

Fast Online Learning of Control Regime Transitions for Adaptive Robotic Mobility

Brian Yamauchi

Abstract—We introduce a new framework, Model Transition Control (MTC), that models robot control problems as sets of linear control regimes linked by nonlinear transitions, and a new learning algorithm, Dynamic Threshold Learning (DTL), that learns the boundaries of these control regimes in real-time. We demonstrate that DTL can learn to prevent understeer and oversteer while controlling a simulated high-speed vehicle. We also show that DTL can enable an iRobot PackBot to avoid rollover in rough terrain and to actively shift its center-of-gravity to maintain balance when climbing obstacles. In all cases, DTL is able to learn control regime boundaries in a few minutes, often with single-digit numbers of learning trials.

I. INTRODUCTION

FOR the Dynamo Project, we have developed techniques to allow robots to rapidly adapt their mobility behaviors to different environments. We have formulated Model Transition Control (MTC) as a framework for modeling nonlinear control problems as sets of linear control regimes linked by nonlinear transitions. We have developed Dynamic Threshold Learning (DTL), a real-time, online learning algorithm that enables robots to quickly learn the transitions between control regimes.

We have applied MTC and DTL to control problems in both simulation and the real world. In simulation, we have demonstrated that DTL can be used to enable a high-speed wheeled vehicle to learn to prevent understeer and oversteer during aggressive cornering maneuvers. In the real world, we have used DTL to enable an iRobot PackBot to learn how to avoid rolling over in rough terrain. We have also used DTL to enable a PackBot to learn how to actively shift its center-of-gravity to climb over tall obstacles.

II. DYNAMIC THRESHOLD LEARNING

MTC represents control regimes using a *state-action map* (SAM) that represents possible robot states and actions and the corresponding control regimes. Fig. 1 shows a simple SAM for braking for a wheeled vehicle. This SAM has one state axis (velocity), one action axis (braking effort), and two control regimes. In the GRIP regime (green) all tires maintain their grip on the road. In the SKID regime (red) one or more tires skid. The task of this control system is to keep the vehicle within the GRIP regime.

The SAM for an MTC controller can be designed by hand,

Manuscript received September 16, 2011. This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Grant W91CRB-11-C-0049.

Brian Yamauchi is with iRobot Corporation, Bedford, MA, 01730 (phone: 781-430-3291; email: yamauchi@irobot.com).

but that approach is time-consuming and requires experimentation to determine the regime boundaries. DTL provides a means to automate this experimentation and learn the regime boundaries online as the robot interacts with the world.

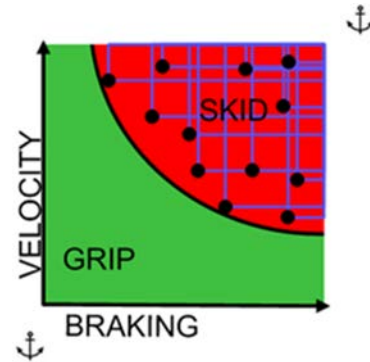


Fig. 1. A simple SAM for braking, with a GRIP control regime (green) and a SKID control regime (red). The anchor point for the GRIP regime is at zero velocity and braking. The anchor point for the SKID regime is at maximum velocity and braking. The black circles represent individual observations of the SKID regime, and the blue lines delineate the corresponding region that is inferred to be part of the SKID regime.

DTL makes use of a priori domain knowledge to accelerate the learning process. Rather than attempting to learn the entire control problem from scratch, DTL is initialized with qualitative knowledge about the problem domain. DTL then rapidly learns the quantitative control parameters to achieve the desired behavior.

DTL represents qualitative knowledge about the control problem in terms of *anchor points*. Each control regime has a single anchor point A_{REGIME} , which represents the point in state-action space that is most likely to produce the corresponding control regime. The anchor point for the GRIP regime (A_{GRIP}) is at zero velocity and zero braking effort. The anchor point for the SKID regime (A_{SKID}) is at maximum velocity (v_{max}) and maximum braking ($brake_{max}$).

