Teaching Ground Vehicles to Navigate Autonomously with Reinforcement Learning

Ting-Che Lin
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`tingchel@andrew.cmu.edu`

Jiahong Ouyang
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`jiahongo@andrew.cmu.edu`

Dicong Qiu
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`dq@cs.cmu.edu`

Yuchi Wang
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`yuchiw@andrew.cmu.edu`

Yang Yang
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
`yy4@andrew.cmu.edu`

Abstract

This work involves teaching ground vehicles autonomous navigation policies. The project consists of two parts: (1) learning to navigate in a simulated environment using reinforcement learning and (2) transferring the trained model and policy onto a physical remote controlled ground vehicle. In this report, we propose an end-to-end deep reinforcement learning (deep Q-network) approach to learn environmental state-action values in high-dimensional continuous state space and generate discretized actions to control a vehicle in simulation. After the model and policy were transferred to a physical ground vehicle, the performance of learned model and policy were evaluated in an engineered maze environment.

1 Introduction

1.1 Motivations and High-level Goal

In 2010, the national statistics from the National Highway Traffic Safety Administration reported 5,419,000 crashes, 30,296 of which involved fatalities, resulting in 32,999 deaths and 2,239,000 injuries[1]. With the increasing number of motor vehicles in recent years, the number of traffic accidents is growing. Most of the cases are caused by human factors such as fatigue and inexperienced driving. With the gradual maturity of deep learning technologies, developing robust and safe self-driving vehicles becomes tangible.

Approaching autonomous driving in an end-to-end manner has been capturing industrial attentions in the recent years [2]. With the gradual maturity, standardization, and modularization of machine learning and deep learning technologies and toolboxes, researchers have explored the applications of these technologies in self-driving car lane keeping [3] as well as multi-model multi-task control [4]. Our project is a simplified version of these systems and tries to demonstrate the capability of reinforcement learning to navigate an autonomous vehicle without human supervision.

It is intriguing to explore end-to-end reinforcement learning in autonomous navigation. The general goal for this project is to leverage end-to-end deep reinforcement learning to teach ground vehicles to learn autonomous navigation skills in simulation and transfer the learned model and policy to a physical remote controlled ground vehicle, which we test in a physical environment.

1.2 Experiment Hardware Platform

The hardware platform used in this project is a 1/10 scaled remote controlled (RC) ground vehicle[1], including mounting assembly and protective bumpers. The ground vehicle chassis model is *MST FXX-D RWD* and the mounting assembly is set on top of the RC car, which houses a battery, a *Jetson TX1* single-board computer, a USB hub, an inertial measurement unit (IMU), a Lidar and an on-board Wi-Fi module. An image of the experiment hardware platform is shown below in figure 1.

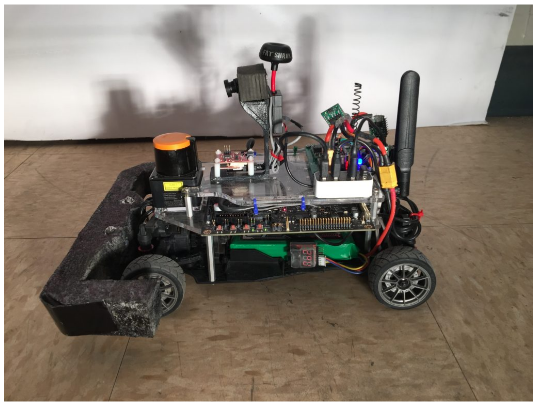


Figure 1: the remote controlled ground vehicle as the experiment hardware platform of the project

1.3 Primary Challenges

Some key challenges have been identified during the realization of the autonomous system, as listed below:

