ETH zürich

ETH Library

Direct State-to-Action Mapping for High DOF Robots Using ELM

Conference Paper

Author(s):
Hwangbo, Jemin; Gehring, Christian; Bellicoso, Dario; Fankhauser, Péter; Siegwart, Roland; Hutter, Marco

Publication date:
2015

Permanent link:
https://doi.org/10.3929/ethz-a-010508287

Rights / license:
In Copyright - Non-Commercial Use Permitted

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

Direct State-to-Action Mapping for High DOF Robots Using ELM

Jemin Hwangbo[1], Christian Gehring, Dario Bellicoso, Péter Fankhauser, Roland Siegwart and Marco Hutter

***Abstract*— Methods of optimizing a single trajectory are mature enough for planning in many applications. Yet such optimization methods applied to high Degree-Of-Freedom robots either consume too much time to be real-time or approximate the dynamics such that they lack physical consistency. In this paper, we present a method of precomputing optimized trajectories and compressing the information to get a compact representation of the optimal policy function. By varying the initial configuration of a robot and optimizing multiple trajectories, the controller gains knowledge about the optimal policy function. Such computation can be performed on a powerful workstation or even supercomputers instead of an on-board computer of the robot. The precomputed optimal trajectories are stored in a Single-hidden Layer Feedforward neural Network (SLFN) using Optimally Pruned Extreme Learning Machine (OP-ELM). This ensures minimal representation of the model and fast evaluation of the SLFN. We first explain our method using a simple time-optimal control problem with an analytical solution. We then demonstrate how this method can work even for high dimensional state by optimizing a foothold strategy of a full quadruped robot in simulation.**

I. INTRODUCTION

Typical robotic learning algorithms obtain a control policy for a specific task represented by a set of parameters. These parameters can be elements in a look-up table that map states into actions, e.g. Q-learning [1], or numbers that approximates a controller, e.g. spline parameters. There is always a trade-off in parameterization. The more general and less approximated the parameterization is, the higher the number of parameters becomes. As the number of parameters increases, storing and sometimes even evaluating the policy function becomes problematic.

The common goal of typical robotic learning algorithms is to find the optimal parameter set from a given parameterization. Our previous work on parameter optimization [2] has shown that parameterized control policies can be optimized with a limited number of rollouts even when we do not have an analytical solution of the gradient. Policy gradient methods [3], sampling based methods [4] and other numerous methods are designed to find policy parameters.

However, to generate a general state-to-action mapping for a high dimensional system, optimizing the mapping using 'trial and error' basis has limitations due to the fact that a single trajectory (i.e. one episode) only visits a very small part of the state space. To address this issue, methods of learning a policy based on optimized trajectories have been suggested [5], [6]. These methods accumulate multiple optimized trajectories and learn the mapping using a supervised learning algorithm.

The method proposed in the paper differs from them in a few aspects. To learn a high dimensional policy, we have to obtain an appropriate subset of the state space that is likely to be visited. In many cases, the robot will visit only a small fraction of it. Our method explores the state space by disturbances. Along optimized trajectories, we disturb the state and/or action to create a set of new initial conditions and the system is deterministic thereafter. By bounding the disturbance, we can conveniently define the region of interest. After necessary post-processing, we obtain a training data that evenly covers the region of interest. Here we do not model the distribution of the action/state as presented in [6] since the disturbances we are interested in robotics is generally due to external causes (i.e. kicking a robot, stepping on an unexpected object, etc). We cannot model the distribution of such disturbances easily. In stead, we define the region of interest and uniform distribution of state within in. This assumption also completely decouples regression from trajectory optimization such that we do not have to solve them iteratively.

Our method optimizes the paramters as well as the underying parameterization. Optimizing a parameterization means that we find an approximation of the policy function with a low memory requirement and fast evaluation time of the function. A simple parameterization of a policy function such as locally weighted regression, e.g. [7], requires the number of parameters exponential to the dimension of the state. A good parameterization method can bypass this problem by exploiting the underlying structure of the optimal policy function.

We approximate the optimal control policy using Single-hidden Layer Feedforward neural Network (SLFN) and minimize the size of the model using Optimally Pruned Extreme Learning Machine (OP-ELM) [8]. OP-ELM is based on Extreme Learning Machine (ELM) [9] and Multi-Response Sparse Regression (MRSR) [10]. ELM converts a nonlinear regression problem to a linear problem by randomly sampling all the variables except the hidden node weights. Following ELM, MRSR chooses a subset of the hidden nodes that best predicts the target variables. MRSR is a multi-variable version of Least Angle Regression [11].

