State Indexed Policy Search by Dynamic Programming

Charles DuHadway 5435537
Yi Gu 5103372

December 14, 2007

Abstract

We consider the reinforcement learning problem of simultaneous trajectory-following and obstacle avoidance by a radio-controlled car. A space-indexed non-stationary controller policy class is chosen that is linear in the features set, where the multiplier of each feature in each controller is learned using the policy search by dynamic programming algorithm. Under the control of the learned policies the radio-controlled car is shown to be capable of reliably following a pre-defined trajectory while avoiding point obstacles along its way.

1. Introduction

We consider the reinforcement learning problem of trajectory-following involving discrete decisions. Casting the problem as a Markov decision process (MDP), we have a tuple (S, A, $\{P_{sa}\}$, γ, R) denoting, respectively, the set of states, the set of actions, the state transition probabilities, the discount factor and the reward function, the objective is to find a policy set Π_{opt} that is optimal with respect to some choice of R. In cases where the system dynamics, R and Π are linear or linearizable, Π_{opt} can be readily solved for e.g. by linear quadratic regulators (LQR) [1]. Our current work however focuses on the case where such linearity does not hold, and therefore generalizes the class of trajectory-following problems to scenarios involving non-trivial environmental constraints.

We use a radio-controlled (RC) car as a test platform, and introduce non-linearities to R and Π through the need to make discrete decisions in trying to avoid a point obstacle along a pre-defined trajectory. We choose the policy search by dynamic programming (PSDP) algorithm as the means to finding Π_{opt}. This is because given a base distribution (indicating roughly how often we expect a good policy to visit each state) PSDP can efficiently compute Π_{opt} in spite of non-linear R and Π [2], hence our trajectory-following problem fits naturally into this framework.

In trajectory-following problems actions customized for local path characteristics are especially useful. Requisite to such exploitation however are: a) a non-stationary policy set i.e. $\pi \in \Pi$ changes as the car traverses the trajectory, b) an accurate correspondence between spatial position and the policy π used. However, the accuracy in this correspondence is inevitably degraded by run-time uncertainties. Consequently we chose for the policies π to be indexed over space instead of time increments, as under such a scheme the correspondence would be immune from the said uncertainties.

The goal of our work is to investigate the feasibility of employing PSDP in trajectory-following problems under a space-indexed scheme. More concretely, we aim to find a suitable parameterization of the system S, action space A, discount mechanism, reward function R and policy class Π, such that the RC car would reliably follow a pre-defined trajectory in the absence of obstacles, and avoid obstacles with some margin in their presence.

We will first present a space-indexed parameterization of the system, and then define the policy class and feature set. This leads on to the algorithm's implementation, starting with a brief definition of PSDP, followed by a section-by-section account of the choices made for the various functions, parameters and other algorithmic components. The results and accompanying analysis are provided in the final part of the paper.

2. System parameterization

Typically, a reasonable system parameterization for a vehicle could be

$$s_t = \begin{bmatrix} x_t & y_t & \theta_t & u_t & v_t & u_{t-1} & v_{t-1} \end{bmatrix}^T$$

where x_t and y_t are the Cartesian coordinates of the vehicle in the world coordinate system (WCS), θ_t the orientation of the car in the WCS, and u_t and v_t are the longitudinal and lateral velocities in the vehicle coordinate system (VCS). All values are with respect to the center of the vehicle, which is assumed to be a rectangle. The subscripts *t* and *t-1* indicate the time instants to which the values correspond. This parameterization is illustrated graphically in Figure 1.

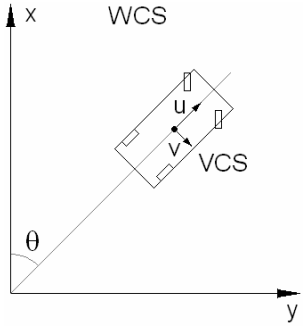


Figure 1. Time-indexed states

Note that time is not included as a state as the mapping of controller to the time instant it is used makes this implicit. Under the space-indexed scheme for a vehicle driving in $\mathbf{R}^2$ however, each controller is mapped to a subspace in $\mathbf{R}$ which we will call a "waypoint". We define each waypoint to be a subspace orthogonal to the trajectory tangent at the point where the waypoint exists. This is illustrated in Figure 2.

