Inverse Reinforcement Learning for Human Centered Trajectory Projection for a Self Driving Car

ANIRUDH.G.J 1BM15EE006
BHARATH.Y.P 1BM15EE013
LAKSHWIN SHREESHA 1BM15EE026
SIDDHARTH.KK 1BM15EE052

PROJECT GUIDE : Dr. P MEENA

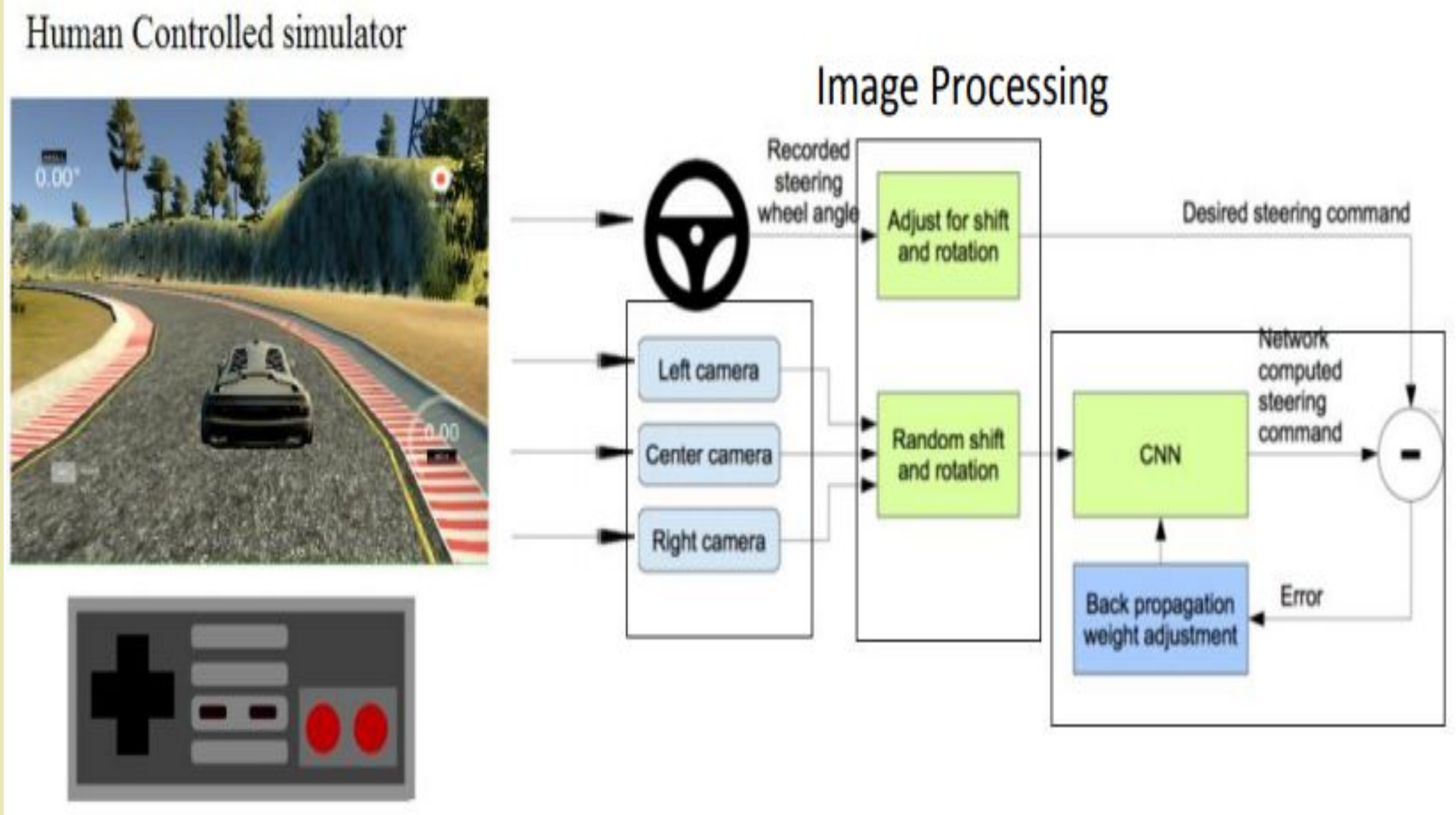
ABSTRACT

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.

