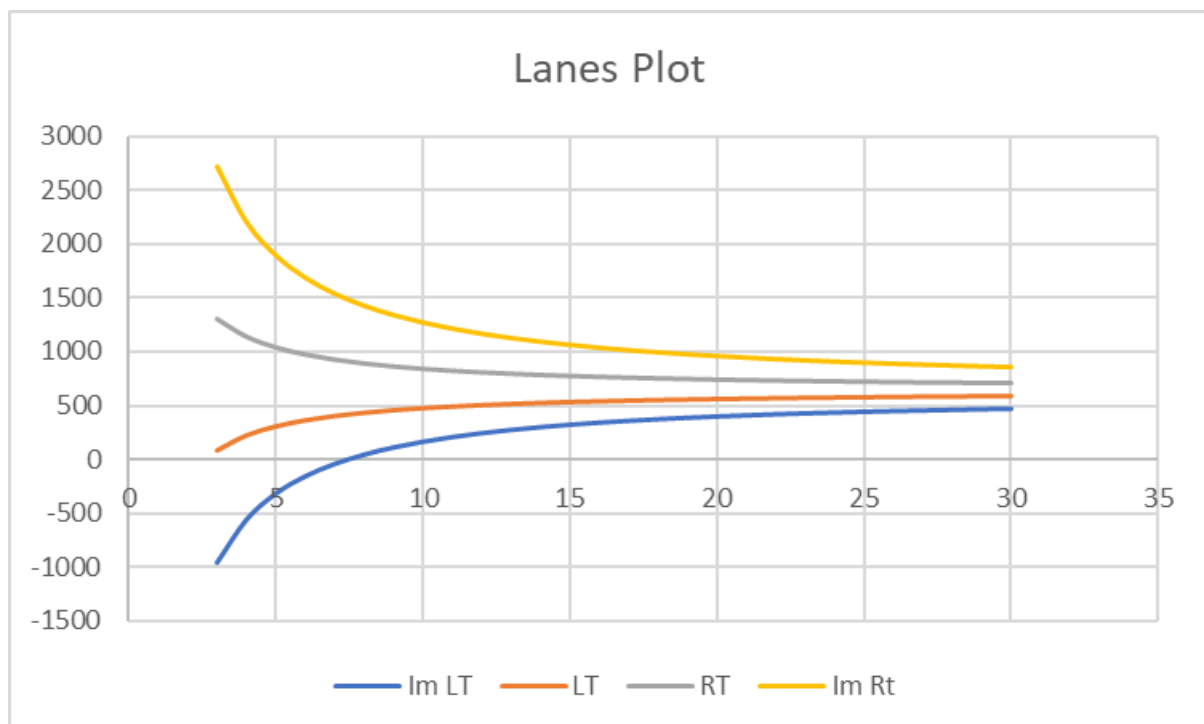
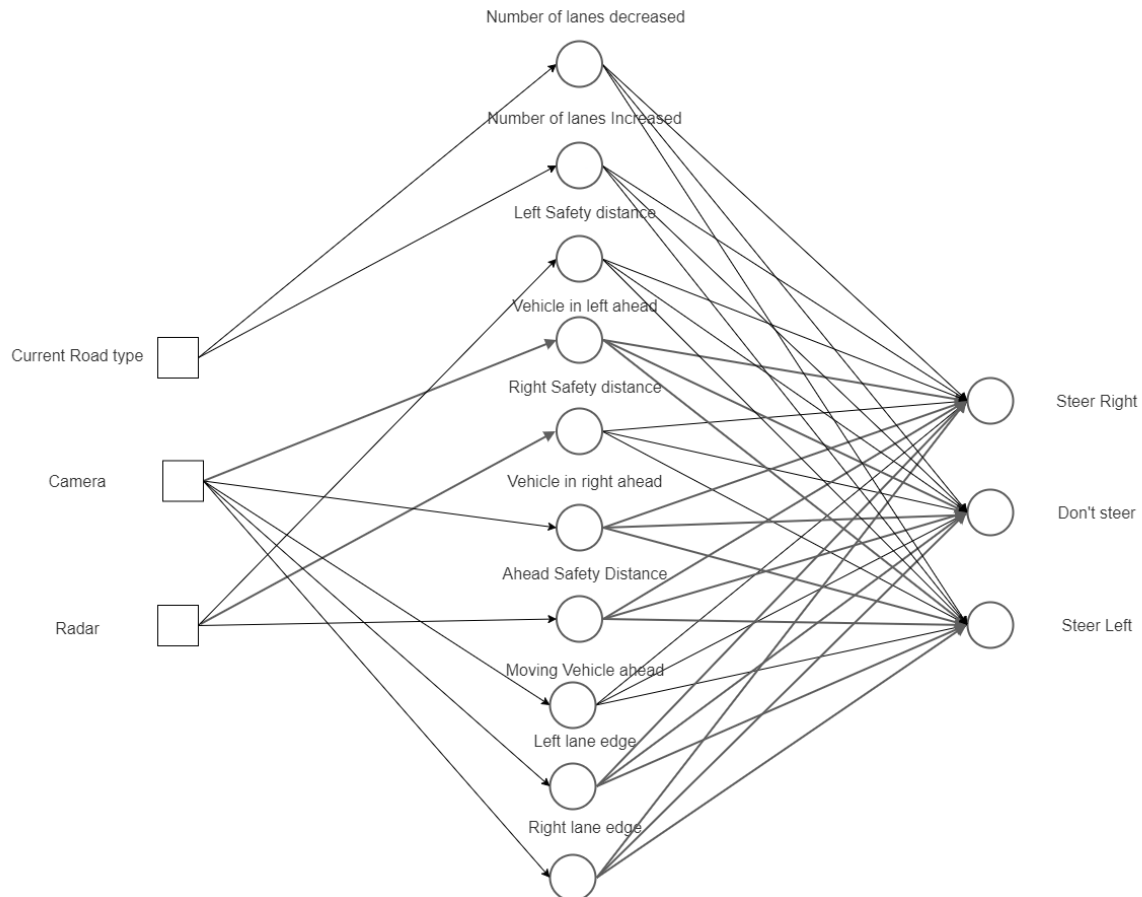


Figure 16: Concise plot of all lanes



## Decision Making NN

The Decision making NN contains a custom made simple NN which is trained with a list of conditions and can also be trained for new conditions. This NN must have an accuracy of 1, as the decision of the system affects the safety of the driver and passengers. So, the NN is trained with 100% of the conditions to make sure no havoc occurs.



*Figure 17: Decision Making Neural Network*

It is also supported by the Knowledge Base to set the input variables based on the road topology and to get the hyperparameters while training itself. Although, this NN has an accuracy of 1 this does not ensure complete safety as the total accuracy of the system still depends on the Object Detection NN and precision of lane equations at every instance.

## Training the custom made NN

This NN is trained with 140 different conditions which can be commonly encountered on a highway. Below is the table of how the training data is defined along with outputs.

L De	L Incr	Left SD	V Left	Right SD	V Right	Ahead SD	V Ahead	LL edge	RL edge	SR	DS	SL
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	0.00	1.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00	0.00	1.00	0.00	0.00	1.00

Figure 18: Training data for Decision Making NN

The last three columns are the outputs which are 'Steer Right', 'Do not Steer' and 'Steer Left'. First 10 columns are the input neurons. Initially, the weight calculation started off by hand calculations and custom activation function. The initial code looked like this (fig:19) when, it was written as a matrix in Google Colab.

```
[ ] import numpy as np
weights = np.array([[0.5,0.33,-0.5],[0.5,0.33,-0.5],[-0.5,0.33,0.5],[-0.5,0.33,0.5],[0.5,0.34,0.5],
[0.5,0.33,-0.5],[-0.5,0.33,0.5],[1,1,1]])
print(weights)

[ ] print("Enter input vector")
for i in range(8):
    a = int(input())
    ip.append(a)
op = np.matmul(ip,weights)
message = ["Steer Right","Do not Steer","Steer Left"] #Output neurons
print(op)
flag = 0
for i in range(len(op)):
    if op[i] >=2 and op[i]==max(op): #activation function (custom)
        print(message[i])
        flag += 1
        break
if flag == 0:
    print(message[0])
```

Figure 19: Code of initial build of Decision Making NN

Later, as it is realised that the NN must take 10 different inputs the NN is expanded and now it is trained using the code below.

```
df = pd.read_excel (r'truth_table updated.xlsx', sheet_name='Sheet1',usecols="A:J")
d = pd.read_excel (r'truth_table updated.xlsx', sheet_name='Sheet1',usecols="K:M")

# The input and output, i.e., truth table, of a NAND gate
x_train = df.to_numpy()[1:139]
y_train = d.to_numpy()[1:139]

# Create neural networks model
model = Sequential()

# Add layers to the model
```

```
model.add(Dense(3, activation='sigmoid', input_dim=10))

# Compile the neural networks model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
epochs = 700
history = model.fit(x_train, y_train, epochs=epochs)
acc = history.history['accuracy'][-1]

if acc <= 1:
    epochs = get_epochs(epochs)
    model.fit(x_train, y_train, epochs=epochs)

model.save('weights_matrix_updated_v1.h5')
```

If the desired accuracy is not met the function “`get_epochs(epochs)`” queries the Knowledge base to get the hyperparameters and retrains automatically when the desired accuracy is reached.