To demonstrate our method, we first optimize a control strategy of a simple time-optimal problem. Then we apply the same method for optimization of foothold selection policy for a full quadruped robot with 18 Degrees-Of-Freedom (DOF) and compare it to the commonly used Inverted

This research was supported by the Swiss National Science Foundation through the National Centre of Competence in Research Robotics.

All authors are associated with Autonomous Systems Lab (ASL), ETH Zurich, Switzerland [1]jhwangbo@ethz.ch

Pendulum (IP) -based model. The resulting controller is a direct mapping from state to the desired foothold for each foot.

II. Method

Since this section refers to works from different research areas, different words might convey the same meaning.[1]

A. Problem Definition

The dynamical system with state $\boldsymbol{x}(t) \in \mathbb{X} \subset \mathbb{R}^d$, control input $\boldsymbol{u}(t) \in \mathbb{U} \subset \mathbb{R}^m$ $(m \leq d)$ has the form

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{f}(\boldsymbol{x}(t), \boldsymbol{u}(t)), \quad \boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{1}$$

where $\boldsymbol{f}$ is an arbitrary nonlinear time independent function.

We define the control input as $\boldsymbol{u}(\boldsymbol{x}(t), \boldsymbol{a}(t))$ where $\boldsymbol{a} \in A \subset \mathbb{R}^n$ is reduced to as action. $\boldsymbol{a}^*$ is the optimal action that minimizes the expected value of a cost function J, which is defined over a finite time horizon $[0, t_e]$ as

$$J = \mathrm{E}[h(\boldsymbol{x}(t_e)) + \int_0^{t_e} g(\boldsymbol{x}(t), \boldsymbol{u}(\boldsymbol{x}(t), \boldsymbol{a}(t)), \mathcal{C})dt]. \tag{2}$$

h is a terminal cost and g is a cumulative cost. $\boldsymbol{u}$ is a control input and $\mathcal{C}$ is a higher level command to the controller. For example, $\boldsymbol{a}$ can be a reference trajectory of a joint position and $\boldsymbol{u}$ can be a set of joint torques commanded to the motors to follow that trajectory. $\mathcal{C}$ is the command or task variable which can be either a command from a different controller or a command from a human operator. It is important to note that we consider J as a function of $\mathcal{C}$. In case the robot cannot follow the command from the human perfectly, it should balance the degree of violation of the command and the rest of the cost function. For this reason, we define the command as a soft constraint.

Due to the presence of $\mathcal{C}$, our optimal action is no longer defined by the state $\boldsymbol{x}$ only. To avoid a confusion, we introduce another variable $\boldsymbol{X} = [\boldsymbol{x}^T, \mathcal{C}^T]^T \in \mathbb{X}_t$. The optimal action is now a function of $\boldsymbol{X}$ only.

In existing optimal control methods, the goal is to find an optimal policy $\pi^* : \boldsymbol{X} \rightarrow \boldsymbol{a}^*$. Our goal is to find $\pi^\dagger : \boldsymbol{X} \rightarrow \boldsymbol{a}^\dagger$ which satisfies

$$\begin{aligned} \pi^\dagger &= \arg\min_{\pi} \int_{\boldsymbol{x}} \triangle \boldsymbol{a}^T \boldsymbol{M} \triangle \boldsymbol{a} \; d\boldsymbol{X}, \\ \triangle \boldsymbol{a} &= \boldsymbol{a}(\boldsymbol{X}|\pi^*) - \boldsymbol{a}(\boldsymbol{X}|\pi), \\ \text{subject to} \quad & \mathcal{S} < \tau_{\mathcal{S}} \end{aligned} \tag{3}$$

where $\mathcal{S}$ is the size of the memory required to store the policy and $\tau_{\mathcal{S}}$ is the threshold for memory size. $\boldsymbol{M}$ is a positive definite constant metric tensor that defines distance in the action space.

It is most likely we cannot express $\boldsymbol{a}(\boldsymbol{X}|\pi^*)$ with common elementary functions. If we can, it is probably due to the fact that $\boldsymbol{a}(\boldsymbol{X}|\pi^*)$ has an analytical solution. We want to express $\boldsymbol{a}(\boldsymbol{X}|\pi^*)$ with high accuracy in a limited memory space.

[1]"Basis function" in regression can also be referred as "kernel", "activation function" or "feature". We use "kernel" since it is used in [8]. "Artificial neural network" can be shortened as "neural network" in this paper and a few others. SLFN is sometimes referred as ELM network when it works with ELM.

