Efficient Learning of Dynamics Models Using Terrain Classification

Bethany R. Leffler and **Christopher R. Mansley** and **Michael L. Littman**[1]

Abstract. Terrain classification in robotics has heavily focused on determining a region for traversal, while also labeling obstacles. Our work attempts to expand this essentially binary viewpoint and to use terrain classifiers as an indicator for switching between a set of system dynamics. By learning multiple models of the system dynamics, the robot is able to assess alternative paths based on traversal costs of different terrain types instead of strict distance metrics. We demonstrate a system that reliably learns an optimal control policy using this additional terrain information and contrast it with several systems based on more traditional methods that fail to reliably complete the same task.

1 Introduction

Terrain classification for road following is often a binary road (passable) versus non-road (impassable) classification. As such, the robot cannot distinguish more or less passable terrains so as to prefer more passable terrains but allowing for less passable terrains to be traversed if alternatives are limited or if doing so would lead to a greatly reduced travel time. In this work, we take a more utility-oriented approach, distinguishing between a wider class of terrains, learning distinct dynamics models for each, and using these models to plan cheapest paths taking terrain into consideration.

The work in this paper frames this problem in the setting of reinforcement learning [16]. Recent work in the field has provided bounds on how much experience is needed to learn optimal control policies [10]. These bounds are typically proven with no assumptions about similarities between states, which leads to learning algorithms that scale at best linearly with the number of states. In a robotic domain, where the space of inputs is potentially infinite, these bounds are typically not useful. Instead, this paper builds on related work that exploits structure in the underlying state space to reduce the quantity of information needed to learn accurate models [12]. By using automatically extracted classes to index separate dynamics models for learning, the agent is able to perform more flexible path planning.

2 Related Work

A typical implicit assumption in robot-navigation research is that the robot has one dynamics model, which describes how it traverses from one state to the next. One example is in the representation of the state transition model in a Kalman filter's predict step, which captures the prediction of the next state from the current state [9]. One of the ways of ensuring that this assumption holds is to have the system follow surfaces appropriate for the model. This approach is often referred to as road following [5]. A common way to determine what is a road and what is not is to use a supervised learning algorithm to construct a terrain classifier [13]. When roads are not marked, such as on dirt roads or in clearings, a combination of sensors such as laser range finders and video cameras can be used in this binary terrain classification task to train an agent to learn to recognize the road through supervised [14] and self-supervised techniques [6]. Research has also been done in terrain modeling to determine the pass-ability of an area when the ground cannot be seen [17].

Figure 1. Image of the LEGO® Mindstorms NXT robot in the experimental environment.

These approaches, when successful, result in the robot navigating correctly to its goal location along a single terrain class. However, the path taken might not be optimal in a utility-theoretic sense. By using the assumption that there is only one dynamics model in the world, the agent is unable to fully calculate the best path to the goal. The plan or policy generated would be sub-optimal because a single model would try to encompass both the good (road) and bad (off-road) parts of the world; this oversimplification can lead to improper policy cost estimation. By learning multiple dynamics models, the agent can more fully model the dynamics of the environment and calculate a better policy.

From a reinforcement-learning perspective, navigation algorithms are often assessed based on the agent's learned policy and the how far that policy is from optimal. Obtaining a good policy often means fully exploring the environment. Exploration, though, comes at a cost, and therefore must be done in an efficient manner [1]. Even an efficient exploration algorithm may not be enough to converge to a good policy in a reasonable amount of time. For this reason, many algorithms use generalization techniques, such as function approximation, to limit the amount of exploration needed to learn about the entire environment [8].

[1] Rutgers University

Our contributions in this paper include using classes or types extracted from images of terrain to allow for the generalization of action models across states, which speeds up the learning time while allowing for efficient exploration. We also demonstrate experimental environments that cannot be completed reliably without the use of separate state dynamics, showing that not only does this technique speed learning, but some situations actually require it.

