COMMUNITY PROJECT REPORT
ON
VOICE ASSISTED SELF DRIVING CAR

Submitted in partial fulfilment of the requirements for the completion of course of

COMMUNITY PROJECT

SUBMITTED BY:

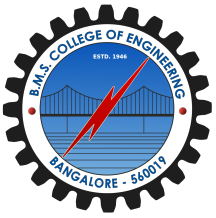
Anirudh G J	[1BM15EE006]
Bharath Y P	[1BM15EE013]
Lakshwin Shreesha	[1BM15EE026]
Siddharth K	[1BM15EE052]

Under the guidance of,

Dr P Meena

Professor

Department of Electrical and Electronics Engineering, BMSCE



Department of Electrical and Electronics Engineering

BMS COLLEGE OF ENGINEERING

(Autonomous College under Visvesvaraya Technological University, Belgaum)

Bull Temple Road, Basavangudi, Bangalore -560019

ACKNOWLEDGEMENT

Any achievement, be it scholastic or otherwise does not depend solely on the individual efforts but on the guidance and cooperation of people around.

Many people in their own capacities have helped us in carrying out this community project. We owe a debt of gratitude to each one of them.

We would like to thank our guide Dr P Meena , Professor , Department of EEE, BMSCE, Bangalore for the vision and foresight which inspired us to conceive this community Project. One simply couldn't wish for a better or friendlier supervisor.

TABLE OF CONTENTS

GLOSSARY

1. **Environment** : It is a world (virtual or real) where an agent explores it so as to understand how the world works. (Example: A house)
2. **State** : Every position which the agent can take in the environment is defined by unique vector (Example: Position of a robot inside the house)
3. **Action** : The agent can explore the environment by trying to change its state using actions inside the environment. (Example: Robot closing the door in the house)
4. **Agent** : Agent is defined as an entity which is explores the environment by performing some actions in the environment.
5. **Reward Function** : For every action taken by the agent in the world, the environment returns a reward which says how good was it to take that action from a given state.(Example: Robot performing an action of colliding with the wall will yield a negative reward and similarly positive reward if it performs an action of avoiding the walls in the world).
6. **Policy Function** : This function is responsible for making decisions or performing actions based on the state at which the agent is inside the environment so as to obtain maximum rewards.
7. **Value function** : This function gives an estimate on how good it is to be in a given state.
8. **Reinforcement Learning** : This is a learning algorithm which makes the agent to take suitable action in the environment so as to yield maximum reward at the end of the process.
9. **Inverse Reinforcement Learning** : This is a technique used to model the reward function for complex environments where the reward function is also complex.
10. **Neural Network** : A Function approximator where input is mapped to output by training features/weights in the network by backpropagation algorithm.
11. **Genotype :** A series of rules for constructing the neural network
12. **Phenotype :**The neural network encoded by the genotype

ABSTRACT

Reinforcement learning (RL) is the very basic and most intuitive form of trial and error learning, it is the way by which most of the living organisms with some form of thinking capabilities learn. It is the way by which a new born human baby learns to take its first steps, that is by taking random actions initially and then slowly figuring out the actions which lead to the forward walking motion. Basically the agent tries to choose its actions in such a way that the rewards that is gets from that particular behavior are maximized. Now to make the agent perform different behaviors, it is the reward structure that one must modify/exploit. But assume we only have the knowledge of the behavior of the expert with us, then how do we estimate the reward structure given a particular behavior in the environment? This is where **Inverse Reinforcement Learning** comes into picture,given the optimal expert policy,we try to determine the underlying reward structure.Inverse reinforcement learning will try to infer the goal of the teacher.We would be able to use expert data to start with a baseline policy that can be safely improved over time. This baseline policy should be significantly better than a randomly initialized policy, which speeds up convergence.The benefit of this method is that at each stage, the policy being tested is the best estimate for the optimal policy of the system.The goal is to learn from a wide range of human data and perform tasks that are beyond the abilities of human experts. Our inspiration to navigate a car with expert demonstrations arises from the Inverse Reinforcement learning paradigm and this report presents several algorithms for the same and their drawbacks.

INTRODUCTION

In reinforcement Learning the agent tries to learn the policy(set of rules to reach a goal) for a designed reward function.Inverse Reinforcement Learning does the exact opposite in that it seeks to learn the reward function from a given behavior or policy ie it seeks to model the preferences of another agent using its observed behavior, thereby avoiding a manual specification of its reward function. IRL due to its advantages has attracted several researchers in the communities of artificial intelligence, psychology, control theory, and machine learning. IRL is appealing because of its potential to use data recorded in everyday tasks (e.g., driving data) to build autonomous agents capable of modeling and socially collaborating with others in our society.In this project we study this problem and associated advances in a structured way and apply it for an application for self driving.Consider a self driving car in the position B as shown in the figure.For the car B has to safely merge into the congested road is should learn the behavior of the car at the position A.Previously collected trajectories of cars in position A, near freeway entry ramps maybe useful to learn the preferences of a typical driver as it approaches a merge.In cases like this designing a reward function would be a complex task , hence Inverse Reinforcement Learning can be used to model the Reward Function indirectly.

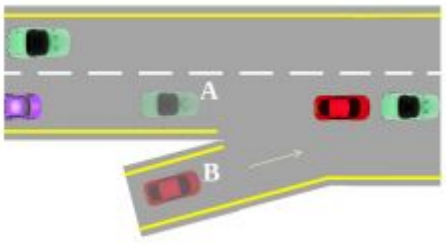


Figure showing one of the use cases of Inverse Reinforcement Learning.Red car at position B is trying to merge into the lane, and Green car at position A is the immediate traffic. The transparent images of cars show their positions before merging, and the opaque images depict one of their possible positions after the merger.

PROBLEM STATEMENT

Programming an autonomous vehicle for every unique scenario encountered in real world is a humongous task, time consuming and a very inefficient method to achieve the task of autonomous driving. Artificial Intelligence techniques such as Machine Learning algorithms can be used to solve these kind of problems where one need not program an Artificial intelligent agent for unique scenarios.

Widely used machine learning algorithms such as Supervised Learning and Unsupervised Learning also has its own disadvantages of not adjusting to new scenarios. Reinforcement Learning and Inverse Reinforcement Learning algorithms are used to achieve the state of truly autonomous driving where the agent can adjust itself to scenarios which were never seen before.

Our goal is to teach an autonomous vehicle to drive in a complex environment by observing the demonstrations (consisting of State features such as images, voices and actions taken at every state) performed by using Inverse Reinforcement Learning algorithm.