B. Approximation of the Reconstruction Error

Evaluation of $\triangle \boldsymbol{a}$ requires $\boldsymbol{a}(\boldsymbol{X}|\pi^*)$ which is not available in our problems. We approximate this quantity using optimal trajectories. It is easier to find $\boldsymbol{a}_k^*(t)$, which is the optimal action sequence from a given initial condition $\boldsymbol{X}(0) = \boldsymbol{X}_k$ and $0 < t < t_f$. There exists many methods of optimizing a single trajectory [12]. These methods give us pairs of $\boldsymbol{a}_k^*(t)$ and $\boldsymbol{X}_k^*(t)$ which satisfies $\boldsymbol{a}_k^*(t) = \boldsymbol{a}(\boldsymbol{X}(t)|\pi^*)$. Then we numerically approximate the minimization in (3) as

$$\begin{aligned} \pi^\dagger \approx \tilde{\pi}^\dagger &\equiv \arg\min_{\pi} \sum_{L} \epsilon^T \boldsymbol{M} \epsilon, \\ \epsilon &= \boldsymbol{a}_k^*(t) - \boldsymbol{a}(\boldsymbol{X}(t)|\pi), \\ \text{subject to} \quad & \mathcal{S} < \tau_{\mathcal{S}}, \end{aligned} \tag{4}$$

where L is a subset of $D = \{(\boldsymbol{X}_k^*(t), \boldsymbol{a}_k^*(t)) | k \in K, \; 0 < t < t_f\}$ and K is a set of initial conditions where we obtain optimal trajectories. In order for $\tilde{\pi}^\dagger$ to be a good numerical approximation of $\pi^\dagger$, at least in the region of interest, we need training points uniformly distributed over either $\mathbb{X}_t$ or the region of interest. It is likely that the points are densely located near a certain manifold (e.g. limit cycle). Therefore, using D directly for learning the mapping will likely result in unsatisfactory results. Assuming sufficiency of training points, we sparsify the data by rejecting one of two points that are closer than a given threshold. We use Mahalanobis metric to define the distance of two points.

If the dimension of the state space is low, we can subsample K to get a low-discrepancy set in $\mathbb{X}_t$. However, if that is not the case, it might be too time-consuming to do so. Alternatively, we define a disturbance space $O = E \times A$, where E is a space of possible external disturbances and the symbol $\times$ represents a Cartesian product. We sample a disturbance $o \in O$ and apply it to already optimized trajectories to get new initial conditions. Here we make an assumption that the state can be disturbed only by external disturbances and mistakes in commanding actions. This method is simplified for systems with a unique optimal limit cycle (e.g. legged locomotion on a flat terrain) since all trajectories will approach to the limit cycle.

C. Policy Representation

In this work, we choose SLFN to store our policy. The structure of our SLFN can be mathematically modeled as

$$a_j = \sum_{i=1}^{N} \beta_{ij} z_i(\boldsymbol{w}_i \boldsymbol{x} + \boldsymbol{b}_i), \tag{5}$$

where β_{ij} is the connection weight between ith hidden node and jth output layer, $\boldsymbol{w}_i = [w_{1i}, w_{2,i}, ..., w_{di}]$ is the connection weights between input layer and ith hidden node and z_i is the ith hidden node. Following the SLFN format, we can redefine the problem as minimizing the reconstruction error of SLFN with a given number of hidden layer nodes.

D. Optimization of SLFN

Suppose that we have obtained some of the optimal actions $\boldsymbol{a}_k^*(t)$ and their corresponding states $\boldsymbol{X}_k^*(t)$, MRSR [10] first

normalizes $\boldsymbol{a}_k^*(t)$ and $\boldsymbol{X}_k^*(t)$ to unit normal distributions. Then, given a pool of the hidden layer kernel G, MRSR sequentially selects an kernel from G and adds it to a SLFN, which is initialized to zero hidden node. The added kernel is the one which has the highest sum of absolute correlation with the residuals from the previous SLFN prediction. Consequently, if we stop the algorithm at p^{th} step, we obtain a smaller pool of kernels $H \subset G$ with $|H| = p$ that accurately predicts the action. The expression $|H|$ represents the cardinality of H.

To obtain G, we sample $\boldsymbol{w}$, $\boldsymbol{b}$ and types of z. For this step, we are free to use a different regression model. However, as Miche et. al. [8] suggested, SLFN structure can successfully reconstruct many different types of functions.

