Deep Reinforcement Learning based Vehicle Navigation amongst pedestrians using a Grid-based state representation

Niranjan Deshpande, Anne Spalanzani

Deep Reinforcement Learning based Vehicle Navigation amongst pedestrians using a Grid-based state representation*

Niranjan Deshpande[1] and Anne Spalanzani[1]

***Abstract*— Autonomous navigation in structured urban environments amongst pedestrians is a challenging and less explored problem. In this work, we propose to use a deep reinforcement learning based method to solve this problem of navigation. A Deep Q-Network based agent is trained in a simulator for a typical intersection crossing setup amongst pedestrians. We propose a grid based representation as a state space input to the learning agent. With this grid based representation and our reward function the agent learns a policy capable of driving safely around pedestrians and also follow the traffic rules.**

I. Introduction

The field of autonomous vehicles has been growing vastly since the last few decades due to its ability to reduce road accidents and traffic congestions. One of the important aspects of autonomous vehicles is navigation and traditionally predefined rule-based state machine approaches have been used for the same. These approaches require expert domain knowledge and it would be time consuming and error prone to build a system for all possible situations. Hence, for complex highway and urban environments such techniques will not be sufficient. These environments would consist of other vehicles and pedestrians and the autonomous vehicle would be required to behave more intelligently and humanly. A method which could learn its desired behavior from its own experiences feels more intuitive and similar to how humans learn their tasks.

In the last few years, reinforcement learning (RL) combined with deep learning has been able to achieve remarkable success in various areas such as robotics [1], [2], [3], continuous control [4], [5] and video games [6], [7], [8]. For example, [7] build an RL agent which could achieve a super human like performance and beat the world champion in the game of Go. This has motivated researches to explore the use of deep reinforcement learning for navigation of autonomous vehicles, for example, in [10], [11], [12], [13], [14].

However, much of the research has focused on lane changing and speed control behavior for highway environments in the presence of other vehicles. To our knowledge approaches considering a typical structured urban environment in the presence of pedestrians is missing.

In this work, we explore the use of Deep Q-Network (DQN)[4] based decision making for autonomous vehicle navigation. We consider a typical intersection crossing scenario in urban environments with pedestrian crossing. Towards this goal, we propose a grid-based representation of the environment as a state space for the DQN learning agent.

[1]Univ. Grenoble Alpes, Inria, 38000 Grenoble, France FirstName.LastName@inria.fr

II. Related Work

Research in navigation for autonomous vehicles has a long history and dates back to two-three decades. A detailed survey about the same is discussed in [15]. However, in this section we mention more recent learning based approaches which are more relevant to this work. We further classify these approaches into two categories in relevance to the problem dealt in this work.

A. Navigation for highways

One of the first approaches exploring DQN for navigation was in [9], where a simulated car was successfully trained for turning operations in a racing game setup. In [16], an agent was trained for autonomous car driving using raw sensor images as inputs. In [12] a deep RL framework is proposed where an agent is trained to learn driving, given environmental conditions. The novelty of this approach is that instead of raw sensor images, the DQN agent is given environmental states. The authors of [13] proposed a Deep Q-Network based approach for training agents for speed and lane change decisions in a highway driving case. The input to the neural network is a vector of 27 elements, representing: ego vehicles speed, available lanes and states of 8 surrounding vehicles. Two different agents were trained in this study, one which controlled only the lane change decision while other which controlled both, speed and lane change decisions. Also, in [17] a Deep Q-Network is used to learn maneuver decisions for highway scenarios. It introduces the concept of a compact semantic state representation which is passed as input to the network. This representation stores relations between 8 other surrounding entities of the environment with respect to the ego vehicle. The problem of intersection handling in case of partial knowledge is treated in [14] by using Deep Q-Network. It also shows that deep reinforcement learning agents are capable of learning active sensing behaviors for safety navigating in the case of occlusions.

