Generalization of Optimal Motion Trajectories for Bipedal Walking

Alexander Werner† Dietrich Trautmann† Dongheui Lee* Roberto Lampariello†

***Abstract*— Control of robot locomotion profits from the use of pre-planned trajectories. This paper presents a way to generalize globally optimal and dynamically consistent trajectories for cyclic bipedal walking. A small task-space consisting of stride-length and step time is mapped to spline parameters which fully define the optimal joint space motion. The paper presents the impact of different machine learning algorithms for velocity and torque optimal trajectories with respect to optimality and feasibility. To demonstrate the usefulness of the trajectories, a control approach is presented that allows general walking including transitions between points in the task-space.**

I. Introduction

As many aspects of robotics technology improve, robots cover more tasks such as manipulating objects in complex environments. Often designed with humans in mind, these environments feature challenges like stairs or uneven terrain. It is therefore desirable that robots have matching abilities when it comes to locomotion in these environments. This motivates the use of legged locomotion for our robotic platforms.

However, legged locomotion comes with rather complex kinematics and dynamics which still provide plenty of challenges for planning and control. All approaches create a mapping between the task of moving the robot somewhere and a suitable trajectory which deals with the full complexity of the robot. Models of varying complexity [1] are used to create this mapping. Creating a trajectory using a model that contains all degrees of freedom and takes into account the systems limitations is however not feasible when the solution should be created on-line, taking into account the task and the environment.

The approach developed in this work, first presented in [2], provides this mapping from the task-space to the trajectory. For this it uses a combined approach of trajectory planning using non-linear optimization and generalization of these results by current machine learning methods. The optimization allows exploitation of the dynamics of the system by using strong models. This paper contains an analysis of the performance of a set of machine learning algorithms in providing this mapping for a successful execution of the task.

* Chair of Automatic Control Engineering, Department of Electrical and Computer Engineering, Technische Universität München (TUM)

† Institute of Robotics and Mechatronics, German Aerospace Center (DLR)

We have selected the stride length and cycle time as task-space variables, allowing arbitrary cyclic walking on flat ground. Two cost functions, joint velocity and joint torque, were evaluated, posing different levels of complexity for the machine learning. We computed dense data sets for a task-space grid on which a set of machine learning algorithms were applied. The quality of the machine learning results was evaluated for different sampling densities of the task-space. For practical application, the required density is always a compromise between computation time and loss in optimality, with a lower bound defined by feasibility of the generated trajectories. The paper analyses this trade-off.

The method is applied to 2D bipedal walking with 6 joints and flat feet stepping on level ground. The system has realistic mass and actuator properties that are inspired by existing robots. All system limitations found to be relevant for a successful execution in [1] are respected by the trajectories. The planning approach presented here provides cyclic gait patterns for any required task which lies within the learned region. The pattern can be computed from the learned model within an execution time that enables its use in reactive on-line step planning algorithms. The method is general and can be applied to three dimensional walking robots as well.

The paper is organized as follows: after presenting the related work in Section II, the problem is described in Section III. A more detailed explanation of the optimization problem and the methods to generate the data set used in the machine learning is given in Section IV and Section V. The following Section VI described the applied machine learning algorithms and their parameterization. Section VII gives results for the machine learning process which are validated on different levels. A simulation demonstrates the use of the trajectories in combination with a feed-back. The paper provides a discussion in Section VIII and Section IX.

II. Related Work

Using a similar approach the task of robotic ball-catching was studied in [2]. There, a mapping from a ball velocity vector to an optimal robot trajectory to catch the ball is learned. This allows real-time execution while respecting the dynamic constraint of the robot.

Learning approaches are also applied differentially onto existing trajectories to compensate for the tracking errors of feed-back control schemes build on reduced models. [3] presents a way to reduce the tracking error

of the Zero-Moment Points for bipedal walking by using iterative learning control. The approach uses a simplified model and compensates for the effects of the neglected dynamics using machine learning.

In [1] we present an optimization problem to solve three dimensional bipedal walking. The approach uses an instantaneous double-support phase. This was resolved in [4] with a 2-phase walking motion, with a time-variant optimized contact force distribution. Additionally, trajectory generation for serial-elastic robots is addressed. This work borrows the formulation of the optimization problem from [4] and applies it to a planar, rigid, bipedal robot.

III. Problem Statement

In qualitative terms, the task is to provide motions which make a bipedal robot traverse a horizontal distance by attaching and detaching selected contacts with the ground in a cyclic pattern. When looking at the geometry of one cycle, the motion can be described by the stride length of one step k_s. The motion is divided into two phases, the single- and the double-support phase. The time allowed for one step, comprised of both phases, is specified as k_t. These two parameters make up the task specification:

$$\boldsymbol{k} = \begin{bmatrix} k_s \\ k_t \end{bmatrix} \tag{1}$$

The trajectory is defined in the joint space of the robot $\boldsymbol{q} \in \mathbb{R}^{N_{\text{JOINTS}}}$ through a B-spline function $\boldsymbol{q} = f_{\text{SPLINE}}(\boldsymbol{p}, t)$ with the parameters $\boldsymbol{p} \in \mathbb{R}^{N_{\text{SPLINE}}}$. The trajectory is required to fulfill the robot kinematics and dynamics.

