Dynamic Optimality in Real-Time:
A Learning Framework for Near-Optimal Robot Motions

Roman Weitschat, Sami Haddadin, Felix Huber and Alin Albu-Schäffer

Abstract—Elastic robots have a distinct feature that makes them especially interesting to optimal control: their ability to mechanically store and release potential energy. However, solving any kind of optimal control problem for such highly nonlinear dynamics is feasible only numerically, i.e. offline. In turn, optimal solutions would only contribute a clear benefit for dynamic environments/tasks (apart from rather general insights), if they would be accessible/generalizable in real-time. In this paper, we propose a framework for executing near-optimal motions for elastic arms in real-time. We approach the problem as follows. First, we define a set of prototypical optimal control problems. These represent a reasonable set of motions that an intrinsically elastic robot arm is sought to execute. Exemplary, we solve the optimal control problem for some of these prototypes in a roughly covered task space. Then, we encode the resulting optimal trajectories in a dynamical system via Dynamic Movement Primitives (DMPs). Finally, a distance and cost function based metric forms the basis to generalize from the learned parameterizations to a new unsolved optimal control problem in real-time. In short, we intend to overcome the well known problems of optimal control and learning with associated generalization: being offline and being suboptimal, respectively.

Generating robot trajectories is one of the largest fields of robotics research. Initially, it started from purely geometric (mostly linear interpolation, circular interpolation or polynomial splines) approaches in the early days for calculating point-to-point or N-via-point motions without taking into consideration the environment or the robot dynamics. Since rigid industrial robots were supposed to execute purely geometric tasks, the low-level controller ensures accurate tracking behavior of the device, while trajectory generation remains on kinematic level. A recent very nice overview on these techniques can be found in [2]. In order to solve not only the free space problem but also the general motion planning problem, a vast range of methods for planning global motions in complex, however mostly static environments were developed. They ensure that the robot can be brought collision free from an initial configuration to a desired final configuration [3], [4], [5], [6], given a solution exists. Typically, such planning methods generate a statically valid path in a timeframe of seconds and are usually suboptimal (in particular in the dynamic sense). However, dynamic optimality e.g. in terms of minimal joint velocity or energy consumption is certainly a desirable property, which can be expressed on kinematic or dynamic level, respectively. In fact, for intrinsically elastic systems it offers entirely new ways to control and will certainly affect even global motion planning at some point. In industrial robotics, time optimal motions are of primary concern. For generating kinematically optimal trajectories in real-time, several efficient algorithms became available [7]. However, as all aforementioned schemes work on kinematic level only (some also up to kinodynamics), they can never fully exploit the inherent dynamic capabilities of a robot. Strong insights in this respect were e.g. obtained in [8], [9] by solving the time optimal tracking problem while taking into account the rigid body dynamics and the constrained motor torques. These results allow to execute dynamically optimal tracking in real-time for non-redundant rigid manipulators. Further work on solving optimal control (OC) problems for manipulators, however purely offline, can e.g. be found in [10], [11]. Discussions and results on full body motions are given in [12], where the problem of optimal running is solved with direct methods. Particularly challenging and interesting becomes the achievement of dynamic optimality if being faced with intrinsically elastic robots such the DLR Hand-Arm System (Hasy) [1], see Fig. 1. This is due to the fact that these systems are able to store and release potential elastic energy, which is especially helpful for carrying out explosive motions such as e.g. throwing. The effect was thoroughly analyzed for 1DoF variable stiffness systems (VSA) in [13], [14], [15], where analytic solutions could still be derived for some cases. For multiple degrees of freedom, however, only numerical solutions can be found due to the strongly nonlinear and coupled behavior [16]. Nonetheless, as elastic systems are designed for acting in dynamic environments at close proximity to humans, it is clear that pure offline solutions are neither applicable nor sufficient.

