A Machine Learning Approach to Developing Rigid-body Dynamics Simulators for Quadruped Trot Gaits

Jin-Wook Lee
leepc12@stanford.edu

***Abstract*— I present a machine learning based rigid-body dynamics simulator for trot gaits of the quadruped LittleDog[1]. My contribution can be divided into two parts: the first part for the reduction of one-time-step prediction error, and the second part for a more accurate prediction over longer time scales. First, in order to reduce the one-step prediction error, I compared three regression methods: 1st order linear regression (LR), linear weight projection regression (LWPR) [1] and Gaussian process regression (GPR) [2]. Although GPR shows the highest accuracy, its cost for computation and storage – $O(n^3)$ and $O(n^2)$, respectively – is too high to handle a large amount of training data which are required in my problem. Therefore, I developed a sparse GPR, called "local" GPR (LGPR). In LGPR, training data are divided into k clusters by k-means, and a locally full GPR model is constructed for each cluster. The prediction is done with the local model closest to a test data point, in terms of the normalized Euclidean distance. The cost for computation is tractable, approximately $O(m^3)$ where m=n/k, making it possible to use a large amount of training data. I showed that LGPR significantly reduces the one-step prediction error. Second, to reduce the accumulation of prediction error, I proposed a projection method, called α-$PROJ$. I found that using LGPR as it is accumulates the prediction error too much over longer time scales. One of the main sources of such error turned out to be in some predicted states that are not within the dynamically feasible region. α-$PROJ$ projects a predicted state to such region. I defined the region without using any specific information of LittleDog (e.g. max/min joint angles). It is only defined by the training data. Thus, this projection method can be applied to other robot systems without modification. α-$PROJ$ highly improved the prediction over longer time scales. The empirical results show that these key ideas led to the significant reduction of prediction error and better performance than the ODE[2] simulator.**

I. INTRODUCTION

A. SIMULATOR FOR QUADRUPED TROT GAITS

In recent years, Machine learning algorithms have been extensively applied to the development of a robotic controller to learn the robot's dynamics. When developing such controllers, it is very useful to have an accurate simulator in order to test the controller's performance before applying it to a real robot. Especially for the ground robots, there have been several rigid-body dynamics engines, the most widely used one being Open Dynamics Engine (ODE). Although ODE is used in wide-ranging applications not only for robotics but also for games and animations, it is not accurate enough for engineering purposes. Furthermore, in general, it is very hard to get an accurate dynamics model for complex robot systems [3]. Various characteristics of a robot such as a complex body shape, joint motors and other specifications, which can significantly affect its dynamics, are far too complicated to be input to an ODE type simulator.

Fig. 1. The LittleDog robot

In order to resolve these challenges, I propose a learning based approach to developing an accurate simulator (hereafter, an ML simulator) that does not require any specific information, but learns from data for a robot of interest. I focus on learning dynamics of trot gaits of the quadruped LittleDog on a flat terrain.

B. APPROACH IN THIS RESEARCH

Denote by $\mathbf{s_t}$ a state of the robot at a certain discrete time-step t, then our goal is to predict $\mathbf{s_{t+1}}$ for $t = 0, 1, \ldots, T$ like the following:

$$\mathbf{s_{t+1}} = predict(\mathbf{s_t}, \mathbf{u_t}).$$

A state $\mathbf{s}$ is $\in \mathbb{R}^N$ and an action $\mathbf{u}$ is $\in \mathbb{R}^M$, where N = 33 and M = 12 for the problem I deal with. The initial state $\mathbf{s_0}$ and actions $\mathbf{u_t}$ for $0 \leq t \leq T$ are given for each time step. $\mathbf{s}$ consists of the body's position, linear velocity, orientation, angle of joints and time-derivatives for them, such that it can solely represent the state of the robot at time t. Unlike ordinary regression problems, developing this kind of ML simulator is challenging for two reasons. First, the ML simulator should predict multi-dimensional outputs for every element of $\mathbf{s_{t+1}}$ accurately. Second, a predicted state $\mathbf{s_{t+1}}$ must be used as a feature set to predict its successive state $\mathbf{s_{t+2}}$. This means that once $\mathbf{s_t}$ deviates from the dynamically feasible region, the prediction error will quickly accumulate or diverge. In order to resolve these difficulties, I constructed largely two goals. First, I focused on reducing the one-step prediction error. I compared various regression methods: 1st order linear regression (LR), linear weighted projection regression (LWPR) and Gaussian process regression (GPR). GPR was the best among them.