Given an observation that the robot is in state S_i and takes action A_i , resulting in control regime R , we infer that all points (S_j, A_j) that are closer to the anchor point A_R along every axis are also part of regime R . For example, if DTL observes that the vehicle skids when it is driving 100 kph and applies 50% braking effort, DTL would then assume that skidding would also occur at 120 kph and 70% braking effort (or any other combination of a higher speed and a greater braking effort).

DTL assumes that control regimes are contiguous and that a line can be drawn from the anchor point of any regime to

any other point within the regime without crossing into another regime. We believe that this will not be a major limitation in most real world problems. For example, it seems unlikely (though not impossible) that skidding would occur from braking with 60% effort and 70% effort, but not from 65% effort.

Fig. 2 shows a block diagram for an MTC controller using DTL. An *exploratory controller* provides raw commands based on sensor inputs, which are used to explore the state-action space. For example, this could be a PID controller or a simple set of reactive rules. Alternately, a human operator can provide teleoperation inputs that serve the role of the exploratory controller. The exploratory controller provides aggressive responses that are likely to result in both desirable and undesirable control regimes.

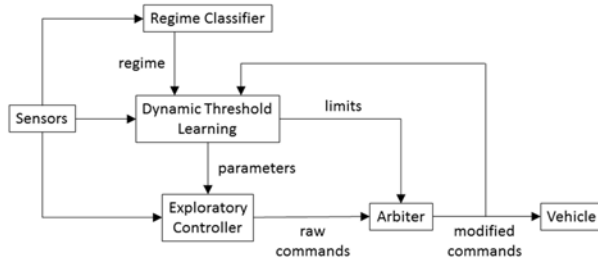


Fig. 2. Block diagram for an MTC controller using DTL.

If the DTL module predicts that a command from the exploratory controller will result in an undesirable control regime, then the DTL will modify the output in one of two ways. First, it can modify the parameters in the exploratory controller so that it will only generate outputs that remain within the desired control regime. Second, it can apply explicit limits to the command outputs via the arbiter.

The result is that an MTC-controlled robot will initially behave very aggressively, and this will result in the robot experiencing undesirable control regimes. DTL will quickly learn which state-action combinations lead to undesirable control regimes and will modify the control system so that it no longer directs the robot into these regimes.

III. SIMULATED HIGH-SPEED VEHICLE CONTROL

For our first experiments with DTL, we used The Open Race Car Simulator (TORCS), an open-source, high-speed race vehicle simulator. TORCS is available online at <http://torcs.sourceforge.net>. We chose TORCS because it provides physical modeling of high-speed wheeled vehicles, including tire-surface interaction for different surface materials and geometries, aerodynamic drag, and handling parameters for different vehicles. TORCS also provides an easy-to-use interface for implementing autonomous driving behaviors and linking these behaviors with the simulator.

We developed an MTC-based dynamic stability control (DSC) system for simulated TORCS vehicles to prevent understeer and oversteer. Conventional DSC systems are only capable of reacting to understeer and oversteer once they occur. These systems detect the difference between the actual and commanded yaw rates and then apply brakes at

individual wheels and/or reduce the throttle to reduce understeer/oversteer. Our MTC-based DSC system predicts which state-action combinations will result in understeer or oversteer. It then modifies the steering and throttle commands to prevent understeer and oversteer from occurring in the first place.

A. Exploratory Controller

The exploratory controller attempts to steer the car toward the center of the current track segment while maintaining a desired speed. The steering control rule is:

$$\theta_{steer} = (\theta_{track} - \theta_{car}) - \frac{y_{car}}{w_{track}} \quad (1)$$

where θ_{steer} is the steering command, θ_{track} is the track orientation in world coordinates, θ_{car} is the vehicle heading in world coordinates, y_{car} is the lateral position of the car relative to the current track segment, and w_{track} is the width of the current track segment.

