Some fancy title

Guan-Horng Liu

May 5, 2017

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
George A. Kantor
Manuela Veloso
Devin Schwab

*Submitted in partial fulfillment of the requirements
for the degree of Masters of Computer Science.*

Abstract

TODO

Acknowledgments

TODO

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid urbanization globally in the recent past has led to severe road congestion, rise in pollution levels and an increase in road accidents, therefore presenting a very grim picture of the current state of urban transportation. At the moment, private automobiles are widely recognized as an unsustainable solution for the future of personal urban mobility [1]. Fortunately however, great strides have been made in the development of autonomous driving technologies, energized by the successful demonstrations by some teams at the DARPA Urban Challenge [2, 3]. While offering an opportunity to develop sustainable and safe solutions to personal mobility [4], they also hint at a complete overhaul of the urban transportation landscape by ushering in Autonomous Vehicles-on-Demand.

1.1 Planning

1.2 The Rise and Challenge of End-to-end Learning

In the past few years, there is a great interest in applying a more end-to-end approach [5, 6, 7] wherein one can learn a complex mapping that goes directly from the input to the output by leveraging the availability to a large volume of task specific data. More recently, the approach has been pushed one step forward thanks to the success of deep reinforcement learning (DRL), which has shown to achieve human-level performance on many gaming environments [8, 9, 10]. DRL provides a much better formulation that allows policy improvement with feedback, while the traditional deep supervised learning-based driving requires labeling and may not be able to deal with the problem of accumulating errors [11].

Despite the successful applications of deep neural net as a highly non-linear function approximator, it is often criticized as very data-sensitive and performing more like a purely black box. These characteristics prevent the engineers from semantically investigating the some crucial property such as robustness and generalization in order to extend to the real-world applications.

thus not intuitively allow engineers to semantic . For the

In fact, with a closer look at the recent rise of autonomous driving,

sensitive to training data and often used as a black box one of the key challenges for this approach to extend to the end-to-end controller in robotics applications is the black-box nature of the deep neural net.

the lack of the capability to visualize the semantic meaning.

Despite the powerful deep neural net as a function approximator introduced by, one of the key challenges for However, one of the key challenges for the end-to-end approach is to hard to visualize semantic meaning black box -> no guarantee on raise safety concern ... no use of sensor fusion

It is clear sensor fusion is indispensable for autonomous driving, in order to improve accuracy and robustness in the vehicle's algorithmic decision making. Indeed, multi-modal perception was an integral part of autonomous navigation solutions and even played a critical role in their success [3] before the advent of end-to-end deep learning based approaches. Sensor fusion offers several advantages namely robustness to individual sensor noise/failure, improved object classification and tracking [12, 13, 14], robustness to varying weather and environmental conditions, etc.

Multi-modal deep learning, in general, is an active area of research in other domains like audiovisual systems [15], gesture recognition [16], text/speech and language models [17, 18], etc. However, in the space of end-to-end sensorimotor control for autonomous navigation, this multi-modal outlook has not received much attention and is need of the hour.

Currently, the most promising demonstration of an effective end-to-end framework for autonomous driving is from NVIDIA [6]. While this work has some form of sensor fusion, it should be noted that they fuse information from the *same* type of sensor (cameras in this case) but placed in different, but strategic, locations on the car. We argue that a more robust strategy for sensor-fusion is to obtain information from different type of sensors.

Recently, there is also a great interest on extending DRL approaches to multi-input fashion in order to tackle complex robotics tasks such as human-robot-interaction [19] and manipulation [20]. Recently, Mirowski et al. [21] proposed an novel approach namely *NAV A3C* and uses information such as vision, depth, and agent velocity for maze navigation. However, information such as depth is only used during training as an auxiliary loss, and it is absent during testing. Moreover, the system dynamic in maze is relatively simple compared with autonomous driving. Here, we are more interested in the aspect of sensor fusion No partial sensor failure is addressed.

We argue that this problem is critical as a further step toward the real-world robotics application given the current state-of-the-art DRL agents on many realistic simulators.

Chapter 2

Related Work

TODO

Stochastic regularization is an active area of research in deep learning made popular by the success of, *Dropout* [22]. Following this landmark paper, numerous extensions were proposed to further generalize this idea such as *Blockout* [23], *DropConnect* [24], *Zoneout* [25], etc. In the similar vein, two interesting techniques have been proposed for specialized regularization in the multi-modal setting namely ModDrop [16] and ModOut [26]. Given a much wider set of sensors to choose from, ModOut attempts to identify which sensors are actually needed to fully observe the system behavior. This is out of the scope of this work. Here, we assume that all the sensors are critical and we only focus on improving the state information based on inputs from multiple observers. ModDrop is much closer in spirit to the proposed *Sensor Dropout (SD)*. However, unlike ModDrop, pre-training with individual sensor inputs using separate loss functions is not required. A network can be directly constructed in an end-to-end fashion and *Sensor Dropout* can be directly applied at the sensor fusion layer just like Dropout. Its appeal lies in its simplicity during implementation and is designed to be applicable even to the DRL setting. As far as we know, this is the first attempt at applying stochastic regularization in a DRL setting with the spirit of sensor fusion.

Chapter 3

Kinodynamic Planning for All-Terrain Vehicle

In this chapter, we propose a model-based local planner for off-road navigation application. In addition to the basic capabilities such as static obstacles avoidance, the ATV should perform smooth but aggressive maneuvering in high-speed with a complex vehicle dynamic model. Although vehicle planning can be treated as a well-investigated kinodynamic planning problem in control space, several challenges raised when operating in off-road environment. First, the vehicle dynamic in off-road environment is much more unpredictable in contrast to on-road condition. Factor such as wheel-terrain interaction for modeling the sliding effect is still an active research area. Secondly, developing a local planner for high speed operation can be categorized as an anytime planning problem. Thus, computational efficiency should be taken into account for the real-time concern.

3.1 Planner Modules Design

The block diagram of our planner modules is shown in Fig. 3.1, which can be separated into two modules with respect to functionality. The collision check module constructed a global simplified occupancy grid from vehicle current odometry and pure point cloud data, then communicated with RRT-based planner for collision check service. The velocity command is generated by planner and sent directly to the on-board velocity controller for execution. Visualization tools are also built via Rviz interface. The detail of two modules is described as follow:

3.1.1 Collision Check Module

Occupancy grid is a commonly-used data structure for obstacles detection. It stores one or multiple probabilities in each grid cell, and increases or decreases them based on sensor model. Since our testing scenario is relatively flat without noise, a simplified version of occupancy grid is used in the matter of fast implementation, in which we replaced the probabilities with a counter. Three different methods for obstacles segmentation were investigated and described below:

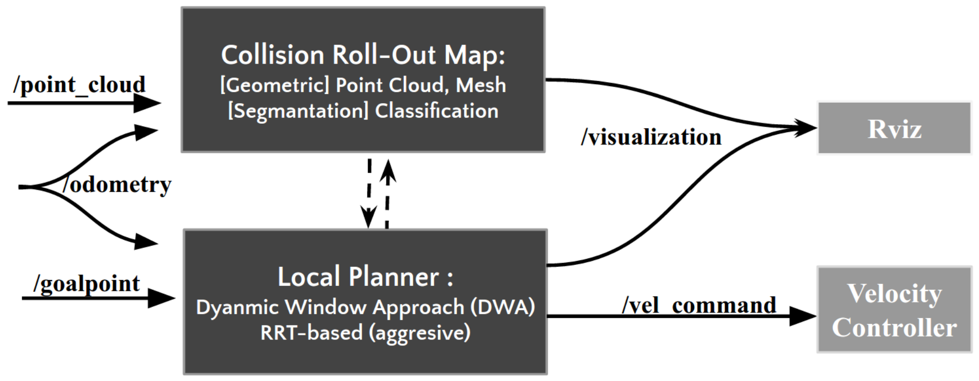
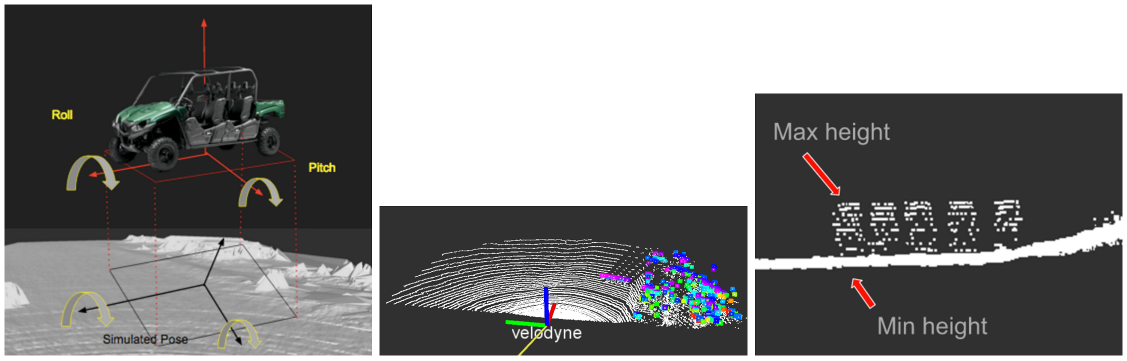


Figure 3.1: Block Diagram of our planner API.