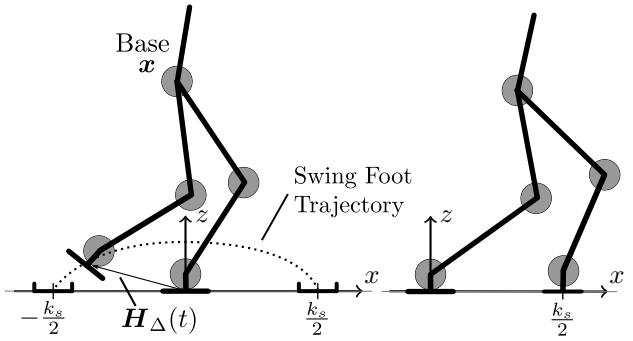

Fig. 1. Kinematic task of a symmetric step. Left: Schematic motion of the swing foot during the single-support phase with start and goal foot step locations. Right: Closed kinematic Loop during the double-support phase.

The task $\boldsymbol{k}$ determines the location of the foot at start and during the double-support phase as displayed in Fig. 1:

$$\begin{aligned} \boldsymbol{H}_\Delta(0) &= \boldsymbol{H}_{start} \quad s_{start} = -s_{goal} = \tfrac{k_s}{2} \\ \boldsymbol{H}_\Delta(t) &= \boldsymbol{H}_{goal} \quad \forall t \in [r_p \cdot k_t; k_t] \end{aligned} \tag{2}$$

where $\boldsymbol{H}_\Delta$ is the homogeneous transformation matrix of the relative position of the swing foot to the stance foot. This matrix contains only a translation in x of magnitude s_{start} and s_{goal} respectively. The ratio between phase time of the single-support phase and time of the complete step is defined by r_p. The stance foot is assumed to have the position $\boldsymbol{I}$. This assumption allows the computation of the full base state $\boldsymbol{x}$ (shown in Fig. 1) contained in the full system state $\boldsymbol{y}$ using a kinematic relation f_{KIN}:

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{q} \end{bmatrix} \qquad \boldsymbol{x} = f_{\text{KIN}}^{-1}(\boldsymbol{q}) \tag{3}$$

given only the joint space $\boldsymbol{q}$. The dynamics of the system, described in [4], are given by:

$$\boldsymbol{M}(\boldsymbol{y})\ddot{\boldsymbol{y}} + \boldsymbol{C}(\boldsymbol{y}, \dot{\boldsymbol{y}})\dot{\boldsymbol{y}} + \boldsymbol{g}(\boldsymbol{y}) = \boldsymbol{S}^T\boldsymbol{\tau} + \sum_{i=0}^{N_C} \boldsymbol{J}_i^T(\boldsymbol{y})\boldsymbol{W}_i, \tag{4}$$

with the required joint torques $\boldsymbol{\tau}$ and generated contact forces $\boldsymbol{W}_i$. $\boldsymbol{\tau}$ must satisfy the robots hardware limitations, whereas $\boldsymbol{W}_i$ must lie within the set of forces that keep a stable contact.

Additionally the goal is to provide an optimal trajectory with respect to one of the following cost functions:

$$\begin{aligned} \Gamma_{\dot{q}}(\boldsymbol{p}) &= \int \dot{\boldsymbol{q}}^T \cdot \dot{\boldsymbol{q}} \; \mathrm{dt} \\ \Gamma_{\tau}(\boldsymbol{p}) &= \int \boldsymbol{\tau}^T \cdot \boldsymbol{\tau} \; \mathrm{dt} \end{aligned} \tag{5}$$

Essentially, the mapping between $\boldsymbol{k} \rightarrow \boldsymbol{p}$ can be revealed through the solution of an optimization problem, as described in the following Section IV. Considerable computation power and time is required to compute $\boldsymbol{p}$ given $\boldsymbol{k}$ such that the resulting trajectory is optimal and feasible given the non-linear cost function and constraints.

To provide a solution quickly enough that it can be used in a robot, where $\boldsymbol{k}$ is known only a very short time in advance, the mapping should be generalized through machine learning algorithms. The algorithms applied are described in Section VI.

To generate the data set for generalization, the function is sampled at a set of training points $\boldsymbol{k}_{\text{TRAIN}}$ yielding a set of points $\boldsymbol{p}_{TRAIN}$. This data set must be sufficiently smooth in $\boldsymbol{k}$ to lead to acceptable results. The computation of this data set is described in Section V.