The speed control rule is:

$$a = k_v(v_{desired} - v_{current}) \quad (2)$$

where a is the acceleration command, k_v is the proportional control gain, $v_{desired}$ is the desired velocity, and $v_{current}$ is the current velocity. Initially, $v_{desired}$ is set to the maximum speed of the vehicle.

B. Preventing Understeer

We define an understeer threshold that is a function of speed, turn radius, and the friction coefficient of the track surface. This threshold specifies the maximum speed that the vehicle can drive for a given turn radius without risking understeer.

The maximum traction available is given by:

$$F = ma = \mu gm \rightarrow a = \mu g \quad (3)$$

where F is the force applied by the tires, m is the mass of the vehicle, μ is the coefficient of friction, and g is the acceleration due to gravity. The lateral acceleration during a turn is:

$$a = \frac{v^2}{r} \quad (4)$$

where a is the lateral acceleration, v is the vehicle velocity, and r is the turn radius. We predict understeer when the lateral acceleration exceeds the maximum traction:

$$\frac{v^2}{r} > \mu g = T_{understeer} \quad (5)$$

where $T_{understeer}$ is the understeer threshold, which is initialized to its maximum value.

The task of learning the SAM for the understeer prevention behavior is equivalent to learning $T_{understeers}$.

assuming the regime boundary is linear in the space defined by the state variable for the track radius and the action variable for velocity. In this SAM, the anchor point for the GRIP regime is at maximum r and zero v . The anchor point for the UNDERSTEER regime is at minimum r and maximum v .

Understeer is detected by the regime classifier when the absolute yaw rate $\dot{\theta}_{current}$ is less the absolute steering command θ_{steer} multiplied by a constant $k_{understeer}$ that is less than 1:

$$|\dot{\theta}_{current}| < k_{understeer} |\theta_{steer}| \quad (6)$$

If understeer is detected, but understeer is not predicted by (5), then $T_{understeer}$ is reduced to the current lateral acceleration:

$$T_{understeer} = \frac{v^2}{r} \quad (7)$$

If DTL predicts that the current desired velocity will exceed the understeer threshold for the turn radius of the current track segment,

$$\frac{v_{desired}^2}{r} > T_{understeer} \quad (8)$$

then DTL reduces the desired velocity to a value that is not predicted to cause understeer:

$$v_{desired} = \sqrt{r T_{understeer}} \quad (9)$$

C. Preventing Oversteer

We define an oversteer threshold as a function of steering and acceleration inputs. This threshold represents the maximum combination of steering and throttle inputs that can be commanded without triggering oversteer.

We predict oversteer when

$$k_{steer} |\theta_{steer}| + k_{accel} a > T_{oversteer} \quad (10)$$

where k_{steer} and k_{accel} are fixed coefficients for weighting the steering and acceleration inputs and $T_{oversteer}$ is the oversteer threshold, which is initialized to its maximum value.

The regime classifier detects oversteer when the absolute yaw rate $\dot{\theta}_{current}$ is greater than the absolute steering command θ_{steer} multiplied by a constant $k_{oversteer}$ that is greater than 1:

$$|\dot{\theta}_{current}| > k_{oversteer} |\theta_{steer}| \quad (11)$$

If oversteer is detected, but not predicted, then we reduce the oversteer threshold based on the current steering and acceleration commands:

$$T_{oversteer} = k_{steer} |\theta_{steer}| + k_{accel} a \quad (12)$$

If the steering and acceleration commands proposed by the exploratory controller are predicted to exceed the oversteer threshold:

$$k_{steer} |\theta_{steer}| + k_{accel} a + k_{margin} > T_{oversteer} \quad (13)$$

where k_{margin} is a fixed safety margin, then we reduce the acceleration command (if possible) to a value that is not predicted to cause oversteer:

$$a_{mod} = \min\left(\frac{T_{oversteer} - k_{steer} |\theta_{steer}| - k_{margin}}{k_{accel}}, 0\right) \quad (14)$$

where a_{mod} is the modified acceleration command. We do not reduce the steering command, because doing so would be the equivalent of inducing understeer and would reduce the maneuverability of the vehicle.