3 Background

Previously, we introduced the idea of relocatable action models (RAM) to enhance exploration [11]. Instead of using the traditional Markov decision process (MDP) representation of states, actions, and transition functions, we used the RAM representation of states, actions, *types* and *outcomes* [15]. This alternative representation of the underlying MDP is defined by 8-tuple $\langle S, A, C, O, \kappa, t, \eta, r \rangle$, where S is a set of states, A is a set of actions, C is a set of *types*, O is a set of *outcomes*, $r : S \rightarrow \Re$ is the state dependent reward function, and $\kappa : S \rightarrow C$ is a mapping indicating how each state maps to a type or cluster. The function $t : C \times A \rightarrow \Pr(O)$ is a mapping known as the *relocatable action model*. It captures effects of different actions in a state-independent way by mapping a type and an action to a probability distribution over possible outcomes. In this work, outcomes are the change in location and orientation in robot coordinates. The mapping $\eta : S \times O \rightarrow S$ is the *next-state function*. It takes a state and an outcome and computes the resulting next state by transforming from robot coordinates to world coordinates.

In this paper, our vision-based terrain-classification system is what defines the type mapping κ. This mapping converts perceptual features of the state space into a finite set of terrain types. We show that κ can be computed *a priori* using a generic vision system. With κ and η defined, the learner can focus on the terrain-specific action model, resulting in fast, accurate learning.

Whereas prior work [11] used a hand-tuned classification of states to types, the current paper shows that this mapping can be extracted automatically and used successfully in the learning setting.

4 System Architecture

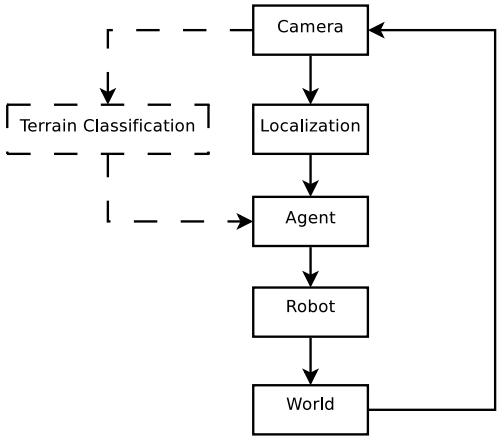


Figure 2. Flow chart of the system architecture. The dashed lines indicate information passing that occurs at startup.

Figure 2 shows the flow of data through our system for a robotic domain. Before the robot is placed in the environment, a picture is taken with an overhead camera and sent through an image-segmentation engine to determine terrain classification (see Section 4.1). Classification information is then stored as an additional feature for each state.

Once the state space is defined, the robot is placed in its starting configuration and the agent queries the localization system for the robot's position in the world. Using this information, the agent, with the guidance of the RAM-Rmax algorithm (see Figure 3), chooses which action to take based on the outcomes it has previously seen.

The selected action is then sent to the robot to execute. After execution is complete, the agent once again retrieves the robot's location information and uses it to calculate the latest outcome, which is added to a list of outcomes seen in the same terrain type. The agent then chooses the next action to take. This process continues until the localization system tells the agent that the robot is in the goal region or out of bounds. These occurrences end an episode; the robot is placed back in the starting location to execute another episode.

4.1 Terrain Classification

We used a IEEE1394 video camera to take an image of the world without the robot and then fed the image into the Edge Detection and Image SegmentatiON (EDISON) system [4] where similar terrains are determined based on color, texture, and proximity of pixels. Since the image-segmentation system determines the number of clusters on its own, the one parameter to be set is the minimum number of pixels that a cluster can contain. This value is set to the number of pixels that the robot occupies in the image to ensure that all of the robot's wheels can occupy a single patch of terrain at the same time.

The clustered image that EDISON returns is then converted into an indexed image. To limit the number of spurious clusters, all clusters that appear in fewer than $\frac{|S|}{10}$ states have their pixels reassigned to adjacent segments. The remaining clusters are used to determine the terrain-type feature of the state space. These terrain classifications will be used to determine to which dynamics model the learned outcomes will be applied.