We dropped the Leave-One-Out validation method of OP-ELM since we have a fixed memory size and it is computationally expensive to rank the whole G. This modification leads to a larger G and consequently a lower average reconstruction error for the same number of kernels. Overfitting is generally not a problem since $|D| \gg |H|$ but we check the quality of the regressor with a separate training set.

III. Demonstration With a Simple Task

To illustrate our method, we introduce a simple problem. We want to find time-optimal control of

$$\begin{aligned} &\dot{x}_1 = x_2 + u_1, \quad \dot{x}_2 = u_2 \\ &x_1(T) = x_2(T) = 0 \\ &\boldsymbol{U} = \{(u_1, u_2), u_1^2 + u_2^2 \leq 1\}. \end{aligned} \tag{6}$$

This problem was originally introduced by Pontryagin et. al. [13] and the numerical solution is described in [14]. We want to find the time-optimal trajectory and its corresponding action trajectory. Problem (6) can be converted to boundary value problem using Pontryagin's minimum principle as,

$$\begin{aligned} &\dot{x}_1 = x_5(x_2 + \frac{x_3}{\sqrt{x_3^2 + x_4^2}}), \\ &\dot{x}_2 = x_5 \frac{x_4}{\sqrt{x_3^2 + x_4^2}}, \\ &\dot{p}_1 = 0, \ \dot{p}_2 = -x_5 x_3, \ \dot{x}_5 = 0, \\ &x_1(1) = 0, \ x_2(1) = 0, \ x_3(1)^2 + x_4(1)^2 = 1, \end{aligned} \tag{7}$$

By solving the converted problem with a given initial condition, we obtained the sequence of actions as

$$u_1 = \frac{x_3}{\sqrt{x_3^2 + x_4^2}}, \ u_2 = \frac{x_4}{\sqrt{x_3^2 + x_4^2}}. \tag{8}$$

We generated 1,000 trajectories by solving this boundary value problems with initial condition for both x_1 and x_2 (i.e. K) sampled from $\mathcal{U}(-2.0, 2.0)$, which is a continuous uniform distribution in the given interval. Note that these trajectories are numerically optimized even though they are very close to the true optimal solution due to the simplicity of the problem.

We extract about 10,000 state-action pairs (i.e. L) from all the trajectories and sample 20,000 random kernels consists of linear, sigmoid and Gaussian kernels for regression. We additionally generate 100 trajectories with random initial states and extract about 1,000 state-action pairs as a testing set. Using MRSR, we choose the best 300 kernels which results in 0.08 error from the testing set. Higher number of kernels resulted in lower testing error but we picked a reasonable number to demonstrate low memory requirement of our method. We will call this policy π_{ELM}. We simulate the system with 5 different initial points sampled from $\mathcal{U}(-1.5, 1.5)$ and with both policies. We also find the optimal trajectory obtained from boundary value problem solver, $bvp4C$ function in MATLAB. Even though this trajectory is not the true optimum due to the fact that it is a numerical solution, it is very close to the true optimum and we call it an optimal trajectory for simplicity. Since none of the methods can reach the final point perfectly, we terminate the simulation if $||\boldsymbol{x}|| < 0.01$ or $t > 5.0$.

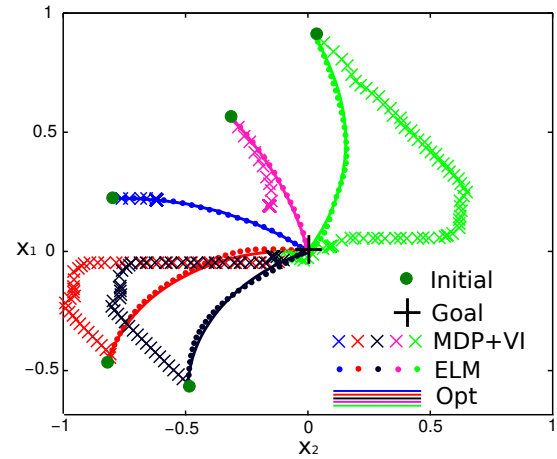

Fig. 1: Trajectories generated by the ELM-based controller and the controller obtained from MDP+VI are plotted. Optimal trajectories obtained using Pontryagin's minimum principle (Opt) are plotted as a reference.

