Learning Omnidirectional Path Following Using Dimensionality Reduction

J. Zico Kolter and Andrew Y. Ng
Computer Science Department
Stanford University
Stanford, CA 94305
Email: {kolter, ang}@cs.stanford.edu

***Abstract*— We consider the task of omnidirectional path following for a quadruped robot: moving a four-legged robot along any arbitrary path while turning in any arbitrary manner. Learning a controller capable of such motion requires learning the parameters of a very high-dimensional policy class, which requires a prohibitively large amount of data to be collected on the real robot. Although learning such a policy can be much easier in a model (or "simulator") of the system, it can be extremely difficult to build a sufficiently accurate simulator. In this paper we propose a method that uses a (possibly inaccurate) simulator to identify a low-dimensional subspace of policies that is robust to variations in model dynamics. Because this policy class is low-dimensional, we can learn an instance from this class on the real system using much less data than would be required to learn a policy in the original class. In our approach, we sample several models from a distribution over the kinematic and dynamics parameters of the simulator, then use the Reduced Rank Regression (RRR) algorithm to identify a low-dimensional class of policies that spans the space of controllers across all sampled models. We present a successful application of this technique to the task of omnidirectional path following, and demonstrate improvement over a number of alternative methods, including a hand-tuned controller. We present, to the best of our knowledge, the first controller capable of omnidirectional path following with parameters optimized simultaneously for *all* directions of motion and turning rates.**

I. Introduction

In this paper we consider the task of omnidirectional path following: moving a four-legged robot along any arbitrary path while turning in any arbitrary manner. Examples of such motion include walking in a circle while facing the circle's center, following a straight line while spinning around, or any other such maneuver. In this paper we focus on "trot" gaits: gaits where the robot moves two feet at a time.

A key technical challenge in building such a robust gait is maintaining balance of the robot. Since the trot gait moves two feet at once, the polygon formed by the supporting feet reduces to a line, and it is very difficult to maintain any standard static or dynamic stability criterion. Fortunately, when executing a "constant" maneuver — that is, moving in a constant direction while possibly turning at a constant rate — it is possible to approximately balance the robot by offsetting the center of mass by some fixed distance. However, in omnidirectional path following we are frequently changing the direction of motion and turning rate, so a complete policy must determine the proper center offset for *any* given direction and turning rate. This results in a high-dimensional class of control policies[1], which therefore tends to require a prohibitively large amount of data from the real robot in order to learn.

It can be much easier to learn this high-dimensional policy in a model (or "simulator") of the system than on the real robot; there are many advantages to *model-based reinforcement learning* (RL): the simulator can be made deterministic, we can save and replay states, we can take gradients of the parameters, there is no risk of damage to the real robot, etc. However, it can be extremely difficult to build a sufficiently accurate simulator. Often times a policy learned in the simulator will perform very poorly on the real system.

In this paper, we propose a method that makes use of a (possibly inaccurate) simulator to identify a low-dimensional subspace of policies that is robust to variations in model dynamics. Our method is as follows: First, we sample randomly from a *distribution* over the kinematic and dynamic parameters of the simulator. We learn parameters for the high-dimensional policy in each of these simulation models. To identify a low-dimensional class of policies that can represent the learned policies across all the sampled models we formulate a dimensionality reduction optimization problem that can be solved via the Reduced Rank Regression (RRR) algorithm [26]. Finally, we learn a policy in this low-dimensional class on the real robot; this requires much less data than would be required to learn a policy in the original, high-dimensional class of policies. We present a successful application of this technique to the task of omnidirectional path following, and demonstrate improvement over a number of alternative control methods, including a hand-tuned controller. We also present, to the best of our knowledge, the first controller capable of omnidirectional motion with parameters optimized simultaneously for *all* directions of motion and turning rates — in the next section we clarify exactly how this work differs from previous work in quadruped locomotion.

The rest of this paper is organized as follows. In Section II we present background material and related work. In Section III we give an overview of our controller (the full parametrization of the controller is given in the Appendix) and present an online learning algorithm that is capable of learning balancing

[1] In this paper we focus on linearly-parametrized policies — that is, policies that are specified by a set of linear coefficients. In this case, the dimension of the policy is simply the number of free parameters.

offsets for any *fixed* direction of motion and turning rate. In Section IV we formally present our method for identifying a low-dimensional policy class by sampling from a distribution over simulator models. In Section V we present and analyze our results on the real robotic system. Finally, in Section VI we give concluding remarks.

II. Background and Related Work

A. Quadruped Locomotion

There has been a great deal of work in recent years, both in the robotics and the machine learning communities, on quadruped locomotion. Much of the research has focused on static walking (moving one leg at a time, thereby keeping the robot statically stable) including statically stable gaits capable of omnidirectional path following [19]. In addition, there has been much work on designing static gaits that can navigate irregular terrain [6, 5, 11, 18]. However, in this paper we focus on the dynamic trot gait over flat terrain, which requires very different approaches.