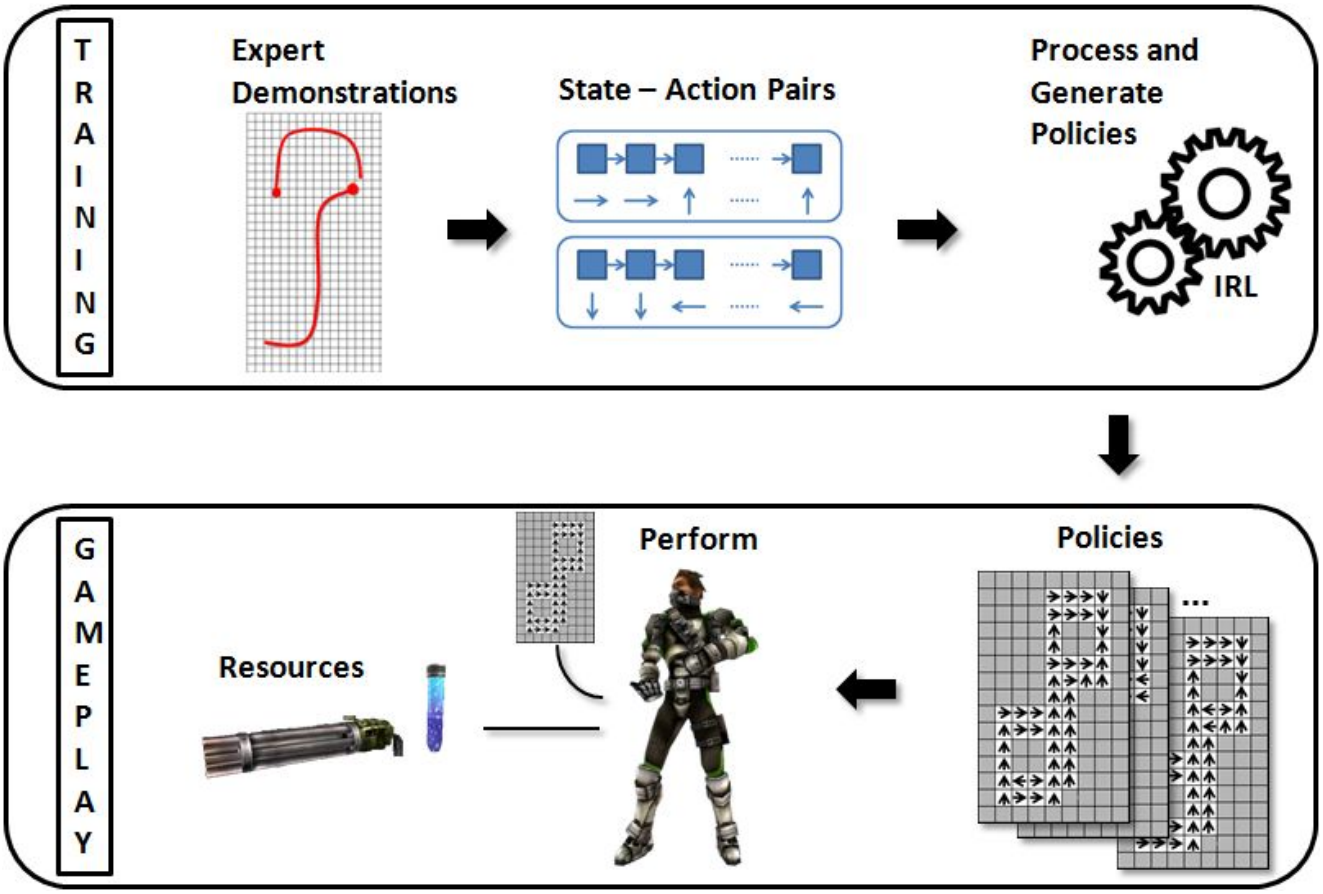
PROBLEM DEFINITION

An MDP is a set of states (S), actions (A) and transition probabilities (θ) between states when an action is taken in a state. Additionally, each state-action pair corresponds to a reward (R). A discount factor (γ) is used while aggregating rewards corresponding to a trajectory of state action pairs. A policy (π) describes a set of actions to be taken over the state space. The optimal policy ($\pi *$), then, maximizes the expected discounted sum of rewards between two given states (start and goal) in an episodic task (which repetitively solves the same problem).State value (v) is the expected return (sum of discounted R values) when an arbitrary π is followed, starting at that state.

The goal of IRL is to learn a reward function for each state based on parts of a given policy (a demonstration). In a broader sense, the goal is to be able to generate a policy over a state space (S), which is correlated to what has been demonstrated. For the purpose of this work, the demonstrations received by the algorithm are assumed to be performed by an expert, meaning that they are assumed to be optimal. A demonstration (D) consists of numerous examples, each of which is a trace of the optimal policy through state space. These are represented in the form of sequences of state-action pairs (s, a).

One method to generate a policy is to generate state values for each state based on state features. It is assumed that a weighted combination of state features can provide a quantitative evaluation of a state.The first problem, then, is to learn a mapping from state features to state values that produces a policy for which state-action pairs are consistent with the given examples.The second problem, then, is to learn a non-linear mapping from these values to state reward, which produces a policy consistent with the given examples (as described for the first problem).

