Robotics and Autonomous Systems 94 (2017) 1–11

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

Generation of movements with boundary conditions based on optimal control theory

Sebastian Herzog, Florentin Wörgötter, Tomas Kulvicius *

Department of Computational Neuroscience, University of Göttingen, Germany

HIGHLIGHTS

- Novel trajectory generation method for generalization of accurate movements with boundary conditions.
- Originates from optimal control theory and is based on a second order dynamic system.
- Has most of the properties of the state-of-the-art trajectory generation methods.
- Can reproduce trajectories with zero error.
- Has great potential for various robotic applications, especially, where high accuracy is required.

ARTICLE INFO

Article history:
Available online 27 April 2017

Keywords:
Dynamic movement primitives
Movement generalization
Accurate trajectory representation
Boundary conditions
Optimal control

ABSTRACT

Trajectory generation methods play an important role in robotics since they are essential for the execution of actions. In this paper we present a novel trajectory generation method for generalization of accurate movements with boundary conditions. Our approach originates from optimal control theory and is based on a second order dynamic system. We evaluate our method and compare it to the state of the art movement generation methods in both simulations and real robot experiments. We show that the new method is very compact in its representation and can reproduce reference trajectories with zero error. Moreover, it has most of the features of the state of the art movement generation methods such as robustness to perturbations and generalization to new position and velocity boundary conditions. We believe that, due to these features, our method may have potential for robotic applications where high accuracy is required paired with flexibility, for example, in modern industrial robotic applications, where more flexibility will be demanded as well as in medical robotics.

1. Introduction

Many different robotic trajectory generation methods exist ranging from the conventional methods such as splines [1,2] to the state of the art methods such as dynamic movement primitives and its modifications (DMPs, [3–8]), Gaussian mixture models (GMMs, [9,10]), and a recent framework called probabilistic movement primitives (PMPs, [11]). DMPs, GMMs and PMPs have some advantages as compared to splines since they are robust to perturbations, can generalize to new situations and can be augmented by additional coupling terms and learning. DMPs and GMMs are based on dynamic attractor systems and converge to the target (end-point) asymptotically. This means that there will be always a small error at the desired end-point of both position and velocity profile which might be disadvantageous in applications where high precision is required, e.g., in industrial or medical robotics. The problem of errors at the boundary (position and velocity) has been solved by the PMPs, however, due to their probabilistic nature PMPs cannot represent the whole trajectory as accurate as human demonstration, i.e., with zero error. Note that in general DMPs, GMMs and PMPs are not meant to represent demonstrated trajectories accurately but rather initialize and train a model to reproduce similar trajectories.

Several recent approaches utilized optimal feedback control in order to optimize robot motions [12,13]. Inspired by these approaches, we present a novel framework for trajectory generation which has most of the features of the state of the art methods and is, in addition, highly accurate. Here by "trajectory generation" we mean that trajectories are generated on-line (similar to DMPs and GMMs), also in cases where boundary conditions are changed. We call our method *Optimal Control Primitives (OCPs)* as it originates and is derived from the optimal control theory. The novelty of this approach is a combination of a linear–quadratic regulator (LQR) controller with a representation of trajectories utilizing Chebyshev polynomials ([14], similar to splines but different from splines

* Corresponding author.
E-mail address: toku@mmmi.sdu.dk (T. Kulvicius).

http://dx.doi.org/10.1016/j.robot.2017.04.006
0921-8890/© 2017 Elsevier B.V. All rights reserved.

can also use polynomials of higher order). The advantage of the LQR controller is that, different from a PID controller, it allows controlling the deviation from a desired position trajectory as well as the deviation from a desired velocity profile. We will show that our method can reproduce demonstrated trajectories with zero error and has most of the features of the state of the art methods such as DMPs, GMMs and PMPs. Moreover, we will demonstrate that our method has less error (deviation from the demonstrated trajectory) as compared to DMPs and PMPs when generalizing to new boundary conditions.

In [15] optimal control was used to derive control systems for every part of a robot to perform a specific motion, e.g., rotate or move a finger. Systems like this were extended to even more complex systems by building skeleton models [16]. Such skeleton models can be used for even very complex cases like modelling and identification of emotional movements [17]. In our study, instead of building models for control of specific motions for particular robots, we present a general framework for generation of on-line trajectories (similar to DMPs, GMMs and PMPs) based on optimal control theory (i.e., robot unspecific).

The paper is organized as follows. We first will start with a formalism of our new method which will be presented in Section 2. Afterwards, in Section 3 we will present results from simulations where we will evaluate and compare our method to DMPs and PMPs. This will be followed by a validation and application of our method in two robot experiments. Finally, we will discuss our results and conclude our paper in Section 4.

2. Methods

We derive our trajectory generation method from optimal control theory where we assume a second-order control system with a state vector $\mathbf{x} = [\xi, \dot{\xi}]^T$ where ξ is the position and $\dot{\xi}$ the velocity and u is a control signal:

$$\frac{d\xi}{dt^2} = \frac{d\dot{\xi}}{dt} = u. \tag{1}$$

Note that a trajectory is determined by two variables; position and velocity. To control the trajectory the velocity state can be manipulated by changing u. Using Assumption 1 it is possible to model the motion of an arbitrary particle in space controlled by a force $u(t)$ depending on the time t, which leads to the canonical frame of a second-order control system, the double integrator:

$$\dot{\mathbf{x}} = A\mathbf{x}(t) + Bu(t), \tag{2}$$

where $\mathbf{x}(t) \in \mathbb{R}^2$, $u(t) \in \mathbb{R}$, and $\mathbf{x}(0)$ is given. Here, $\mathbf{x} =$ is the state vector containing the position ξ and the velocity $\dot{\xi}$ of a motion in a one-dimensional space. Matrices A and B are defined as follows:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

We also require that $|u(t)| \leq u_{max}$, where u_{max} is the maximal acceleration, i.e., acceleration is bounded.

Therefore, the motion of the particle is treated as a movement trajectory where the position and the velocity are coupled. To accurately encode and reproduce a trajectory from human demonstration $\mathbf{x}_r$ (here human demonstration serves as a reference trajectory $\mathbf{x}_r$) we have to assure the following condition (known as *trajectory tracking* problem [18,19]):

$$\lim_{t->T} \|\mathbf{x}(t) - \mathbf{x}_r(t)\| = 0, \tag{3}$$

where T is the terminal time (duration of the movement trajectory). Here $\|.\|$ is the Euclidean norm. In order to fulfil condition (3), we have to appropriately choose $u(t)$. For simplicity the time parameter t will not be used in the following equations. Thus, to fulfil (3) *gain scheduled* control is used such that

$$u = -K(\mathbf{x} - \mathbf{x}_r). \tag{4}$$

To acquire K the linear–quadratic regulator method (LQR) is used with the gain scheduled control

$$u = -K\mathbf{x} \tag{5}$$

and the cost function

$$\hat{J} = \frac{1}{2}\int_0^T \left(\mathbf{x}^T Q_x \mathbf{x} + u^T Q_u u\right) dt + \frac{1}{2}\mathbf{x}^T(T) P \mathbf{x}(T). \tag{6}$$