4.2 Localization

The localization system is a standard fiducial-based system, which for these experiments acts as an indoor global positioning system (GPS). Using the same overhead camera as above, the location of a marker affixed to the robot is obtained using commonly available color-segmentation software [2]. The type of marker used is based on work done for a robotic soccer application [3]. The typical accuracy of this system is less than 5mm.

4.3 Learning The Dynamics Model

After an action is taken in state s and the position of the new state s' is fed back to the agent, the displacement and change in orientation between s and s' are calculated. These differences are stored as a list of *outcomes* for $\kappa(s)$. Instead of maintaining a list with an outcome for every action taken, which could grow without bound, the list of outcomes keeps a tally t_C of how many times that outcome C has been seen. This approach allows for proper calculation of the probability of seeing that outcome while minimizing the size of the list that the algorithm has to traverse. Since the number of possible outcomes is finite due to the granularity of the localization system, the maximum size of the outcome list is also finite.

4.4 Planning

When deciding which actions to take in the environment, the agent uses a parameterized variation of the Rmax algorithm that sets the expected reward of taking an action in a terrain to the maximum reward, R_{max}, if that combination of terrain and action has not been experienced often. An action becomes "known" for a particular terrain when the robot has performed that action in that terrain M times, where M is a free parameter. Once the action is "known", the value for that action becomes the solution to $Q(s,a) =$

$$r(s,a) + \gamma \left(\sum_{o \in O} \frac{t_C(\kappa(s),a,o)}{z} \times \max_{a' \in A} Q(\eta(s,o),a') \right), \quad (1)$$

which is updated as t_C changes Here, z is a normalization constant.

To calculate values, all outcomes for that terrain type need to be mapped back from feature space into state space. For instance, if the robot is in state s with $x = 35$, $y = 25$, $\theta = 90.0$ and the outcome o_1 is a displacement of magnitude 2.0 in direction 315.0 and the change in orientation is 314.0 degrees, then s' for that outcome would be the state with the features $x = 36$, $y = 26$, $\theta = 44.0$. This calculation is done for every outcome and the reward for each possible s' is weighted by its probability and summed.

Once the value for each action is calculated by the algorithm (see Figure 3), the agent chooses the action with the highest value and sends the corresponding command to the robot via Bluetooth®. While planning can take several milliseconds, these calculations can be performed while the robot is performing its actions (each action takes approximately one second), resulting in no computational delay.

Global data structures: Q, t_C
Constants: R_{max}, M

INITIALIZE():
  for $c \in C, a \in A, o \in O$:
    $t_C(s,a,o) = 0$
  for $s \in S, a \in A$:
    $Q(s,a) = R_{max}$

UPDATE(s, a, s'):
  $o = s' - s^{\dagger}$
  $t_C(\kappa(s),a,o) = t_C(\kappa(s),a,o) + 1$
  for $s \in S$:
    for $a \in A$
      $Q(s,a) = R_{max}$
  repeat until $Q(s,a)$ stops changing:
    for $s \in S$:
      for $a \in A$:
        $z = \sum_{o \in O} t_C(\kappa(s),a,o)$
        if $z \geq M$
          $Q(s,a) = r(s,a) + \gamma \sum_{o \in O} \frac{t_C(\kappa(s),a,o)}{z} \times \max_{a' \in A} Q(\eta(s,o),a')$

† This subtraction is a transformation expressing s' in the coordinate frame of s.

Figure 3. The RAM-Rmax algorithm.

5 Experiment

To quantify the benefits of using image segmentation for better performance, experiments were performed in a real world robotic environment.

5.1 Experimental Setup

For our experiment, we ran a LEGO® Mindstorms NXT (see Figure 1) on a multi-surface environment. This domain, shown in Figure 4, consisted of a highly variable region comprised of rocks embedded in wax and a more deterministic carpeted area. The goal was for the agent to begin in the start location (indicated in the figure by an arrow) and end in the goal without going outside the environmental boundaries. The rewards were -1 for going out of bounds, $+1$ for reaching the goal, and -0.01 for taking an action. Reaching the goal and going out of bounds ended the episode and resulted in the agent getting moved back to the start location.