THE OVERALL BLOCK DIAGRAM

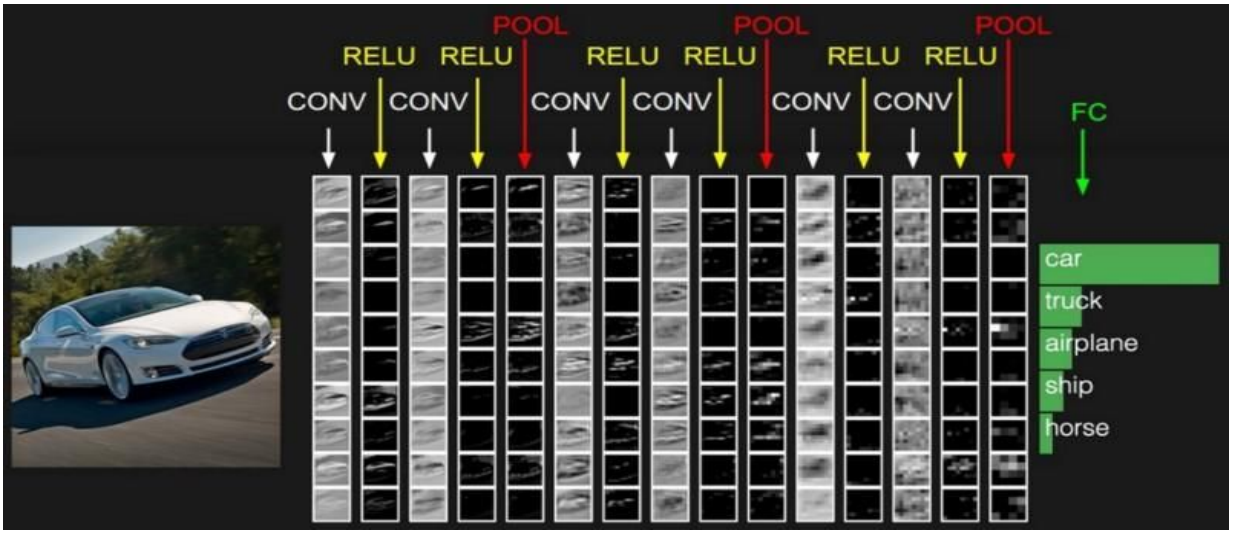


LITERATURE SURVEY

1) END TO END LEARNING WITH BEHAVIORAL CLONING

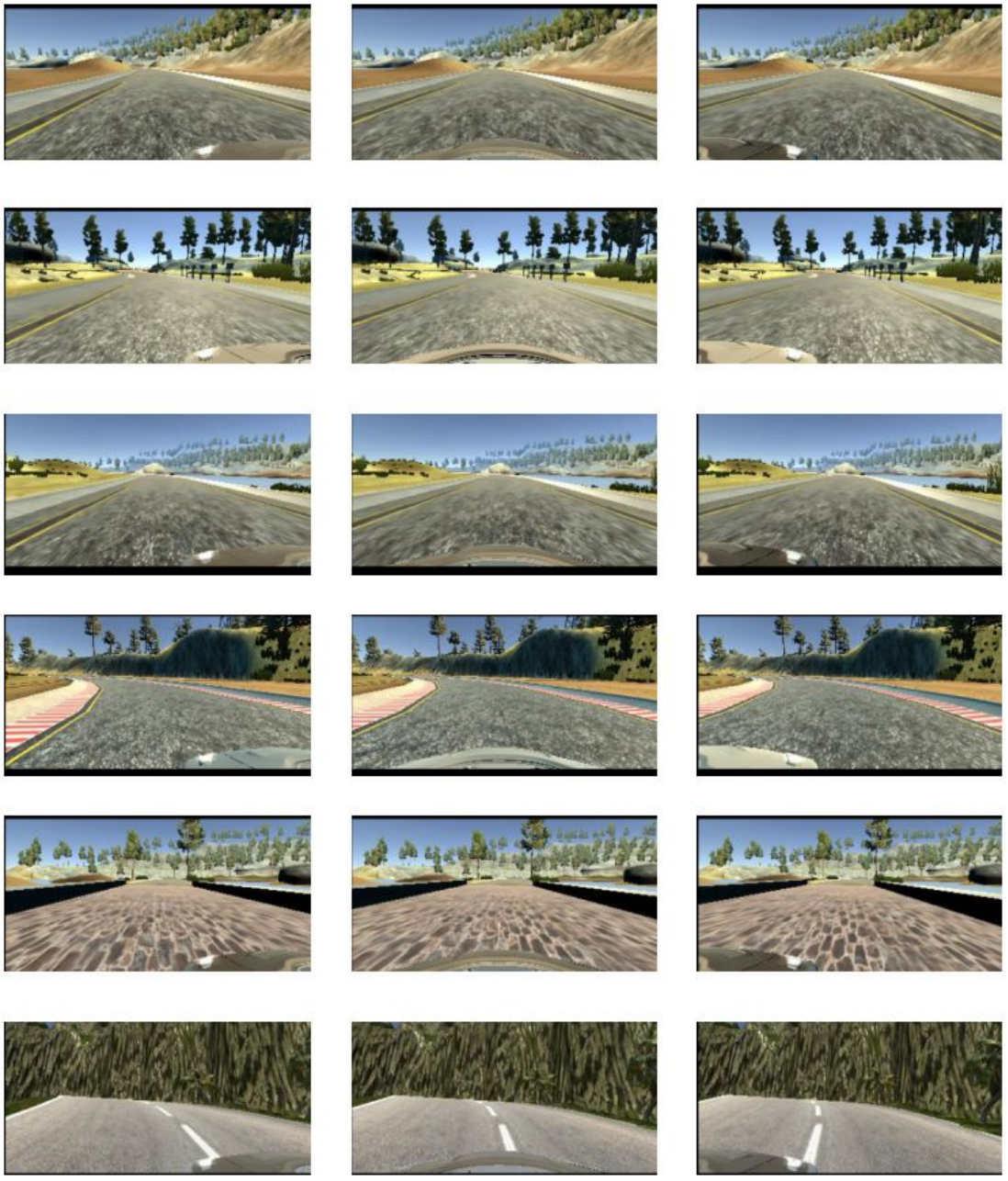
A. INTRODUCTION

Behavioral cloning is a method by which human sub cognitive skills can be captured and reproduced in a 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. The learning program outputs a set of rules that reproduce the skilled behavior. This method can be used to construct automatic control systems for complex tasks for which classical control theory is inadequate. It can also be used for training. Here the learning program is a Convolutional Neural Network. [1] CNNs, like neural networks, are made up of neurons with learnable weights and biases. Each neuron receives several inputs, takes a weighted sum over them, pass it through an activation function and responds with an output. The architecture of these networks makes them best suited for pattern recognition in images.



Typical Architecture of a CNN

The goal is to teach a Convolutional Neural Network (CNN) to drive a car in a simulator. The simulator used was the Self Driving simulator provided by Udacity, built on the Unity engine. The car is equipped with three cameras that provide video streams and records the values of the steering angle, speed, throttle and brake. The steering angle is the only thing that needs to be predicted, but more advanced models might also want to predict throttle and brake. This turns out to be a regression task, which is very different from usual applications of CNNs for classification purposes.



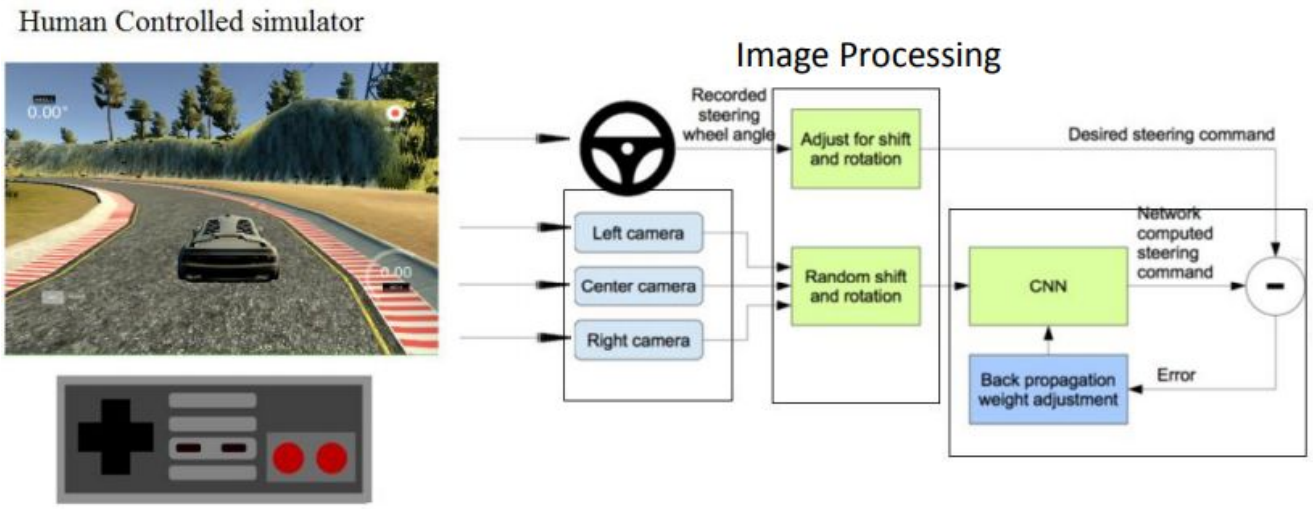
Camera Views from the simulator

This amount of data is collected my manually controlling the car, ie manually guiding the car through the circuit and capturing the parameters like throttle, steering angle, braking force, etc for every time frame while simultaneously capturing three images. These images are treated as the training data set for the CNN.

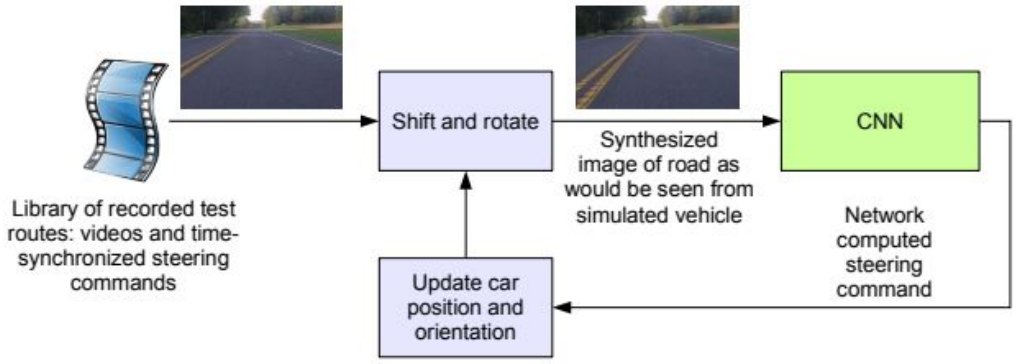
B. IMPLEMENTATION

i. Block Diagram:

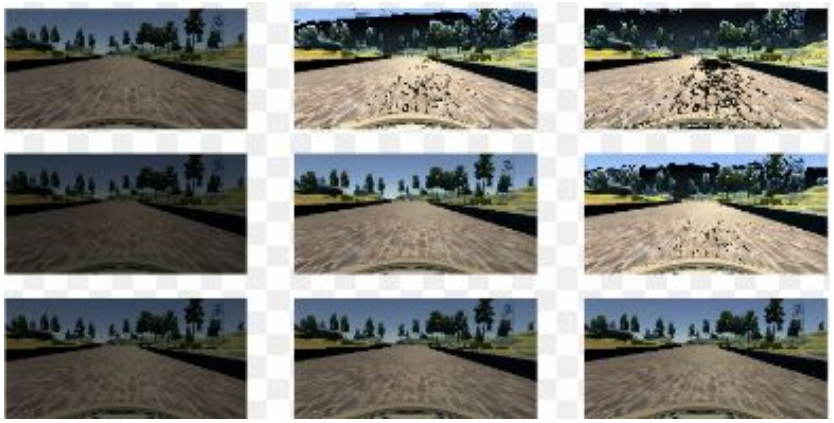
a. Training:



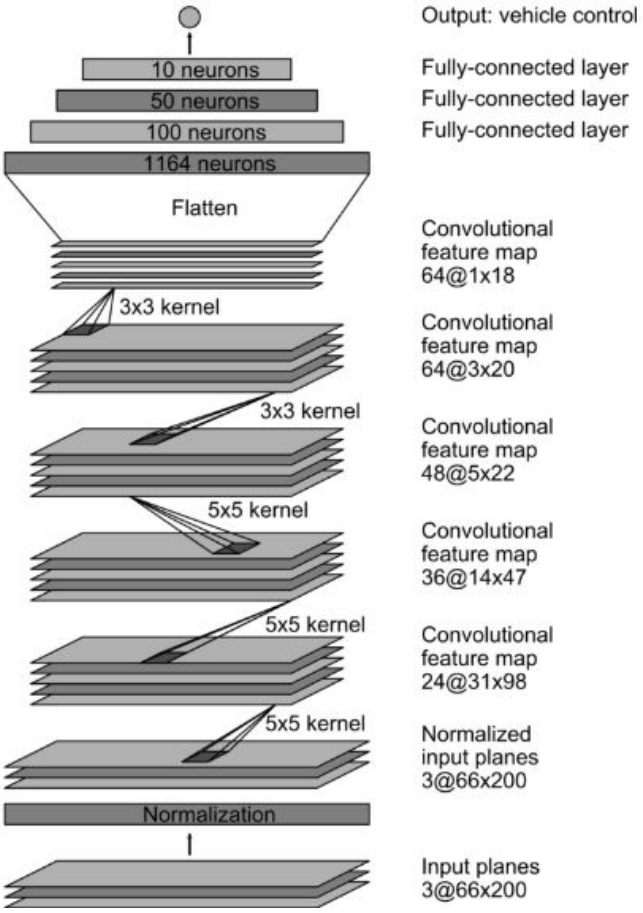
b. Testing:



ii. Data Augmentation and Image Processing:



Data Set Preparation: Data pre-processing and Augmentation before giving to Neural Network. After selecting the final set of frames we augment the data by adding artificial shifts and rotations to teach the network how to recover from a poor position or orientation.
CNN Architecture Used:

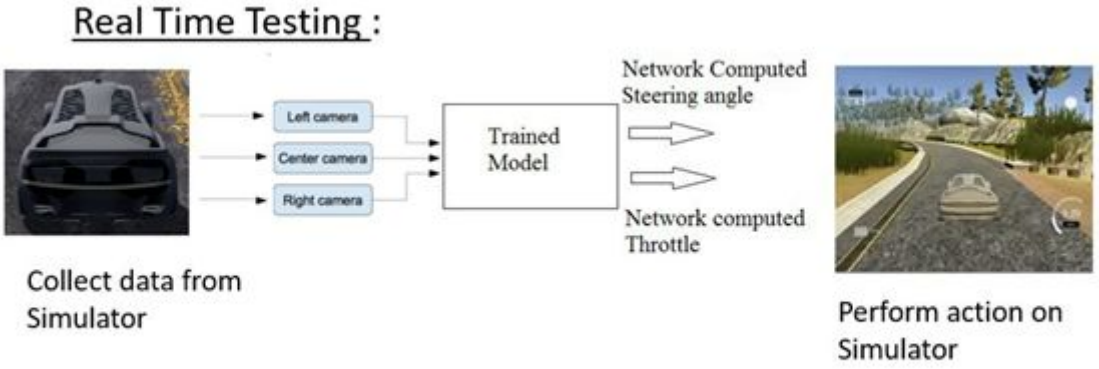


We train the weights of our network to minimize the mean squared error between the steering command output by the network and the command of either the human driver, or the adjusted steering command for off centre and augmented images.

The network consists of 9 layers, including a normalization layer, 5 convolutional layers and 3 fully connected layers. The input image is split into RGB planes and passed to the network. The first layer of the network performs image normalization. The normalizer is hard-coded and is not adjusted in the learning process. Performing normalization in the network allows the normalization scheme to be altered with the network architecture and to be accelerated via GPU processing.[2] The convolutional layers were designed to perform feature extraction and were chosen empirically through a series of experiments that varied layer configurations. We use strided convolutions in the first three convolutional layers with a 2×2 stride and a 5×5 kernel and a non-strided convolution with a 3×3 kernel size in the last two convolutional layers. We follow the five convolutional layers with three fully connected layers leading to an output control

value which is the inverse turning radius. The fully connected layers are designed to function as a controller for steering, but we note that by training the system end-to-end, it is not possible to make a clean break between which parts of the network function primarily as feature extractor and which serve as controller. The simulation results can be seen in [3].

iii. Testing on Trained model :



C. Conclusion and Remarks:

Demonstrated that 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.

The following were the drawbacks:

• Primitive approach of mapping pixel values to steering angle

• Not learning motives of user

• Dependent on environment, bad results on different environment

2) MAXIMUM ENTROPY INVERSE REINFORCEMENT LEARNING

A. INTRODUCTION

Inverse Reinforcement Learning algorithm solves the task of obtaining an approximate Reward Function. Maximum Entropy Inverse Reinforcement Learning approximates a neural network with approximates a Reward function of an environment with the help of expert trajectories.

The expert trajectories consist of an array of demonstration where the expert's movements and states are recorded at every steps.

The reward function generated by the algorithm will be modelled in such a way that the expert's demonstrations would receive a maximum reward. Reinforcement Learning algorithm is then used to obtain a Policy Function which yields a high reward for the given reward function.

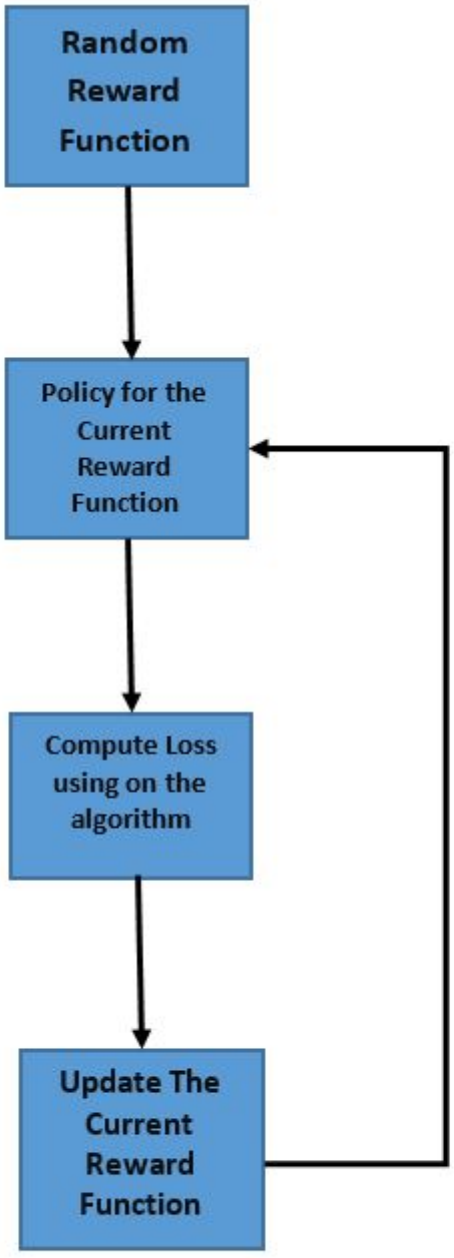
The Policy function is compared with the expert's demonstrations and Loss is computed. The gradient obtained from this loss function is then used to tune the reward function which gives better rewards to expert's trajectories. The policy function is again evaluated for the new reward function in the next iteration and the process repeats until convergence criteria is reached. The reward function generated is said to be converged when the policy function actions is almost similar to expert's actions.