Here, $Q_x \geq 0$, $Q_u \geq 0$, $P \geq 0$ are symmetric, positive (semi-) definite matrices. The Hamiltonian formulation for this is

$$H = \mathbf{x}^T Q_x \mathbf{x} + u^T Q_u u + \lambda^T (A\mathbf{x} + Bu).$$

But instead of solving the Hamiltonian problem the algebraic Riccati equation is considered

$$PA + A^T P - PBQ_u^{-1}B^T P + Q_x = 0, \tag{7}$$

where

$$P = \begin{bmatrix} a & b \\ b & c \end{bmatrix}, Q_x = \begin{bmatrix} q_1^2 & 0 \\ 0 & q_2^2 \end{bmatrix} \text{ and } Q_u = 1$$

are used to solve (7) such that

$$u = -Q_u^{-1}B^T P\mathbf{x}, \tag{8}$$

where

$$K := Q_u^{-1}B^T P = \begin{bmatrix} q_1 & \sqrt{q_2^2 + 2q_1} \end{bmatrix}. \tag{9}$$

Setting K to (2) results in

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + B(-K(\mathbf{x} - \mathbf{x}_r)) \\ &= (A - BK)\mathbf{x} + BK\mathbf{x}_r. \end{aligned} \tag{10}$$

A detailed derivation of gain matrix K is given in the Appendix A.

Since the dynamics of the system is given by (10) the choice of q_1, q_2 is important. High values of q_1 will result in a fast reaction of the system in order to compensate deviations from the position profile. This fast reaction can lead to jumps in the velocity profile, e.g., see Fig. 6(b1). On the other hand, low values of q_1 will result in a slow reaction of the system which as a consequence will lead to higher errors in tracking accuracy. In case of very short trajectories, a too slow reaction of the system (very low values of q_1) can even lead to discontinuities.

By tuning the values q_1, q_2 it is possible to adjust the weighting between the importance of position ξ and velocity $\dot{\xi}$. This may be useful in cases where $\dot{\xi}$ is not the derivative of ξ, i.e., if we want to have a velocity profile different from the velocity profile of the original trajectory. If $q_1 \gg q_2$ then it would lead to a trajectory where the position profile is more accurate than the velocity profile, and vice versa, if $q_2 \gg q_1$ then the system is more accurate in reproducing velocity profile but would deviate substantially from the position profile. An example of such a case is shown in Fig. 1 where we can see that increasing q_1 with respect to q_2 leads to a trajectory where the position profile is more accurate than the velocity profile (see panel (a)), i.e., the system tries to follow the given position profile and will be more inaccurate in the velocity profile. Vice versa, if $q_2 \gg q_1$ then the system is more accurate in reproducing velocity profile but deviates from the position profile (see panel (b)). Since in our other cases the velocity profile $\dot{\xi}$ is indeed the derivative of the position profile ξ, we used $q_1 = q_2 = 1$, i.e., the tracker is supposed to track both position and velocity accurately. Here, Q_u acts as a stiffness parameter, where

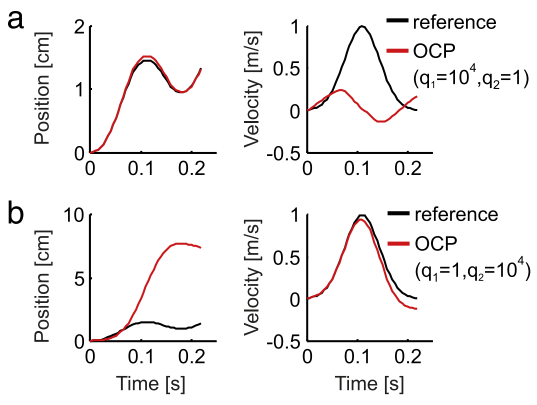


Fig. 1. Influence of parameters q_1 and q_2 on accuracy in reproduction of position and velocity profiles: (a) $q_1 = 10^4$ and $q_2 = 1$, and (b) $q_1 = 1$ and $q_2 = 10^4$. Here we used the X-profile of letter 'a' (see Fig. 4) as reference trajectory. Note that in this case the reference velocity is not the derivative of position profile (we generated velocity profile manually).

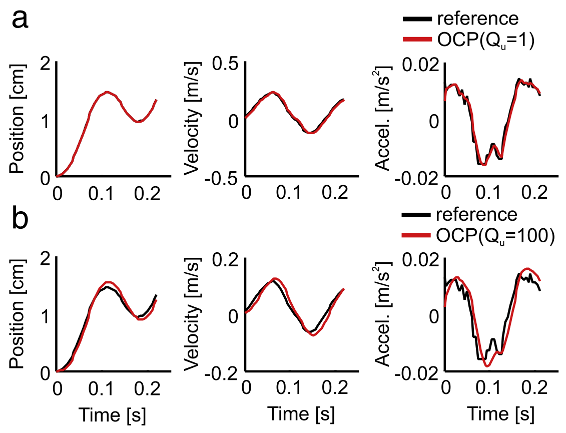


Fig. 2. Influence of parameter Q_u on accuracy in reproduction of position, velocity, and acceleration profiles: : (a) $Q_u = 1$, and (b) $Q_u = 100$. Here we used the X-profile of letter 'a' (see Fig. 4) as reference trajectory.

increasing this parameter leads to a stiffer behaviour of the control u. It can be used to adjust the balance between tracking accuracy and smoothness. A relatively small value of Q_u indicates a strong and strict controller, which may lead to a spiky trajectory, and vice versa, a value above 1 leads to a less strict control behaviour which may result in a smoother trajectory but with bigger deviations from reference trajectory. An example of such a case is shown in Fig. 2 where we can see that an increase in Q_u value leads to smoother but less accurate trajectories. In our case we used $Q_u = 1$ in all other experiments.

2.1. Trajectory representation

In the general case, as presented above, OCPs require a complete reference trajectory in order to reproduce it, whereas DMPs, GMMs and PMPs allow encoding motion by a set of parameters, where the number of needed parameters is less than the number of points in the reference trajectory [4,20,11]. In the following we will formalize how an equally compact representation can also be obtained for OCPs.

To get a compact representation of a trajectory $\mathbf{x}$, u is reduced to a discrete number of points $u^* = \{u_1^*, u_2^*, \ldots, u_{\hat{T}}^*\}$ where $\hat{T} \ll T$. In addition, the system (2) is extended to

$$\dot{\mathbf{x}} = A\mathbf{x} + B\phi(u), \tag{11}$$

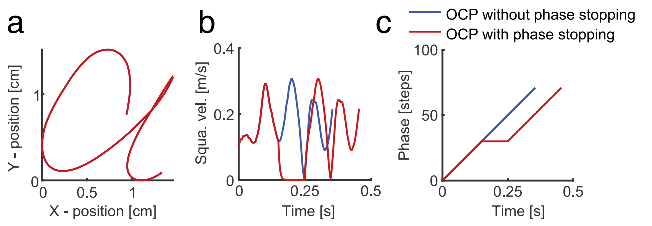


Fig. 3. An example of phase stopping: (a) position profile, (b) velocity profile, and (c) phase variable τ.

where $\phi(.)$ is an arbitrary interpolation operator which is used to reconstruct u from u^*.