(a) Mesh with ODE simulation (b) RANSAC segmentation. (c) Height map algorithm

Figure 3.2: Three different methods of collision check. Note that for (b), the original and the the processed point cloud is represented with white and colored dot, respectively.

i. Mesh Representation with Simulation in Open Dynamic Engine (ODE)

The original method implemented on the vehicle uses ODE and mesh data to simulate the vehicle pose on the ground. Collision is reported if an intersection was detected between vehicle and mesh or the simulated roll and pitch were beyond user-defined thresholds.

ii. Plane Removal with RANSAC Segmentation

The second approach for collision check module is using RANSAC segmentation from Point Cloud Library (PCL) to fit the plane model. In our case, the plane model is the ground of our testing environment. We extract the outliers from RANSAC for obstacle detection. As shown in Fig. 3.2b, the whtle point cloud is the original data from Velodyne Lidar. The colorful point cloud is the outliers from RANSAC.

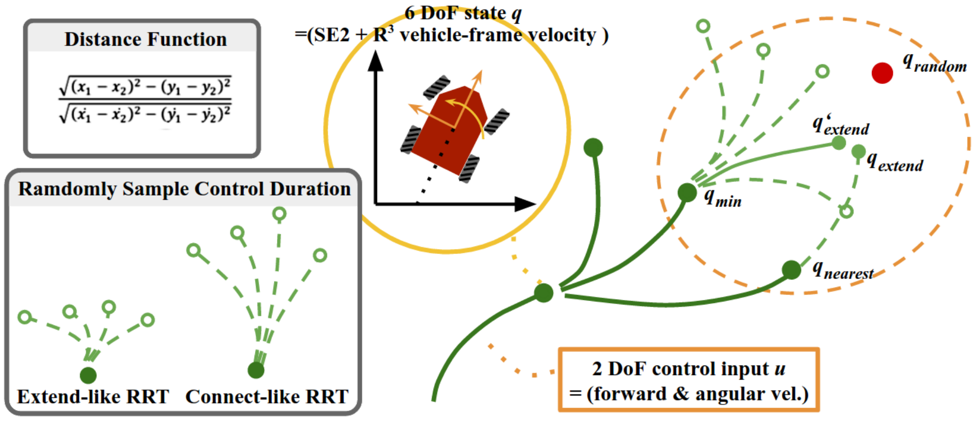

Distance Function
$\frac{\sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2}}{\sqrt{(\dot{x}_1 - \dot{x}_2)^2 - (\dot{y}_1 - \dot{y}_2)^2}}$
Ramdomly Sample Control Duration
Extend-like RRT
Connect-like RRT
6 DoF state q
=(SE2 + R^3 vehicle-frame velocity)
2 DoF control input u
= (forward & angular vel.)
q_{random}
q'_{extend}
q_{extend}
q_{min}
$q_{nearest}$

Figure 3.3: The visualization of RRT-based planner.

iii. Height Map Algorithm

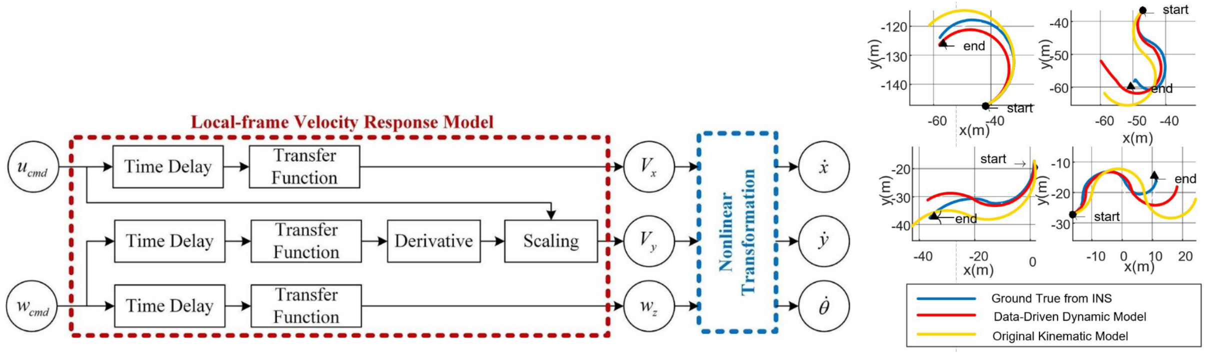
The third method we used is height map algorithm. This is a simple and efficient algorithm in terms of computation. It calculates the height differences within one grid. If the height difference is greater than user-defined threshold, it will be categorized as an obstacle. As shown in Fig. 3.2c, the artificial obstacle is approximately 1.5 meters. We set the threshold to be 1 meter. Thus it will be recognized as an obstacle.

Since collision check is the most computationally expensive part of our system and we cannot afford to collide our platform with the obstacles. Efficiency and reliability are the most important requirements. The mesh representation is a good approach for future application such as driving on rough terrain. However, it was not feasible for our scenario in terms of computation consumption. The RANSAC segmentation is sensitive to off-road conditions; the plane model cannot be perfectly fit on rough terrain. In addition, the dusty environment in off-road driving create noises and interference to the Lidar. Considering our requirements and the discussion mentioned above, we choose height map algorithm as our final approach. ItâĂŹs the fastest and the most reliable. Furthermore, to optimize the computing efficiency, we used bitwise operation instead of multiplication and the obstacle size are dilated to increase robustness of our system.

3.1.2 RRT-based Planner Module

Instead of using traditional search-based planner such as A* or D*, we use a sample-based planner as our development platform. This critical choice comes from an insight that sample-based planner is more efficient for solving a high dimensional planning problem, which gives us a powerful tool when we want to utilize a more complex dynamic vehicle model for state propagation. Besides, maneuvering in wilderness can be seen as a generalized planning problem where discretizing the world based on resolution might not generate a smooth path.

Simulation was first conducted within Open Motion Planning Library (OMPL) framework to investigate the performance among different sample-based planners. We used OMPL.app to simulate the behavior of different planners based our vehicle model and a specific map. Also,



(a) (b)

Figure 3.4: (a) Data flow of vehicle dynamic response model. The model takes two velocity control commands as input and estimates the velocity response in vehicle frame. (b) Comparison between data-driven dynamic model and original kinematic model. Note the time interval in each graph is 20 seconds.

we used a benchmark tool provided by OMPL to compare the performance of different planners with different parameters. It turned out RRT planner is one of the competitive candidates. Since all of us are more familiar with RRT planner, we decided to use it as our baseline planner.

Fig 3.3 visualizes the RRT tree. Each node represents a 6 DoF states $q = [x, y, \theta, v_{forward}, v_{sliding}, w]^T$, where the first three terms stand as a SE2 state space, and the last three terms stand as a vehicle-frame velocity in 2D plane. For control space, we utilize velocity control input, i.e. 2 DoF including forward and angular velocity, since it is commonly-used for autonomous ground vehicle, and is already built on the current vehicle controller system.

In order to overcome the unpredictable characteristic of vehicle dynamic in off-road environment, an experimental-based model was addressed via field testing. More specifically, data is collected on field among different set of control inputs, then a predictive model $\dot{x} = f(x, u, t)$ is constructed off-line with standard system identification process. As shown in Fig. 3.4a, the dynamic response of each velocity component at vehicle-frame is modeled as a first/second-order transfer function with time delay. Since the data-driven vehicle response model gives us a good estimation of the sliding velocity, which could plays a non-negligible role in off-road cases, it shows in Fig 3.4b a sufficient state estimation result when compared with the original kinematic model.

As a standard procedure of RRT in control space, a control input set and its operation duration should be determined in order to propagate toward an extended state after a random state is sampled. We first uniformly sampled the forward velocity command within an adjustable region. This command region is determined at each iteration based on previously issued command, current vehicle status, and previous executed path such that the vehicle will hold the velocity consistency without jerky output. Then, shooting method [X] is used to determined the angular velocity command. Another modification we applied is the random sampling of control duration (see Fig. 3.3). The initial thought came from our simulations in Section III, where we discovered choosing either extended-like RRT or connected-like RRT stands as an crucial factor for solving different maze scenarios. We believe loosening the constraint with such randomly sample

fix this

mechanism will generalize our planner for various problems.

Finding an optimal plan is crucial for motion planning but computationally expensive if using a sample -based control-space planner. To overcome it, we implemented a time-optimal RRT* work from Frazzoli []. Moving from RRT toward RRT* includes two more optimization steps in each iteration: reconnecting of extended state, and tree edges trimming. For this project , only the first of two optimization steps was implemented because the trimming process in the second step includes updating the whole children tree, which will trade off with planning time. Since it is almost impossible to propagate to the same state in 6 DoF state space, the re-connection process from extended state to minimal-cost state instead of nearest state will require updating the extended state (i.e. q'_{extend} q_{extend} in Fig. 3.3) if they are close enough. Last, since we would like to minimize our traveling time, the traveling time is estimated when calculated distance instead of Euclidean distance in 2D state space.

remove this

The re-planning process is designed as follow: at the beginning of each planner loop, vehicle status, collision check map, and goal point are updated to formulate a RRT control-space planning problem. Then, we solved such problem many times and published the best one we had so far when the planner loop terminates. There is no tree maintenance process, i.e. we throw away the whole RRT tree, in each solving process. Though re-generating the whole RRT tree makes no sense at first glance, in practical we found out such design can prevent the planner from publishing poor solution if bad tree structure was built at the beginning of growing stage. We admitted that if an optimal solution can be obtained from one single shot, the replanning setting would have changed significantly. Since there is a time delay between when vehicle status is updated and when the velocity command generated by RRT-based planner is executed, we utilize the same data-driven dynamic model to estimate the vehicle state after such time delay. The parameters used in our local planner are listed in TABLE I.