D. Experimental Results



Fig. 3. MTC DSC controller driving simulated Ferrari 360 Modena in TORCS on a twisting alpine road course. The inset at the top right corner of the screen shows the track layout, and the red dot on the track represents the vehicle's current position.

We tested the MTC DSC controller on a wide range of track types including speedways with long straights and sweeping turns that allow high speeds; complex road courses with tight corners that allow moderate speeds (Fig. 3); and small dirt tracks with tight corners that require slow speeds.

The MTC DSC controller quickly learned appropriate values for both $T_{understeer}$ and $T_{oversteer}$. The controller typically learned the parameters for a particular track in one or two laps and under three minutes. After these parameters were learned, the controller was able to completely avoid situations that would cause understeer and oversteer.

We experimented with tracks containing both asphalt (high friction) and dirt (low friction) surfaces. For these tracks, DTL maintained separate $T_{understeer}$ and $T_{oversteer}$ parameters for asphalt and dirt learned the correct parameters for the current terrain type. On a real vehicle, this would require a terrain classification system using sensors such as vision.

We also experimented with different vehicle types with widely varying performance and handling characteristics – from the Ferrari 360 Modena to a NASCAR stock car to a

Subaru WRX STI rally car to a dune buggy. The MTC DSC controller was able to learn the appropriate parameters to prevent understeer and oversteer for each of these vehicles.

IV. PACKBOT ROLLOVER PREVENTION

For our second task, we developed an MTC-based controller to prevent rollover using a real iRobot PackBot. The SAM for this task has a state variable for the robot's pitch angle and an action variable for the robot's speed. The anchor point for the NORMAL regime is at zero pitch and speed. The anchor point for the ROLLOVER regime is at maximum pitch and speed.

These experiments were performed using stock PackBot hardware. The PackBot's pitch and roll angles were determined using the robot's standard onboard fluid pitch-roll sensor. The PackBot's speed was determined using its wheel encoders.

The role of the exploratory controller is performed by a human operator. Initially, translation and rotation commands are sent unchanged to the robot. As the operator explores the space of robot actions, DTL observes which combinations of pitch and speed result in rollover and learns the boundaries between the NORMAL and ROLLOVER control regimes.

The regime classifier identifies the current regime based on the output of the roll sensor:

$$REGIME(\varphi) = \begin{cases} NORMAL & \text{if } \varphi \leq 90^\circ \\ ROLLOVER & \text{if } \varphi > 90^\circ \end{cases} \quad (15)$$

where φ is the robot's roll angle. This classifier would work equally well using the pitch angle.

DTL learns to predict rollover when the pitch angle exceeds a threshold function of velocity:

$$\theta \geq T_{rollover}(v_{current}) \quad (16)$$

where θ is the robot's pitch angle (0° = level, 90° = vertical), $T_{oversteer}$ is the rollover threshold function, and v is the robot's velocity.

DTL maintains a history of recent robot state-action pairs.

$$H = \{(\theta_0, v_0), (\theta_1, v_1) \dots (\theta_n, v_n)\} \quad (17)$$

where H is the state-action history, θ_i is the robot's pitch angle at i timesteps prior to the present and v_i is the robot's velocity at i timesteps prior to the present and n is the number of entries in the history. In our experiments, the timesteps were 0.1 seconds long and the history contained the most recent 5 state-action values.

