

Some fancy title

Guan-Horng Liu

June 8, 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

The author would like to thank ...

Contents

1	Introduction	1
1.1	Motivation	1
1.1.1	Traditional Motion Planner	1
1.1.2	Learning End-to-end Controller	2
1.2	Approaches	4
2	Related Work	5
2.1	Sample Based Planning	5
2.2	Deep Reinforcement Learning (DRL)	6
2.2.1	Continuous Action Space Algorithms	7
2.3	Stochastic regularization	9
2.4	Deep Inverse Reinforcement Learning (DIRL)	9
3	Model Predictive Planning	13
3.1	Vehicle Response Model	13
3.2	Planner Design	15
3.2.1	Collision Check Module	15
3.2.2	Sample-based Planner Module	17
3.3	Experiments and Analysis	19
3.3.1	Vehicle Model Verification	20
3.3.2	Planner Demo	20
4	Learning End-to-end Multimodal Sensor Policy	23
4.1	Proposed Methods	24
4.1.1	Multimodal Network Architecture	24
4.1.2	Sensor-based Dropout (SD)	24
4.1.3	Augmenting Auxiliary Loss	26
4.2	Evaluation and Analysis	26
4.2.1	Platform Setup	26
4.2.2	Experimental Results	28
4.3	Discussion	32
4.3.1	Full Sub-Policy Analysis	32
4.3.2	Visualize Policy Attention Region	33
4.4	Supplementary Material	34

5 Learning from Demonstration using DIRL	35
5.1 Proposed Methods	35
5.1.1 Learning from Failure for DIRL	35
5.1.2 Modeling Optimality Ambiguity	36
5.2 Experiments Results	38
6 Conclusion and Future Work	41
Bibliography	43

List of Figures

1.1	Some of the well-known autonomous navigation systems: (a) The Carnegie Mellon Tartan that won the Urban Challenge in 2007 [1]. (b) Uber’s self-driving vehicle. (c) Crusher. [2]	1
1.2	Demonstrations of the end-to-end approach on autonomous navigation applied to (a) the realistic simulator (GTA V), and (b) the full-size vehicle by NVIDIA[3].	2
1.3	Recent successes on applying deep reinforcement learning (DRL) algorithms to real-world robotics problems: (a) manipulation [4]. (b) autonomous navigation. [5]	3
2.1	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.	8
2.2	The block diagram of reinforcement learning and inverse reinforcement learning.	9
2.3	Schematic illustration of deep inverse reinforcement learning.	10
3.1	(a) Notation of vehicle velocity in local frame. (b) Data flow of conventional dynamic response model. The model takes two velocity control commands as input and estimates the velocity response in vehicle frame. (c) Schematic illustration of neural-network based response model from [6].	14
3.2	Block Diagram of our planner API.	15
3.3	Three different methods of collision check. Note that for (b), the original and the processed point cloud is represented with white and colored dot, respectively.	16
3.4	Simulation of RRT planner with vehicle model derived in Sec. 3.1	17
3.5	The visualization of RRT-based planner.	18
3.6	The testing vehicle platform and the on-board sensor.	20
3.7	Simulated velocity response w.r.t. time steps. Note that the blue line is measured under an accurate GPS/INS sensor with RTK signal. We refer this measurement as our ground true data.	21
3.8	Comparison between data-driven dynamic model and original kinematic model. Note the time interval in each graph is 20 seconds.	22
3.9	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.	22

4.1	Illustration of multimodal sensor policy network augmented with Sensor Dropout. The operation $*$ stands for element-wised multiplication. The dropping configuration of Sensor Dropout is sampled from a categorical distribution, which the network takes as an additional input. 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.	25
4.2	Sensors used in the TORCS racing car simulator: <i>Sensor 1</i> : Physical information such as velocity (a), position, and orientation (b), <i>Sensor 2</i> : Laser range finder (c), and <i>Sensor 3</i> : Front-view camera (d).	27
4.3	Training performance comparison of three baseline single sensor policies, and the proposed multimodal policies, with and without Sensor Dropout.	28
4.4	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.	29
4.5	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.	29
4.6	Policy performance when facing random sensor failure.	30
4.7	Two-dimensional PCA embedding of the representations in the last hidden layer assigned by the policy networks.	31
4.8	The variance of all the actions induced by sub-policy under each multimodal sensor policy. <i>Upper-left</i> : naive policy without any regularization. <i>Upper-right</i> : with standard Dropout. <i>Lower-left</i> : with Sensor Dropout. <i>Lower-right</i> : with Sensor Dropout and auxiliary loss.	32
4.9	The full analysis of the performance of the total 6 sub-policies. The (1), (2), and (3) labels in y-axis represent physical state, laser, and image, respectively. The x-axis represent the remaining performance w.r.t. the SD policy with all sensor, i.e. (1)+(2)+(3).	33
4.10	(a)The visualization of the magnitude of gradient for each neuron. The whiter color means the higher gradient. The color bar represents three different sensor modules: physical state(blue), Laser(green), and Image(red). (b) The gradient responses of actions on the image input for each of the multimodal agents. The top 20% gradients are marked red.	33
5.1	Block diagram of deep inverse reinforcement learning.	36
5.2	The extracted feature maps from LiDAR. The flow chart is shown on the left of the figure, while examples of the actual collected data on filed are shown on the right of the figure.	37
5.3	Training curve of DIRL in Gascola dataset.	37
5.4	Samples of intermediate cost maps during training.	38
5.5	Final cost map on testing set. The left, middle, and right image represent the output cost map, the same cost map overlapped with expected SVF, and with demonstrations, respectively.	39

List of Tables

3.1	Simulation Benchmark	17
3.2	Planner Hyper-parameter	19
4.1	Model Specification	28
4.2	Performance of Policy	30
4.3	Results of the sensitivity metric.	30
4.4	Covariance of the first three Principle Component	34
4.5	Action Variation w.r.t. multimodal sensor	34

Chapter 1

Introduction

1.1 Motivation

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 [7]. 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 [1, 8]. By offering an opportunity to develop sustainable and safe solutions to personal mobility [9], they also hint at a complete overhaul of the urban transportation landscape by ushering in Autonomous Vehicles-on-Demand. While there is a foreseeable trend on operating autonomous driving technologies on-road within decade, extreme/aggressive motion planning in the off-road situations or unstructured terrains has gained great interests recently [10, 11, 12, 13, 14].

1.1.1 Traditional Motion Planner

The two most common approaches for ground vehicles motion planning are graph based planner and sample based planner. Both approaches have shown their robustnesses in the DRAPA Urban



Figure 1.1: Some of the well-known autonomous navigation systems: (a) The Carnegie Mellon Tartan that won the Urban Challenge in 2007 [1]. (b) Uber's self-driving vehicle. (c) Crusher. [2]

Challenge [15, 16] and UPI program [2, 17]. The feasible trajectory can be solved efficiently under certain reasonable assumptions, then fine-tuned with optimization techniques to ensure criteria such as smoothness and kinematic constraint [18].

Though graph based planner has shown a great development on off-road navigation system [2, 17], it is limited to the low dimensional configuration space. In fact, one of the main challenges for off-road motion planning comes from the fact that the vehicle dynamics in off-road environment is much more unpredictable in contrast to the on-road conditions. Factors such as wheel-terrain interaction for modeling the sliding effect is still an active research area [19, 20]. Maneuvering agilely and aggressively on the rough terrain requires a much complex vehicle model, which inevitably leads to a higher dimensional configuration space, and thus slows down the standard graph based approaches.

Sample based planner on the other hand is more efficient in solving higher-dimensional motion planning. Built upon the successful applications on many robotics problems such as manipulations [21], researchers have tried to extend the approach to an aggressive motion planner such as drifting [22]. However, unlike graph based planner that guarantees a certain level of optimality, sample based planner in general only permits asymptotic convergence to the optimal solution. Moreover, it is not straight-forward to implement in kinodynamic problems.

Despite the pros and cons of each approach, they both share a common frameworks where a intelligent pipeline is built to efficiently gather incoming sensor data and take suitable control actions with good repeatability and fault-tolerance. The resulting autonomous navigation system is addressed in a modular fashion, where specialized algorithms were developed for each subsystem and finally integrated with some fine tuning.

1.1.2 Learning End-to-end Controller

More recently, however, there is a great interest in applying an end-to-end approach 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. This approach is purported to perform better since we are optimizing the whole system. The end-to-end approaches have become more appealing with the use of deep learning in robotics and has shown successes in developing vi-suomotor policies for autonomous driving [3, 23, 24]. However, the traditional deep supervised



Figure 1.2: Demonstrations of the end-to-end approach on autonomous navigation applied to (a) the realistic simulator (GTA V), and (b) the full-size vehicle by NVIDIA[3].

learning-based driving requires a great deal of human annotation and may not be able to deal with the problem of accumulating errors [25].

On the other hand, deep reinforcement learning (DRL) offers a better formulation that allows policy improvement with feedback, and has been shown to achieve human-level performance on several video games [26, 27, 28]. The choice of DRL over the more tried and tested waters of Supervised Learning comes from the interest of better generalization and allowing for a more exploration-friendly setting. Recently, there have been a great demands and improvements on realistic simulator [29, 30] with the hope that the acquired learning in such a simulator can be transferable to the real world with minimal tuning [31].

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 where enhancing safety and accuracy to the maximum possible extent is top priority, developing policies that operate with multiple inputs is the need of the hour. In this light, several recent works in DRL have tried to solve the complex robotics tasks such as human-robot-interaction [32], manipulation [33] and maze navigation [34] with multimodal inputs. However, none of the works have explicitly formulated the multimodal sensor policy in term of sensor fusion, a indispensable technique that improve accuracy and robustness in an autonomous navigation setting. 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.

In fact, multimodal perception was an integral part of autonomous navigation solutions and even played a critical role in their success [8] before the advent of end-to-end deep learning based approaches. It offers several advantages, namely robustness to individual sensor noise/failure, improved object classification and tracking [35, 36, 37], robustness to varying weather and environmental conditions, etc. It is worth mentioning that though Mirowski et al. [34] uses multimodal sensors to navigate through a maze, information such as depth is only used as an auxiliary loss, i.e. it is not an input to the trained policy but is only used to improve the learning outcomes. In fact, Mirowski et al. [34] pointed out that the naive RGBD policy performs worse than predicting depth as a regression task,

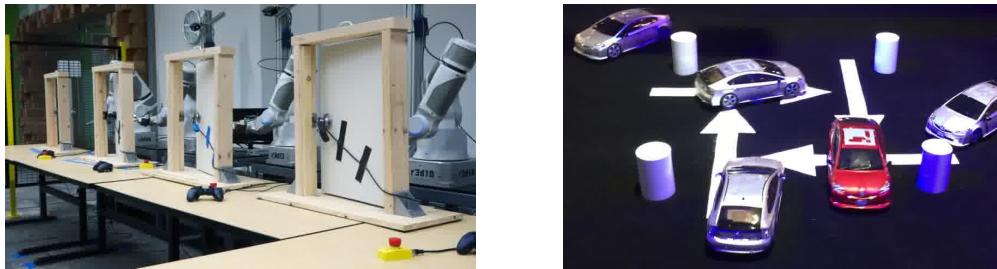


Figure 1.3: Recent successes on applying deep reinforcement learning (DRL) algorithms to real-world robotics problems: (a) manipulation [4]. (b) autonomous navigation. [5]

1.2 Approaches

In this work, we first proposed a model-based local planner solve the optimal kinodynamic motion planning problem. The local planner is built upon a vehicle model that effectively captures important state for off-road navigation such as lateral velocity. Practical problems such as re-planning and optimization are addressed under the standard sample based approach, but modified appropriately to encourage a more aggressive maneuvering. The resulting model-predictive planner was tested on the full-size All-Terrain Vehicle (ATV) in the off-road conditions.

In addition, we also present an alternative end-to-end controller that uses multi-sensor input to learn an autonomous navigation policy in a physics-based gaming environment called TORCS [38]. To show the effectiveness of multimodal perception, we picked two popular continuous action DRL algorithms namely Normalized Advantage Function (NAF) [39] and Deep Deterministic Policy Gradient (DDPG) [40], and augmented them to accept multimodal inputs. Though a multimodal sensor policy may greatly improve 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. Our approach reduces the policy sensitivity to a particular sensor subset, and make it capable of functioning even in the face of partial sensor failure. Based on the Sensor Dropout, we further embedded the standard DRL loss with an auxiliary loss that help reduce the action variations of the multimodal policy. As far as we know, we are the first to address the multimodal policy learning in terms of sensor fusion.

Through empirical testing we demonstrate that our proposed multimodal sensor policy can 1) operate with minimal performance drop in noisy environments, and 2) remain functional even in the face of a sensor subset failure. Finally, through the visualization of gradients, we show that the learned policies are conditioned on the same latent input distribution despite having multiple sensory observations spaces - a hallmark of true sensor-fusion. Suitable metrics have been devised to study this behavior and highlight its applicability to other domains.

Note that either the traditional or the end-to-end planner require the cost/reward heuristic function to guild the policy and formally define the optimization problem. However, unlike in the urban environment where cost function can be well defined in a rule-based structure, finding a cost function for off-road navigation can be nontrivial and often requires lot of handy tuning. [41] Here, motivated by the recent successes [42, 43], we also investigate into the using deep neural network as the function approximator, and infer the cost/reward function under inverse reinforcement learning platform.

The thesis is organized as follows: In Chapter 2 we briefly go through the preliminary background, including the basic sample based planner, the formal definition of deep (inverse) reinforcement learning, and the two continuous action agents used in this work. In Chapter 3 we formulate our model predictive planner, followed with the derivation of vehicle model, detailed planner design for optimal planning, and finally the experimental results on field. In Chapter 4 we propose a new technique at the aspect of sensor fusion in order to improve the performance of the end-to-end controller with multi-sensor inputs. In Chapter 5 we investigate the traversability analysis with the recently popular studies on deep inverse reinforcement learning. Finally, we conclude the work in 6 and address some of the interesting direction for future works.

Chapter 2

Related Work

2.1 Sample Based Planning

Sample based planners search for the feasible paths through the configuration space in a probabilistic fashion. Similar to graph-based approaches, the validity of configuration space need not be explicitly formulated, instead only a function that determines whether a specific configuration is valid. The probabilistic property makes sample based planners suitable to tackle high dimensional configuration space planning problems, yet with the trade-off on the sub-optimality of the solution. The Rapidly-exploring Random Tree (RRT) is a sampling based planner which build trees of valid trajectories (edges) between points in configuration space (nodes). As shown in Algorithm 1 [44], to grow these graphs a new point in configuration space is sampled, and the closest point on the existing tree is extended towards this sampled point.