3.2 Experiments

3.2.1 Platform Introduction

Our platform comes from an undergoing project at Robotics Institute cooperated between Field Robotics Center (FRC) and Yamaha Motor Company, which aimed to develop an autonomous vehicle for off-road driving in wilderness environment. Yamaha Viking VI side-by-side ATV is served as our main testing platform, which is shown in Fig. 3.5. The vehicle is equipped with custom drive-by-wire system, velocity controller, and navigation sensors such as GPS/INS, LiDAR, and RGB-D camera. Currently developing software modules include classification, pose estimation, global and local planner.

Our testing scenario is designed as followed : the vehicle should perform S-shape maneuvering without collision with any static obstacles based on a model-based local planner. Standard operating speed ranges from 10 to $40kph$; the baseline testing speed in our case is set as $20kph$ with the top speed of $30kph$.

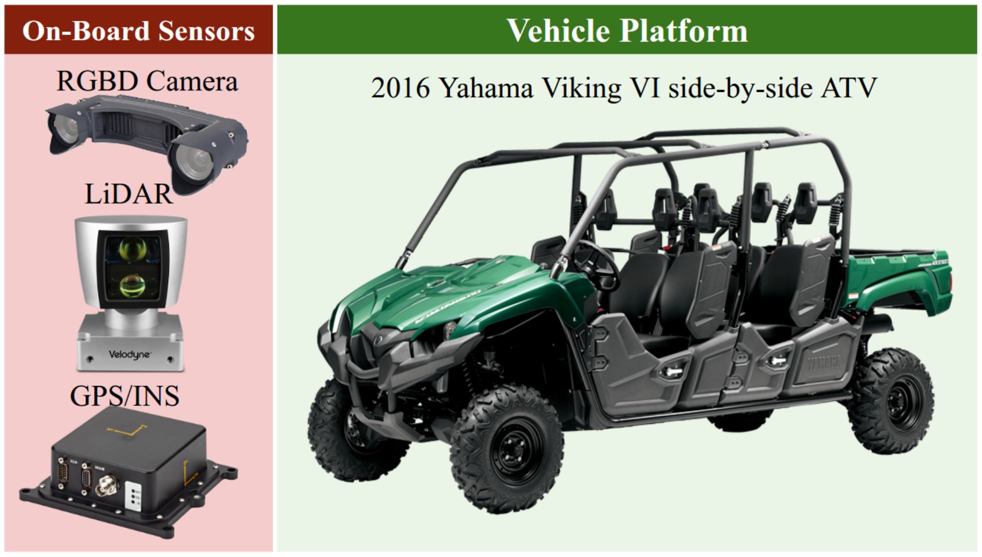


Figure 3.5: The testing vehicle platform and the on-board sensor.

3.2.2 Results

Experiments were conducted on field to show the feasibility of our approach. Three 3-meter-length static obstacles were placed on the sides of a rectangle shaped field with 10-meter width and 150-meter length. The vehicle successfully avoided all the obstacles at 20 kph. However, driving at higher speed (30kph) sometimes made collision map vulnerable to noise such as dust and sand blow up when the vehicle drove through, which highly affect the path quality outputted by planner. The example of planning path generation is visualized in Fig. . We have recorded some videos on-line: [1] [2]

3.2.3 Discussion

6. Conclusion and Future Work A RRT-based local planner for high-speed maneuvering is proposed for the application off-road navigation. Several methods are investigated for obstacle detection, with the final version implemented with height map algorithm. A simplified version of occupancy grid is built in global frame when vehicle is moving. Several modifications is implemented in order to obtain the minimal traveling-time trajectory, with a data-driven vehicle model is utilized for state propagation. Our planner can successfully avoid three obstacles on turnpike with vehicle velocity up to 30 kph. Future works include constructing a more complex model in 3D space, strengthening the collision map with more informative data such as mesh, or classification segmentation, and planner optimization.

[1]Simulation in OMPL.app: `http://ppt.cc/sBLAh`
[2]On-field testing with path visualization: `https://youtu.be/LibnO8_Sjm0`

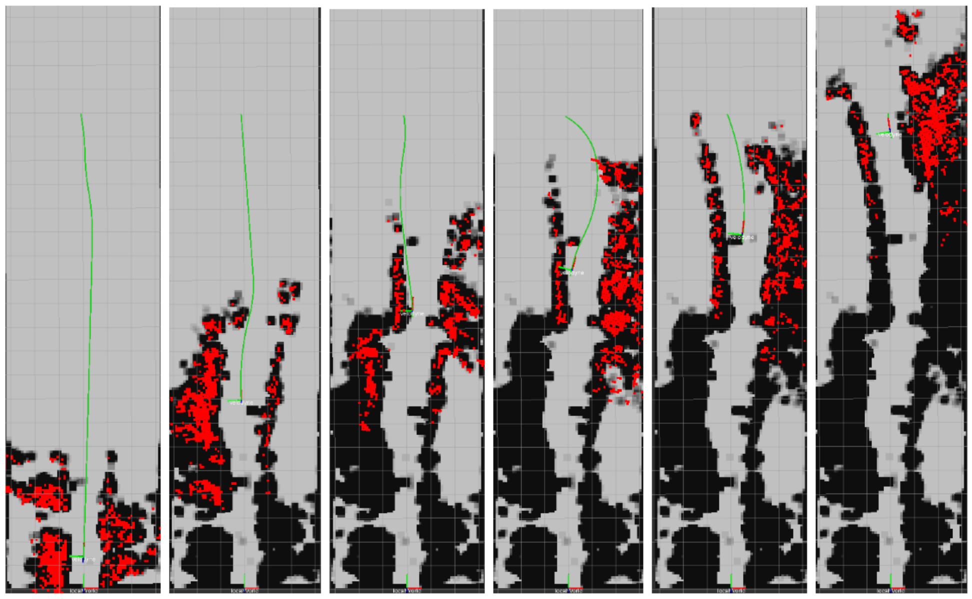


Figure 3.6: Screenshots of planner visualization output. The green path is the trajectory where each point is encoded with a 6 DoF state, 2 DoF control input, and a control duration. The red points are the filtered point cloud segmented as obstacles.

Chapter 4

Deep Inverse Reinforcement Learning

TODO

Chapter 5

Multimodal Deep Reinforcement Learning

In this chapter, we present an alternative end-to-end controller that maps the multi-sensor input directly to the action space based on deep reinforcement learning (DRL), a recently very popular research field. The autonomous navigation policy is trained and tested exhaustively to verify its performance in a physics-based gaming environment called TORCS [27].

While previous work in DRL predominantly learned policies based on a single input modality, i.e., either low-dimensional physical states, or high-dimensional pixels. For autonomous driving task where enhancing safety and accuracy to the maximum extent possible is emphasized, developing policies that operate with multiple inputs is crucial. To show the effectiveness of multi-modal perception, we pick two popular continuous action DRL algorithms namely Normalized Advantage Function (NAF) [28] and Deep Deterministic Policy Gradient (DDPG) [29], and augment them to accept multi-modal input.

Moreover, we also observe that while a multi-modal policy greatly improves the reward, it might rely heavily on all the inputs to the extent that it may fail completely even if a single sensor broke down fully or partially. This undesirable consequence renders sensor redundancy useless. To ensure that the learned policy does not succumb to such over-fitting, we apply a novel stochastic regularization method called *Sensor Dropout* during training.

The chapter is organized as follows. Section 5.1 summarizes the background of DRL and two DRL algorithms we extended in this work. Section 5.2 introduces multi-modal network architecture, and propose a new stochastic regularization technique called *Sensor Dropout*. The performance of Sensor Dropout is then validated in Section 5.3. In Section 5.4, we summarize our results and discuss key insights obtained through this exercise.

5.1 Background

5.1.1 Deep Reinforcement Learning (DRL)

We consider a standard Reinforcement Learning (RL) setup, where an agent operates in an environment E. At each discrete time step t, the agent observes a state $s_t \in \mathcal{S}$, picks an action $a_t \in \mathcal{A}$, and receives a scalar reward $r(s_t, a_t) \in \mathbb{R}$ from the environment. The return $R_t = \sum_{i=t}^{T} \gamma^{(i-t)} r(s_i, a_i)$ is defined as total discounted future reward at time step t, with γ being a discount factor $\in [0, 1]$. The objective of the agent is to learn a policy that eventually maximizes the expected return, as shown below:

$$J = \mathbb{E}_{s_i, r_i \sim E\ , a_i \sim \pi}[R_1] \tag{5.1}$$

The learned policy, π, can be formulated as either stochastic $\pi(a|s) = \mathbb{P}(a|s)$, or deterministic $a = \mu(s)$. The value function V^π and action-value function Q^π describe the expected return for each state and state-action pair upon following a policy π.

$$V^\pi(s_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E, a_{i \geq t} \sim \pi}[R_t | a_t, s_t] \tag{5.2}$$

$$\begin{aligned} Q^\pi(s_t, a_t) = \mathbb{E}_{r_{i \geq t}, s_{i>t} \sim E}[r(s_t, a_t) \\ + \gamma \mathbb{E}_{a_{i>t} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]] \end{aligned} \tag{5.3}$$

Finally, an advantage function $A^\pi(s_t, a_t)$ is defined as the additional reward or advantage that the agent will have for executing some action a_t at state s_t and its is given by $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$.