However, GPR suffers from extremely high cost for computation $O(n^3)$ and storage $O(n^2)$ such that only thousands of training data points are actually used for a prediction. Thus, I adopted a sparse GPR, called subset of regressors (SoR) [4], [5], [6]. For this kind of sparse GPRs, it is very important to choose a good set of m support data points which can represent the entire n ($\gg m$) training data points. I proposed an efficient method that finds such support data points by a k-fold-CV test in the training data pool.

This selection method outperformed over the ordinary random sampling. However, the accuracy was not good enough to be used as a simulator again. Thus, I developed a "local" GPR (LGPR) that divides training data points into k clusters by k-means, and predict with only one local model closest to

[1] Designed and built by Boston Dynamics
[2] Open Dynamics Engine, www.ode.org

a test data point. This further reduced the one-step prediction error, and the computational cost also reduced to a tractable level, $O(m^3)$ where m=n/k, while exploiting all data points in the training data pool.

The second part of my contribution is on the reduction of the prediction error over longer time scales (hereafter, the sequential prediction error). The sequential prediction error accumulates as time step increases because each one-step prediction error is not zero. This accumulation can cause a predicted state $\mathbf{s_t}|_{t\gg 0}$ to deviate from the feasible data region (FDR), in which robot's dynamic constraints are satisfied. I proposed a projection method called α-$PROJ$ that prevents this. I show that the sequential prediction error significantly decreased.

Section II describes data collection and feature selection. In Section III, various regression methods are compared. Section IV proposes the method of choosing support data points. Section V explains the proposed novel LGPR. Section VI presents the projection method α-$PROJ$ that significantly reduces the sequential prediction error. Section VII presents empirical results, and Section VIII contains concluding remarks.

II. DATA AND FEATURE SELECTION

I collected data from LittleDog's trot gaits. Each data point is encoded as $\{(\mathbf{s_t}, \mathbf{u_t}), \mathbf{s_{t+1}}\}$, the feature part and the response part. Each element of the feature part $(\mathbf{s_t}, \mathbf{u_t})$ has its own scale, so each element is normalized to have a unit-variance. I split the collected data into two categories, $TrainPool$ and $TestPool$, such that all the training data come from $TrainPool$ and all the test data from $TestPool$.

The time step was 0.01 second and one trot move lasted for 4 seconds so that each move has about 400 data points. There were 11 types of moves: F/FL/FR(turning or moving straight forward), B/BL/BR (turning or moving straight backward), L/R (moving sidewise), DL/DR (moving diagonally forward) and S (standing). For each type of trot gaits, there were 20 moves, 14 for $TrainPool$ and 6 for $TestPool$.

Now, prior to the discussion of the selection of a proper feature set, let me define a robot's state $\mathbf{s}$ here in more detail. There are 10 rigid-bodies including the robot body, eight leg parts and the ground. The joint constraints among these rigid-bodies restrict the degrees of freedom (DOF) of the system to 18; so, I have 18 dimensional configuration space: $q = (x, y, z, \phi, \theta, \psi, \varphi_1^1, \ldots, \varphi_3^4)$. I denote by (x, y, z) the position of the center of gravity (COG) for the robot body. (ϕ, θ, ψ) means the roll-pitch-yaw orientation of the body. φ_j^i represents the j-th joint angle for the i-th leg, where each leg has three joint angles. Additionally, I consider the first order time-derivatives, i.e. the linear velocity and the angular rate of the orientation/joint angles. One might think that a state should be written as the following:

$$\mathbf{s} = (x_w, y_w, z_w, \dot{x}_w, \dot{y}_w, \dot{z}_w, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}, \varphi_1^1, \ldots, \varphi_3^4, \rho_1^1, \ldots, \rho_3^4) \in \mathbb{R}^{36} \quad (1)$$

where the superscript w stands for the world frame, b means the body frame, and ρ_j^i denotes the angular rate of each joint.