There has also been a great deal of work on developing omnidirectional trot gaits, in particular for the Sony AIBO robot. Much of this work is based on a trot gait, originally developed by Hengst et. al. [13], that is potentially capable of moving in any direction while turning at any rate. While the parameters of the gait presented in [13] are hand-tuned, there has been much subsequent work on gait optimization of this and similar gaits. Several gait optimization methods have been investigated, including evolutionary search methods [14, 8, 27], direction set minimization [16] and policy gradient techniques [17]. However, while the controllers in these papers are *capable* of omnidirectional motion in that they can walk in any direction while turning at any rate, all the papers listed above focus on optimizing parameters only for one *single* direction of motion at a time (usually forward walking). Additionally, as noted in [27], gaits optimized for one type of motion typically perform poorly on other maneuvers — for example, a gait optimized for forward walking typically performs poorly when attempting to walk backwards. In contrast, in this paper we focus on learning a policy that performs well for *all* directions of motion and turning rates.

Another vein of research in dynamic quadruped gaits follows work by Raibert [24, 25]. This and more recent work [22, 21] achieve dynamic gaits — both trot gaits and gallop gaits, which include a flight phase — by "hopping" on compliant legs, which typically employ some form of hydraulics. While this is a powerful technique that can allow for very fast locomotion, we discovered very quickly that our robot was not capable of generating sufficient force to jump off the ground, effectively disallowing such methods.

There is also a great deal of work on dynamic stability criteria for legged robots [12, 23]. One of the most well-known criteria is to ensure that the Zero Moment Point (or ZMP) [29] — which is similar to the center of mass projected on to the ground plane, except that it accounts for inertial forces acting on the robot — never leaves the supporting polygon. However, when the robot has only two rounded feet on the ground the supporting polygon reduces to a line, making it difficult to maintain the ZMP exactly on this line. In addition, we found it difficult to calculate the ZMP precisely on the real robot due to the inaccuracy of the on-board sensors. For these reasons, in this paper we focus on an approximate method for balancing the robot.

B. Learning and Control

The method we propose in this paper is related to the area of robust control theory. For a general overview of robust control, see [31] and [9]. However, our work differs from standard robust control theory in that the typical goal of robust control is to find a *single* policy that performs well in a *wide variety* of possible models. In contrast, we make no assumption that any such policy exists — indeed, in our application the optimal policy depends very much on the particular model dynamics — but rather we want to identify a *subspace* of policies that is robust to variation in the model dynamics. In the final step of our method, we then search this subspace to find a policy that is specific to the dynamics of the real system.

To identify the low-dimensional subspace of policies, we pose an optimization problem that can be solved via the Reduced Rank Regression (RRR) algorithm. The RRR setting was first discussed by Anderson [1]. Izenman [15] developed the solution that we apply in this paper, coined the term "Reduced Rank Regression," and discussed the relationship between this algorithm, Canonical Correlation Analysis (CCA) and Principle Component Analysis (PCA). RRR is discussed in great detail in [26], and there is a great deal of active research on this algorithm, both from theoretical [2] and numerical perspectives [10].

In the machine learning literature, the problem we formulate can be viewed as an instance of multi-task learning [7, 3]. However, our setting does differ slightly from the prototypical multi-task learning paradigm, since we do not ultimately care about performance on most of the tasks, except insofar as it helps us learn the one task we care about — i.e., we don't care how well the policies perform in the simulation models, just how well the final controller performs on the real system.

Finally, we note that there has been recent work on the application of dimensionality reduction techniques to control and reinforcement learning. Mahadevan [20] uses Graph Laplacian methods to learn a low-dimensional representation of value functions on a Markov Decision Process (MDP). Roy et. al. [28] use dimensionality reduction to compactly represent belief states in a Partially Observable MDP. Our work is similar in spirit, except that we apply dimensionality reduction to the space of controllers, to learn a low-dimensional representation of the control policies themselves.

III. A Controller for Omnidirectional Path Following

In this section we present a parametrized gait for the quadruped robot that is capable of omnidirectional path following. The design builds upon recent work on trot gaits for quadruped locomotion [13, 17]. The robot used in this work

Fig. 1. The LittleDog robot, designed and built by Boston Dynamics.

is shown in Figure 1. The robot, known as "LittleDog," was designed and built by Boston Dynamics, Inc and is equipped with an internal IMU and foot force sensors. State estimation is performed via a motion capture system that tracks reflective markers on the robot.

Our controller uses inverse kinematics to specify locations for the four feet in Euclidean coordinates relative to the robot's body. While two feet move along the ground, the other two feet moving through the air in a box pattern; this moves the robot forward [13].[2] We achieve lateral movement by rotating the angle of all the four feet, and turn by skewing the angles of the front and back or left and right feet. We specify paths for the robot as linear splines, with each point specifying a desired position and angle for the robot. The controller is closed-loop: every time step (10 milliseconds) we use the current state estimate to find a direction and turning angle that forces the robot to follow the specified path. A more detailed description of the gait is given in the appendix.

A. *Learning To Balance*

We found that by far the most challenging aspect of designing a robust controller was balancing the robot as it moved. To balance the robot, our controller offsets the center of mass of the robot by some specified amount $(x_{\text{off}}, y_{\text{off}})$.[3] Without a proper center offset, the robot will "limp" visibly while walking. The challenge is to find a function that determines the proper center offset for any given direction of motion and turning rate. That is, given a direction angle ψ and turning rate ω, we want to find functions f_x and f_y such that

$$x_{\text{off}} = f_x(\psi, \omega), \quad y_{\text{off}} = f_y(\psi, \omega).$$

Since the direction angle and turning rates are inherently periodic — i.e., a direction angle of 2π is identical to a directon angle of 0 — the Fourier bases are a natural means of representing these functions. We therefore represent f_x as

$$f_x(\psi, \omega) = \theta_x^T \phi(\psi, \omega)$$

where $\theta_x \in \mathbb{R}^k$ is a vector of coefficients and

$$\begin{aligned}\phi(\psi, \omega) \quad = \quad & [\cos(i\psi)\cos(j\omega), \cos(i\psi)\sin(j\omega), \\ & \sin(i\psi)\cos(j\omega), \sin(i\psi)\sin(j\omega)], \\ & i, j = 1, 2, \ldots\end{aligned}$$

denotes first k principle Fourier basis functions of ψ and ω — here the range of i and j are chosen so that the dimension of ϕ is also k. The function f_y is represented in the same manner, and learning a parametrization of the controller requires learning the coefficients θ_x and θ_y of this approximation.

