Implementation of Reinforcement Learning Simulated Model on Physical UGV Using Robot Operating System for Continual Learning

Edgar M. Perez, Abhijit Majumdar, Patrick Benavidez and Mo Jamshidi
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
San Antonio, TX 78249
Email: edgarmprzz@gmail.com, abhijit.g.majumdar@gmail.com, patrick.benavidez@utsa.edu, mojamshidi4@gmail.com

***Abstract*—Artificial intelligence (AI) has been an issue in robotics, since AI is based on iterative algorithms. In general, simulations of physical models are used to show the outcome of learning algorithms or show proof of concepts. Since models are generated based on parameter estimations of training data, it is crucial to iterate a significant amount of times in order to model an accurate classification function. Thus, it would take a substantial amount of time for a robot to generate such a function. In this research, an implementation of reinforced learning will be applied on a unmanned ground vehicle (UGV) learning simulated model that will be translated into a physical UGV using Robot Operating System (ROS) to test performance of the given model.**

***Index Terms*—Unmanned ground Vehicle (UGV), Reinforcement Learning, Robot Operating System (ROS), Continual Learning,Deep Q-Network (DQN)**

I. Introduction

In the recent years, there has been a lot of development in design and construction of driverless vehicles. Although, most autonomous research is focused on UGV which are differential drive, industry application demands the implementation of autonomy for Ackerman-steered vehicles. There are several commercially available UGVs in the market with stable and efficient controllers, often pre-programmed to perform certain tasks the same industrial way.

Over the past few years, deep learning has contributed an increase of computational capability for machine learning. One application of deep learning reinforcement learning (RL), which focuses on learning optimal policies for sequential decision making by optimizing an accumulation of future rewards [1]- [2]. Q-learning is a very common algorithm for reinforcement learning, but it is known that this method tends to learn unrealistically because it tends to overestimate action values [3]. Nevertheless, Q-learning has a powerful framework for controlling dynamic systems such as robotic systems.

Modeling for systems simulations, in this case dynamic systems, can be convenient because it can be very inexpensive, especially if the purpose is to test learning algorithms. Further, computers can simulate a dynamic system much faster than a real-world dynamic system [4]. This expedites the learning process for a dynamic system; however, for most cases, simulations are prone to work under any circumstances.

Robot Operating System (ROS) plays a major role in expediting the learning process of the physical system. Essentially, ROS behaves as a catalyst to train the data on a powerful computer to generate a new model for the physical system. ROS is mainly used to ease communication between two or more different, independent systems that compile and drive softwares to accelerate development and research in robotics.

In this research, a combination of a real dynamic system and virtual dynamic system is implemented.This is done by, using virtual simulation of UGV to perform initial learning, and then transferring the learned model onto a physical system whose characteristics are similar to the one in the virtual environment, e.g., *bicycle* model of a car. Once the model is generated and transferred onto the UGV using ROS, an observation and comparison can be determined based on virtual and physical results. Since physical system contains same properties as virtual agent, further collection of data can be obtained in the physical environment to continue training and to generate a new model. Based on these results, an inference can be made for similar research in the field of mobile robotics. For example, when controlling a complex system such as non-holonomic car model, reinforcement learning algorithms could achieve controllable results by configuring the sensor placement, the dimension of UGV and the steering limitations to adjust to the new model.

This paper is organized by first describing related work in Section II. In Section III, a background on Multi-agent Virtual Exploration in Deep Q-Learning (MVEDQL) [5] and Deep Q-Network (DQN) [6] reinforcement learning algorithm will be explained for further understanding of the system's architecture. Hardware components are described in Section IV. This is followed by ROS communication network system between the virtual agent model generator and the physical system in Section V. Experimental results will be discussed in

*This work was supported by Grant number FA8750-15-2-0116 from Air Force Research Laboratory and OSD through a contract with North Carolina Agricultural and Technical State University.

978-1-7281-0457-7/2019/$31.00

Section VI, followed by conclusions and future developments in Section VII.