An entirely different approach to generate task trajectories originates from trajectory learning and generalization, where considerable effort was put into the development of the learning-by-demonstration (LbD) paradigm. Typically, a desired motion is taught to an actively or passively backdrivable robot [17], [18] kinesthetically [19], [20]. Alternatively, human motion tracking is used for generating trajectories, which are transferred to the robot and
can then also be refined via kinesthetic teaching [21] or other iterative learning/optimization techniques. In order to generalize the demonstrated trajectories, one has to find a suitable representation that builds on the acquired data. A particularly interesting one is a dynamical forced system in the form of a Dynamic Movement Primitive (DMP) [22]. As the trajectories are generated by means of a second, or higher order differential equation, it is relatively simple to incorporate disturbance forces for collision avoidance or retraction [23], [24]. LbD is a very flexible framework, which can be applied to various types of problems [25], [26]. Nonetheless, there are also natural limitations with existing schemes, as the generated motions are certainly suboptimal (both kinematically and dynamically).

This brings us to the main idea of our paper. We intend to overcome the limitations of each approach by combining them into a single Optimal Motion Framework (OMF) that utilizes the advantages of both:

- Implicit representation of the robot dynamics in optimal control solutions
- Complexity reduction, generalization, and real-time capability of learning dynamical systems

The paper is organized as follows. Section II introduces our approach. Sec. III describes the concept of prototypical optimal control problems. In the following, Sec. IV provides an overview on how we embed optimal trajectories into DMPs and then generalize with the help of a distance and/or cost function based metric. Section V describes various simulations and experiments that underline the validity and "near-optimality" of our approach. Section VI concludes the paper.

II. APPROACH

As one is generally not interested in the solution of any possible OC problem, we first define a reasonable set of representative optimal control problems that can be roughly divided into reaching type motions and tracking type motions\(^1\). The former represent a rather abstract task, which concrete joint space trajectory for optimally solving the task (also in operational space) is to be found. The latter problem is specified by a full (joint or operational space) trajectory that shall be optimally tracked, i.e. we need to find the joint space motion that ensures both, minimal tracking error and possibly other costs, such as energy related ones. However, for both motion types we encode a set of optimal joint space trajectories for each task into a modified version of the original DMP formulation, whereas for the latter case a two-staged DMP approach is utilized, one for generalizing reference motions (in joint or task space) and one for generalizing the optimal solutions for tracking this operational space movement. Please note that for sake of clarity, we focus on reaching type motions in this paper. We show that the embedding of optimal motions that inherently capture the robot dynamics leads to the ability to generate near-optimal motions in real-time for each of the considered problems. Figure 2 depicts the overall framework target structure in comparison to the LbD approach. Specifically, we

1) use dynamically optimal trajectories coming from solving complex nonlinear optimal control problems as learning input,
2) encode them into an optimized dynamical system,
3) and use a metric, based on the cost function and/or geometric distance, for selecting a near-optimal parameter set in real-time for the generalizing step.

A somewhat related approach to ours can be found in [27], where kinematically optimal trajectories were used as template trajectories that are subject to situation related costs for collision avoidance. Together with a situation descriptor these are subsequently generalized via another offline stage, which, however, does not optimize from scratch and is therefore considerably faster than the generation of the original training data. First work on using energy optimal solutions for catching tasks and applying different learning approaches for generalization can be found in [28]. There, optimal trajectories are encoded in B-splines or trapezoidal functions, which are then generalized with different state-of-the-art machine learning techniques as e.g. Nearest-Neighbor (NN) or Support Vector Machines (SVM).

Our approach discriminates from these existing techniques in the sense that we also consider elastic systems (several additional constraints and nonlinearities have to be considered). Furthermore, we combine optimal control (which offers, in contrast to pure nonlinear optimization, the tools to give stronger hints for global optimality of a solution) with generalization algorithms by means of dynamical systems (DMPs), and also use the cost function of the according optimal control problem as an underlying generalization metric. Furthermore, instead of treating only a single motion control problem, we make an attempt to give a rather general methodology for a set of control problems that cover most typical robot motion tasks, which, however, can be easily extended if needed.

In the following section, we shortly introduce some basics on optimal control for VSA\(^2\) and then describe our selection of prototypical optimal control problems.

---

\(^1\)Please note that we do not claim completeness. It is straight forward to extend the approach to new problems as well. In this paper we focus on the methodology.