IV. Optimization Problem

The requirements on the trajectory are formulated as inequality constraints $\boldsymbol{h}(\boldsymbol{p})$ and equality constraints $\boldsymbol{e}(\boldsymbol{p}, \boldsymbol{k})$ in the optimization problem:

$$\begin{aligned} \underset{\boldsymbol{p}}{\text{minimize}} \quad & \Gamma(\boldsymbol{p}) \\ \text{subject to} \quad & \boldsymbol{e}(\boldsymbol{p}, \boldsymbol{k}) = 0 \\ & \boldsymbol{h}(\boldsymbol{p}) < 0 \end{aligned} \tag{6}$$

The inequality constraints contain joint position and joint velocity limits which are linear in $\boldsymbol{p}$. To compute collision free trajectories, a safe distance between the swing foot and the ground is enforced as inequality constraint which are non-linear in $\boldsymbol{p}$. Additionally the required joint torques $\boldsymbol{\tau}$ and contact forces $\boldsymbol{W}_i$ given by

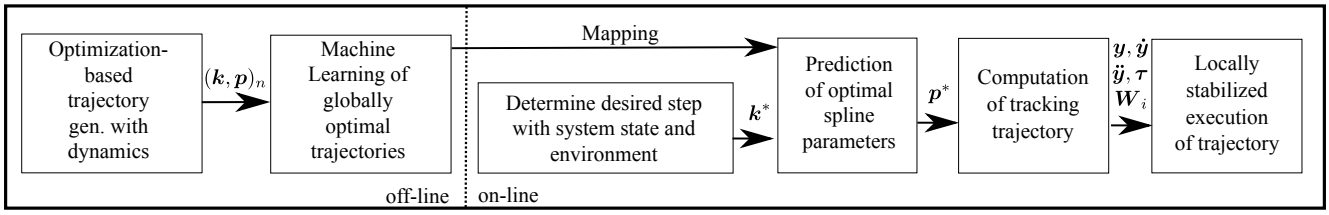

Fig. 2. Pipeline used to generate on-line walking based on a set of globally optimal trajectories $(\boldsymbol{k}, \boldsymbol{p})_n$. Given the task $\boldsymbol{k}^*$ the required spline parameters $\boldsymbol{p}^*$ are computed.

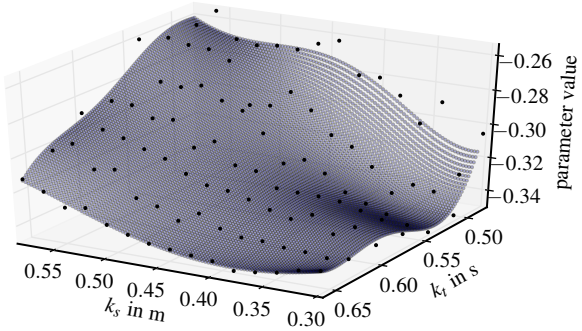

Fig. 3. Non-linearity of a parameter p_{38} over $\boldsymbol{k}$ for the torque cost function. This parameter does not exhibit the regions found with this cost function in other parameters. Sparse black markers: samples generated by the optimization. Dense mesh: predicted values using Gaussian process regression.

(4) are implemented as inequality constraints, which are highly non-linear in $\boldsymbol{p}$. A simplification was made for this data set: the redundancy of (4) in the contact forces $\boldsymbol{W}_i$ was resolved by taking the least-square solution. This can be replaced by a parameterization of the contact force distribution as already presented in [4].

The contact forces are constrained to fulfill the friction cone and Zero-Moment Point (ZMP) conditions [1]. These enforce that the contact is a full surface contact throughout the contact phase.

The specified optimization problem is further restricted to finding a solution in the subspace of $\boldsymbol{p}$ which defines cyclic gait motions. A motion is considered to be cyclic if it can be repeated for the following step when swapping the leg joints [1].

The computation of the constraints is possible through inverse dynamics for any given t, without having to solve the differential equation (4) in t. Building on this property, the optimization problem is approximated by satisfying the constraints and computing the cost function only at a discrete, set number of points within the time span of the trajectory.

The optimization problem is solved by using an interior point method implemented in [5].

V. Data Set Generation from Optimization

Once the optimization problem is implemented, it is possible to generate a set $\boldsymbol{p}_{\text{TRAIN}}$ for a given $\boldsymbol{k}_{\text{TRAIN}}$. The key to a successful generalization is that the data set is sufficiently smooth. As $\boldsymbol{p}$ is computed by an iterative numeric scheme and the cost functions (5) are potentially not globally convex, the results provided by the optimization must be treated with care.