1. END TO END LEARNING



Human Controlled simulator
Image Processing
Recorded steering wheel angle
Adjust for shift and rotation
Desired steering command
Left camera
Center camera
Right camera
Random shift and rotation
CNN
Network computed steering command
Back propagation weight adjustment
Error

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

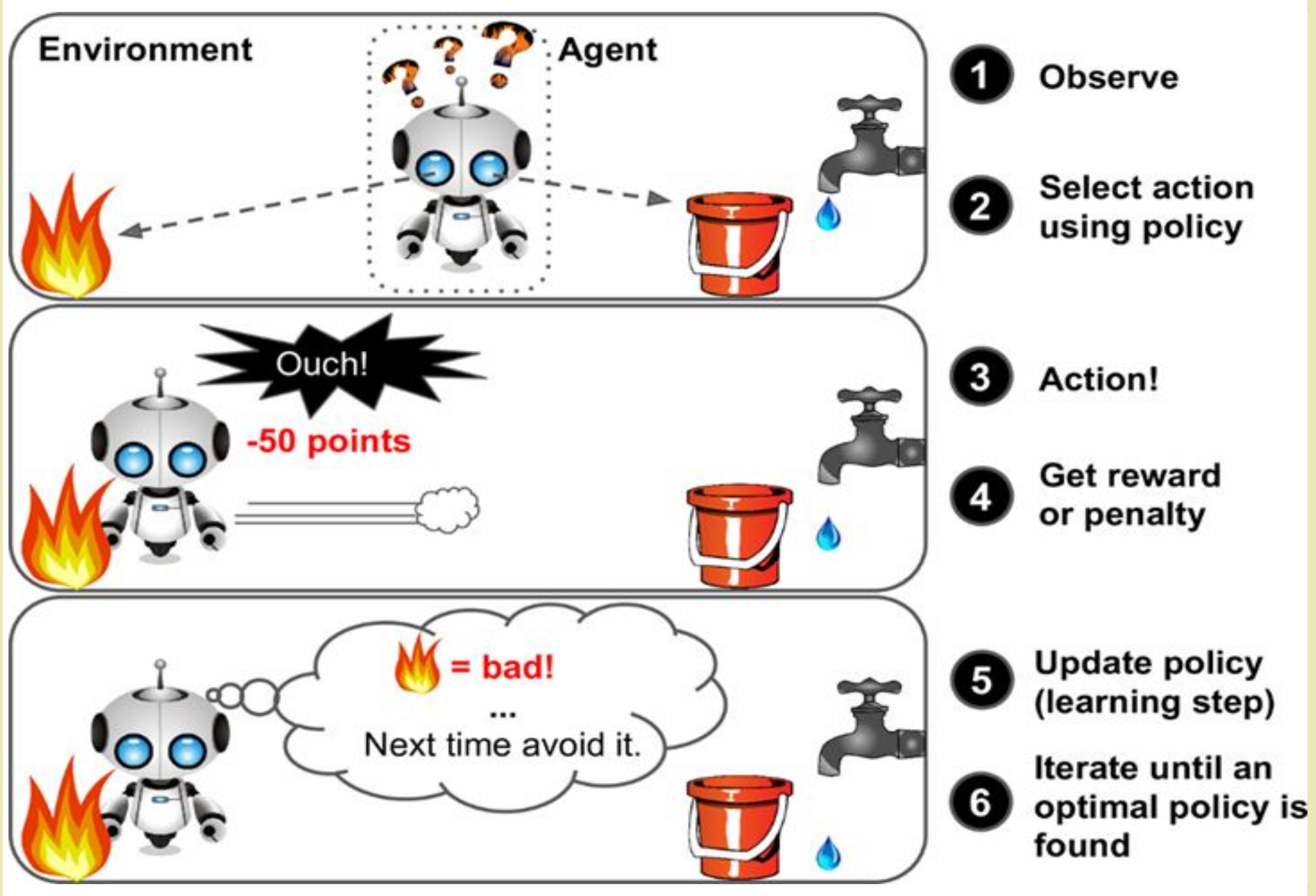
Observation Vector (4x1)	Min	Max
Cart Position	-2.4	2.4
Cart Velocity	-Inf	Inf
Pole Angle	~ -41.8°	~ 41.8°
Pole Velocity At Tip	-Inf	Inf