II. Related Work

There are examples in the literature showing a learning *Ackermann* virtual agent that simulates the performance of a car approximation by the *bicycle* model to evaluate its performance after using reinforcement learning [4]. Existing simulations often focus on enhancing graphics of the environment rather than using the computer's resources for training the model. This has being done in non-*Ackermann* UGV applications as well, for example, Ramezani and Lee applied Deep Reinforcement learning (DRL) on the Kobuki Turtlebot 2 [7]. They found that training a model with no initialization of Deep Neural Network (DNN) [8] weights lead them into a battery problem. After analyzing a few solution to this problem; they concluded that it was best to train the robot using initialized DNN wights generated in a Gazebo simulated environment to reduce the Q-function using stochastic gradient decent, and converge to an optimal policy. There are other known robots that have use and performed under reinforcement learning. For example, OBELIX is a wheeled mobile robot that learned to push boxes using a value function-based approach. In section 5 of this paper [9], an attempt of using the trained model by the simulation turn out to have disappointing results. The authors argue that having a poor simulation could have lead into negative results. This paper is useful since they had to deal with sensor data, which is the main training data for our model. Similarly, another robot known as the autonomous helicopter by Bagnell and Schneider [10], used the incorporation of high-order, non-linear simulated model into their Carnegie Mellon's autonomous R-50 helicopter control computer, to drive their system. Their results were encouraging, and demonstrated that policy search can generate controllers that are applicable to robotics systems.

III. System Architecture

The use of DRL demands the availability of powerful systems to perform computations for the learning process at an accelerated rate. Added to that are methods in DRL where simulations of an environment can help expedite the learning. Considering an idea simulated environment, a model can learn to behave accurately in the environment to achieve a goal set by the user. The benefit of using a simulation is that, with more number of simulated agents, cumulative learning can expedite the learning process, since simulations can be parallelized and executed much faster than real-time scenarios [5].

We consider a system described by earlier research [4], where an Ackermann-steered UGV is designed to learn to reach a destination while avoiding collisions with the environment and the obstacles in it. Sensors on the UGV are placed with different orientations to monitor said obstacles, however, there is no prior knowledge provided to the UGV about the nature of the obstacles. As has been already shown in previous research, use of DRL enables such learning of a model and hence can be used as an end-to-end controller for the UGV to navigate through complex environments to achieve its goal of reaching a destination. Since the object of this research is to extend such learning to a physical system, a small scale UGV is built. The UGV has three distance sensors configured in a similar configuration as is in the simulation, and the UGV has also an Ackermann-steered drive system. A general overview of the system is shown in Figure 1.

We set up a communication interface with the physical UGV using ROS since this allows for scaling the system architecture to include multiple UGVs. The ROS network is used by the system to perform the following tasks:

- Instruct the UGV to accelerate or decelerate
- Instruct the UGV to steer at a particular angle
- Obtain distance sensor readings from UGV
- Obtain position readings from UGV
- Obtain orientation readings from UGV

As stated earlier, use of a powerful computer system is desired to enable faster learning with multiple simulated UGVs in a virtual environment. For this reason, the simulator interacts with the physical UGV using the ROS network described above. The simulator is executed on a Host computer, which hosts the virtual system simulation and also handles communication with the physical UGV.

The process starts by training the system on the simulator only. Multiple simulated UGVs are used to train in an arbitrary environment, all with the objective to reach a goal while avoiding obstacles. The controller process in the simulator is given by the neural network, which in turn, performs Q-Learning. This controller is made generic enough to be adaptable and agnostic of the type of environment it is used with. However it is not independent of some of the physical properties of the UGV. For this reason, actual physical properties like the width and length of the physical UGV are used to define the simulated UGV in the virtual environment.

Once the model is trained using the simulation, it is transferred onto the physical UGV. The physical UGV is then placed in a enclosed area, with arbitrarily placed obstacles. This shows how the DRL model is versatile enough to use its generic learning onto a system which is different from the environment it was trained in. During this process, the UGV transmits its sensor readings along with the position and orientation information asynchronously, which are received by the Host computer over ROS, where the information is synced to establish a valid state of the UGV. This state is pre-processed before feeding into the DRL controller process to generate an action to be taken, given the state of the UGV. The action predicted by the DRL controller is a pair of forward or backward, velocity and steering. These values are sent back to the UGV over ROS to instruct the motors for corresponding motion.