In our case an interpolation using Chebyshev polynomials of the First Kind was used [14]:

$$\phi(\mathbf{u}^*) = \sum_{k=0}^{N} c_k T_k(u^*),$$

where the coefficients c_k will converge to $a_k = \frac{2}{\pi}\int_{-1}^{1}\frac{f(u^*)T_k(u^*)du^*}{\sqrt{1-u^{*2}}}$ as $k \to \infty$. Here, $T_k(u^*) = \cos(k * \arccos(u^*))$ is the Chebyshev polynomial of kth order.

In case a higher compression is required, it may be necessary to regularize u^* in such a way that the error between the original u and $\phi(u)$ is as small as possible. This can be achieved, for example, by using a genetic algorithm [21].

In summary, to represent a trajectory in a compact form, instead of Gaussian kernels as used in DMPs, GMMs and PMPs, we use Chebyshev polynomials[1] for which we only need to know $N + 1$ via points (sampled from the reference trajectory) if we need N polynomials in order to reconstruct u^*.

In order to make the system only indirectly dependent on time (phase based as in DMPs and PMPs [3,11]) and be able to represent both discrete and periodic movements we replace the time variable t in (2) by a phase function $\tau(.)$. For generation of discrete motions τ is simply a linear function

$$\tau(t) = t.$$

In case of periodic (repetitive) movements one can use a simple periodic function with respect to time $\tau(t)$ where T defines time of one period:

$$\tau(t) = t \bmod T,$$

where mod denotes the modulo operation.

An example of phase stopping is shown in Fig. 3 where we stopped phase variable τ be setting it to a constant value at approx. 0.13 s for some period of time and let the system proceed as normal at approx. 0.25 s (see panel (a)). One can see that the system is stalled (velocity goes down and stays at zero) for that period of time (see panel (b)) but recovers and follows the reference velocity profile as soon as the phase variable has been "released". As expected, there is no deviation from the position profile (see panel (a)). Note that in the following experiments the phase variable is not altered when generalizing to new boundary conditions.

2.2. Generalization

The ability to generalize to new situations, i.e., to new start-and/or end-points is a relevant feature of modern trajectory generators. This generalization is implemented in the OCP framework in

[1] A Matlab demo and source code of our framework can be downloaded from http://www.dpi.physik.uni-goettingen.de/cns/index.php?page=optimal-control-primitves.

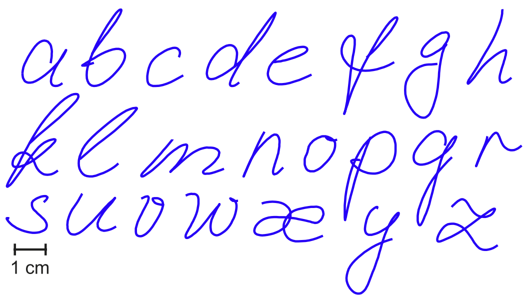


Fig. 4. Dataset used for comparison of different movement generation frameworks.

the following way. In general, in order to generate a trajectory with new boundary conditions, e.g., with a new start-point $\mathbf{x}(0) = \mu$, and a new end-point $\mathbf{x}(T) = \nu$ the *two point boundary* value problem has to be solved. In order to decrease computational cost the *initial* value problem is solved two times, instead. For the new start state the Eq. (10) is solved with a $\mathbf{x}(0) = \mu$ and the known u from (4). For the new terminal state we set $\mathbf{x}(0) = \nu$ and u becomes

$$\hat{u} = u_T, u_{T-1}, \ldots, u_1. \tag{12}$$

Let us define $\mathbf{x}^{(init)}$ as the solution of (10) for the new initial state and $\mathbf{x}^{(ter)}$ as the solution for the new terminal state. Then, the whole solution for a trajectory with changed initial and terminal states becomes

$$\mathbf{x} = x_1^{(init)}, \ldots, x_{\lfloor T/2 \rfloor}^{(init)}, x_{\lfloor T/2 \rfloor+1}^{(ter)}, \ldots, x_1^{(ter)}. \tag{13}$$

Note that in case of very short trajectories with less then four points, where

$$x_{\lfloor T/2 \rfloor}^{(int)} - x_{\lfloor T/2 \rfloor+1}^{(ter)} \leq \Delta x, \tag{14}$$

with Δx being the maximal allowed discontinuity, it is necessary to solve the two point boundary problem like in [22]. However, this was not necessary for any of our test cases as presented in the next section.

3. Results

In the following we will compare the performance of our framework to the performance of DMPs and PMPs with respect to three aspects: (1) accuracy in reproduction of human trajectories, (2) robustness to perturbations, and (3) position and velocity generalization.

For benchmarking and evaluation we used human handwritten letters (see Fig. 4) as already used by some previous studies [4,8]. We used the same procedure in order to obtain letter samples as used by Kulvicius et al. [8]. Data was collected by utilizing a pen tablet (Wacom Intuos3 A3 Wide DTP) with a size of 48.8 cm × 30.5 cm, resolution of 5080 lpi and a sampling rate of 200 Hz.

For comparison with DMPs we used a version of DMPs for generation of hitting and batting movements [5] since it allows generalization to both position and velocity boundary conditions. For learning the weights of the Gaussian kernels we used the $\delta -$ *rule* as described in Kulvicius et al. [8]. For comparison with PMPs we used the framework presented in [11] and for learning PMP weights we used an expectation maximization algorithm approach were

$$\mathbf{w} = (\phi^T \phi + I\epsilon_y)^{-1}(\phi^T \mathbf{y}_{ref}) \tag{15}$$

with the basis matrix ϕ, identity matrix I, a regularization parameter ϵ_y to increase the condition number of the system and a reference trajectory $\mathbf{y}_{ref}$.

3.1. Trajectory reproduction

First of all we compare the performance of our framework to DMPs and PMPs with respect to trajectory reproduction. For qualitative comparison we use the letter "a" from the letter dataset as shown in Fig. 4. Here we analysed how the accuracy of trajectory reproduction is influenced by the number of the basis functions (Gaussian kernels in case of DMPs and PMPs, and Chebyshev polynomials in case of OCPs) used to represent the trajectory. Note that for DMPs and PMPs we used equally distributed kernels with equal variance where the variance was tuned in order to get best accuracy in trajectory reproduction. The results for reproduction of a human motion trajectory are shown in Fig. 5 where we used 10, 15 and 20 basis functions. We can see that OCPs can reproduce the reference trajectory more accurately than the other methods when 15 and 20 basis functions are used. We also quantified the movement reproduction property of the different approaches statistically where we used all 23 letters from the dataset (see Fig. 4). We calculated the accuracy of trajectory reproduction for different numbers of basis functions: 10, 12, 14...40. Statistics for all three methods are presented in Fig. 5(d) where we can see that OCPs require fewer basis function to reproduce trajectory accurately as compared to DMPs and PMPs. In case of OCPs, it was already possible to reproduce trajectories with zero error with 32 basis functions, whereas for DMPs and PMPs the error drops around 35 basis functions and is below 0.1% for position and below 5% for the velocity profile. Based on these results, in order to make a fair comparison of the different methods, in the following experiments we used 35 basis functions for all three methods.