If the robot rolls over and rollover was not predicted, then for each state-action pair in the history, DTL reduces the pitch threshold for the velocity and all greater velocities:

$$\forall(\theta_i, v_i) \in H: \forall v \geq v_i: T_{rollover}(v_i) = \min(\theta_i, T_{rollover}(v_i)) \quad (18)$$

If the current pitch angle exceeds the pitch threshold for the speed commanded by the operator

$$\theta_{pitch} \geq T_{rollover}(v_{desired}) \quad (19)$$

where $v_{desired}$ is the desired velocity sent by the operator, then the DTL module will reduce the speed command sent to the robot to the maximum speed that is not predicted to cause rollover:

$$v_{command} = \max(v) \mid \theta_{pitch} < T_{rollover}(v) \quad (20)$$

where $v_{command}$ is the velocity command that is sent to the robot.



Fig. 4. PackBot attempts to climb obstacle and starts to roll over backwards.

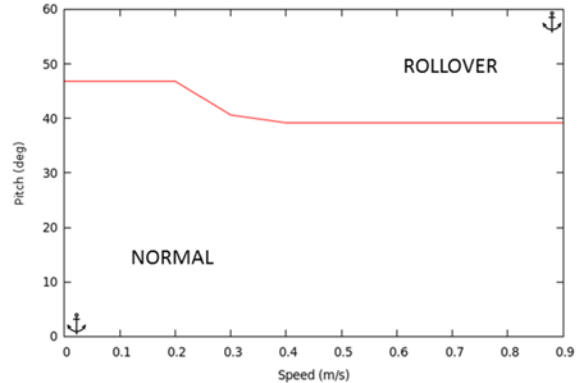


Fig. 5. SAM learned by DTL for the rollover prevention task. The NORMAL regime anchor point is at zero pitch and speed. The ROLLOVER regime anchor point is at maximum pitch and speed.

In each trial, we drove the PackBot forward at a fixed speed onto a tall obstacle (Fig. 4). As the robot drove forward, it reached a point where it rolled over backward. DTL then learned each state-action pair in the history that led up to the rollover event and lowered the pitch thresholds for the corresponding robot velocities. We repeated this procedure at three different speeds (0.25 m/s, 0.5 m/s, and 1.0 m/s). DTL was able to learn how to prevent rollover at 0.25 m/s after just 2 trials. The thresholds learned for 0.25 m/s were also sufficient to prevent rollover at 0.5 m/s but not 1.0 m/s. After 5 more trials, DTL learned thresholds for preventing rollover at all speeds up to 1.0 m/s.

Fig. 5 shows the SAM learned by DTL during a typical

rollover prevention learning experiment. This SAM indicates that if the robot ever exceeds a pitch angle of 47° , it should stop moving forward. However, if the pitch angle is between 39° and 47° , it can move forward safely at a slower speed.

V. PACKBOT ACTIVE WEIGHT SHIFTING

For our third task, we developed an MTC-based controller that used DTL to learn how to move the PackBot's manipulator arm to actively shift its center of gravity. The SAM for this task includes a state variable for the robot's pitch angle and an action variable for the rotation angle of the arm's shoulder joint. The anchor point for the NORMAL regime is at a pitch angle of zero and a shoulder angle of zero (arm horizontal). The anchor point for the ROLLOVER regime is at a pitch angle of 90° and a shoulder angle of 90° (arm vertical).

The role of the exploratory controller is performed by a human operator. For each trial, the arm is initially positioned with the shoulder joint in the vertical orientation (Fig. 6). The operator drives the robot forward at a fixed speed. As the robot climbs the obstacle, it reaches a point at which it tips backward and rolls over. By rotating the shoulder joint, the robot can shift its center-of-gravity forward and avoid rolling over backwards. The learning task is to determine the necessary arm angle to prevent rollover as a function of the robot's pitch angle.



Fig. 6. PackBot attempts to climb obstacle with shoulder joint in vertical orientation (90°).

