Centre for Theoretical Neuroscience
Technical Report

TR name UW-CTN-TR-20102708-001
October 7th, 2010

Dynamic scaling for efficient, low-cost control of high-precision movements in large environments

Travis DeWolf

Centre for Theoretical Neuroscience, Waterloo, ON. http://compneuro.uwaterloo.ca/cnrglab

Dynamic scaling for efficient, low-cost control of high-precision movements in large environments

Travis DeWolf
University of Waterloo
Waterloo, ON

Abstract

This paper presents the dynamic scaling technique (DST), a method for control in large environments that dramatically reduces the resources required to achieve highly accurate movements. The DST uses a low resolution representation of the environment to calculate an initial approximately optimal trajectory and refines the control signal as the target is neared. Simulation results are presented and the effect of representation resolution on accuracy and computational efficiency is analyzed.

Keywords: Dynamic scaling technique; linear Bellman controllers; control theory; high precision movements

1 Introduction

Recent developments in optimal control theory have presented a new technique for solving Bellman's equation; by applying minimal constraints to the cost function, the problem of solving for the optimal cost-to-go can be reduced to a linear form (Todorov, 2009c). These Linear Bellman Controllers (LBCs) are able to solve offline for the optimal trajectory to a target from any state the solution is found by an eigenvector calculation on a matrix which encodes the passive dynamics and state costs of the system.

The implications of a linear form of the Bellman equation in control theory are far reaching. It allows for many new and powerful techniques, such as the compositionality of optimal control laws (Todorov, 2009a) and identification of movement primitives through observation (Todorov, 2009b). Applications have just begun to be explored, but already its applicability to the field of robotics is apparent. Currently, however, high precision control in large environments requires a very high resolution representation of the environment. This incurs exorbitant computational and storage requirements on the control system. The work presented here addresses this problem, presenting an efficient, low-cost control method for high precision movements in large environments.

2 Linear Bellman Controllers

LBCs operate by controlling the transition probabilities, $u(\cdot|\cdot)$, and effecting control signals to realize desired probability distributions over the potential next-states of the system. To derive the linear Bellman equation, start with the Bellman equation

$$v(x) = \min_u \{l(x,u) + E_{x' \sim p(\cdot|u,x)}[v(x')]\}, \qquad (1)$$

which specifies the cost to move optimally to the target from a state, $v(x)$, by summing the cost of moving from the current state optimally, $l(x,u)$, and the expected cost-to-go from the next state, $E_{x' \ p(\cdot|u,x)}[v(x')]$. For LBCs, since probability distributions are being compared, the cost function will be based on the Kullback-Leibler (KL) divergence function,

$$\begin{aligned} l(x,u) &= q(x) + KL(u(\cdot|x)||p(\cdot|x)) \\ &= q(x) + E_{x' \sim u(\cdot|x)}\left[log \frac{u(x'|x)}{p(x'|x)}\right]. \end{aligned} \qquad (2)$$

The linear Bellman equation is derived by exponentiating the cost-to-go function

$$z(x) = \exp(-v(x)), \qquad (3)$$

resulting in $z(x)$, which is termed the 'desirability' function due to being large where the cost-to-go v is small. Substituting Eq.(2) into Eq.(1) and rewriting in terms of z gives

$$-\log(z(x)) = q(x) + \min_u \left\{ E_{x' \sim u(\cdot|x)} \left[\log \frac{u(x'|x)}{p(x'|x) z(x')} \right] \right\}. \qquad (4)$$

Importantly, the lower term of the expected cost-to-go function must be normalized before the rules of KL divergence can be applied. To do this, a normalization term

$$\mathcal{G}[z](x) = \sum_{x'} p(x'|x) z(x') = E_{x' \sim p(\cdot|x)}[z(x')] \qquad (5)$$

must be applied.