- **The assembly of the vehicle**. At the beginning, we tried to build the car from scratch ourselves, but since some of the key components were not ordered on time, we decided to borrow existing hardware instead. The availability of an off-the-shelf robotic hardware platform for experiment is an important factor impacting the progress of the project.
- **The usage of hardware platform.** The vehicle has an on-board Wi-Fi module to communicate with the control computer. We had a difficult time setting up the router and establishing a link between these two components. In addition, integrating our code into the existing framework of the RC car also proved to be a challenge.
- **The usage of simulator.** Before the midterm, we tried the SUMO simulator, but we found that this simulator is fundamentally a traffic simulator and only supports moving forward or stopping. Therefore we switch from SUMO to Gazebo so that we can incorporate realistic physics and steering control into our model.
- **Transferring learned policy from simulation to the real world.** The primary technical challenge was figuring out how to transfer the learned policy effectively to the real world. We approach this challenge by keeping the real world observations as close as possible to the observations in the simulator. For example, we used a non-reflective cardboard like material to construct our maze because it provides a very clean laser scanner results. Additionally, we also tuned the turning angle of the car so it relate closely to the turning angle of the car in the simulator.

2 System Architecture

The overall system can be divided into three parts:

- The **deep reinforcement learning agent (the deep Q-network) controller**, which is controls the behavior of the car in both the simulated environment and the physical world.

[1]Credit to the MRSD'16 Team D

- The **car simulator**, which simulates the car dynamics in a virtual environment and provides us with a suitable environment to train the deep reinforcement learning agent (the deep Q-network);
- The **remote controlled ground vehicle**, also known as the **RC car**, is the physical platform that hosts the deep reinforcement learning agent allows us to visualize the learned model and test the policy in the physical world.

The software architecture of the deep reinforcement learning agent controller and the hardware architecture of the remote controlled ground vehicle will be further discussed in the following sections.

2.1 Controller Software Architecture

From the software perspective, the drivers that connect the ROS system with the car simulator and the physical remote controlled ground vehicle are needed. Besides, the core of the software architecture is an end-to-end deep Q-network controller, which consists a feature extractor that extract the features from the raw sensor input to reduce state dimension and a deep Q-network that estimates the state-action values of different actions in specific states.

To better explain our software architecture, a diagram was created to illustrates different components of the software and the relationships among them, as shown below in figure 2.

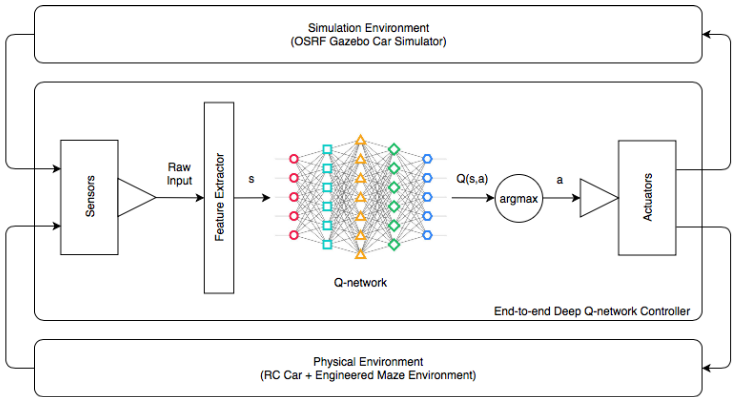


Figure 2: the software architecture of the controller system

The deep reinforcement learning agent (the deep Q-network) controller is the core part of the software architecture. It takes in the Lidar input of either the simulated car in the simulation environment or the physical remote controlled ground vehicle platform in the physical environment. And it sends out control output to either the simulated car or the physical vehicle. Within the controller, the raw data from the sensors first go through a feature extractor, which perform dimension reduction on the raw sensing input. The implementation of the feature extractor can be an auto-encoder neural network [5], or just passing the raw sensing input to the next component if the Q-network can actually handle the raw data. A deep Q-network [6] follows the feature extractor, and takes in a feature space value as the state and estimates the state-action values associated with each action (discretized in a continuous action space). An argmax operator is used to select the final action which is then sent to the actuators to execute, producing an effect on the external environment. We are using a prebuilt Gazebo simulator environment however we have implemented our own DQN network.

2.2 Simulation Environment

Due to the nature of reinforcement learning, it is highly impractical to perform all the training in the real world. Instead, it is much more realistic to train in simulation and fine-tune in physical environment. A simulator provides several benefits that can facilitates the training process: (1) the simulated environment can be modified and reset conveniently; (2) it is possible to train a models in parallel; (3) it does not depends on the mechanical components of the hardware platform, which is important since the vehicle will be crashing constantly during training and requires manual reset.