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

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.

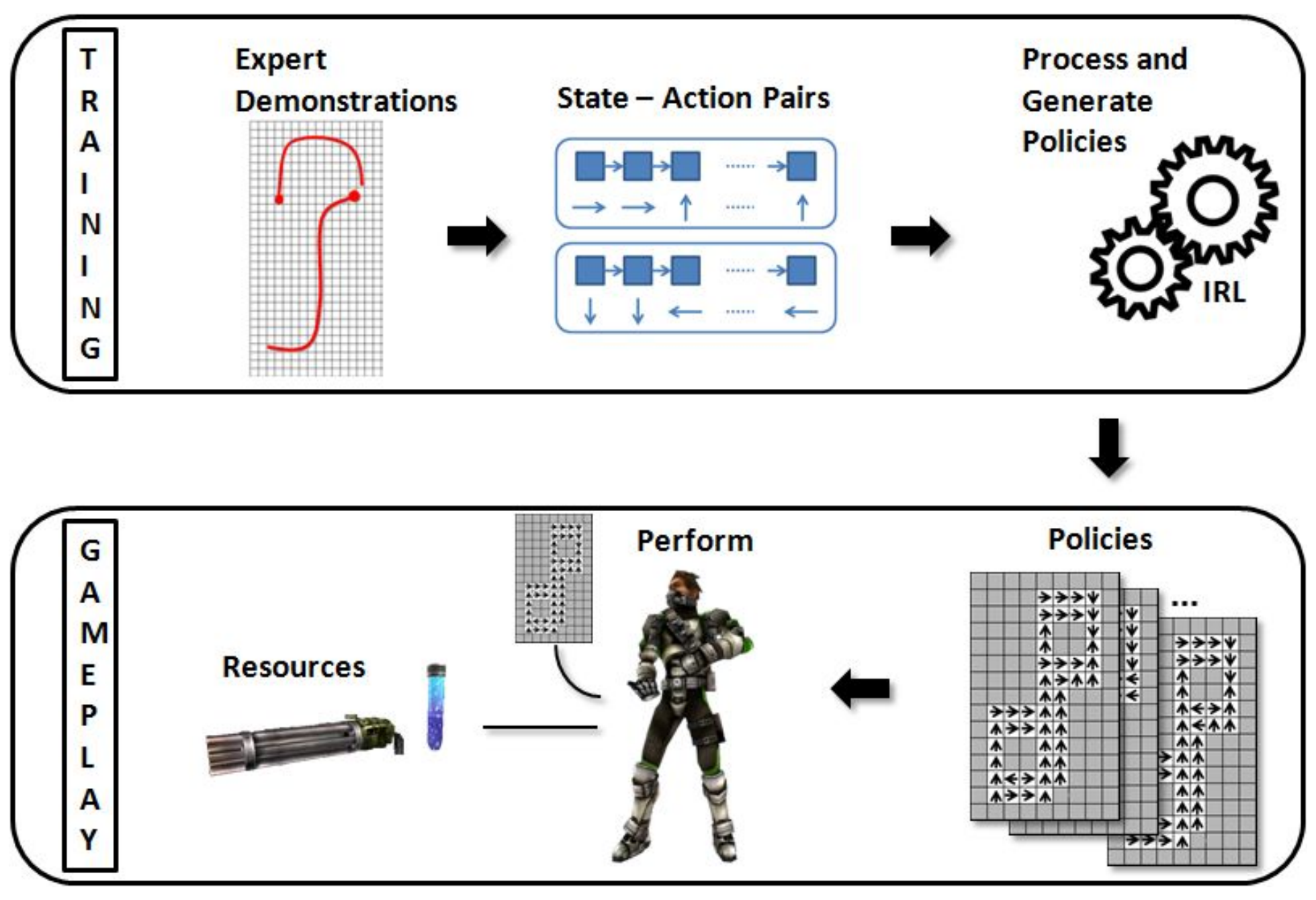
OVERVIEW

REINFORCEMENT LEARNING

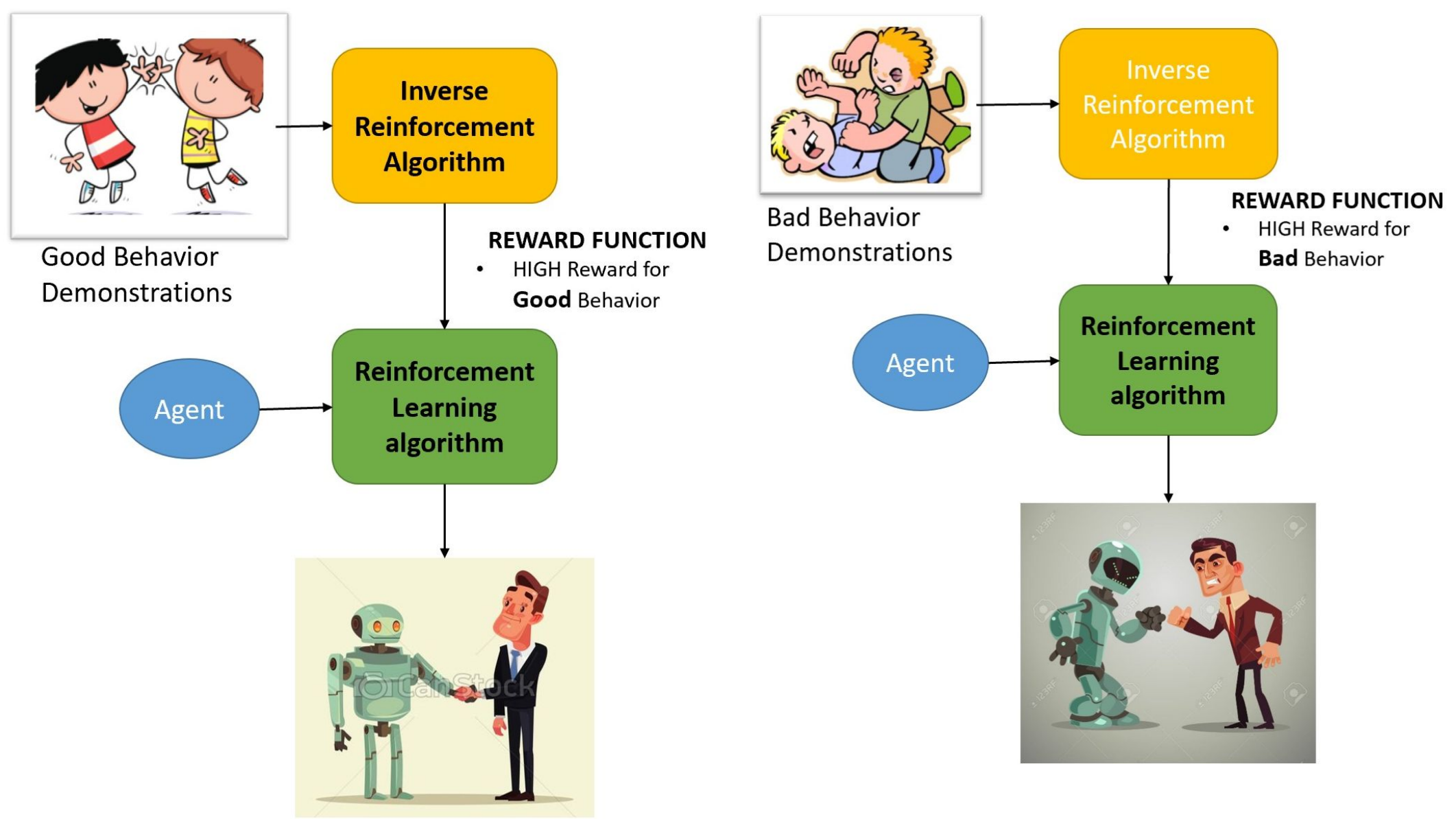


Environment
Agent
Ouch!
-50 points
= bad!
...
Next time avoid it.
1 Observe
2 Select action using policy
3 Action!
4 Get reward or penalty
5 Update policy (learning step)
6 Iterate until an optimal policy is found