We also tested a common method which is modeling the system as a Markov Decision Process (MDP) and solving it by Value Iteration (VI). To model the system as MDP, we need to convert it to a discrete system with bounded state. We build the discrete set of states S, discrete set of possible actions A, the transition matrix T and the transition cost R. We force bounding of the states simply by projecting it to the surface when it is outside the state bounds, $(-2.0, 2.0)$. We used discretization level $n = 85$, which means that there are 85×85 discrete states and 85×85 possible actions. We obtain discrete mapping π_{MDP}^* using value iteration and linearly interpolate it for control.

The summary of the result can be found in Tab. I and Fig. 1. Note that none of the trajectories from MDP approached the terminal state close enough to terminate. π_{ELM} generates much better solutions with less memory space. The biggest problem of MDP is the discretization of the system dynamics. The value function is formed by small groups in where the cost-to-go's are the same. Consequently, the controller simply finds the first one it sees as the optimal

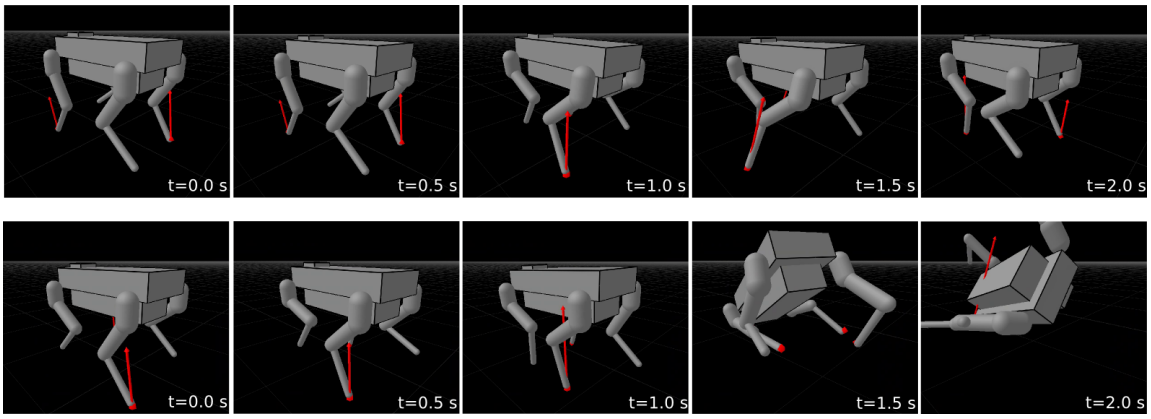

Fig. 2: StarlETH model responds to an Impulse ($I_x = 20, I_y = 15$ kgm/s) following a command ($v_x = 0.6$ m/s, $\omega_z = 0.0$ rad/s) with optimized footholds (Top) and with IP based controller (Bottom). Red arrows represent the ground reaction forces.

TABLE I: Quality of Trajectories

	N. to store	Avg. time taken	Avg. time to compute
Our method	906	1.047 s	0.005 s
MDP	14,450	5.0 s	N/A
Optimal	N/A	0.990 s	0.52 s

control and becomes biased to one direction. Such weakness is manifested in the trajectory in Fig. 1. This problem can be solved with much finer discretization level but $n = 85$ case already takes too long to be practical. Note that the size of the transition matrix T is already 2×85^2. We only have two rows since the system is deterministic and we only store the indices of the states. Computation time to build a SLFN takes a couple of minutes including the generation of optimal trajectories whereas the computation time to solve MDP with VI is several hours. Computation time to evaluate the SLFN is two magnitudes lower than generating an optimal trajectory for this particular problem but the gap is generally bigger for high dimensional systems.

IV. Demonstration with a High Dimensional System

In this section, we will show an example of optimization of foothold strategy of a quadruped robot. The system we have chosen is a model of StarlETH [15]. StarlETH has 4 legs and 3 actuators per leg forming hip abduction/adduction (HAA), hip flexion/extension (HFE), and knee flexion/extension (KFE) joints. It is about 25 kg in weight and 60 cm in height.

The human operator gives a velocity command to the robot $\mathcal{C} = [v_x, \omega_z]$ which is a set of heading velocity and turning rate. The robot should learn how to follow the command from the human and decide where to step while considering its current state. In this way, the controller can also work with a high level planner which plans the global path of the robot.

Our work will be built on top of a locomotion controller described in [16]. The controller decides where to step using an IP model which is a standard model in legged robot control [17], [18]. The parameters of the IP model is hand tuned to work the best for StarlETH. Our goal is to develop a new foot placement strategy that is more robust in disturbance.

The gait chosen for this task is walking trot. The gait pattern generation defines the contact timing and virtual model controller [19] stabilizes the torso.