For simulator trade study, we have explored and tested different off-the-shelf self-driving car simulators, namely the Simulation of Urban Mobility (SUMO) environment [7], the CARLA Simulator [8], the OSRF car simulator [2], and Gazebo [9] along with OpenAI Gym [10] extension called `gym-gazebo` [11]. The SUMO environment is a multi-agent vehicles simulator, which is suitable for city-scale traffic applications but is insufficient in simulating detailed vehicle dynamics. CARLA is a high resolution and realistic simulator incorporating detailed car dynamics and driving environment settings. However, it is beyond the simulation requirements of our project, and introduces unnecessary complexity into the project. With similar reasons to CARLA, the OSRF car simulator is not considered to be used in this project.

With the above, `gym-gazebo` is chosen as the simulation environment for the project, because it provides users with a realistic physics-based simulation of the subject robot and allows easy integration with existing hardware. It also has a variety of built-in simulated sensors and predefined robot dynamics, and supports ROS naturally. The similarity of the robot dynamics within the simulator can be easily transferred to the RC car used as our hardware platform. A pre-built environment that comes with `gym-gazebo` was used for the training of our model.

2.3 Ground Vehicle Hardware Architecture

The remote controlled ground vehicle consists of sensing module, on-board computing module, communication module, power supply module, actuators and chassis, as shown in figure 3 below.

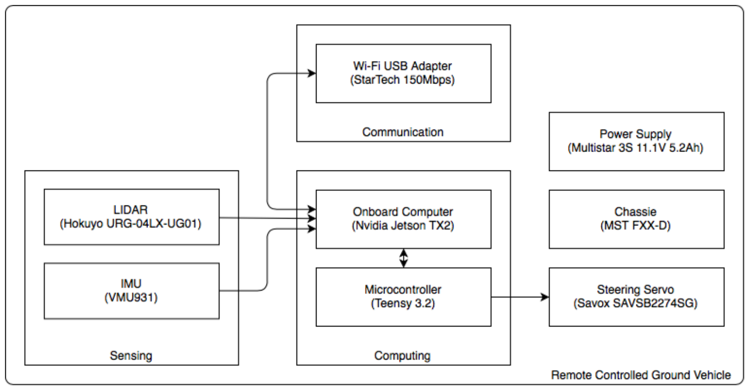

Wi-Fi USB Adapter (StarTech 150Mbps)
Communication
Power Supply (Multistar 3S 11.1V 5.2Ah)
LIDAR (Hokuyo URG-04LX-UG01)
Onboard Computer (Nvidia Jetson TX2)
Chassie (MST FXX-D)
IMU (VMU931)
Microcontroller (Teensy 3.2)
Steering Servo (Savox SAVSB2274SG)
Sensing
Computing
Remote Controlled Ground Vehicle

Figure 3: the hardware architecture of the remote controlled ground vehicle

For the sensing module, we are going to take advantage of the Lidar and IMU sensors on the hardware platform, whose data will flow into the on-board computer for further processing. The hardware platform is also equipped with a Wi-Fi adapter that connects the ground vehicle and the master computer the operator uses. The micro-controller also connects to the on-board computer converting action output to low-level motor signals to drive the vehicle. The power supply is actually a on-board battery that power all the electronic parts of the vehicle. We have built an RC car however for our final demo, we are using an existing car from a past team.

3 Implemented Components

3.1 Learning from Simulation

For reinforcement learning, we are using the `gym-gazebo` simulator, a ROS-based Gazebo simulator that interfaces with Open AI gym [11]. We are choosing this simulator for several key reasons. First, we would like to use the ROS architecture as much as possible because most open-source robotic toolkits and support are on the ROS platform, we would need to interact with a robot that is likely controlled by ROS, and all of our team members are familiar with ROS. This is in contrast to other car simulators such as AirSim or Carla which, although more powerful, are harder to setup, require more computational resources, and do not have Lidar sensor output.