Algorithm 1 RRT

```
1:  $T \leftarrow \text{InitTree}(x_{start})$ 
2: while GoalNotReached( $T, X_{goal}$ ) do
3:    $x_{sample} \in X \leftarrow \text{Sample}()$ 
4:    $x_{nearest} \leftarrow \text{Nearest}(T, x_{sample})$ 
5:    $x_{new} \leftarrow \text{Extend}(x_{nearest}, x_{sample})$ 
6:    $T \leftarrow \text{UpdateTree}((x_{nearest}, x_{new}), T)$ 
7: end while
```

The Extend function constructs a linearly trajectory in the configuration space from $x_{nearest}$ towards x_{sample} . The extension terminates at x_{new} either when x_{sample} is reached or the trajectory enters an invalid region, which can be specified by the validity function mentioned in the previous paragraph, or a hyper-parameter specified as the *step size*. However, The standard RRT planner does not guarantee optimality of the solution.

Karaman and Frazzoli [45] proposed an extension version of RRT that follow a asymptotic convergence to the optimal solution. As shown in Algorithm 2, the new algorithm, named RRT*, preserves the original structure yet embeds the following two additional procedures.

1. Instead of extending the tree from $x_{nearest}$ towards x_{new} , a local search on the subset X_{near} that centered at $x_{nearest}$ is performed in order to find an alternative candidate. The

new node x_{min} should be collision free and follows the constraint: $cost(x_{start}, x_{min}) + cost(x_{min}, x_{new}) < cost(x_{start}, x_{nearest}) + cost(x_{nearest}, x_{new})$.

2. Replace the original edges from x_{min} to $x_{nearest}$ with $E(x_{min}, x_{new})$ and $E(x_{new}, x_{nearest})$ if the resulting trajectory has lesser cost from x_{start} to $x_{nearest}$.

Algorithm 2 New Extend and Update Function in RRT*

```

1:  $x_{new} \leftarrow \text{Extend}(x_{nearest}, x_{sample})$ 
2:  $X_{min} \leftarrow \text{LocalSearch}(T, x_{nearest})$ 
3:  $x_{min} \leftarrow x_{nearest}$ 
4: for each  $x \in X_{min}$  do
5:    $oldCost \leftarrow cost(x_{start}, x_{nearest}) + cost(x_{nearest}, x_{new})$ 
6:    $newCost \leftarrow cost(x_{start}, x) + cost(x, x_{new})$ 
7:   if  $newCost < oldCost$  then
8:      $x_{min} \leftarrow x$ 
9:   end if
10: end for
11:  $T \leftarrow \text{UpdateTree}((x_{min}, x_{new}), T)$ 
12: for each  $x \in X_{min}$  do
13:    $oldCost \leftarrow cost(x, x_{nearest})$ 
14:    $newCost \leftarrow cost(x_{new}, x_{nearest})$ 
15:   if  $newCost < oldCost$  then
16:      $E \leftarrow E \setminus (x, x_{nearest})$ 
17:      $E \leftarrow E \cup (x_{new}, x_{nearest})$ 
18:   end if
19: end for

```

2.2 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:

$$J = \mathbb{E}_{s_i, r_i \sim E, a_i \sim \pi}[R_1] \quad (2.1)$$

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 .

$$V^\pi(s_t) = \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E, a_{i \geq t} \sim \pi}[R_t | a_t, s_t] \quad (2.2)$$

$$\begin{aligned} Q^\pi(s_t, a_t) &= \mathbb{E}_{r_{i \geq t}, s_{i > t} \sim E}[r(s_t, a_t) \\ &\quad + \gamma \mathbb{E}_{a_{i > t} \sim \pi}[Q^\pi(s_{t+1}, a_{t+1})]] \end{aligned} \quad (2.3)$$

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. [26] 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 [39, 40, 46, 47].

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 multimodal architecture, we briefly summarize the two algorithms below.

2.2.1 Continuous Action Space Algorithms

Normalized Advantage Function (NAF)

Q-learning [48] 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 (2.1) can be reached by minimizing the square loss Bellman error

$$L = \frac{1}{N} \sum_i^N (y_i - Q(s_i, a_i | \theta^Q))^2 \quad (2.4)$$

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[26], and has been shown to emulate human performance [27] in many Atari games using just image pixels as input. However, in all of these games, action choices are limited and discrete. Recently, Gu et al. [39] 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

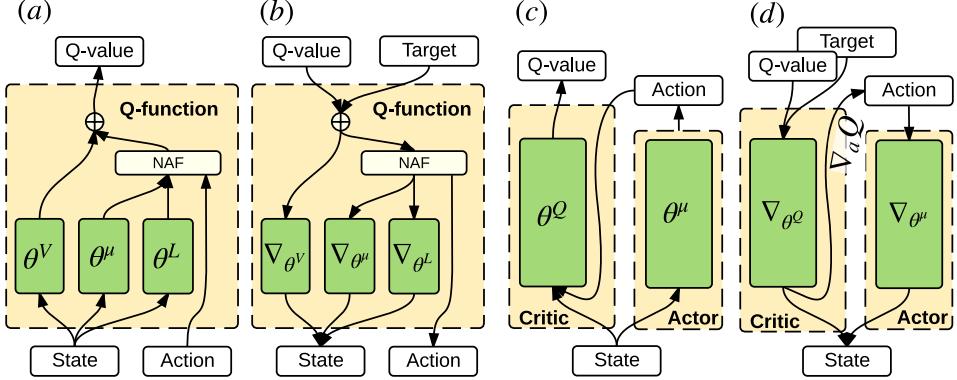


Figure 2.1: Schematic illustration of (a) forward and (b) back-propagation for NAF, and (c) forward and (d) back-propagation for DDPG. Green modules are functions approximated with Deep Nets.

states.

$$Q(s, a|\theta^Q) = A(s, a|\theta^\mu, \theta^L) + V(s|\theta^V) \quad (2.5)$$

$$\begin{aligned} A(s, a|\theta^\mu, \theta^L) &= -\frac{1}{2}(a - \mu(s|\theta^\mu))^T P(s|\theta^L) \\ &\quad (a - \mu(s|\theta^\mu)) \end{aligned} \quad (2.6)$$

$$P(s|\theta^L) = L(s|\theta^L)^T L(s|\theta^L) \quad (2.7)$$

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. 2.1 (a) and (b), respectively.

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. [49], 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.

$$\nabla_{\theta^\mu} J = \mathbb{E}[\nabla_a Q(s, a|\theta^Q) \nabla_a \mu(s)] \quad (2.8)$$

Silver et al. [49] proved that using the policy gradient calculated in (2.8) to update model parameters leads to the maximum expected reward.

Building on this result, Lillicrap et al. [40] proposed an extension of DPG with deep architecture to generalize their prior success with discrete action spaces [27] 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 [27] were utilized for stable learning. In

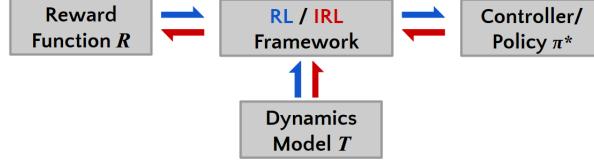


Figure 2.2: The block diagram of reinforcement learning and inverse reinforcement learning.

order to explore the full state and action space, an exploration policy was constructed by adding Ornstein-Uhlenbeck noise process [50]. 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. 2.1 (c) and (d), respectively.

2.3 Stochastic regularization

Stochastic regularization is an active area of research in deep learning made popular by the success of, *Dropout* [51]. Following this landmark paper, numerous extensions were proposed to further generalize this idea such as *Blockout* [52], *DropConnect* [53], *Zoneout* [54], etc. In the similar vein, two interesting techniques have been proposed for specialized regularization in the multimodal setting namely *ModDrop* [55] and *ModOut* [56]. 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.

2.4 Deep Inverse Reinforcement Learning (DIRL)

As shown in Fig. 2.2, in the standard reinforcement learning, we are interested in learning the optimal policy that maximizes the total expected rewards collected from the environment. However, for some of the interesting problems in robotics field, designing a reward function can be nontrivial and often requires lots of handy tuning. On the other hand, the consequences of the policy are usually relatively easy to observe. These problems fall in the category of *inverse reinforcement learning (IRL)* [57] (also imitation learning, learning from demonstration).

Given a set of expert demonstrations $D = \{\xi_i\}_{i=1}^N$, where each trajectory ξ_i consists of a sequence of state-action pair $\xi_i = \{(s_j, a_j)\}_{j=1}^K$, the goal of IRL is to infer the underlying reward function that leads to the policy π . We parametrized the reward function r with θ . Abbeel and

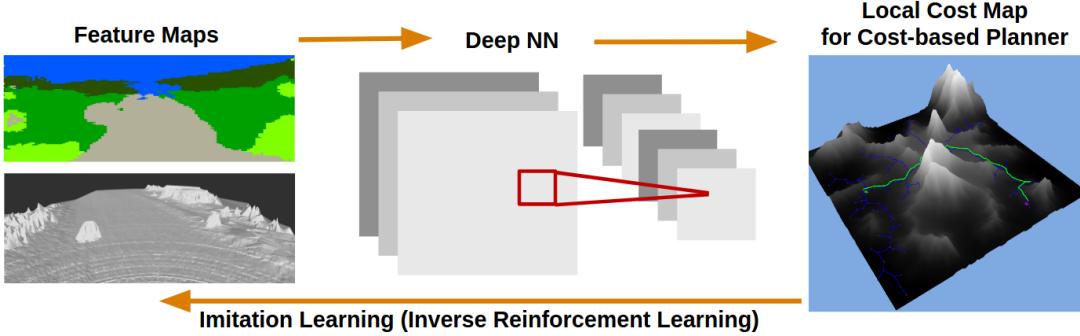


Figure 2.3: Schematic illustration of deep inverse reinforcement learning.

Ng [58] proposed a strategy of matching feature expectations between the demonstration policy and the learner's behavior.

Ratliff et al. [59] proposed a clever framework named *Maximum Margin Planning* (MMP), where learning to plan by mimicking the expert behavior is cast as a structured prediction problem. Under the structured margin criteria, the objective can be formulated in the following form:

$$L(\theta) = \frac{1}{N} \sum_{i=1}^N \beta_i (\max_{\mu \in \mathbb{G}_i} (\theta^T F_i + l_i^T) \mu - \theta^T F_i \mu_i)^q + \frac{\lambda}{2} \|\theta\|^2 \quad (2.9)$$

Here, μ_i , F_i , and l_i represent the state visited frequencies (SVF), feature matrix, and margin loss vector of the i^{th} trajectory, respectively. Note that Eq. 2.9 assumes the linearity constraint on the reward function. The parameter θ is optimized in a way that minimizes the q -norm mean losses between the optimal trajectory under the loss-augmented reward structure, and the demonstration trajectories. The loss vector alleviates the ill-posed problem in the standard IRL [58], where trivial solution such as all-zeroed weights guarantees all policies to be optimal. The intuition behind MMP implies the resulting solutions should look significantly better than the alternative policies.

However, both of the approaches fail to capture the ambiguity of the observed policies. It is worth mentioning that for complex problems such as off-road navigation or racing planning, the assumption on the optimality of the demonstrations is indeed strong. In fact, the optimal policy is usually unknown, and the behaviors may differ from each expert player. Specifically, when the demonstrations are imperfect, or the planning algorithms captures only part of the relevant feature space, the resulting reward function from the previous approaches cannot perfectly describe the observed trajectories. In this light, Ziebart et al. [60] reformulated the problem under the principle of maximum entropy. The resulting algorithm, known as maximum entropy IRL (ME-IRL), models the demonstrations with a stochastic policy by assuming the probability of executing of each trajectory being exponentially proportional to its total accumulated rewards. The gradient of the objective function under linear reward function is simply the difference between the expected empirical feature counts and the learner's expected feature counts.

Recently, Wulfmeier et al. [42] shew that the gradient calculated in ME-IRL can be naturally

extended to the back-propagation in the standard deep supervised learning. By re-framing the problem within the standard Bayesian inference as a MAP estimation problem, the objective function is simply:

$$L(\theta) = \log P(D, \theta|r) = \log P(D|r) + \log P(\theta) \quad (2.10)$$

The negative log-likelihood contains two terms: the data term L_D and a standard weight decay term L_θ as the regularization. The gradient of the first term is simply:

$$\frac{\partial L_D}{\partial \theta} = \frac{\partial L_D}{\partial r} \frac{\partial r}{\partial \theta} \quad (2.11)$$

$$= (\mu_D - \mathbb{E}[\mu]) \cdot \frac{\partial r}{\partial \theta} \quad (2.12)$$

Note the gradient degenerate to ME-IRL if the reward function is linear with respect to input feature. If the reward function is approximated by a deep network, the latter term $\partial r/\partial \theta$ can be calculated efficiently under back-propagation, and the objective function can be optimized using gradient-based approaches. This framework, called deep maximum entropy deep inverse reinforcement learning (ME-DIRL or simply DDIRL) has shown to successfully apply in urban autonomous navigation [43]. The schematic illustration of the DDIRL is summarized in Fig. 5.1.

Chapter 3

Model Predictive Planning

In this chapter, we propose a model-based local planner for high-speed maneuvering in the off-road navigation application. In order to overcome the highly unpredictable vehicle dynamics in the unstructured terrain, we derive our vehicle model in a data-driven fashion. The high-dimensional vehicle model is used as a standpoint of our model-predictive local planner.

The local planner is constructed with Rapidly-exploring Random Tree (RRT) [21] as its template. Despite the fact that sample-based algorithms are generally more suitable for high-dimensional state space planning, it is not straightforward to be extended to kinodynamic planning. To operate to planning process in control space, our planner slightly departs from the standard RRT so as to perform at least a certain level of trajectory optimization given the limited computational cycle. The cost function is designed using traveling time to encourage the vehicle for a more aggressive maneuvering.

We use height map algorithm to efficiently process the incoming cloud point data and detect collisions in the future. The potential obstacles is accumulated on a simplified occupancy grid to produce a collision check grid map for local planner. Finally, the proposed planner is tested on a full-size all-terrain vehicle (ATV) in the off-road environment. We show that our planner is capable of performing smooth yet aggressive avoid static obstacles with high speed up to 30kph.

The chapter is organized as follows: Section 3.1 summarizes the derivation of two vehicle models. In Section 3.2, we introduce our sample-based planner, and detail each specific technique implementation we design. Finally, Section 3.1 shows the experimental results for the vehicle model, and the videos of high-speed navigation on a full-size all-terrain vehicle.

3.1 Vehicle Response Model

The forward predictive model $\dot{x} = f(x, u)$ is defined as a function that maps a pair of control action $u \in \mathcal{A}$ and state $x \in \mathcal{S}$ to the next state. As shown in Fig. 3.1a, in our application, the control space consists of the 2DOF executed velocity command, i.e. forward and rotational velocity, and the state space consists of the actual velocity responses in 2D plane, i.e. $[v_x, v_y, w]$, in the local frame. It is worth mentioning that the sliding velocity v_y is not negligible on the rough terrain, and in fact plays a crucial role for off-road application.