The adopted method to deal with the existence of global minima is to execute a global search based on random initial parameters. To ensure (near to) global optimality, the sample with the best cost function is selected, potentially creating non-contiguous regions in the mapping $\boldsymbol{k} \rightarrow \boldsymbol{p}$.

Additionally, the smoothness can be destroyed by noise which comes from the termination criterion in the optimization algorithm. When looking at the mapping $\boldsymbol{k} \rightarrow \Gamma$ noise is mostly not present. However, in the parameter space there might be considerable noise because

$$\frac{\partial \Gamma}{\partial p_i} << \frac{\partial \Gamma}{\partial p_j} \text{ for } i \neq j \tag{7}$$

can differ significantly. Even though all solutions found might be feasible with respect to the constraints, this noise can seriously deteriorate some machine learning algorithms so that the predicted $\boldsymbol{p}^*$ violates the constraints. The chosen solution is to re-optimize the results of the one solver execution and maximize the difference between initial cost value and solver tolerance.

The cost function $\Gamma_\tau(\boldsymbol{p})$ is not only non-linear in $\boldsymbol{p}$, it also has very different sensitivity (7) for different parameters. This is caused by the quadratic function and by the different inertias of some segments of the robot in different contact configurations. An example of the non-linearity in the data set is displayed in Fig. 3.

The parameter k_s was varied in the range $[0; 1.2]$ m, k_t was varied in $[0.5; 1.08]$ s. This allows the generation of any cyclic gait with these ranges. For the robot with $N_{\text{JOINTS}} = 6$, N_{SPLINE} was chosen to be 192. The restricted optimization space $N_{\text{CYCLIC}} = 132$ was used for the optimization.

VI. Applied Machine Learning Methods

In this Section the applied machine learning methods are introduced. For the given data set, the properties of each algorithm were studied and the most suitable configuration for the algorithm was determined.

n-Nearest Neighbors (n-NN): This method [6], [7] interpolates locally between the closest n samples. Straightforward uniform weights and distance weights for the neighbors were applied. With $n = 1$ this method degenerates to a pure look-up table. We applied n-NN

up $n = 7$, since a higher number of neighbors added no significant benefit.

Regression Tree (RT): An additional simple and fast regression method is the RT [6]. Here the task-space is partitioned into m regions and the average of the samples in each region is computed. We evaluated RT with multiple but fixed maximum depth in the range $d = [3, 7]$, while we also applied this method to estimate an optimal tree depth with respect to the data (no fixed maximum depth). The minimum leaf size was one sample, while our condition for minimum split was two samples.

Polynomial Regression (PR): PR [6] models the mapping through a n-th degree polynomial plane. We used $n = [1; 7]$-order polynomials, while applying l_2 regularization.

Support Vector Regression (SVR): One of the more sophisticated methods applied was the support vector machine, since it can also be applied to regression problems. The two variants ϵ-SVR [8] and ν-SVR [9], [10] were evaluated. The free model parameters ϵ and ν for the respective algorithms, and C as the regularization parameter in both approaches, were experimentally chosen. ϵ controls the size of the ϵ-tube, within which errors are not penalized. With ν in ν-SVR, the number of support vectors and the number of training errors can be adjusted. Additionally, we compared the performance of a polynomial kernel and a Gaussian radial basis function (RBF) kernel.

Gaussian Process Regression (GPR): Another non-parametric method is GPR [11], [12]. This method relies on classical probability theory, where not only the most probable value but also the uncertainty of this value is computed. A Gaussian Process does not model the function $f(\boldsymbol{k})$ directly, but rather considers the correlation between the input $\boldsymbol{k}$ and learned $\boldsymbol{k}$ points. We applied kernels using squared exponential, as well as absolute exponential, cubic and linear correlation to exploit the distribution of our data. The parameter nugget (regularization) was found through exploration.

VII. Results

This section follows Fig. 2 and evaluates the computed $\boldsymbol{p}^*$ in different ways. First we present the best determined parametrization of each machine learning method introduced in Section VI. All implementations of the applied algorithms were taken from the library scikit-learn [13]. Second, we compare the machine learning methods, parameterized as described above, regarding the accuracy of the predicted trajectory parameters $\boldsymbol{p}^*$, and prediction runtime. After that, we analyse $\boldsymbol{p}^*$ using the cost function and constraints specified in the optimization problem. Finally the trajectories are applied to a simulated robot.

A. Parametrization of Machine Learning Methods

In order to find the best parameters for each machine learning method we conducted a grid search on the parameter space with the goal of maintaining generalization and low model error. The best parameters for each method are summarized in Table I.