In high dimensional state/action space, these functions are usually approximated by a suitable parametrization. Accordingly, we define θ^Q, θ^V, θ^A, θ^π, and θ^μ as the parameters for approximating Q, V, A, π, and μ functions, respectively. It was generally believed that using non-linear function approximators for both Q and V functions would lead to unstable learning in practice. Recently, Mnih et al. [8] applied two novel modifications, namely *replay buffer* and *target network*, to stabilize the learning with deep nets. Later, several variants were introduced that exploited deep architectures and extended to learning tasks with continuous actions [28, 29, 30, 31].

To exhaustively analyze the effect of multi-sensor input and the new stochastic regularization technique, we picked two algorithms, namely DDPG and NAF. It is worth noting that the two algorithms are very different, with DDPG being an off-policy actor-critic method and NAF an off-policy value-based one. By augmenting these two algorithms, we highlight that any DRL algorithm, modified appropriately, can benefit from using multiple inputs. Before introducing the multi-modal architecture, we briefly summarize the two algorithms below.

5.1.2 Normalized Advantage Function (NAF)

Q-learning [32] is an off-policy model-free algorithm, where agent learns an approximated Q function, and follows a greedy policy $\mu(s) = \arg\max_a Q(s, a)$ at each step. The objective function (5.1) can be reached by minimizing the square loss Bellman error

$$L = \frac{1}{N} \sum_{i}^{N} (y_i - Q(s_i, a_i | \theta^Q))^2 \tag{5.4}$$

where target y_i is defined as $r(s_i, a_i) + \gamma Q(s_{i+1}, \mu(s_{i+1}))$.

Deep Q-Network(DQN) parametrized Q function with deep architecture[8], and has been shown to emulate human performance [9] in many Atari games using just image pixels as input.

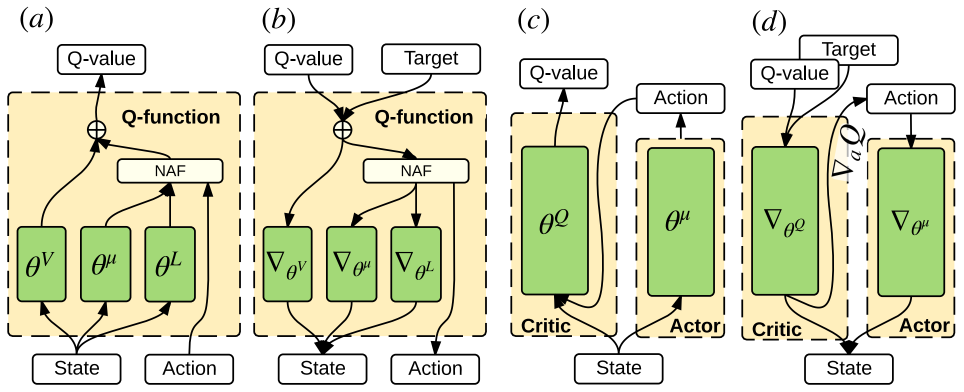


Figure 5.1: Schematic illustration of (a) forward and (b) back-propagation for NAF, and (c) forward and (d) back-propagation for DDPG. Green modules are functions approximated with Deep Nets.

However, in all of these games, action choices are limited and discrete. Recently, Gu et al. [28] proposed a continuous variant of Deep Q-Learning by a clever network construction. The Q network, which they called Normalized Advantage Function (NAF), parameterized the advantage function quadratically over the action space, and is weighted by non-linear feature of states.

$$Q(s, a|\theta^Q) = A(s, a|\theta^\mu, \theta^L) + V(s|\theta^V) \tag{5.5}$$

$$A(s, a|\theta^\mu, \theta^L) = -\frac{1}{2}(a - \mu(s|\theta^\mu))^T P(s|\theta^L) (a - \mu(s|\theta^\mu)) \tag{5.6}$$

$$P(s|\theta^L) = L(s|\theta^L)^T L(s|\theta^L) \tag{5.7}$$

During run-time, the greedy policy can be performed by simply taking the output of sub-network $a = \mu(s|\theta^\mu)$. The data flow at forward prediction and back-propagation steps are shown in Fig. 5.1 (a) and (b), respectively.

5.1.3 Deep Deterministic Policy Gradient (DDPG)

An alternative approach to continuous RL tasks was the use of an actor-critic framework, which maintains an explicit policy function, called *actor*, and an action-value function called as *critic*. In Silver et al. [33], a novel *deterministic* policy gradient (DPG) approach was proposed and it was shown that deterministic policy gradients have a model-free form and follow the gradient of the action-value function.

$$\nabla_{\theta^\mu} J = \mathbb{E}[\nabla_a Q(s, a|\theta^Q) \nabla_a \mu(s)] \tag{5.8}$$

Silver et al. [33] proved that using the policy gradient calculated in (5.8) to update model parameters leads to the maximum expected reward.

Building on this result, Lillicrap et al. [29] proposed an extension of DPG with deep architecture to generalize their prior success with discrete action spaces [9] onto continuous spaces. Using the DPG, an off-policy algorithm was developed to estimate the Q function using a differentiable function approximator. Similar techniques as in [9] were utilized for stable learning. In order to explore the full state and action space, an exploration policy was constructed by adding Ornstein-Uhlenbeck noise process [34]. In short, actions are chosen stochastically but a deterministic policy gradient is learned. The data flow for prediction and back-propagation steps are shown in Fig. 5.1 (c) and (d), respectively.

5.2 Proposed Methods

Multi-modal DRL aims to leverage the availability of multiple, potentially imperfect, sensor inputs to improve learned policy. Most autonomous driving vehicles have been equipped with an array of sensors like GPS, Lidar, Camera, and Odometer, etc [35]. While one would offer a long range noisy estimate, the other would offer a shorter range accurate one. When combined though, the resulting observer will have a good and reliable estimate of the environment. This problem is critical as a further step toward the real-world robotics application given the current state-of-the-art DRL agents on many realistic simulators.

5.2.1 Multi-modal Network Architecture

We denote a set of observations composed from M sensors as, $S = [S^{(1)}\ S^{(2)}\ ..\ S^{(M)}]^T$, where $S^{(i)}$ stands for observation from i^{th} sensor. In the multi-modal network, each sensory signal is pre-processed along independent paths. Each path has a feature extraction module with an appropriate network architecture, using randomly initialized or pre-trained weights. In this work, we use three different inputs namely image, laser scan and physical parameters (like wheel speed, position, odometry, etc. The details of each of the feature extraction module are listed in the Appendix. The modularized feature extraction stages for multiple inputs naturally allows for independent extraction of salient information that is transferable (with some tuning if needed) to other applications like collision avoidance, pedestrian detection and tracking, etc. The schematic illustration of modularized Multi-modal architecture is shown in Fig. 5.2. The outputs of feature extraction modules are eventually flattened and concatenated to form the multi-modal state.

5.2.2 Sensor-based Dropout (SD)

As shown in Fig.5.2, consider the multi-modal state $\tilde{S}$, obtained from feature extraction and given by $\tilde{S} = [\tilde{S}^{(1)}\ \tilde{S}^{(2)}\ ..\ \tilde{S}^{(M)}]^T$, where $\tilde{S}^{(i)} = [\tilde{X}_1^{(i)}\ \tilde{X}_2^{(i)}\ ..\ \tilde{X}_{K_i}^{(i)}]^T$. The objective is to generate a random mask $\mathbf{c} = [\delta_c^{(1)}\ \delta_c^{(2)}\ ..\ \delta_c^{(M)}]^T$, where $\delta_c^{(i)} \in \{0, 1\}$ represents the on/off indicator for the i^{th} modality. However, we need that at least one sensor remains *on* at any given instant in order to maintain training stability. Therefore, we cannot naively extend original dropout idea to sensor blocks $\tilde{S}^{(i)}$.

Let each block's on/off status be sampled from a Bernoulli distribution. We can generate 2^M random layer configurations masks and one of them is all the blocks switched off, i.e.,

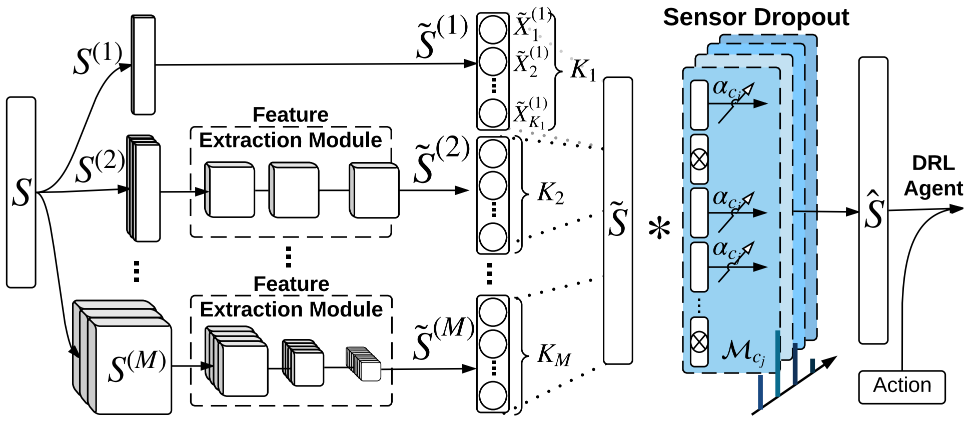


Figure 5.2: Illustration of Multi-modal Architecture and Sensor Dropout. The feature extraction module can be either pure identity function (modality 1), or convolution-based layer (modality $2 \rightarrow M$). The operation $*$ stands for element-wised multiplication.