The NN uses sigmoid activation function. While compiling the model ‘adam optimizer’ along with ‘binary cross entropy’ loss function are used to obtain the weights and bias of the model. The model is saved into ‘.h5’ format and can be used later to predict the output based on the input variables. The graphs below (fig:20) indicate the training accuracy and loss, and validation accuracy and loss of the system.

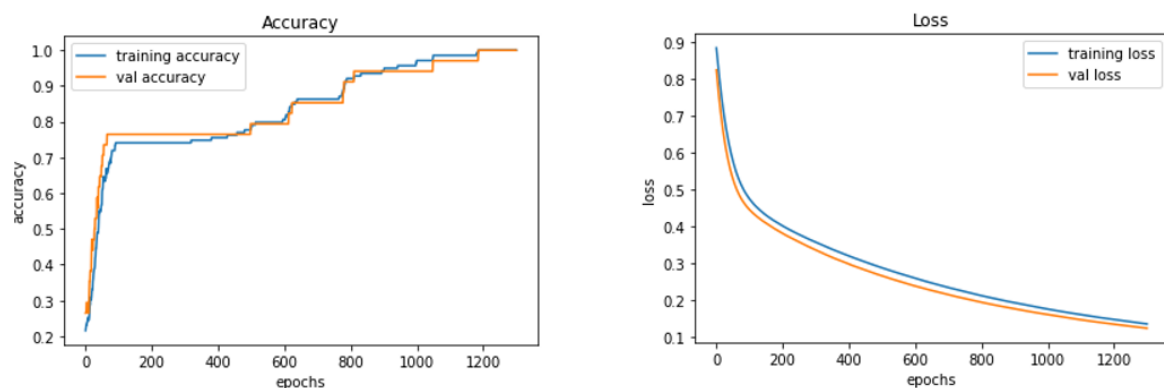


Figure 20: Accuracy vs Epochs and Loss vs Epochs graphs

The figure below shows a generalized concept map of the various blocks in the system and their relationship with other blocks. Each time an image is extracted from the video, it gets stored in database along with the output decision. This database is used

as a log file to see how the system has performed at different conditions and gives insights about how the system can be trained further to improve accuracy.

## Concept Map

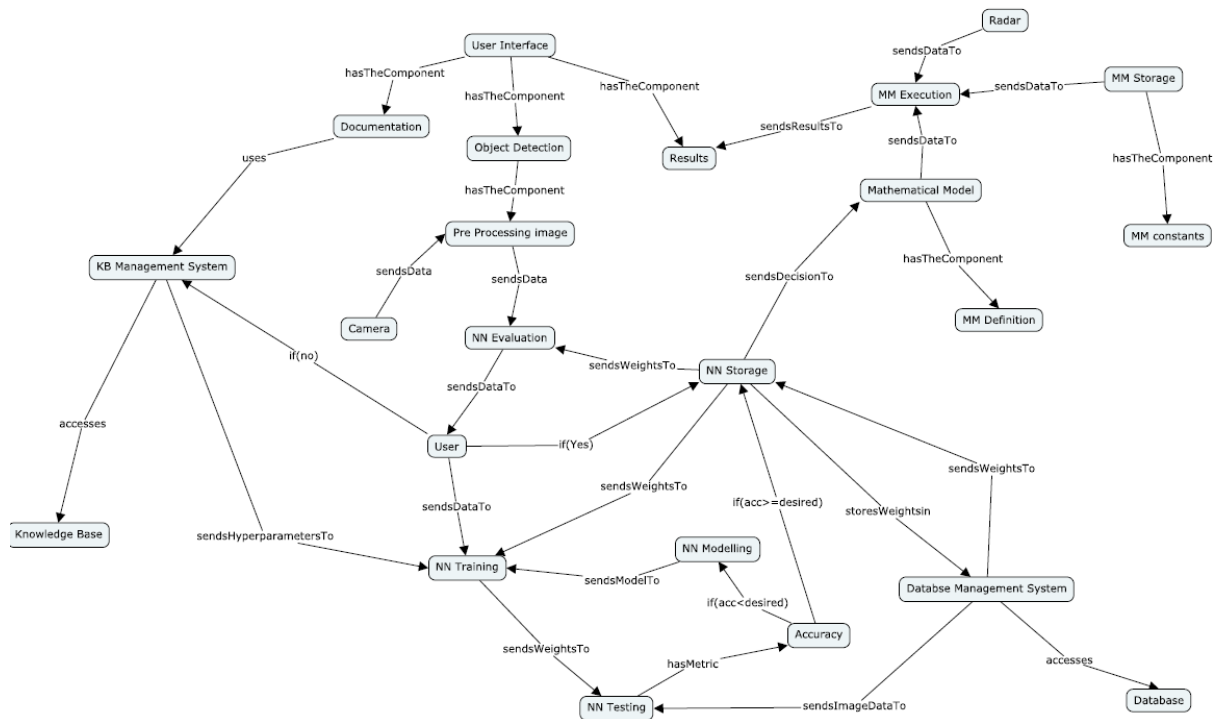


Figure 21: Concept map of the system

## Methodology

The model is designed to make a decision based on the environment around the Ego Vehicle and is displayed on the dashboard. In Video processing block the video is cut into frames and are fed to the Neural networks to detect lanes and objects in the images. The Object detection ANN detects various classes of the objects in the Image and returns their positions as the coordinates of bounding boxes to the Mathematical Model. The Mathematical model contains the equations of lanes , camera parameters like focal length, principal point. Based on the conditions defined in the model the approximate positions of the objects on the road and their depths from the Ego Vehicle are calculated.

## UML Diagram

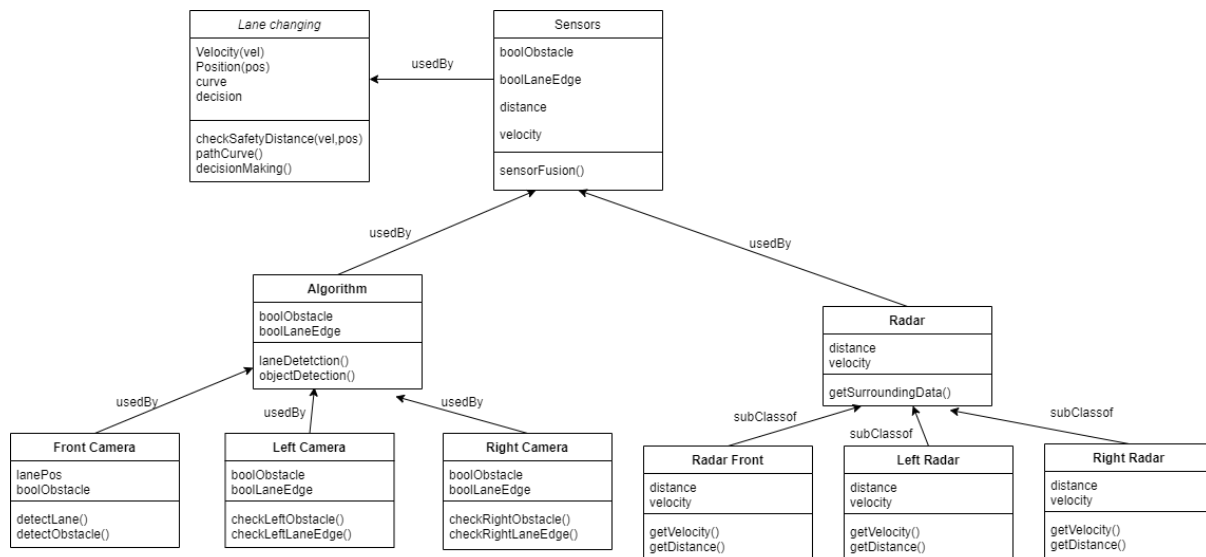


Figure 22: UML Diagram

The UML Diagram shows the list of methods in different classes. The reader is requested to make a note that the UML diagram showed above does not reveal the actual methods and classes used in the software to protect the work of the author. This gives a general idea about how the hierarchy of the software is developed.

## Swimlane Diagram

The swimlane diagram represents the flow of the processes in each block of the system. It is used to represent the pipeline of the process flow and eliminates redundancies. In simple terms it shows a clear narrative between the interactions of various functional blocks present in the system. They act like a bridge between various blocks thus ensuring a clean and clear process flow throughout the system.