Since there are discrepancies between the real world interactions of the physical UGV in its environment compared to the simulated interactions of the virtual UGV in the simulated environment, the learned model may not suffice for determining the optimal control strategy to achieve its goal. For example, even though the sensors on the simulated UGV are

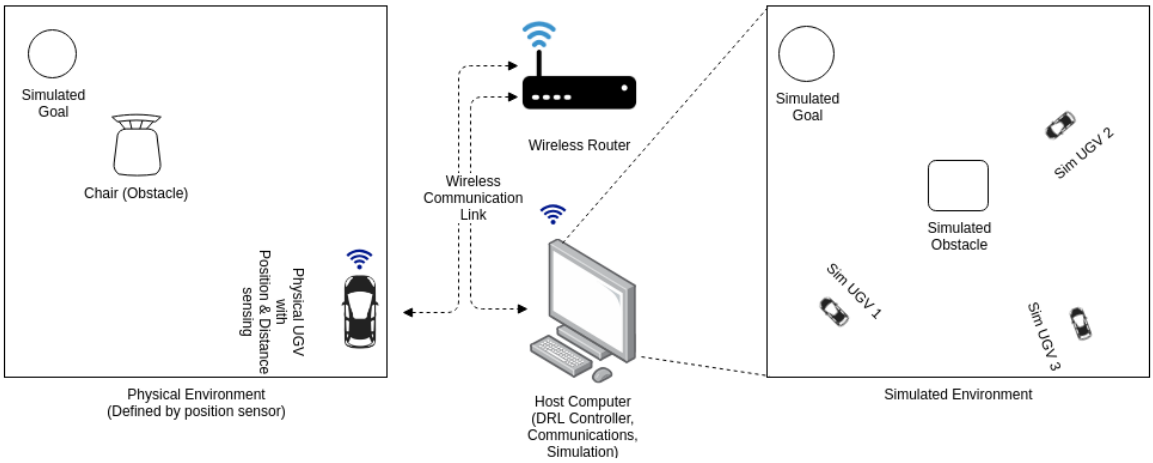

Fig. 1: System Architecture

approximated to have a field of view of a *ray*, physical sensors have a conical shape in the field of being sensed. To capture these discrepancies, while also leveraging the availability of a simulated environment to expedite training, the action taken by the physical UGV along with the previous and next state are recorded into a dataset while the real UGV is run using the learned model. Such a system is used in lieu of developing continuous integration into the system, wherein the system can self-correct and keep learning to potentially learn to navigate its environment better, even when the initial model of the environment might not be ideal (exact to the real-world). The two different approaches used to solve this problem are discussed below.

A. Re-modeling simulated environment

With the newly collected dataset, another model is generated, which aims at replicating the actual dynamics of the UGV with the data. This approach is feasible since inputs and outputs of system are obtained, while the real world system is running, and hence a model can be derived to fit this data.

$$y = f(x) \tag{1}$$

Where,

$y \rightarrow$ UGV dynamic system responses
$x \rightarrow$ Input commands to the UGV system
$f \rightarrow$ A non-linear mapping of the input to the output of the UGV dynamics

There exist different methods of solving for f such as adaptive control methods, fuzzy logic systems and neural network fitting using gradient descent.

B. Inferring Obstacles

While the UGV is traversing through the environment, there would be cases where an obstacle is encountered, which had not been modeled into the simulation while training. Hence, the trained model might not be aware of the optimal way to navigate around the obstacle. This approach involves mapping the object as the UGV navigates around it. Since the sensors provide range sensing data by measuring the obstacle distance from the UGV and since the position of the UGV (hence the sensing) is also known, an inference over time by a closed-loop structure constructs the shape of the new object on the host system. An illustration of this is shown in Figure 2. When a new obstacle is registered, it is added to an array of the type of obstacles/situation that can exist into the simulation. Multiple virtual UGVs are continually trained on these modified obstacles/situations to produce a solution to the problem. An example of where this can be used is where the UGV gets stuck in a corner of a rectangular environment limits. In such cases, extreme actions need to be made by the UGV, which needs to relearn the model to overcome this problem. In addition of such situation and retraining several times in the simulation can potentially provide a solution to circumstance unforeseen by the UGV.

C. Virtual Agent Model

A reinforcement learning virtual simulator was built in Python to enable compatibility of built-in libraries commonly used in simulations such as *math*, *random*, and *numpy*. The model and dynamics of the virtual agent follow those of a rear wheel drive, Ackermann steered vehicle. Similarly, the behavior of the agent is also bicycle model of a rear wheel drive car. The agent's dimensions such as length and width are configurable. Thus, the virtual agent can adopt the shape of real system which promotes the idea of real-world self-driving applications. The following equations are describing the kinematics of the virtual agent:

$$s_t = [x_t, y_t, \omega_t] \tag{2}$$

$$\dot{x_t} = v_t \cos(\omega_t) \tag{3}$$

$$\dot{y_t} = v_t \sin(\omega_t) \tag{4}$$

$$\dot{\omega} = \frac{v_t}{L} \tan(\psi) \tag{5}$$

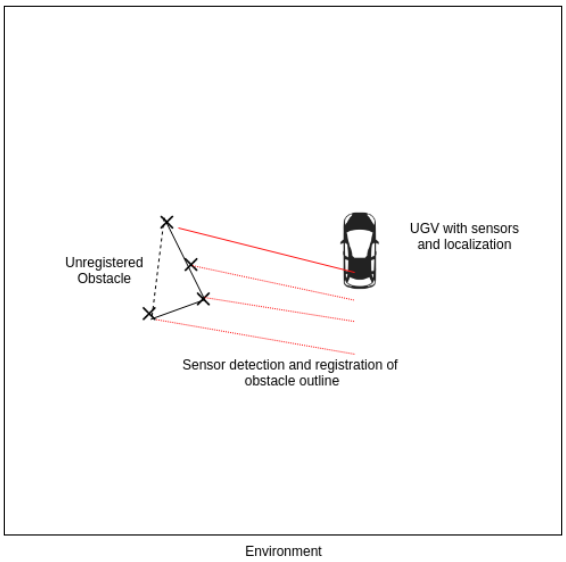

Fig. 2: Example of unknown object registration

where:

s_t: the position and the orientation of the car at time t

x_t, y_t: car position in environment at time t

ω_t: orientation of the car in environment at time t

v_t: the velocity of the car at time t

ψ_t: steering angle of the car at time t

L: length between the front and rear axle of the car

These agents are equipped with laser sensing devices which measure the distance to the nearest obstacle in that direction. The sensors are configurable, and can be changed to read a desired range of distance. Figure 1 illustrates the interaction of the agent with the corresponding sensor readings of the simulated environment.

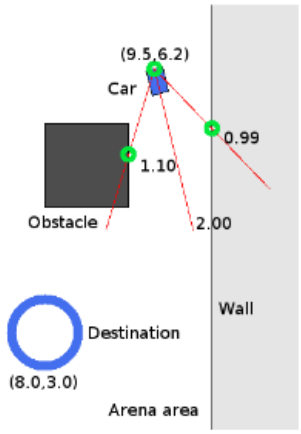

Fig. 3: *Ackermann* UGV on Simulated Environment

IV. Hardware description and dynamic model of self learning UGV

This section is broken into four different sections to specify the properties of the UGV which is currently used as the physical system of this research paper.

A. UGV Specifications and Dimensions

The vehicle selected for the research platform is based on a traditional remote control car with Ackermann- steering. The dimensions of UGV are 10.4"x6.2"x4.7". It is powered with a lithium polymer cell phone battery bank. It can reach a maximum speed of 15 km/h. ground, grass, and sand are all applicable fields where this UGV can navigate.

Fig. 4: Image of Physical UGV

B. Sensor Configuration of UGV

The sensors initially selected for this project are, inertial measurement unit (IMU), time-of-flight (TOF) distance sensor, infrared laser scan localization. The BNO055 IMU is 9-DOF (Degree of Freedom) sensor that provides absolute orientation, angular velocity, acceleration, magnetic field strength, linear acceleration, gravity vectors and temperature for compensation. The time of flight distance sensor uses infrared laser source and a matching sensor to detect the distance of a surface (obstacle) located in front of it.

C. Computing Platform of UGV

The Raspberry Pi 3 is the main processing unit in the vehicle. The main idea is to incorporate all modules into the Raspberry Pi 3. Arduino boards are initially selected to interface with some of the sensors that required either an I2C bus or an UART port for communication. The Kinetic Kame version of the ROS is installed on the Host Processor. ROS provides all the software libraries required to interface with all devices mentioned above.

D. Dynamics Model of UGV

The following nonlinear continuous time equations are based on the kinematic bicycle model shown in Figure 5 [11]:

$$\dot{x} = v\cos(\psi + \beta) \tag{6}$$

$$\dot{y} = v\sin(\psi + \beta) \tag{7}$$

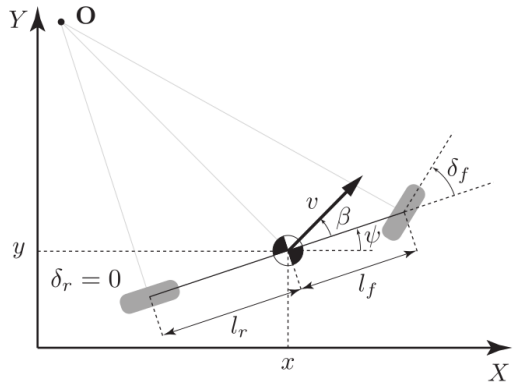

Fig. 5: The kinematic Model of a Bicycle.