With this motivation, we first consider the problem of finding the center offset for a *fixed* direction angle ψ and turning rate ω. We designed a online learning algorithm that, for fixed ψ and ω, dynamically adjusts the center offsets during walking so as to balance the robot. The intuition behind this algorithm is that if the robot is balanced, then the two moving feet should hit the ground simultaneously. If the two feet do not hit the ground simultaneously, then the algorithm looks at which of the two feet hit the ground first, and adjusts the center offsets accordingly. If, for example, the back leg hits the ground before the front leg, then the algorithm will shift the center of mass forward, thereby tilting the robot forward, and encouraging the front leg to hit the ground sooner. The precise updates are given by

$$\begin{aligned} x_{\text{off}} &:= x_{\text{off}} + \alpha(g(t_{\text{FL}} - t_{\text{BR}}) + g(t_{\text{FR}} - t_{\text{BL}})) \\ y_{\text{off}} &:= y_{\text{off}} + \alpha g((t_{\text{FL}} - t_{\text{BR}}) - (t_{\text{FR}} - t_{\text{BL}})) \end{aligned} \tag{1}$$

where α is a learning rate, $t_{\text{FL}}, t_{\text{FR}}, t_{\text{BL}}, t_{\text{BR}}$ are the foot contact times for the four feet respectively, and $g(x) = x^2 \cdot \text{sgn}(x)$. So, for example, if the back left leg hits before the front right, $t_{\text{FR}} - t_{\text{BL}} > 0$, so x_{off} is increased, shifting the center of mass forward. The algorithm is similar in spirit to the previously mentioned gait optimization algorithms for the Sony AIBO gaits [14, 16, 8, 17, 27] in that it performs online optimization of the gait parameters for a fixed direction angle and turning rate. We implemented this algorithm both in simulation and on the actual robot; for the actual robot we used foot force sensors to determine when a foot hit the ground.[4] Convergence to a stable center position is generally quite fast, about a minute or two on the real robot (assuming no adverse situations arise, which we will discuss shortly).

Given a collection of direction angles, turning rates, and their corresponding center offsets, we can learn the coefficients θ_x and θ_y by least squares regression. Specifically, if we are given a set of n direction angles, turning rates, and resulting

[2] We also experimented with other locus shapes for moving the feet that are common in the literature, such as a half ellipse or a trapezoid [17], but found little difference on our robot in terms of performance.

[3] In the coordinate system we use, the positive x axis points in the direction that the robot is facing, and the positive y axis points to the robot's left.

[4] Although the foot sensors on the LittleDog robot are not particularly accurate in many situations, the trot gait hits the feet into the ground hard enough that the foot sensors can act as a simple Boolean switch indicating whether or not the feet have hit the ground.

x center offsets, $\{\psi_i, \omega_i, x_{\text{off},i}\}$, $i = 1 \ldots n$, then we can learn the parameters $\theta_x \in \mathbb{R}^k$ by solving the optimization problem

$$\min_{\theta_x} \|y - X\theta_x\|_2^2 \tag{2}$$

where $X \in \mathbb{R}^{n \times k}$ and $y \in \mathbb{R}^n$ are design matrices defined as

$$X = [\phi(\psi_1, \omega_1) \ldots \phi(\psi_n, \omega_n)]^T \quad y = [x_{\text{off},1} \ldots x_{\text{off},n}]^T . \tag{3}$$

The solution to this problem is given by, $\theta_x = (X^T X)^{-1} X^T y$, and by a well known sample complexity result [4], we need $\Omega(k)$ data points to find such a solution.

Unfortunately, computing a sufficient number center offsets on the real robot is a time-consuming task. Although the algorithm described above can converge in about a minute under ideal circumstances, several situations arise that can slow convergence considerably. Communication with the robot is done over a wireless channel, and packet loss can make the robot slip unexpectedly, which causes incorrect adjustments to the robot. Additionally, an improper center offset (as would occur before the algorithm converged) can make the robot move in a way that degrades the joint angle calibration by bashing its feet into the ground. Although it is significantly easier to find proper joint offsets in simulation, it is difficult to create a simulator that accurately reflects the center offset positions in the real robot. Indeed, we invested a great deal of time trying to learn parameters for the simulator that reflected the real system as accurately as possible, but still could not build a simulator that behaved sufficiently similarly to the real robot. The method we present in the next section allows us to deal with this problem, and efficiently learn center offsets for the real robot by combining learning in both simulation and the real robot.

IV. Identifying a Low Dimensional Policy Class Using Dimensionality Reduction

In this section we present a method for identifying a low-dimensional policy class by making use of a potentially innaccurate simulator. Although we focus in this paper on the application of this method to the specific task of learning a policy for omnidirectional path following, the formulations we present in this section are general.