B. Navigation amongst pedestrians

Recently some approaches have proposed the use of Partially Observable Markov Decision Process (POMDP) for autonomous driving amongst pedestrians. For instance, in [18] pedestrian intention is modeled by assigning a goal location to each pedestrian and its behavior is conditioned on the intention as a hidden variable. It uses an A* planner for global path planning and a POMDP planner for velocity control. In [19], POMDP is used for intention-aware motion planning. It uses a version of potential field method to model pedestrian behavior, wherein a pedestrian is attracted

towards a goal and repelled by the robot vehicle. In [20] another POMDP approach is proposed which models both pedestrian intentions and interactions by using a pedestrian motion model predictor called PORCA (Pedestrian Optimal Reciprocal Collision Avoidance). However, online POMDP planners are computationally complex as they require large resources for computation and also they consume much time. This limits their use for real autonomous driving platforms.

As mentioned above, in the recent years, several approaches have been proposed using deep reinforcement learning for autonomous navigation. However, these approaches [9], [12], [13], [17], [16] usually consider highway setup with a focus on lane change behavior and speed control and do not consider the presence of pedestrians. Also, the approaches in [19], [18], [20] dealing with the problem of navigation around pedestrians usually consider an unstructured shared environment. In such environments, the agents share a common space with the pedestrians and the aim is to reach the goal while avoiding pedestrians by navigating around them.

However, in this work, we consider a structured urban setup where the agent and pedestrians co-exist in the environment with separate regions assigned and usually interact at predefined areas, for example, at crossings. To our knowledge, this is the first work exploring the use of deep reinforcement learning for navigation amongst pedestrians in structured urban environment. More specifically, we use Deep Q-Network algorithm and consider a typical intersection crossing scenario. We propose a grid based approach for representing pedestrian information as a state space input to the learning agent. This representation is capable of handling crowded situations as well, unlike, for example [13], [17], which considers vehicles only in 8 directions around the ego-vehicle.

III. Technical Background

A. Reinforcement Learning

In a typical reinforcement learning framework, an agent at time step t, gets the state of its environment s_t and takes an action a_t following a policy $\pi(a|s)$, that is, the probability of taking an action a, given a state s. Taking an action takes the agent to the next state s_{t+1} and it receives a numerical reward r_t for taking an action. A reinforcement learning problem is often formulated as a Markov Decision Process (MPD)[21] (S, A, P, R), where S is the set of states, A being the set of actions, P representing state transition probability function and R is the reward function. MDP follows the Markov property that the next state only depends on the current state-action pair and is independent of all the previous states and actions. The goal of reinforcement learning is to learn an optimal policy π^* that maximizes the discounted future reward. defined as follows:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{1}$$

where $\gamma \in [0, 1]$ is a discount factor for deciding the importance of future rewards.

B. Deep Q Network (DQN)

Q-learning is one of the most used algorithms of reinforcement learning where the agent tries to learn an optimal state-action value function. This function $Q^\pi(s, a)$ defines the expected reward when being in a state s and taking an action a following a policy π, mathematically expressed as:

$$Q^\pi = E[R_t|s_t = s, a_t = a]$$

Q-learning tries to learn an optimal state-value function $Q^*(s, a)$. This optimal function follows the Bellman equation:

$$Q^*(s, a) = E[r + \gamma \underset{a_{t+1}}{max} Q^*(s_{t+1}, a_{t+1})|s, a]$$

In DQN, a neural network is used as function approximator for the state-action function, that is, $Q^* \approx Q(s, a, \theta)$, where θ are the weights of the network. The network is trained by adjusting its parameters θ to minimize the error between the expected reward and the state-action value predicted by the network. The agents experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ are stored in an experience replay memory. Then at every time step t, a mini batch of size M of experiences are randomly selected from the memory and passed to the network for updating the parameters θ using the loss function:

$$L(\theta) = E_M[(r + \gamma \underset{a_{t+1}}{max} Q(s_{t+1}, a_{t+1}; \theta^-) - Q(s_t, a_t; \theta))^2]$$

In the above equation, θ^- indicates the parameters of a target network. In DQN, a target network is used to make the learning process more stable and its parameters are updated only periodically using the latest parameters θ of actual network.