3.2. Robustness to perturbations

Secondly, we looked at the behaviour of our trajectory generation method with respect to perturbation robustness where we again compared our method against DMPs. We perturbed the system at some arbitrary time by shifting the position profile in both X and Y direction by 30%. The results of such a test are shown in Fig. 6 where we present the behaviour of DMPs in panel (a) and two cases for the OCPs in panel (b): (1) without bounded acceleration (OCP) and (2) with bounded acceleration (OCP bound). Allowing the solver for the dynamic system to do arbitrary big steps for the acceleration leads to an unbounded response, bounding the acceleration by u_{max} leads to the bounded response. Note that in our case u_{max} was set arbitrary for demonstration purposes only. In general, it could be set to the maximum acceleration obtained from the demonstration trajectory or it could be set to the maximum allowed acceleration of the robot used. We can observe that in case of unbounded acceleration the OCP system produces a velocity jump, which is different from the behaviour of the DMP system and might be even dangerous for robotic applications. This is due to the fact that OCPs are optimal with respect to minimal position and velocity deviation depending on the choice of Q_u, where the system tries to come back to the original trajectory, as soon as possible. Such undesired jumps in the velocity profile can be avoided by limiting the maximal acceleration u_{max}. We can see that by limiting acceleration (we set the value to the maximum acceleration of the reference trajectory) we can obtain much smoother response of the system. Moreover, we can also observe that the OCP system converges to the original path much sooner as compared to DMPs.

3.3. Position and velocity generalization

We also analysed how well can our trajectory generation method generalize to new situations, i.e., when position or velocity boundary conditions are changed. Here we compared the

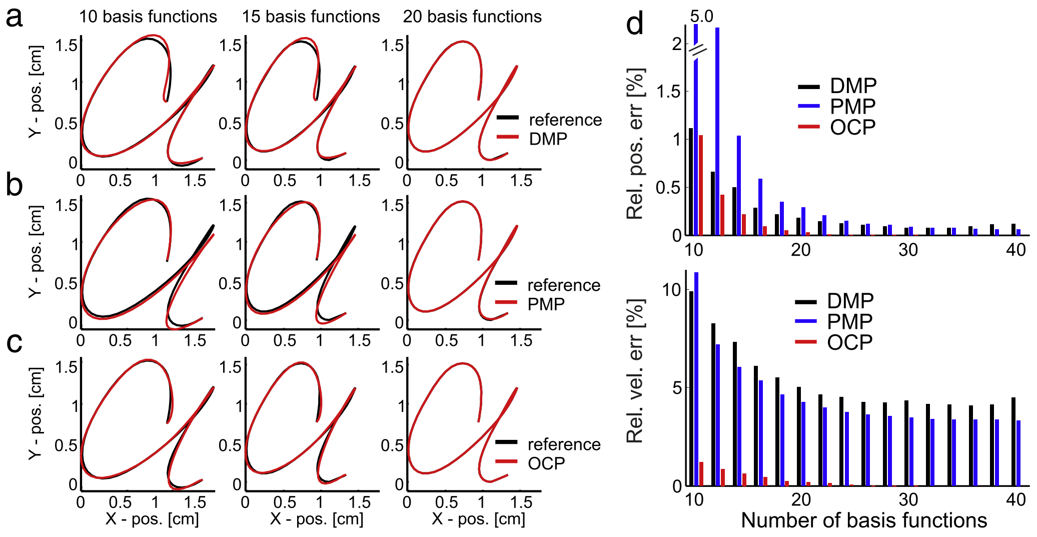


Fig. 5. Results for reproduction of human demonstration (reference) for (a) DMPs, (b) PMPs, and (c) OCPs. Position profiles are shown for each case. (d) Influence of the number of basis functions on the accuracy of trajectory reproduction. Mean relative position (top) and velocity (bottom) error for the whole trajectory obtained from 23 letters.

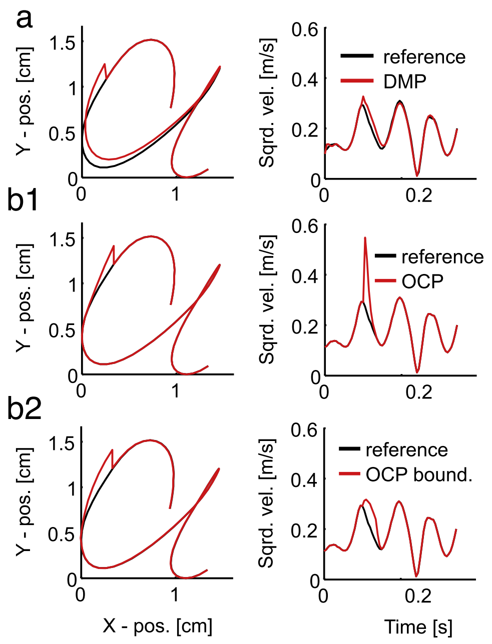


Fig. 6. Comparison of robustness to perturbation between (a) DMPs, (b1) OCPs without bounded acceleration (OCP), and (b2) with bounded acceleration (OCP bound). Position (left) and velocity (right) profiles are shown for each case.

performance of our framework to both DMPs and PMPs. Note that in case of PMPs it is rather control than generalization, since one requires to recalculate kernel parameters whenever boundary conditions are changed. First, we compared the behaviours of the three systems when position boundary conditions are changed. To do so, we changed the end-point position (for both X and Y components) by relatively increasing the end-point value of the reference trajectory by 30%. Note that in this case we kept the velocity boundary conditions the same than those for the reference trajectory. Qualitative results are presented in Fig. 7(a1–c1) where we can see that performance of PMPs and OCPs is different from that of DMPs. In case of DMPs we get much larger deviations from the original trajectory (along the whole trajectory) as compared to PMPs and OCPs where deviations are only at the end of the trajectory. Statistics for the position generalization are shown in Fig. 7(d1) where we show average relative position and velocity error obtained from all 23 letters (see Fig. 4). In general, we can observe that, when using DMPs, deviation from the reference trajectory increase dramatically when increasing the change in end-point position. Also, the deviation of the velocity profile is much higher as compared to PMPs and OCPs. The results also demonstrate that OCPs produce smaller deviations from the reference trajectory (in both position and velocity profile) as those observed with PMPs. Mean relative position error at the desired new end-point for DMPs was 0.008%, whereas for PMPs and OCPs the error was zero. This is due to the fact that DMPs converge to the end point only asymptotically [4].

Similar to position generalization, we also performed a test where we compared all three frameworks with respect to the change of velocity boundary conditions. So, here we changed the end-point velocity by scaling it relatively to the end-point velocity of the human demonstration. End-point position was kept unchanged. Qualitative results for the 30% end-point velocity change are shown in Fig. 7(a2–c2) whereas statistics are shown in Fig. 7(d2). Similar to the results obtained from position generalization we observe that DMPs result in much larger position and velocity deviations as compared to PMPs and OCPs. Also as in the previous case, OCPs outperform PMPs by producing only deviations at the very end of the trajectory, which is necessary in order to meet the new boundary conditions. Similar to the case of position generalization, relative velocity error at the desired new end-point for PMPs and DMPs was zero, whereas mean relative velocity error at the desired new end-point for DMPs was 0.434%, which again is due to the asymptotic convergence. We summarize results for position and velocity generalization in Fig. 8(a) and (b), respectively, where we show mean relative position and velocity errors for the whole trajectory obtained from 23 letters. Results demonstrate that OCPs outperform both DMPs and PMPs in that the OCP-system can follow the reference trajectory better when generalizing to new boundary conditions.