The intuition behind our algorithm is that even if it is very difficult to find the precise dynamic parameters that would make a simulator accurately reflect the real world, it may be much easier to specify a *distribution* over the potential variations in model dynamics.[5] Therefore, even if the simulator does not mimic the real system exactly, by considering a distribution over the model parameters, it can allow us identify a smaller subspace of policies in which to search. The method we propose is as follows:

1) Draw m random samples from a distribution over the dynamic and kinematic parameters of the simulator. Each set of parameters now defines a differently perturbed simulation model.
2) In each simulation model, collect n data points. For example, in our setting each of these data points corresponds to a particular direction angle ψ, turning rate ω, and the resulting center offset (x_{off} or y_{off}) found by the online balancing algorithm described previously.
3) Use the Reduced Rank Regression algorithm to learn a small set of basis vectors that span the major axes of variation in the space of controllers over all the sampled models.
4) For the real robot, learn the parameters of the policy in this low-dimensional class.

More formally, suppose we are given a matrix of feature vectors $X \in \mathbb{R}^{n \times k}$ and we collect a set of of output vectors $\{y^{(i)} \in \mathbb{R}^n\}, i = 1, \ldots, m$, where each of the $y^{(i)}$'s corresponds to the data points collected from the ith simulator. In our setting these matrices are given by (3), i.e., the rows of X are the k-dimensional Fourier bases, and the entries of $y^{(i)}$ are the center offsets found by the online balancing algorithm (notice that we now require the i superscript on the y vectors, since the resulting center offsets will vary between the different simulator models). Rather than learn the coefficients for each simulator individually, as in (2), we consider all the m simulator models jointly, by forming the design matrices $Y \in \mathbb{R}^{n \times m}$ and $\Theta \in \mathbb{R}^{k \times m}$,

$$Y = \left[y^{(1)} \ldots y^{(m)}\right], \quad \Theta = \left[\theta^{(i)} \ldots \theta^{(m)}\right]$$

and considering the problem[6]

$$\min_{\Theta} \|Y - X\Theta\|_F^2. \tag{4}$$

However, solving this problem is identical to solving each of the least squares problems (2) individually for each i; if we want to learn the coefficients for a policy on the real robot, this approach offers us no advantage.

Instead, we consider matrices $A \in \mathbb{R}^{k \times \ell}, B \in \mathbb{R}^{\ell \times m}$, with $\ell \ll k$ and the problem

$$\min_{A,B} \|Y - XAB\|_F^2 . \tag{5}$$

In this setting, the A matrix selects ℓ linear combinations of the columns of X, and the B matrix contains the coefficients for these linear combinations. In other words, this approximates the coefficients as $\theta^{(i)} \approx Ab^{(i)}$, were $b^{(i)} \in \mathbb{R}^\ell$ is the ith column of B. The matrix A forms a basis for representing the coefficients in *all* the m simulation models.

The key advantage of this approach comes when we consider learning the parameters $\theta \in \mathbb{R}^k$ of a policy on the real robot. We approximate θ as a linear combination of the columns of A, i.e., $\theta = Ab$. However, since A, as defined above, has only ℓ columns, we only need to learn the ℓ-dimensional coefficient vector b in order to approximate θ. By a standard sample complexity result [4], this requires only $O(\ell)$ examples, and since $\ell \ll k$, this greatly reduces the amount of data required to learn the policy on the real system.

[5]This claim is discussed further in Subsection IV-B.

[6]Here, $\|\bullet\|_F^2$ denotes the squared Frobenius norm, $\|A\|_F^2 = \sum_{i,j} A_{ij}^2$.

In order to motivate the exact optimization problem presented in (5), we discuss other possible approaches to the problem. First, we could solve (4) to find the least squares solution Θ, then run Principal Component Analysis (PCA) to find a set of basis vectors that could accurately represent all the columns of Θ. However, this approach ultimately minimizes the wrong quantity: we do not truly care about the error in approximating the coefficients $\theta^{(i)}$ themselves, but rather the error in approximating the actual data points $y^{(i)}$. Second, we could run PCA on the Y matrix, which would result in a set of basis vectors that could represent the data points across all the different simulation models. However, we would require some way of extending these bases to new data points not in the training set, and this can be difficult.[7] Instead, the minimization problem (5) truly represents the error quantity that we are interested in — how well our coefficients can approximate the data points Y.

Despite the fact that the optimization problem (5) is non-convex, it can be solved efficiently (and exactly) by the Reduced Rank Regression algorithm. We begin by noting that (5) is identical to the problem

$$\begin{array}{rl} \min_{\Theta} & \|Y - X\Theta\|_F^2 \\ \text{s.t.} & \text{rank}(\Theta) = \ell. \end{array} \tag{6}$$

The solution to this Reduced Rank Regression problem, (6), is given by

$$\Theta = (X^T X)^{-1} X^T Y V V^T \tag{7}$$

where the columns of V are the ℓ principle eigenvectors of

$$Y^T X (X^T X)^{-1} X^T Y. \tag{8}$$

This result is proved in [26, Theorem 2.2]. Optimal values of A and B can be read directly from this solution,

$$A = (X^T X)^{-1} X^T Y V \quad B = V^T.$$

Notice that the Reduced Rank Regression solution can be interpreted as the least squares solution $(X^T X)^{-1} X^T Y$ projected into the subspace spanned by V. When V is full rank — i.e., there is no rank constraint — then $VV^T = I$, and the solution naturally coincides with the least squares solution.