$\mathbf{c_0} = [0^{(1)}\ 0^{(2)}\ ..\ 0^{(M)}]^T$. The probability of this event occurring decreases exponentially in the size of M, therefore not having much impact as formulated in Srivastava et al. [22], where its applied to individual nodes of a layer. However, when M is small as in the case of Sensor Dropout, the probability of $\mathbf{c_0}$ occurring is high enough to severely impact training performance. Therefore, we explicitly remove this rogue case and only consider $2^M - 1$ layer configuration masks. Another motivation for Sensor Dropout is to allow for each block $\tilde{S}^{(i)}$ to have a unique dropout probability $p^{(i)}$, if required, with the added cost of more hyper-parameter tuning. However, this tuning is tractable and meaningful in this setup.

Finally, for this work, we slightly depart from Srivastava et al. [22] in modeling our dropout layer. Instead of modeling Dropout as random process where any sensor block $\tilde{S}^{(i)}$ of the layer is switched on/off with a *fix* probability p, we propose an alternative view where the random variable is the layer configuration $\mathbf{c}$ itself and not individual block state $\tilde{S}^{(i)}$. Since there are $N = 2^M - 1$ possible states for $\mathbf{c}$, we accordingly sample from an N-state categorical distribution. We denote the probability of a layer configuration $\mathbf{c_j}$ occurring with p_j, where the subscript j ranges from 1 to N. It is easy to derive the corresponding pseudo-Bernoulli[1] distribution for switching on a sensor block $\tilde{S}^{(i)}$. Let us denote that as $p^{(i)}$.

$$p^{(i)} = \sum_{j=1}^{N} \delta_{c_j}^{(i)} p_j \tag{5.9}$$

Once the sensor dropout mask is applied, we have to deal with the issue of rescaling the non-zero weights. For this, we follow the same convention as in Srivastava et al. [22] but extend it to

[1] We wish to point out that $p^{(i)}$ is pseudo-Bernoulli as we restrict our attention to cases where at least one sensor block is switched on at any given instant in the layer. This implies that, while the switching on of any sensor block $\tilde{S}^{(i)}$ is independent of the other, switching off is not. So the distribution is no longer fully independent.

Algorithm 1 M-DRL with Sensor Dropout

```
Input: Initialize DRL Algorithm AGENT,
        State pre-processed network $f(\cdot|\theta)$,
        $N$ network configurations $\mathbb{C} = \{c_j\}_{j=1\,..\,N}$,
        $N$-state categorical distribution $\mathbb{P}(X = c_j) = p_j$,
Initialize experience replay buffers $\mathcal{B} = \{\mathcal{B}_j\}_{j=1\,..\,N}$
for $episode = 1$ to $Eps$ do
    Receive initial state $s_1$
    for $t = 1$ to $T$ do
        $\tilde{s}_t \leftarrow f(s_t|\theta)$
        $\hat{s}_t \leftarrow$ SENSORDROPOUT$(\tilde{s}_t, c_j)$, where $c_j \sim \mathbb{P}$
        $a_t \leftarrow$AGENT.ACT$(\hat{s}_t)$
        $r_t, s_{t+1}, terminate \leftarrow$ ENV.OBSERVE$(s_t, a_t)$
        $\mathcal{B}_j \leftarrow$ QUERY$(\mathcal{B}, c_j)$
        $\mathcal{B}_j$.APPEND$(s_t, a_t, r_t, s_{t+1})$
        AGENT.UPDATE$(\mathcal{B}_j)$
    end for
end for
```

sensor blocks as opposed to individual nodes.

$$\alpha_{c_j} = \frac{\sum_{i=1}^{M} K_i}{\sum_{i=1}^{M} \delta_{c_j}^{(i)} K_i}. \tag{5.10}$$

Note that the scaling ratio is different from the original Dropout [22], which preserved a *neural-wised* constant scaling ratio. Instead, we applied a configuration-dependent scaling which aims to maintain a balanced weighted summation over multiple sensors given different dimensions after feature extraction. It can be interpreted as an approximation of an equally weighted geometric mean of policies suggested by each combination made out of the sensor blocks.

Finally, the output of Sensor Dropout for the k^{th} node in i^{th} sensor block $\tilde{S}^{(i)}$ for some layer configuration $\mathbf{c_j}$ can be shown as,

$$\hat{S}_{c_j,k}^{(i)} = \mathcal{M}_{c_j}^{(i)} \tilde{X}_k^{(i)}, \qquad \text{where } \mathcal{M}_{c_j}^{(i)} = \alpha_{c_j} \delta_{c_j}^{(i)}. \tag{5.11}$$

$\mathcal{M}_{c_j}^{(k)}$ is an augmented mask encapsulating both dropout and re-scaling.

5.2.3 Augmenting Sensor Dropout with Experience Replay

Experience Replay is a commonly used technique in many off-policy DRL algorithms to break the time-based correlation in the agent's experience and thereby stabilize the training process. It differs from the standard deep supervised learning setting, in that batches are sampled from a memory buffer containing all the experience history. However, extending a dropout-like regularization technique to DRL setting to operate on the experience replay is non-trivial and can cause instability because the target values during training are usually non-stationary. In the following paragraph we investigate the instability issue of dropout-like regularization and argue that it is only combinatorially tractable if blocks are dropped out instead of individual nodes. The Sensor Dropout is not only principled but also well-motivated.

First, let us denote an instance of the experience as $[s_t, a_t, r_t, s_{t+1}]$, where $a_t = \mu(s_t, \tilde{\mathcal{M}_t})$ is generated from the policy network, with a random mask $\tilde{\mathcal{M}_t}$ sampled from an arbitrary distribution at the time step t. Note that under the dropout setting, the experience is now *mask configuration dependent*. During Bellman's updates, we calculate the one-step look-ahead target value with $Q^{'}(s_{t+1}, \mu^{'}(s_{t+1}, \tilde{\mathcal{M}}_{t+1}))$. If two masks, i.e. $\tilde{\mathcal{M}_t}$ and $\tilde{\mathcal{M}}_{t+1}$ are not the same, the behavior of μ and $\mu^{'}$ can be dramatircally different at least in the *beginning of training* as they use different sensor modalities as input and thus causing instability during training.

However, under the implementation of Sensor Dropout, this issue can be mitigated by simply maintaining multiple replay buffers for each of the sensor drop configuration $\mathbf{c_j}$. This is not possible with original Dropout as the number of replay buffers will grow exponentially with layer size. Sensor Dropout, on the other hand, makes this tractable as it only grows in the number of sensors. During batch updates, a specific replay buffer is queried given the current layer configuration $\mathbf{c_j}$, followed by the standard batch sampling and update network. This computational advantage makes the principle of sensor-based dropout more attractive and ideally suited in the DRL setting.

The overall Multimodal framework with Sensor Dropout regularization is summarized in Algorithm 1. It can be seen from the above discussion that Sensor Dropout approach is general enough to work with any existing off-policy DRL algorithm in a multi-modal setting. For on-policy algorithms however, Sensor Dropout can be implemented by fixing layer configuration $\mathbf{c_j}$ for every episode. In the next section, we validate the above-mentioned techniques in an open-source racing game called TORCS [27] and analyze their efficacy.

5.3 Evaluation and Analysis

In this section, we outline our experimental setup using TORCS simulator and then exhaustively compare the performance of the trained policies on both DDPG and NAF algorithms with and without Sensor Dropout. The exhaustive analysis includes policy robustness, sensitivity, and dependency on each sensor modality. Finally, we use a perturbation-based technique to visualize the learned policies and finally discuss some of the interesting findings of this exercise.

5.3.1 Platform Setup

TORCS Simulator:

The proposed approach is verified on TORCS [27], a popular open-source car racing simulator that is capable of simulating physically realistic vehicle dynamics as well as multiple sensing modalities [36] to build sophisticated AI agents. We limit our objective to only achieving autonomous navigation without any obstacles or other cars. This problem is kept simpler deliberately to focus our attention more on analyzing the performance of the proposed multi-modal

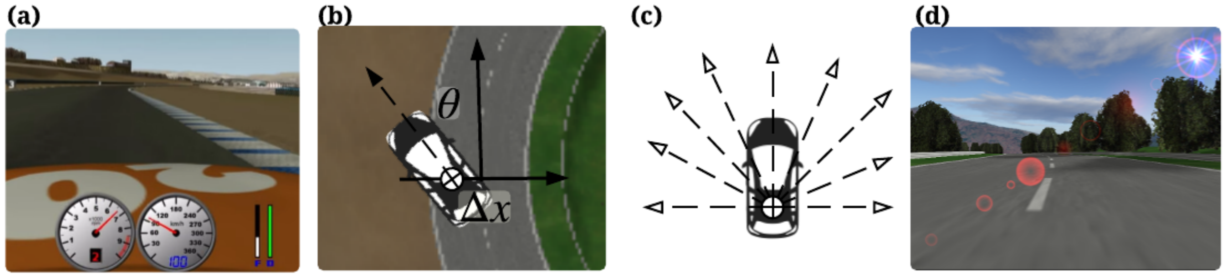


Figure 5.3: Sensors used in the TORCS racing car simulator: *Sensor 1:* Physical information such as velocity (a), position, and orientation (b), *Sensor 2:* Laser range finder (c), and *Sensor 3:* Front-view camera (d). Sensor dimensionality details listed in Sec. 5.3.1.