In addition, we also performed a test were we compared behaviour of all three methods when we changed both position and velocity boundary conditions at the same time. As in the previous cases we scaled both position and velocity end-points by 30%. Results are presented in Fig. 7(a3–d3). As expected, we can see that results are similar to those as already presented in Fig. 7(a1–d1) and Fig. 7(a2–d2), where, as in previous cases, for DMPs we obtain large deviations from the reference movement along the

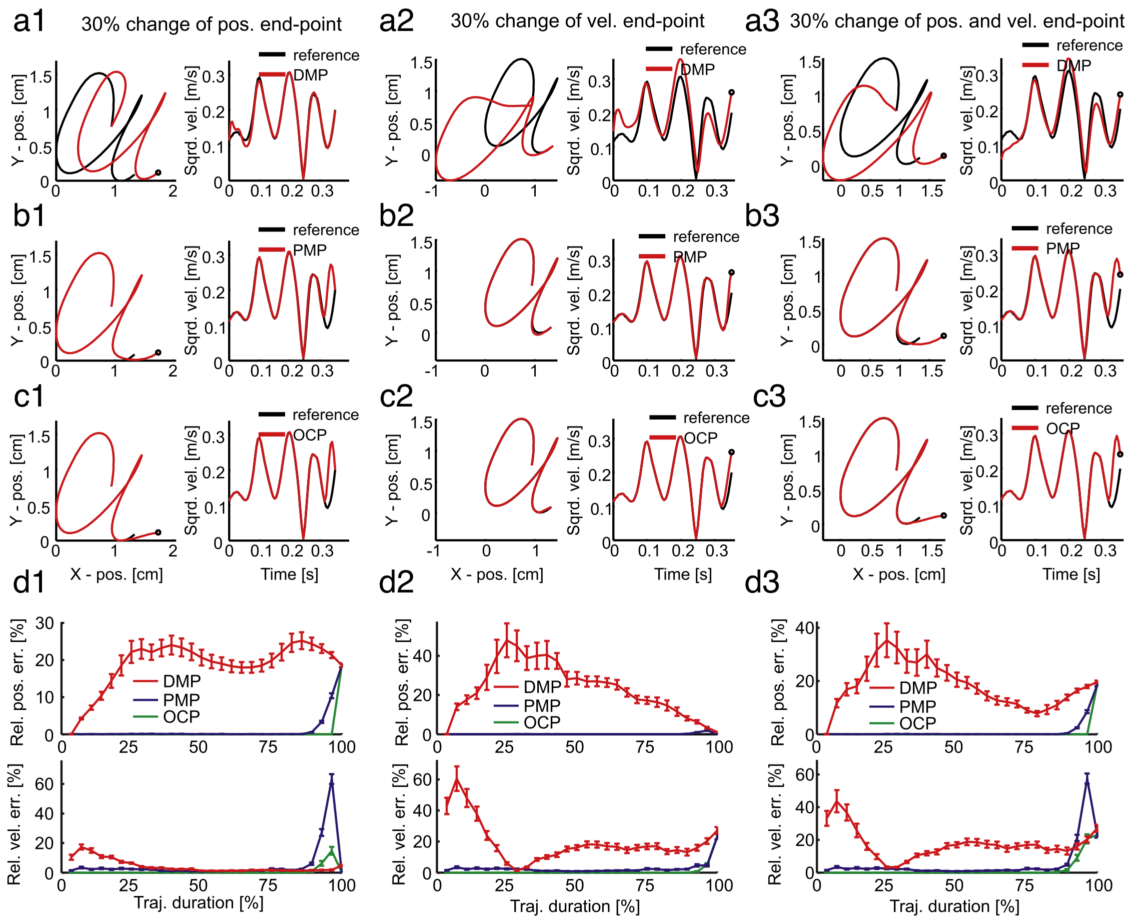


Fig. 7. Examples of (a1–c1) position and (a2–c2) velocity, and (a3–c3) position and velocity generalization for (a) DMPs, (b) PMPs and (c) OCPs. Here we changed position and/or velocity at the end-point by 30%. Position (left) and velocity (right) profiles are shown for each case. The circles denote new end-points. (d) Statistics for (d1) position, (d2) velocity, and (d3) position and velocity generalization. Mean relative position (top) and velocity (bottom) error are shown for each case. Error bars denote standard deviation.

whole trajectory when boundary conditions are changed, whereas PMPs and DMPs deviate from the reference movement only at the end of the trajectory, with the advantage of OCPs in tracking the trajectory more accurately as compared to PMPs. Mean relative position and velocity error at the desired new end-point for DMPs was 0.118% and 0.435%, respectively, whereas for PMPs and OCPs, as in previous cases, the errors were zero.

We summarize results for position and velocity generalization in Fig. 8 where we show mean relative position and velocity errors for the whole trajectory obtained from 23 letters for all tree cases as presented above: (1) change of position end-point (see panel (a)), (2) change of the velocity end-point (see panel (b)), and (3) change of the both position and velocity end-points (see panel (c)). Results demonstrate that OCPs outperform both DMPs and PMPs in that the OCP-system can follow the reference trajectory better when generalizing to new boundary conditions.

3.4. Robot experiments

DMPs are currently probably still the most widely used dynamic and adaptive method for the generation of generalizable trajectories. Thus, we compared the performance of our approach against DMPs where the differences between the systems becomes most clearly visible. Performance differences of OCPs and PMP had been quantified above in Figs. 5, 7 and 8.

For this, we compared performance of OCPs and DMPs in two robot experiments: (1) a pouring task and (2) a via-point task, where the task for the robot was to generalize to new situations

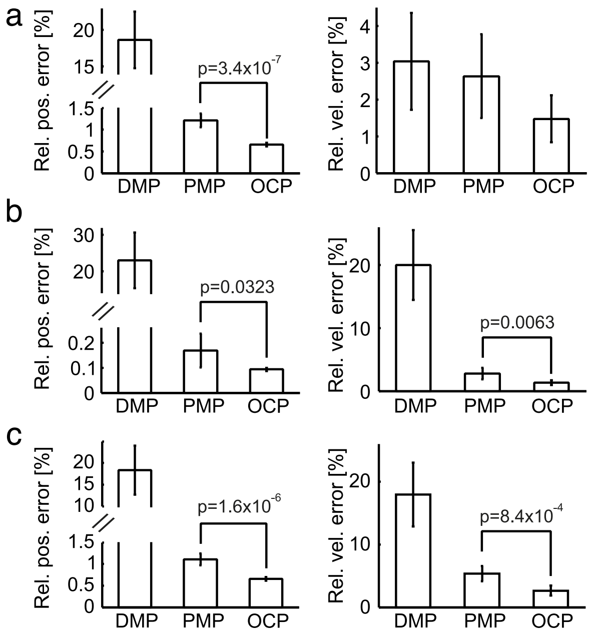


Fig. 8. Statistics for (a) position, (b) velocity and (c) both position and velocity generalization. Mean relative position (left) and velocity (right) error for the whole trajectory is shown for each case. Error bars denote confidence intervals (95%) of mean, where p stands for the probability of the *t-test* (significance level $\alpha = 0.05$).