After learning A and B, the final step in our algorithm is to learn a policy on the real robot. To do this we collect, from the real system, a small set of data points $y \in \mathbb{R}^p$ for some (new) set of feature vectors $X \in \mathbb{R}^{p \times k}$, and solve the least squares problem

$$\min_b \|y - XAb\|_2^2$$

For $b \in \mathbb{R}^\ell$. For the reasons mentioned above, b can be estimated using much less data that it would take to learn the full coefficient vector $\theta \in \mathbb{R}^k$. After learning b, we approximate the robot's policy parameters as $\theta = Ab$.

[7] There are certainly methods for doing this. For example, the Nyström approximation [30] could be used compute a non-parametric approximation to the output y' for previously unseen feature vector x'. However, this would require computing the outputs corresponding to x' for all (or at least many) of the simulators, making this technique less applicable for the real-time situations we are concerned with.

A. Non-uniform Features

Note that one restriction of the model as presented above is that the feature vectors X must be the same for all simulations. In our particular application of Reduced Rank Regression, this does not seem to be overly restrictive, since the data points are generated by a simulator. Therefore, we can typically choose the data points to be whatever we desire, and so can simply restrict them to be the same across all the simulation models. Alternatively, the framework can be extended to handle the case where the different simulation models have different feature vectors, though we no longer know of any method which is guarenteed to find the globally optimal solution.

Let $X^{(i)} \in \mathbb{R}^{n^{(i)} \times k}$ denote the feature vector for the ith simulation model. We now want to minimize

$$\min_{A,B} \sum_{i=1}^{m} \|y^{(i)} - X^{(i)} A b^{(i)}\|_2^2 \tag{9}$$

where $b^{(i)}$ is the ith column of B. This is referred to as the Seemingly Unrelated Regressions (SUR) model of Reduced Rank Regression [26, Chapter 7]. We know of no closed form for the solution in this case, but must instead resort to approximate iterative methods. Note that (9), while not convex in A and B jointly, is quadratic in either argument alone. Therefore, we can apply alternating minimization: we first hold A fixed and optimize over B, then hold B fixed and optimize over A; because each step involves unconstrained minimization of a quadratic form, it can be solved very efficiently by standard numerical codes. We repeat this process until convergence to a fixed point. More elaborate algorithms typically impose some form of normalization constraints on the two matrices [26].

We conducted extensive experiments with alternating minimization algorithms, when the feature vectors were chosen to be different across the different simulations, and we found performance to be nearly identical to the standard case for our particular data set. Since the general method we propose does allow for the feature vectors to be the same across all the simulation models, we focus on the previous setting, where the problem can be solved exactly.

B. Further Discussion

Since our approach requires specifying a distribution over kinematic and dynamic parameters of the simulator, the question naturally arises as to how we may come up with such a distribution, and whether specifying this distribution is truly easier than simply finding the "correct" set of parameters for the simulator. However, in reality it is likely that there does not exist *any* set of parameters for the simulator that reflects the real world. As mentioned in the previous section, we expended a great deal of energy trying to match the simulator to the real system as closely as possible, and still did not achieve a faithful representation. This is a common theme in robust control: stochasticity of the simulator is important not only because we believe the real world to be stochastic to a degree, but because the stochasticity acts as a surrogate

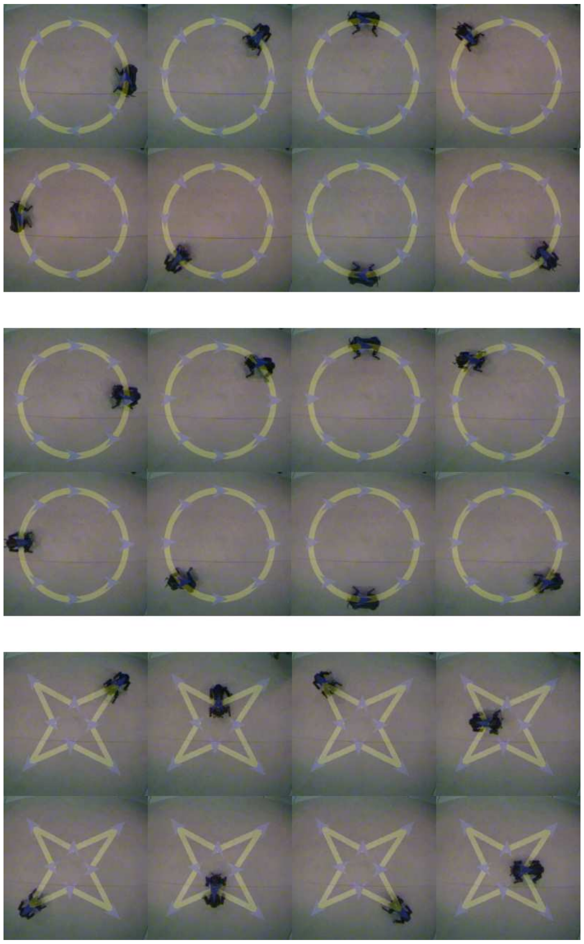
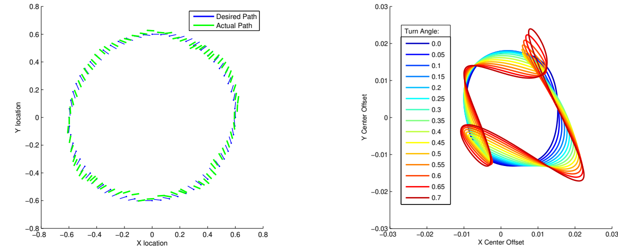


Fig. 2. Pictures of the quadruped robot following several paths.

for the unmodelled effects of the real world. Therefore, we use a straightforward approach to modeling the distribution over simulators: we use independent Gaussian distributions over several of the kinematic and dynamic parameters. This approach worked very well in practice, as we will show in the next section, and it was generally insensitive to the variance of the distributions.