IV. Methodology

A. MDP Formulation

1) State Space:

The information about the ego-vehicle and pedestrians is represented in the form of a 3-D tensor as shown in Figure 1 . It consists of multiple 2-D grids with each layer of the grid storing a feature about the ego vehicle and surrounding pedestrians. Specifically, following features are captured in each layer:

- *layer 0: id* of i^{th} entity
- *layer 1:* speed v_i of i^{th} entity w.r.t ego-vehicle
- *layer 2:* heading direction ϕ_i of i^{th} entity w.r.t ego-vehicle
- *layer 3:* region occupied by i^{th} entity.

Speed is expressed in *meters per second (m/s)*, heading direction in *degree* (ranging from *0 to 360 degrees*). For this work, a region could belong to one of the following types: *road, crossing, sidewalk*.

For the ego vehicle, a fixed *id* is assigned to it and stored in the first layer (*layer 0*) of the tensor. Its heading direction is fixed to zero and stored in the next layer (*layer 2*). The region occupancy layer (*layer 3*) gets updated as either *road* or *crossing* based on current location of the vehicle. The speed

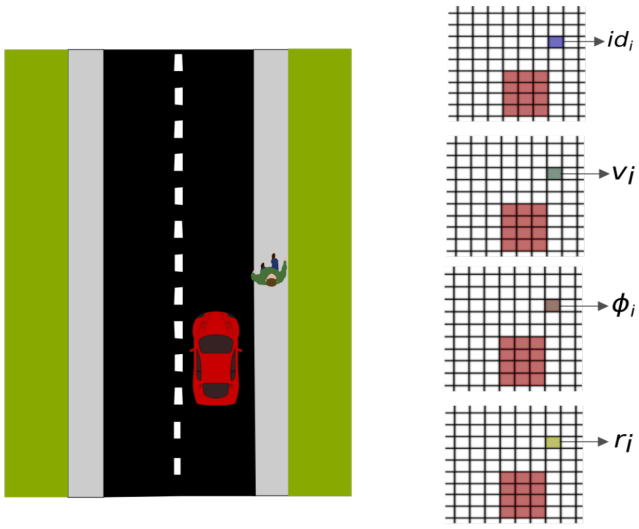


Fig. 1: *A portion of environment shown on left converted to different layers of the grid shown on right. Different colors for pedestrian in different grids represent different information.*

of the ego-vehicle gets updated at every step and accordingly the speed layer (*layer 1*) is updated. For pedestrians, each pedestrian has a unique *id* assigned to it (stored in *layer 0*), followed by its speed relative to the ego vehicle (in *layer 1*), its heading direction relative to ego vehicle (*layer 2*) and finally at the last layer (*layer 3*) the region which it occupies. This information about the region is not yet completely exploited in the current work and will be useful for more complex scenarios of the future.

Each grid is a 70 x 30 matrix corresponding to the range of 60 meters in the front and 10 meters behind respectively and 15 meters on each side of the ego vehicle. This makes each cell of the grid of size 1x1 meters. We assume that vehicles are usually 4-5 meters in length and 2-3 meters in width and a pedestrian has a footprint of 1x1 meters occupying 1 grid.

2) Action Space:

The action space for the agent consists of four discrete actions:

TABLE I: Set of actions

Action	Description
a_0	full brake
a_1	decelerate
a_2	continue
a_3	accelerate

3) Reward Function:

Specifically, for this work the aim of the agent is to learn a policy such that it avoids collision with near by pedestrians and at the same time try to maintain a desired speed when there are no pedestrians around. To encourage the agent for maintaining a high speed, a small positive reward is given at each time step. This reward is normalized over the maximum speed v_{max}. The v_{max} here indicates maximum allowable speed and not the maximum achievable speed by the ego-vehicle. Also, a penalty of -5 is given for exceeding the speed limit or -2 if the vehicle stays at zero speed. This is to encourage the vehicle to keep moving. The rewards for speed are summarized as:

$$r_s = \begin{cases} v_{ego}/v_{max} & \\ -5 & \text{if } v_{ego} > v_{max}, \text{ else } 0 \\ -2 & \text{if } v_{ego} <= 0, \text{ else } 0 \end{cases} \quad (2)$$

where v_{max} is maximum allowable speed and v_{ego} is speed of ego vehicle at the corresponding time step.