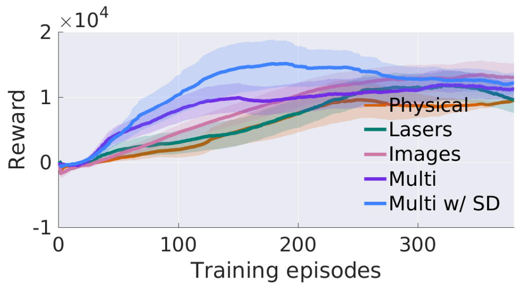
configurations using extensive quantitative and qualitative testing. We believe however that it is representative of the larger urban navigation problem, and we ensure that our proposed network architecture is task-agnostic and transferable.

In order to make the learning problem representative of the real-world setting, we picked the following sensors from the TORCS package: the 2D laser range finder, front-view camera with RGB channel, vehicle state - position and speed. The action space is a continuous vector in $\mathbb{R}^3$, whose elements represent acceleration, steering angle, and braking, respectively. Moreover, all the actions are bounded and for this learning problem they are also normalized. An exploration strategy is injected adding an Ornstein-Uhlenbeck process noise [34] to the output of the policy network. The choice of reward function is slightly different from Lillicrap et al. [29] and Mnih et al. [30] as an additional penalty term to penalize side-ways drifting along the track was added. In practice, this modification leads to more stable policies during training [37].

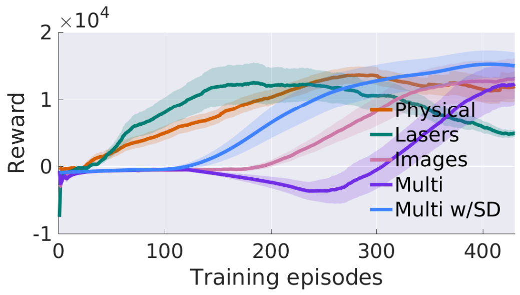
Most of the related works use either low-dimensional physical state such as joint angles, or high-dimensional pixels as input. Here, we use the following sensing modalities for our state description: (1) We define *Sensor 1* as a 10 DOF hybrid state containing all physical information, including 3D velocity (3 DOF), position and orientation with respect to track center-line (2 DOF), and finally rotational speed of 4 wheels (4 DOF) and engine (1 DOF). (2) *Sensor 2* consists of 4 consecutive laser scans (i.e., at time t, we input scans from times $t,\ t-1,\ t-2\ \&\ t-3$). Here, each laser scan is composed of 19 readings spanning a $180°$ field-of-view in the the front of car. Finally, as *Sensor 3*, we supply 4 consecutive color images capturing the car's front-view and each one has a resolution 64×64. These three representations are used separately to develop our baseline single sensor based policies.

Multi-modal State and Sensor Dropout Implementation:

The multi-modal state on the other hand has access to all sensors at any given time step. When Sensor Dropout (SD) is applied, agent will randomly lose access to a strict subset of sensors. During training, the agent will observe the sensor data from one of the following three combinations: (1)$(physical\ state,\ laser)$, (2) $(image)$, or (3) $(physical\ state,\ laser,\ image)$. Note that we report on a the smaller subset in order to better analyze the effects of SD. A more detailed



(a) NAF



(b) DDPG

Figure 5.4: Training performance comparison of three baseline single sensor policies, and the proposed multi-modal policies, with and without Sensor Dropout.

commentary on sensor-fusion configurations and the results of using all layer configurations follow in Section 5.4.2, with network details listed in the Appendix.

We have experimented with following three different settings:(a) $[p_1 \; p_2] = [0.25 \; 0.25]$, (b) $[p_1 \; p_2] = [0.4 \; 0.4]$, or (c) $[p_1 \; p_2] = [0.5 \; 0.5]$. Here, p_1 and p_2 represent the probability of using the first and second combinations, respectively, while the probability of the third combination p_3 is given by $p_3 = 1 - p_1 - p_2$. For the first two configurations, the agent has access to all sensors occasionally, but in the last case, the agent only has access to a subset of sensors at any given point. The best learned policy among three configurations is reported in the policy analysis.

5.3.2 Experimental Results

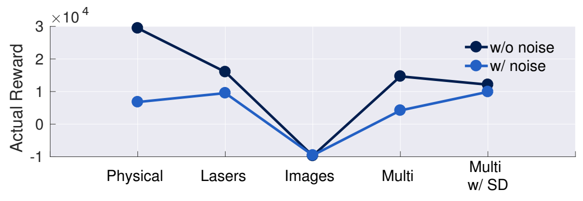
Training Summary:

The training performance, for all the proposed models and their corresponding baselines, is shown in Fig. 5.4. As expected, using high dimensional sensory input directly impacts convergence rate of the policy. However, despite the curse of dimensionality, the convergence rate improves with the addition of Sensor Dropout, and quite drastically so for DDPG (see Fig. 5.4b).

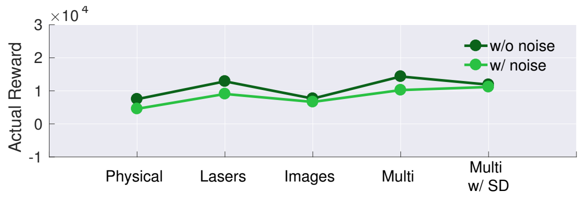
To assess the degree of generalization of the learned policies, we pick the the best learned policies for each model and test its performance on other racing tracks. As summarized in Fig. 5.5, we find that the DDPG agent is capable of finding a reasonable policy regardless of the representation of states. The performance of NAF on the other hand is greatly affected by the dimensionality of the agent's input. In fact, the agent fails to achieve even a non-zero reward during testing, when trained with image as input. However, using multi-sensor information and thereby reducing the fused input dimension aids in learning a good policy. Network architecture details are described in Section 5.4.3.

Policy Robustness Analysis:

Note that, we assume perfect sensing during the training. However, to test performance in a more realistic scenario, we simulate mildly imperfect sensing by adding gaussian noise. The perturbation sensitivity is compatible with a more standard Average Fisher Sensitivity (AFS)

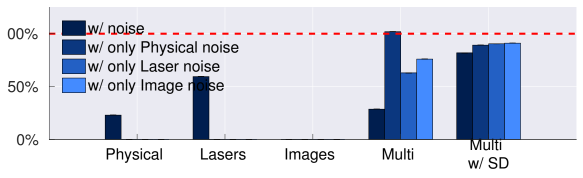


(a) NAF

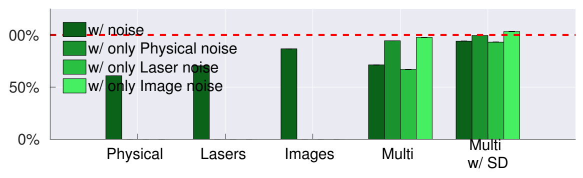


(b) DDPG

Figure 5.5: Policy Robustness Analysis: Darker lines connects average rewards of leaned policies with accurate sensing while the lighter lines connects the corresponding policies in the face of sensor noise.



(a) NAF



(b) DDPG

Figure 5.6: Policy Robustness Analysis: The bar box measures the relative scale among each of the models when noise is introduced. The red dotted lines show the performance without noise.

according to [38]. Policy performance with and without noise is plotted for comparison in Fig. 5.5 and 5.6. While Fig. 5.5 plots the actual reward performance, Fig. 5.6 summarizes the relative performance compared with a noiseless environment.

The performance of the NAF agent drops dramatically when the noise is introduced. We also observe that NAF in the multi-modal is sensitive to states from sensors which are easily interpretable such as laser scanners. This effect shows that using an over-complete state representation holds a risk of the agent learning an undesired policy where the influence of different features gets unbalanced. The regularization introduced by Sensor Dropout alleviates this issue and learns a stable policy on both algorithms, with only slight decrease of the performance compared with multi-modal agents trained without SD. In summary, with the addition of noise the performance drop is sometimes severe in a single input policy, as seen for NAF with physical state input. In comparison, the drop is ore contained for the multi-modal policy and almost negligible when Sensor Dropout is used.