Second, in the method presented here, learning the high-dimensional policy in each perturbed model is framed as a least-squares regression task. This is advantageous because it allows us to optimally solve the joint policy learning problem over all perturbed models with a rank constraint — squared error loss is the only loss we are aware of can be optimally solved with such a rank constraint. Were we to employ a different method for learning the high-dimensional policies in each perturbed model, such as policy gradient, then we would have to resort to heuristics for enforcing the rank constraint on the policy coefficient matrix. This may be necessary in some cases, if learning a policy cannot be framed as an ordinary least squares problem, but here we focus on the case where the problem can be solved optimally, as this was sufficient for our task of omnidirectional path following.



(a) Desired and actual trajectories for the learned controller on path 1.

(b) Learned center offset curves for several different turning angles.

Fig. 3. Trajectory and center offsets for the learned controller.

V. Experimental Results

In this section we present experimental results on applying our method to learn a controller for omnidirectional path following. The simulator we built is based on the physical specifications of the robot and uses the Open Dynamics Engine (ODE) physics environment.[8]

Our experimental design was as follows: We first sampled 100 simulation models from a distribution over the simulator parameters.[9] In each of these simulation models, we used the online balancing algorithm described in Section III to find center offsets for a variety of fixed directions and turning rates.[10] In our experiments, we constrained the centering function (in both the x and y directions), to be a linear combination of the first $k = 49$ Fourier bases of the direction angle and turning rate. We then applied the Reduced Rank Regression algorithm to learn a low-dimensional representation of this function with only $\ell = 2$ parameters, effectively reducing the number of parameters by more than 95%.[11] Finally, to learn a policy on the actual robot, we used the online centering algorithm to compute proper center locations for 12 fixed maneuvers on the robot and used these data points to estimate the parameters of the low-dimensional policy.

To evaluate the performance of the omnidirectional gait and the learned centering function, we used three benchmark path splines: 1) moving in a circle while spinning in a direction opposite to the circle's curvature; 2) moving in a circle, aligned with the circle's tangent curve; and 3) moving in a circle keeping a fixed heading. To quantify performance of the robot

[8]ODE is available at http://www.ode.org.

[9]Experimental details: we varied the simulators primarily by adding a constant bias to each of the joint angles, where these bias terms were sampled from a Gaussian distribution. We also experimented with varying several other parameters, such as the centers of mass, weights, torques, and friction coefficients, but found that none of these had as great an effect on the resulting policies as the joint biases. This is somewhat unsurprising, since the real robot has constant joint biases. However, we reiterate the caveat mentioned in the previous section: it is not simply that we need to learn the correct joint biases in order to achieve a perfect simulator; rather, the results suggest that perturbing the joint biases results in a class of policies that is robust to the typical variations in model dynamics.

[10]For each model, we generated 100 data points, with turning angles spaced evenly between -1.0 and 1.0, and direction angles from 0 to 2π.

[11]Two bases was the smallest number that achieved a good controller with the data we collected: one basis vector was not enough, three basis vectors performed comparably to two, but had no visible advantage, and four basis vectors began to over-fit to the data we collected from the real robot, and started to perform worse.

Metric	Path	Learned Centering	No Centering	Hand-tuned Centering
Loop Time (sec)	1	**31.65 ± 2.43**	46.70 ± 5.94	34.33 ± 1.19
	2	**20.50 ± 0.18**	32.10 ± 1.79	31.69 ± 0.45
	3	**25.58 ± 1.46**	40.07 ± 0.62	28.57 ± 2.21
Foot Hit RMSE (sec)	1	**0.092 ± 0.009**	0.120 ± 0.013	0.098 ± 0.009
	2	**0.063 ± 0.007**	0.151 ± 0.016	0.106 ± 0.010
	3	**0.084 ± 0.006**	0.129 ± 0.007	0.097 ± 0.006
Distance RMSE (cm)	1	**1.79 ± 0.09**	2.42 ± 0.10	1.84 ± 0.07
	2	**1.03 ± 0.36**	2.80 ± 0.41	1.98 ± 0.21
	3	**1.58 ± 0.11**	2.03 ± 0.07	1.85 ± 0.16
Angle RMSE (rad)	1	0.079 ± 0.006	0.075 ± 0.009	**0.067 ± 0.013**
	2	**0.070 ± 0.011**	**0.070 ± 0.002**	0.077 ± 0.006
	3	**0.046 ± 0.007**	0.058 ± 0.012	0.071 ± 0.009

TABLE I
PERFORMANCE OF THE DIFFERENT CENTERING METHODS ON EACH OF THE THREE BENCHMARK PATHS, AVERAGED OVER 5 RUNS, WITH 95% CONFIDENCE INTERVALS.

on these different tasks, we used four metrics: 1) the amount of time it took for the robot to complete an entire loop around the circle; 2) the root mean squared difference of of the foot hits (i.e., the time difference between when the two moving feet hit the ground); 3) the root mean squared error of the robot's Euclidean distance from the desired path; and 4) the root mean squared difference between the robot's desired angle and its actual angle.