In this project the system begins to function when the user switches on the Lane change assist system in the vehicle or it is automatically started in case of the Ego vehicle. The below diagram (fig:23) shows an ideal way of interaction between the various blocks and their components.

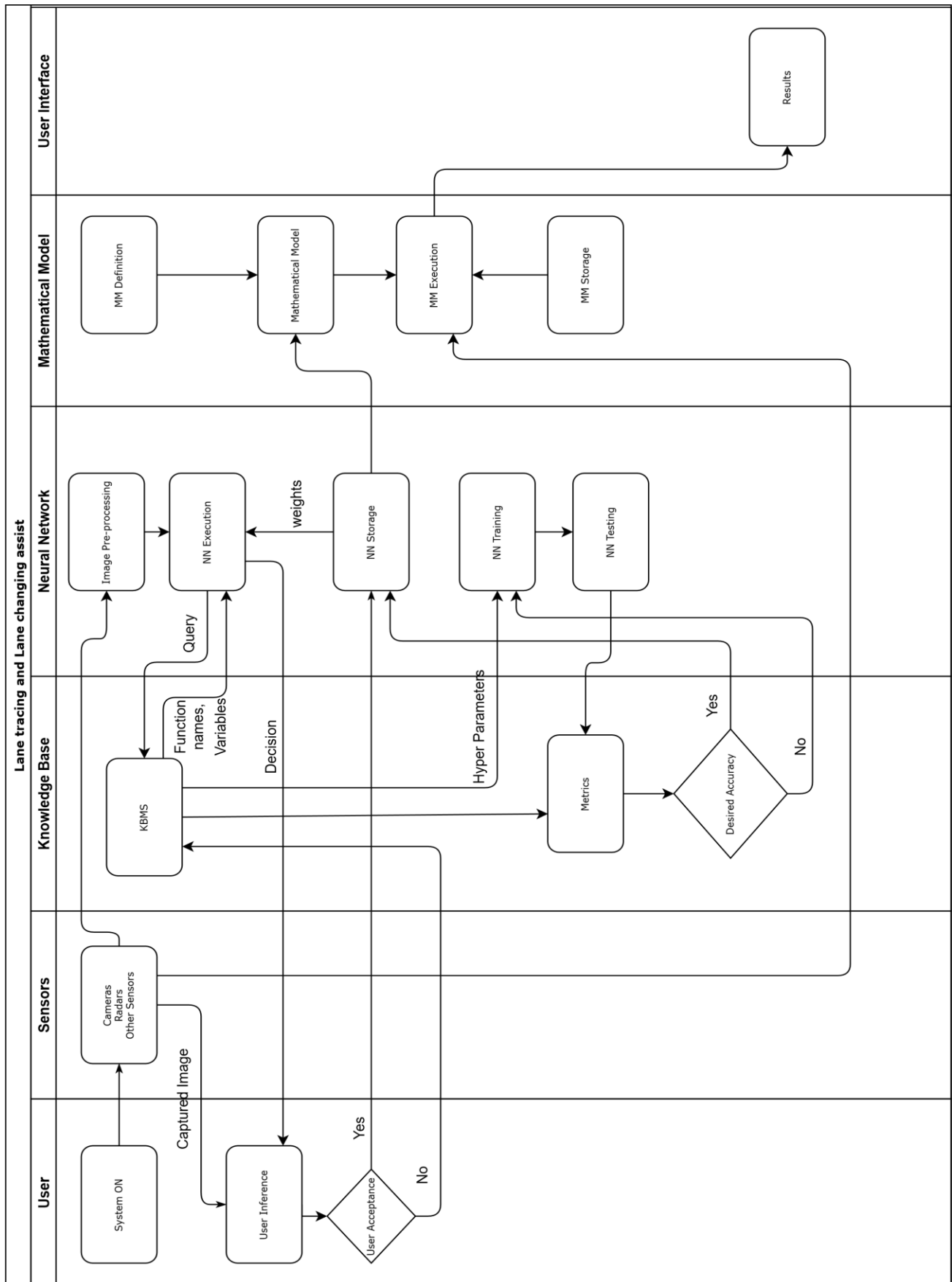


Figure 23: Swimlane Diagram

Once the depths are calculated the depths of each object are fed into lane equations to get their positions on the road. Below code determines the presence of an object in current lane, immediate left lane, and immediate right lane. These equations work good only in case of straight lines and they work at perfection if the Ego vehicle is going nearly to the centreline of the lane.

```
if depth <= 50:
    i_lt = eqn_ilt(depth)
    lt = eqn_lt(depth)

    i_rt = eqn_irt(depth)
    rt = eqn_rt(depth)

    if seg_vals[0]['rois'][car][1] >= rt and seg_vals[0]['rois'][car][1] < i_rt :
        print("vehicle is to the immediate right lane of ego")
        lane_info.append(1)
        r_depth[car] = depth

    elif seg_vals[0]['rois'][car][3] <= lt and seg_vals[0]['rois'][car][3] > i_lt :
        print("vehicle is to the immediate left lane of ego")
        lane_info.append(-1)
        l_depth[car] = depth

    elif seg_vals[0]['rois'][car][3] < rt and seg_vals[0]['rois'][car][1] > lt:
        print("vehicle is ahead")
        lane_info.append(0)
        a_depth[car] = depth
    else:
        print("Not in ROI")
```

The obtained depths and the object's positions are sent to the function to set the input variables of the Decision making NN.

```
def set_vars(lane_info, r_depth, l_depth, a_depth):
    SD = 0
    rt_veh, lt_veh, ah_veh = 0, 0, 0
    rt_sd, lt_sd, a_sd = 0, 0, 0
    if 1 in lane_info:
        rt_veh = 1
        if min(r_depth.values()) > 30:
            rt_sd = 1
    if -1 in lane_info:
        lt_veh = 1
        if min(l_depth.values()) > 30:
```

```
        lt_sd = 1
    if 0 in lane_info:
        ah_veh = 1
        if min(a_depth.values()) > 30:
            a_sd = 1
    return lt_sd,lt_veh,rt_sd,rt_veh,a_sd,ah_veh
```

At this point there was a revelation that the remaining variables cannot be found with the help of the code as it would increase the computational effort on the system. So, a Knowledge base is used to query the number of lanes on a specific type of road and use that number to determine the current lane of the Ego Vehicle so that the remaining variables LD,LI,LL and RL are set based on the road topology.

```
g = r.Graph()
g.parse(r'ltademo.owl',format="ttl")

qres = g.query("""
SELECT ?subject ?value

WHERE
{
:National_Highway :hasLanes ?value

}
""")
```

For example, the above code shows how the maximum number of lanes on a type of road named 'National\_highway' are queried from the KB using SPARQL syntax and RDFlibrary in Python.

After setting all the input variables to either 1 or 0 based on the environment detected from the NNs and Mathematical model along with KB, these variables become the test variables for Decision Making NN. The output is found using the below code

```
model = keras.models.load_model(r"weights_matrix_updated.h5")
op = model.predict(np.array([[lane_dec,lane_inc,lt_sd,lt_veh,rt_sd,rt_veh,
                              a_sd,ah_veh,ll_edge,rl_edge]]))

message = ["Steer Right","Do not Steer","Steer Left"] #Output neurons
output = op[0]
```



```
for i in range(len(output)):
    if output[i]==max(output): #activation function
        print(message[i])
        msg = message[i]
        break
```

This code prints the output message to the driver, or it can be used for controlling the Ego Vehicle.

## Software Implementation

The whole software interface of the system is built on Python. This makes use of various libraries like numpy, keras, rdflib etc. The interface application is built using Flask web application framework. To store the data SQLite database is used. The database stores the data of Image path, decision, and user response. The figure (fig:24) below shows how the web app interface of the system looks like. This app has various options for the user to know more about the project.