One difficulty of this environment is the difference in dynamics models. Figure 5 shows the outcomes observed by the robot on both the rock and carpet surfaces. The center of the circle represents the starting location of the robot. The dashed lines indicate the angle (in degrees) and distance (in pixels) of displacement. From left to right, this figure shows the outcomes of the left turn, go forward, and right turn actions on for the rock (top) and carpet (middle) surfaces. The bottom row shows the same outcomes as above, but combines the terrains to demonstrate the amount of noise that is introduced when the terrains are not distinguished. Some actions, such as turning right on rocks, are more sparse than others due to the number of times that an action was taken during exploration.

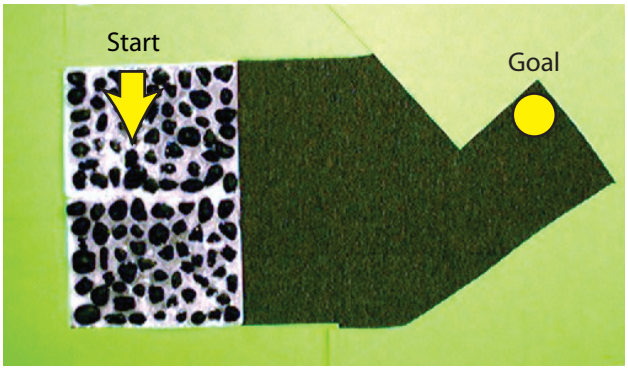
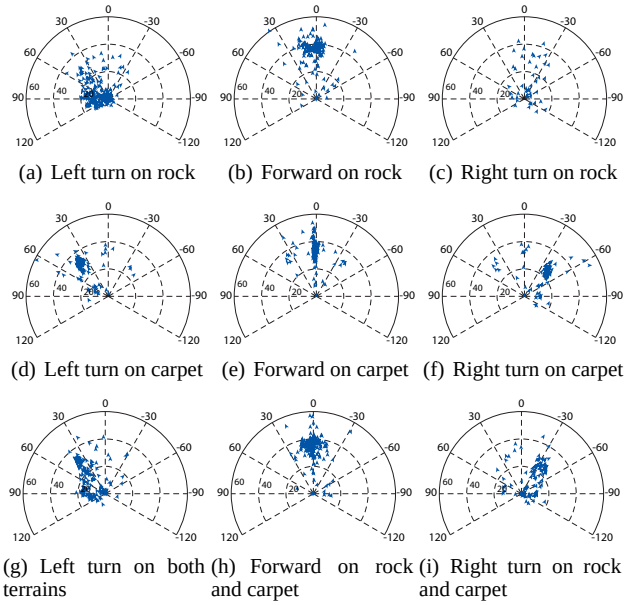


Figure 4. Image of the environment. The start location and orientation is marked with an arrow. The goal location is indicated by the circle. Green pieces of poster board are shown here marking the boundaries of the environment.



(a) Left turn on rock (b) Forward on rock (c) Right turn on rock

(d) Left turn on carpet (e) Forward on carpet (f) Right turn on carpet

(g) Left turn on both terrains (h) Forward on rock and carpet (i) Right turn on rock and carpet

Figure 5. Outcomes learned by the robot for different actions and surfaces.

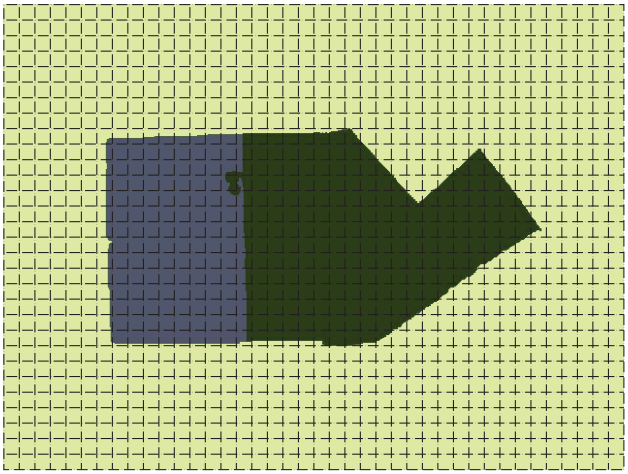


Figure 6. Resulting discretized segmented image from EDISON of the environment showing two different surface types. Several states were mislabeled due to the image processing algorithm, but these mislabelings did not harm the results.