For this task, the SAM is represented as an array of values where $SAM(\theta, \alpha)$ represents the control regime for the robot when its pitch angle is θ and its shoulder angle is α . Initially, all of the array values are set to NORMAL. When the regime classifier identifies that the robot has entered the ROLLOVER regime, DTL updates the SAM for all entries in the history.

$$\forall(\theta_i, \alpha_i) \in H: \forall \theta + \theta_{margin} \geq \theta_i, \alpha \geq \alpha_i: SAM(\theta, \alpha) = \text{ROLLOVER} \quad (21)$$

where θ_i is the robot's pitch angle at i timesteps prior to the present, α_i is the robot's shoulder angle at i timesteps prior to the present, and θ_{margin} is a safety margin. Whenever DTL detects a rollover event, it predicts that rollover will occur again at the same or greater pitch and the same or greater

shoulder angle.

As in the rollover prevention task, the DTL acts as a driver-assist behavior for weight shifting. DTL predicts that the robot will rollover when

$$SAM(\theta, \alpha) = \text{ROLLOVER} \quad (22)$$

where θ is the current pitch angle and α is the current shoulder angle. When DTL predicts rollover, it overrides the current operator command, stops the robot, and rotates the shoulder joint forward to the maximum angle for which it does not predict ROLLOVER.

$$\alpha_{command} = \text{MAX}(\alpha) \mid SAM(\theta, \alpha) = \text{NORMAL} \quad (23)$$

where $\alpha_{command}$ is the joint position command that is sent to the robot. Using this MTC-based controller, the operator can simply command the robot to move forward, and the controller will automatically rotate the arm forward to shift the center-of-gravity and prevent rollover (Fig. 7).



Fig. 7. MTC-based weight shifting controller moves arm forward to allow PackBot to climb over obstacle without rolling over backwards.

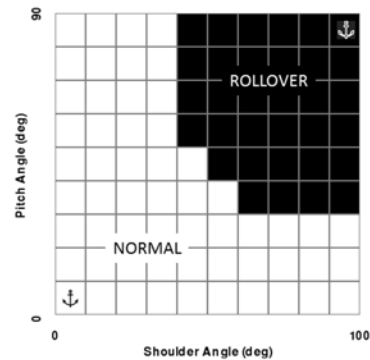


Fig. 8. SAM learned by DTL for the weight shifting task. The anchor point for the NORMAL regime is at zero pitch and shoulder angle. The anchor point for the ROLLOVER regime is at the maximum pitch and shoulder angle.

Fig. 8. shows a SAM learned by DTL for weight shifting. In this SAM, the pitch and shoulder axes are quantized at 10° intervals, where each grid cell corresponds to a single SAM array value. In our experiments, DTL was able to learn each SAM in 4 trials. A tradeoff exists between the resolution of the SAM grid and the number of trials required.

Finer resolutions allow for more precise identification of the regime boundary, but require more trials to learn an effective SAM.

VI. RELATED WORK

Reinforcement learning techniques [1], such as temporal difference learning [2] and Q learning [3], have been applied to a wide range of robot control problems, including robot juggling [4], obstacle avoidance and corridor following [5], quadruped obstacle negotiation [6] and gap jumping [7], helicopter control [8], and many others. Most previous research has been done offline or using discrete, rather than continuous state-spaces, but [4] and [8] are examples of real-time control learning in continuous state and action spaces.

DTL is similar to reinforcement learning in that it uses reinforcement in the form of undesirable regime detection to learn an appropriate control policy. Like most successful applications of reinforcement learning to robotics, DTL makes use of domain knowledge about the problem to reduce the search space so that learning can occur in a tractable amount of time [1].

DTL differs from previous reinforcement learning algorithms in modeling the state-action space as a set of linear control problems linked by nonlinear transitions. DTL requires additional a priori domain knowledge in terms of the control regimes that may occur and the state-action pairs (anchor points) that are most likely to generate these control regimes. DTL also requires the ability to identify the current control regime based on sensor inputs.