\(^2\)VSA stands for Virtual Spring Damper.
III. OPTIMAL CONTROL

A. Optimal control for elastic joints

Let us consider the class of systems that is described by a set of first order differential equations \( \dot{\vartheta}(t) = f(\vartheta(t), u(t)) \), where \( \vartheta \) denotes the state vector and \( u \) the control input, respectively. The initial state is denoted by \( \vartheta(0) = \vartheta_0 \), the final constraints are \( \varphi(\vartheta(f), f_f) = 0 \), and the set of path constraints is \( c(\vartheta(t), u(t), t) \leq 0 \). Solving an optimal control problems aims at finding the control input \( u^* \) that minimizes a given cost function

\[
\min_{u(t)} J = h(\vartheta(f), f_f) + \int_{t_0}^{t_f} g(\vartheta(t), u(t), t)dt,
\]

with \( h(\vartheta(t), f_f) \) being the terminal and \( \int_{t_0}^{t_f} g(\vartheta(t), u(t), t)dt \) the running cost. This cost function basically denotes the task to be accomplished and may take various forms. The dynamics of a full VSA arm can be described by

\[
M(q)\ddot{q} + C(q, \dot{q})\dot{q} + g(q) = \tau_f \
B\dot{\Theta} + \tau_f = \tau_m \
\tau_f = \tau_f(\Theta, q, \sigma),
\]

where \( \Theta \) and \( q \) are the motor and link side position, respectively. The position of the stiffness actuator is denoted \( \sigma \), which is treated as a constant parameter in this paper. \( M(q) \), \( C(q, \dot{q}) \), \( g(q) \), and \( B \) are the link side inertia matrix, the centrifugal and Coriolis vector, the gravity torque, and the motor inertia. \( \tau_m \) denotes the torque acting through the positioning motor\(^3\). The vector \( \tau_f \) represents the elastic joint torque, which is a nonlinear function of the elastic deflection \( \varphi = \Theta - q \) and \( \sigma \). Please note that we assume that for a given \( \sigma \) it is possible to solve \( 4 \) for \( \Theta \). The optimal trajectories in this paper are obtained with the nonlinear optimal control solver GPOPS\(^2\) available in MATLAB. It uses the hp-adaptive Legendre-Gauss-Radau quadrature integral pseudospectral method for solving general nonlinear optimal control problems.

The considered concrete nonlinear dynamics we intend to solve is a 2 DoF dynamic HASy model, see Figure 3. It serves as a benchmark problem for an intrinsically elastic robot with nonlinear torque deflection characteristics. Please note that in this work we do not make use of the possibility to change this relation via adjusting \( \sigma \), i.e. we assume the robot to be intrinsically elastic but the stiffness adjuster remains constant. Subsequently, we discuss several problems, which we consider to be important for elastic systems.

B. Prototypical optimal control problems

By nature, the motion generation problem is infinitely large and generally poorly defined in the sense of what a desired motion should exactly look like. Therefore, it is rather hard to find a general optimal control problem that inherently contains all possible instantiations and thus captures the essence of motion. Therefore, we pragmatically resolve this dilemma by introducing prototypical optimal control problems. These are sought to find an optimal control input \( u^* \) that generates a distinct type of motion for a given robotic system and potentially a secondary system the robot is associated to (e.g. an object to be manipulated with another target dynamics). The following classification aims at grouping motion behaviors according to their “higher-level” target. We coarsely distinguish between following main problems:

1) reaching type motion
2) tracking type motion

In the following classification of prototypical OC problems (see Fig. 4), \( N(\vartheta) \) denotes the neighborhood of the system state \( \vartheta \) according to a suitable metric (typically spatially). \( \vartheta_0, \vartheta_f, \vartheta_f \) denote initial state and velocity, and final state and velocity.