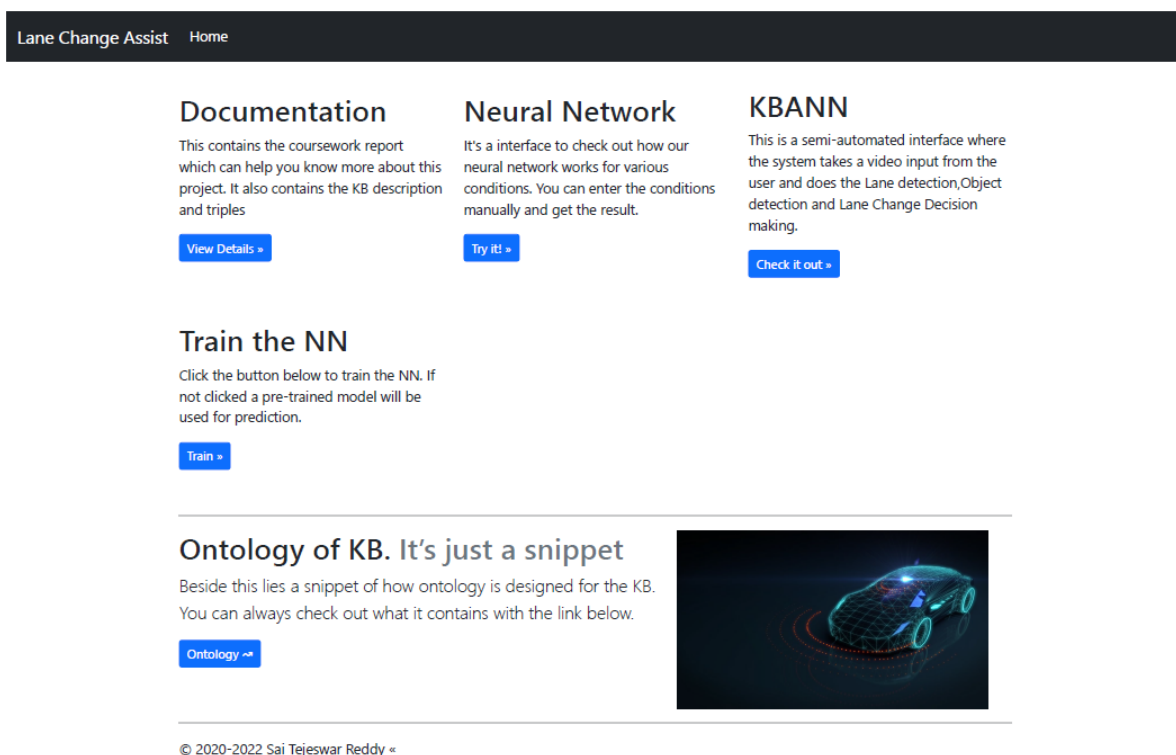


Figure 24: Web application interface

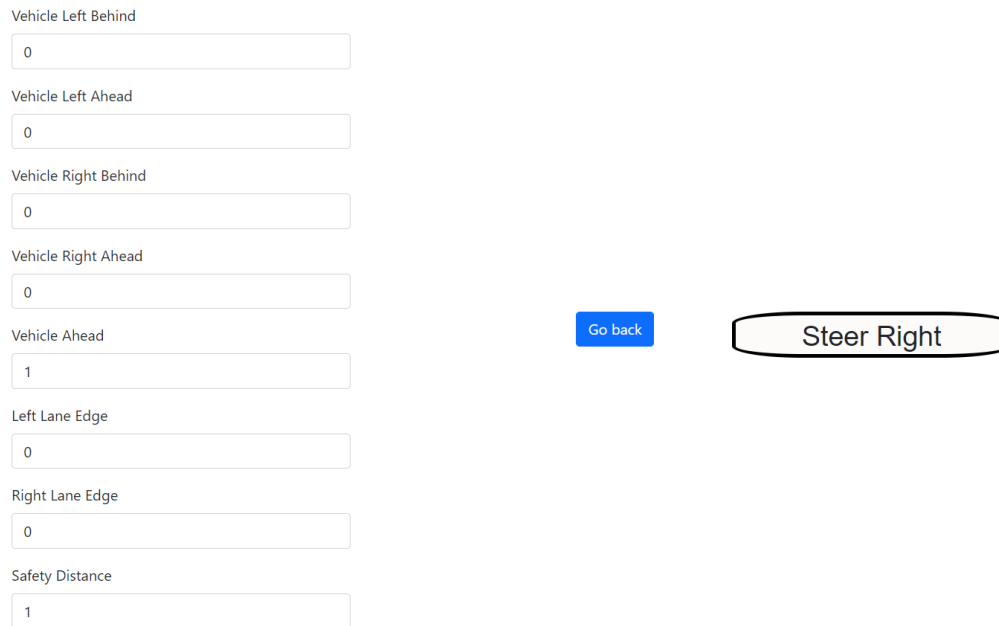
## Neural Network Interface

The second step followed the tree view by creating an interface for the decision-making neural network using C#. The figure below shows a snippet of the interface.

The screenshot shows a Windows Form titled 'Form1' with a light gray background. On the left, there are eight input fields, each with a label and a numerical value: 'Vehicle left behind' (1), 'Vehicle left ahead' (1), 'Vehicle right behind' (1), 'Vehicle right ahead' (1), 'Vehicle ahead' (0), 'Left lane edge' (0), 'Right lane edge' (0), and 'Safety distance' (1). In the center, there is a button with the text 'Click for output' highlighted by a blue rectangular border. To the right of the button, there is a box labeled 'Output' containing the text 'Do not Steer'.

*Figure 25: Neural Network form created in C#*

As the implementation of neural networks is easier in python than any other languages, in the further steps the code is completely written in Python and the interface of the web application is designed using HTML5 and CSS. The initial build of the app contained just 8 inputs as that was planned at the time of its initial build (fig:26). This was later expanded into 10 inputs with different variables than earlier. The second figure (fig:27) below shows the latest build of the application of testing Decision Making Neural Network.



Vehicle Left Behind

Vehicle Left Ahead

Vehicle Right Behind

Vehicle Right Ahead

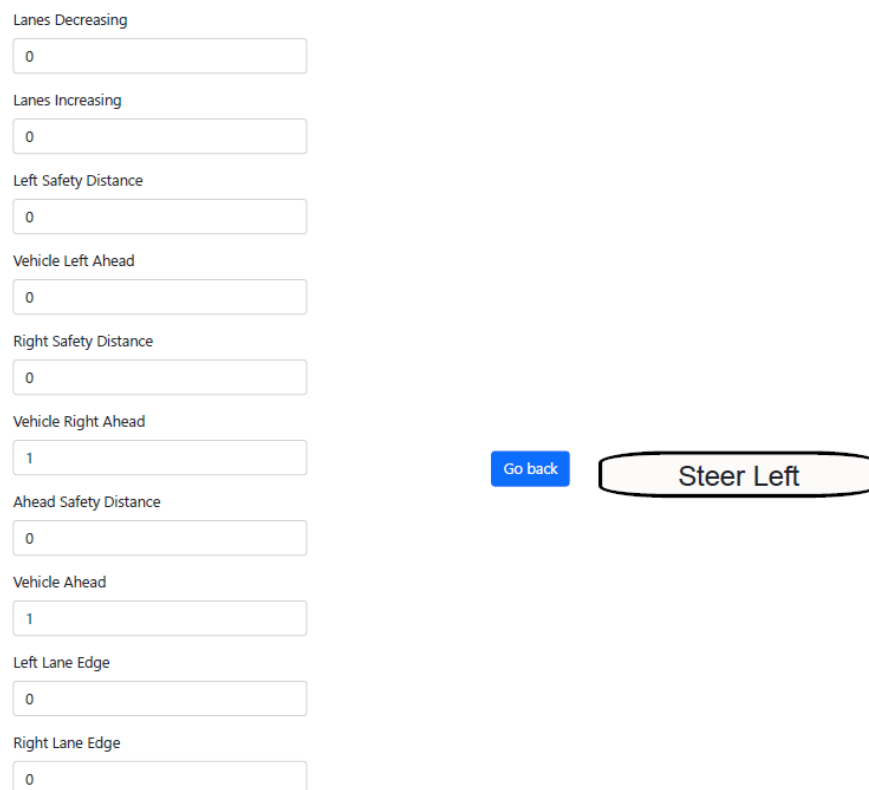
Vehicle Ahead

Left Lane Edge

Right Lane Edge

Safety Distance

*Figure 26: Initial build of Manual Neural Network testing interface*



Lanes Decreasing

Lanes Increasing

Left Safety Distance

Vehicle Left Ahead

Right Safety Distance

Vehicle Right Ahead

Ahead Safety Distance

Vehicle Ahead

Left Lane Edge

Right Lane Edge

*Figure 27: Latest build of Manual Neural network testing interface*

## Knowledge Base

The URIs used in the KB are shown below (fig:28).

```
@prefix : <http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#> .  
@prefix lcs: <http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#> .  
@prefix owl: <http://www.w3.org/2002/07/owl#> .  
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .  
@prefix xml: <http://www.w3.org/XML/1998/namespace> .  
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .  
@base <http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#> .
```