However, taking into account the fact that features should be relevant to predicting a response, three features (x_w, y_w and ψ) should be removed from the feature set. A Robot's x-y position (x_w,y_w) does not actually affect robot's behavior, and so doesn't the yaw angle ψ. Excluding these features is very important to define the feasible data region (FDR) because FDR must be bounded. I will discuss FDR later in Section VI. For the similar reason, $\dot{x}_w$,$\dot{y}_w$ and $\dot{z}_w$ should be rewritten to be originated from the body frame. Because the time step is fixed to be 0.01s, I used the difference terms Δx_b,Δy_b,Δz_b,$\Delta\phi$, $\Delta\theta$ and $\Delta\psi$ rather than the velocity terms $\dot{x}_b$, $\dot{y}_b$, $\dot{z}_b$, $\dot{\phi}$, $\dot{\theta}$ and $\dot{\psi}$. Finally, a state $\mathbf{s}$ can be rewritten as:

$$\mathbf{s} = (z_w, \Delta x_b, \Delta y_b, \Delta z_b, \phi, \theta, \Delta\phi, \Delta\theta, \Delta\psi, \varphi_1^1, \ldots, \varphi_3^4, \rho_1^1, \ldots, \rho_3^4) \in \mathbb{R}^{33} \quad (2)$$

An action $\mathbf{u}$ is simply $\mathbf{u} = (\tilde{\varphi}_1^1, \ldots, \tilde{\varphi}_3^4) \in \mathbb{R}^{12}$ where $\tilde{\varphi}_j^i$ denotes the desired joint angle for the next time step. Therefore, there are 45(=33+12) features and 12 responses.

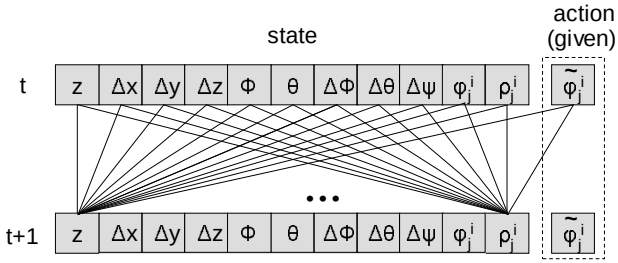

Fig. 2. One-step prediction for a multi-dimensional response

III. COMPARISON OF REGRESSION METHODS

For the one-step prediction, the problem I consider has a multi-dimensional response $\mathbf{s_{t+1}}$. As illustrated in Fig 2, I used a state $\mathbf{s_t}$ and an action $\mathbf{u_t}$ to predict each element of the response $\mathbf{s_{t+1}}$. I applied the same regression method to predict each $\mathbf{s_{t+1}}^i$ ($\mathbf{s}^i$ denotes the i-th element of a state $\mathbf{s}$).

To reduce the one-step prediction error first, I compared three regression methods: 1st order linear regression (LR) as a baseline approach, linear weighted projection regression (LWPR) [1] and Gaussian process regression (GPR) [2].

LR predicts a multi-dimensional response $\bar{Y} = X_*(X^\top X)^{-1}X^\top Y$, where X and Y are a design matrix and a response matrix, respectively. X_* is a test feature set.

LWPR manages K locally linear models, called a receptive field, with a center c_k. The prediction can be done by computing the weighted mean of each model: $\hat{y} = \sum_{k=1}^K w_k\hat{y} / \sum_{k=1}^K w_k$. Each model's weight is $w_k = \exp(-(x-c_k)^\top D_k(x-c_k)/2)$ where D_k is a distance metric in a Gaussian kernel. LWPR learns training data one by one to update its center c_k and parameters, and increase or decrease the number of receptive fields if necessary. LWPR is parametric; so it can predict quickly and handle a large number of training data. However, the major drawback of LWPR is the required manual tuning of many highly data-dependent parameters [3].