All the simulations for optimization and testing are performed in MULE [20] and ODE [21] environment. In this work, MULE is used for optimization mainly due to its speed and accuracy in dynamics and ODE is used for validating the final controller.

A. Task

The task of a single trajectory optimization is to trot for 5.0 seconds while maintaining stable torso height and orientation. The cost function which can be written as

$$J = \sum_{0 \leq i \leq N_c} w_i C_i(\boldsymbol{\theta}, \boldsymbol{q}, \dot{\boldsymbol{q}}, \boldsymbol{\tau}, \mathcal{C}). \tag{9}$$

It consists of four different cumulative costs which are energy consumption, control command violation, height instability and orientation instability. There are also three limit costs that penalize the violation of the hard limits joint position, joint velocity and power output. These costs are acting as constraints to guide the optimizer to a feasible solution. Lastly, there is one failure cost which penalizes the robot falling.

B. Optimization of Individual Trajectories

We use the optimization framework used in [22]. The framework is rollout-based and requires many forward simulation of the dynamics. Currently it is running on four cores of an Intel i7-3740QM and the total optimization time is 3 days.

Generating a good set of initial condition K is very important for the final result. Since the dimensionality of O is very large and our computation power is limited, we picked impulses on the main body as our disturbances. The intuition is from the IP-based controllers, which only considers the main body velocity and still shows a great performance

in many robots. We apply $-20 < I_x < 20$ kgm/s and $0 < I_y < 15$ kgm/s randomly where x is the heading axis and y is the lateral axis. Note that we apply only positive impulses in y axis since the robot is symmetric but both negative and positive impulses in x axis since the robot is only running forward in the training. This is not the only correct combination but a correct combination is important not to have a redundant data set.

We use the symmetry of the robot to speed up the training. The optimized footholds for all four feet can be transfered to all other feet by changing the coordinate system of $\boldsymbol{x}$ and $\boldsymbol{a}$. Consequently, we only have to train one model.

For optimizing such a highly unstable system, it is important to have a good initial policy. We will set the initial policy using IP-based controller which is known to be stable in most cases. We then optimize the correction vector to the IP-based controller.

Starting from no impulse case, we gradually increase the magnitude of the impulse. To speed up the convergence, we get the initial guess of the optimal footholds by averaging previously optimized footholds with similar impulse cases. Therefore, computation time is sub-linear to the number of trajectories. Optimization of one of the impulse cases is depicted in Fig. 2. The optimized policy on the top is able to avoid falling whereas the IP-based controller cannot.

Figure. 3 shows scatter plot of the obtained data projected on a 2D plane formed by y-component of the torso velocity and y-component of the foothold position. It can be clearly observed that these two variables are highly correlated. This result shows the same trend as in IP-based controller when the yaw rate of the torso is zero. However, by varying the yaw rate, we can obtain very dramatic change in the action. This shows that our policy incorporates the effect of all variables.

C. Building $\pi^\dagger$

The last step of our method is compressing the previously optimized actions. 880 different trajectories were used to extract a set of 60,000 state-to-action tuples L.

We assume that the state of the stance legs are well described by the body states due to the fact that the robot operates relatively close to the limit cycle. Therefore, we simplified the leg states to the swing phase. The human operator is commanding velocities in body frame so there are only nine relevant states from the main body, namely six velocities and three positions (height, roll and pitch). Including the command $\mathcal{C} = [v_x, \omega_z]$ and the gait phase in $\boldsymbol{X}$, we get $\boldsymbol{X} \in \mathbb{R}^{12}$. The action we get is the location of the desired foothold relative to the hip, which gives us $\boldsymbol{a} \in \mathbb{R}^2$. It is also possible to model each axes of $\boldsymbol{a}$ separately but it will results in a bigger model. By having a single model, many of the hidden nodes are reused for different variables.

OP-ELM requires predefined kernel types. We use linear and sigmoid kernels for this problem. Gaussian, trigonometric and polynomial kernels are all good choices in general due to their simplicity and low cost of computation. We sample 30,000 kernels that best fit to the testing set. We obtain testing set by optimizing 100 trajectories with initial condition generated with random impulses. The average reconstruction error from the testing set with 1,000 kernels is 7 mm.