[2]`https://github.com/osrf/car_demo`

Furthermore, gym-gazebo comes with predefined environments and vehicles that eliminate the trouble for us to design, configure, and integrate our own vehicles. Specifically, we are currently using the GazeboCircuit2TurtlebotLidar-v0 environment which is a simple square maze with walls that is designed for vehicles with planar Lidar's. In addition, we are currently training with a Turtlebot vehicle as it is the default vehicle for the simulator and is equipped with a planar Lidar which is similar to the Hokuyo on the RC car. To limit the scope of the project, we have constrained the action space to be a one dimensional discrete action space for only the steering. The steering angle is discretized between -0.5 to 0.5 in 21 steps and the acceleration is set at a constant 0.2.

Recall that our original goal is to simply teach a car to drive itself. We would like our agent to be able to navigate around without colliding with obstacles. To that extent, we are not imposing any goal or destination constraint. We do not care where our vehicle goes as long as it is able to keep moving and avoid obstacles. As a result, we have shaped our rewards as follows: a linearly decreasing reward from 5 to 0.5 that scales with the absolute value of the steering angle and a -200 reward for a collision. We want the agent to move forward so we penalize actions that result in circling in place but our ultimate goal is to avoid collisions so we penalize it heavily for that.

We are currently using a Deep Q Network (DQN) as our architecture. The DQN is a 3 layered feedforward neural network with 300 units per layer and uses ReLU activation. In DQN, we approximate the action-value function with a deep neural network $Q(S, A)$. At each step of an episode, we choose a action A_t from states S using a policy derived from Q. We observe the next state S' and the reward R and we update the weights of the Q-Network:

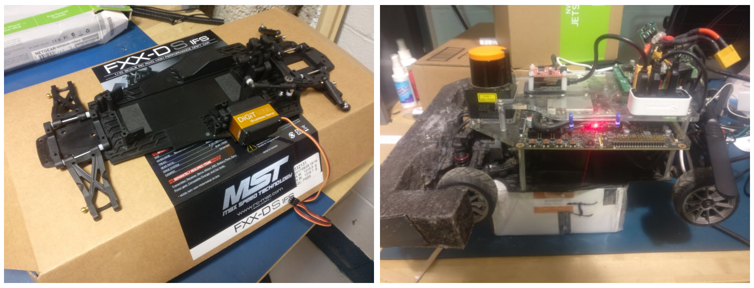
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{1}$$

In addition, we are using two modifications to the vanilla DQN architecture to improve performance: experience replay to remove correlation between successive data points and a target network to evaluate the agent's actions and stabilize the training [12].

In hindsight, it may have been better to pursue a method such as DDPG[13] that operates directly over a continuous action space, thus eliminating the need for discretization. However, at the time, we were more confident in implementing and debugging a DQN and we also did not believe that our action-space was complex enough for a DDPG to make a big difference.

3.2 Ground Vehicle Assembly

For the ground vehicle, we have assembled the chassis and the necessary electrical components, as shown below in figure 4a. However, after the midterm, the solid axle plate was still missing which is essential for the transmission of motor torque to the wheels. The assembling of the experimental platform has been severely obstruct the progress of the entire project.



(a) assembled chassis **(b)** adapted borrowed RC car

Figure 4: the initial assembled RC car chassis and the RC car hardware platform borrowed from MRSD'16 Team E after custom adaptation for the requirements of the project

In order to boost the progress, it has been decided to borrow the existing RC car platform from MRSD'16 Team E and adapt the borrowed platform to the requirements of our project instead. The adapted borrowed experimental hardware platform is shown in figure 4b. The necessary control software has been installed on the RC car and the control this platform through Wi-Fi has also been addressed. The detailed documentation on MRSD'16 Team E has provided great support for our project.

4 System Evaluation

A video documenting the performance evaluation of the system is available online[3].

4.1 Component Evaluations

4.1.1 Hardware Platform

The performance of the ground vehicle hardware platform is evaluated based on the fidelity of the steering angle and the accuracy of the laser scanner's input. To evaluate the fidelity of the steering angle, we will issue several steering angle commands and measure the actual output vs the angle commands. The actual output angle should be within 5 degrees of the issued command angle. For the laser scanner, we will evaluate it empirically by visualizing the laser scan output and comparing the output to the real environment.