Previous works [61, 62, 63] for unmanned ground vehicle (UGV), formulated the predictive

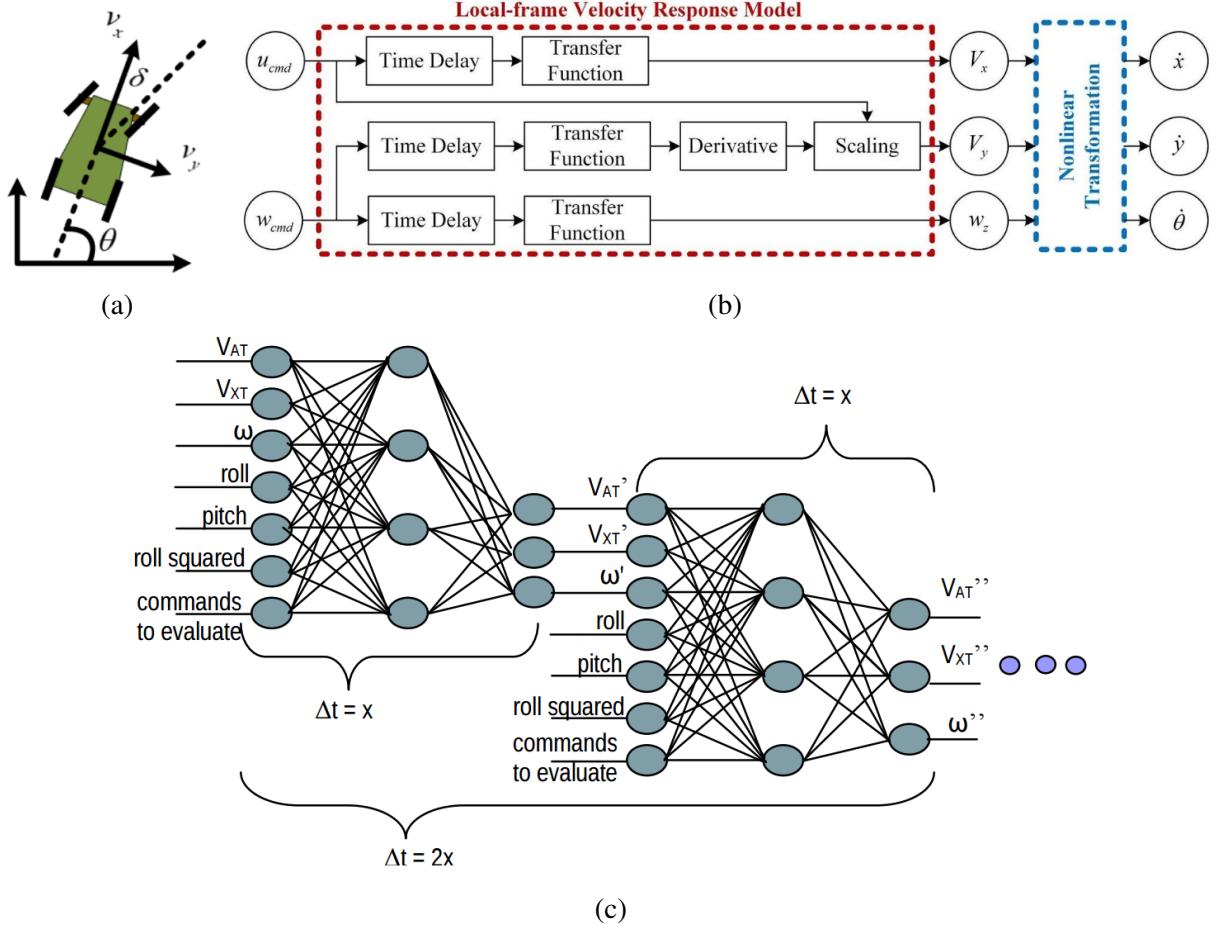


Figure 3.1: (a) Notation of vehicle velocity in local frame. (b) Data flow of conventional dynamic response model. The model takes two velocity control commands as input and estimates the velocity response in vehicle frame. (c) Schematic illustration of neural-network based response model from [6].

model in a modular fashion, where specialized algorithms were developed for each sub-system and later integrated with some fine tunings. Though this approach offers rich semantic representation to allow researchers to better examine the performance of each module, modeling sub-systems such as actuator dynamic, vehicle suspension model, and wheel-terrain interaction can be very complex and computational expensive. Here, we limit our scope to a more intuitive yet effective method. Instead of modularizing the process where the accumulating errors due to the imperfect approximations may propagate and scale up to harm the overall performance, we approximate the predictive model in a more end-to-end fashion.

We derive the model with two different approaches. The first approach is based on the standard system identification process by deriving the transfer function. The forward and angular velocity, i.e. v_x and w_z , responses take the forward and angular velocity commands as inputs, respectively. The lateral velocity response on the other hand uses both commands as input. Our observation is that the UGV experiences lateral sliding whenever the rotational speed changes,

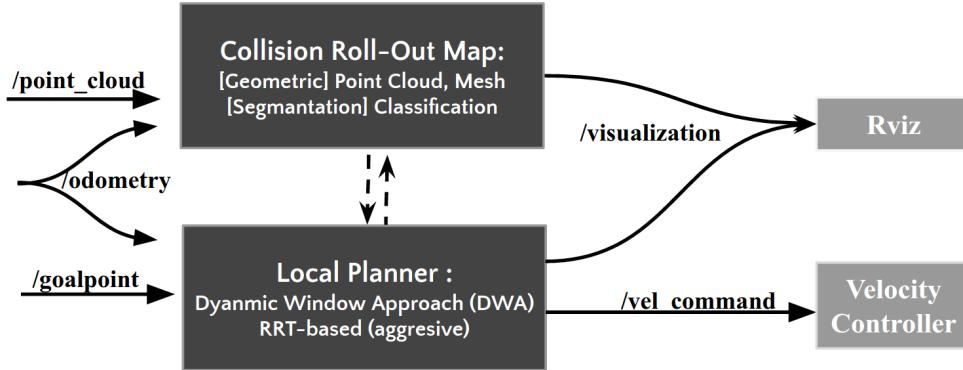


Figure 3.2: Block Diagram of our planner API.

with the magnitude of the angular acceleration being proportional to the forward velocity. The corresponding block diagram is summarized in Fig. 3.1b. The relative position with respect to the origin in local frame can be calculated by integrating through velocities.

For the second approach, we implement the neural-network model based on previous work [6]. As shown in Fig. 3.1c, the neural network takes additional informations, such as roll, pitch, and yaw angle, as its input. Note that the squared roll angle is also provided as the fact that from dynamics standpoint, the vehicle should respond in a near symmetrical manner if it is rolled to the right or rolled to the left.

We verified the performances of the two models on predicting vehicle states response on the full-size all-terrain vehicle in Section 3.3. The two models are referred as *Conventional Dynamic Model* and *NNet Model* for convenience.

3.2 Planner Design

The block diagram of our planner is shown in Fig. 3.2. It can be separated into two modules with respect to functionality. The collision check module constructed a global simplified occupancy grid given the point cloud data and the odometry. The local planner module takes the odometry and the goal state as inputs, performs minimal calculations, and finally sends the velocity commands directly to the on-board velocity controller. The two modules interact with each other in the way that the local planner acquires collision detection from roll-out map. The detail of two modules is described as follows:

3.2.1 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:

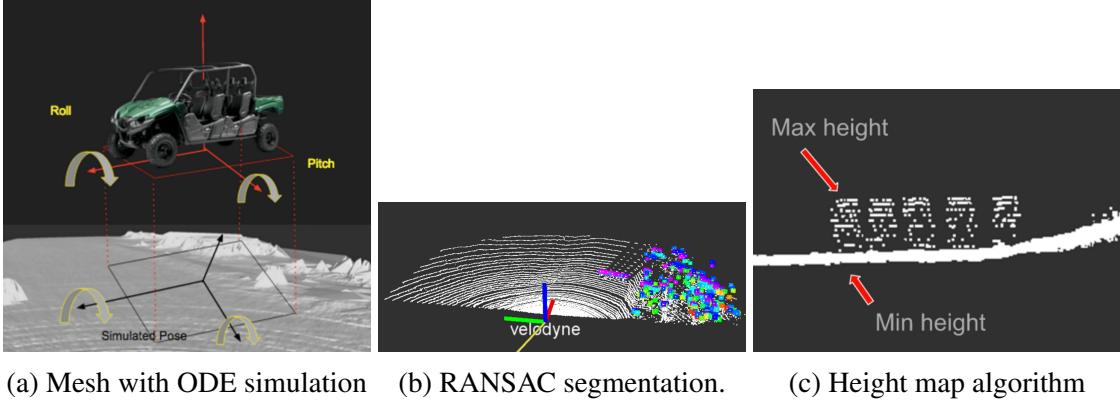


Figure 3.3: Three different methods of collision check. Note that for (b), the original and the processed point cloud is represented with white and colored dot, respectively.

i. Mesh Representation with Simulation in Open Dynamic Engine (ODE) [64]

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 [65]

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. 3.3b, the white point cloud is the original data, while the colored point cloud is the outliers from RANSAC.

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. 3.3c, 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. Though 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 on the other hand 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. It's the fastest and the most reliable. Furthermore, to optimize the computing efficiency, we used bitwise

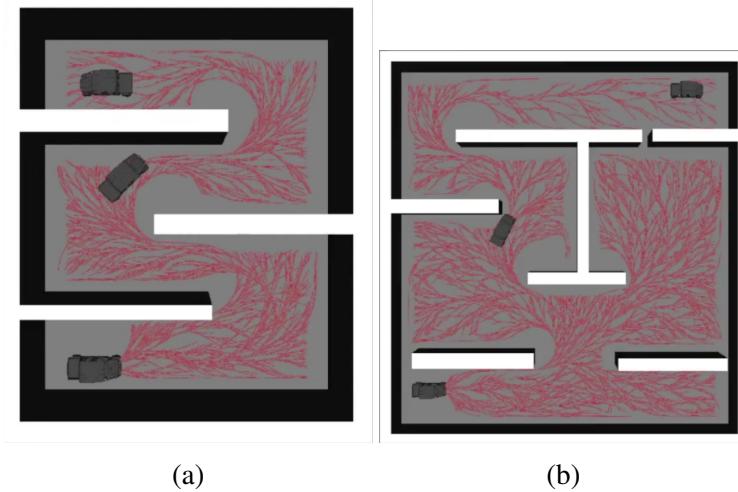


Figure 3.4: Simulation of RRT planner with vehicle model derived in Sec. 3.1

Table 3.1: Simulation Benchmark

	PDST [66]	EST [67]	RRT [21]	KPIECE [68]
TIME	15.04	15.05	15.03	15.07
SOLUTION CLEARANCE	1.63	1.54	1.38	1.70
SOLUTION DIFFERENCE	1.05	1.11	0.98	1.30

operation instead of multiplication and the obstacle size are dilated to increase robustness of our system.

3.2.2 Sample-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.

Simulation

Our planner is built upon Open Motion Planning Library (OMPL) [69], an open-source motion planning library that includes a wide range of sample-based planners and a built-in simulation platform called *OMPL.app*. The template of the sampled-based planner is determined from a simulation on *OMPL.app*, where we generate several mazes scenario and implement the vehicle response model on several available off-the-shelf planners. The simulated benchmark result is shown in Table 3.1. The simulation environment is shown in Fig. 3.4, with demo video available at <http://ppt.cc/sBLAh>. We observe that the RRT algorithm gives a faster solving time and a smaller path clearance under our vehicle model constraint. With other advantages such

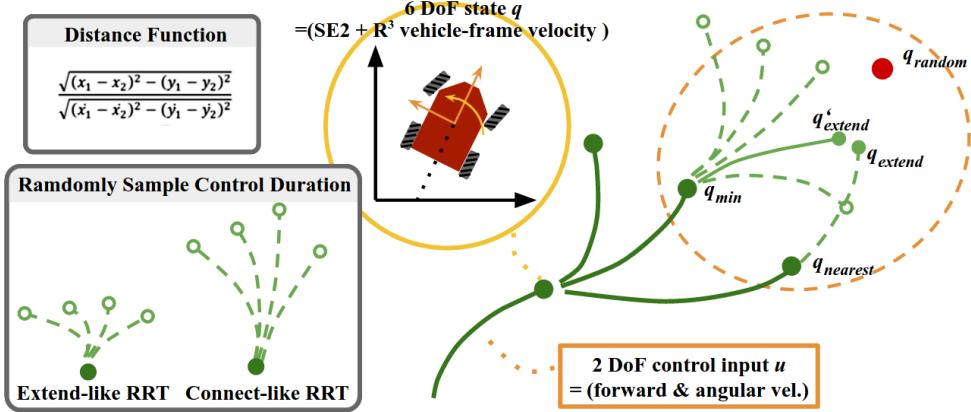


Figure 3.5: The visualization of RRT-based planner.

as flexibility and extensibility, we use RRT as our planner template and extend it for our own purpose.

Implementation Details

Fig 3.5 visualizes the RRT tree. Each node represents a 6 DoF states, $q = [x, y, \theta, v_{forward}, v_{sliding}, w]^T$. The first three terms represent a standard SE2 state space, and the last three terms stand as a vehicle-frame velocity in 2D plane. For control space, we follow the convention used in Section 3.1, which includes 2DOF forward and angular velocity.

During the standard process when RRT extends its leaf, a sequence of control input and its operation duration should be determined in order to propagate toward an extended state q_{extend} after a random state q_{random} is sampled. In order to perform a smooth control throughout the trajectory for off-road navigation, we first uniformly sample the forward velocity command within an adjustable region. This command region is determined at run-time 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, the shooting method is used to determine the angular velocity command.

As shown in the dotted circle in Fig. 3.5, the basic RRT-like algorithm preserves a fixed extended step size throughout planning time. While using a larger step size may result in a much smaller solving time yet a jerky trajectory, the trajectory propagated with a smaller step size is usually smoother but computational expensive. The former one is usually referred to *Connected RRT*, and the latter one is called *Extended RRT*. In our implementation, instead of fixing the control duration as a hyper-parameter, we randomly sample the control duration. Our motivations come from the results of our maze simulation, where we observe that the control duration affects the performance of the planner a lot in each different maze configuration. We believe loosening the step size constraint with such randomly sample mechanism will generalize our planner for various situations encountered at off-road environment.