In a KL divergence, the global minimum is achieved when the two distributions are equal, so the minimum of the expected cost-to-go will be

$$u^*(x'|x) = \frac{px'|xz(x')}{\mathcal{G}[z](x)}, \tag{6}$$

where u^* represents the optimal action. Substituting the optimal action into the Bellman equation, accounting for the normalization term and exponentiating gives

$$z(x) = \exp(-q(x))\mathcal{G}[z](x), \tag{7}$$

where Eq.(7) is linear in terms of z. Eq.(7) can be rewritten in vector notation by enumerating the states from 1 to n, using the n-dimensional column vectors $\mathbf{z}$ and $\mathbf{q}$ to represent $z(x)$ and $q(x)$, and the n-by-n matrix P to represent $p(x'|x)$, with current states, x, listed down the rows and potential next-states, x', across the columns. This can then be rewritten as an eigenvector problem by defining the matrix $M = \text{diag}(\exp(-\mathbf{q}))P$ and substituting into Eq.(7)

$$\mathbf{z} = M\mathbf{z}. \tag{8}$$

See (Todorov, 2009c) for details on the derivation.

Informally, LBCs calculate the optimal cost-to-go from each state by first identifying the set of possible system states, enumerating them, and setting up a passive dynamics matrix $\mathbf{P}$ whose columns and rows are indexed by this list of all system states. The matrix $\mathbf{P}$ encodes the transition probabilities under the passive dynamics of the system, where the row indices represent a current system state, $\mathbf{x}$, and the columns represent a possible next-step system state, $\mathbf{x}'$. Importantly, the Bellman equation is not solved for states that are not represented in the list of enumerated states. For an LBC to obtain millimeter accuracy inside a metre squared operating environment, then, there must be a state representing every point at millimeter resolution.

In addition to the passive dynamics matrix, $\mathbf{P}$, the LBC also specifies a diagonal cost matrix, $\mathbf{Q}$, that assigns a cost value to every state based on the cost function and desired movement. These two matrices are then multiplied together and the primary eigenvector is found.

The problem of setting up, storing, and solving for the primary eigenvector of the $\mathbf{QP}$ matrix quickly becomes intractable with an increasing number of represented states. Here we present the dynamic scaling technique (DST), a novel method for efficient, low-cost control of high-precision movements in large environments under an LBC.

The remainder of this paper is organized as follows: In the next section the DST is introduced, subsequently that results from simulations are presented, followed by a comparison with standard methods and a performance versus cost analysis. The paper concludes with a discussion of the implications of the presented results and potential avenues for future development.

3 Dynamic scaling technique

The basic algorithm of the DST is straightforward. First, create a low resolution representation of the operating space and solve for the optimal movement. Then, as the target is approached, reduce the size of the operating space such the borders are determined by the new location of the end-effector and the target, and solve for the optimal control law again. Continue this process until the system has been moved to the target location with the desired precision.

The DST takes advantage of the observation that relative movement in 3D or 2D space passive dynamics remains unaltered as the size of the environment scales up or down. For example, whenever the system is at position (1,1) and has a velocity of (0,1), the next state will be (1,2), regardless of whether a unit represents a millimetre, centimetre, or metre.

Consequently, we can realize a significant computational savings by allowing the LBC to calculate the passive dynamics matrix $\mathbf{P}$, once, and recalculate only the diagonal cost matrix $\mathbf{Q}$ as movement scale changes. The number of system states that are represented in P will determine how close to optimal the resulting control law will be, as it will not always be possible to move optimally to a target without a large number of available states throughout the operating space. This can be seen by thinking of the environment as a grid, where it is only possible to move from point to point on the grid; the more points on the grid, the more options the system has to realize the optimal trajectory of continuous space. Clearly, then, there is a trade between the approximately optimal movements and the calculations and storage space required. As shown below, the computational saving can be up to 5 orders of magnitude, while realizing nearly optimal movement.

This massive cost reduction realized by the DST makes high precision movements in large environments feasible for systems with restricted resources, such as those of an on-board processor controlling robotic limb placement, or the neural-circuit control systems found in animals with fine-grained motor control.

4 System simulation

To demonstrate the application of the DST here we describe a simple, linear 2D system model, and its control using the DST. The system being modeled is a two-jointed planar arm. All simulations were run in MATLAB. It is important to note that none of the techniques

used are restricted to linear systems. For simplicity a linear model was chosen, but any nonlinear function can be used to describe the passive dynamics of the system.

Let the system be defined by the state space equations:

$$\mathbf{x} = \begin{bmatrix} p_x & p_y & \dot{p}_x & \dot{p}_y \end{bmatrix}^T, \quad \dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where p_x and p_y represent x and y position, and $\dot{p}_x$ and $\dot{p}_y$ represent x and y velocities.