Policy Sensitivity Analysis:

In this part, we further validate our hypothesis Sensor Dropout (SD) reduces the learned policy's acute dependence on a subset of sensors in a multi-modal setting. First, we considered a scenario when malfunction of a sensor has been detected by the system, and the agent need to rely on the remaining sensors to make the decision. During testing, we manually blocked off part of the sensor modules, and scaled the rest of observation using the same rescaling mechanism as proposed in Section 5.2.2. Fig. 5.7 reports the average of the normalized reward, i.e. total reward divided by total steps, of each model. For both the algorithms, models trained with SD

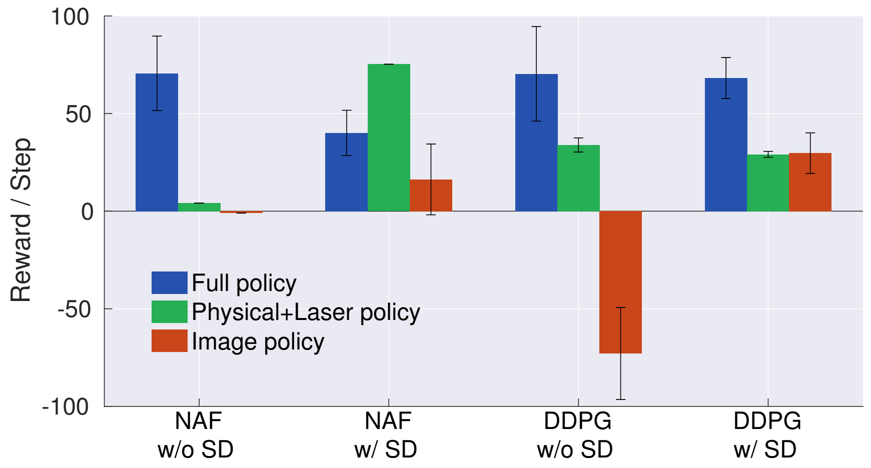


Figure 5.7: Policy Sensitivity to Sensor Failure: The blue, green, and red bars denote full multi-modal policy, multi-modal with no image input, and multi-modal with no laser and physical state input, respectively.

Table 5.1: Results of the $\mathcal{T}_2^1$ dependency metric.

		Training Env. Mean	Training Env. Median	Testing Env. Mean	Testing Env. Median
NAF	w/o SD	1.651	1.871	1.722	1.940
	w/ SD	**1.284**	**1.379**	**1.086**	**1.112**
DDPG	w/o SD	1.458	1.449	1.468	1.437
	w/ SD	**1.168**	**1.072**	**1.171**	**1.120**

performed comparatively better when decisions were forced to depend on a subset of sensors than the ones trained without SD. We observed that the policy could either be highly correlated, which is the case of NAF w/o SD in Fig. 5.7, or depend on sensor subset, as shown in DDPG w/o SD in Fig. 5.7. We therefore emphasize that introducing Sensor Dropout during training has significant practical benefits, in that the policies induced by different configurations of sensors are learned at the same time.

Policy Dependency Analysis:

To further examine the impact of Sensor Dropout on effective sensor fusion, we monitor the extent to which the learned policy depends on each sensor block by measuring the gradient of the policy output w.r.t a subset block $\tilde{S}^{(i)}$. In a multi-input-to-output mapping, this denotes the relative weights/importance of each input.

Since in this work, SD was implemented to drop either (1) $(physical\ state,\ laser)$ or (2) $vision$, we define the *dependency* metric for this problem as the ratio between configurations (1)

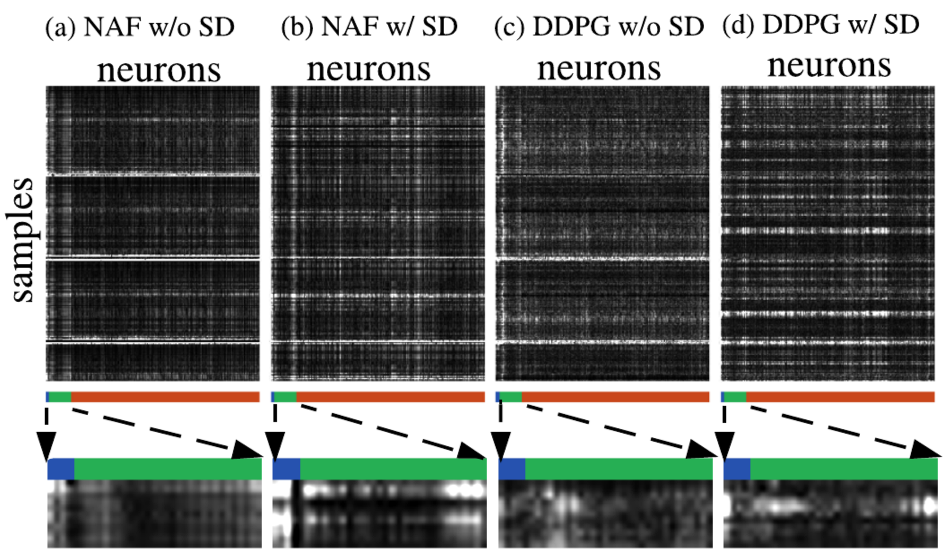


Figure 5.8: The visualization of the magnitude of gradient for each neuron on training environment. The whiter color means the higher gradient. The color bar represents three different sensor modules: physical state(blue), Laser(green), and Image(red).

and (2), as shown below.

$$\mathcal{T}_2^1 = \frac{1}{M}\sum_{i=1}^{M} \frac{\left| \nabla_{\tilde{S}_i^{(1)}} \mu(\tilde{S}|\theta^\mu)\Big|_{S_i} \right|}{\left| \nabla_{\tilde{S}_i^{(2)}} \mu(\tilde{S}|\theta^\mu)\Big|_{S_i} \right|} \tag{5.12}$$

Assuming the fusion-of-interest is between the above-mentioned two subsets, we intend to show that, using SD, the metric should get closer to one, indicating nearly equal importance to both the sensing modalities. This metric is evaluated for policies learned with and without SD for NAF and DDPG and the mean values are reported in Table 5.1. For this, the data was collected from a stable policy by evaluating it on both training and testing environments. We can verify that, using SD, NAF policies become dramatically more dependent all sensors. Additionally, we visualize the dependence on each neuron by estimating the average gradient. As shown in Fig. 5.8, models trained with SD tend to make better use of visual information (location is indicated by the red bar.). Without SD however, agents tend to rely more heavily on physical state and laser inputs.

5.3.3 Visualizing Policy

Visualizing weights:

The Eq. (5.12) can also represent the salient regions where the learned policy pays attention. The visualization is motivated from the salient map analysis [39], which has also been applied

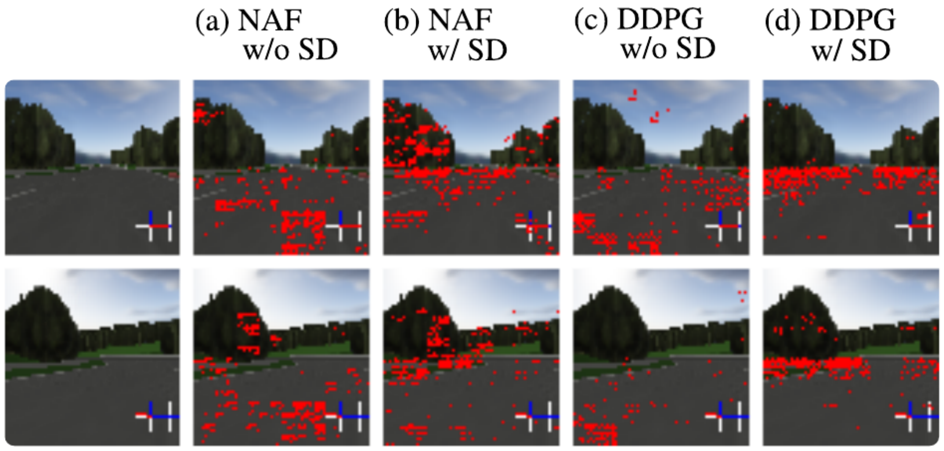


Figure 5.9: The gradient responses of actions on the image input for each of the multi-modal agents. The top 20% gradients are marked red.

to DRL study recently [40]. As shown in Fig. 5.8, we observe that models trained with SD have higher gradients on neurons corresponding to the corner inputs of the laser sensor, indicating that a more sparse and meaningful policy is learned. These corner inputs corresponded to the laser beams that are oriented perpendicularly to the vehicle's direction of motion, and give an estimate of its relative position on the track. Clearly, the network is able to automatically infer and weight locations providing salient information. This behavior is consistent in both DDPG and NAF.

To look for similar patterns, in Fig. 5.9, image pixels with higher gradients are marked to visualize and interpret the policy's view of the world. We pick two scenarios, 1) straight track and 2) sharp left turn, depicted by the first and second rows in the figure. Models trained with SD tend to capture relevant visual information such as road boundary. In comparisons, models trained without SD have a relatively low and unclear gradients over both laser and image sensor state space.

5.4 Discussion

5.4.1 Sensor Dropout v/s *standard* Dropout

In addition to demonstrating the value of using Sensor Dropout, it also critical to verify whether similar performance can be extracted with Dropout or not. Therefore, we implemented the Dropout on our multi-modal agent with dropping probabilities of 0.5 and 0.25 as baselines. Both models performed poorly in the testing environment, even with perfect sensing. For DDPG, mean reward using dropout was only 2.2×10^3, compared with 1.1×10^4 and 1.4×10^4, obtained on vanilla-multi-modal policy and multi-modal + SD, respectively. Similar results were observed with NAF too. This furthers our claim that traditional implementation of dropout increases the risk of destabilizing the DRL policy.

5.4.2 Fusion Layer Configurations

Following the notations used in Section 5.3.1, we observe that as the probabilities $p_1,\ p_2$ increase, the policy convergence rate and the normalized average rewards induced by each sensor subset also increase, as described in Section 5.3.2. However, the performance of the full policy decreases in both training and testing environment. This empirical result suggests a more principled way to apply Sensor Dropout. For instance, we can start with a high value for $(p_1,\ p_2)$ and benefit from faster policy convergence for each sensor subset and then gradually decrease the probability to promote fusion.

add NAF Using all layer configurations in DDPG gives good results. The best performing sensor subset is Laser+Image (reward of 61 compared in Fig 5.7).