$$\dot{\phi} = \frac{v}{l_r} \sin(\beta) \tag{8}$$

$$\dot{v} = a \tag{9}$$

$$\beta = \tan^{-1}\left(\frac{l_r}{l_f + l_r} \tan(\delta_f)\right) \tag{10}$$

where:

x and y : coordinates of the center of mass in the inertial frame (X, Y)
ψ : inertial heading
v : speed of the vehicle's center of mass
l_f : distance from the center of mass of the vehicle to the front axle
l_r : distance from the center of mass of the vehicle to the rear axle
β : angle of the velocity of the center of mass with respect to the longitudinal axis of the car
a : acceleration of the center of mass
δ_f: front steering angle

The system model described above has δ_f and a as control inputs. The following equations describe the dynamic bicycle model.

$$\ddot{x} = \dot{\psi}\dot{y} + a_x \tag{11}$$

$$\ddot{y} = -\dot{\psi}\dot{x} + \frac{2}{m}\left(F_{c,f}\cos\delta_f + F_{c,r}\right) \tag{12}$$

$$\ddot{\psi} = \frac{2}{I_z}\left(l_f F_{c,f} - l_r F_{c,r}\right) \tag{13}$$

$$\dot{X} = \dot{x}\cos\psi - \dot{y}\sin\psi \tag{14}$$

$$\dot{Y} = \dot{x}\sin\psi + \dot{y}\cos\psi \tag{15}$$

where:

$\dot{x}$: longitudinal speed
$\dot{y}$: lateral speed
$\dot{\psi}$: yaw rate
I_z: yaw inertia
m: mass
$F_{c,f}$: lateral tire forces at the front wheel
$F_{c,r}$: lateral tire forces at the rear wheel

The definition of the linear tire model is given by:

$$F_{c,i} = -C_{\alpha_i}\alpha_i \tag{16}$$

where:

$i \in \{f, r\}$
α_i : tire slip angle
C_{α_i} : tire cornering stiffness

V. ROS Network System

ROS is a powerful tool when it comes to communications between autonomous systems. ROS needs a master node in order to operate and manage communication between nodes [12]. Since the physical system's microcmoputer is not powerful enough to process and train the collected data, a transaction of the collected data by the physical model to the Graphics Processing Unit (GPU) must be made in order to process and generate a new model efficiently. To achieve this, the GPU is going to be set at the master on the network and the physical system as the slave.

Figure 6, demonstrates a high level communication network system between the physical system and GPU virtual system.

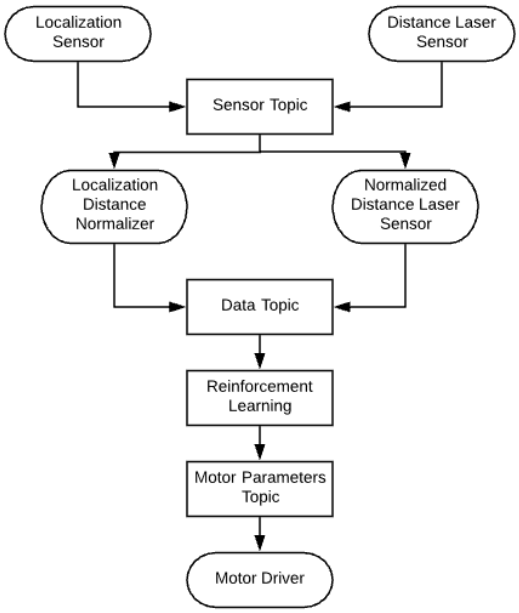

Fig. 6: ROS Network System

Both localization and laser distance sensors are from the physical system. These data are important to drive the UGV, given that the initial model is already imported from virtual simulator. With real-time readings, a new model can be generated and adapt to the new, physical system. The localization distance normalizer is a ratio relative to the size of the map, so the distance magnitude of UGV to the goal never exceeds the size of the map. The node entitled "normalized distance sensor