Figure 28: URIs used in KB

The knowledge base consists of various classes (fig: 30 & 31) that include processes in the system, vehicles, sensors, road infrastructure, road sign etc. Below is a list of triples for some of the processes which the code queries to get the tree view (fig:29).

```
lcs:Process1 rdf:type owl:Class ;  
..... rdfs:subClassOf lcs:Process ;  
..... rdfs:comment "Lane Tracing and Lane changing block.  
This is a parent block for all the sub processes."^^xsd:string ;  
..... rdfs:label "LT and LC"^^xsd:string .  
  
### http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Process1-1  
lcs:Process1-1 rdf:type owl:Class ;  
..... rdfs:subClassOf lcs:Process1 ;  
..... rdfs:comment "LT - Lane Tracing  
This detects the lanes on the road and traces them on the image for the system to identify the lane boundaries."^^xsd:string ;  
..... rdfs:label "LT"^^xsd:string .  
  
### http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Process1-2  
lcs:Process1-2 rdf:type owl:Class ;  
..... rdfs:subClassOf lcs:Process1 ;  
..... rdfs:comment "PP - Predicting Possibilities  
This block reads the environment and predicts the possibilities of the system to perform the action of Lane changing safely."^^xsd:string ;  
..... rdfs:label "PP"^^xsd:string .
```

Figure 29: Turtle code of processes in ontology

```
lcs:Roundabout rdf:type owl:Class ;  
..... rdfs:subClassOf lcs:Infrastructure .  
  
### http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#School_Nursery  
lcs:School_Nursery rdf:type owl:Class ;  
..... rdfs:subClassOf lcs:Traffic_Sign .  
  
### http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Sign  
lcs:Sign rdf:type owl:Class ;  
..... rdfs:subClassOf lcs:Infrastructure .  
  
### http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Slow_down  
lcs:Slow_down rdf:type owl:Class ;  
..... rdfs:subClassOf lcs:Traffic_Sign .
```

Figure 30: Snippet of some classes used in KB

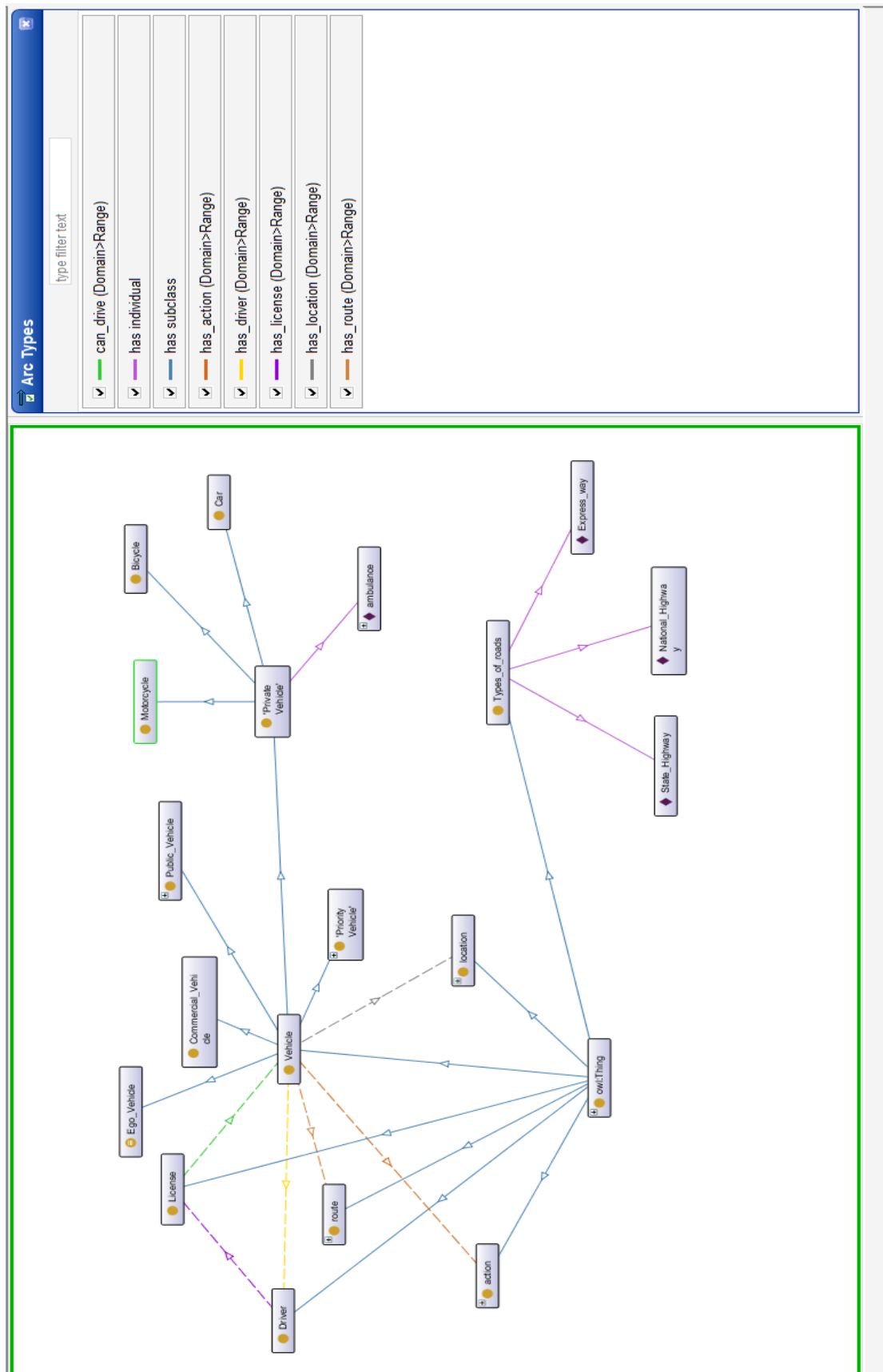


Figure 31: Ontology graph of some classes in KB

The knowledge base also contains the names of the functions that are integral parts of the system. Storing the functions in the Knowledge base and calling them through querying is a better way of encapsulation and protecting the software as the system will always ask Knowledge base to call the function instead of calling on its own. This step ensures the system to be more cautious and we can alter the function of the software by making changes in the KB. The code below shows how the function calling is done through query and necessary imports are done.

```
def dq(inArgs,outArgs):
    g = r.Graph()
    g.parse(r'ltademo.owl',format="ttl")
    outs = []
    qres = g.query("""

        SELECT    ?name ?module

        WHERE
        {
            ?subject    lcs:inPar        "%s";
                        lcs:outPar        "%s";
                        lcs:moduleName    ?module;
                        rdfs:label        ?name.
        }
        ""%(inArgs,outArgs))

    for row in qres:
        outs.append(str(row["module"]))
        outs.append(str(row["name"]))

    return outs
```

Above function returns the name of the function stored in the KB by taking in the input arguments and output arguments of the function, which are unique for every method. This method is called using the following code.

```
inArgs = "seg_vals,count" #take input from ...
outArgs = "msg"
[module,func_name] = dq(inArgs,outArgs)
module = 'flask_app_3.' + module
mod = importlib.import_module(module)
```

This will import the function into the current script, and it can be called using a line of code.

```
ip,msg,index = getattr(mod,func_name)(seg_vals,count,filename)
```

The input arguments for the respective function can be send in the second pair of braces as shown above. Similarly, this type of function calling is employed in various parts of the code where the methods are to be protected. Below code snippet demonstrates the same followed by a picture (fig:32) showing how the function names are written in KB.

```
inArgs = "lane_info,r_depth,l_depth,a_depth" #take input from ...
outArgs = "lt_sd,lt_veh,rt_sd,rt_veh,a_sd,ah_veh"

[module,func_name] = dq(inArgs,outArgs)
module = 'flask_app_3.' + module
mod = importlib.import_module(module) #to get set_vars method
inArgs = "cur_lane,rt_turn,lt_turn" #take input from ...
outArgs = "cur_lane"

[module,func_name] = dq(inArgs,outArgs)
module = 'flask_app_3.' + module
mod = importlib.import_module(module) #lane_counting
```

```
lcs:final_result.rdf:type owl:Class;
.....rdfs:subClassOf lcs:Function_calls;
.....lcs:inPar "seg_vals,count";
.....lcs:moduleName "demo";
.....lcs:outPar "msg";
.....rdfs:label "final_result"..

### http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#lane_counting
lcs:lane_counting.rdf:type owl:Class;
.....rdfs:subClassOf lcs:Function_calls;
.....lcs:inPar "cur_lane,rt_turn,lt_turn";
.....lcs:moduleName "query";
.....lcs:outPar "cur_lane";
.....rdfs:label "lane_counting"..

### http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#m_model
lcs:m_model.rdf:type owl:Class;
.....rdfs:subClassOf lcs:Function_calls;
.....lcs:inPar "seg_vals";
.....lcs:moduleName "mm";
.....lcs:outPar "lane_info,r_depth,l_depth,a_depth";
.....rdfs:label "m_model"..
```

Figure 32: Function names in Ontology

## Inconsistency Checking

The KB must be checked for inconsistencies if there are any. As the final KB had no inconsistencies the system performed well. But, to demonstrate what inconsistency exactly is, an inconsistency in KB is coercively added to the KB. It is shown below (fig:33).

```
###.http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Priority_Vehicle
lcs:Priority_Vehicle rdf:type owl:Class;
..... rdfs:subClassOf lcs:Vehicle;
..... owl:disjointWith lcs:Private_Vehicle..
```