No reward is assigned when the agent reaches the goal. This is because we believe autonomous navigation is a continuous process and do not want the agent to influence its learning based on the nearness to the goal.

For avoiding collisions with pedestrians, a penalty r_c of -40 is allotted in case of collision and the episode is terminated. Also, a penalizing reward r_{nc} of -10 is given if the vehicle ends up in a near collision situation, defined as being one vehicle length (5 meters) from the pedestrian. The total reward function is defined as:

$$R = r_s + r_{nc} + r_c$$

V. Experiments

A. Simulation setup

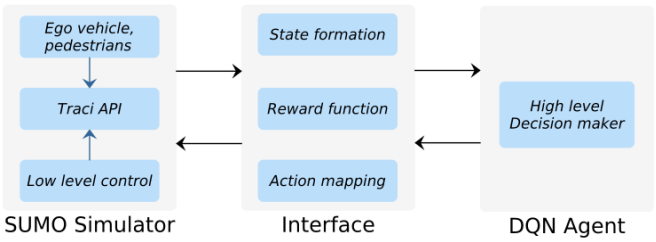


Fig. 2: System Architecture

The implementation details of our simulation system are mentioned in this section. The system consists of three components as shown in Figure 2. The first component is the simulator for which we use the traffic simulator SUMO [22]. The low level control is taken care of by SUMO simulator itself. The ego-vehicle navigates over a predefined route set in the simulator. The intermediate interface component is implemented using the OpenAI Gym toolkit [23]. It is an open source library for developing reinforcement learning environments. The third component is the DQN agent developed with the help of Keras-RL library [24].

Data flow between these three components happens in the following manner: the simulator provides information about the ego-vehicle and pedestrians through a python interface called *Traci*. This information is processed by the interface component and passed (along with associated reward) to the DQN agent. The agent then selects an appropriate discrete

action and returns it to the interface where it is further processed and passed to the simulator. This processed action (speed control command) is then executed by the simulator.

B. Network architecture

For this work we use the architecture presented in [17]. This architecture is inspired from the DQN approach proposed in [4] which takes raw sensor images as inputs and uses a series of convolutional layers in order to extract the low level features from the image.

Since our state space consists of features about the ego-vehicle and nearby pedestrians instead of raw images, we employ a simple fully connected neural network as a function approximator for the DQN agent. It consists of 4 fully connected layers with 512, 512, 256, 64 units in the first, second, third and fourth layer respectively.

C. Network training

For training the agent we use the Deep Q-Network algorithm mentioned in section III-B. For generating training data, the agent runs on an intersection crossing scenario developed in SUMO simulator. The agent is trained for 10,00,000 steps, with each step generating an experience consisting of current observed state, the action selected, the corresponding reward received and the subsequent next state. Each experience is stored in an experience replay memory which can hold upto 100,000 experiences. Learning begins when the replay memory reaches a threshold of 10,000 experiences. A mini-batch of size 32 is selected randomly from this replay memory in order to update the weights of the network. A target network is maintained whose weights are updated after every 10,000 steps. A discount factor of 0.9 is used to discount the future rewards during the training of the DQN agent. The DQN agent uses an optimization method called RMSProp [25] with a learning rate of 0.00025 and decay of 0.95.

During training, to allow exploration at an early stage, an ϵ greedy policy is followed. At each step, an action is selected randomly with a probability of ϵ, else an action with highest Q-value is selected. The ϵ is initialized to 1 and it decreases linearly over 10,00,000 steps until it reaches a minimum value set to 0.1, thus, decreasing the agents exploratory behavior.

The agent is not provided with any information about its nearness to the goal. This is done because the aim is to train the agent driving in urban scenarios of infinite length and do not influence its behavior based on the nearness to the goal.

D. Scenario

Experiments are conducted for an intersection crossing scenario as shown in Figure 3. Two test cases are considered, in the first case, there are no pedestrians across the walkway, and hence, no crossing happens at the intersection. While in the second case, the pedestrians are crossing the intersection.