- Reaching type motions
  1) Reaching motion (grasping): \( \vartheta_0, \dot{\vartheta}_0 = 0 \rightarrow \vartheta_f, \dot{\vartheta}_f = 0 \), considering minimum final error and minimum energy
  2) Explosive motion (throwing motion, catching) \( \vartheta_0, \dot{\vartheta}_0 = 0 \rightarrow N(\vartheta_f), \|\dot{\vartheta}_{max}\| \), considering minimum final error and minimum energy
  3) Explosive target motion (throwing motion, boxing, hitting) (dynamic grasping) \( \vartheta_0, \dot{\vartheta}_0 = 0 \rightarrow N(\vartheta_f), \dot{\vartheta}_f \), considering minimum final error
  4) Implicit target motion (object manipulation, ballistic throw e.g. bucket throw) \( \vartheta_0, \dot{\vartheta}_0 = 0 \rightarrow N(\vartheta_f) \) while, considering minimum final error and minimum energy
  5) Impulsive motion (braking motion, stopping catching) \( \vartheta_0, \dot{\vartheta}_0 \rightarrow N(\vartheta_f), \dot{\vartheta}_f = 0 \), considering minimum final error and minimum energy

Please note that we intentionally excluded time optimal problems, as we leave them for future work.

- Tracking type motions (not necessarily optimal control problems\(^4\))

2Please note that we introduce the problem formulation for VSA in general and not for the elastic case only. The subsequent learning and generalization, however, is then done for the case without explicit stiffness control input.

3We neglect the dynamics of the usually significantly smaller and therefore faster stiffness adjustment motor.

4Depending on the available degrees of freedom, there might be only a single solution to the problem and no solution space to be explored.
1. Tracking (gestures, constrained motion primitives): $(\dot{\vartheta}_0, \dot{\vartheta}_0) \rightarrow \vartheta(t) \rightarrow (\vartheta_f, \vartheta_f)$.

2. Cyclic tracking (stirring, cranking, shaking): $(\dot{\vartheta}_0, \dot{\vartheta}_0) \rightarrow \vartheta(t)$ with $\dot{\vartheta}(t) = \dot{\vartheta}(t-P)$, $T = \text{const.}$, and $p \in \mathbb{N}^+$. 

Next, we discuss the embedding of the optimal trajectories into the DMP formulation and how the cost function might be exploited for generalization.

IV. OPTIMAL MOTION FRAMEWORK

**Algorithm 1: DMP generation**

```plaintext
input : motion type, c(q), parameters \{\xi_k\}
output: w^*, \Phi^*

for \( k \leftarrow 1 \) to \( m \) do
  \( [q_k^*, \dot{q}_k^*, \ddot{q}_k^*] = \min_u J(\text{motion type, c(q), } \xi_k) ; \)
  for \( i \leftarrow 1 \) to \( n \) do
    \( f_i(t_i) = -\tau_i q_i(t_i) + \kappa(t_i)(q_i(t) - \dot{q}_i(t)) - D\ddot{q}_i(t_i) ; \)
    \( x_i = \text{exp}\left\{ -\frac{\alpha}{\kappa} t_i \right\} ; \)
    \( \xi_i = [\xi_i; \cdots; \xi_i]_{\dim = M \times 1} ; \)
    \( F_k^* = [F_k^*; F_k^{\tau}(t_i)] ; \)
    \( X = [X; x_i^*] ; \)
  end
  \[ [w_k^*, \Phi^*] = \min \Gamma[\Phi^j, F_k^*, X] \rightarrow w^j \rightarrow f^j_{\ddot{q}} \]
end
```

It is parametrized by a set of optimal weights \( w_k^* \in \mathbb{R}^N \) for each dimension \( l \in M \) and an optimal parameter vector for the Gaussian kernels. The Gaussian basis is defined as

\[
\psi_l(x) = e^{-h_i(x-c_i)^2}
\]

with \( f_l^* := F^*(1: n, l) \), which needs to be set up for every \( l \) and \( k \) (\( k \) is omitted for brevity). It can be solved with efficient existing approaches [30]. Despite this being already a reasonably well working approach, this system choice leaves some issues unresolved. In particular, if choosing \( \kappa \) to be constant, as is typically done, the initial force \( \kappa(q_0 - g) \) produces a very large acceleration force \( f^*(t = 0) \neq 0 \), which is clearly not capturing the essence of a point-to-point trajectory that is initially at rest. This can be solved by introducing a time-varying continuously increasing, however, bounded stiffness \( \kappa(t) \). A possible choice that preserves stability and convergence is