Figure 33: Disjoint classes in KB

This shows that the class “Priority Vehicle” is ‘disjointWith’ the class “Private\_Vehicle”. Now an individual named “ambulance” is added to the ontology and is said as a type of both “Priority Vehicle” and “Private\_Vehicle”.

```
###.http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#ambulance
lcs:ambulance rdf:type owl:NamedIndividual.,
..... lcs:Priority_Vehicle.,
..... lcs:Private_Vehicle..
```

Figure 34: Inconsistency in Ontology

This(fig:34) clearly is inconsistent as a same individual cannot be in two disjoint classes. This inconsistency can be checked with following code.

```
g = r.Graph()
g.parse(r'ltademo.owl', format="ttl")
qres = g.query("""
                SELECT ?objects ?object
                WHERE {
                    lcs:ambulance rdf:type      ?object.
                    ?object      rdfs:label     ?objects.
                }
            """)

classes = []
for row in qres:
    classes.append(str(row["objects"]))
print(classes)
qres1 = g.query("""
                SELECT ?object1 ?object2
                WHERE {
                    ?object1 rdfs:label "%s".
                    ?object2 rdfs:label "%s".
                    ?object1 owl:disjointWith ?object2.
                }""%(classes[1],classes[0]))
```



Now, as the system queries the KB two times, if there is anything in the variable 'qres1' then it means that there are inconsistencies in the KB. It can be checked like this:

```
for anyvalue in qres1:  
    print("inconsistent")
```

As it is difficult to check for inconsistencies in large ontologies, in python we have a library called "owlready2" in which there are functions to check for consistency of a KB. When we run the reasoner, it automatically checks for inconsistencies in the KB. If there are any then the reasoner creates a class called "inconsistent\_classes" in the KB. If there is any presence of such classes, we can say that the KB is inconsistent.

```
import owlready2 as owl  
onto = owl.get_ontology("file://trial.owl").load()  
  
owl.sync_reasoner_pellet(infer_property_values = True, infer_data_property_val  
ues = True)  
  
if len(list(onto.inconsistent_classes())) == 0:  
    print("ontology is consistent")  
else:  
    print("ontology is incosistent")  
onto.save()
```

Using the reasoner makes the inconsistency check faster and more accurate. There are other reasoners like 'hermit' which is one of the popular reasoners used in Ontologies.

## Rules in KB – Inconsistency check & Inference

In this project, a rule is written using Semantic Web Rule Language (SWRL). This rule is used to automatically retrain the Decision-Making Neural Network when desired accuracy is not achieved. Below code demonstrates how the rule is written using SWRL to decide the hyperparameters (in this case number of epochs) required for training the NN. Below code illustrates how it is written in python script using owlready2 library.

```
import owlready2 as owl  
onto = owl.get_ontology("file://trial.owl").load()  
  
def get_epochs(epoch):
```

```
with onto:
    class Hyper_parameters(owl.Thing):
        pass
    class multiplier(Hyper_parameters >> int, owl.FunctionalProperty):
        pass
    class current_epoch(Hyper_parameters >> int, owl.FunctionalProperty):
        pass
    class next_epoch(Hyper_parameters >> int, owl.FunctionalProperty):
        pass

    rule = owl.Imp()
    rule.set_as_rule("""Hyper_parameters(?d), multiplier(?d, ?p), current_epoch(?d, ?n),
        multiply(?r, ?p, ?n) -> next_epoch(?d, ?r)""")

hyp = Hyper_parameters(multiplier = 2, current_epoch = epoch)
owl.sync_reasoner_pellet(infer_property_values = True, infer_data_property_values = True)
if len(list(onto.inconsistent_classes())) == 0:
    print("ontology is consistent")
else:
    print("ontology is incosistent")
onto.save()

return hyp.next_epoch
```

This function receives current number of epochs with which the first training of NN is done. If desired accuracy is not achieved, then the number of epochs for next training will be multiplied by 2 ( doubled ).

In addition to this rule there are other rules written in OWL format with which we can get some inferences. To demonstrate this, in this project few rules are written using OWL which are show in the picture (fig:35) below.

```
###. http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Ego_Vehicle
lcs:Ego_Vehicle rdf:type owl:Class;
..... owl:equivalentClass [ owl:intersectionOf ( lcs:Vehicle
..... [ rdf:type owl:Class;
..... owl:complementOf lcs:Priority_Vehicle
..... ) ];
..... rdf:type owl:Class
..... ] .
```

Figure 35: OWL Rule in KB

This is the rule written in OWL format from which we can infer that “Ego\_vehicle” is a class which is **equivalentTo** the class “Vehicle” and does not include the class “Priority\_Vehicle”, which is already a subclass of “Vehicle”.

Demonstration of inference can be done in python with a few lines of code as show below.

```
from owlready2 import *

inferences = get_ontology("file://ltademo.owl").load()

with inferences:
    sync_reasoner()

inferences.save("trial1.owl")
```

The initial ontology looks like shown below, which means that the class “Private Vehicle” is **subClassOf** the class “Vehicle”

```
<owl:Class rdf:about="http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Private_Vehicle">
  <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#Vehicle"/>
  <rdfs:label>Private Vehicle</rdfs:label>
</owl:Class>
```

After the rule is added as shown above in the picture above where the code is written in turtle syntax, the reasoner is run to infer the rules and it written into a new XML file. The inferred fact looks like below, which means that the class “Private Vehicle” is now a **subClassOf** the class “Ego Vehicle”

```
<owl:Class rdf:about="#Private_Vehicle">
  <rdfs:subClassOf rdf:resource="#Vehicle"/>
  <rdfs:subClassOf rdf:resource="#Ego_Vehicle"/>
  <rdfs:label>Private Vehicle</rdfs:label>
</owl:Class>
```

This automatic inference engine has a limitation i.e., the source file of the KB must be written in XML format and not in turtle syntax.

One can see the whole ontology of the project in the web application by clicking on the ontology button, which force downloads a pdf file that has the ontology.

## Automatic Knowledge Base ANN ( KBANN )

There is an option in the web application to check out the automatic KBANN, which is an integral part of this project. Below snippets of code show the process that happens once the user uploads a video.

Video is extracted into frames and the frames are fed into Lane detection and Object Detection NNs.

```
while success:
    success, image = vidObj.read()
    if count%500 == 0:
        cv2.imwrite(fr"frames{count}.jpg",image)
        cv2.imwrite(fr" frames{count}.jpg",road_lines(image))
        seg_vals = seg.segmentImage(fr"frames{count}.jpg", segment_target_classes
= target_classes, show_bboxes=True, output_image_name=fr"frames{count}.jpg")

        ip,msg,index = getattr(mod,func_name)(seg_vals,count,filename)

        inputs = np.append(inputs,np.array([ip]),axis = 0)
        outputs = np.append(outputs,np.array([[index]]),axis = 0)
```

The extracted frames are simultaneously stored in different folders. The actual path is not shown in the code as it reveals the details of the system. The method “[road\\_lines](#)” takes in an image and detects the lanes in it and writes the lane mask on the image and it is saved into a folder. The method [segmentImage\(\)](#) is a built-in function in [pixellib](#)<sup>[6]</sup> library which is used for instance segmentation of the image. The variable ‘seg\_vals’ stores the output values of the method as a dictionary that contains classes of objects detected, bounding box coordinates.

The image is also added into database at every iteration with the line of code below.

```
db.session.add(img)
```

This database is created with SQLite database<sup>[7]</sup>.

The logs of the system are also saved in Excel sheet for later use. Below code shows how the data is stored into excel sheet. Once all the data is written into Excel sheet the database session is committed to save the database.

```
workbook = xlswriter.Workbook(r'log_file.xlsx')

worksheet = workbook.add_worksheet("My sheet2")

for i,j in zip(inputs,outputs):
    col = 0
    for k in i:
        worksheet.write(row, col, k)
        col += 1
    worksheet.write(row, col+1, j)
    row += 1
workbook.close()
db.session.commit()
```

After the system detects the objects and makes decision it is displayed to the user to accept if the system has performed correctly and takes the response from the user. User response is recorded and stored in database. If there are any responses where the user records it as a wrong result, then all such images are taken and annotated separately with correct output and the neural network is retrained once again if the new data contains at least 400 images. This retraining is done manually as it requires the annotations and human intervention in deciding the correct output.