When calculating the passive dynamics matrix, $\mathbf{P}$, the above equation is used to determine the next position of the system in the (x, y) grid, with the control signal set to zero. Importantly, in the LBC controller formulation, the control signal operates in the same space as the noise. This means that the transition probabilities in P are over the controlled parameters, which for this system is velocity (Todorov, 2009c).

The cost matrix, $\mathbf{Q}$, is generated using a simple cost function that assigns cost based on a weighted summation of the Euclidean distance of the (x, y) position and velocities of a state to the target position and velocities. Although this is termed the 'cost' matrix, what is actually encoded is not the state costs, but the negative exponential of the state costs, $\exp(-\mathbf{q})$, more accurately referred to as the state desirabilities.

Before control begins, the system solves for control signals to move optimally to each of the four corners of the 2D environment, which are stored in memory as learned movements. During execution the system will then 'draw' a box in the environment between the end-effect and the target, which is referred to as the operating environment. The system then chooses a control signal from the appropriate learned movement, determined by which corner the target is in, and scales based on the size of the operating environment. A psuedocode version of the DST is presented in Algorithm 1.

The implementation of the DST presented in this paper rescales the operating space after every movement, and recalculates the optimal command inside the new reduced area. This method of rescaling was chosen to provides an upper bound for the number of times the control signal must be calculated for a given grid size. Consequently, it is a worst case scenario for detmining the improvement in efficiency. An implementation that preventing any rescaling once the grid size reaches the desired precision, and makes more than one move before rescaling would cause a further reduction in the number of calculations required.

Algorithm 1 DST psuedocode

$P = passiveDynamics(equationsOfMotion)$
{The cost functions for each of the target corners}
$Q = [Q00, Q01, Q11, Q10]$
{The control signals for each of the target corners}
$U = genControl(P, Q)$
while $(distToTarget \geq THRESHOLD)$ **do**
  $[opEnv, dX, dY] = genOpEnv(target, X)$
  {X is the system state}
  $corner = whichCorner(opEnv, target)$
  {Use corner and system state to get control signal}
  $currentU = U[corner]$
  $u = currentU[getIndex(X)]$
  {Scale the control signal by the size of the operating environment grid units}
  $u = u * [dX, dY]$
  {Send the control signal to the system for execution}
  $X = nextState(X, u)$
end while

Table 1: Simulation results

Grid size	Avg path error	Gen time (s)
3	320.9993	0.1042
5	182.2730	0.1334
7	143.4651	0.4391
10	115.4379	1.3671
12	110.3509	11.6857
15	108.6061	38.1416

5 Simulation results

These simulations were run on a quad-core 3.2GHz machine with 6GB ram running Ubuntu 10.04 (64-bit). The trajectories to 25 evenly spaced targets, (x=[10:20:90], y=[10:20:90]), with millimeter accuracy, were calculated using the DST. To determine the quality of the computed path, we calculated the path error as the difference between the chosen trajectory and a straight line (the optimal trajectory). This was done for grids with 3, 5, 7, 10, 12, and 15 represented points in the operating environment. The results and comparison between the different resolution grids' generation and simulation runtimes are displayed in Table 1 and Figure 1.

The results show that at the grid size 10, the time required to generate the optimal movement at each step starts to significantly increase with grid size and simultaneously the improvement to the chosen trajectory be-

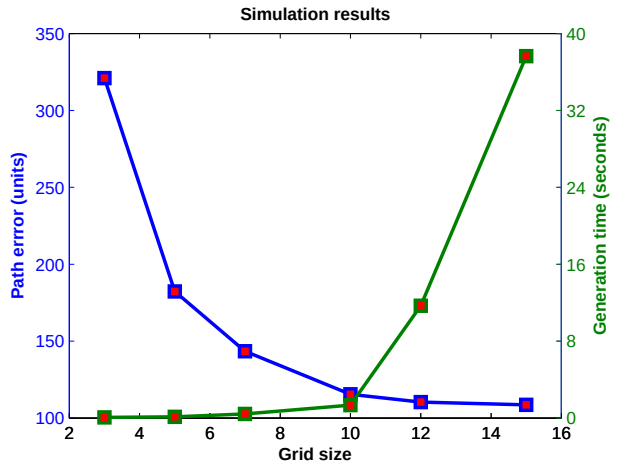

Figure 1: The path error of trajectories calculated using grids with a varying number of represented points in the operating space and the required time to generate the command at each step (in seconds).