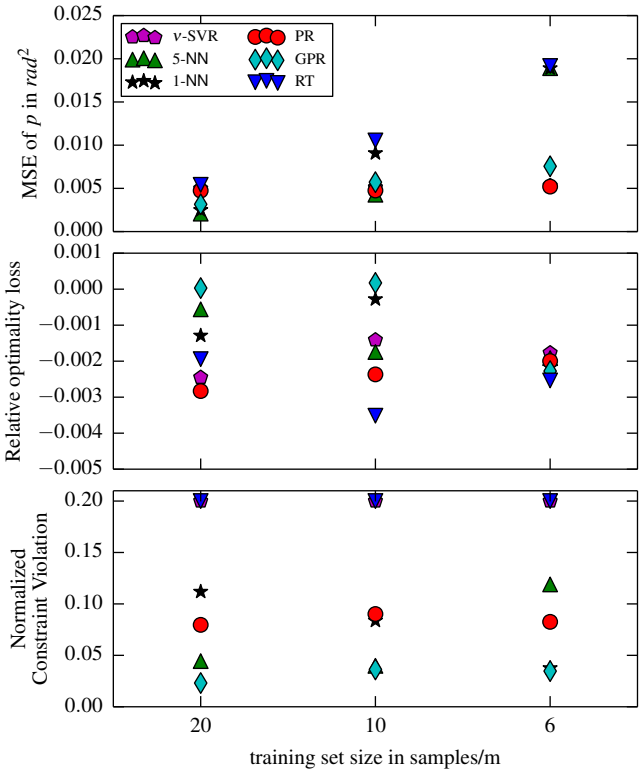

Fig. 4. Mean squared error in the parameter space for different machine learning methods and different sampling densities in the stride-length (k_l) direction of the task-space $\boldsymbol{k}$ for $\Gamma_{\dot{q}}$.

TABLE I
Best Parametrization of each Method

ML-Methods	Parameters
n-NN	neighbors $n = 5$ weight distance function
RT	maxium depth $d = 5$
PR	$5th$ order polynomial
ν-SVR	Gaussian kernel $\nu = 4.73 \cdot 10^{-1}$ $C = 1 \cdot 10^3$
GPR	absolute exponential correlation nugget $= 3 \cdot 10^-2$

B. Results for applied Machine Learning Methods

The quality of the mapping $\boldsymbol{k} \rightarrow \boldsymbol{p}$ provided by the different machine learning methods described in Section VI was evaluated with a 11-fold cross validation. The validation was done in terms of mean-square error of $\boldsymbol{p}^*$, predicted with the machine learning methods against $\boldsymbol{p}$, provided by the optimization which was not part of the training set. The test set was filtered to not contain points which fall in a zone along the edges of the task-space area. This zone was set to 0.3 of the allowed interval of the respective task-space direction.

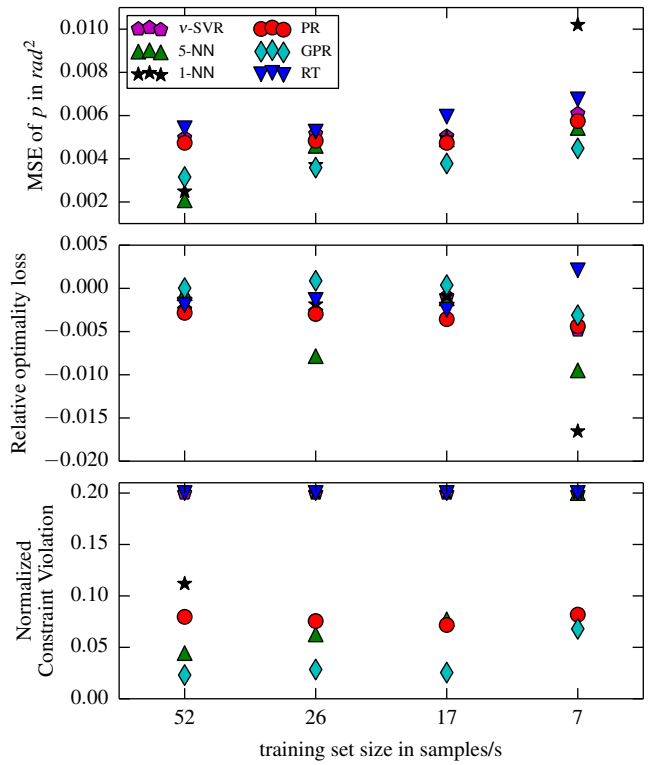

Fig. 5. Mean squared error in the parameter space $\boldsymbol{p}$ for different machine learning methods and different sampling densities in the step time (k_t) direction of the task-space $\boldsymbol{k}$ for $\Gamma_{\dot{q}}$.