5.4.3 Implementation of NAF with Multi-modal

Since NAF is not designed for high-dimensional state space, unstable policies result if extended naively to operate with image as input. To mitigate this problem, we add two additional fully-connected layers to reduce the state dimensionality to apply the NAF algorithm. We consider this slight modification as a useful and practical method to embed a low-dimensional representation of visual information. However, purely image-based policy learned with the NAF agent still results in an unstable behavior with large negative rewards *during testing*. This could be a case of over-fitting where the experience from training environment does not generalize well to fully operate in the state space, and the agent may have just memorized specific but irrelevant visual cues. Surprisingly, the multi-modal representation alleviated the issue by fusing visual state with other sensory information.

5.4.4 Policy Profile

5.4.5 Adding Auxiliary Task for Policy Network

5.5 Conclusions

In this work, we extend popular DRL algorithms like DDPG and NAF to the multi-modal setting. Additionally, we introduce a new stochastic regularization technique called Sensor Dropout to promote an effective fusing of information from multiple sensors.

Through extensive empirical testing we show the following exciting results,

1. Multimodal-DRL with Sensor Dropout(SD) reduces performance drop in a noisy environment from $\approx 30\%$ to just 5%, when compared to a baseline single sensor system.
2. Policies learned using SD best leverage the multi-modal setting by greatly reducing over-dependence on any one sensing modality. Additionally, for each sensor it was observed that SD enforces sparsity and promotes each sensor to base the policy primarily on intuitive and salient features.
3. A multi-modal policy with SD guarantees functionality even in a face a sensor failure. This is a huge plus and the best use-case for the need for redundancy in safety-critical

application like autonomous navigation.

Bibliography

[1] William J Mitchell, Chris E Borroni-Bird, and Lawrence D Burns. *Reinventing the automobile: Personal urban mobility for the 21st century*. 2010. 1

[2] Chris Urmson, Joshua Anhalt, Drew Bagnell, Christopher Baker, Robert Bittner, MN Clark, John Dolan, Dave Duggins, Tugrul Galatali, Chris Geyer, et al. Autonomous driving in urban environments: Boss and the urban challenge. *Journal of Field Robotics*, 25(8):425–466, 2008. 1

[3] Chris Urmson, J Andrew Bagnell, Christopher R Baker, Martial Hebert, Alonzo Kelly, Raj Rajkumar, Paul E Rybski, Sebastian Scherer, Reid Simmons, Sanjiv Singh, et al. Tartan racing: A multi-modal approach to the darpa urban challenge. 2007. 1, 1.2

[4] Walther Wachenfeld, Hermann Winner, J Chris Gerdes, Barbara Lenz, Markus Maurer, Sven Beiker, Eva Fraedrich, and Thomas Winkle. Use cases for autonomous driving. In *Autonomous Driving*, pages 9–37. Springer, 2016. 1

[5] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015. 1.2

[6] 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. 1.2

[7] Huazhe Xu, Yang Gao, Fisher Yu, and Trevor Darrell. End-to-end learning of driving models from large-scale video datasets. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 1.2

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS'13 Workshop on Deep Learning*, 2013. 1.2, 5.1.1, 5.1.2

[9] 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):529–533, 2015. 1.2, 5.1.2, 5.1.3

[10] Xue Bin Peng, Glen Berseth, and Michiel van de Panne. Terrain-adaptive locomotion skills using deep reinforcement learning. *ACM Transactions on Graphics (Proc. SIGGRAPH 2016)*, 35(4), 2016. 1.2

[11] Stéphane Ross, Geoffrey J Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, volume 1, page 6, 2011. 1.2

[12] Jos Elfring, Rein Appeldoorn, and Maurice Kwakkernaat. Multisensor simultaneous vehicle tracking and shape estimation. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 630–635. IEEE, 2016. 1.2

[13] Hyunggi Cho, Young-Woo Seo, BVK Vijaya Kumar, and Ragunathan Raj Rajkumar. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1836–1843. IEEE, 2014. 1.2

[14] Michael Darms, Paul Rybski, and Chris Urmson. Classification and tracking of dynamic objects with multiple sensors for autonomous driving in urban environments. In *Intelligent Vehicles Symposium, 2008 IEEE*, pages 1197–1202. IEEE, 2008. 1.2

[15] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011. 1.2

[16] Natalia Neverova, Christian Wolf, Graham Taylor, and Florian Nebout. Moddrop: adaptive multi-modal gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(8):1692–1706, 2016. 1.2, 2

[17] Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. Multimodal neural language models. In *international Conference on Machine Learning (ICML)*, volume 14, pages 595–603, 2014. 1.2

[18] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012. 1.2

[19] Ahmed Hussain Qureshi, Yutaka Nakamura, Yuichiro Yoshikawa, and Hiroshi Ishiguro. Robot gains social intelligence through multimodal deep reinforcement learning. In *Humanoid Robots (Humanoids), 2016 IEEE-RAS 16th International Conference on*, pages 745–751. IEEE, 2016. 1.2

[20] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016. 1.2

[21] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andy Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. In *International Conference on Learning Representations (ICLR)*, 2017. 1.2

[22] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 2, 5.2.2, 5.2.2, 5.2.2

[23] Calvin Murdock, Zhen Li, Howard Zhou, and Tom Duerig. Blockout: Dynamic model selection for hierarchical deep networks. In *Proceedings of the IEEE Conference on Com-*

puter Vision and Pattern Recognition, pages 2583–2591, 2016. 2

[24] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 1058–1066, 2013. 2

[25] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. Zoneout: Regularizing rnns by randomly preserving hidden activations. *arXiv preprint arXiv:1606.01305*, 2016. 2

[26] Fan Li, Natalia Neverova, Christian Wolf, and Graham Taylor. Modout: Learning to fuse modalities via stochastic regularization. *Journal of Computational Vision and Imaging Systems*, 2(1), 2016. 2

[27] Bernhard Wymann, E Espié, C Guionneau, C Dimitrakakis, R Coulom, and A Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 2000. 5, 5.2.3, 5.3.1

[28] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 2829–2838, 2016. 5, 5.1.1, 5.1.2

[29] 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. In *International Conference on Learning Representations (ICLR)*, 2016. 5, 5.1.1, 5.1.3, 5.3.1

[30] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy P Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, 2016. 5.1.1, 5.3.1

[31] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015. 5.1.1

[32] Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pages 1057–1063, 1999. 5.1.2

[33] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 5.1.3, 5.1.3

[34] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930. 5.1.3, 5.3.1

[35] Ratan Hudda, Clint Kelly, Garrett Long, Jun Luo, Atul Pandit, Dave Phillips, Lubab Sheet, and Ikhlaq Sidhu. Self driving cars. 2013. 5.2

[36] Naoto Yoshida. Gym-torcs. `https://github.com/ugo-nama-kun/gym_torcs`, 2016. 5.3.1

[37] Yan-Pan Lau. Using keras and deep deterministic policy gradient to play torcs. `https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html`, 2016. 5.3.1

[38] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 5.3.2

[39] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. 2015. 5.3.3

[40] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33th International Conference on Machine Learning (ICML-16)*, 2016. 5.3.3

[41] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.

[42] Marco Pavone. Autonomous mobility-on-demand systems for future urban mobility. In *Autonomous Driving*, pages 387–404. Springer, 2016.

[43] Tomasz Janasz and Uwe Schneidewind. The future of automobility. In *Shaping the Digital Enterprise*, pages 253–285. Springer, 2017.

[44] Kevin Spieser, Kyle Treleaven, Rick Zhang, Emilio Frazzoli, Daniel Morton, and Marco Pavone. Toward a systematic approach to the design and evaluation of automated mobility-on-demand systems: A case study in singapore. In *Road Vehicle Automation*, pages 229–245. Springer, 2014.

[45] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[46] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning (ICML)*, pages 1–9, 2013.

[47] Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. *arXiv preprint arXiv:1609.07152*, 2016.

[48] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[49] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[50] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[51] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Deep compositional question answering with neural module networks. *CoRR*, abs/1511.02799, 2015. URL `http://arxiv.org/abs/1511.02799`.

[52] Andrei A Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint*

arXiv:1610.04286, 2016.

[53] Richard Liaw, Sanjay Krishnan, Animesh Garg, Daniel Crankshaw, Joseph E Gonzalez, and Ken Goldberg. Composing meta-policies for autonomous driving using hierarchical deep reinforcement learning.

[54] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[55] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

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

[57] Udacity. Udacity's self-driving car simulator. `https://github.com/udacity/self-driving-car-sim`, 2017.

[58] Matthias Plappert. keras-rl. `https://github.com/matthiasplappert/keras-rl`, 2016.

[59] Fereshteh Sadeghi and Sergey Levine. (cad)2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

[60] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.

[61] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[62] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[63] François Chollet. Keras (2015). *URL http://keras. io*.

[64] Craig Quiter. Deepdrive: self-driving car ai, 2016. URL `http://deepdrive.io/`.

[65] P. Langley. Crafting papers on machine learning. In Pat Langley, editor, *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 1207–1216, Stanford, CA, 2000. Morgan Kaufmann.

[66] Shixiang Gu, Timothy P. Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *International Conference on Learning Representations (ICLR)*, 2017.

[67] Eder Santana and George Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.

[68] Allen Nie. Stochastic dropout: Activation-level dropout to learn better neural language models.

[69] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2, 2015.