Due to the close proximity of the goal to boundary, the agent needs an accurate dynamics model to reliably reach the goal. To make this task even more difficult, the actions were limited to going forward, turning left, and turning right. Not allowing the agent to move backwards increased the need for the agent to accurately approach the goal reliably. For example, if the robot enters the narrow portion of the environment facing away from the goal, it is not able to turn around without going out of bounds. As such, a robot with an inaccurate transition model would be likely to think this task is impossible.

For the experiments, we compared the RAM-Rmax and fitted Q-learning [7] algorithms with and without image segmentation. We would have liked to run the Rmax algorithm for a comparison, but it was not plausible to do so due to the robot's battery life—too much experience was needed to train the algorithm. All algorithms were informed of the reward function—it did not need to be learned. The algorithm with no image segmentation learns one action model for the entire domain. Figure 6 shows the results of the EDISON image-segmentation engine when fed in the image of the world and the minimum region to segment. Recall that the minimum region to segment was specified to be the number of pixels that the robot occupies in the image, which for this experiment was 4000 pixels.

For both agents, the world was discretized to a forty by thirty by ten state space instead of the camera's full resolution of 640 by 480 by 360 degrees of orientation. This coarse discretization was used to limit the number of states that the robot could occupy at once. Lastly, each algorithm had the value of M set to ten, which was determined after informal experimentation.

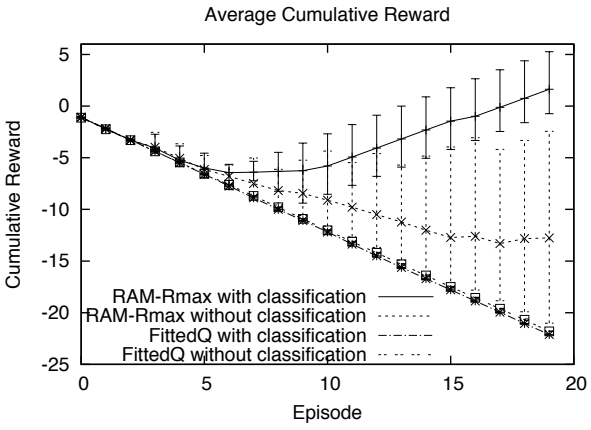


Figure 7. Graph of the RAM-Rmax and fitted Q-learning algorithms' average cumulative reward with and without terrain classification via image segmentation.

6 Results

Figure 7 shows the average performance and standard deviation of the RAM-Rmax and fitted Q-learning algorithms with and without image segmentation over five runs of twenty episodes. When RAM-Rmax used image segmentation to determine the number of surface types in the environment, RAM-Rmax reached the goal in 61% of

the episodes as opposed to 22% of the episodes when using the assumption that all surfaces were the same. This difference is shown in greater detail when looking specifically in the last 10 episodes after some learning had taken place. Narrowed to these instances, the success rates are 96% and 34%, respectively. Fitted Q-learning was not able to reach the goal in any of the runs with or without the image segmentation. Doubling the number of episodes to 40 in a run also did not result in any positive reward. Indeed, published results with this algorithm suggest hundreds or thousands of episodes are often needed.