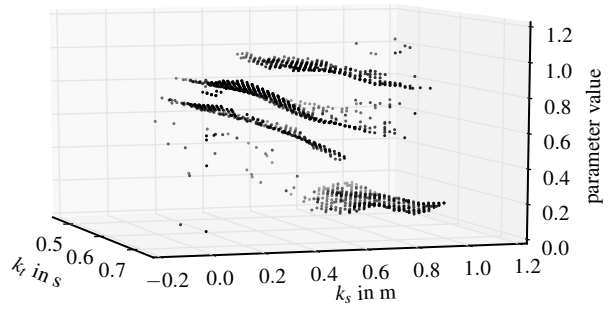

Fig. 6. One parameter in $\boldsymbol{p}$ illustrating the structure of the data set for Γ_τ. The different local minima are clearly distinguishable surfaces.

Computation time being a concern for scaling this approach to 3 dimensional walking and a larger task-space, the sample density was varied in the task space dimensions. This allowed us to find a sample density per dimension with acceptable optimality loss and constraint violation combined with the minimal computation time. Fig. 4 shows the reduction of sampling density in the direction of k_s, Fig. 5 for the direction k_t.

Results for the cost function Γ_τ where far worse initially. After a closer look into that data set it was revealed that this was clearly due to the existence of local minima. This means the global search provided multiple $\boldsymbol{p}$ for a given $\boldsymbol{k}$ associated with different costs Γ. In order to use always the best solution we separated the data set into clusters using Gaussian Mixture Models and removed all but the best cluster. Completely automating this process was very hard as the clusters have a branch like structure and can also not be separated using the cost value as additional parameter. The structure of the results is shown in Fig. 6. After separation of the data, the algorithms perform equally well as for $\Gamma_{\dot{q}}$.

TABLE II
Parameter Prediction Runtime

ML-Methods	**Samples in training set** 220	110	55	28
RT	2.3e-6	2.3e-6	2.3e-6	2.5e-6
PR	1.1e-5	1.1e-5	1.1e-5	1.0e-5
GPR	6.2e-5	4.1e-5	3.4e-5	2.6e-5
n-NN	2.1e-4	2.2e-4	2.1e-4	2.2e-4
ν-SVR	2.5e-3	1.6e-3	1.3e-3	1.0e-3

Prediction Runtime: Execution time of the prediction was also evaluated. The results for the different combinations of methods and size of the data set are presented in Table II.

C. Cost Value and Constraint Violation

The learned mapping $\boldsymbol{k} \rightarrow \boldsymbol{p}$ was also evaluated using the cost and constraint functions of the optimization problem that was used to generate the data set. For this evaluation, the same test samples and same subsampling already described in Sec. VII-B were used. Fig. 4 and Fig. 5 show the loss in optimality and constraint violations for the different methods. Before taking the max-norm, the constraint violations for inequality constraints with upper and lower bound were normalized using the width of the allowed interval. The remaining constraints were not normalized.

Constructing a trajectory which is feasible is considered critical for successful execution. The following constraints are the ones most active in the optimization problem and thus shape the trajectory, these are also the ones most likely to be violated by a prediction error in $\boldsymbol{p}^*$ because they are already at their bounds:

- Equality constraints on $\boldsymbol{H}_\Delta$ during the double-support phase
- Zero-Moment point and friction cone constraints
- Joint velocity and joint torque constraints

D. Simulation

In the process of generating the data set $(\boldsymbol{k}_{\text{TRAIN}}, \boldsymbol{p}_{\text{TRAIN}})$ the real system dynamics were only guaranteed not to be satisfied at a discrete set of points within the time span of the trajectory. We therefore used a standard rigid-body simulation environment [14] to evaluate the computed trajectories. The environment solves the equations of motion (4) for t together with the contact dynamics at a rate of 1 ms. The robot motions are controlled by $\boldsymbol{\tau}$.

For a given trajectory represented by $\boldsymbol{p}$ the full system state $\boldsymbol{y}, \dot{\boldsymbol{y}}$ and the required accelerations $\ddot{\boldsymbol{y}}$ are computed

using the B-spline function and the fixed-base assumption, as employed in the optimization process. Using (4), the required joint torques $\boldsymbol{\tau}$ and contact forces $\boldsymbol{W}_i$ are computed in exactly the same way as in the optimization problem.

For evaluation of a single pattern, the simulation environment is initialized with the start state $\boldsymbol{y}(0), \dot{\boldsymbol{y}}(0)$ of the pattern. The following computed $\boldsymbol{\tau}$ is then passed to the simulator:

$$\boldsymbol{\tau} = \boldsymbol{\tau}^* + \mathrm{PD}(\boldsymbol{q}^*, \tilde{\boldsymbol{q}}, \dot{\boldsymbol{q}}^*, \tilde{\dot{\boldsymbol{q}}}) + \boldsymbol{\tau}_{\mathrm{balance}} \tag{8}$$