\[
\kappa(t) = g_k \left( \frac{1}{1 + e^{-g_k k_0 t (\frac{\xi_k}{k_0} - 1)}} \right),
\]

where \( g_k \) is the final value of the function, \( k_0 \) is the initial value (\( k_0 > 0 \)), and \( k_0 \) is the slope. Of course, the particular implementation may be chosen differently.
A further issue when using an original DMP design is the dependency of number of weights and accuracy-reproduction of the original input data. To minimize computation time and disk space, we want to use only few Gaussian basis functions. As described in [30], this can be solved by parameterizing the Gaussian widths as $h_i = \frac{\beta_i}{\epsilon_i + \gamma_i}$, and finding the optimal parameter vector $\Phi = [\alpha_x \beta_x \gamma_x]$ by suitable minimization. For this, the error cost metric $\Gamma = \frac{1}{T} \sum_{i=1}^{T} ||f_i - f_{\text{approx},i}||$ is to be minimized (an alternative choice would be $\Gamma = \frac{1}{T} \sum_{i=1}^{T} ||y_i - y_{\text{approx},i}||$), see the last step of Algorithm 1\footnote{For brevity, the index $j$ indicates the required minimization loop.}. This problem can be solved with standard SQP solvers. This optimization is carried out for increasing $N$ and finally, the minimal $N$ that produces a negligible force error only is selected. After having obtained the optimal parameter set $[w^*, \Phi^*]$, one may then generalize the motion along $t \in [0 \cdots t_f]$ for different goals $g$ by solving

$$q(t) = \frac{1}{\tau_d} \int_0^{t_f} f_\text{approx}(x) + \kappa(t)(g - q) - D\tau q \, dt + q(0).$$

(7)

From (7) we obtain the link position, velocity, and acceleration which can then be inserted into (2). The motor trajectories $\Theta(t)$ are obtained by solving (2) for $\tau_d$, and then use (4).

(7) can be implemented in real-time, leading to a scheme that is capable of producing near-optimal solutions for different start and end configurations based on a finite set of optimal trajectories. We decided to use DMPs for generating optimal real-time trajectories because we want to use force generated trajectories to have the ability to affect the desired trajectory online, for example collision avoidance or constraint adherence (this is, however, not part of this paper). Such requirements cannot be easily complied with a two staged optimization approach, where the optimal control problem is solved offline first and generalization is done via linear optimal control approaches for sufficient closeness to the offline solutions. Important to notice is that the tracking type motions (motion type 6.+7. from Fig. 4) can also be covered with the DMP approach (see [22] for the original formulation). Our extensions for matching the special needs of elastic robots, however, are out of the scope of the paper and therefore omitted for brevity.

Next we discuss our approach to generalization.

A. Generalization for DMP motions

Generalization of DMPs has e.g. been addressed in [31]. In order to make use of the fact that “close” trajectories presumably encode more relevant information for a new goal, distance based weighting is applied for the extrapolation step. For the reaching movement generalization, we may apply the method from [32]

$$w_i^j(y_g) = \frac{\sum_{\forall k: \sigma_k \leq \delta} w_k^j(y_k) \sigma_k^{-1}}{\sum_{\forall k: \delta \leq \sigma_k} \sigma_k^{-1}},$$

(8)

Equation (8) is a sum of kernel weights multiplied with the inverse of the geometric or energetic distances between the
previously learned and the new goal. $\sigma_k$ is defined as

$$\sigma_k = \begin{cases} ||y_g - y_k|| + \epsilon, & ||y_g - y_k|| + \epsilon < \delta \\ \epsilon, & ||y_g - y_k|| + \epsilon \geq \delta, \end{cases}$$

(9)

where $y_g$ denotes the new goal position or goal cost function value, respectively, $y_k$ represents the surrounding sample goal $k$, $\delta$ is a distance limitation and $\epsilon$ prevents division by zero. $\delta$ is a chosen metric for selecting close Gaussian bases of interest and is large enough that at least one Gaussian basis is used. The introduced metric in (9) is discussed in Sec. V in more detail.

After having introduced all necessary tools to solve some exemplary task, we now present simulations and experiments obtained with the OMF.