The ego-vehicle begins from a predefined start position and has a goal to reach. This is considered as an episode of our process. Each time step in the simulator is equivalent to 1 second. Maximum number of steps per episode is set to 300 steps equivalent to 300 seconds. However, the episode ends (before 300 steps) if the goal is reached or if there is a collision and the simulator is restarted to begin the next episode, where the agent is positioned again at the same start position. The purpose of adding a goal is to generate more training data around the intersection for collision situations.

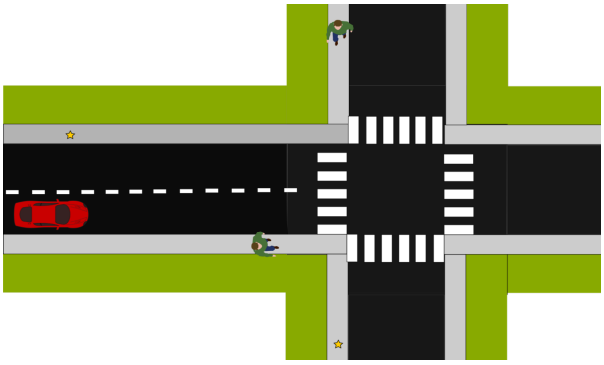


Fig. 3: Intersection crossing scenario

In the simulator, pedestrians move at an average speed of 1.0 m/s, while the maximum achievable speed for the ego vehicle is set to 15 m/s. However, the maximum allowable speed v_{max} is set to 10 m/s. The purpose of keeping achievable speed higher than the allowable is to make sure that the agent also learns traffic rules (speed constriants in our case) along with collision avoidance. The agent is penalized for violating the speed constrains according to the reward function mentioned in IV-A.3.

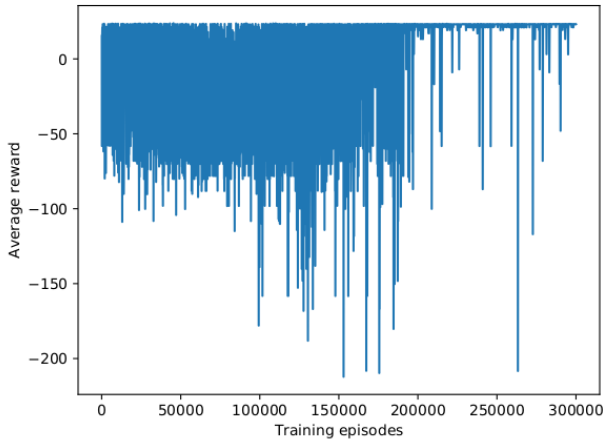
Discrete actions in the action space discussed in section IV-A.2 are converted to corresponding acceleration values and passed to the SUMO simulator. The following acceleration values are used: *full-brake* $(-5m/s^2)$, *decelerate* $(-1m/s^2)$, *continue* $(0m/s^2)$ and *accelerate* $(1m/s^2)$. Trials are run for 10 episodes for each case, and the ego-vehicles average speeds and rewards are recorded. The start position for the ego-vehicle and the pedestrians are varied for every trial.

E. Results

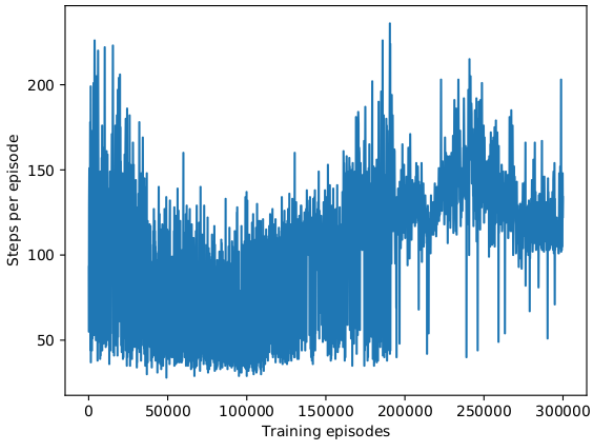
Figure 4a shows the average reward curve per episode while training. Quantitatively, after about 200,000 episodes the driving policy appears to work well as the rewards are increased indicating that it is able to avoid collisions and drive at desired speed. However, the average reward shows a sharp drop at about $260,000^{th}$ episode indicating that there was a collision in this episode. Also, from figure 4b it is seen that after a lot of fluctuations, the number of steps executed per episode increase and stabilize at around $200,000^{th}$ episode mark.