where $\boldsymbol{\tau}^*$ are the torques computed by inverse dynamics using (4) with the system state $\boldsymbol{y}^*$ defined through the spline function parameterized by $\boldsymbol{p}^*$. Additionally, a joint impedance controller is applied to improve tracking $\boldsymbol{q}^*$ and $\dot{\boldsymbol{q}}^*$ with the current system state $\tilde{\boldsymbol{q}}$ and $\tilde{\dot{\boldsymbol{q}}}$ respectively. To stabilize the walking motion, $\boldsymbol{\tau}_{\mathrm{balance}}$, similarly to [15] albeit without the constrained optimization problem, is introduced:

$$\begin{aligned} \boldsymbol{\tau}_{\mathrm{balance}} = \;& c_R \boldsymbol{J}_R^T \mathrm{Adj}^T(H_{\mathrm{RB}}) \boldsymbol{W}_B \\ & + c_L \boldsymbol{J}_L^T \mathrm{Adj}^T(H_{\mathrm{LB}}) \boldsymbol{W}_B \\ \boldsymbol{W}_B = \;& \mathrm{PD}(x_{\mathrm{B}}^*, \tilde{x}_B, \dot{x}_{\mathrm{B}}^*, \tilde{\dot{x}}_B) \end{aligned} \tag{9}$$

with x_{B}^* and $\dot{x}_{\mathrm{B}}^*$ the position of the robot computed with $\boldsymbol{p}^*$. The scalar weights c_R and c_L are zero if the contact is not active and otherwise distribute a unit gain on the active contacts. A Cartesian impedance controller computes the Wrench $\boldsymbol{W}_B$ acting on the central base body of the robot and thus ensures that this body follows the desired trajectory. As no forces can be directly produced on the body, but only through the feet in contact with the floor, the wrench is projected to the foot frames using the adjoints $Adj(H_R B)$ and $Adj(H_L B)$ respectively. Theses forces are then realized by computing the required torques in the joint space using the body Jacobians J_R and J_L respectively.

All parameters $\boldsymbol{p}$ from the optimization or from the machine learning can be evaluated this way. As the contact stability is paramount to a successful execution of a trajectory, the ZMP can be compared for ideal and learned trajectories. This is displayed for an example case in Fig. 7.

In the application, the desired foot step locations defining k are known one step ahead, which allows one cycle to predict $\boldsymbol{p}$, possibly for multiple values of k.

E. Transition steps

Varying distance between the foot steps requires a non-cyclic gait pattern, which can be created by combining two trajectories. Given the phasing variable α and the task/spline parameter tuples $(\boldsymbol{k}_1, \boldsymbol{p}_1)$ and $(\boldsymbol{k}_2, \boldsymbol{p}_2)$ a transition trajectory can be computed as:

$$\begin{aligned} \boldsymbol{y} = \; & b(\alpha) f_{\mathrm{SPLINE}}(\boldsymbol{p}_1, \alpha \cdot k_{t,1}) \quad + \\ & (1 - b(\alpha)) f_{\mathrm{SPLINE}}(\boldsymbol{p}_2, \alpha \cdot k_{t,2}) \\ b(\alpha) = \; & \begin{cases} b_{\mathrm{BC}}(\alpha / r_p) & \text{for } \alpha < r_p \\ 1 & \text{for } \alpha \geq r_p \end{cases} \end{aligned} \tag{10}$$

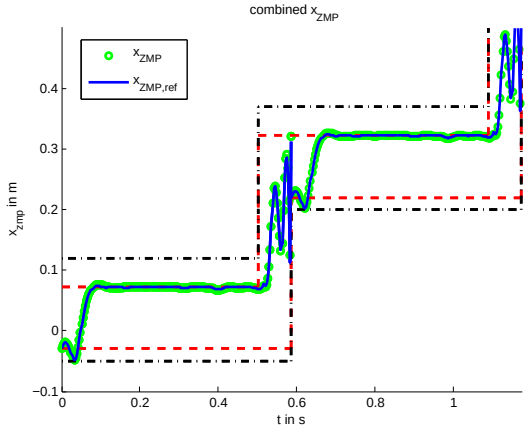

Fig. 7. $x_{\mathrm{ZMP,ref}}(t)$ and $x_{\mathrm{ZMP}}(t)$, resulting for the trajectory generated by optimization and x_{ZMP} computed for the predicted trajectory displayed for two consecutive steps. The conservative constraints imposed on the trajectory in the optimization problem are shown in dashed red, the system constraints are shown in dash-dotted black lines. The violation of the conservative constraints for both trajectories is due to the approximation of the optimization problem at a limited set of points along t.

with the polynomial, twice continuously differentiable blending function b_{BC} which ensures continuous transitions for $\boldsymbol{y} \ldots \ddot{\boldsymbol{y}}$, thus continuous $\boldsymbol{\tau}$.