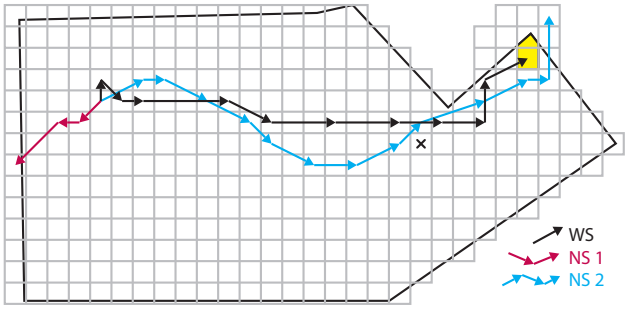


Figure 8. Diagram showing paths learned by the algorithm with segmentation (WS) and with no segmentation (NS) en route to the goal (shown in yellow). NS1 demonstrates a sample path in which the agent judged the goal as unreachable and so minimized negative reward by exiting the environment quickly. NS2 shows a sample path in which the agent's inaccurate model causeed it to miss the goal. "X" marks the state used for the example in Section 7.

The figure also shows a great variation in the performance of the RAM-Rmax algorithm when it did not use image segmentation. The reason for this difference is the variability in the dynamics model that the agent learned for each run. By learning varying dynamics models, the chosen path of the same agent changed drastically between runs as shown Figure 8. In two of the five runs, the agent thought that it was valuable to navigate towards the goal from early in the run. However, the dynamics model learned was noisy, and the agent would accidentally drive out of bounds when approaching the goal. In the other half of the runs, the agent did not think that it was possible to reach the goal based on its learned dynamics model, and, therefore, would drive out of the environment as quickly as possible to minimize negative reward.

In contrast, the agent that used image segmentation learned that the rocky surface was unpredictable, but that the carpet surface allowed for consistent actions. Once these two surfaces were learned, the agent was able to arrive at the goal reliably, seldom over-shooting of the goal, as shown in Figure 8.

7 Discussion

The two RAM-Rmax algorithms outperformed both of the fitted Q-learning algorithms due to the efficiency with which they use experience data. RAM-Rmax with and without image segmentation was able to use its learned model to generalize between states because of the similarity of the dynamics in these states. Since fitted Q-learning does not model the environment, its generalization ability was limited to exploiting local consistency in the value function. Throughout the twenty episodes, neither of the fitted Q-learning algorithms had been able to take advantage of any structure and were still active exploring to learn values. As such, they had learned very little and ended up going out of bounds every time.

The performance discrepancy of the two RAM-Rmax algorithms can be better explained when looking at the learned value function of the two algorithms. For example, when using the RAM-Rmax algorithm with image segmentation, the average expected reward of a state near the goal ($X = 25$, $Y = 15$, $\theta = 0$, marked with an "X" in Figure 8) was 0.450 with a standard deviation of 0.194 in comparison with the algorithm without image segmentation which on average calculated the value of the same state to be 0.1782 with a standard deviation of 0.373. These values vary because of the dynamics models learned. The algorithm without image segmentation performed as if the goal was on the rocky surface. By increasing the amount of context the RAM-Rmax algorithm uses to distinguish the dynamics, it is able to better model its environment.

However, there is a limit to how much improvement additional context can add. If the terrain classifier were to have found three types instead of two, the agent might have been able to model the dynamics of when its front wheels were on one surface and its back wheel on another. If the agent declared each rock its own surface, the agent would be able to model its likelihood of getting stuck on that rock. The problem with this approach is scalability. The more surface types the learner recognizes, the less it generalizes and the more exploration it needs to do. In the limit, as the number of types approaches the number of states, this algorithm becomes equivalent to (non-generalizing) Rmax.

By scaling the number of terrain types, an algorithm implicitly assumes information about the structure of the environment. At one end of the scale, the assumption is that there is one class that all the world adheres to as in road following. At the other, each state has its own idiosyncrasies and, therefore, should be modeled individually.

8 Conclusions and Future Work

Optimal decisions for path planning require accurate models for predicting the outcomes of actions. In this paper, we examined a method for building accurate models by partitioning learning experiences according to a set of automatically extracted terrain classes. We found that the resulting dynamics models led to a utility-based path planning system that learned quickly to make effective decisions. Compared to approaches that overgeneralize (estimate a single transition model) or undergeneralize (estimate separate models for each state), this approach makes effective use of experience and succeeded at reaching the goal location at a significantly higher rate in our experiments.