The payoff for these additional requirements is greatly accelerated learning. While many reinforcement learning algorithms require hundreds or thousands of trials and hours of real time to learn robotics tasks, DTL is able to learn highly dynamic control tasks in minutes, often with the number of required trials in the single digits.

However, our initial experiments have been conducted using low-dimensional state-action spaces. As with all learning techniques, DTL will need to address the curse of dimensionality. We believe the DTL will scale well due to the large amount of information that can be gleaned from a single observation and the use of domain knowledge in the form of anchor points.

DTL is also similar to adaptive fuzzy control [9] in terms of using heuristic a priori knowledge about the control problem and then modifying the initial control rules based on feedback from the environment. DTL differs both in the specific learning algorithm provided and in the focus on learning behaviors that must deal with abrupt discontinuities in control regimes, such as the transition that occurs from normal driving to snap oversteer.

One limitation of our approach is that the robot must be able to survive the undesirable control regimes experienced during learning or be cheap enough to be easily replaced. In the case of the PackBot learning rollover prevention and weight shifting, it was very useful that we had a rugged platform that could survive multiple falls. In the case of

high-speed vehicle control learning with full-sized vehicles, learning may need to occur in simulation first and then be transferred to the actual vehicle for testing.

VII. CONCLUSIONS AND FUTURE WORK

We have introduced MTC, a new framework for developing robot controllers, and DTL, a new algorithm for fast online learning of control regime thresholds. We have demonstrated that DTL can rapidly learn these thresholds in three different task domains: preventing understeer and oversteer for simulated high-speed vehicles, preventing rollover for the PackBot, and enabling the PackBot to use active weight shifting to climb obstacles that it would not be otherwise able to climb.

In future work, we plan to study how DTL scales to higher-dimensional state-action spaces. For example, we plan to develop rollover prevention for lateral as well as longitudinal rollover, and to develop weight shifting behaviors in three dimensions to maintain balance over rough terrain. We also plan to study how to implement “unlearning” to correct for noisy data or incorrect regime classifications.

We plan to apply MTC-based understeer and oversteer prevention to a real high-speed UGV based on a radio-controlled car platform that is capable of driving at speeds up to 27 m/s (60 mph). We also plan to learn additional driver assist behaviors to make it easier for operators to control UGVs at high speeds.

REFERENCES

- [1] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285, 1996.
- [2] R. S. Sutton, “Learning to predict by the methods of temporal differences,” *Machine Learning*, vol. 3, pp. 9-44, 1988.
- [3] C. Watkins, “Learning from delayed rewards,” Ph.D. dissertation, University of Cambridge, Cambridge, UK, 1989.
- [4] S. Schaal and C. G. Atkeson, “Robot juggling: An implementation of memory-based learning,” *Control Systems Magazine*, vol. 14, no. 1, pp. 57-71, 1994.
- [5] W. D. Smart and L. P. Kaelbling, “Effective reinforcement learning for mobile robots,” in *Proc. 2002 IEEE Int. Conf. on Robotics and Automation*, Washington, DC, 2002, pp. 3404-3410.
- [6] H. Lee, Y. Shen, C.-H. Yu, G. Singh, and A. Ng, “Quadruped robot obstacle negotiation via reinforcement learning,” in *Proc. 2006 IEEE Int. Conf. on Robotics and Automation*, Orlando, FL, 2006, pp. 3003-3010.
- [7] S. Schaal and C. G. Atkeson, “Learning control in robotics,” *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 20-29, 2010.
- [8] P. Abbeel, A. Coates, M. Quigley, A. Ng, “An application of reinforcement learning to aerobatic helicopter flight,” in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. Platt and T. Hoffman, Ed. Cambridge, MA: MIT Press.
- [9] K. Passino and S. Yurkovich, *Fuzzy Control*, Menlo Park, CA: Addison-Wesley Longman, 1994.