4.1.2 Deep Q Network

The performance of the DQN implementation can be directly evaluated from the learning curve as shown in section 4.3.2. We evaluated the DQN by checking if it learned the dynamics of the simulated environment and was able to avoid the walls when navigating. Specifically, our reward for the agent is a linear scaling of the angle if there is no collision and a flat -200 if there is collision. Based on this reward structure, we expected the simulated agent to be able to travel mostly in a straight line but turn to avoid the obstacle walls in the environment. The environment was not assigned with any higher level goals such as reaching a certain location in the environment, and there was neither any terminal state aside from being in a collision. Thus, in the optimal case, the environment is not a finite horizon one and it is expected the agent would be able to navigate forever in this setup.

4.2 Complete System Evaluation

Evaluation of the complete system is comprised of evaluation of the individual subsystems and the integration of the physical platform with the simulated agent. Specifically, it was expected the physical RC car could be able to move forward, backward, left and right, while detecting objects in the environment accurately with a planar Lidar. Furthermore, it shall be able to interact with ROS. Additionally, we evaluated the DQN based on its ability to avoid detected walls and obstacles.

Together, we evaluated the complete system on a physical track that closely mirrored the virtual track in the simulation environment. Owing to limitations in available space and material, we were unable to create a track in the same scale as the simulated environment. However, the agent was trained with a very simple reward structure and therefore the high-level layout of the track shall not impact the actual performance. A smaller track with roughly the same width and curves is sufficient for the evaluation of the complete system.

We evaluated the complete system based on the number of collisions it had with the walls. This metric is closely related to the reward and the training process.

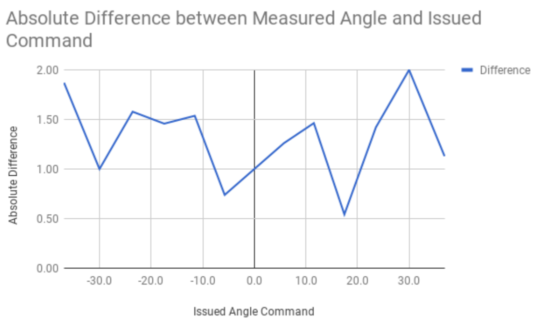
4.3 Results

4.3.1 Hardware Platform

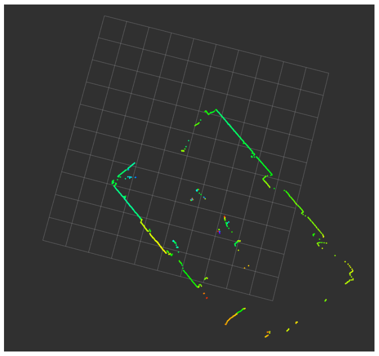
Steering Angle Evaluation. We evaluated the performance of the ground vehicle steering by assessing the absolute difference between the measured angle and the issued command angle. The criterion of this evaluation is that the difference between the two angles shall be within 5 degrees. It was observed in the evaluation experiment that the difference between the actual angle and the measured angle is within 2 degrees as shown in figure 5a, which suggested the system has satisfied the steering performance criterion.

[3] `https://youtu.be/zcsYNvxJ0bQ`

Lidar Perception Evaluation. We evaluated the Lidar perception performance by using it to scan the MRSD Lab in the NSH basement and visualize the output point cloud. The evaluation experiment suggested this component was successful in outputting a point cloud skeleton which empirically resembled the outline of the lab, as shown in figure 5b.



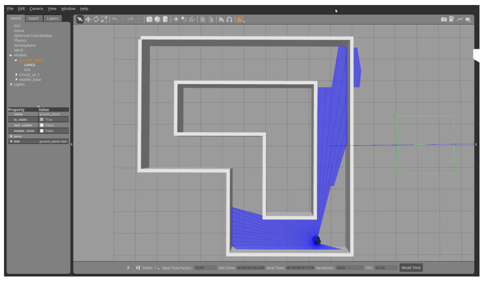
(a) steering angle evaluation result



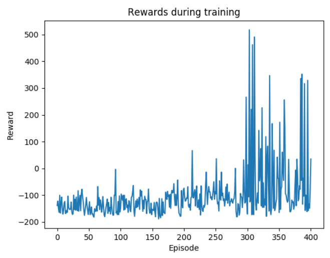
(b) Lidar perception evaluation result

Figure 5: evaluation results of the hardware platform, including a graph of the absolute difference between the measured angle and the issued command for steering angle evaluation and the point cloud resembling the MRSD Lab outline for Lidar perception evaluation