B. ALGORITHM OVERVIEW

Algorithm

$\theta_0 \leftarrow RewardParameterInitialization$
for $i = 1 : N$
  $R_i \leftarrow UpdateReward(\theta_i)$
  $\pi_i \leftarrow UpdatePolicy(R_i)$
  $E \leftarrow PolicyPropagation(\pi_i)$
  Compute $\nabla L(\theta)$
  $\theta_0 \leftarrow UpdateRewardParameter(\nabla L(\theta))$

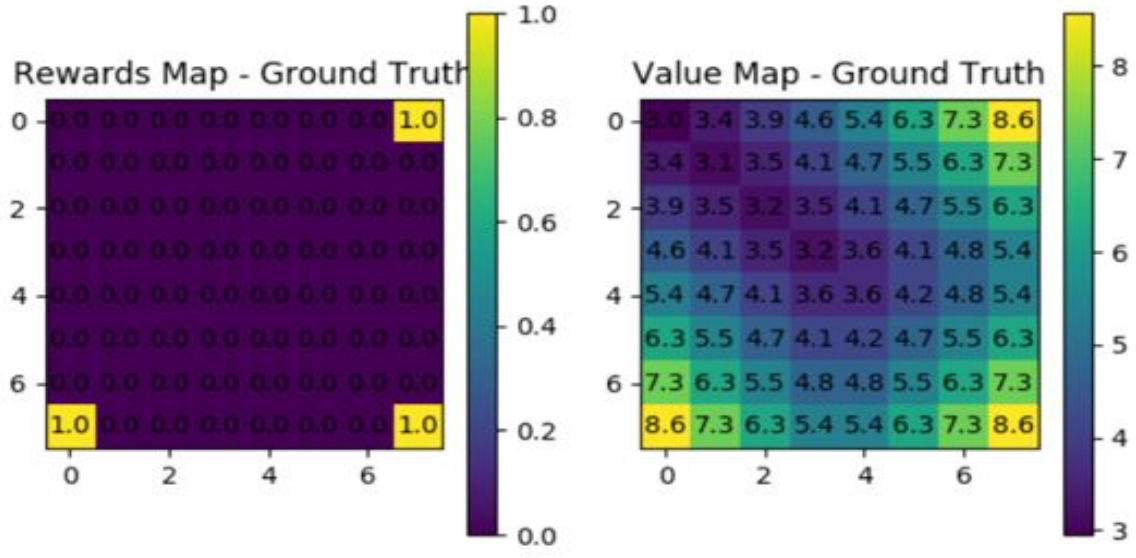
C. BLOCK DIAGRAM



Block Diagram of the Algorithm
Maximum Entropy Inverse Reinforcement Learning

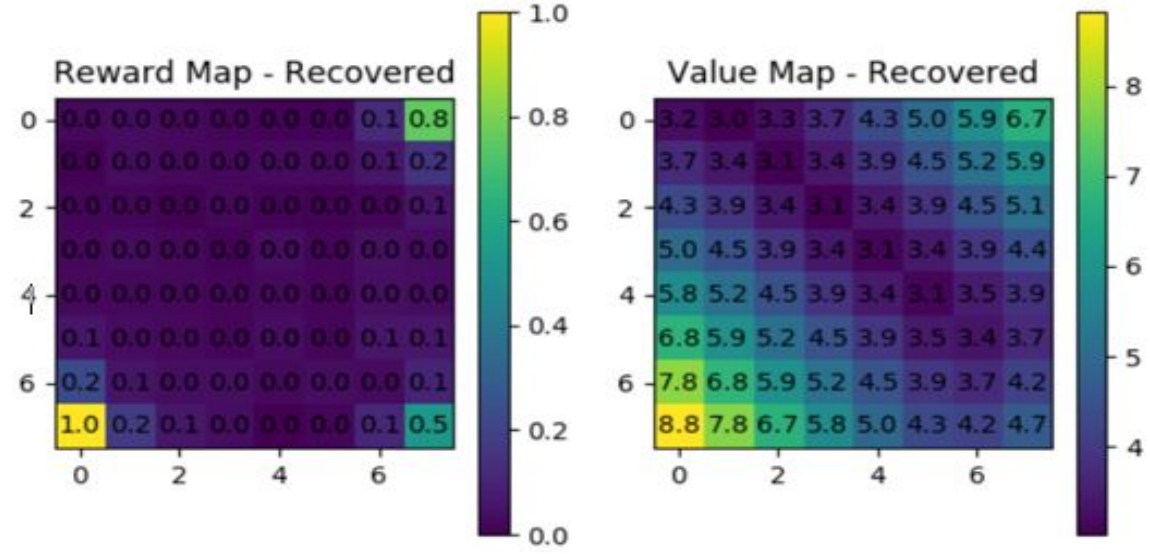
D. WORKING PRINCIPLE OF THE ALGORITHM

GridWorld Environment: The environment consists of set of cells which the agent can be in and would be able to move to neighbouring cells by four actions (Go up, down, left, right). The Reward would be high if the agent moves to either of the state (top right, bottom right, bottom left) as shown in the figure. Value map would be appropriately generated based on the reward map. Value map shows how good it is to be in a given cell, higher the value, closer to the cell which gives high reward. The four labelled diagrams shown below are self explanatory which consists of reward function obtained from the algorithm and value map obtained from that reward function.



Ideal Reward function for the grid world

Value Map for the given reward function which determines the policy



Recovered Reward function obtained from the algorithm

Value map generated for the Recovered Reward Function

This algorithm will be used for Training an Autonomous vehicle which learns from the demonstrations provided by the expert. The expert drives the vehicle by following all the traffic rules, avoid hitting pedestrians, stick to the lane, driving under the speed limit at all times etc.
The algorithm looks at the demonstrations provided by the expert and generates an optimal reward function which considers the driving rule which the expert considered. This optimal function can then be used to generate a policy function which the autonomous vehicle can follow.

E. EXPERIMENTAL RESULTS

The following algorithm was implemented in Cartpole environment where an expert(us) recorded the series of state and action pair as demonstrations and the algorithm generated a reward function based on the expert's demonstrations. The generated reward function was then used to obtain an optimal policy using Maximum Entropy Inverse Reinforcement Learning. The behaviour of the optimal policy obtained showed that it had understood the motives behind the expert's demonstrations , that is, to try balancing the pole by moving left or right.

Environment Specifications : -

CartPole Environment

Observation Vector (4x1)	Min	Max
Cart Position	-2.4	2.4
Cart Velocity	-Inf	Inf
Pole Angle	~ -41.8°	~ 41.8°
Pole Velocity At Tip	-Inf	Inf

Discrete Action (1x1)	Vector
Push cart to the left	[0]
Push cart to the right	[1]

Figure showing cartpole environment

Ideal Reward Function which the Expert Follows:

- +1 Reward if the cart position lies between -2.4 to 2.4 (within the edge of the screen) and if the pole angle is less than 12°.
- -1 Reward if the above case is not True. The game terminates if the agent receives -1 Reward.