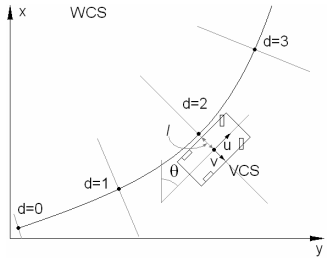


Figure 2. Space-indexed states

To localize the vehicle along a waypoint, we define l as the cross-track error between where the target trajectory intersects the waypoint and where the center of the vehicle intersects with the same waypoint. This gives a space-indexed system parameterization

$$s_d = \begin{bmatrix} t_d & l_d & \theta_d & u_d & v_d & u_{d-1} & v_{d-1} \end{bmatrix}^T$$

where t_d denotes the time at space index d along the pre-defined target trajectory, while θ_d, u_d and v_d are defined similarly as before except they now denote state values at a space index d. Note that the position of the vehicle is implicitly but completely specified.

The simple bicycle model $\dot{s} = f(s,a)$ is used to describe the vehicle dynamics, where $a \in$ A is an action. State propagation from index d to d+1 is achieved by first computing Δt such that at $t_{d+1}=t_d+\Delta t$ the vehicle is precisely at waypoint d+1, then states at d+1 are computed by propagating s_d forward through $f(s,a)$ by exactly Δt. For brevity, states capturing historic values of the steering angle used to model the steering dynamics are not shown.

3. Policy class

The ultimate performance of the algorithm depends critically on the policy class, which is defined by both its format and the features that constitutes it. As mentioned previously, the class of policy employed is non-stationary i.e. $\prod(\Theta)=(\pi_1(\theta_1),\ldots,\pi_{D-1}(\theta_{D-1}))$, where $\Theta=(\theta_1,\ldots,\theta_{D-1})$, so that we have a one-to-one mapping between each index d and stationary policy $\pi_d(\theta_d)$.

All $\pi_d(\theta_d)$: S→A come form the same policy class, and for simplicity we chose each $\pi_d(\theta_d)$ to be linear in the feature set i.e.

$$\pi_d(\theta_d) = \theta_d{}^T \cdot g(s_d)$$

$$\theta_d = \begin{bmatrix} \theta_{d,1} & \theta_{d,2} & \theta_{d,3} & \theta_{d,4} \end{bmatrix}^T$$

$$g(s_d) = \begin{bmatrix} g_1 & g_2 & g_3 & g_4 \end{bmatrix}^T$$

g_1 is simply the cross-track error l_d, g_2 is the yaw of the car with respect to the target trajectory tangent at d; for example if the orientation of the vehicle at d in WCS is 40° while the trajectory tangent at d in WCS is 37°, then g_2=3°. Together g_1 and g_2 keep the vehicle close and parallel to the target trajectory.

g_3 is the anticipated clearance surplus from the obstacle. For instance if l_d=1.5 cm (1.5 cm to the right of the trajectory looking in the direction of travel) as the car is approaching a point obstacle some distance ahead situated 5 cm to the left of the trajectory, and the desired clearance is 20 cm to the right of the obstacle, then g_3=(1.5+5)-20=-13.5 cm i.e. the car needs to be 13.5 cm further to the right to avoid colliding with the obstacle. As a conservative measure, g_3's value is offset by 5 cm so that g_3=-25 cm at -20 cm real clearance, increasing to -5 cm at 0 cm real clearance, beyond which its absolute value decays exponentially while remaining negative.

g_3 is intended to steer the car off the target trajectory away and from an obstacle. g_3=0 when no obstacle is within the control horizon. We allow the car to re-evaluate which side to pass an obstacle on at each step, the rule being that if the obstacle is currently to the left of the car, then pass it on the right and vice versa, and g_3 is computed accordingly.

g_4 is the obstacle conditioned yaw. Its value is non-zero only when the car is approaching an obstacle and is steering in a direction that would reduce the clearance. In such cases, g_4 is yaw of the trajectory tangent as measured in the VCS. This feature is included to preserve any clearance achieved via g_3, since an existing clearance would tend to make g_3 small, and a controller without g_4 would be dominated by g_1 and g_2.

This concludes the definition of the policy class, and we note in particular that the policy class is clearly non-linear and cannot be easily cast into a form suitable for a linear solver.