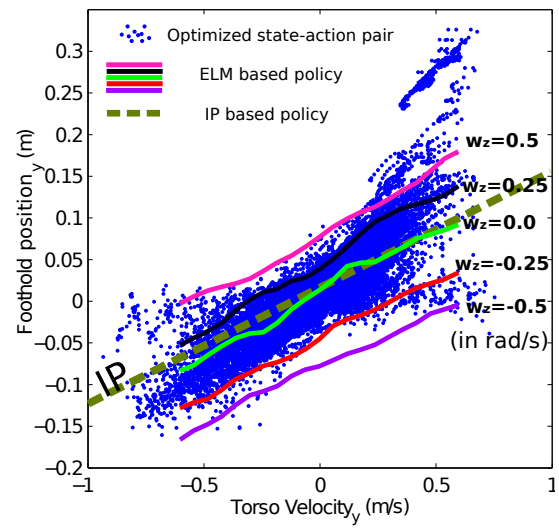

Fig. 3: Training data projected on 2D plane formed by y-axis of the torso velocity and y-axis of the foothold position is plotted (blue dots). IP-based foothold position is plotted in dark green dotted line. In the case of our optimized policy, we vary both yaw rate and lateral velocity and plot the output with solid lines with different colors.

D. Result and Discussion

The resulting SLFN is stored in a single XML file which takes less than 600 kB. Evaluation time of SLFN is around 68 μs on one core of an Intel i7-3740QM.

The resulting controller reads the continuous velocity command of $\boldsymbol{x}_{com} = [v_x, \omega_z]$ from an operator and state $\boldsymbol{x}_{int}$ from the robot and outputs a foothold for each foot accordingly. We first tested our controller with a joystick command while the robot is free to move. Fig 4 shows how the controller responds when it is disturbed with a 4 kg ball thrown at 6 m/s while following velocity command of v_x =0.4 m/s and ω_z=0.25 rad/s. The related video demonstration can be found at: http://youtu.be/9h17wxgaIlM

More quantitative analysis is done with impulses. We test our controller and the IP-based controller for 1,000 randomly sampled disturbances and measured the failure rate (torso hitting the ground) and the average cost for the successful cases for both controllers. The disturbances are sampled in a uniform distribution as $I_x \sim U(-20, 20)$ and $I_y \sim U(0, 15)$ in a unit of kgm/s. We also randomly sample the impact timing to make sure that we measure the robustness for the entire gait cycle. The result of the simulation is shown in Fig. 5. The optimized controller fail only 19.1 % of the time whereas the IP-based controller fail 29.3 %. The average costs for successful cases are 905.0 and 1202.2 respectively.

V. CONCLUSIONS

We presented a method of compressing optimized trajectories in a form of direct mapping from states to action. Since

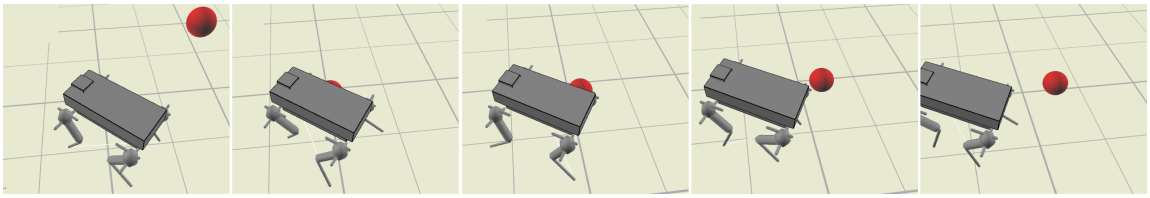

Fig. 4: The final foothold controller responds to the disturbance by a 4 kg ball thrown at 6 m/s in ODE environment. $v_x = 0.6$ m/s and $\omega_z = 0.25$ rad/s is commanded by the human operator.

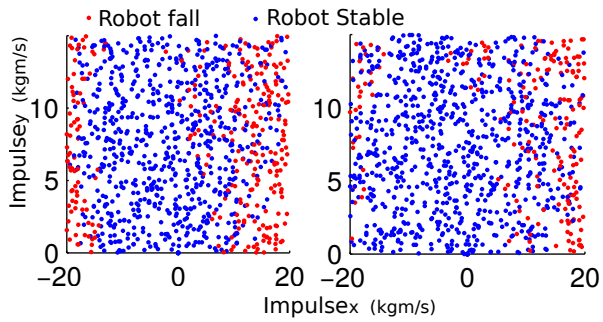

Fig. 5: The robot is disturbed at 1,000 different impulse cases with both IP-based controller (left) and optimized controller (right) with the failure rate of 29.3% and 19.1% respectively.

this method allows to precompute the optimal trajectories, more computational power and time were available. This allowed us to use more accurate model for optimization which was not possible for controllers that run in real time on real robots. To overcome "the Curse of dimensionality" of the current policy representation methods, we employed OP-ELM to learn and compress the policy. The resulting mapping from states to action is compact and can be evaluated in very short time. We first demonstrated our method with a simple time-optimal control problem. It worked almost as well as the optimum trajectories with very little computation time. Our method also successfully built a controller that outputs desired foothold locations for a high DoF quadruped robot. The resulting controller file was less than 600 kB in XML format and its computational cost was also very low.