3) APPRENTICESHIP LEARNING VIA INVERSE REINFORCEMENT LEARNING

A. INTRODUCTION

Apprenticeship learning derives its basis from the Inverse Reinforcement learning (IRL) formalism, wherein given a sequential decision making problem posed in the Markov decision process (MDP) formalism, a number of standard algorithms exist for finding an optimal or near-optimal policy. In complex tasks, say highway driving even the reward function is frequently difficult to specify manually and apprenticeship learning at its core achieves this by formulating a linear reward function which is based on several desiderata, such as maintaining safe following distance, keeping away from the curb, staying far from any pedestrians, maintaining a reasonable speed, perhaps a slight preference for driving in the middle lane, not changing lanes too often, etcetera.

This algorithm explored is based on the work presented in Apprenticeship learning, (Abeel and Ng) [1] wherein the expert is trying (without necessarily succeeding) to optimize an unknown reward function that can be expressed as a linear combination of known "features." Even though the results presented at the end does not guarantee that the algorithm will correctly recover the expert's true reward function, it is shown that the algorithm will nonetheless find a policy that performs as well as the expert, where performance is measured with respect to the expert's unknown reward function.

B. UNDERSTANDING THE MATH

A finite-state Markov decision process (MDP) is a tuple (S, A, T, γ, D, R), where S is a finite set of states, A is a set of actions, T = {Psa} is a set of state transition probabilities (here, Psa is the state transition distribution upon taking action a in state s), $\gamma \in [0, 1)$ is a discount factor, D is the initial-state distribution, from which the start state s0 is drawn and R : S 7→ A is the reward function, which is assume to be bounded in absolute value by 1. The MDP\R denotes an MDP without a reward function, i.e., a tuple of the form (S, A, T, γ, D). The algorithm assumes that there exists a set of features

φ : S → [0, 1]k over states, and that there is some "true" reward function R∗ (s) = w ∗ · φ(s), where $w * \in R k$. 3, where φ contains the sensor readings from the agent in question. It is also assumed that the weights are bounded in value by 1: ||w||* ≤ 1.

A policy π is a mapping from states to probability distributions over actions. The value of a policy π is

$$
\begin{aligned}
E_{s_0 \sim D}[V^{\pi}(s_0)] &= E[\textstyle\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi] && (1)\\
&= E[\textstyle\sum_{t=0}^{\infty} \gamma^t w \cdot \phi(s_t)|\pi] && (2)\\
&= w \cdot E[\textstyle\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] && (3)
\end{aligned}
$$

Here, the expectation is taken with respect to the random state sequence s0, s1, …Sn drawn by starting from a state s0 ~ D, and picking actions according to π. The expected discounted accumulated feature value vector μ(π), or more succinctly the feature expectations, can be formulated as:

$$\mu(\pi) = E[\sum_{t=0}^{\infty} \gamma^t \phi(s_t)|\pi] \in \mathbb{R}^k. \qquad (4)$$

The value function is expressed as

$$E_{s_0 \sim D}[V^{\pi}(s_0)] = w \cdot \mu(\pi).$$

From an estimate of the experts feature expectation vector $\mu_e(\pi) = \mu_e(\pi_e)$,
given a set of **m** trajectories **{s (i) 0 , s (i) 1 , . . .};** i=1,2..m, generated by the expert, the empirical estimate for μ_e is:

$$\hat{\mu}_E = \frac{1}{m} \sum_{i=1}^{m} \sum_{t=0}^{\infty} \gamma^t \phi(s_t^{(i)}). \qquad (5)$$

The cost function is optimized using a quadratic optimizer such as a neural net.

C. ALGORITHM

The problem is the following: Given an MDP\R, a feature mapping **φ** and the expert's feature expectations μ_e, find a policy whose performance is close to that of the expert's, on the unknown reward function $\mathbf{R^* = w^{*T} \varphi}$.

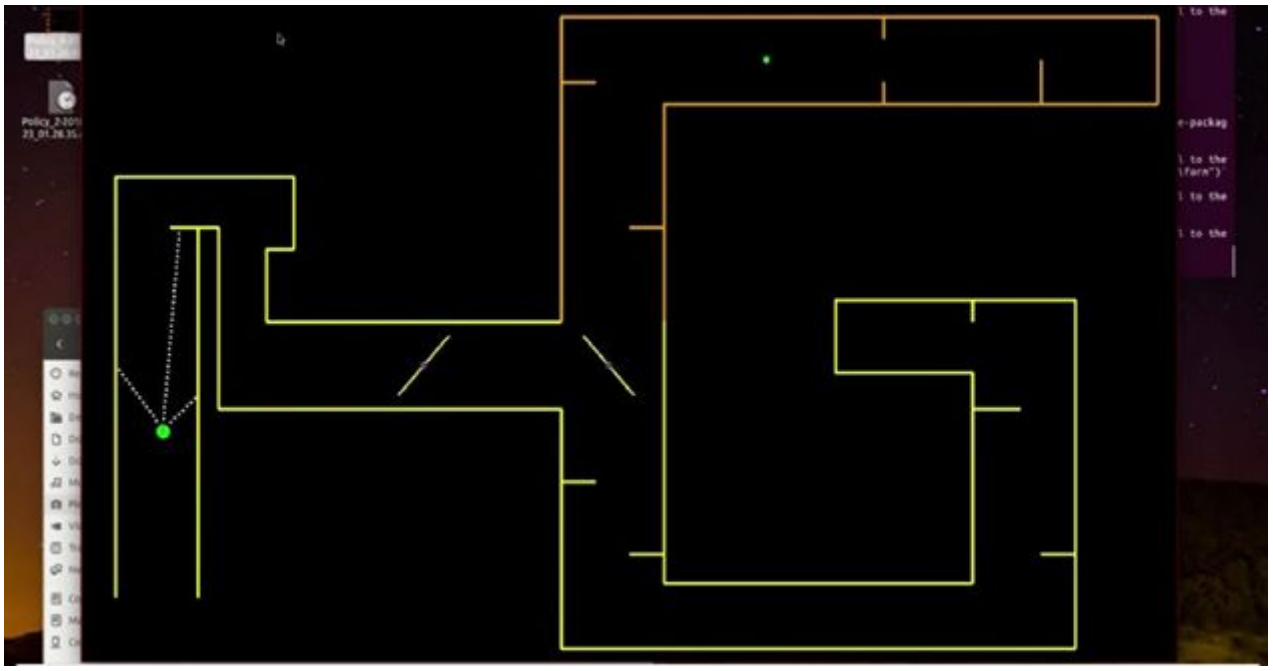
This is accomplished by finding a policy **π˜** such that $\|\boldsymbol{\mu}(\tilde{\pi}) - \boldsymbol{\mu}_E\| \leq$ **€.** For such a **π˜,**

The steps to determine the policy **π˜:**

1. Randomly pick some policy $\pi^{(0)}$, compute (or approximate via Monte Carlo) $\mu^{(0)} = \mu(\pi^{(0)})$, and set $i = 1$.
2. Compute $t^{(i)} = \max_{w:\|w\|_2 \leq 1} \min_{j \in \{0..(i-1)\}} w^T(\mu_E - \mu^{(j)})$, and let $w^{(i)}$ be the value of w that attains this maximum.
3. If $t^{(i)} \leq \epsilon$, then terminate.
4. Using the RL algorithm, compute the optimal policy $\pi^{(i)}$ for the MDP using rewards $R = (w^{(i)})^T \phi$.
5. Compute (or estimate) $\mu^{(i)} = \mu(\pi^{(i)})$.
6. Set $i = i + 1$, and go back to step 2.

Step 2 presented here is a neural network paradigm which treats the feature expectations as the input and adjusts the weights to reduce the error between the random policy **μ(π)** and $\boldsymbol{\mu}_E$.

D. IMPLEMENTATION



The above figure shows maze environment built to test the algorithm. The white dots represent the extent to which the agent's sensors extend

Testing environment: -

Agent: the agent is a small green circle with its heading direction indicated by a blue line.

Sensors: the agent is equipped with 3 distance/colour sensors, and this is the only information that the agent has about the environment.