4. Algorithm implementation

4.1 Definition

We start by considering a trajectory that spans D space indices and a policy class as defined in 3, we define the value function $V(s)$

$$V_{\pi_t,\ldots,\pi_{D-1}}(s) \equiv \frac{1}{D} E\left[\sum_{i=d}^{D-1} R(s_i) \mid s_d = s; (\pi_d,\ldots,\pi_{D-1})\right]$$
$$\equiv R(s) + E_{s\sim P_{s\pi_d(s)}}\left[V_{\pi_{d+1},\ldots,\pi_{D-1}}(s)\right]$$

where $s \sim P_{s\pi_d(s)}$ indicates that the expectation is with respect to s drawn from the state transition probability $P_{s\pi_d(s)}$. The PSDP algorithm takes the form:

for $d = D\text{-}1, D\text{-}2,\ldots,0$

$$Set\ \pi_d = \arg\max_{\pi\in\Pi} E_{s\sim\mu_d}[V_{\pi,\pi_{d+1},\ldots,\pi_{D-1}}(s)]$$

where we assume we are given a sequence of base distributions μ_d over the states. PSDP is thus a policy iteration algorithm that yields an optimal π_d for every space index. The dynamic programming part is evident in that the solution for μ_d starts from the end of the trajectory towards the beginning.

4.2 Initialization

We now turn our attention to μ_d, and note that we do not know the set of base distributions (μ_0, μ_1,…, μ_{D-1}) over a pre-defined trajectory. However if we had a controller that could simultaneously track a target trajectory and avoid obstacles, then by running the RC car over the trajectory using that controller a large m number of times, we would see a representative distribution of s at d across all m trials. In this way we have effectively sampled from μ_d without knowing the explicit form of μ_d; the problem is that if we had such a controller, we wouldn't have needed to run PSDP to begin with. One way around this catch 22 is to start PSDP by sampling from a poor approximation of μ_d i.e.

1. *Run the RC car over the target trajectory without obstacles m times using a suboptimal yet reasonable controller. Add obstacles to each of the m trajectories.*
2. *Derive a set of policies Π using PSDP based on these m samples.*
3. *Run the RC car over m trajectories with obstacles using Π.*
4. *Iterate over steps 2 and 3 until Π converges, at which point we deem that we have been sampling from a good approximation of (μ_0, μ_1,…, μ_{D-1}).*

In our implementation, the suboptimal controller is derived in two steps. First we compute a noiseless state trajectory using a heuristically-tuned PI controller over the target trajectory with no obstacles. The resulting state sequence is used as the starting point for finding a LQR controller using differential dynamic programming (DDP). We finally obtain m samples by using the LQR controller in step 1 above.

Graphically, the PSDP algorithm as defined in 4.1 and initialized in 4.2 appears as in Figure 3.

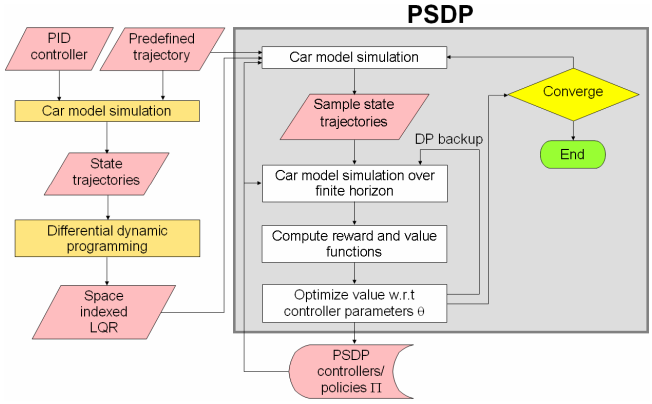


Figure 3. PSDP algorithmic flow

4.3 Action space

The control of an RC car involves steering and throttle. If we choose to drive the vehicle at constant velocity, then our action space is simplified to comprise of only steering. The steering angle range of the RC car is a continuous interval from -28° (full lock left) to 28° (full lock right). In order to keep the calculation of $V(s)$ tractable however this interval is discretized to 10 points:

A={-28°, -21.78°,…, 21.78°, 28°}.

4.4 Transition probability

Two mechanisms influence the transition from state s_d to s_{d+1} viz. system dynamics and noise. The former is captured by $f(s,a)$, while the noise is modeled as zero-mean Gaussian measurement noise. Consequently, our transition probability is

$$s_{d+1} \sim N\left(s_d + f(s_d, a_d)\Delta t, \sigma^2\right)$$

where σ^2 is chosen to be consistent with variance of state measurements obtained from the RC car's Kalman filter.

4.5 Discount factor

Recall that $V(s)$ is the expected sum of R(s) over distance, and terms for which $\gamma^h R(s_{d+h-1}) < \varepsilon$ for some small ε are ignored as they have negligible effect (those more than h-1 indices down the length of the trajectory from d). If γ is close to 1 then the learned policies are anticipatory of the future and tend to be more globally optimal. However the closer γ is to 1 the larger p becomes for a given ε, with a corresponding longer computation time. If γ is close to 0, the computation time would be shorter, but

the policies would also be less sensitive to future high-cost consequences to current actions, making them only locally optimal.