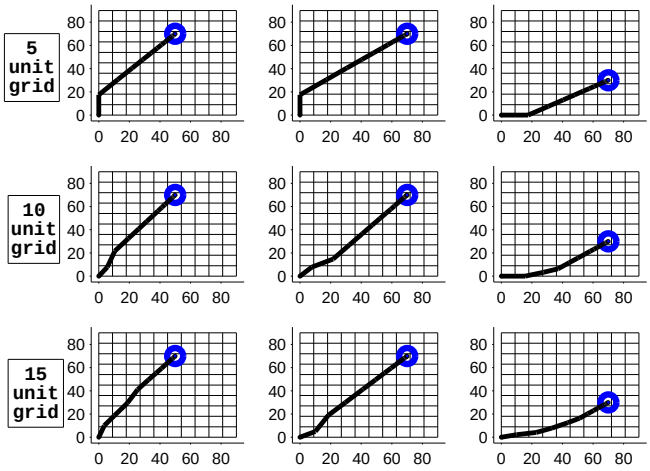

Figure 2: In this figure are the trajectories to 3 different targets ($[50, 70], [70, 70], [20, 70]$) as planned by LBCs with different grid sizes implementing the DST.

comes less and less noteworthy. This reduction in path improvement as the grid size continues to increase can also be seen by graphing the specified trajectories at different grid sizes, as shown in Figure 2.

The second issue that arises from the simulation results is that higher resolution grids need larger maximum step sizes to achieve approximately optimal results. Because the system balances the state costs specified by the cost matrix and the control cost, which is the cost of applying a control signal to change the trajectory from that specified by the passive dynamics matrix[1], when the grid size increases, decreasing the distance between represented points, a simple state cost function weighting the states based on the euclidean distance of their (x, y) location from the target becomes insufficient. Restated, the difference between the cost of the current and neighboring state is not necessarily large enough to justify moving the system.

To address this, either a different function must be used to compute the state costs for each grid size, or the range of velocities must be increased to a point that the difference between the current state cost and reachable states becomes significant. For some cases, the range of velocities must be greatly expanded, or optimal movement can actually be hindered by a higher resolution representation of the operating area, as shown in Figure 3. The above results show trajectories found with maximum velocities of 1 for grids sized 3, 5, 7, maximum velocity of 2 for the grid with 10 points, and 4 for grids size 12 and 15. Here, even though the range of velocities was increased, it was not sufficient to prevent increasing the error of the resulting trajectory.

Because the system must increase the number of represented (x, y) velocities to deal with the increasing resolution, the size of the passive dynamics matrix increases dramatically with the resolution. The number of elements in the passive dynamics matrix is significant to the system calculations because the eigenvector of this matrix scaled by the cost matrix must be calculated to find the optimal path to the target. Even with large grid sizes and velocity ranges, however, the DST achieves a significant reduction in required resources. Figure 4 displays a comparison of the total number of elements in the passive dynamics matrix when moving to a target inside a meter square operating space with millimeter precision with and without the DST on a logarithmic scaling, highlighting the difference in resource requirements.

The displayed number of states required for the controllers implementing the DST is a summation of the number of states at each iteration. As stated before, the method of rescaling implemented in this paper is an upper bound on the number of iterations for a controller to perform.

6 Discussion

By examining the results of these simulations there are several important issues that arise. The first is the importance of the number of points that will be represented by the system during a given iteration of the DST. The velocity range over the grid must be plausible for the system. If the DST is being implemented on a high level controller which does not control the system directly, but

[1]See (Todorov, 2009c) for details

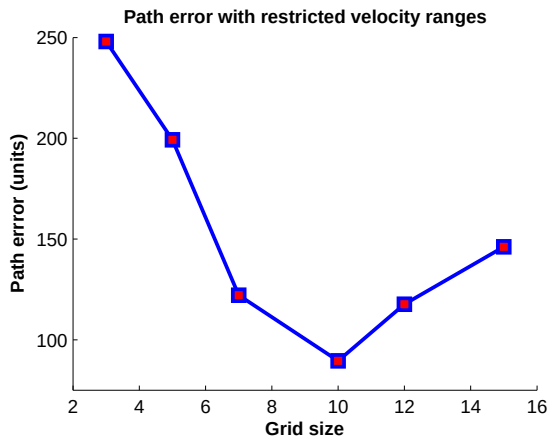

Figure 3: The path errors that occur with different grid sizes when the maximum velocities (or maximum step size across the grid) is held constant as the number of represented states increases.