State Space: the state of the agent consists of 8 observable features (**φ**) **:**

1. Distance sensor 1 reading
2. Distance sensor 2 reading
3. Distance sensor 3 reading
4. No. of sensors seeing black color (/3 to normalize)
5. No. of sensors seeing yellow color (/3 to normalize)
6. No. of sensors seeing brown color (/3 to normalize)
7. No. of sensors seeing red color (/3 to normalize)
8. Boolean to indicate a crash/bump into an obstacle. (1:crash, 0:alive)

Note, the normalization is done to ensure that every observable feature value is in the range [0,1] which is a necessary condition on the rewards for the IRL algorithm to converge.

Rewards: the reward after every frame is calculated as a weighted linear combination of the feature values observed in that respective frame. Here the reward r_t in the t_{th} frame is calculated by the dot product of the weight vector w with the vector of feature values in t_{th} frame, that is the state vector ϕ_t. Such that $R_t = w^T * \phi_t$

Available Actions: with every new frame, the agent automatically takes a *forward* step, the available actions can either turn the agent *left*, *right* or *do nothing* that is a simple forward step, note that the turning actions include the forward motion as well, it is not an in-place rotation.

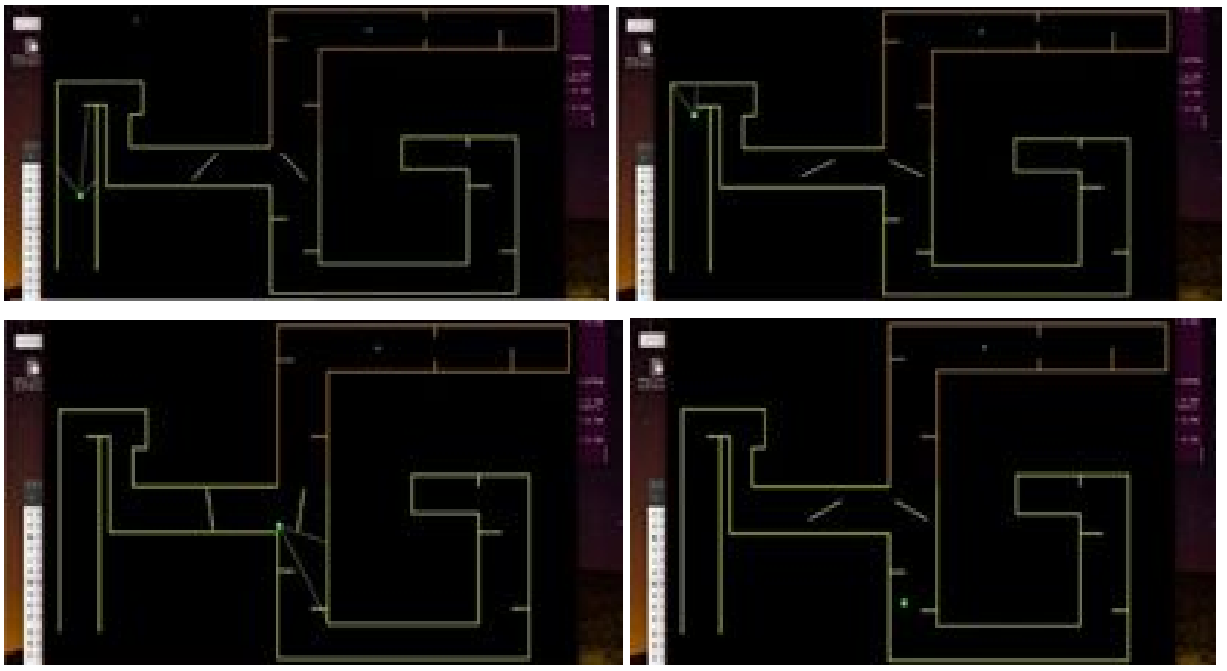
Obstacles: the environment consists of rigid walls with moving obstacles. The environment is designed in this way for easy testing of the IRL algorithm.

The Starting position(state) of the bot is fixed, as according to the IRL algorithm it is necessary that the starting state is same for all the iterations.

The test results are as follows:

The expert trajectory was provided by manually traversing the obstacles thereby providing the feature expert expectation vector.

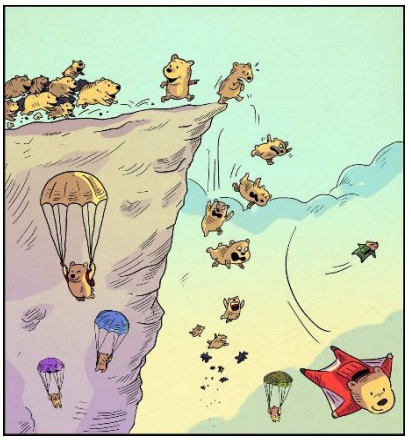
The 12th policy generated showed the best behaviour amongst all the policies generated by the algorithm:



The above sequence of images shows the best policy generated by the algorithm

4) NEURO EVOLUTION BASED INVERSE REINFORCEMENT LEARNING

A. **INTRODUCTION**



Evolutionary algorithms are a class of problem solving algorithms that work similar to biological evolution in that it improves a population of individuals instead of just one individual.There is an environment that can only support a certain number of individuals, these individuals compete for certain resources. Because the environment can not support a limitless population size, there has to be some selection which members of the species are able to survive and to reproduce.The individuals that are able to compete for the resources in the best way are favoured by this selection process.This is based on Darwinian Theory " **SURVIVAL OF THE FITTEST**".Some specific traits that makes a individual fitter than its rivals can be inherited, this provides a system where on average each generation is fitter than the previous one.

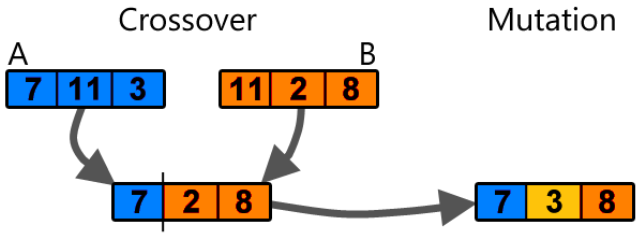
The ***phenotype*** is the form of the individual in the environment where its fitness is tested. The ***genotype*** is the encoding of the phenotype which can be inherited by offspring.For example in biology, Someone's body and mind would be considered the phenotype and someone's chromosomes would be considered the genotype.

Evolutionary algorithms provide a structured search through the solution space. It is an iterative procedure where each iteration equals a generation. In each generation individual solutions are evaluated, the best solutions recombined into offspring, the offspring is subject to some mutation and at last the best solutions are select to survive to the next generation. In this process the phenotype representation is the representation that is evaluated and the genotype is the representation that is mutated and recombined into offspring.

Due to this clear distinction between the genotype and phenotype evolutionary algorithms perform well in unsupervised learning problems and more complex problems.

B. METHODOLOGY

Genetic algorithms are a type of evolutionary is a very simple step beyond random guessing. It works as follows: Imagine if we have 100 sets of random weights for a neural network, and evaluate the neural network with each set of weights to see how well it performs a certain task. After doing this, we keep only the best 20 set of weights. We reshape each set of weights into one-dimensional arrays. Then, we can populate the remaining 80 set of weights by randomly choosing from the 20 that we kept, and applying a simple ***crossover and mutation*** operation to form a new set of weights.