Note that these metrics obviously depend on more than just the balancing controller — speed, for example, will of course depend on the actual speed parameters of the trot gait. However, we found that good parameters for everything but the balancing controller were fairly easy to choose, and the same values were optimal, regardless of the balancing policy used. Therefore, the differences in speed/accuracy between the different controllers we present is entirely a function of how well the controller is capable of balancing — for example, if the robot is unable to balance it will slip frequently and its speed will be much slower than if it can balance well.

We also note that prior to beginning our work on learning basis functions, we spent a significant amount of time attempting to hand-code a centering controller for the robot. We present results for this hand-tuned controller, since we feel it represents an accurate estimate of the performance attainable by hand tuning parameters. We also evaluated the performance of the omnidirectional gait with no centering.

Figure 2 shows pictures of the robots following some of the benchmark paths, as well as an additional star-shaped path. Videos of these experiments are available at:

`http://www.stanford.edu/~kolter/omnivideos`

Table I shows the performance of each centering method, for each of the four metrics, on all three benchmark paths.[12] As can be seen, the learned controller outperforms the other methods in nearly all cases. As the distance and angle errors indicate, the learned controller was able to track the desired trajectory fairly accurately. Figure 3(a) shows the actual and desired position and orientation for the learned centering controller on the first path. Figure 3(b) shows the learned center offset predictor trained on data from the real robot. This figure partially explains why hand-tuning controller can be so difficult: at higher turning angles the proper centers form unintuitive looping patterns.

[12]The foot hit errors should not be interpreted too literally. Although they give a sense of the difference between the three controllers, the foot sensors are rather imprecise, and a few bad falls greatly affect the average. They therefore should not be viewed as an accurate reflection of the foot timings in a typical motion.

VI. CONCLUSIONS

In this paper, we presented a method for using a (possibly inaccurate) simulator to identify a low-dimensional subspace of policies that is robust to variations in model dynamics. We formulate this task as a optimization problem that can be solved efficiently via the Reduced Rank Regression algorithm. We demonstrate a successful application of this technique to the problem of omnidirectional path following for a quadruped robot, and show improvement over both a method with no balancing control, and a hand-tuned controller. This technique enables us to achieve, to the best of our knowledge, the first omnidirectional controller with parameters optimized simultaneously for all directions of motion and turning rates.

VII. ACKNOWLEDGMENTS

Thanks to Morgan Quigley for help filming the videos and to the anonymous reviewers for helpful suggestions. Z. Kolter was partially supported by an NSF Graduate Research Fellowship. This work was also supported by the DARPA Learning Locomotion program under contract number FA8650-05-C-7261.

REFERENCES

[1] T. W. Anderson. Estimating linear restrictions on regression coefficients for multivariate normal distributions. *Annals of Mathematical Statistics*, 22(3):327–351, 1951.

[2] T. W. Anderson. Asymptotic distribution of the reduced rank regression estimator under general conditions. *Annals of Statistics*, 27:1141–1154, 1999.

[3] Rie Kubota Ando and Tong Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6:1817–1853, 2005.

[4] Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, 1999.

[5] Shaoping Bai and K.H. Low. Terrain evaluation and its appliaction to path planning for walking machines. *Advanced Robotics*, 15(7):729–748, 2001.

[6] Shaoping Bai, K.H. Low, Gerald Seet, and Teresa Zielinska. A new free gaint generation for quadrupeds based on primary/secondary gait. In *Proceedings of the 1999 IEEE ICRA*, 1999.

[7] Rich Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997.

[8] Sonia Chernova and Maneula Veloso. An evolutionary approach to gait learning for four-legged robots. In *Proceedings of the 2004 IEEE IROS*, 2004.

[9] Geir E. Dullerud and Fernando Paganini. *A Course in Robust Control Theory: A Convex Approach* vol 36 in *Texts in Applied Mathematics*. Springer, 2000.

[10] Lars Eldén and Berkant Savas. The maximum likelihood estimate in reduced-rank regression. *Numerical Linear Algebra with Applications*, 12:731–741, 2005.

[11] Joaquin Estremera and Pablo Gonzalex de Santos. Free gaits for quadruped robots over irregular terrain. *The International Journal of Robotics Research*, 21(2):115–130, 2005.

[12] Elena Garcia, Joaquin Estremera, and Pablo Gonzalez de Santos. A comparative study of stability margins for walking machines. *Robotica*, 20:595–606, 2002.

[13] Bernhard Hengst, Darren Ibbotson, Son Bao Pham, and Claude Sammut. Omnidirectional locomotion for quadruped robots. In *RoboCup 2001: Robot Soccer World Cup V*, pages 368–373, 2002.

[14] Gregory S. Hornby, S. Takamura, J. Yokono, T. Yamamoto O. Hanagata, and M. Fujita. Evolving robust gaits with aibo. In *Proceedings of the 2000 IEEE ICRA*, pages 3040–3045, 2000.

[15] Alan J. Izenman. Reduced-rank regression for the multivariate linear model. *Journal of Multivariate Analysis*, 5:248–264, 1975.

[16] Min Sub Kim and William Uther. Automatic gait optimization for quadruped robots. In *Proceedings of the 2003 Australian Conference on Robotics and Automation*, 2003.