V. SIMULATION AND EXPERIMENTS

First, the generalization of energy-like optimal point-to-point motions with the HASy is shown and different generalization approaches compared to each other. Then, optimal throwing movements are investigated and experimentally verified on the HASy robot.

A. Learning and generalization of optimal reaching movements

The mostly used movement in robotics is a point-to-point task. Typically, such reaching motions are generated by common interpolation schemes. The results are, however, suboptimal (e.g. in the energetic sense). With the OMF, our solution to the problem is as follows. First, a grid of optimal solutions with minimal required motor energy is calculated over the entire workspace, see Fig. 5. The cost function to solve minimum input energy point-to-point motions is defined as

$$\min J = \int_{t_0}^{t_f} \left( \frac{1}{2} w_{\Theta_1} \dot{\Theta}_1^2 + \frac{1}{2} w_{\Theta_2} \dot{\Theta}_2^2 \right) dt,$$

(10)

with $w_{\Theta_1} = 1$, $w_{\Theta_2} = 1$ and $\dot{\Theta}_i$ being the motor velocity of joint $i = [1,4]$. Ideally, one would choose $\tau_m \dot{\Theta}$ to be the running cost. However, due to safety reasons the input to the system is practically $\dot{\Theta}$, i.e. $\tau_m$ is not controlled directly. The virtual input is therefore chosen to be $\dot{\Theta}_d$. The equality constraint $q(t_f) - q_d = 0$ is considered to guarantee reaching $q(t_f)$. The arising information regarding the cost function is stored in a database.

Using inverse kinematics, $q_d(t_f)$ and $q(t_f)$ are obtained. A few of the obtained trajectories from (10) are then encoded into DMPs, see Sec. IV. To represent $m = 4$ optimal input trajectories, $n = 40$ Gaussian bases are used. Further parameters are $g_k = 80$, $k_k = 20$, $k_0 = 2 \times 10^{-4}$, and $\alpha_x = 2.8$.

Figure 5 depicts the according results over the entire 2-DOF workspace (upper left plot). A more detailed resolution of the red square is depicted in the bottom left plot. The trajectories of moving to the 4 “corners” of the square are optimal solutions from (10). The accuracy of DMP trajectories is shown in the upper right plot as a comparison between OC and real-time generated DMP trajectories. Finally, the bottom right plot depicts a comparison between DMP generalized trajectories and the optimal ones.

<table>
<thead>
<tr>
<th>$J_{\text{err} %}$</th>
<th>1-NN J</th>
<th>1-NN Y</th>
<th>2-NN J</th>
<th>2-NN Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_{\text{max}} \text{ rad}^2 s^{-2}$</td>
<td>10.33</td>
<td>12.48</td>
<td>5.03</td>
<td>4.08</td>
</tr>
</tbody>
</table>

Table I: COMPARISON OF OC TO OPTIMAL DMP GENERALIZED WITH NEAREST NEIGHBORS (NN) COST J AND GEOMETRIC DISTANCE Y BASED

![Fig. 6. Comparison of cost functions calculated for OC, DMP and IPOL.](image)

To validate the performance of our approach, we give some comparison to other solutions. A straightforward one is the application of geometry based Nearest Neighbor (NN). On the other hand, it is possible to interpolate weights as explained in Sec. IV A. The according numerical comparisons to the offline generated ground truth OC trajectories are listed in Tab. I. As a meaningful error metric the relative error with respect to the ground truth cost is considered:

$$J_{\text{err}} = \frac{\sum_{i=1}^{N} J_{\text{DMP},i} - J_{\text{OC},i}}{\sum_{i=1}^{N} J_{\text{OC},i}},$$

(11)

$N$ is the number of optimized and simulated samples. $J_{\text{max}}$ is the maximum error of $J_{\text{DMP},i} - J_{\text{OC},i}$. The grid in the experiment was chosen as point-to-point motion from $y = 0.9 : 1.0$ and $z = 0.35 : 0.45$ with a step size of 0.02 m along each direction, see Fig. 5 (bottom left). $J_{\text{OC}}$ is the result of solving the optimal control from (10) and $J_{\text{DMP}}$ is obtained with the OMF and calculating the cost with

$$J = \sum_{i=1}^{N} \int_{t_0}^{t_f} \dot{\Theta}_i^2 dt.$$ In this case, the best generalization results are obtained with the distance based metric with $\delta = 0.09 \ 7$. To compare with standard trajectory generation, a 5th order polynomial trajectory was used to reach the goal configuration as well [33]. The according Ipol5 routine is given by

$$q = a_0 + a_1 t^3 + a_2 t^4 + a_3 t^5,$$

(12)

$^{5}$2-NN Y denotes the interpolation between the first and second NN geometric distance based.