We are planning to work on building more memory efficient and parallelizable algorithm to regress the function. We will also investigate on building the whole robot control using our method.

References

[1] C. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[2] J. Hwangbo, C. Gehring, H. Sommer, R. Siegwart, and J. Buchli, "Rock*-efficient black-box optimization for policy learning," in *International Conference on Humanoid Robots*. IEEE/RAS, 2014.

[3] J. Peters and S. Schaal, "Policy gradient methods for robotics," in *International Conference on Intelligent Robots and Systems*. IEEE/RSJ, 2006, pp. 2219–2225.

[4] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[5] I. Mordatch and E. Todorov, "Combining the benefits of function approximation and trajectory optimization," in *Robotics: Science and Systems (RSS)*, 2014.

[6] S. Levine and V. Koltun, "Learning complex neural network policies with trajectory optimization," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 829–837.

[7] C. G. Atkeson, A. W. Moore, and S. Schaal, "Locally weighted learning for control," in *Lazy learning*. Springer, 1997, pp. 75–113.

[8] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, "Op-elm: optimally pruned extreme learning machine," *Neural Networks, IEEE Transactions on*, vol. 21, no. 1, pp. 158–162, 2010.

[9] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: theory and applications," *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.

[10] T. Similä and J. Tikka, "Multiresponse sparse regression with application to multidimensional scaling," in *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*. Springer, 2005, pp. 97–102.

[11] B. Efron, T. Hastie, I. Johnstone, R. Tibshirani, *et al.*, "Least angle regression," *The Annals of statistics*, vol. 32, no. 2, pp. 407–499, 2004.

[12] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[13] L. Pontryagin, V. Boltyanskii, R. Gamkrelidze, and E. Mishchenko, *The Mathematical Theory of Optimal Processes*. Interscience, 1962.

[14] S. Avvakumov and Y. N. Kiselev, "Boundary value problem for ordinary differential equations with applications to optimal control," *Spectral and Evolution Problems 2000*, vol. 10, 2000.

[15] M. Hutter, C. Gehring, M. Bloesch, M. Hoepflinger, C. D. Remy, and R. Siegwart, "Starleth: A compliant quadrupedal robot for fast, efficient, and versatile locomotion," in *Int. Conf. on Climbing and Walking Robots (CLAWAR)*, 2012.

[16] C. Gehring, S. Coros, M. Hutter, M. Bloesch, M. A. Hoepflinger, and R. Siegwart, "Control of dynamic gaits for a quadrupedal robot," in *International Conference on Robotics and Automation*. IEEE, 2013, pp. 3287–3292.

[17] T. Sugihara, Y. Nakamura, and H. Inoue, "Real-time humanoid motion generation through zmp manipulation based on inverted pendulum control," in *International Conference on Robotics and Automation*, vol. 2. IEEE, 2002, pp. 1404–1409.

[18] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by a simple three-dimensional inverted pendulum model," *Advanced Robotics*, vol. 17, no. 2, pp. 131–147, 2003.

[19] J. Pratt, C.-M. Chew, A. Torres, P. Dilworth, and G. Pratt, "Virtual model control: An intuitive approach for bipedal locomotion," *The International Journal of Robotics Research*, vol. 20, no. 2, pp. 129–143, 2001.

[20] C. Gehring, R. Diethelm, R. Siegwart, G. Nützi, and R. I. Leine, "An evaluation of moreaus time-stepping scheme for the simulation of a legged robot," in *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. American Society of Mechanical Engineers, 2014.

[21] E. Drumwright, J. Hsu, N. Koenig, and D. Shell, "Extending open dynamics engine for robotics simulation," in *Simulation, Modeling, and Programming for Autonomous Robots*. Springer, 2010, pp. 38–50.

[22] C. Gehring, S. Coros, M. Hutter, M. Bloesch, P. Fankhauser, M. A. Hoepflinger, and R. Siegwart, "Towards automatic discovery of agile gaits for quadrupedal robots," in *IEEE International Conference on Robotics and Automation*. IEEE, 2014, pp. 4243–4248.