Optimal kinodynamic motion planning for a sample based planner can be very computationally expensive. As described in Section 2.1, moving from RRT toward RRT* includes two more

Table 3.2: Planner Hyper-parameter

GOAL BIAS		0.7
PLANNER RATE		2 Hz
SOLVING RATE		20 Hz
CONTROL DURATION		1.5–6 SEC
CONTROL INPUT	FORWARD	10–30 KPH
	ANGULAR	-0.5–0.5 RAD/S

optimization steps in each iteration: (1) reconnecting to extended state, and (2) tree edges trimming. Here, we implement a time-optimal RRT* algorithm motivated from Frazzoli et al. [22]. However, only the first of two optimization steps is implemented because the trimming process includes updating the whole children tree, which will trade off with planning time. The extend function in the first procedure is implemented slightly different from the basic version in that instead of connecting q_{extend} directly to q_{\min} , we apply shooting method and replace q_{extend} with q''_{extend} . The intuition is that it is nearly impossible to propagate to the *same* node in 6 DoF state space with arbitrary control commands. The replacement of q_{extend} is accepted only if the new state q''_{extend} is close enough to q_{extend} . Finally, We also follow the same setting by using the traveling time instead of standard Euclidean distance as the cost function. The planner thus optimizes the trajectory by minimizing the traveling time and is encouraged for an agile maneuvering under the dynamics constraints.

The re-planning process is designed as follows: 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. The problem is solved multiple times, and each of the best solution in each iteration is stored. When the computational time exceeds, the solution with minimal cost is outputted for execution. Note that the growth tree is abandoned when the next solving iteration starts. In practical such design prevents the planner from publishing poor solution if bad tree structure is built at the beginning of growing stage. We admit that if an optimal solution can be obtained from one single shot, the replanning setting would have changed significantly [70]. It is also worth mentioning that there is a time delay between the time when vehicle status is updated and the time when the velocity command generated by RRT-based planner is executed. To effectively capture the latency, we utilize the same data-driven dynamics model to estimate the vehicle state after such time delay. The parameters used in our local planner are listed in Table 3.2.

3.3 Experiments and Analysis

We use Yamaha Viking VI side-by-side ATV as our main testing platform. As shown in Fig. 3.6, the vehicle is equipped with custom drive-by-wire system, velocity controller, and navigation sensors such as GPS/INS, LiDAR, and RGB-D camera.

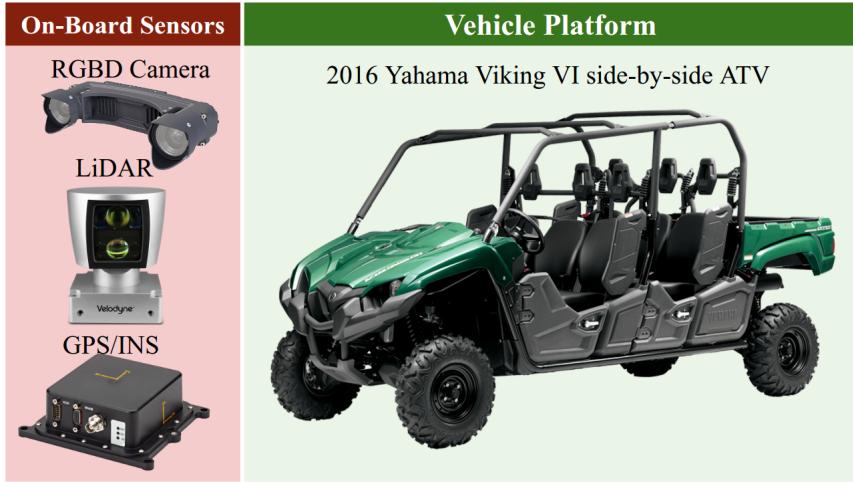


Figure 3.6: The testing vehicle platform and the on-board sensor.

3.3.1 Vehicle Model Verification

The simulated velocity is shown in Fig. 3.7, while the integrated relative position is shown in Fig. 3.8. The differences from the blue and red line reflect the complex of vehicle model on unstructured terrain. Integrating directly from control command through time, as shown in the blue line, will give a poor result on state propagation. Both of the two proposed models are capable to estimate the lateral velocity, which plays a non-negligible role in off-road cases. However, the conventional dynamic model is not capable to capture the nuance variation of the forward velocity. The neural net model on the other hand gives a better estimation.

3.3.2 Planner Demo

We test our planner on an off-road test field located near Gascola, Penn Hills, PA. Our testing scenario is designed as followed: the vehicle should autonomously navigate through a straight turnpike where multiple static obstacles are placed alternatively on both side of track. The static obstacle is 3-meter-length and 1.5-meter height. The turnpike is a rectangle-shaped field with 10-meter width and 150-meter length. The fastest way to go through the obstacles without any collisions is to perform S-shape maneuvering. Since the standard operating speed ranges from 10 to 40kph, we set the baseline testing speed as 20kph with the top speed of 30kph.

As shown in our demo video ¹, the vehicle successfully avoided all the obstacles at 25kph. However, driving at higher speed ($\sim 30\text{kph}$) 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. 3.9.

¹On-field testing with path visualization: https://youtu.be/LibnO8_Sjm0

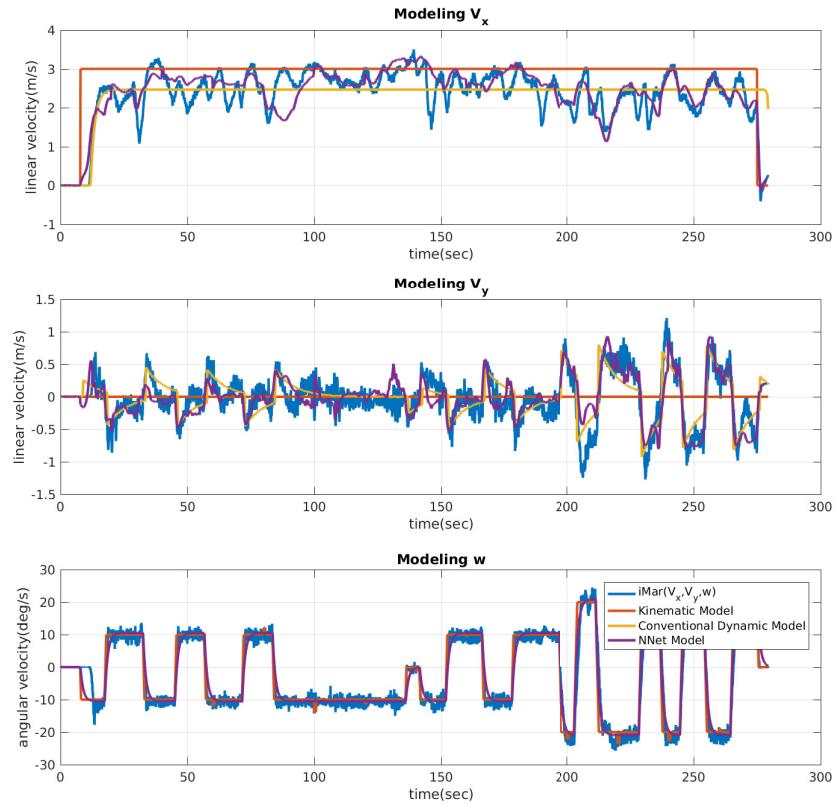


Figure 3.7: Simulated velocity response w.r.t. time steps. Note that the blue line is measured under an accurate GPS/INS sensor with RTK signal. We refer this measurement as our ground true data.

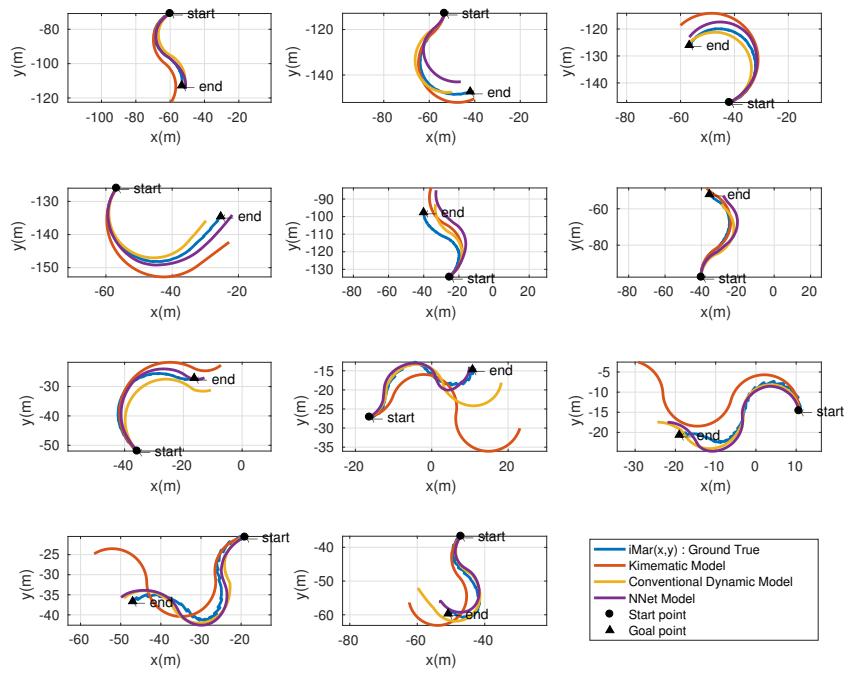


Figure 3.8: Comparison between data-driven dynamic model and original kinematic model. Note the time interval in each graph is 20 seconds.

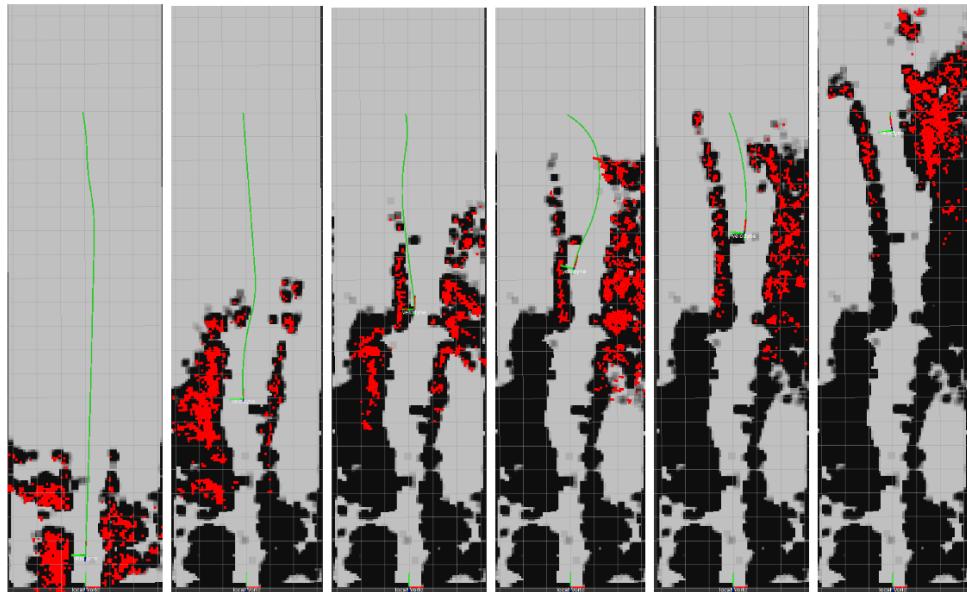


Figure 3.9: Screenshots of planner visualization output. The green path is the trajectory where each point is encoded with a 6 DoF state, 2 DoF control input, and a control duration. The red points are the filtered point cloud segmented as obstacles.

Chapter 4

Learning End-to-end Multimodal Sensor Policy

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. We propose two novel methods that effectively train the multimodal sensor policy without over-fitting to partial sensor subset, namely *Sensor Dropout* and *Auxiliary Loss*. Two popular continuous action DRL algorithms namely Normalized Advantage Function (NAF) [39] and Deep Deterministic Policy Gradient (DDPG) [40], are picked and augmented to accept multimodal input. The resulting autonomous navigation policy is tested exhaustively to verify its performance in a physics-based gaming environment called TORCS [38].

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 $\approx 40\%$ to just 10%, when compared to a baseline single sensor system.
2. Policies learned using SD best leverage the multimodal 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 multimodal 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.

The chapter is organized as follows: Section 4.1 introduces two methods on effectively training a multimodal sensor policy. We first introduce a new stochastic regularization called Sensor Dropout, and details its advantages over the standard Dropout for this problem. The resulting policy can be further fine tuned by adding additional auxiliary losses to reduce the action variance. The performance of Sensor Dropout is then validated in Section 4.2. In Section 4.3, we summarize our results and discuss key insights obtained through this exercise.

4.1 Proposed Methods

Multimodal 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 [71]. 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.

4.1.1 Multimodal 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 multimodal 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 Multimodal architecture is shown in Fig. 4.1. The outputs of feature extraction modules are eventually flattened and concatenated to form the multimodal state.

4.1.2 Sensor-based Dropout (SD)

The Sensor Dropout (SD) is a variant of the vanilla Dropout [51] that maintains dropping configurations on each sensor module instead of individual neuron. Though both methods share a similar motivation on stochastic regularization, SD is more better-motivated for training the multimodal sensor policy. By randomly dropping the sensor block during training, the policy network is encouraged to exploit the modularized structure among each sensor stream. In the application to the complex robotics system, SD has advantages on handling imperfect conditions such as latency, noises, and even partial sensor failure.

As shown in Fig.4.1, consider the multimodal 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 dropping configuration is defined as a M -dimensional vector $\mathbf{c} = [\delta_c^{(1)} \ \delta_c^{(2)} \ .. \ \delta_c^{(M)}]^T$, where each element $\delta_c^{(i)} \in \{0, 1\}$ represents the on/off indicator for the i^{th} sensor modality. We now detail the two main differences between original Dropout and SD along with their interpretations.