We found that the geometrically decaying weighting envelope γ imposes could not yield a satisfactory balance between global-optimality and computation time for any value between 0 and 1. Consequently we sacrificed envelope continuity and instead employed a rectangular weighting envelope. $V(s)$ is thus the sum of a fixed number h of reward functions each of which is weighted equally.

4.6 Reward function

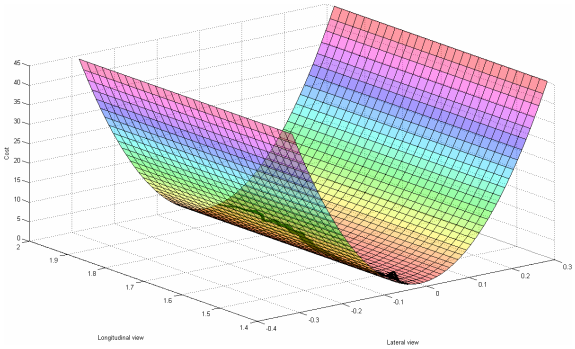
The desired behaviors of the car include: a) close tracking of the target trajectory, b) reliable avoidance of obstacles, and c) use of steering actions that are realizable i.e. limited bandwidth actuation. Since it is more natural to treat these attributes in the negative e.g. we do *not* wish the car to be far from the target trajectory, R(*s*) will henceforth refer to a cost function instead of a reward function. The problem remains unchanged, only now we wish to minimize R(*s*) instead of maximizing it. From these a 4-component cost function was conceived:

$$R(s_d) = \begin{cases} b_1 l^2 + b_2 \dot{q}^2 & \text{if no collision} \\ b_2 \dot{q}^2 + b_3 \left(1 - \frac{c}{c_1}\right) + b_4 & \text{otherwise} \end{cases}$$

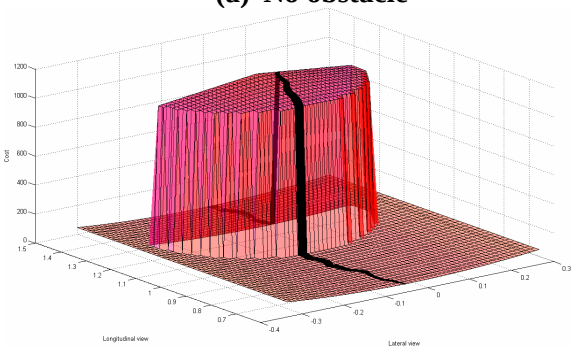
where $\dot{q}$ is derivative of steering angle, c is the distance between the center of the car and a point obstacle, and c_1 is the desired minimum clearance to an obstacle. If $c < c_1$, then a collision is considered to have occurred – this effectively treats the car as a circular object and was a conscious and conservative choice.

The term involving l penalizes deviation from the target path, the term involving $\dot{q}$ penalizes rapid changes in steering, the constant b_4 penalizes a collision and the term involving c and c_1 enhances the repulsion of the car away from the obstacle during learning.

Figure 4 is a visualization aid to R(*s*), which has discrete regimes and is clearly non-linear. Note in particular the expected parabolic appearance of the b_1 contribution, the half cylinder and the slight half cone atop of it in b) attributable to the b_4 and b_3 terms respectively. The weighting parameters b_1 to b_4 were chosen experimentally.



(a) No obstacle



(b) Collision

Figure 4. Cost function visualization

Note that at this point, we have also fully specified the MDP in the context of our trajectory following problem, having defined the tuple (S, A, $\{P_{sa}\}$, γ, R).

4.7 Optimization

Referring to Figure 3, in each iteration of PSDP we have m sample state trajectories each containing D points. For a DP backup at d:

1. *We start with s_d on the first sample trajectory, note the random number seed, take action $a_1 \in A$.*
2. *Compute $s_{d+1},\ldots,s_{d+h-1}$ (h as defined in 4.5), the corresponding costs R(s) and $V_{a_1}(s)$ (the subscript a_1 indicates unique correspondence to action a_1).*
3. *Re-seed the random number generator with the value noted in 1, take action $a_2 \in A$ and repeat step 2.*
4. *Repeat steps 2 and 3 for actions a_3 to a_{10}.*
5. *Repeat steps 1 to 4 for sample trajectories 2 to m.*