Blending two trajectories with (10) results in a suboptimal total trajectory, but increases the application of the computed trajectories to general walking, more so than just executing one cyclic motion. This is very relevant as the feed-back controller described in the previous section does not have enough control authority to stabilize the system in case of significant disturbances. The only way to stabilize the system is to modify k for the following steps, commonly know as step adaptation.

The suboptimality of these blended trajectories can be remedied by providing optimized transitions steps, this however increases the task-space dimensionality.

VIII. Discussion

The machine learning algorithms presented in Section VI perform well on the smooth data set generated with $\Gamma_{\dot{\boldsymbol{q}}}$ where optimality loss is very small and constraint violations are reasonably small for GPR, as shown in Section VII.

The data set generated with the cost function $\Gamma_{\boldsymbol{\tau}}$ with local minima, required further processing, after which the different methods performed similarly as in the velocity cost function case.

For both data sets the critical result is the maximum relative constraint violation, as this determines how conservative the constraints in the optimization algorithm must be configured to achieve consistently feasible trajectories. As these limits already have to be reduced to account for head-room for the feed-back control, to ensure that the actuators are not saturated by small disturbances. The 5% violation of the constraints seems acceptable, as past experience shows that the head-room for the controller must to be more the 10%, or more, of

the two-sided inequality constraints interval, depending on the quality of the rigid-body model used.

Our approach is to generalize trajectories which are generated off-line, using nonlinear optimization. This takes the dynamic properties of the system, with respect to both feasibility and optimality, into account. In contrast to this, the trajectory generation approaches which operate on-line are mostly restricted to kinematic planning and higher-level heuristics to find a feasible trajectory in the available time. Solving the problem of trajectory generation off-line does not have this problem. However, on-line approaches are always general when it comes to varying starting conditions of the desired trajectory. We show that the off-line generated trajectories can also be applied in presence of disturbances, of course without optimality.

IX. Conclusion

The presented approach of generalizing optimal trajectories is feasible, although the quality of the results strongly depends on the machine learning method the user employs, as well as the sampling densities used. For any of the learning methods discussed in the paper, it is unlikely that suboptimal trajectories will result from our new approach. Instead, the limiting factor for the threshold of the sampling density is the maximum constraint violation.

Future work will be focussed on extending the feedback aspects with a step adaptation based on the system state and evaluating the performance in the direction of robustness. Additionally, the aim is to extend the formulation to the three-dimensional case.

The essential trade-off is between sampling density and conservativeness of the inequality constraints. For the equality constraints during the double-support phase the criterion is the allowed error of the relative foot positions.

The representation of the optimal results has the advantages of being compact, extensible to high task-space dimensions and quick enough for use in conjunction with a step planner that obtains feasible steps on-line from sensor data.

References

[1] A. Werner, R. Lampariello, and C. Ott, "Optimization-based generation and experimental validation of optimal walking trajectories for biped robots," in *International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 4373–4379.

[2] R. Lampariello, D. Nguyen-Tuong, C. Castellini, G. Hirzinger, and J. Peters, "Trajectory planning for optimal robot catching in real-time," in *International Conference on Robotics and Automation (ICRA)*, 2011, pp. 3719–3726.

[3] K. Hu and D. Lee, "Bipedal locomotion primitive learning, control and prediction from human data," in *Proc. the 10th International IFAC Symposium on Robot Control (SYROCO)*, 2012, pp. 536–542.

[4] A. Werner, R. Lampariello, and C. Ott, "Trajectory optimization for walking robots with series elastic actuators," in *53rd Conference on Decision and Control (CDC)*, 2014, pp. 2964–2970.

[5] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, pp. 25–57, 2006.

[6] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.

[7] T. Cover and P. Hart, "Nearest neighbor pattern classification," *Information Theory, IEEE Transactions on*, vol. 13, no. 1, pp. 21–27, 1967.

[8] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and computing*, vol. 14, no. 3, pp. 199–222, 2004.

[9] B. Schölkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, "New support vector algorithms," *Neural computation*, vol. 12, no. 5, pp. 1207–1245, 2000.

[10] C.-C. Chang and C.-J. Lin, "Training nu-support vector regression: theory and algorithms," *Neural Computation*, vol. 14, no. 8, pp. 1959–1978, 2002.

[11] C. Rasmussen and C. Williams, "Gaussian processes for machine learning," *Gaussian Processes for Machine Learning*, 2006.

[12] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *The Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] F. Kanehiro, H. Hirukawa, and S. Kajita, "Openhrp: Open architecture humanoid robotics platform," *The International Journal of Robotics Research*, vol. 23, pp. 155–165, 2004.

[15] C. Ott, M. A. Roa, and G. Hirzinger, "Posture and balance control for biped robots based on contact force optimization," in *International Conference on Humanoid Robots (Humanoids)*, 2011, pp. 26–33.