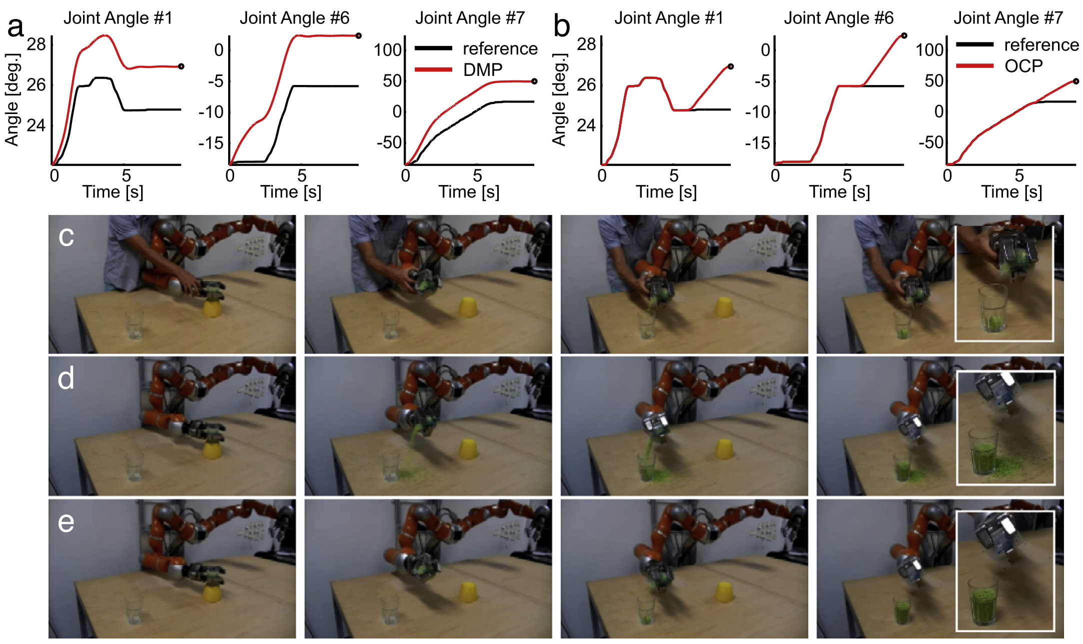


Fig. 9. Results from pouring experiment. (a,b) Resulting joint (position) trajectories for a new position end-point obtained with (a) DMPs and (b) OCPs. Note that here we show only joint trajectories which differed from human demonstration. The circles denote new end-points. (c–e) Selected frames from action execution performed by (c) a human, (d) a robot utilizing DMPs and (e) a robot utilizing OCPs.

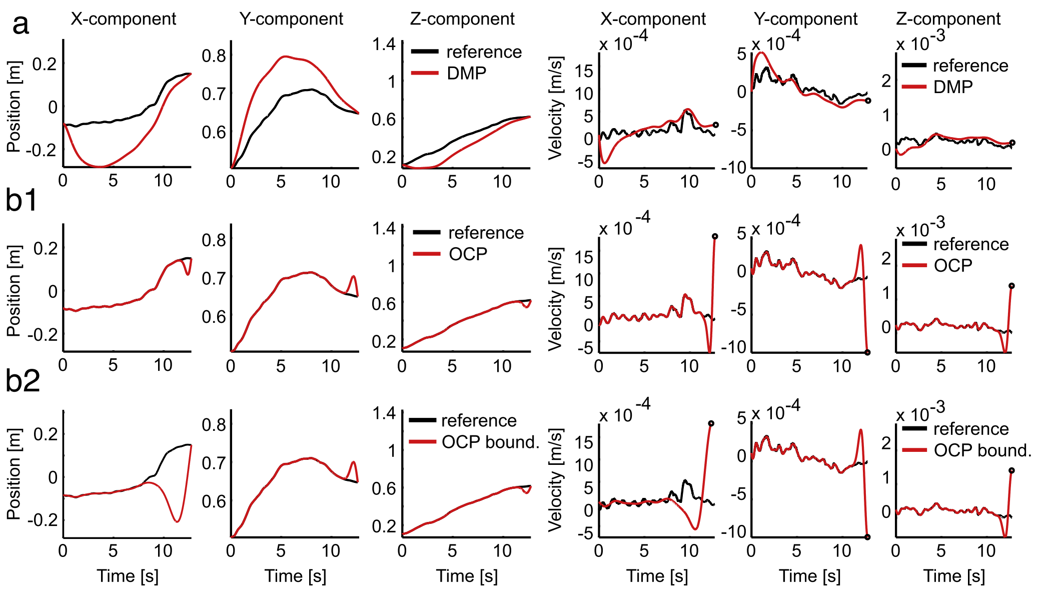
based on a human demonstrated trajectory. For our experiments we used a seven DOF "KUKA-LWR" robot-arm. Human trajectories were recorded by using kinaesthetic guidance and were encoded with DMPs and OCPs as described above.

In the first experiment, the pouring task, a human demonstrated how to pour 50 ml of sand from a container containing 200 ml of sand into an empty glass (see Fig. 9(c) and supplementary video, Appendix B). The task for the robot, different from human demonstration, was to empty the container completely, i.e., to pour all the sand (200 ml) from the container into the glass. To do so, obviously, the robot needed to tilt the container much more as compared to human demonstration. In this case we performed the task in joint-space were we manually set the end-points of three joint position profiles in order to tilt the wrist as much as to empty the glass completely. Note that different from other approaches used in learning to pour tasks [23,24] we do not change parameters (weights in case of DMPs) of the trajectory generation method but only boundary conditions in order to generalize to the new situation. The resulting trajectories for DMPs and OCPs are shown in Fig. 9(a) and (b), respectively. We can see that in the case of DMPs, as already demonstrated in Fig. 7(a1), the trajectory is changed along the whole path when generalizing to new position end-points. This leads to the fact that the robot starts tilting the container too early and, as a consequence, spills some of the sand on the table (see Fig. 9(d)). By contrast, OCPs change the trajectory only at the end of the movement, which enables the robot to pour the sand into the glass without spilling.