$^{7}$This value was found by minimization of the relative error using standard SQP.
with \( t_f = 1 \text{s} \), \( a_0 = q_0 \), \( a_1 = \frac{20q_f - 20q_0}{2t_f^3} \), \( a_2 = \frac{30q_f - 30q_0}{2t_f^4} \), \( a_3 = \frac{12q_f - 120q_0}{2t_f^5} \), \( q_0 \) denotes the start and \( q_f \) the final configuration. Figure 6 depicts the overall comparison between OC generated trajectories, DMPs with best distance and cost based generalization, and polynomial interpolation (Average \( J_{err} = 48.03 \) %). To sum up, the distance based generalization yields the best results for the examined task.

Next, we discuss how the OMF is able to generate optimal trajectories for hitting the bin in real-time (trajectory planning runs at \( = 200 \) Hz). The throwing task is an illustrative example for explosive movements. Figure 7 shows the entire movement, consisting of the object’s flight trajectory and the end effector’s velocity. Four upper right: Learned trajectories in Cartesian space with flight trajectory of the ball. Lower four: Generalized throwing trajectories and velocities in joint space with resulting motor trajectories in comparison with optimal trajectories.

In this paper, we presented an approach for generalizing optimal throwing movements. Top left: Optimal and generalized throwing trajectories in Cartesian space with flight trajectory of the ball. Four upper right: Learned trajectories and velocities in joint space with resulting optimal trajectories. Lower four: Generalized throwing trajectories and velocities in joint space with resulting motor trajectories in comparison with optimal trajectories.

In the considered example, \( m = 3 \) trajectories for different throwing distances were optimized and generalized with \( n = 200 \) Gaussian bases. Furthermore, non-optimized goals are simulated for a comparison between the desired and resulting goal positions, see Fig. 7.

The experimental setup is as follows. A bin is placed at a desired position along the throwing plane. With a Microsoft Kinect the distance \( d_{Bin} \) to the bin is measured and then, the OMF is able to generate the correct trajectory for hitting the bin in real-time (trajectory planning runs at 1 kHz). Figure 8 depicts a photo series of three different bin distances being experimentally validated.

Figure 7 shows the entire movement, consisting of the throwing task and the subsequent stopping motion. These trajectories visualize the comparison between online OMF trajectories and optimal ones. Again, the generalized motions are very close to the true optimal solution. Opening the hand for releasing the ball is triggered at \( t = 1 \text{s} \). The repetitive accuracy to hit the bin is very high (in average \( > 80 \) %), if the ball can be placed repetitively in the hand, which needs some training. In another experiment, we simply let HASy throw the ball to our ball catching demonstrator Justin, whose location is again measured. Justin is then able to track the incoming ball and catch it accordingly, see Fig. 9. Important to notice is that the systems are not communicating with each other.

**VI. Conclusion**

In this paper, we presented an approach for generalizing optimal motions in real-time based on optimal sample trajectories that are then encoded into a DMP system for generalization. The developed optimal motion framework (OMF) performs a variety of optimal motions and was validated in simulation and experimentally on the anthropomorphic robot HASy. Our results demonstrate the possibility of combining optimality and generality even for its use in real-time. Learned optimal trajectories can be exactly reconstructed in real-time and even the extrapolation to other tasks show near-optimal behavior. The comparison of common movement generators and generalized point-to-point movements shows that the generalized trajectories produce significantly better results in terms of the considered cost function.
ACKNOWLEDGMENT

This work has been partially funded by the European Commission’s Sixth Framework Programme as part of the project SAPHARI under grant no. 287513.

REFERENCES