GPR assumes $y = f(x) + \varepsilon$ with additive Gaussian noise ε with variance σ_n^2. It predicts a response $\bar{f}(x_*) = k_*^\top(K + \sigma_n^2 I)^{-1}y$ with variance $V[f_*] = k(x_*, x_*) - k_*^\top(K + \sigma_n^2 I)^{-1}k_*$, where $k_* = \phi(x_*)^\top\Sigma_p\Phi$, $k(x_p, x_q) = \phi(x_p)^\top\Sigma_p\phi(x_q)$ and $K = \Phi\Sigma_p\Phi$. GPR requires less data-specific tuning of hyperparameters of its covariance function [3], and can be easily optimized by common gradient ascent algorithms. However, it suffers from extremely high computational cost $O(n^3)$ and memory usage $O(n^2)$ for inverting $(K + \sigma_n^2 I)$. Various sparse approximations were applied to full GPR for resolving this drawback [7]. I will

discuss this in the following section. For GPR, I chose the automatic relevance determination (ARD: a squared exponential kernel with individual distance metric for each feature). Then I optimized its hyperparameters by using a conjugate gradient descent that minimizes the negative marginal log-likelihood. These optimized hyperparameters were also used to tune the distance metric of each feature for LWPR.

I evaluated these three regressers by a hold-out cross-validation test with the identical test data set. I used normalized mean squared error (nMSE) as an error measurement metric such that I can easily compare the error of each element of a state. In terms of test error, GPR outperformed over other two regressors as shown in Table I. Therefore, I decided to use GPR for implementing the ML simulator.

IV. SAMPLE SELECTION FOR SPARSE GPR

Due to GPR's high cost of computation and memory, I considered sparse GPR methods that are designed for handling a larger set of training data. I tried one of them, called subset of regressors (SoR) [4], [5], [6]. The key idea for SoR's approximation is that it chooses m 'support data points' that can represent the entire training data pool $TrainPool$. In SoR, the rest $(n\text{-}m)$ training data points are assumed to be independent with one another and they only communicate through m support data points. SoR's cost for computation and memory is $O(m^2n)$ and $O(mn)$, respectively. Needless to say, it is very important for SoR to choose m good support data points out of the entire n training data points. These data points should be well-distributed so that it can cover up most of $TrainPool$. There are several approaches that jointly find optimal hyperparameters and support data points where m is given [8]. However, especially for the high dimensional and data-rich problem I consider, I couldn't obtain the solution in a reasonable time.

Instead, I invented a simple-but-efficient method that finds support data points by doing k-fold-CV test in the training data pool $TrainPool$. Because I used a squared exponential kernel, covariances among data points are closely related to their normalized Euclidean distances. Therefore, if a data point is not predicted well in a leave-one-out-cross-validation (LOOCV) test, the point is likely to be separated from other data points. To cover up a wider range of $TrainPool$, these separated data points should be included in the support data points set as many as possible. The key idea of my sampling method is measuring 'separateness' of each training data point, and giving them 'separateness distribution'. Sampling support data points from this distribution prevents choosing too many from clustered data points, such that selected ones can be well-distributed in $TrainPool$.

LOOCV test is the ideal one for my sampling method. However, the problem is that full GPR can hold only thousands of training data points for each time of CV test. Thus, I applied k-fold-CV test such that a small number of training data points are involved in each CV test. First, I randomly divided the entire n data points in $TrainPool$ into k groups, and did k times of CV test by full GPR. n/k training data points and $n(k\text{-}1)/k$ test data points are involved in each time of the CV test. Each data point in $TrainPool$ was tested k-1 times, and hence had k-1 number of prediction errors. Summing up these error for each data point and normalizing it over n training data points gave the 'separateness distribution' over the entire training data pool.

Because I applied individual regression model for each element of a state $\mathbf{s}$, 'separateness distribution' should be computed for each element separately.

V. LOCAL GPR

SoR gave less prediction error. However, the one-step prediction should be as accurate as possible because the sequential prediction error can be greatly accumulated through a number of one-step predictions. Thus, I invented a novel sparse GPR called local GPR (LGPR). LGPR can take advantage of using the entire training data points. It divides $TrainPool$ into k clusters by k-means, and construct k locally full GPR models. A prediction for a test data point is done with the closest local model only. This is similar to the block-diagonal approximation of a covariance matrix. However, it is different in that only one local model is activated for a prediction, so computationally less expensive. Of course, the hyperparameters are re-optimized for each local model.