Firstly, note that the dimension of the dropping vector \mathbf{c} is much lower than the one in the standard Dropout ($\sum_{i=1}^M K_i$). As a consequence, the probability of the event where all sensors are dropped out (i.e. $\mathbf{c}_0 = [0^{(1)} \ 0^{(2)} \ .. \ 0^{(M)}]^T$) is not negligible in SD. To explicitly remove \mathbf{c}_0 , we slightly depart from [51] in modeling the SD layer. Instead of modeling SD as random process where any sensor block $\tilde{S}^{(i)}$ is switched on/off with a *fixed* probability p , we define the

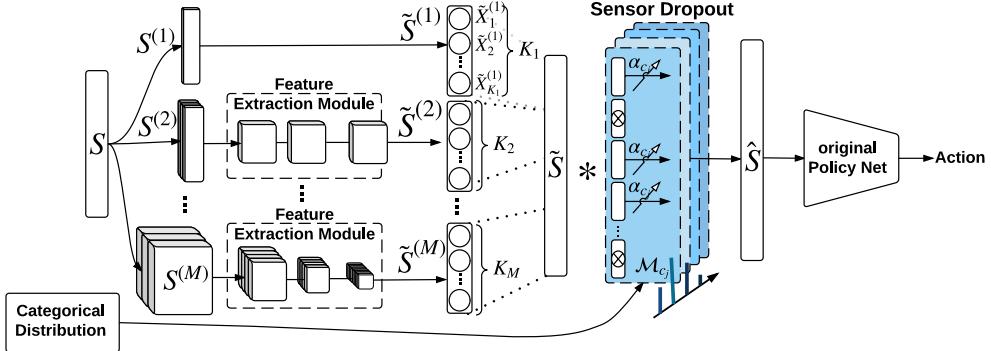


Figure 4.1: Illustration of multimodal sensor policy network augmented with Sensor Dropout. The operation $*$ stands for element-wised multiplication. The dropping configuration of Sensor Dropout is sampled from a categorical distribution, which the network takes as an additional input. The feature extraction module can be either pure identity function (modality 1), or convolution-based layer (modality $2 \rightarrow M$). The operation $*$ stands for element-wised multiplication.

random variable as the dropping configuration c itself. Since there are $N = 2^M - 1$ possible states for c , we accordingly sample from an N -state categorical distribution \mathbb{P} . The categorical distribution not only offers convenience in analysis and interpretation over standard Bernoulli on sensor blocks, but is better-motivated in that it can be adaptive to the current sensor reliability during run-time. We denote the probability of a dropping configuration c_j occurring with p_j , where the subscript j ranges from 1 to N . The corresponding pseudo-Bernoulli¹ distribution for switching on a sensor block $\tilde{S}^{(i)}$ can be calculated as $p^{(i)} = \sum_{j=1}^N \delta_{c_j}^{(i)} p_j$.

Another difference from the standard Dropout is the rescaling process. Unlike the standard Dropout which preserves a fix scaling ratio after dropping neurons, the rescaling ratio in SD is formulated as a function of the dropping configuration and sensor dimensions instead. The intuition is to keep the weighted summations equivalent among different dropping configurations in order to activate the later hidden layers. The scaling ratio is calculated as $\alpha_{c_j} = \frac{\sum_{i=1}^M K_i}{\sum_{i=1}^M \delta_{c_j}^{(i)} K_i}$.

In summary, the output of SD for the k^{th} feature in i^{th} sensor block (i.e. $\tilde{S}^{(i)}$) given a dropping configuration c_j can be shown as,

$$\hat{S}_{c_j,k}^{(i)} = \mathcal{M}_{c_j}^{(i)} \tilde{X}_k^{(i)}, \quad \text{where } \mathcal{M}_{c_j}^{(i)} = \alpha_{c_j} \delta_{c_j}^{(i)}. \quad (4.1)$$

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

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.

4.1.3 Augmenting Auxiliary Loss

An alternative interpretation of the SD-augmented policy is that sub-policy induced by each sensor combination are jointly optimized during training. Denote the ultimate policy and sub-policy induced by each sensor combination as $\mu_{c \sim \mathbb{P}}$ and μ_{c_j} , respectively. The final output maintains a geometric mean over N different actions.

Despite the expectation of the total policy gradients for each sub-policy is the same, SD provides no guarantees on the consistency of these actions. To encourage the policy network to extract salient features from each sensor that can be embedded with similar representation in the policy network, we further augment an auxiliary loss that penalizes the inconsistency among μ_{c_j} . This additional penalty term provides an alternative gradient that reduces the variation of the ultimate policy, i.e. $\text{Var}[\mu_{c \sim \mathbb{P}}]$.

The mechanism is motivated from the recent successes [34, 72, 73, 74] that exploit how adding the auxiliary tasks help greatly improve both agent's performance and convergence rate. However, unlike most previous works that pick up the auxiliary tasks carefully from the ground truth environment, we formulate the *target action* from the policy network itself. Under the standard actor-critic architecture, the target action is defined as the sub-policy $\tilde{\mu}_{c^*}$ among target actor network $\tilde{\mu}_{c \sim \mathbb{P}}$ that maximize the target critic values \tilde{Q} .

$$L_{aux} = \lambda \sum_{i=1}^N (\mu_{c_j}(s_i) - \tilde{\mu}_{c^*}(s_i))^2 \quad (4.2)$$

$$\text{where } c^* = \underset{c_j \sim \mathbb{P}}{\operatorname{argmax}} \sum_{i=1}^N \tilde{Q}(s_i, \tilde{\mu}_{c_j}(s_i)) \quad (4.3)$$

Here, λ is an additional hyper-parameter that indicates the importance ratio between the two losses, and N is the batch size for off-policy learning.

4.2 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.

4.2.1 Platform Setup

TORCS Simulator The proposed approach was verified on TORCS [38], a popular open-source car racing simulator that is capable of simulating physically realistic vehicle dynamics as well as multiple sensing modalities [75] to build sophisticated AI agents. 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

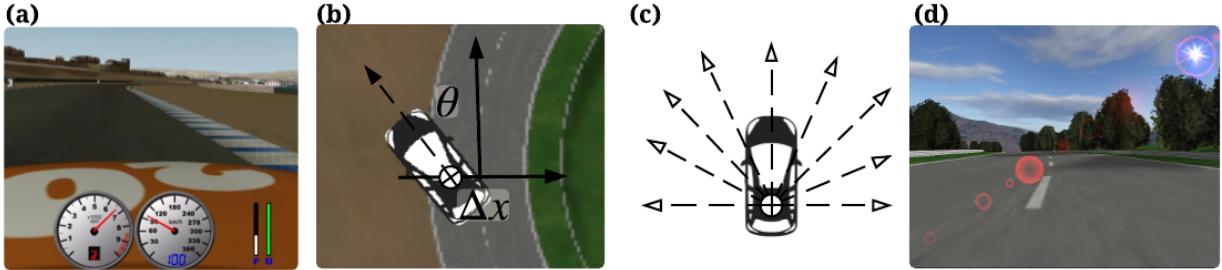


Figure 4.2: Sensors used in the TORCS racing car simulator: *Sensor 1*: Physical information such as velocity (a), position, and orientation (b), *Sensor 2*: Laser range finder (c), and *Sensor 3*: Front-view camera (d).

- position and speed. The action space is a continuous vector in \mathbb{R}^2 , whose elements represent steering angle, and acceleration.

As shown in Fig. 4.2, the physical state is a 10 DOF hybrid state, 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). Each laser scan is composed of 19 readings spanning a 180° field-of-view in the front of car. Finally, camera provides RGB channels with resolution 64×64 . We use the following sensing modalities for our state description: (1) We define *Sensor 1* as a hybrid state containing physical-based information such as odometry and simulated GPS signal. (2) *Sensor 2* consists of 4 consecutive laser scans (i.e., at time t , we input scans from times t , $t - 1$, $t - 2$ & $t - 3$). Finally, as *Sensor 3*, we supply 4 consecutive color images capturing the car’s front-view. These three representations are used separately to develop our baseline uni-modal sensor policies. The multimodal state on the other hand has access to all sensors at any given point. When Sensor Dropout (SD) is applied, agent will randomly lose access to a strict subset of sensors. The categorical distribution is initialized with a uniform distribution among total 7 possible combinations of sensor subset, and the best learned policy is reported here.

An exploration strategy is injected adding an Ornstein-Uhlenbeck process noise [50] to the output of the policy network. The choice of reward function is slightly different from Lillicrap et al. [40] and Mnih et al. [46] 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 [76].

Network Architecture For laser feature extraction module, we use two 1D convolution layers with 4 filters of size 4×1 , while image feature extraction is composed of three 2D convolution layers: one layer of 16 filters of size 4×4 and striding length 4, followed by two layers each with 32 filters of size 2×2 and striding length 2. Batch normalization is followed after every convolution layer. All these extraction modules are fused and are later followed up with two fully-connected layers of 200 hidden units each. All hidden layers have *relu* activations. The final layer of the critic network use *leaner* activation, while the output of the actor network are bounded using *tanh* activation. We use sigmoid activation for the output of L network in NAF. In practice, it leads to a more stable training for high dimensional state space. We trained with

Table 4.1: Model Specification

Model ID	State Dimensionality	Description
Physical	10	
Lasers	4×19	4 consecutive laser scans
Images	$12 \times 64 \times 64$	4 consecutive RGB image
Multi	$10+1 \times 19+3 \times 64 \times 64$	all sensor streams at current time step

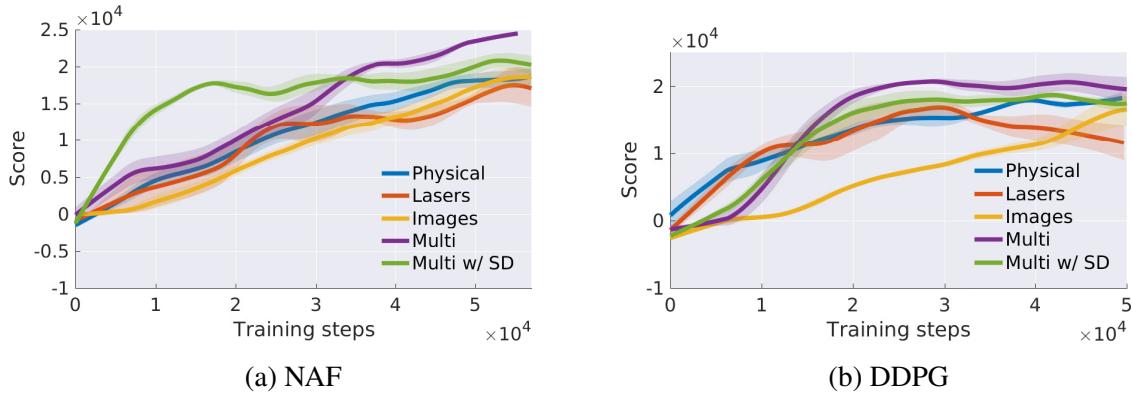


Figure 4.3: Training performance comparison of three baseline single sensor policies, and the proposed multimodal policies, with and without Sensor Dropout.

minibatch size of 16.

We used Adam [77] for learning the network parameters. For DDPG, the learning rates for actor and critic are 10^{-4} and 10^{-3} , respectively. We allow the actor and critic to maintain its own feature extraction module. In practice, sharing the same extraction module can lead to unstable training. Note that the NAF algorithm maintains three separate networks, which represent the value function ($V(s|\theta^V)$), policy network ($\mu(s|\theta^\mu)$), and the state-dependent covariance matrix in the action space ($P(s|\theta^L)$), respectively. In order to maintain a similar experiment setting and avoid unstable training, we maintain two independent feature extraction modules for θ^μ , and both θ^V and θ^L . In a similar vein, we apply a learning rate of 10^{-4} for θ^μ , and 10^{-3} for both θ^μ and θ^V .

4.2.2 Experimental Results

Training Summary

The training performance, for all the proposed models and their corresponding baselines, is shown in Fig. 4.3. The blue, red, and orange line represents three uni-modal policies. For DDPG, using high dimensional sensory input directly impacts convergence rate of the policy. (Note that the *Images* uni-policy has a much larger dimensional state space compared with *Multi* policy.) Counter-intuitively, NAF performs a nearly linear improvement over training steps, and

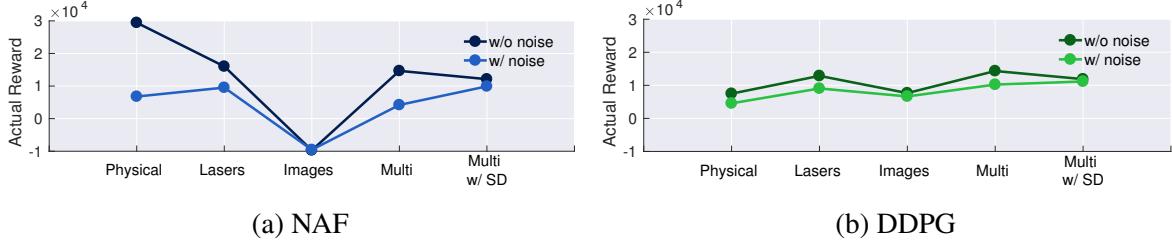


Figure 4.4: Policy Robustness Analysis: Darker lines connects average rewards of leaned policies with accurate sensing while the lighter lines connects the corresponding policies in the face of sensor noise.

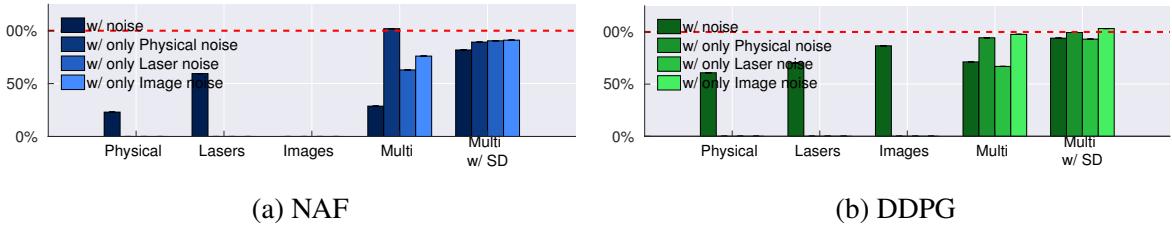


Figure 4.5: Policy Robustness Analysis: The bar box measures the relative scale among each of the models when noise is introduced. The red dotted lines show the performance without noise.

is relatively insensitive to the dimensionality of the state space. However, adding Sensor Dropout (SD) dramatically increases the convergence rate. Note that for both algorithms, the final performance for multimodal sensor policies trained with SD is slightly lower than training without SD, indicating that SD has a stochastic regularization effect similar to original Dropout.

Comparison with Baseline Models

Uni-modal policies: 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) according to [78]. Policy performance with and without noise is plotted for comparison in Fig. 4.4 and 4.5. While Fig. 4.4 plots the actual reward performance, Fig. 4.5 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 multimodal 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 multimodal 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 more contained for the multimodal policy and almost negligible when Sensor Dropout is used.

Table 4.2: Performance of Policy

POLICY	w/o NOISE	w/ NOISE	PERFORMANCE DROP
MULTI UNI-MODAL w/ META CONTROLLER	1.51 ± 0.57	0.73 ± 0.40	51.7 %
MULTIMODAL w/ SD	2.54 ± 0.08	2.29 ± 0.60	9.8 %

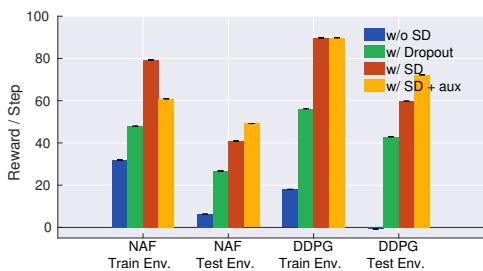


Figure 4.6: Policy performance when facing random sensor failure.

Table 4.3: Results of the sensitivity metric.