[17] Nate Kohl and Peter Stone. Machine learning for fast quadrupedal locomotion. In *The Nineteenth AAAI*, pages 611–616, July 2004.

[18] Honglak Lee, Yirong Shen, Chih-Han Yu, Gurjeet Singh, and Andrew Y. Ng. Quadruped robot obstacle negociation via reinforcement learning. In *The 2006 IEEE ICRA*, 2006.

[19] Shugen Ma, Tomiyama Takashi, and Hideyuki Waka. Omnidirectional staic walking of a quadruped robot. *IEEE Transactions of Robotics*, 21(2):152–161, 2005.

[20] Sridhar Mahadevan. Proto-value functions: Developmental reinforcement learning. In *Proceedings of the 22nd ICML*, 2005.

[21] J. Gordon Nichol, Surya P.N. Singh, Kennetch J. Waldron, Luther R. Palmer III, and David E. Orin. System design of a quadrupedal galloping machine. *International Journal of Robotics Research*, 23(10–11):1013–1027, 2004.

[22] Ioannis Poulakakis, James A. Smith, and Martin Buehler. Experimentally validated boudning bodels for the scout II quadrupedal robot. In *Proceedings of the 2004 IEEE ICRA*, 2004.

[23] Jerry E. Pratt and Russ Tedrake. Velocity-based stability margins for fast bipedal walking. In *Proceedings of the First Ruperto Carola Symposium on Fast Motions in Biomechanics and Robotics: Optimization and Feedback Control*, 2005.

[24] Marc H. Raibert. *Legged Robots that Balance*. MIT Press, 1986.

[25] Marc H. Raibert. Trotting, pacing, and bounding by a quadruped robot. *Journal of Biomechanics*, 23:79–98, 1990.

[26] Gregory C. Reinsel and Raja P. Velu. *Multivariate Reduced-Rank Regression: Theory And Appplications*. Springer-Verlag, 1998.

[27] Thomas Rofer. Evolutionary gait-optimization using a fitness function based on proprioception. In *RobotCup 2004: Robot World Cup VIII*, pages 310–322, 2005.

[28] Nicholas Roy, Geoffrey Gordon, and Sebastian Thrun. Finding approximate pomdp solutions through belief compression. *Journal of Artificial Intelligence Research*, 23:1–40, 2005.

[29] Miomir Vukobratovic and Branislav Vorovac. Zero-moment point – thirty five years of its life. *International Journal of Humanoid Robotics*, 1(1):157–173, 2004.

[30] Christopher K. I. Williams and Matthias Seeger. Using the nystrom method to speed up kernel machines. In *NIPS 13*, pages 682–688, 2001.

[31] Kemin Zhou, John Doyle, and Keither Glover. *Robust and Optimal Controler*. Prentice Hall, 1996.

Appendix

The omnidirectional controller is parameterized by 15 parameters:

- (f_x, f_y, f_z): the x, y, z coordinates of the front left foot in its center position (other feet are properly reflected).
- (b_x, b_y, b_z): maximum size of the foot locus boxes.
- $(x_{\text{off}}, y_{\text{off}})$: x and y offfsets for the center of mass of the robot. As discussed in the paper, these are not specified to be constant values, but depend on the direction angle and turning rate.
- t: time it takes for one cycle through the gait (moving all four feet).
- $d_{\text{ang}}, d_{\text{dist}}$: the d parameters specify how closely the dog should follow its path spline (see below).
- $\omega_{\max}, c_\omega$: turning parameters. We restrict the turning angle of the robot ω to always be less than $\omega_{\max}$. c_ω is a factor that multiplies the turning angle (see below).
- ψ, ω: direction angle and turning angle for the robot respectively. Determined by the position of the robot relative to the path spline.

During the first $t/2$ seconds of the gait period, the robot moves the front left and back right legs in the in the box pattern, while moving the front right and back left legs on the ground (and vice versa during the next $t/2$ seconds) [13]. The legs are moved at a constant velocity along each side of the box, at whatever speed is necessary to move them through the box pattern in the allotted time.

The x and y positions of the ends of the box relative to the foot are determined by ψ and ω. Let α_{FL} be the desired tilt angle for the front left foot. The two endpoints of the foot's motion are given by

$$\begin{aligned}(x_0, y_0) &= (-(b_x/2)\cos(\alpha_{\text{FL}}), -(b_y/2)\sin(\alpha_{\text{FL}}))\\ (x_{t/2}, y_{t/2}) &= ((b_x/2)\cos(\alpha_{\text{FL}}), (b_y/2)\sin(\alpha_{\text{FL}})).\end{aligned}$$

More intuitively, we can think of rotating the box for the front foot by α_{FL} and setting its length so that it lies in the ellipse formed by the tangent points b_x and b_y. The desired tilt is given by the following formula (with the signs reversed appropriatedly for the other feet):

$$\alpha_{\text{FL}} = \psi + \cos(\psi)\omega + \sin(\psi)\omega.$$

Trajectories for the robot are described as linear splines, with each point representing a location in two dimensional space (paths do not have a z component), and an angle. The direction angle of the robot is set to be the direction to the point on the spline that is d_{dist} ahead of the robot. The turning rate is specified in much the same way: we look at the desired angle for the point on the spline d_{ang} ahead of the robot, and let ω be c_ω times the difference between this desired angle and the current angle of the robot.