Each local model has approximately $m = n/k$ training data points, so the computational cost of LGPR will be $O(m^3)$; the overhead to find the closest local model is ignored. LGPR can be a better alternative for high dimensional problems which require a very large set of training data for an accurate prediction.

VI. PROJECTION TO TRAINING DATA POOL

In the previous sections I focused on the reduction of the one-step prediction error. One might think that the sequential prediction will be done well through a number of one-step predictions in a row. However, a predicted state $\mathbf{s_t}|_{t\gg 0}$ will greatly deviate from the dynamically feasible region by the accumulation of one-step prediction errors. Therefore, it is required to deal with such deviation.

In the real world or the ODE system, a state $\mathbf{s_{t+1}}$ always satisfies the robot's dynamic constraints. For example, a z-position of each foot should be over zero and each joint's angle should be bounded. I will denote such constrained region by the feasible data region (FDR). FDR is truly bounded since I excluded some features (x_w and y_w) that are not bounded in Section II.

For most one-step predictions by an ML simulator, a predicted state $\mathbf{s_{t+1}}$ usually deviates from FDR a little at least. Because the prediction for $\mathbf{s_{t+2}}$ uses such $\mathbf{s_{t+1}}$ as features, the prediction error will be further amplified.

Thus, we need to adjust the deviated $\mathbf{s_{t+1}}$ to be in FDR. However, I cannot explicitly define FDR, e.g. letting each foot's z-position over zero, because the goal of this research is on the development of an ML simulator that does not consider any specific information about the robot.

Because the problem I deal with is data-rich, I can collect a very large $TrainPool$ covering most of FDR. If I collect infinite number of training data, $TrainPool$ will be equal to FDR. The simplest way of utilizing $TrainPool$ to adjust a deviated state $\mathbf{s_{t+1}}$ is giving the upperbound U^i and the lowerbound L^i to each element of the state $\mathbf{s_{t+1}}^i$ like the following:

$$\mathbf{s_{t+1}}^i := \min(\{U^i, \max(\{\mathbf{s_{t+1}}^i, L^i\})\}).$$

TABLE I

COMPARISON OF NMSE FOR VARIOUS REGRESSION METHODS LR, LWPR, GPR, SOR AND LGPR

Method	z_w	Δx_b	Δy_b	Δz_b	ϕ	θ	$\Delta\phi$	$\Delta\theta$	$\Delta\psi$	ρ_1^1	ρ_1^2	ρ_1^3	ρ_1^4
LR	0.0355	0.3247	0.2701	0.4675	0.0216	0.0121	0.5326	0.3062	0.3612	0.1003	0.0865	0.0875	0.0720
LWPR	0.0293	0.2594	0.1902	0.3008	0.0149	0.0112	0.3246	0.2292	0.2091	0.0620	0.0302	0.0511	0.0623
GPR	0.0231	0.1352	0.1497	0.2442	0.0104	0.0073	0.2406	0.1782	0.1678	0.0511	0.0243	0.0437	0.0597
SoR	0.0180	0.1270	0.1258	0.2020	0.0093	0.0061	0.1987	0.1508	0.1507	0.0218	0.0129	0.0390	0.0437
LGPR	0.0154	0.1223	0.1234	0.1673	0.0072	0.0055	0.1742	0.1394	0.1370	0.0210	0.0114	0.0257	0.0293

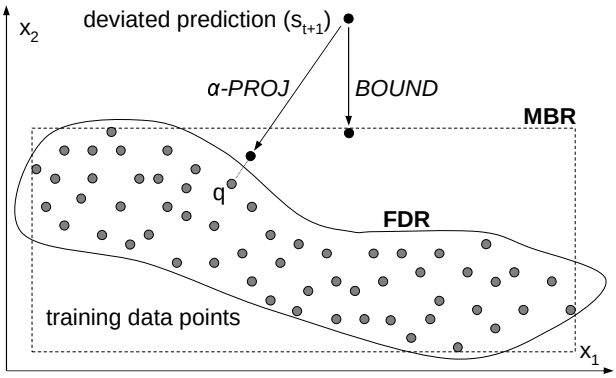

Fig. 3. Comparison of two adjusting methods, α-$PROJ$ and $BOUND$

Of course, the boundaries are those observed in $TrainPool$ and this method is called hereafter $BOUND$. However, there can be a tighter adjustment called α-$PROJ$ as shown in Fig. 3. Let me denote by $q = \min_{p \in TrainPool} \|p - \mathbf{s_{t+1}}\|$; the training data point closest to the deviated prediction $\mathbf{s_{t+1}}$. Then α-$PROJ$ can be written as the following adjustment:

$$\mathbf{s_{t+1}} := (1-\alpha)\mathbf{s_{t+1}} + \alpha q, \quad 0 \leq \alpha \leq 1$$