OVERALL BLOCK DIAGRAM

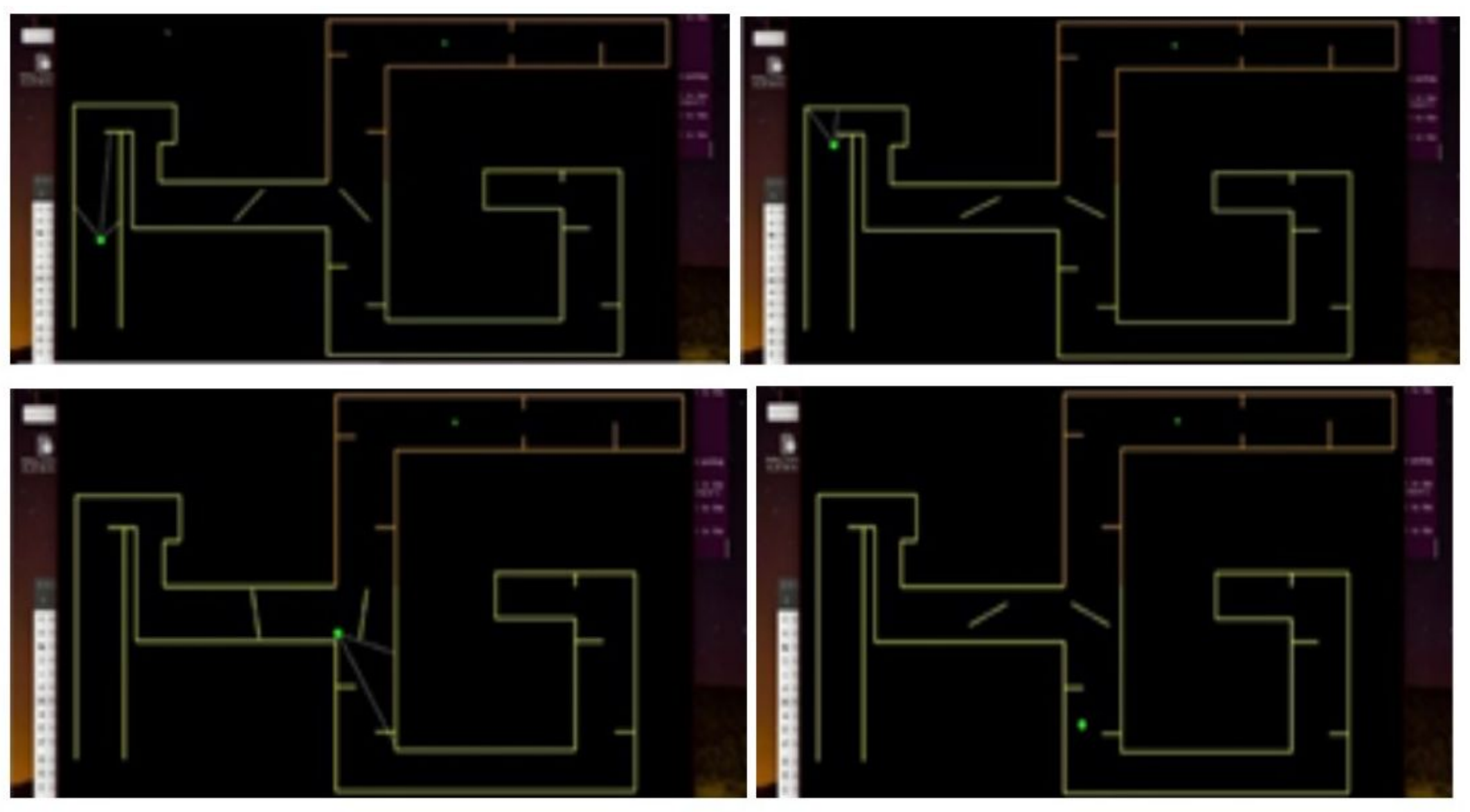


INVERSE REINFORCEMENT LEARNING



Good Behavior Demonstrations
Inverse Reinforcement Algorithm
REWARD FUNCTION
• HIGH Reward for Good Behavior
Agent
Reinforcement Learning algorithm
Bad Behavior Demonstrations
Inverse Reinforcement Algorithm
REWARD FUNCTION
• HIGH Reward for Bad Behavior