	TRAINING ENV.	TESTING ENV.	
		w/o SD	w/ SD
NAF	w/o SD	1.651	1.722
	w/ SD	1.284	1.086
DDPG	w/o SD	1.458	1.468
	w/ SD	1.168	1.171

Multiple Uni-modal Policy: Another intuitive baseline for the multi-sensor problem is to train each uni-modal sensor policy separately. Once individual policies are learned, we can train an additional meta controller that select which policy to follow given the current state. For this, we take the best trained policies of each sensor, and train a meta controller that takes the concatenated feature state from each sensor and output a *3DOF* softmax layer as the probability of choosing each uni-modal sensor policy. The meta controller is trained under the standard policy gradient method, which can be found in [79] for more details.

Policy performance with and without noise for two approaches are summarized in Table 4.2. The performance of the baseline policy drops dramatically once noise is introduced, which implies that without any regularization the uni-modal policy is prone to overfit. In fact, with the addition of noise the performance drop is sometimes severe in physical-based or laser-based policy. In comparison, the policy trained with SD reaches a higher score in both scenarios, and the drop is almost negligible.

Policy Robustness Analysis

In this part, we further validate our hypothesis that SD reduces the learned policy’s acute dependence on a subset of sensors in a multimodal 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 randomly blocked off part of the sensor modules, and scaled the rest of observation using the same rescaling mechanism as proposed in Section 4.1.2. Fig. 4.6 reports the average of the normalized reward of each model. A naive multimodal policy without any stochastic regularization (blue bar) performs poorly in the face of partial sensor failure and transfer tasks. Adding original Dropout does make the policy more generalized, yet the performance is not comparable with SD or with SD and auxiliary loss. Interestingly, by reducing the variance of the multimodal sensor policy with auxiliary loss, policy

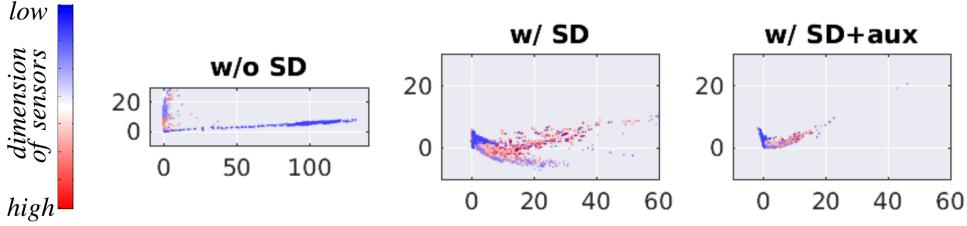


Figure 4.7: Two-dimensional PCA embedding of the representations in the last hidden layer assigned by the policy networks.

tends to have a better generalization among other environments.

Policy Sensitivity Analysis

To further examine the impact of SD 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 $\tilde{S}^{(i)}$. The technique is motivated from the salient map analysis [80], which has also been applied to DRL study recently [81]. To better analyze the effects of SD, we report on a the smaller subset by implementing SD layer to drop either (1) (*physical*, *laser*) or (2) *vision*. Consequently, the *sensitivity* metric is formulated as the relative sensitivity of the policy on two sensor subsets. If the ratio increases, the agent’s dependence shifts toward the sensor block in the numerator and vice versa. Assuming the fusion-of-interest is between the above-mentioned two subsets, we show in Table 4.3 that, using SD, the metric get closer to 1.0, indicating nearly equal importance to both the sensing modalities. The *sensitivity metric* is calculated as

$$\mathcal{T}_2^1 = \frac{1}{M} \sum_{i=1}^M \frac{\left| \nabla_{\tilde{S}_i^{(1)}} \mu(\tilde{S} | \theta^\mu) \right|_{S_i}}{\left| \nabla_{\tilde{S}_i^{(2)}} \mu(\tilde{S} | \theta^\mu) \right|_{S_i}} \quad (4.4)$$

Effect of Auxiliary Loss

In this experiment we verify how the auxiliary loss helps reshape the multimodal sensor policy and reduce the action variance. We extract the representations of the last hidden layer assigned by the policy network throughout an fixed episode. At every time step, the representation induced by each sensor combination is collected. Our intuition is that the latent space represents how the policy network interprets the incoming sensor stream for reaction. Based on this assumption, an ideal multimodal sensor policy should map different sensor streams to an similar distribution as long as the information provided by each combination is representative to lead to the same output.

Fig. 4.7 shows the two-dimensional Principle Component Analysis (PCA) embedding on the latent space of each sub-policy. The blue dots correspond to the representations induced by the sub-policy that use high dimensional sensor (e.g. *vision*) as its input. On the other hand, the red

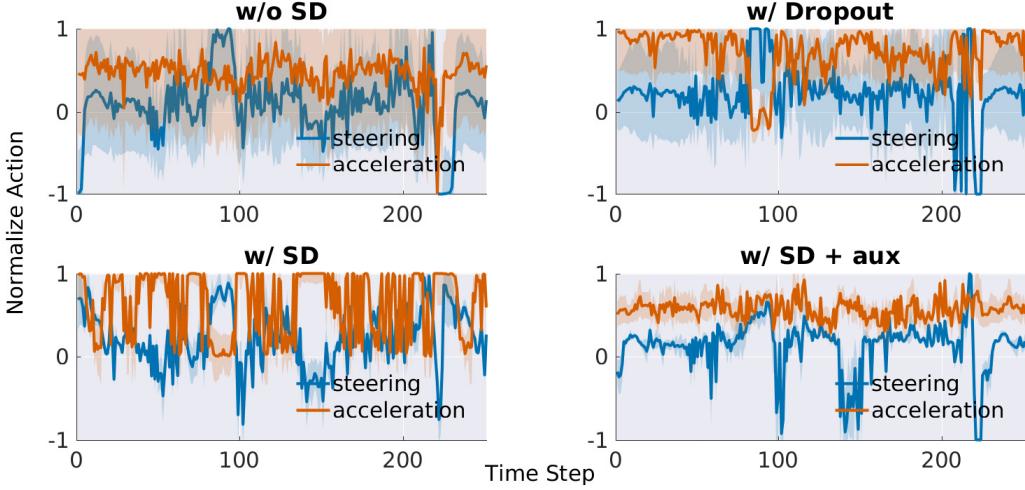


Figure 4.8: The variance of all the actions induced by sub-policy under each multimodal sensor policy. *Upper-left*: naive policy without any regularization. *Upper-right*: with standard Dropout. *Lower-left*: with Sensor Dropout. *Lower-right*: with Sensor Dropout and auxiliary loss.

dots represent the one with lower sensor stream such as odometry and range finder. Note that in practice, the covariance of the first third principle components contain around 85%. We provide the actual covariances for each component in the supplementary material.

As shown in Fig. 4.7, the naive multimodal sensor policy has a scattered distribution over the latent space, indicating that representative information from each sensor is treated very differently. In comparison, the policy trained with SD has a concentrated distribution, yet it is still distinguishable w.r.t. different sensors. Adding the auxiliary training loss encourages the true sensor fusion as the distribution becomes more integrated. During training, the policy is not only forced to explicitly make decisions under each sensor combination, but also penalized with the disagreements among multimodal sensor policies. In fact, as shown in Fig. 4.8, the concentration of the latent space directly affect the action variance induced by each sub-policy. We provide the action variance value in the supplementary material.

4.3 Discussion

4.3.1 Full Sub-Policy Analysis

The performance of each sub-policy is summarized in Fig. 4.9. As shown in the first and third column, the performance of the naive multimodal sensor policy (red) and the policy trained with standard Dropout (blue) drop dramatically as the policies lose access to image, which share 87.9% of the total multimodal state. Though Dropout increases the performance of the policy in the testing environment, the generalization is limited to using full multmodel state as input. On the other hand, Sensor Dropout (SD) generalize the policy across *sensor module* that make the sub-policies successfully transfer to the testing environment. It is worth mentioning that the

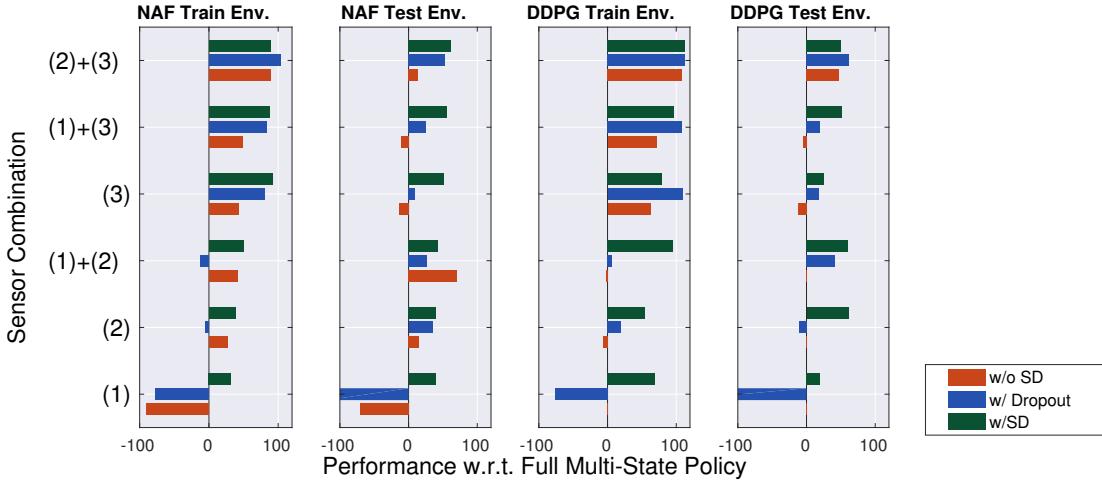


Figure 4.9: The full analysis of the performance of the total 6 sub-policies. The (1), (2), and (3) labels in y-axis represent physical state, laser, and image, respectively. The x-axis represent the remaining performance w.r.t. the SD policy with all sensor, i.e. (1)+(2)+(3).

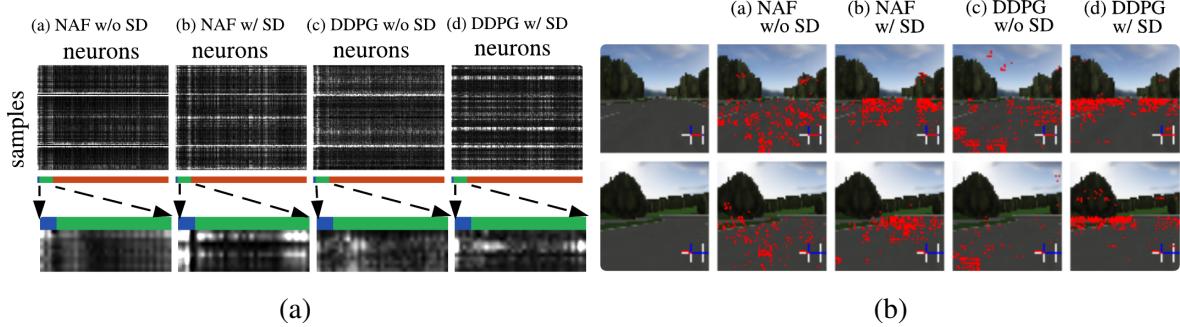


Figure 4.10: (a)The visualization of the magnitude of gradient for each neuron. The whiter color means the higher gradient. The color bar represents three different sensor modules: physical state(blue), Laser(green), and Image(red). (b) The gradient responses of actions on the image input for each of the multimodal agents. The top 20% gradients are marked red.

policies trained with SD is capable to operate when both laser and image sensor are blocked.

4.3.2 Visualize Policy Attention Region

The average gradient in the policy sensitivity section can also be used to visualize the region among each sensor that the policy network pays attention to. As shown in Fig. 4.10a, we observe that policies 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. To look for similar patterns, in Fig. 4.10b, 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. Note that though policies trained without SD tend to focus

Table 4.4: Covariance of the first three Principle Component

PRINCIPLE COMPONENT	NAF			DDPG		
	w/oSD	w/SD	w/SD+AUX	w/oSD	w/SD	w/SD+AUX
FIRST (%)	94.9	82.0	58.9	93.4	59.2	47.4
SECOND (%)	4.1	12.3	25.2	3.1	20.7	21.9
THIRD (%)	0.6	3.1	5.3	1.6	6.2	6.1

Table 4.5: Action Variation w.r.t. multimodal sensor

	NAF			DDPG		
	w/oSD	w/SD	w/SD+AUX	w/oSD	w/SD	w/SD+AUX
STEERING	0.1177	0.0819	0.0135	0.3329	0.0302	0.0290
ACCELERATION	0.4559	0.0472	0.0186	0.5714	0.0427	0.0143

more on the road, those areas are in plain color and offer little salient information. In conclusion, policies trained with SD are more sensitive to features such as road boundary, which is crucial for long horizon planning. In comparison, network trained without SD has a relatively low and unclear gradients over both laser and image sensor state space.

4.4 Supplementary Material

The covariance of PCA and the actual action variance is summarized in Table 4.4 and 4.5, respectively.

Chapter 5

Learning from Demonstration using DDIRL

Most of the robotics problems can be solved under optimization framework. For the motion planning problem, the objective is to find a sequence of control actions that minimizes the accumulated costs toward the goal state. Reinforcement learning on the other hand formulates the problem as finding a policy that maximizes the expected accumulated rewards collected from the environment. Both problems require a definition of the cost/reward function of the interested robotics tasks. However, unlike in the urban environment where cost function can be well defined in a rule-based structure, finding a cost function for off-road navigation can be nontrivial and often requires lot of handy tuning. [41] Here, motivated by the recent successes [42, 43] to using deep neural net to approximate the cost function, we implemented a similar algorithm for off-road application.

5.1 Proposed Methods

The pipeline of deep inverse reinforcement learning (DDIRL) is shown in Fig. 5.1. The raw sensor inputs first process through the feature extraction module. The resulting feature maps stand as the inputs of the deep neural network that approximate the cost function. Once the cost map is generated, we can formulate the corresponding Markov Decision Process (MDP) problem and solve for the expected state-visited frequencies (SVF). Finally, we compare the expected SVF inferred under the current reward structure with the target SVF calculated from the demonstrations, and the gradient is applied using Eq. 2.12.

5.1.1 Learning from Failure for DDIRL

The fundamental challenge rises with DDIRL is the spatially sparse gradient feedback. Since the gradient signals mainly and only come from the difference between demonstrations and expected trajectories, the loss feedbacks in training will inevitably focus more on the traversible regions. The problem can be alleviated by pre-training the network under standard image segmentation framework [82], which provide a pixel-wise feedback for error terms. Here, we propose an alternative approach that re-formulated the some problem with *negative* demonstrations.