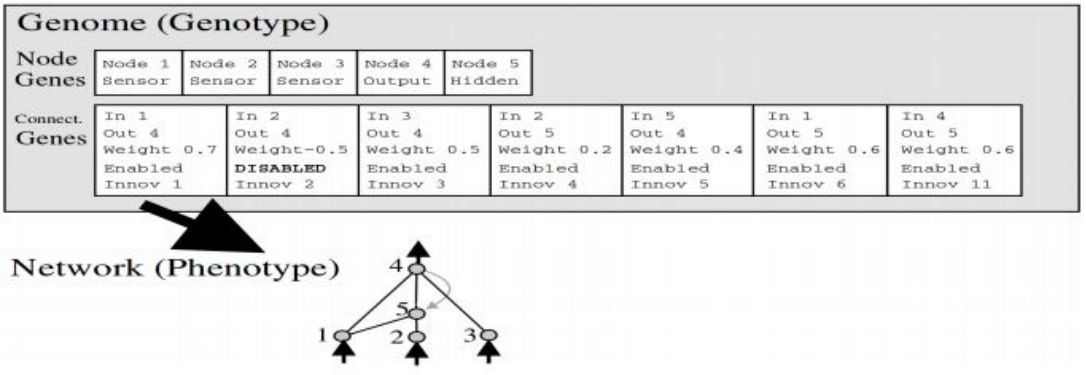
The set of the new 80 weights will be some mutated combination of the top 20, and once we have a full set of 100 weights again, we can repeat the task of evaluating the neural network with each set of weights again and repeat the evolution process until we obtain a set of weights that satisfy our needs.Reward function in our case is a combination of sensor weights from an optimal policy.Hence Reward function can defined by iteratively designing and evaluating the Neural Network , with each iteration showing better results than the previous ones.

C. NEURO EVOLUTION OF AUGMENTED TOPOLOGIES BASED INVERSE REINFORCEMENT LEARNING

NEURO EVOLUTION OF AUGMENTED TOPOLOGIES(NEAT) evolves neural networks using Genetic Algorithm, guided by a fitness function. Each member of the population corresponds to a genotype or genome and a phenotype (the actual neural network). A genome consists of node genes (listing input, output and hidden nodes) and connection genes (listing connections between nodes and associated weights).

NEAT begins with relatively less complex neural networks and then increases complexity based on a fitness requirement. Specifically, the initial network is perceptron-like and only comprises of input and output neurons. The gene is evolved by either addition of a neuron into a connection path or by creating a connection between existing neurons.

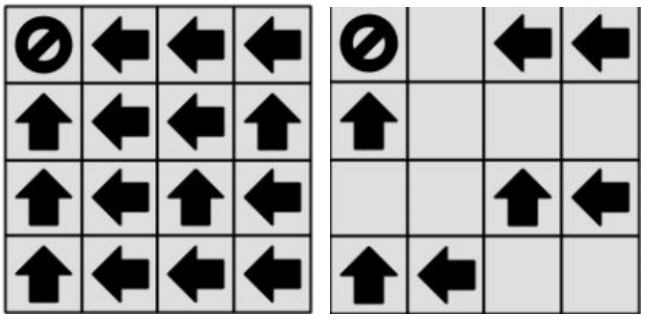
The result of *NEAT-IRL* is a neural network which can produce state values based on state features. Neural networks represented by a genome population in NEAT are considered to use state features as input and produce state value as output instead of generating a function which produces state reward.State values are used to generate a corresponding policy. To incorporate learning by demonstration, this policy is matched with the demonstration and then the neural network is evolved.



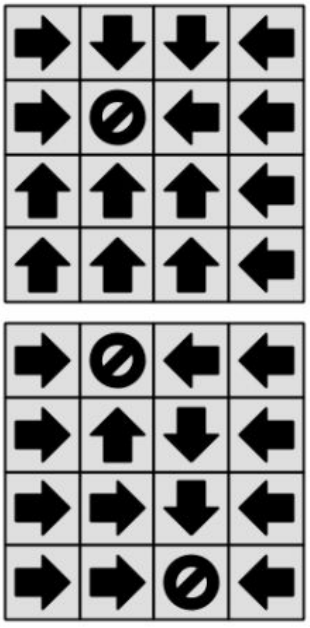
The figure shows Genotype to phenotype mapping . A genotype corresponds to the encoding of the nodes and connections of a neural network. These are represented as genes. The phenotype is the actual neural network generated based on its genotype.

D. IMPLEMENTATION

The environment used for this work is a grid world Markov Decision Process (MDP). In this environment, an agent has 5 possible actions: move up, move down, move left, move right, do nothing.



The above left figure shows the optimal policy to reach the goal given arbitrary position. The right figure shows 4 demonstrations with length 2.



The above figures show the solutions of NEAT based IRL algorithm.It can be seen from the figures the algorithm performs well even with increased complexity (2 goals in the bottom figure)

OBSERVATIONS AND COMMENTS :

1. The algorithm discussed above outperforms other algorithms in low state space environments .
2. As compared to the other discussed algorithms NEAT IRL produces optimal results without converging into local optimums.
3. The computation time is linear and hence faster .(Computation time increases with increase in the state state) .
4. The algorithm performs well even for non-deterministic MDPs

CONCLUSION AND FUTURE SCOPE

This project was mainly focused on understanding how the existing algorithms work in controlled environments to grasp their application to real world uncontrolled environments.
Our research and subsequent implementation reached out to five algorithms that are currently state of the art. In all our approaches we assumed access to demonstrations by an expert that is trying to maximize a reward function which in turn builds an optimal policy close to that of the expert.
In behavioral cloning/imitation learning we carried out tests on a completely controlled environment and learnt that the algorithm was not flexible enough and was quite trivial for us to adopt it for the task of self driving. The concept of apprenticeship learning was deployed in a maze environment wherein we found it to encompass the much needed flexibility, being able to grasp the motives of the expert to a higher degree. However scaling it to higher dimensional, continuous spaces is still a disadvantage which it heavily suffers from. The concept of maximum entropy deep inverse reinforcement learning is where the disadvantage of scaling it to higher dimensional problems levels out with computational complexity. Our tests on the cart-pole environment proved to be successful and convergence was achieved in a fewer number of iterations compared to the algorithms aforementioned. A significant boost of deep max for highly complex environments is also presented in an algorithm called Guided-cost learning, which we will consider in the future.
Finally the concept of Neuro-evolution presents the NEAT (Neuro Evolution of Augmented Topologies based inverse reinforcement learning) which takes an evolutionary approach incorporating permutations between several combinations of actions to enable the agent to uncover the underlying optimal behavior. Several tests presented in literature have proved to bolster our choice to incorporate this technique for navigation.
Our results have motivated us to stick to the concept of Deep max entropy, NEAT and Guided cost learning. Our future scope would be to combine these three in a robust manner so as to be able to carry out exhaustive tests in a custom environment on Gazebo (ROS) and to deploy the same as hardware in a model environment commensurate of a real world town.

Additional input/state features like Depth image from the camera, Voice recordings from the driver which can uniquely distinguish states will be used in the next phase of the project where high dimensional inputs obtained from the complex world will be used for the algorithm to achieve a state of autonomous driving.

REFERENCES

[1] http://cs231n.github.io/convolutional-networks/#overview

[2] https://arxiv.org/abs/1604.07316

[3]https://drive.google.com/open?id=1aa1IW3HAxEBmgWDVS1Ty_MYkm2jg4zQ2

[4]Net-Scale Technologies, Inc. Autonomous off-road vehicle control using end-to-end learning, July 2004. Final technical report.

[5]Dean A. Pomerleau. ALVINN, an autonomous land vehicle in a neural network. Technical report, Carnegie Mellon University, 1989

[6]Abbeel, P., and Ng, A. Y. 2004. Apprenticeship learning via inverse reinforcement learning

[7]Ziebart et al, Maximum Entropy Inverse Reinforcement Learning, 2008

[8]End to End Learning for Self-Driving Cars Mariusz Bojarski, Davide Del Testa Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal

[9]Budhraja, K.K. and Oates, T., 2017, June. Neuroevolution-based inverse reinforcement learning. In Evolutionary Computation (CEC), 2017 IEEE Congress on (pp. 67-76). IEEE.