I used nMSE for each element as an error measurement metric for the one-step prediction error. However, for the sequential prediction error, I used two other different error metrics: the non-cumulative error and the cumulative error. The non-cumulative sequential prediction error is measured for each element as $|\mathbf{s}_t^{i*} - \mathbf{s}_t^i|$, where $\mathbf{s}_t^{i*}$ denotes the i-th element of the actual state at time t. The reason why I call it 'non-cumulative' is that every element of a state $\mathbf{s}$ is non-cumulative. As I already discussed in Section II, each element except the z-position z_w is not cumulative because it is originated from the body frame. Also, z_w does not accumulate due to the gravity. Therefore, it is not possible to observe a cumulative behavior of the sequential prediction error with these elements of $\mathbf{s}$. Instead, I computed (x_w, y_w) from $\mathbf{s}$ and used $\sqrt{(x_w - x_w^*)^2 + (y_w - y_w^*)^2}$ as the cumulative sequential prediction error.

The empirical results for no adjustment, $BOUND$ and α-$PROJ$ with various α, are given in the following section.

VII. EMPIRICAL RESULTS

A. RESULTS OF ONE-STEP PREDICTION

$TrainPool$ and $TestPool$ have 43267 and 19299 data points, respectively. Computation was performed on a laptop with Intel Core-2-Duo L2500 2.0GHz CPU and 2GB RAM.

Table I shows the comparison of performance for all regression methods I discussed above. Different size of training data set was used for each regression method. This is because some of regressors (full GPR and SoR) suffer from very high cost for computation and memory as the size of training data increases. I mentioned in Section IV that storage costs

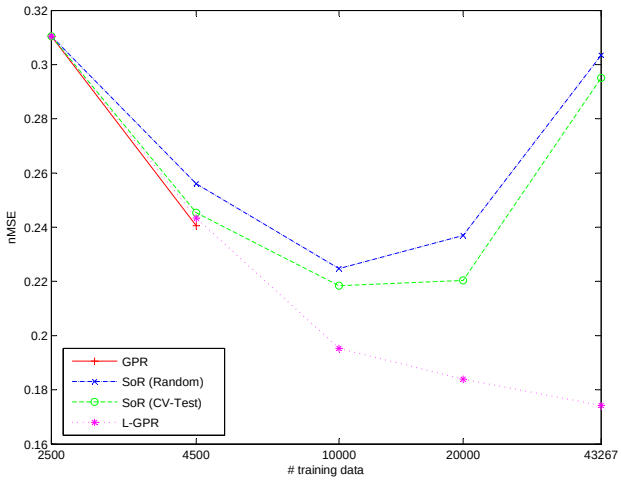

Fig. 4. nMSE of various regression methods for predicting $\Delta\phi$

for full GPR and SoR are $O(n^2)$ and $O(mn)$, respectively. Therefore, full GPR couldn't work with more than 4500 training data due to lack of memory, and SoR couldn't hold mn over 20000000 ($\approx 4500^2$) for the same reason. I carefully optimized each regressor's hyperparameters and determined the size of training data set such that it yields the best performance considering the limitation of the system resource. All of 43267 training data were used for LR, LWPR, and LGPR. 4500 and 10000 training data were used for full GPR and SoR (where, m=2000), respectively. As shown in Table I, GPRs outperformed over LR and LWPR. LWPR was poorer than expected. Among GPRs, LGPR was the best.