Currently, the system is dependent on the sensory features correlating well with differences between terrains. In future work, we will explore using feature selection to determine which of the robot's multiple sensors best predict terrain characteristics. By doing so, the agent could learn when to use, for example, ladar, IR, and satellite imagery to best estimate terrain features that matter for predicting action outcomes.

Acknowledgment

This material is based upon work supported by the National Science Foundation under grants ITR IIS-0325281 and DGE 0549115.

REFERENCES

[1] Ronen I. Brafman and Moshe Tennenholtz, 'R-MAX—a general polynomial time algorithm for near-optimal reinforcement learning', *Journal of Machine Learning Research*, **3**, 213–231, (2002).

[2] James Bruce, Tucker Balch, and Manuela Veloso, 'Fast and inexpensive color image segmentation for interactive robots', in *Proceedings of IROS-2000*, Japan, (October 2000).

[3] James Bruce and Manuela Veloso, 'Fast and accurate vision-based pattern detection and identification', in *Proceedings of ICRA'03, the 2003 IEEE International Conference on Robotics and Automation*, Taiwan, (May 2003).

[4] Christopher M. Christoudias, Bogdan Georgescu, and Peter Meer, 'Synergism in low level vision', in *ICPR '02: Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 4*, p. 40150, Washington, DC, USA, (2002). IEEE Computer Society.

[5] J. Crisman and Chuck Thorpe, 'UNSCARF, a color vision system for the detection of unstructured roads', in *Proceedings of IEEE International Conference on Robotics and Automation*, volume 3, pp. 2496–2501, (April 1991).

[6] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski, 'Self-supervised monocular road detection in desert terrain', in *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, (August 2006).

[7] Damien Ernst, Pierre Geurts, and Louis Wehenkel, 'Tree-based batch mode reinforcement learning', *J. Mach. Learn. Res.*, **6**, 503–556, (2005).

[8] Andrew Moore Justin Boyan, 'Generalization in reinforcement learning: Safely approximating the value function', in *Neural Information Processing Systems 7*, eds., G. Tesauro, D.S. Touretzky, and T.K. Lee, pp. 369–376, Cambridge, MA, (1995). The MIT Press.

[9] Rudolph Emil Kalman, 'A new approach to linear filtering and prediction problems', *Transactions of the ASME–Journal of Basic Engineering*, **82**(Series D), 35–45, (1960).

[10] Michael J. Keans and Satinder P. Singh, 'Near-optimal reinforcement learning in polynomial time', *Machine Learning*, **49**(2–3), 209–232, (2002).

[11] Bethany R. Leffler, Michael L. Littman, and Timothy Edmunds, 'Efficient reinforcement learning with relocatable action models', in *AAAI-07: Proceedings of the Twenty-Second Conference on Artificial Intelligence*, pp. 572–577, Menlo Park, CA, USA, (2007). The AAAI Press.

[12] Bethany R. Leffler, Michael L. Littman, Alexander L. Strehl, and Thomas J. Walsh, 'Efficient exploration with latent structure', in *Proceedings of Robotics: Science and Systems*, Cambridge, USA, (June 2005).

[13] Dean A. Pomerleau, 'ALVINN: An autonomous land vehicle in a neural network', 305–313, (1989).

[14] Christopher Rasmussen, 'Combining laser range, color, and texture cues for autonomous road following', in *IEEE International Conference on Robotics and Automation*, (2002).

[15] Alexander A. Sherstov and Peter Stone, 'Improving action selection in MDP's via knowledge transfer', in *Proceedings of the Twentieth National Conference on Artificial Intelligence*, (July 2005).

[16] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1998.

[17] Carl Wellington, Aaron Courville, and Anthony (Tony) Stentz, 'Interacting Markov random fields for simultaneous terrain modeling and obstacle detection', in *Proceedings of Robotics: Science and Systems*, (June 2005).