4.3.2 Deep Q Network



(a) training environment with `gym-gazebo`



(b) learning curve of the DQN

Figure 6: a screenshot of the visualized simulation environment with an agent running inside the simulated maze and the corresponding learning curve of the deep Q-network of the agent

Figure 6b shows the current training progress of our vehicle in `gym-gazebo` while Figure 6a shows the simulator itself. As the plot shows, the training is unstable - in fact, we have seen very different results from different initialization and seeds of the agent. However, the agent is definitely able to learn to avoid the obstacles to a noticeable extent. We can see from this link at 150 episodes in the training process that the agent has been able to approximate the value-function somewhat well.

4.3.3 Complete System

We evaluated the entire system on physical tracks that are engineered with similar parameters to the simulated environment. Out of 6 trial, with each comprised of a forward and backward pass through the track, no collision has been observed, though the vehicle sometimes moved closely to a wall. The success rate was 100%. Videos of the trials are available[4].

4.4 Achievements with the System

In this project, we were able to successfully navigate a physical car in a simple constrained environment using deep reinforcement learning. Our major achievements in this project are (1) training an

[4]`https://drive.google.com/open?id=1ODbB6dA0Co-lAanBEZlR5edjznQKIwFA`

autonomous agent in a simulated environment and (2) effectively transferring that learnt policy onto a real, physical agent.

First, training an agent using deep reinforcement learning is not simple, even when using a simulator. Until several years ago, it was believed that modelling an environment as a Markov Decision Process and learning the Q-value was too unstable in reality. It was not until the introduction of deep neural networks and experience replay that progress was seen in the field of Deep RL. Through careful selection of our state input, action-space output, and an appropriate reward function, we were able to achieve convergence with our system in simulation.

Second, transferring from a simulated to a real environment is also challenging due to the inherent differences between the two environments. First, we had to ensure that our physical platform was built to specification and capable of performing the tasks needed. This required careful selection of components and thorough integration of the entire system. Furthermore, there will always be differences between a simulated and real environment and we had to address those differences accordingly. This aspect will be discussed in more detail in Section 4.5.2.

4.5 Main Limitations and Future Work

4.5.1 Deep Q Network

A deep Q network works well for deterministic environments with discrete action spaces but it does not work well for stochastic environments and continuous action spaces. Unfortunately both of these environment types are applicable to our project. In real life, stochastic elements can include dynamic obstacles, wheel slip, and other inconsistencies that are not seen in simulation. In addition, a car's steering, acceleration, and braking are all continuous. A DQN may work well for a simple, constrained environment with only 1 discrete action - as in our case - but it likely will not generalize to more complex environments. There are many avenues with which we can address this in future works. First, we can implement policy gradient methods since these operate directly on the policy and can inherently incorporate stochasticity. As well, we can include a fuller assortment of actions that would be available to an action car. These include acceleration (gas) and braking controls. Lastly, we would prefer not to discretize the action space in the future since it quickly leads into the curse of dimensionality. Instead, methods such as DDPG[13] operate directly over a continuous action-space and are more suitable to our project.

4.5.2 Differences in real and simulated environment

We encountered some issues when trying to port the agent from the simulated environment into a real car with real obstacles. First, we were using a different physical vehicle from the vehicle in simulation. The real vehicle is a 4-wheeled Ackermann drive RC car while the simulated vehicle is a turtlebot that can turn in place. Clearly the dynamics for these two vehicles are very different. We didn't notice any significant issues due to this difference but in the future we would like to remain consistent from simulation to reality.

Furthermore, the sensors specifications do not match completely. Specifically, the Hokuyo Lidar on the RC car has 1081 laser beams while the simulated Lidar has 100 beams. We addressed this problem by simply sampling the beams. In the future, we would increase the capacity of the neural network in the simulator to match the state input of the actual Lidar.