Fig. 4 shows nMSE for predicting $\Delta\phi$, for each GPR method, with respect to different size of training data set. For simplicity, I will not show nMSE for every element of a state. Instead, only the hardest element to be predicted, the roll angle difference $\Delta\phi$, will be compared. Fig. 4 describes that the accuracy of most regressors roughly increases as the size of the training data set. However, some regressors suffer from dealing with such large training set. As I mentioned previously GPR couldn't take advantage of training data over 4500, although its performance was the best with training data less than 4500. I tried to fix the number of support data points m of SoR to be 2000 for its best performance. However, mn couldn't be over 20000000, so it was unavoidable to decrease m from 2000 to 1000 and 400 for the case of n=20000 and n=43267. This is why SoR's performance got worse with n=20000 and n=43267. For LGPR, different number of clusters k was used for each case. I set the size of training data in a single local GPR model to be approximately 1000, so that k is 3, 5, 10, 20 and 40 for each case. LGPR excelled over all other regression methods. We can see that only LGPR can handle very large training data set and the accuracy is the best. Fig. 4 also shows that SoR with k-fold-CV test outperformed over SoR with random selection of m support data points.

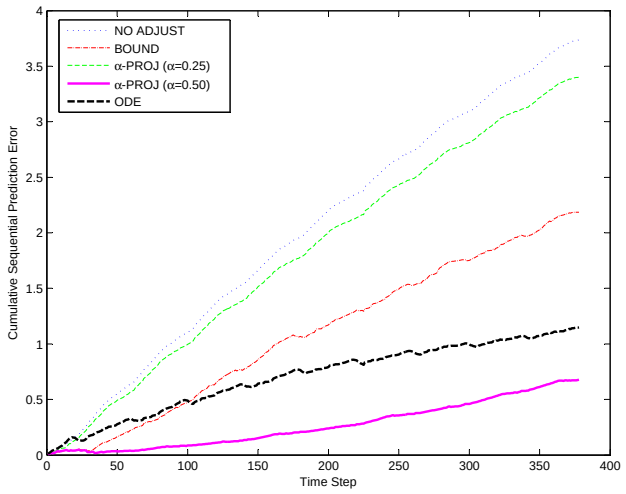

Fig. 5. Cumulative sequential error for the robot moving straight forward

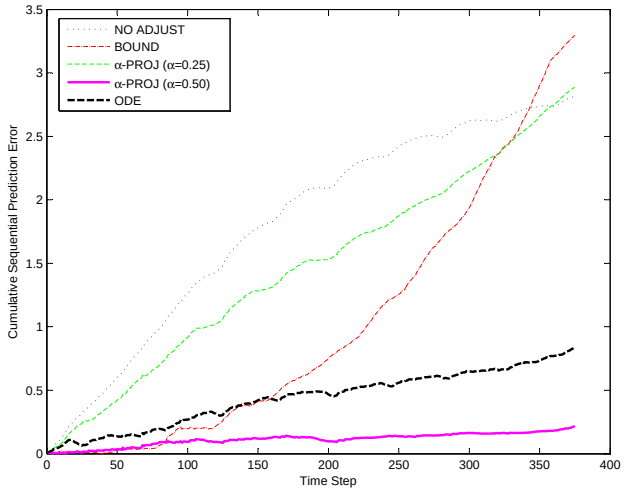

Fig. 6. Cumulative sequential error for the robot turning left backward

B. RESULTS OF SEQUENTIAL PREDICTION

I will compare the two adjusting methods α-$PROJ$ and $BOUND$ in terms of two different error measurement metrics, the cumulative error and the non-cumulative error, as explained in Section VI. First, I will take a look at the cumulative behavior of the sequential error and how two methods relieve it. Second, I will check the non-cumulative error and see if they actually make the sequential prediction converge to FDR. Fig. 5 and Fig. 6 show the cumulative sequential error for LittleDog's two kinds of move (F and BL, see Section II). No adjustment gave the worst result. $BOUND$ and α-$PROJ$ are indeed helpful to reduce the sequential prediction error, but α-$PROJ$, $\alpha > 0.5$, gave the best result. I omitted the result of $\alpha = 0.75$ and $\alpha = 1.00$ because they yielded similar result as $\alpha = 0.50$. At last, the ML Simulator with α-$PROJ$ defeats ODE. In the 3-d visualization of these results, α-$PROJ$, $\alpha > 0.5$ looks very similar to the real robot motion rather than ODE. Note that ODE simulator's parameters are very carefully tuned to make its motion look similar to the real LittleDog. Fig. 7 shows the element-wise error sum for all time steps, $\sum_{t=1}^{T} |\mathbf{s}_t^{i*} - \mathbf{s}_t^i|$; this represents the non-cumulative sequential error. The result implies the convergence criterion for FDR. For this specific problem, α=0.5 is enough for the prediction to converge.