The decision of the Decision Making NN is also written into existing KB as triples for future use. This is done using the code below.

```
df = pd.read_excel(r'log_file.xlsx', sheet_name='My sheet2',usecols="A:J")
d = pd.read_excel(r'log_file.xlsx', sheet_name='My sheet2',usecols="L")
r = pd.read_excel(r'log_file.xlsx', sheet_name='My sheet2',usecols="N")

ip = df.to_numpy()
op = d.to_numpy()
res = r.to_numpy()

n = Namespace("http://example.org/ttab/")
g = Graph()
g.parse('ltademo.owl',format="ttl")
g.bind("log", n)
```

To write the triples into existing KB i.e., “[ltademo.owl](#)”, it is parsed initially into a variable and Namespace URI is created for the new log triples to be added into KB. The log data is taken from the excel sheet which was created before while running the KBANN. Using this data triples can be added into KB using the example code below.

```
g.add((id,n.lane_dec, lane_dec))
g.add((id,n.lane_inc, lane_inc))
g.add((id,n.lt_sd, lt_sd))
g.add((id,n.lt_veh, lt_veh))
g.add((id,n.rt_sd, rt_sd ))
g.add((id,n.rt_veh, rt_veh ))
g.add((id,n.a_sd, a_sd))
g.add((id,n.ah_veh, ah_veh))
g.add((id,n.ll_edge, ll_edge))
g.add((id,n.rl_edge, rl_edge))
g.add((id,n.message, Literal(message[vals[0]])))
```

There is an additional condition that these triples get added into the KB only when the user response is yes, which means that only correct decisions are added into the KB. After adding the triples into the parsed variable, the new triples along with the old KB are written into a new file so that the original file is kept in case of any further use.

```
g.serialize('output_file.ttl',format='ttl')
```

This line of code adds the triples of old KB and new log triples into a new file named “[ouput.ttl](#)” with turtle syntax. Below text shows how the triples are added to the existing Knowledge Base (fig:36).

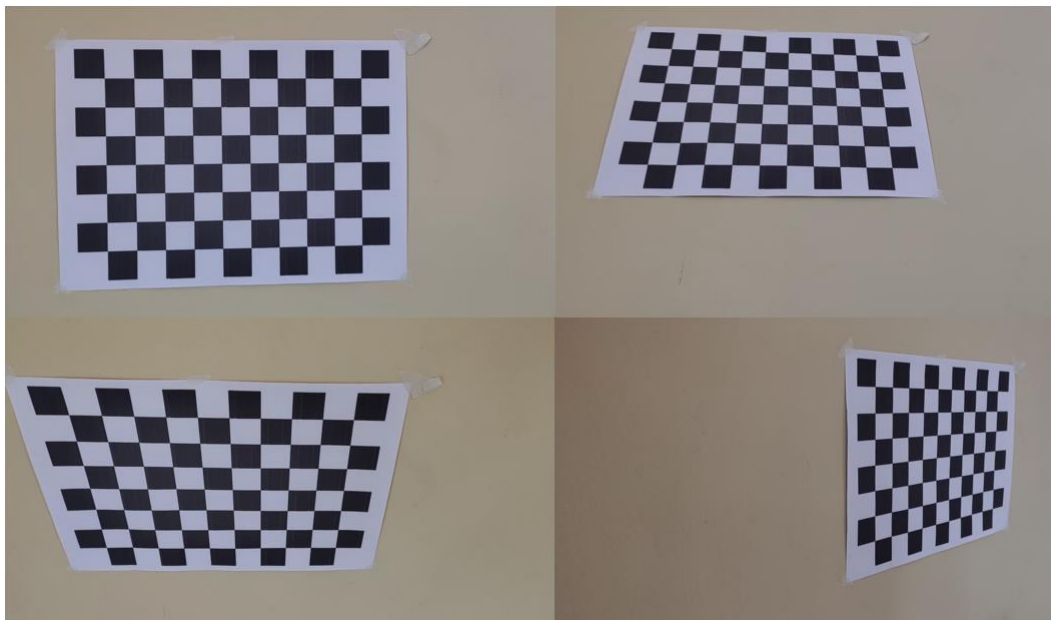
```
@prefix lcs: <http://www.semanticweb.org/saitejeswar/ontologies/2021/2/untitled-ontology-2#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ttab: <http://example.org/ttab/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ttab:1 ttab:a_sd "1" ;
ttab:ah_veh "0" ;
ttab:lane_dec "1" ;
ttab:lane_inc "0" ;
ttab:ll_edge "0" ;
ttab:lt_sd "1" ;
ttab:lt_veh "1" ;
ttab:message "Do not Steer" ;
ttab:rl_edge "1" ;
ttab:rt_sd "0" ;
ttab:rt_veh "1" .
```

Figure 36: Knowledge Base changing

## Testing and Results

The system is tested on a video which is recorded on a National Expressway which had 3 lanes. The video was shot on Vivo 1917 model smartphone. It lasted about 46 seconds and the frame rate was 30fps. The dimensions of the video are 1280x720 pixels. The same phone is calibrated using a traditional method with the help of MATLAB camera calibrator tool. The method involves taking picture of a checkerboard in different angles and then obtain the camera parameters. The pictures below (fig:37) show some of the images used for calibration and the parameters of the camera (fig:38) obtained from MATLAB.



*Figure 37: Calibration Images*

cameraIntrinsics with properties:

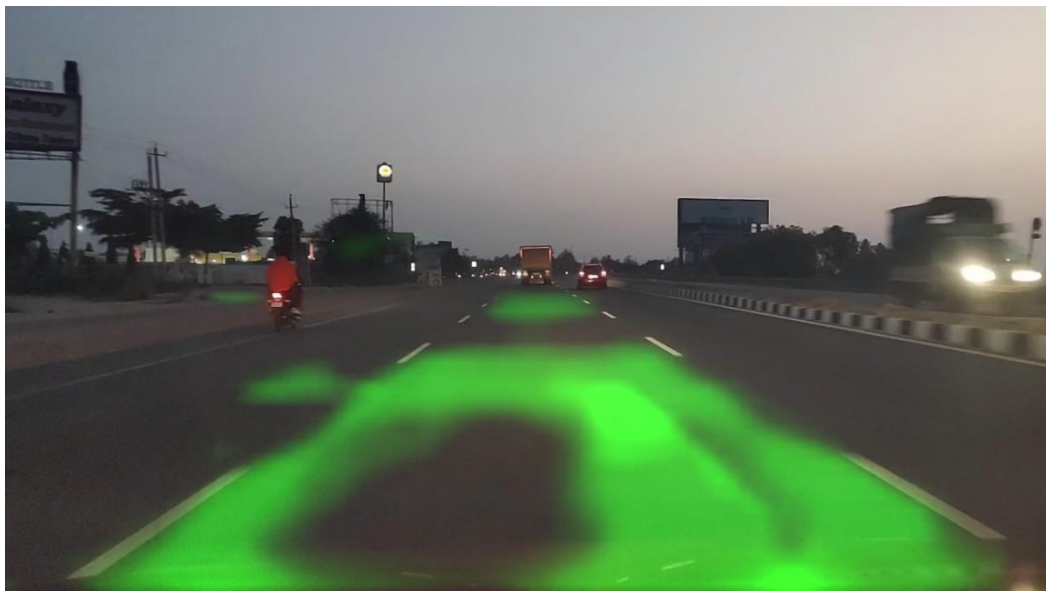
```
FocalLength: [1.1276e+03 1.1302e+03]
PrincipalPoint: [652.0649 365.2392]
ImageSize: [720 1280]
RadialDistortion: [0.0045 0.0624]
TangentialDistortion: [0 0]
Skew: 0
IntrinsicMatrix: [3x3 double]
```

*Figure 38: Camera Parameters*

The results from the Lane detection NN are shown in the images below (fig: 39&40).



*Figure 39: Lane detection Result sample 1*



*Figure 40: Lane detection Result sample 2*

The mask of the lane on the image is patchy because we are looking at each frame of the video. If all these images are made into a video, then due to high frame rate and constantly changing images these patches are not visible to human eye. As far as this project is concerned, the NN showed satisfactory results than the results obtained using OpenCV.



The results of Object detection NN are shown in the images below (fig: 41 & 42).



*Figure 41: Object Detection result sample 1*



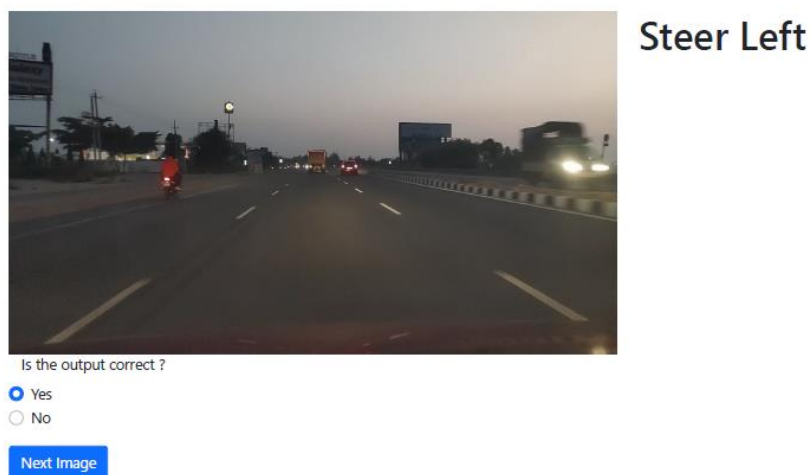
*Figure 42: Object Detection result sample 2*

In some cases, the NN was showing wrong detections due to poor lighting conditions. These wrong detections show a considerable impact on decision making as the Decision Making NN takes 60% of the inputs from this NN. Although, there are some false positives in the detections, this opens a need for retraining the Object detection. To do this, we require a lot of data along with annotations which is a herculean task to accomplish.