Following the same convention in Section 2.4, we now denote the positive and negative

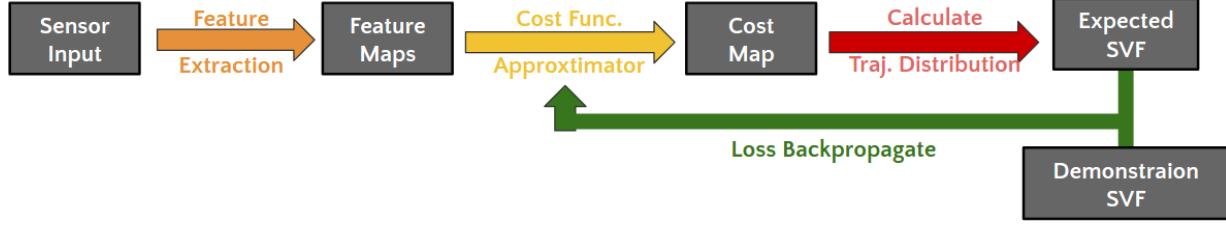


Figure 5.1: Block diagram of deep inverse reinforcement learning.

demonstrations as D_{pos} and D_{neg} , respectively. The log likelihood (Eq. 2.10) can be reformulated as:

$$L(\theta) = \log P(D_{pos}, D_{neg}, \theta | r) \quad (5.1)$$

$$= \log P(D_{pos} | r) + \log P(D_{neg} | r^{-1}) + \log P(\theta) \quad (5.2)$$

With the L2-regularization, the new gradient descent becomes:

$$\frac{\partial L}{\partial \theta} = (\mu_{D_{pos}} - \mathbb{E}_r[\mu]) \frac{\partial r}{\partial \theta} + (\mu_{D_{neg}} - \mathbb{E}_{r^{-1}}[\mu]) \frac{\partial r^{-1}}{\partial r} \frac{\partial r}{\partial \theta} + \frac{\lambda}{2} \|\theta\|^2 \quad (5.3)$$

$$= (\mu_{D_{pos}} - \mathbb{E}_r[\mu] + \mathbb{E}_{r^{-1}}[\mu] - \mu_{D_{neg}}) \cdot \frac{\partial r}{\partial \theta} + \frac{\lambda}{2} \|\theta\|^2 \quad (5.4)$$

where $r^{-1} = \text{constant} - r$ stands for the *inverted* reward map. By jointly optimizing with the negative demonstrations, we can increase the gradient feedbacks for non-traversable regions where the positive demonstrations will never capture.

5.1.2 Modeling Optimality Ambiguity

As described in Section 2.4, the objective function of IRL is usually formulated either in a fashion of structured support vector machines (SSVM), or conditional random fields (CRFs). While Maximum Margin Planning (MMP) fall in the first category, Maximum Entropy IRL (ME-IRL) can represent the second category. However, both categories share a similar form on the gradient of its objective. Recall Eq. 2.9 and Eq. 2.12, while gradient of MMP involves in solving the *optimal* trajectory on the evolving reward map, the gradient of ME-IRL instead requires solving the *expectation* of the trajectory distribution. The rest remains mostly the same. Pletscher et al. [83] proposed a generalized loss that models the observed expert behaviors with Gibbs distribution with an inverse temperature $\beta \in \mathbb{R}^+$.

$$P_\beta(\xi_i | r_\theta) = \frac{1}{Z_\beta} \exp(\beta r_\theta(\xi_i)) \quad (5.5)$$

$$\text{,where } Z_\beta = \sum_{\xi \in \Xi} \exp(\beta r_\theta(\xi)) \text{ is the normalization constant.} \quad (5.6)$$

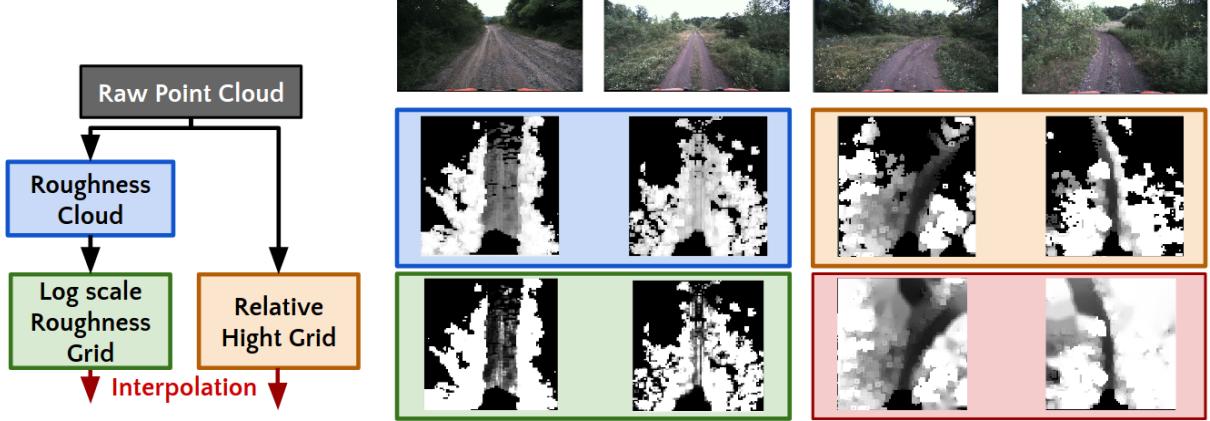


Figure 5.2: The extracted feature maps from LiDAR. The flow chart is shown on the left of the figure, while examples of the actual collected data on file are shown on the right of the figure.

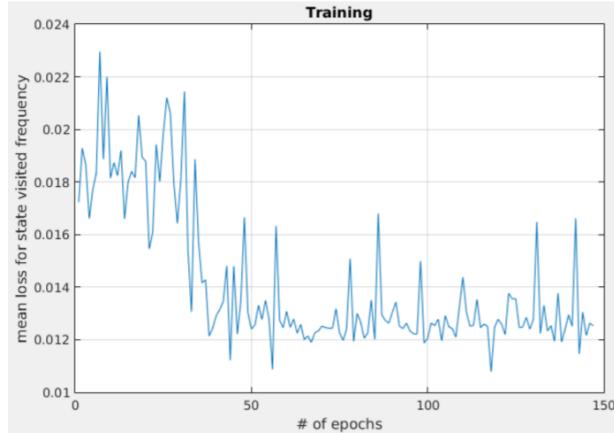


Figure 5.3: Training curve of DIRL in Gascola dataset.

As β increases, the exponent term dominates, and the corresponding distribution become sharper. If $\beta \rightarrow \infty$, the distribution degenerates to solving the arg max, i.e. solving the optimal trajectory. On the other hand, if $\beta \rightarrow 1$, the distribution becomes the same form of the one used in ME-IRL. In other words, the SSVM-like and CRF-like losses are framed under the same framework, and can be seen as two special cases located at the opposite side of the spectrum.

This generalized loss introduces a new parameter β that captures on which level of the *optimality* is presented by the demonstration trajectories. By using this generalized loss in our computational graph, the parameter can be optimized jointly during training.

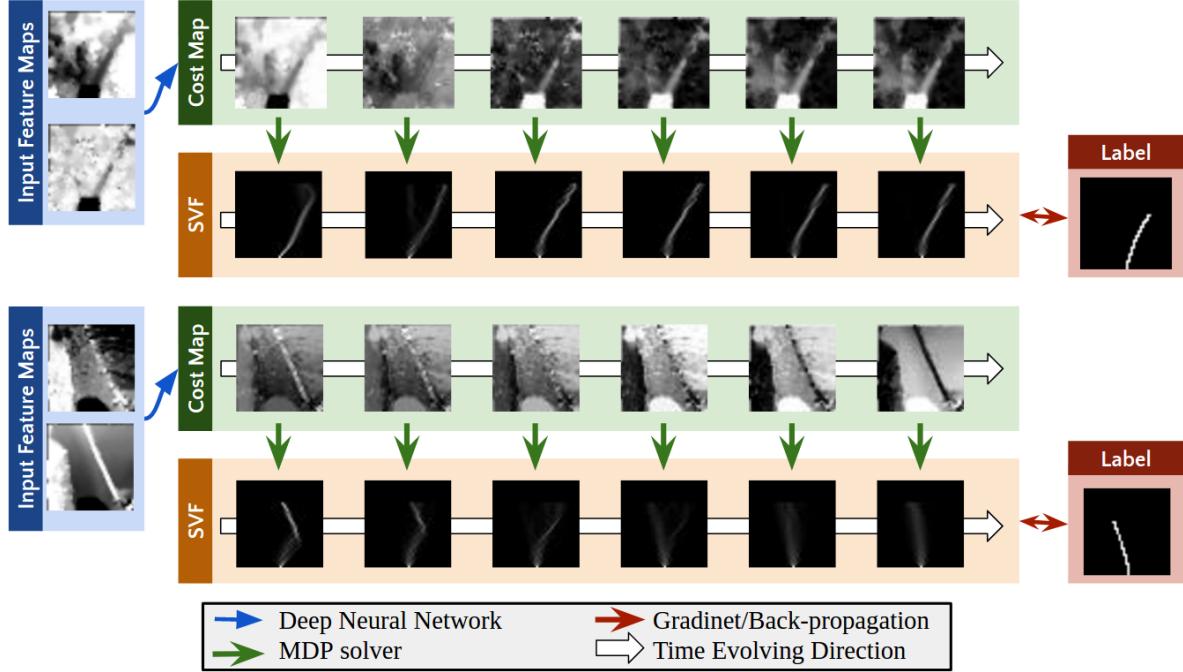


Figure 5.4: Samples of intermediate cost maps during training.

5.2 Experiments Results

Following the similar setup in Chapter 3, we use Yamaha Viking VI side-by-side ATV as our main testing platform, with LiDAR as our primary input sensor. For off-road navigation where scenes can change dramatically, LiDAR usually provides a more reliable measurement for the safety concern. In addition, informative features such as terrain roughness can be inferred straight-forwardly. We collect the dataset with total 150 human demonstrations covering an off-road testing field at Gascola, PA. Each sample covers the $20m \times 20m$ region in the front of the ATV. Since the Gascola dataset is relatively small, we use a shallow multilayer perception as the cost function.

The first feature map captures information about terrain roughness. As shown in Fig. 5.2, the basic approach to infer the terrain roughness from the raw point cloud is to first divide it into patches. For each patch, a normal plane is fitted using standard least square regression. Once the plane parameters are calculated, the *roughness index* can be inferred by averaging plane variance within each patch. However, this naive roughness index is sensitive to hyper-parameter such as patch size and rescaling factors. In the off-road environment where terrain types can vary a lot, this approach can fail to extract reasonable feature when trail becomes narrow, as shown in the second column of Fig. 5.2. To alleviate this issue, we transfer the original roughness index with the log-scale value. The log scale helps amplify the slight difference among traversible terrains, in which we are in general more interested. On the other hand, the non-traversable terrains such as bush or trees are flatten under the log scale. Another informative feature map is the relative

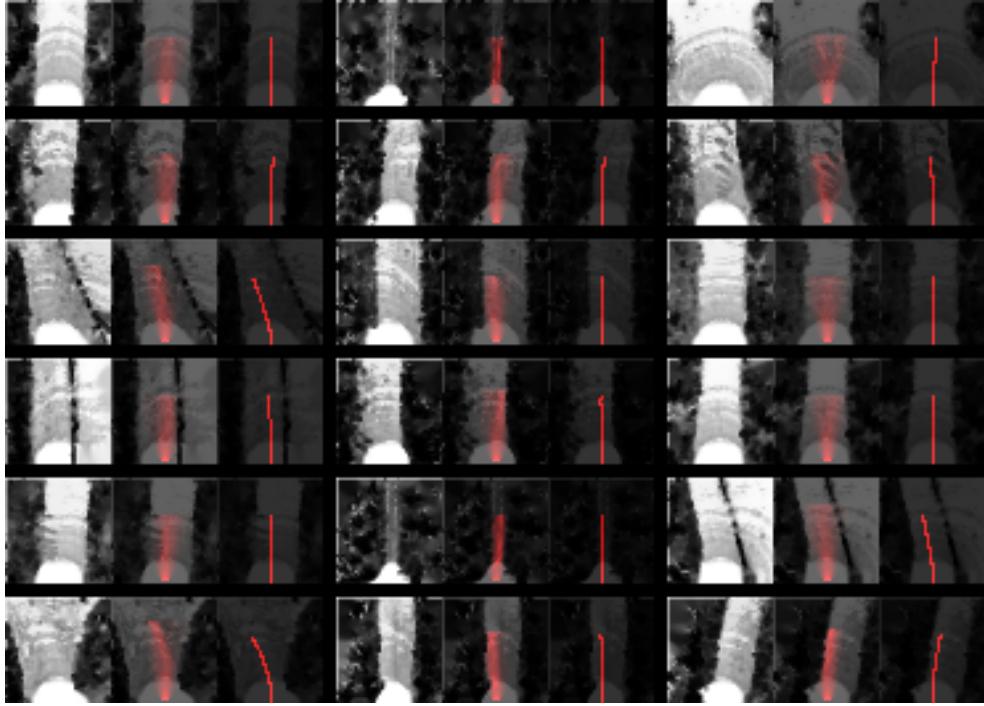


Figure 5.5: Final cost map on testing set. The left, middle, and right image represent the output cost map, the same cost map overlaid with expected SVF, and with demonstrations, respectively.

height with respect to the vehicle standing level.

The intuition behind the two feature maps is that the vehicle should prefer driving over the terrains that are smoother or at the similar horizontal level of its current local frame. Note that both of the feature maps are projected on a top-down view, leaving some parts of the feature maps unoccupied. However, the invisible grids may confuse the network and make the framework fragile to the senses. Though the problem can be alleviated by simply providing an additional feature map that specifies the visibility, it usually requires much more data and the use of convolution layer. [42, 43] Since our dataset is relatively small, we alleviate this issue with an alternative approach. Instead, we apply a standard image in-painting technique [84] that effectively helps infer the invisible region given the geometric shape of the visible trails. The feature maps before and after the interpolation are shown in the orange and red area in Fig. 5.2, respectively.

The training curve is summarized in Fig. 5.3, and two testing examples of the intermediate cost maps during training are visualized in Fig. 5.4. Finally, the resulting cost map on testing data set is shown in Fig. 5.5.

Chapter 6

Conclusion and Future Work

TODO

In this work, we introduce a new stochastic regularization technique called Sensor Dropout to promote an effective fusing of information from multiple sensors. The variance of the resulting policy can be further reduced by introducing an auxiliary loss during training. We show that the aid of SD reduces the policy sensitivity to a particular sensor subset, and make it capable of functioning even in the face of partial sensor failure. Moreover, the policy network is able to automatically infer and weight locations providing salient information. For future work, we wish to extend the framework to other environments such as real robotics systems, and other algorithms like GPS[85], TRPO [47], and Q-Prop [72], etc.. Secondly, systematic investigation into the problems such as how to augment the reward function for other important driving tasks like collision avoidance, and lane changing, and how to adaptively adjust the SD distribution during training are also interesting avenues that merit further study.