Time of Flight Sensors	Physical distance from sensor origin	Average distance recorded from sensor	Standard deviation recorded from sensor	Distance error
Left Sensor	61cm	57.3cm	1.47cm	3.7cm
Middle Sensor	54cm	52.7cm	1.73cm	1.3cm
Right Sensor	61cm	56.8cm	1.41cm	4.2cm
Left Sensor	27cm	26.3cm	1.35cm	0.7cm
Middle Sensor	23cm	23.3cm	1.56cm	0.3cm
Right Sensor	27cm	26.1cm	1.23cm	0.9cm

TABLE I: 100 Time of Flight Sensor Samples at Close and Long Range

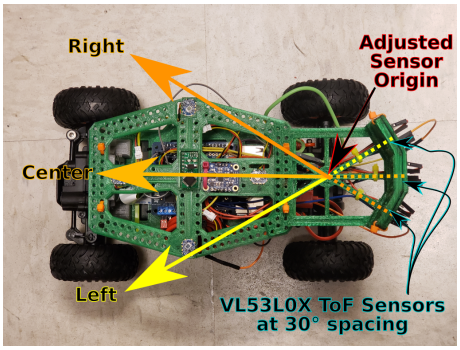

Fig. 7: VL53L0X ToF sensor layout on small Ackermann-steered vehicle. The arrow marked "Center" points towards the front of the vehicle.

laser" subscribes to the raw data published by distance laser sensors to the sensor topic, which then gets translated into meters. This information gets concatenated into a single input vector which then gets processed by reinforcement learning DQN; the output of the DQN decides on one out of six fixed possible outputs, (i.e., go slow and turn right) based on the maximum probabilistic output value of the last layer of the hidden layer.

VI. Results

Accuracy and precision are two very important factors when it comes to robotics. Being able to read reliable data is crucial for optimal results. The time-of-flight (TOF) sensors are currently calibrated to read from the sensor origin as shown in Figure 7. Since the virtual and physical UGV have to be as similar as possible, the sensor origin must be placed at the location shown, see Figure 1. This is a good design because the UGV has lower chance of colliding due to the wide sensing cone shape range that this set up offers. The following results were obtained by positioning the UGV 54cm from a flat obstacle to gather and calculate the following data shown in Table I, then repeating for an obstacle 23 cm away.

VII. Conclusions and Future Work

The developments made in this research will be extended in several ways. The use of continuous learning to keep achieving better performance is essential to adapt to new, unforeseen circumstances. The remodeling of simulated environment and the obstacle inference the process described, is essential to this adaption. Adding more physical UGVs in a variety of physical layouts would diversify the learning process.This would essentially learn a central policy which is much more versatile in avoiding obstacles and achieving its goal. Another potential development would be to make the system independent of the UGV dimensions, there by generalizing the learned model to any sized UGV.

This research gives an introduction of the elements that are needed to conduct a self-learning, physical system using reinforcement learning simulations. This method is expected to facilitate and expedite the learning process of a robot. Experimental validation shows that the physical hardware setup in this research for the tasks assigned is satisfactory for this end. In conclusion, we prepare a physical setup to expedite the process of learning by setting up the hardware for the UGV as well as creating the software framework needed for a continual learning system.

References

[1] M. Sato, K. Abe, and H. Takeda, "Learning control of finite markov chains with an explicit trade-off between estimation and control," *IEEE transactions on systems, man, and cybernetics*, vol. 18, no. 5, pp. 677–684, 1988.

[2] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[3] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.

[4] A. Majumdar, P. Benavidez, and M. Jamshidi, "Lightweight multi car dynamic simulator for reinforcement learning," in *2018 World Automation Congress (WAC)*, June 2018, pp. 1–6.

[5] A. Majumdar, P. Benavidez, and M. Jamshidi, "Multi-agent exploration for faster and reliable deep q-learning convergence in reinforcement learning," in *2018 World Automation Congress (WAC)*. IEEE, 2018, pp. 1–6.

[6] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[7] A. Ramezani Dooraki and D.-J. Lee, "An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments," *Sensors*, vol. 18, no. 10, p. 3575, 2018.

[8] Y. Li, "Deep reinforcement learning: An overview," *arXiv preprint arXiv:1701.07274*, 2017.

[9] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial intelligence*, vol. 55, no. 2-3, pp. 311–365, 1992.

[10] J. A. Bagnell and J. G. Schneider, "Autonomous helicopter control using reinforcement learning policy search methods," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 2, May 2001, pp. 1615–1620 vol.2.

[11] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, June 2015, pp. 1094–1099.

[12] ROS.org. Ros technical overview. [Online]. Available: http://wiki.ros.org/ROS/Technical Overview