In the via-point task we were concerned with precision and generalization to a new end-point velocity. In this scenario we placed three bottles of different size on a table and put bottle caps upside-down on the top of each of them. The task was to flick off all caps from the bottles while moving from one bottle to the other by hitting the caps with a pen which was held in the robot's hand. A human demonstration of such an action is shown in Fig. 10(c) (see also supplementary video, Appendix B). In case of a human demonstration the human stopped at the last bottle, i.e., at the end of the motion the velocity was zero (see the reference velocity profile in Fig. 10(a)). Different from this, the task for the robot was to hit the cap of the last bottle with non-zero velocity. In this case, we performed the action in Cartesian space where we only changed end-point velocity of the X, Y and Z components. We manually changed the end-point velocity for X, Y and Z component to non-zero values. The new velocity at the end-point for OCPs was 0.25 cm/s where for DMPs it was 0.02 cm/s. We could not use higher velocities for DMPs since higher velocities produced too large deviation from the reference trajectory (in position space) and it was not possible to execute this with the robot due to the physical constrains of the machine. Results of the via-point task experiment are shown in Fig. 10 where we can see that in case of DMPs, as already demonstrated in Fig. 7(a2), the position profile deviates a lot when the end-point velocity is changed. In this case the robot hits the bottles with the hand instead of the pen. However, the robot was able to meet the boundary conditions at the end of the trajectory, which is consistent with the results shown in Fig. 7(a2). Different from DMPs, OCPs deviated only at the end of the trajectory (Fig. 10(b)), which is actually expected and desired in order to meet new boundary velocity condition. Here we show two versions of movement generalization with OCPs: (1) OCPs without bounded acceleration (see Fig. 10(b1,e1)), and (2) OCPs with bounded acceleration (see Fig. 10(b2,e2)). In the case of unbounded acceleration, the robot makes a zig-zag motion, i.e., stops before hitting the cap of the last bottle, moves back a bit and then moves forward again (Fig. 10(b1,e1); see also supplementary video, Appendix B). This is due to the fact that the robot requires a larger distance in order to accelerate and produce higher velocity at the end of the motion. In the case of bounded acceleration, we only bounded the acceleration of the X-component (in robot frame). Here, instead of going backward, the robot moves upward and then downward in order to accelerate and produce higher velocity at the end of the motion (Fig. 10(b2,e2); see also supplementary video, Appendix B).

4. Discussion

In this paper we presented a novel trajectory generation method for the generation of highly accurate movements with arbitrary position and/or velocity boundary conditions. We showed that the method is comparable to the state of the art methods



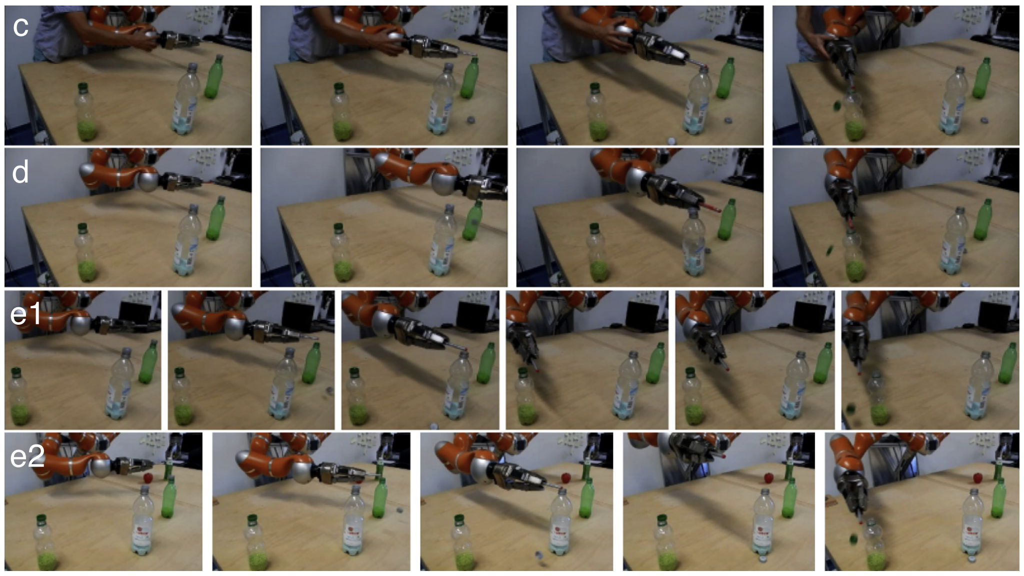


Fig. 10. Results from the via-point task. (a,b) Resulting position and velocity trajectories (X, Y and Z in robot frame) for a new velocity end-point obtained with (a) DMPs, (b1) OCPs without bounded acceleration (OCP), and (b2) OCPs with bounded acceleration (OCP bound). The circles denote new end-points. (c–e) Selected frames from action execution performed by (c) a human, (d) a robot utilizing DMPs, (e1) a robot utilizing OCPs without bounded acceleration, and (e2) a robot utilizing OCPs with bounded acceleration.

such as DMPs [4] and PMPs [11] in terms of features such as compactness of trajectory representation, robustness to perturbations, and generalization. Moreover, we showed that the new method produces more accurate trajectories, i.e., it deviates less from the demonstrated trajectory as compared to DMPs or PMPs when generalizing to new boundary conditions. It is important to note that, as already stated above, DMPs and PMPs are not meant to represent trajectories accurately. This comparison is by no means a devaluation of existing methods but rather shows behavioural differences of different trajectory methods where in some situations one method may be more preferable than the others and vice versa. For example, there may be cases where gradual transition from start-point to a new end-point will be more beneficial as compared to a change only at the end of the trajectory, and vice versa. We demonstrated by two robot experiments that in some situations the trajectories generated using the new method can be beneficial as compared to trajectories generated with DMPs. In general, the behaviour of OCPs is qualitatively similar to PMPs, however, different from PMPs and DMPs, OCPs can achieve higher accuracy in motion reproduction with fewer basis functions.

A comparison of DMPs, PMPs and OCPs in terms of features is provided in Table 1, where we can see that all methods are capable of position and velocity scaling, and generalization to new boundary (position and velocity) conditions. However, PMPs require an additional tracker on top of the trajectory generation model, whereas DMPs and OCPs (similar to DMPs) have a built-in tracker in the model. In general, compact trajectory representation with the OCP framework requires less parameters as compared to DMPs and PMPs. For OCPs, if we want to represent a trajectory with N polynomials, we only need to know the via points, which in total leads to $N + 1$ parameters. For DMPs and PMPs we would need for N Gaussian kernels in total $3N$ parameters (mean, variance and weight for each Gaussian function). Only if all kernels have the same width and are equidistantly distributed one would also have $N + 1$ parameters, but many situations exist (e.g., stiff systems) where the use of identical kernels is not optimal. For comparison, GMMs in total require $4N$ parameters in order

Table 1
Comparison of different movement generation frameworks with respect to their properties.

Property	Splines	DMPs	GMMs	PMPs	OCPs
Time dependence	Direct	Indirect	Independent	Indirect	Indirect
Built-in tracker	-	+	+	-	+
Start- and end-point position/velocity control	+/+	+/+	+/-	+/+	+/+
Via-point control	+	+	+	+	+
Start- and end-point position/velocity generalization	-/-	+/+	+/-	-/-[*]	+/+
Robustness to perturbations	-	+	+	-[**]	+
Acceleration bounding	-	-	-	-[**]	+
Number of parameters for trajectory representation (min/max)	$N+1$	$N+1$ / $3N$	$4N$	$N+1$ / $3N$	Arbitrary[***]

N is the number of basis functions.
[*] PMPs require recalculation of kernel parameters if boundary conditions are changed.
[**] It is not possible without a built-in tracker.
[***] Depends on the basis function.

to represent trajectory by a mixture of N 2D Gaussian kernels (in GMMs movements are represented in a phase space of position and velocity [20]). However, different from our method, GMMs can encode a set of several different trajectories, whereas our method (as DMPs) encodes only a single trajectory.