Bibliography

- [1] 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. (document), 1.1, 1.1
- [2] Anthony Stentz, John Bares, Thomas Pilarski, and David Stager. The crusher system for autonomous navigation. *AUVSIs Unmanned Systems North America*, 3, 2007. (document), 1.1, 1.1.1
- [3] 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. (document), 1.1.2, 1.2
- [4] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *arXiv preprint arXiv:1610.00633*, 2016. (document), 1.3
- [5] Preferred Network LLC. Autonomous robot car control demonstration in ces2016, 2016. URL <https://www.youtube.com/watch?v=7A9UwxvgcV0>. (document), 1.3
- [6] Michael Bode. Learning the forward predictive model for an off-road skid-steer vehicle. 2007. (document), 3.1
- [7] William J Mitchell, Chris E Borroni-Bird, and Lawrence D Burns. *Reinventing the automobile: Personal urban mobility for the 21st century*. 2010. 1.1
- [8] 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, 1.1.2
- [9] 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. 1.1
- [10] J Zico Kolter, Christian Plagemann, David T Jackson, Andrew Y Ng, and Sebastian Thrun. A probabilistic approach to mixed open-loop and closed-loop control, with application to extreme autonomous driving. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 839–845. IEEE, 2010. 1.1
- [11] Grady Williams, Paul Drews, Brian Goldfain, James M Rehg, and Evangelos A

- Theodorou. Aggressive driving with model predictive path integral control. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 1433–1440. IEEE, 2016. 1.1
- [12] Andrew Gray, Yiqi Gao, Theresa Lin, J Karl Hedrick, H Eric Tseng, and Francesco Borrelli. Predictive control for agile semi-autonomous ground vehicles using motion primitives. In *American Control Conference (ACC), 2012*, pages 4239–4244. IEEE, 2012. 1.1
 - [13] Mark Cutler and Jonathan P How. Autonomous drifting using simulation-aided reinforcement learning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5442–5448. IEEE, 2016. 1.1
 - [14] Mark Cutler, Thomas J Walsh, and Jonathan P How. Reinforcement learning with multi-fidelity simulators. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3888–3895. IEEE, 2014. 1.1
 - [15] Sven Koenig and Maxim Likhachev. D^{*} lite. 1.1.1
 - [16] Yoshiaki Kuwata, Gaston A Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P How. Motion planning for urban driving using rrt. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 1681–1686. IEEE, 2008. 1.1.1
 - [17] Alonzo Kelly, Anthony Stentz, Omead Amidi, Mike Bode, David Bradley, Antonio Diaz-Calderon, Mike Happold, Herman Herman, Robert Mandelbaum, Tom Pilarski, et al. Toward reliable off road autonomous vehicles operating in challenging environments. *The International Journal of Robotics Research*, 25(5-6):449–483, 2006. 1.1.1
 - [18] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001:48105, 2008. 1.1.1
 - [19] H Shibly, Karl Iagnemma, and S Dubowsky. An equivalent soil mechanics formulation for rigid wheels in deformable terrain, with application to planetary exploration rovers. *Journal of terramechanics*, 42(1):1–13, 2005. 1.1.1
 - [20] D Rubinstein and R Hitron. A detailed multi-body model for dynamic simulation of off-road tracked vehicles. *Journal of Terramechanics*, 41(2):163–173, 2004. 1.1.1
 - [21] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA’00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000. 1.1.1, 3, 3.1
 - [22] Jeong hwan Jeon, Sertac Karaman, and Emilio Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the rrt. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pages 3276–3282. IEEE, 2011. 1.1.1, 3.2.2
 - [23] 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. 1.1.2
 - [24] 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. 1.1.2

- [25] 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. 1.1.2
- [26] 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. 1.1.2, 2.2, 2.2.1
- [27] 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. 1.1.2, 2.2.1, 2.2.1
- [28] 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. 1.1.2
- [29] Craig Quiter. Deepdrive: self-driving car ai, 2016. URL <http://deepdrive.io/>. 1.1.2
- [30] Udacity. Udacity’s self-driving car simulator. <https://github.com/udacity/self-driving-car-sim>, 2017. 1.1.2
- [31] Ziyan Wang Cewu Lu Yurong You, Xinlei Pan. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017. 1.1.2
- [32] 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. 1.1.2
- [33] 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. 1.1.2
- [34] 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. 1.1.2, 4.1.3
- [35] Jos Elfring, Rein Appeldoorn, and Maurice Kwakkernaak. Multisensor simultaneous vehicle tracking and shape estimation. In *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pages 630–635. IEEE, 2016. 1.1.2
- [36] 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. 1.1.2
- [37] 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. 1.1.2

- [38] 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. 1.2, 4, 4.2.1
- [39] 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. 1.2, 2.2, 2.2.1, 4
- [40] 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. 1.2, 2.2, 2.2.1, 4, 4.2.1
- [41] David Silver, J Andrew Bagnell, and Anthony Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29(12):1565–1592, 2010. 1.2, 5
- [42] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015. 1.2, 2.4, 5, 5.2
- [43] Markus Wulfmeier, Dominic Zeng Wang, and Ingmar Posner. Watch this: Scalable cost-function learning for path planning in urban environments. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 2089–2095. IEEE, 2016. 1.2, 2.4, 5, 5.2
- [44] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998. 2.1
- [45] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011. 2.1
- [46] 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. 2.2, 4.2.1
- [47] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015. 2.2, 6
- [48] 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. 2.2.1
- [49] David Silver, Guy Lever, Nicolas Heess, Thomas Degrif, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 2.2.1, 2.2.1
- [50] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. *Physical review*, 36(5):823, 1930. 2.2.1, 4.2.1
- [51] 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. 2.3, 4.1.2
- [52] Calvin Murdock, Zhen Li, Howard Zhou, and Tom Duerig. Blockout: Dynamic model

- selection for hierarchical deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2583–2591, 2016. 2.3
- [53] 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. 2.3
- [54] 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. 2.3
- [55] 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. 2.3
- [56] 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. 2.3
- [57] Andrew Y Ng, Stuart J Russell, et al. Algorithms for inverse reinforcement learning. 2.4
- [58] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM, 2004. 2.4, 2.4
- [59] Nathan D Ratliff, J Andrew Bagnell, and Martin A Zinkevich. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736. ACM, 2006. 2.4
- [60] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008. 2.4
- [61] Alonzo Kelly, Thomas Howard, and Colin Green. Terrain aware inversion of predictive models for high-performance ugv's. In *Defense and Security Symposium*, pages 65611R–65611R. International Society for Optics and Photonics, 2007. 3.1
- [62] Thomas Howard and Alonzo Kelly. Trajectory generation on rough terrain considering actuator dynamics. In *Field and service robotics*, pages 479–490. Springer, 2006. 3.1
- [63] Thomas M Howard and Alonzo Kelly. Terrain-adaptive generation of optimal continuous trajectories for mobile robots. 3.1
- [64] David Wettergreen and Michael Wagner. Developing a framework for reliable autonomous surface mobility. 2012. 3.2.1
- [65] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. 3.2.1
- [66] Andrew M Ladd and Lydia E Kavraki. Motion planning in the presence of drift, underactuation and discrete system changes. In *Robotics: Science and Systems*, pages 233–240, 2005. 3.1

- [67] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2719–2726. IEEE, 1997. 3.1
- [68] Ioan A Șucan and Lydia E Kavraki. Kinodynamic motion planning by interior-exterior cell exploration. In *Algorithmic Foundation of Robotics VIII*, pages 449–464. Springer, 2009. 3.1
- [69] Ioan A. Șucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. <http://ompl.kavrakilab.org>. 3.2.2
- [70] Dave Ferguson, Nidhi Kalra, and Anthony Stentz. Replanning with rrt*. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 1243–1248. IEEE, 2006. 3.2.2
- [71] Ratan Hudda, Clint Kelly, Garrett Long, Jun Luo, Atul Pandit, Dave Phillips, Lubab Sheet, and Ikhlaq Sidhu. Self driving cars. 2013. 4.1
- [72] 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. 4.1.3, 6
- [73] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. *arXiv preprint arXiv:1609.05521*, 2016. 4.1.3
- [74] Alexey Dosovitskiy and Vladlen Koltun. Learning to act by predicting the future. In *International Conference on Learning Representations (ICLR)*, 2017. 4.1.3
- [75] Naoto Yoshida. Gym-torcs. https://github.com/ugo-nama-kun/gym_torcs, 2016. 4.2.1
- [76] Yan-Pan Lau. Using keras and deep deterministic policy gradient to play torcs. <https://yanpanlau.github.io/2016/10/11/Torcs-Keras.html>, 2016. 4.2.1
- [77] Diederik P Kingma and Jimmy Lei Ba. Adam: A method for stochastic optimization. 2015. 4.2.1
- [78] 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. 4.2.2
- [79] 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. 2017. 4.2.2
- [80] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014. 4.2.2
- [81] Ziyu Wang, Nando de Freitas, and Marc Lanctot. Dueling network architectures for deep reinforcement learning. In *international Conference on Machine Learning (ICML)*, 2016. 4.2.2

- [82] Markus Wulfmeier, Dushyant Rao, and Ingmar Posner. Incorporating human domain knowledge into large scale cost function learning. 2016. 5.1.1
- [83] Patrick Pletscher, Cheng Soon Ong, and Joachim M Buhmann. Entropy and margin maximization for structured output learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 83–98. Springer, 2010. 5.1.2
- [84] Alexandru Telea. An image inpainting technique based on the fast marching method. *Journal of graphics tools*, 9(1):23–34, 2004. 5.2
- [85] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning (ICML)*, pages 1–9, 2013. 6
- [86] Animesh Garg Daniel Crankshaw Joseph E. Gonzalez Ken Goldberg Richard Liaw, Sanjay Krishnan. Composing meta-policies for autonomous driving using hierarchical deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017.
- [87] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *CoRR*, abs/1611.01796, 2016. URL <http://arxiv.org/abs/1611.01796>.
- [88] Sascha Lange, Martin Riedmiller, and Arne Voigtlander. Autonomous reinforcement learning on raw visual input data in a real world application. In *Neural Networks (IJCNN), The 2012 International Joint Conference on*, pages 1–8. IEEE, 2012.
- [89] Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *CoRR*, abs/1611.05397, 2016. URL <http://arxiv.org/abs/1611.05397>.
- [90] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 512–519. IEEE, 2016.
- [91] Manuel Watter, Jost Springenberg, Joschka Boedecker, and Martin Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems*, pages 2746–2754, 2015.
- [92] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2015.
- [93] Marco Pavone. Autonomous mobility-on-demand systems for future urban mobility. In *Autonomous Driving*, pages 387–404. Springer, 2016.
- [94] Tomasz Janasz and Uwe Schneidewind. The future of automobility. In *Shaping the Digital Enterprise*, pages 253–285. Springer, 2017.
- [95] 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.
- [96] 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.
- [97] Nitish Srivastava and Ruslan R Salakhutdinov. Multimodal learning with deep boltzmann machines. In *Advances in neural information processing systems*, pages 2222–2230, 2012.
- [98] 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.
- [99] 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.
- [100] Brandon Amos, Lei Xu, and J. Zico Kolter. Input convex neural networks. *arXiv preprint arXiv:1609.07152*, 2016.
- [101] 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.
- [102] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.
- [103] 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.
- [104] 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>.
- [105] 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.
- [106] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [107] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.
- [108] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.
- [109] Matthias Plappert. keras-rl. <https://github.com/matthiasplappert/keras-rl>, 2016.
- [110] Fereshteh Sadeghi and Sergey Levine. (cad)2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

- [111] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks. *arXiv preprint arXiv:1512.04455*, 2015.
- [112] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [113] 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.
- [114] François Chollet. Keras (2015). *URL <http://keras.io>*.
- [115] 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.
- [116] Eder Santana and George Hotz. Learning a driving simulator. *arXiv preprint arXiv:1608.01230*, 2016.
- [117] Steven Bohez, Tim Verbelen, Elias De Coninck, Bert Vankeirsbilck, Pieter Simoens, and Bart Dhoedt. Sensor fusion for robot control through deep reinforcement learning. *arXiv preprint arXiv:1703.04550*, 2017.
- [118] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.
- [119] Allen Nie. Stochastic dropout: Activation-level dropout to learn better neural language models.
- [120] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. *arXiv preprint arXiv:1506.02142*, 2, 2015.
- [121] James Andrew Bagnell, David Bradley, David Silver, Boris Sofman, and Anthony Stentz. Learning for autonomous navigation. *IEEE Robotics & Automation Magazine*, 17(2):74–84, 2010.
- [122] Keonyup Chu, Minchae Lee, and Myoungho Sunwoo. Local path planning for off-road autonomous driving with avoidance of static obstacles. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1599–1616, 2012.
- [123] Genya Ishigami, Keiji Nagatani, and Kazuya Yoshida. Path planning and evaluation for planetary rovers based on dynamic mobility index. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 601–606. IEEE, 2011.
- [124] Mark Cutler and Jonathan P How. Efficient reinforcement learning for robots using informative simulated priors. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2605–2612. IEEE, 2015.
- [125] Dave Ferguson and Anthony Stentz. Field d*: An interpolation-based path planner and replanner. In *Robotics research*, pages 239–253. Springer, 2007.
- [126] Sven Koenig and Maxim Likhachev. Improved fast replanning for robot navigation in unknown terrain. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE Inter-*

national Conference on, volume 1, pages 968–975. IEEE, 2002.

- [127] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *Advances in Neural Information Processing Systems*, pages 2154–2162, 2016.
- [128] David Stavens and Sebastian Thrun. A self-supervised terrain roughness estimator for off-road autonomous driving. *arXiv preprint arXiv:1206.6872*, 2012.
- [129] Panagiotis Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, 26(4):1373–1385, 2013.
- [130] Sergey Levine, Zoran Popovic, and Vladlen Koltun. Nonlinear inverse reinforcement learning with gaussian processes. In *Advances in Neural Information Processing Systems*, pages 19–27, 2011.
- [131] Abdeslam Boularias, Jens Kober, and Jan Peters. Relative entropy inverse reinforcement learning. In *AISTATS*, pages 182–189, 2011.
- [132] Edouard Klein, Matthieu Geist, Bilal Piot, and Olivier Pietquin. Inverse reinforcement learning through structured classification. In *Advances in Neural Information Processing Systems*, pages 1007–1015, 2012.
- [133] Navid Aghasadeghi and Timothy Bretl. Maximum entropy inverse reinforcement learning in continuous state spaces with path integrals. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 1561–1566. IEEE, 2011.
- [134] Kyungjae Lee, Sungjoon Choi, and Songhwai Oh. Inverse reinforcement learning with leveraged gaussian processes. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 3907–3912. IEEE, 2016.
- [135] Brian D Ziebart, Nathan Ratliff, Garratt Gallagher, Christoph Mertz, Kevin Peterson, J Andrew Bagnell, Martial Hebert, Anind K Dey, and Siddhartha Srinivasa. Planning-based prediction for pedestrians. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 3931–3936. IEEE, 2009.