For further evaluation, the DQN agent is tested for 100 episodes for collisions and in all the cases it manages to avoid collision with almost the same average reward return per episode.

The two test cases are compared (for one episode each) based on two parameters, namely:

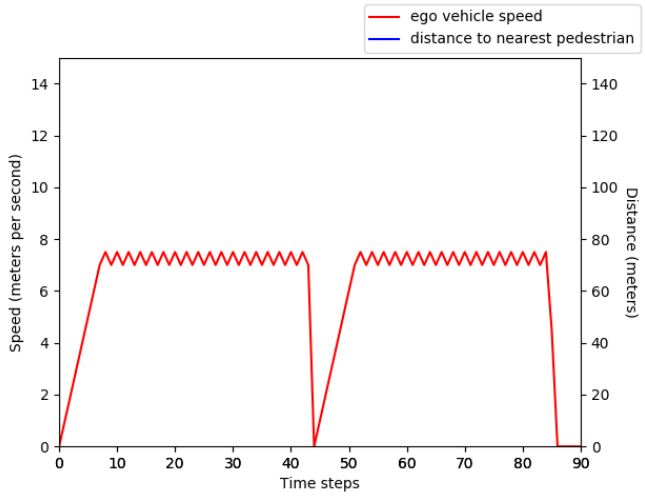


(a) Average reward per episode while training

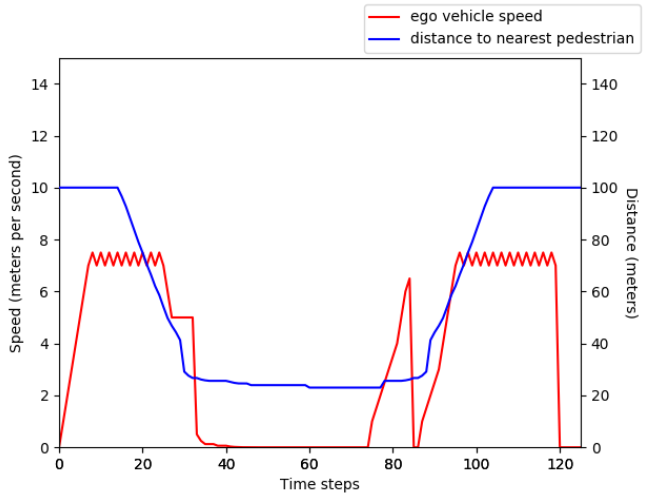


(b) Steps per episode while training

Fig. 4: Agent's training performance



(a) No pedestrian



(b) Pedestrian crossing at the intersection

Fig. 5: Speed vs. Distance comparison

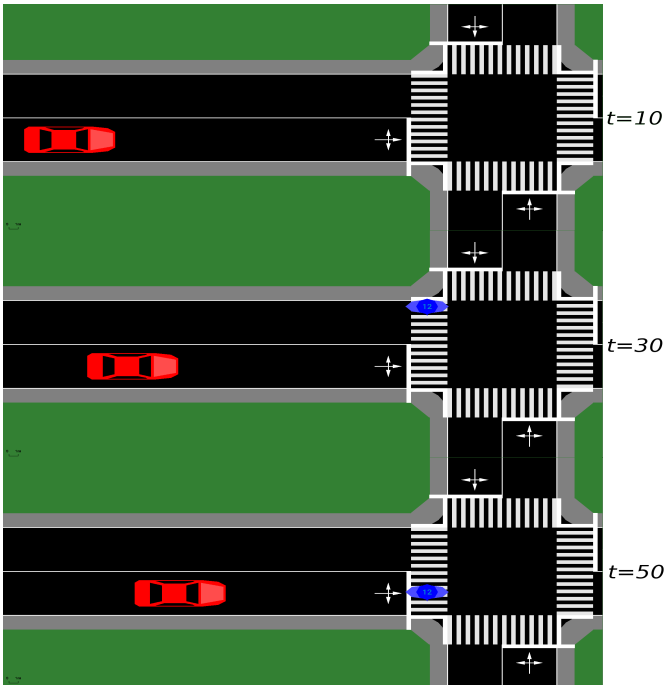
- ***Distance:*** Euclidean distance *(in meters)* to the nearest pedestrian. Distance is initially set to a default value of 100 indicating no nearby pedestrian.
- ***Speed:*** Speed *(in meters per second)* maintained by the ego-vehicle during each step.