Agent
Reinforcement Learning algorithm

3. APPRENTICESHIP LEARNING VIA INVERSE REINFORCEMENT LEARNING



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.

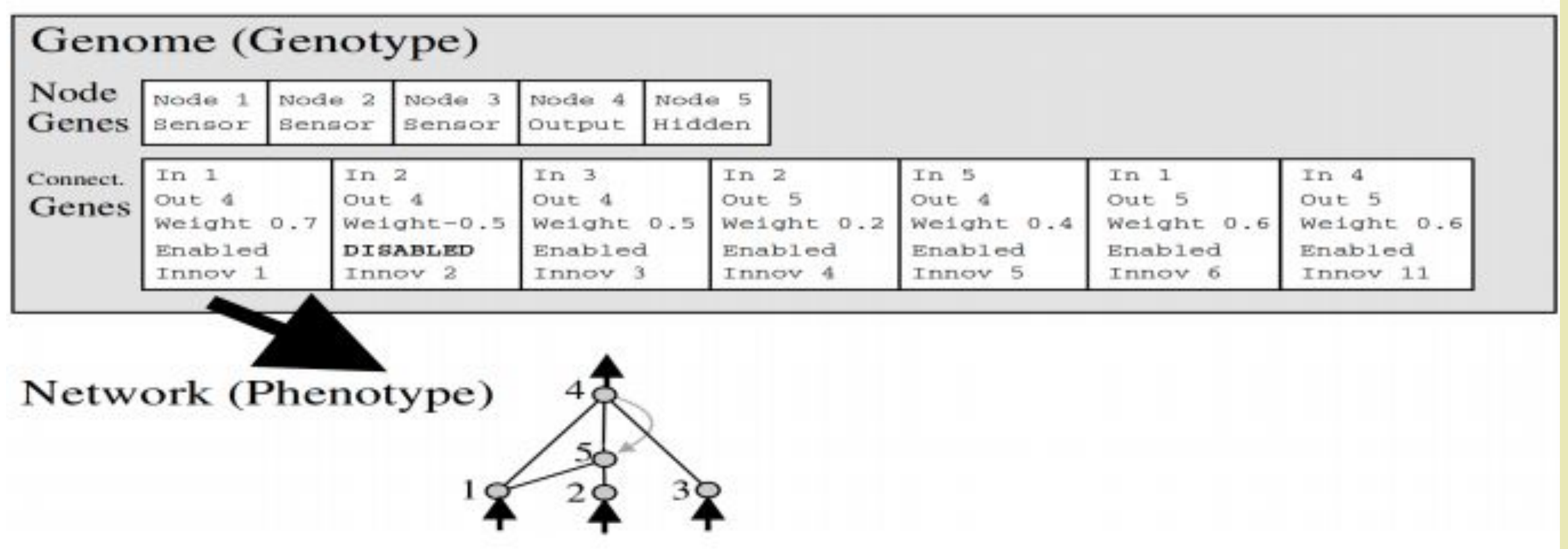
PROBLEM DEFINITION

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 (π ∗), 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.