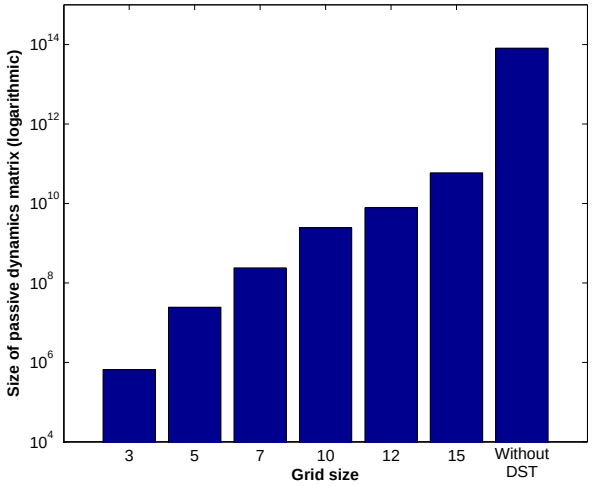

Figure 4: The total number of states that must be represented to reach the target using the different grid sizes presented above and without the DST.

is providing a feedforward control signal to be matched as best as possible by a low level controller, however, then implausible velocity commands will not result in a system failure.

In addition to the above mentioned benefits of the DST, another notable feature is that it provides an approximately optimal trajectory for the system to follow while refinements are calculated, continually increasing the accuracy of the control signal until the desired precision is reached. This is in contrast to system operation without the LBC, where the entire control signal must be calculated offline all at once; the system is at a standstill until the exact optimal trajectory is provided. This becomes especially important when novel environments are encountered and the amount of time taken is important, as is often the case for biological systems controlled by neural circuits.

7 Conclusions

In this paper we have presented the dynamic scaling technique, a method of significantly reducing the resources required to implement high-precision control in large environments through linear Bellman controllers. The DST provides a means of implementing LBCs on restricted systems requiring a high degree of accuracy at a low cost, making LBCs plausible for self-contained robotics and neural-circuit control systems.

The simulation results presented in this paper provide results of a system analogous to a high level controller responsible for providing a feedforward control signal of the abstracted system. This controller issues commands in terms of end effector position, to be converted into a low level control signal directly applicable to the system plant by another controller through quadratic programming or a similar method.

There is still a large amount of room for optimization of the methods presented in this paper,but we believe that the DST is a significant step forward to making LBCs plausible for systems of all kinds, providing a foundation for efficient, low cost control of high precision movements in large environments.

8 Future work

Avenues for future research that we intend to explore include: 1) increasing the number of component movements available, 2) incorporating obstacle avoidance, 3) application to navigation tasks, and 4) exploring potential underlying neural mechanisms that could allow for this type of control to be effected in biological systems.

The first two points are intertwined, the four movement components that were made available in the simple implementation presented in this paper were solved for when the environment was clear of obstacles. The trajectories are straight lines. If, for example, they were solved for with an object in the center of the area, then the path would be curved around the center. An implementation of obstacle avoidance then could be to solve for a variety of different movements with various obstacle locations, then assess the environment and choose the closest, or use movement compositionality techniques (describe in (Todorov, 2009a)).

The application to navigation tasks is also straightforward, the control of an end point in that instance would instead represent the location of the machine on

a map, rather than an end-effector in the immediate environment.

It is thought that the hippocampus and the basal ganglia are strongly involved in environment navigation, and we suspect that such a similar task would involve similar neural components. The investigation into underlying mechanisms will then focus on areas identified to be involved in navigation, as it seems it is an analogous function performed on different information.

References

Todorov, E. (2009a). Compositionality of optimal control laws. *Advances in Neural Information Processing Systems*, *22*, 1856-1864.

Todorov, E. (2009b). Efficient algorithms for inverse optimal control. *Unpublished work, http://www.cs.washington.edu/homes/todorov/papers.htm*.

Todorov, E. (2009c). Efficient computation of optimal actions. In *Pnas* (Vol. 106, p. 11478-11483).