The user interface of the system to record the user response displays the output of the Decision making NN and provides an option to record the response of the user. Below pictures (fig: 43 & 44) show some of the results of the interface.



*Figure 43: Recording User response interface sample image 1*



*Figure 44: Recording User response interface sample image 2*

In the above image even though there is no obstacle within the considered safety distance the system assists to steer left, this happens because of false positive detections in Object detection proving that the system's accuracy and performance is more dependent on the object detection than on the Decision Making NN.

## Conclusion and Further Developments

As said earlier, the system shows a larger reliance on Object detection NN than the Decision Making NN. Hence, the goodness of the system can be evaluated based on the performance of object detection NN. The data below shows the results of the object detection NN in the region of interest of the system for a test video shot on a highway.

Total number of images extracted from the video = 139

Total number of objects to be detected as desired by the human = 375

Number of correct detections = 292  $\equiv$  78%

Number of false positives = 56  $\equiv$  15%

Number of true negatives = 83  $\equiv$  22%

As we know that the safety of the passenger is primary objective of any autonomous vehicle, if there are false positives the decision making NN ensures utmost safety of the system and passenger as the latter is trained with safety as a primary concern. The main problem arrives, and the passenger is not safe when there are true negatives from the object detection. In this case certain decisions may create a havoc. So, a strong focus must be kept on reducing the true negatives of the object detection NN. This increases the safety of the system and makes the system more efficient.

In real-time testing true negatives can be reduced and/or avoided with the help of sensor fusion, that fetches the data from the various sensors fitted on the Ego vehicle. A SLAM approach would also reduce such inaccuracies in the system at the cost of computational time, energy, and money. Currently, a computer with 12gb RAM and 2gb GPU is taking around 15 seconds per detection. As this is the bottle neck of the whole pipeline any improvement in this section would reduce the total time substantially. A high-performance computing chip is necessary in real time to execute such computationally expensive processes.

Although there were some inaccuracies in case of object detection and lane detection the implementation of integrating the Neural networks with Knowledge Base was successful. The use of Knowledge Base along with NN would make the NN more intelligent and safer. Currently there are different types of assistance systems for vehicles of various manufacturers. For example, an assistance system in Audi named “Audi Side Assist”<sup>[8]</sup>, (fig:47) which informs the driver about the vehicles in the blind spot that are approaching the Ego Vehicle from behind. This system just gives the information of objects in blind spots but does not make any decision. The picture below shows how Audi side assist works.

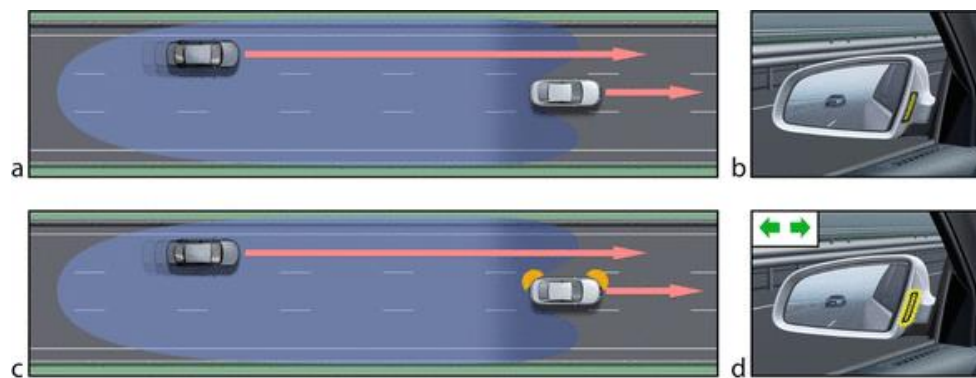


Figure 47: Audi Assist system

There is another system named “Lane change warning”<sup>[9]</sup> (fig:48) which was developed by BMW. This gives a warning to the driver if there is any potential danger on the road, be it an obstacle or any vehicle, while changing the lanes. The image below shows the working of the system.

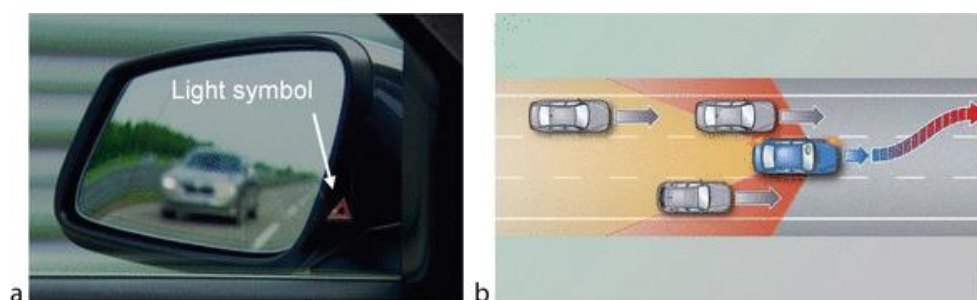


Figure 48: BMW Lane change warning

This system just warns the driver just like the Audi Side assist but does not make any decision which is implementable. Unlike, such assistance systems, the Lane change decision making system, which is developed in this project takes a decision based on the environment which can be implemented in level 3 or greater levels of autonomous vehicles.

An application is developed using MATLAB, that detects the lanes if an image is uploaded to it. In addition to the application a python library called “DrawLanes” is also developed along with the project so that it can be used in python by importing the “DrawLanes” library just like any other library. This library needs an additional installation of MATLAB engine which is run in background whenever the library is imported and any of its methods are called. Reader is requested to make a note that the application and the library are not very optimized to run-in low-end laptops as it requires hardware acceleration for faster execution. A link to download the source code of the library and all the instructions to setup and using the library is given in the appendix.

The performance of the developed system is not ideal as of now, because the whole system relies on just one camera and a big part on the object detection NN. This can be improved in real-time as there would be an additional data from the Radars, Lidar and Sonars fitted on to the Ego Vehicle. The data from these sensors will help the system to understand more clearly about the environment and reducing the false positive object detections through sensor fusion. The decision-making neural network can be trained with more conditions for more accurate and precise decision making. This system can be added with path planning algorithms to perform the action of lane changing which takes us a step closer towards complete automation. As the system develops to be more accurate, it will require more powerful computing chips to do the action faster and safer.

## Appendix

This section contains the link to see the ontology of this project. The whole code is not added in appendix as the main parts of the code are already added in various sections of this report and to protect the work of author. Check the KB by clicking the link below.

[Download KB](#)

[Link to download the DrawLanes python library and application](#)

## References

- [1] Wikipedia contributors. (2021, February 16). Knowledge engineering. In *Wikipedia, The Free Encyclopedia*. Retrieved 13:59, May 4, 2021, from [https://en.wikipedia.org/w/index.php?title=Knowledge\\_engineering&oldid=1007089524](https://en.wikipedia.org/w/index.php?title=Knowledge_engineering&oldid=1007089524)
- [2] *RDF 1.1 Turtle*. Retrieved May 4, 2021, from <https://www.w3.org/TR/turtle/>
- [3] *OWL 2 Web Ontology Language Structural Specification and .* Retrieved May 4, 2021, from <https://www.w3.org/TR/owl2-syntax/>
- [4] *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Retrieved May 4, 2021, from <https://www.w3.org/Submission/SWRL/>
- [5] *matterport/Mask\_RCNN: Mask R-CNN for object detection .. - GitHub*. Retrieved May 4, 2021, from [https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN)
- [6] *PIXELLIB*. Retrieved May 4, 2021, from <https://pixellib.readthedocs.io/>
- [7] *SQLAlchemy - The Database Toolkit for Python*. Retrieved May 4, 2021, from <https://www.sqlalchemy.org/>
- [8] Popken, M. (2006). AUDI side assist. *Hanser Automotive electronics+ systems*, 7(7-8), 54-56.
- [9] Ehmanns, D., Aulbach, J., Strobel, T., Mayser, C., Kopf, M., Discher, C., ... & Orecher, S. (2008). Active Safety and Driver Assistance. *ATZextra worldwide*, 13(8), 114-119.