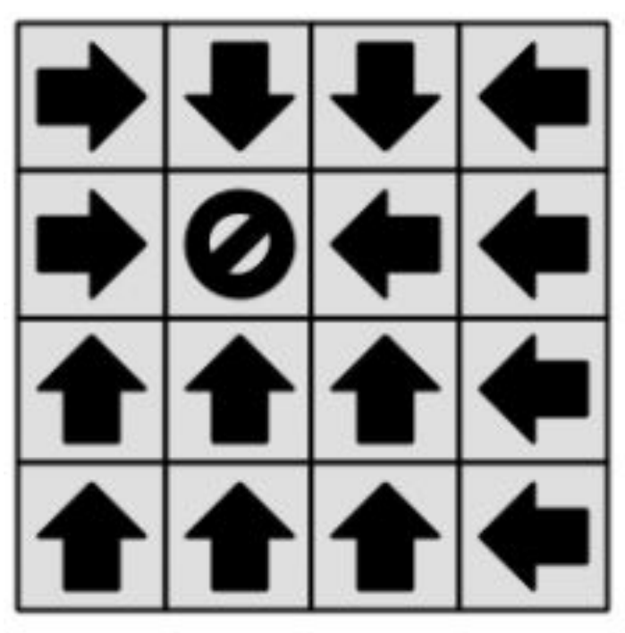
The goal of IRL is to learn a reward function for each state based on parts of a given policy (a demonstration). In a broader sense, the goal is to be able to generate a policy over a state space (S), which is correlated to what has been demonstrated. For the purpose of this work, the demonstrations 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).

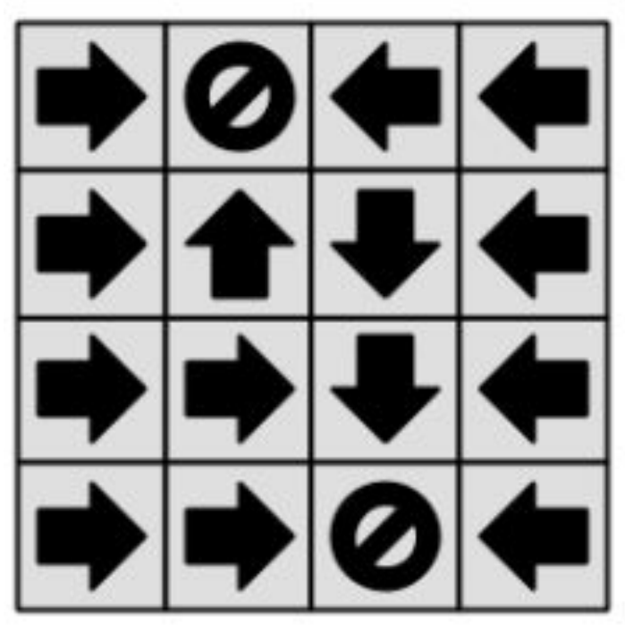
One method to generate a policy is to generate state values for each state based on state features. It is assumed that a weighted combination of state features can provide a quantitative evaluation of a state.The first problem, then, is to learn a mapping from state features to state values that produces 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).

4. NEURO EVOLUTION BASED INVERSE REINFORCEMENT LEARNING



Genome (Genotype)
Node Genes
Node 1 Sensor
Node 2 Sensor
Node 3 Sensor
Node 4 Output
Node 5 Hidden
Connect. Genes
In 1 Out 4 Weight 0.7 Enabled Innov 1
In 2 Out 4 Weight−0.5 DISABLED Innov 2
In 3 Out 4 Weight 0.5 Enabled Innov 3
In 2 Out 5 Weight 0.2 Enabled Innov 4
In 5 Out 4 Weight 0.4 Enabled Innov 5
In 1 Out 5 Weight 0.6 Enabled Innov 6
In 4 Out 5 Weight 0.6 Enabled Innov 11
Network (Phenotype)
4
5
1
2
3





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