Lastly, there was a considerable amount of Lidar noise in the real world. We noticed on the `rviz` display that there are many artifacts in the laser scans due to metallic surfaces and clutter (see figure 5b). This severely affected the performance of the agent. It is an empirical rule in machine learning to avoid unseen data. In the future, the Lidar data can be pre-processed by adding noisy measurements to simulation to make it more realistic.

5 Organization

5.1 Work Division

The team members have been worked collaboratively throughout the entire project under reasonable workload division. Namely, *Ting-Che Lin*, *Jiahong Outyang* and *Yang Yang* have been primarily focused on the hardware assembly, ground vehicle control software. *Yuchi Wang* and *Dicong Qiu* were working on the simulation environment setup and model training. All team members were involved in the model transfer from the simulator to the physical platform and paperwork.

5.2 Primary Obstruction

The hardware platform was the primary obstruction of the project. The original plan for the project was to assemble a new ground vehicle hardware platform for experiment, following the design of an autonomous vehicle project in Carnegie Mellon University. However, due to the insufficient information about the detailed parts requirements and the process of placing orders for parts, the new ground vehicle was not able to be completed. Instead, we borrowed an existing ground vehicle platform from MRSD'16 Team E and changed some parts to adapt the requirements of our project.

5.3 Key Challenges

- **The assembly of the vehicle**. At the beginning, we tried to build the car from scratch ourselves, but since some of the key components were not ordered on time, we decided to borrow existing hardware instead. The availability of an off-the-shelf robotic hardware platform for experiment is an important factor impacting the progress of the project.
- **The usage of hardware platform.** The vehicle has an on-board Wi-Fi module to communicate with the control computer. We had a difficult time setting up the router and establishing a link between these two components. In addition, integrating our code into the existing framework of the RC car also proved to be a challenge.
- **The usage of simulator.** Before the midterm, we tried the SUMO simulator, but we found that this simulator is fundamentally a traffic simulator and only supports moving forward or stopping. Therefore we switch from SUMO to Gazebo so that we can incorporate realistic physics and steering control into our model.
- **Transferring learned policy from simulation to the real world.** The primary technical challenge was figuring out how to transfer the learned policy effectively to the real world. We approach this challenge by keeping the real world observations as close as possible to the observations in the simulator. For example, we used a non-reflective cardboard like material to construct our maze because it provides a very clean laser scanner results. Additionally, we also tuned the turning angle of the car so it relate closely to the turning angle of the car in the simulator.

5.4 Lessons Learned

The primary lesson learned from this project is that we shall have a good planning and schedule for the entire project, especially when hardware is involved. It has been identified that the hardware assembly is a primary obstruction of our project, and due to the hardware issue, we were not able to deliver much better results in more complex environments. If we had organized the hardware parts ordering and the hardware platform assembly better, we might be able to improve our algorithm and handle more complicated situations.

Also, it would be great if the course itself can provide pre-assembled and well-tested hardware platform, so that the students could focus on the core problems, instead of the hardware issues.

If we were to do this project again, we may not be struggling in trying to assemble our own ground vehicle, but use an existing hardware platform, such as the one we borrowed from MRSD'16 Team E, so that we can focus more one the actually algorithm design and implementation for ground vehicle autonomous navigation.

References

[1] National Highway Traffic Safety Administration. Fatality analysis reporting system, 2010.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[3] Zhilu Chen and Xinming Huang. End-to-end learning for lane keeping of self-driving cars. In *Intelligent Vehicles Symposium (IV), 2017 IEEE*, pages 1856–1860. IEEE, 2017.

[4] Zhengyuan Yang, Yixuan Zhang, Jerry Yu, Junjie Cai, and Jiebo Luo. End-to-end multi-modal multi-task vehicle control for self-driving cars with visual perception. *arXiv preprint arXiv:1801.06734*, 2018.

[5] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[7] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[9] N Koenig and J Hsu. The many faces of simulation: Use cases for a general purpose simulator. In *Proc. of the ICRA*, volume 13, pages 10–11, 2013.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[11] Iker Zamora, Nestor Gonzalez Lopez, Victor Mayoral Vilches, and Alejandro Hernandez Cordero. Extending the openai gym for robotics: a toolkit for reinforcement learning using ros and gazebo. *arXiv preprint arXiv:1608.05742*, 2016.

[12] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

[13] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2015.