At the end of the 5 steps, we have a record of how $V(s)$ varies with action at index d for each of the m samples. We then use the coordinate descend algorithm with dynamic resolution to find the θ that minimizes the expected value i.e. we solve

$$\pi_d(\theta_d) = \underset{\pi_d(\theta_d) \in \Pi(\Theta)}{\arg\min}\, E_{s \sim \mu_d}[V_{\pi, \pi_{d+1}, \ldots, \pi_{d+h-1}}(s)]$$

To expedite the numeric search, the starting point is chosen to be the least squares estimate $\theta_{init} = (B^T B)^{-1} B^T b$ where b is the vector of value-minimizing steering angle

specific to each trajectory and B is a matrix containing the feature values associated with each trajectory. This is based on the observation that the least squares solution of θ to individual steering actions that minimize individual $V(s)$ is often close to the solution that minimized the aggregate.

5. Results and discussion

5.1 Performance

The learned controllers performed well in both simulation and the physical system. Example trajectories from both are shown in Figure 5. To quantitatively evaluate the performance of a learned controller we used both the cost function and obstacle collision rate. The controller shown in Figure 5 had a collision rate of 3.3%.

The target and actual trajectories are shown in blue and black respectively. Obstacles are shown as double red circles. The inner circle denotes the closest the physical car could pass by the obstacle without collision if perfectly aligned (exactly tangent to the obstacle). The outer circle denotes the farthest the car could be and still manage to hit the obstacle given worst case alignment (exactly orthogonal to the obstacle). Note the controller perform wells despite the obvious difference in sampling rates between the actual and simulated systems.

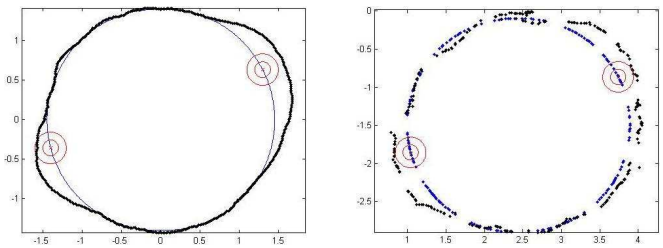


Figure 5. Simulated and physical trajectories

Circular trajectories were used to test the actual car system due to physical limitations in our tracking system. Similar performance was observed in simulation for both straight and oval trajectories and we expect our system to perform well over a large variety of trajectories.

5.2 Choosing a policy class and cost function

Choosing the appropriate policy class and cost function was non-trivial and required a series of tests to choose between several alternatives. Figure 6 shows the results of one such test comparing two cost functions: one with an accurate, bounding box collision test and one with a conservatively inaccurate bounding circle collision test. In this case the latter's performance far exceeded the former's.

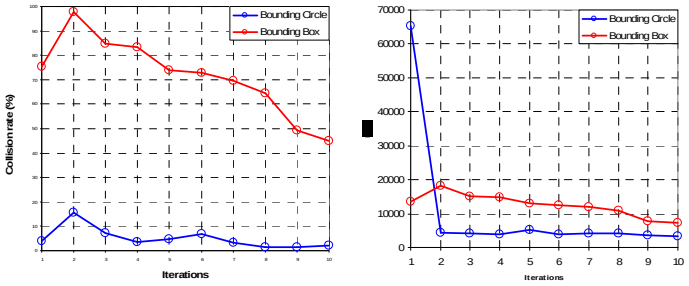


Figure 6. Bounding box vs. circle collision test

Similar tests were used to choose the specific feature set and cost function described above. Among other things we learned that the standard integral term was unnecessary for our learning task. We also found it important to use a cost function with no flat areas.

5.3 Conclusions

We have implemented a system that demonstrates the feasibility and effectiveness of employing PSDP in a state-indexed scheme to solve a reinforcement learning problem with a non-linear policy class and reward function. Our approach takes full advantage of PSDP's strengths while eliminating its dependence on accurate space-time correlation.

5.4 Next Steps

The obvious next step is a detailed comparison of state-indexed PSDP with time-indexed PSDP. We are confident that such a comparison will clearly demonstrate the superiority of state-based PSDP. This should be especially clear over long or non-uniform trajectories.

Acknowledgments

The authors wish to acknowledge valuable inputs from Andrew Ng and Adam Coates during the course of this research project.

References

[1] Abbeel P., Coates A., Quigley M., and Ng A. Y.. *An Application of Reinforcement Learning to Aerobatic Helicopter Flight*, in NIPS 19, 2007

[2] Bagnell J. A., Kakade S., Ng Y A., and Schneider J.. *Policy search by dynamic programming*, in NIPS 16, 2004.