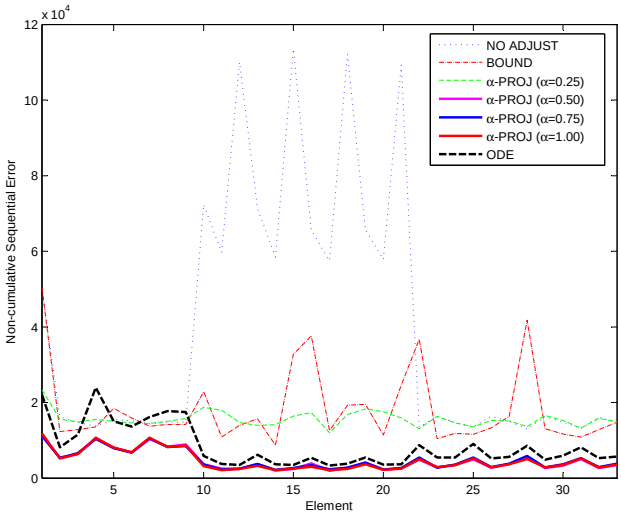

Fig. 7. Element-wise non-cumulative sequential error

VIII. CONCLUSION

I developed a simulator that learns and predicts quadruped trot gaits without explicitly considering the dynamics over it. I invented the local Gaussian process regression (LGPR) that significantly reduces the one-time-step prediction error by taking advantage of a large set of training data. It has k locally full GPR models and predicts with the local model closest to each test data point. I also proposed the projection method called α-$PROJ$ that can prevent the prediction over longer time scales from diverging. By projecting a predicted state to the feasible data region (FDR) in which dynamics constraints are satisfied, α-$PROJ$ greatly reduces the prediction error over longer time scales. Empirical results validate that my approach is more efficient for simulating quadruped trot gaits than the ODE simulator and other methods I tried.

This research can be extended to problems with harder motions such as galloping and jumping. Adding height/slope map of a terrain to a feature set will allow the ML simulator to deal with a bumpy terrain. Local GPR actually approximates the original covariance matrix as the block diagonal one. Allowing the blocks to overlap (one data point is belong to multiple local clusters) can improve the prediction. Multi-task learning can also be applied to reduce the prediction error.

Acknowledgement: I thank Zico Kolter for his helpful advice on this project.

REFERENCES

[1] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental online learning in high dimensions," *Neural Computation*, vol. 17, no. 12, pp. 2602–2634, 2005.

[2] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA: The MIT Press, 2006.

[3] J. P. Duy Nguyen-Tuong, "Local gaussian process regression for real-time model-based robot control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sept. 2008.

[4] M. Seeger, "Some aspects of the spline smoothing approach to non-parametric regression curve fitting," in *J. Roy. Stat. Soc.*, vol. B, 47(1). Springer-Verlag, 1985, pp. 1–52.

[5] G. Wahba, G. Wahba, X. Lin, X. Lin, F. Gao, F. Gao, D. Xiang, D. Xiang, R. Klein, and B. Klein, "The bias-variance trade-off and the randomized gacv," in *Advances in Neural Information Processing Systems*. MIT Press, 1999, pp. 620–626.

[6] A. J. Smola and P. Bartlett, "Sparse greedy gaussian process regression," in *Advances in Neural Information Processing Systems 13*. MIT Press, 2001, pp. 619–625.

[7] J. Quiñonero Candela and C. E. Rasmussen, "A unifying view of sparse approximate gaussian process regression," *Journal of Machine Learning Research*, vol. 6, pp. 1939–1959, 2005.

[8] M. Seeger, "Fast forward selection to speed up sparse Gaussian process regression," in *Workshop on Artificial Intelligence and Statistics 9*, 2003.