As shown in figure 5a, in case of no pedestrian at the intersection crossing, the vehicle tries to maintain a speed of about $7.5m/s$ which is well within the desired speed limit of $v_{max} = 10m/s$. In case of pedestrian crossing the intersection, as shown in figure 5b, as the distance to the nearest pedestrian (crossing the intersection) decreases, the speed of the vehicle drops to zero while the pedestrian is crossing. A sharp speed drop is observed in both the graphs (at around 44^{th} step in 5a and at around 85^{th} step in 5b). This is due to the fact that the agent is making a turn at the intersection around these steps and hence its speed slows down. However, this behavior is handled by the simulator itself and not learned by the agent. Figure 6 shows samples of an episode for intersection crossing test case. The Q-values of the corresponding actions are recorded at every time step. However, for the purpose of visualization Q-values of some of the steps are shown (in figure 6b). Initially the agent starts with high estimated Q-value for the action *accelerate*. However, as a pedestrian starts approaching the crossing, the Q-values for the actions *full-brake* and *decelerate* start increasing. After the pedestrian passes, the agent resumes its driving further indicated by a high Q-value for *accelerate* action at $t = 80$.

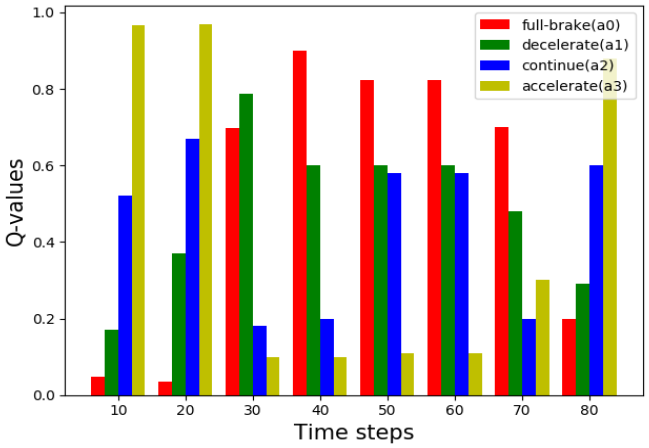
VI. Conclusion And Future Work

In this work, we deal with the problem of autonomous navigation decision making in an urban environment setup in presence of pedestrians. We consider a typical intersection crossing scenario and train a Deep Q-learning based agent. We propose a grid based representation as a state space for the agent. Preliminary tests are conducted and the results show that agent learns the desired behaviour.

For future work, the first step will be to incorporate more



(a)



(b)

Fig. 6

Fig. 7: Simulation samples and q-values

complex scenarios and train the agent for the same. We also plan to explore using a deeper neural network architecture for this by incorporating convolutional layers. We also plan to extend our research towards pedestrian intentions detection and incorporating this information explicitly in our state space.

Another direction of future work is to extend our approach to also incorporate other vehicles along with pedestrians for urban environments.

ACKNOWLEDGMENT

This work is funded under project CAMPUS (Connected Automated Mobilty Platform for Urban Sustainability) sponsored by Programme d'Investissements d'Avenir (PIA) of french Agence de l'Environnement et de la Maîtrise de l'Énergie (ADEME).

REFERENCES

[1] J. Xin, H. Zhao, D. Liu, and M. Li, "Deep Reinforcement Learning in Mobile Robot Path Planning," no. 16, pp. 7112–7116, 2017.

[2] X. Lei, Z. Zhang, and P. Dong, "Dynamic Path Planning of Unknown Environment Based on Deep Reinforcement Learning," *Journal of Robotics*, pp. 1–10, 2018.

[3] M. P. Deisenroth, "A Survey on Policy Search for Robotics," *Foundations and Trends in Robotics*, vol. 2, pp. 1–142, 2013.

[4] V. Mnih *et al.*, "Human-level control through deep reinforcement learning." *Nature*, vol. 518, pp. 529–533, 2015.

[5] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," arXiv preprint arXiv:1509.02971, 2015.

[6] M. R. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, "Playing Atari with Deep Reinforcement Learning," *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2315–2321, 2016.

[7] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[8] B. J. Min and K. J. Kim, "Learning to play visual doom using model-free episodic control," *IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 223–225, 2017.

[9] A. Ganesh, J. Charalel, and N. Xu, "Deep Reinforcement Learning for Simulated Autonomous Driving," *Stanford Project*, pp. 1–6, 2015.

[10] L. Marina and A. Sandu, "Deep Reinforcement Learning for Autonomous Vehicles - State of the Art," vol. 10, no. 2, arXiv preprint arXiv:1701.08878, 2017.

[11] P. Wang, C. Y. Chan, and A. De La Fortelle, "A Reinforcement Learning Based Approach for Automated Lane Change Maneuvers," *IEEE Intelligent Vehicles Symposium*, no. 4, pp. 1379–1384, 2018.

[12] A. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep Reinforcement Learning framework for Autonomous Driving," *Electronic Imaging*, pp. 70–76, 2017.

[13] C. J. Hoel, K. Wolff, and L. Laine, "Automated Speed and Lane Change Decision Making using Deep Reinforcement Learning," *IEEE Conference on Intelligent Transportation Systems (ITSC)*, pp. 2148–2155, 2018.

[14] D. Isele, R. Rahimi, A. Cosgun, K. Subramanian, and K. Fujimura, "Navigating Occluded Intersections with Autonomous Vehicles using Deep Reinforcement Learning," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2034–2039, 2017.

[15] W. Schwarting, J. Alonso-Mora, and D. Rus, "Planning and decision-making for autonomous vehicles," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, no. 1, pp. 187–210, 2018.

[16] M. Vitelli and A. Nayebi, "Deep Reinforcement Learning Approach to Autonomous Driving," in: arXiv, 2017.

[17] P. Wolf, K. Kurzer, T. Wingert, F. Kuhnt, and J. M. Zöllner, "Adaptive Behavior Generation for Autonomous Driving using Deep Reinforcement Learning with Compact Semantic States," *IEEE Intelligent Vehicles Symposium*, no. 4, pp. 993–1000, 2018.

[18] H. Bai, S. Cai, N. Ye, D. Hsu, and W. S. Lee, "Intention-aware online POMDP planning for autonomous driving in a crowd," *IEEE International Conference on Robotics and Automation (ICRA)*, no. June, pp. 454–460, 2015.

[19] T. Bandyopadhyay, K. S. Won, E. Frazzoli, and D. Hsu, "Intention-Aware Motion Planning," *Proc. of the 10th Workshop on the Algorithmic Foundations of Robotics*, pp. 475–491, 2012.

[20] Y. Luo, P. Cai, A. Bera, D. Hsu, W. S. Lee, and D. Manocha, "PORCA: Modeling and Planning for Autonomous Driving among Many Pedestrians," arXiv preprint arXiv:1805.11833, 2018.

[21] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *MIT press*, 2018.

[22] P. A. Lopez *et al.*, "Microscopic traffic simulation using sumo," 2018.

[23] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," in: arXiv preprint arXiv:1606.01540, 2016.

[24] M. Plappert, "keras-rl," 2016, GitHub repository https://github.com/matthiasplappert/.

[25] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the Gradient by a Running Average of Its Recent Magnitude," *COURSERA: Neural Networks for Machine Learning*, 4, 26-31.