As already stated in introduction our approach utilizes Linear–Quadratic Regulator (LQR) control for tracking of trajectories. We have chosen LQR for the following reasons. In general, as shown in [25,26] and [27], LQR controller can be designed as a Proportional Integral (PI) controller. Moreover, as demonstrated by [28], the more specialized LQR controller has superior properties regarding oscillations as compared to a general Proportional Integral Derivative (PID) controller. Thus, we utilized LQR and not standard PID controller due to better tracking properties.

Our approach is similar to the one presented in [29], where LQR controller is utilized to track a desired trajectory by a UAV. Different from that approach, in our case, we used a more general model, the double integrator with a mass equal to one. Since our model is simpler, there is no need to reduce the dimensions of the problem like it is done in [29], which allows a more intuitive use of the controller and the parametrization. In addition, it is easier to add further constrains if necessary.

It is important to stress that the main advantage of our approach compared to other frameworks is that OCPs are not restricted with respect the way a trajectory is encoded, i.e., it does not necessarily have to be encoded with Chebyshev polynomials as shown in this study. In our case, different from DMPs, PMPs and GMMs, a trajectory is a (control-)signal, which means that one can use and apply any kind of control theory and/or signal processing in order to represent a trajectory. For example, a trajectory could be represented by a parabola or a sine-wave, too. Also, in contrast to spline based approaches like [30], the trajectory in our approach is a continues function and allows to bound the maximal acceleration and change the start and terminal point. These properties should also allow linking the OCP framework to existing control frameworks like ZMP [31] as well as admittance or impedance based control [32,33] much easier than for DMPs or PMPs. Future investigation into this are planned but would exceed the scope of this paper.

Common industrial applications, such as gluing or welding, often require highly exact position and velocity profiles for the to-be-performed trajectory. For example, the robot should not decelerate at corners, by which the gluing (or welding) track would become uneven. The attractive features of our method – compactness and high accuracy – could have a great potential especially for such applications.

Acknowledgements

This research was partially supported by the funding from BioRobotics Center, University of Southern Denmark (FORSK2020; 2014-2018). The research leading to these results has also received funding from the European Community's Seventh Framework Programme FP7/2007-2013 (Programme and Theme: ICT-2011.2.1, Cognitive Systems and Robotics) under grant agreement no. 600578, ACAT.

Authors thank Mohamad Javad Aein for the help with robot experiments.

Appendix A

In the following we will provide a detailed derivation of gain K (see Eq. (4)).

To acquire K the linear–quadratic regulator method (LQR) is used with the cost function

$$\hat{J} = \frac{1}{2}\int_0^T \left(\mathbf{x}^T Q_x \mathbf{x} + u^T Q_u u\right) dt + \frac{1}{2}\mathbf{x}^T(T) P \mathbf{x}(T). \qquad (A.1)$$

In order to solve cost function the Hamiltonian

$$H = \mathbf{x}^T Q_x \mathbf{x} + u^T Q_u u + \lambda^T (A\mathbf{x} + Bu)$$

should be solved. Instead of solving H directly the following conditions

$$\dot{\mathbf{x}} = \left(\frac{\partial H}{\partial \lambda}\right)^T = A\mathbf{x} + Bu \qquad \mathbf{x}(0) = x_0 \qquad (A.2)$$

$$-\dot{\lambda} = \left(\frac{\partial H}{\partial x}\right)^T = Q_x \mathbf{x} + A^T \lambda \qquad \lambda(T) = P_1 x(T) \qquad (A.3)$$

$$0 = \frac{\partial H}{\partial u} = Q_u + \lambda^T B \qquad (A.4)$$

can be deduced leading to

$$u = -Q_u^{-1} B^T \lambda. \qquad (A.5)$$

Substituting $\lambda(t) = P(t)x(t)$ into the conditions we get

$$\dot{\lambda} = \dot{P}\mathbf{x} + P\dot{\mathbf{x}} = P(A\mathbf{x} - BQ_u^{-1}B^T P)\mathbf{x} = Q_x \mathbf{x} + A^T P \mathbf{x}. \qquad (A.6)$$

To satisfy this equation a $P(t)$ is needed such that

$$-\dot{P} = PA + A^T P - PBQ_u^{-1}B^T P + Q_x \qquad P(T) = P_1, \qquad (A.7)$$

which can be rewritten as

$$PA + A^T P - PBQ_u^{-1}B^T P + Q_x = 0. \tag{A.8}$$

This equation is called *algebraic Riccati equation* and is numerically solvable, leading to the linear control law for the LQR problem

$$u = -Q_u^{-1}B^T Px \tag{A.9}$$

with

$$Q_x = \begin{bmatrix} q_1^2 & 0 \\ 0 & q_2^2 \end{bmatrix}, \qquad Q_u = 1.$$

and P represented symbolically as

$$P = \begin{bmatrix} a & b \\ c & d \end{bmatrix}.$$

By setting all variables into the *algebraic Riccati equation* and solving for P this leads to

$$\begin{aligned} 0 &= PA + A^T P - PBQ_u^{-1}B^T P + Q_x \\ &= \begin{bmatrix} q_1^2 - b^2 & a - bc \\ a - bc & 2b + q_2^2 - c^2 \end{bmatrix}. \end{aligned} \tag{A.10}$$

For three unknown variables this can be rewritten as

$$\begin{aligned} q_1^2 - b^2 &= 0 \\ a - bc &= 0 \\ 2b + q_2^2 - c^2 &= 0, \end{aligned}$$

which leads to

$$\begin{aligned} q_1^2 &= b^2 \\ &\Rightarrow b = \pm q_1 \\ &\Longrightarrow \pm 2q_1 + q_2^2 - c^2 = 0 \\ &\Longrightarrow c = \pm\sqrt{q_2^2 \pm 2q_1}. \end{aligned}$$

Implying $c > 0$ and $q_1 > 0$ we obtain

$$P = \begin{bmatrix} q_1\sqrt{q_2^2 + 2q_1} & q_1 \\ q_1 & \sqrt{q_2^2 + 2q_1} \end{bmatrix}. \tag{A.11}$$

Finally we can compute the feedback gain matrix K:

$$K = Q_u^{-1}B^T P = \begin{bmatrix} q_1 & \sqrt{q_2^2 + 2q_1} \end{bmatrix}. \tag{A.12}$$

In our framework we use the LQR feedback controller for reference tracking, where the reference input $\mathbf{x}_r$ is known. In order to track the reference trajectory accurately we have to satisfy the following error condition $\lim_{t->T} \|\mathbf{x} - \mathbf{x}_r\| = 0$ or be minimal. For the closed loop system our control is $u = -K(\mathbf{x} - \mathbf{x}_r)$ and by setting it into Eq. (2) this leads to

$$\begin{aligned} \dot{\mathbf{x}} &= A\mathbf{x} + Bu \\ &= A\mathbf{x} + B(-K(\mathbf{x} - \mathbf{x}_r)) \\ &= A\mathbf{x} - BK\mathbf{x}_r + BK\mathbf{x} \\ &= (A - BK)\mathbf{x} + BK\mathbf{x}_r = \begin{bmatrix} 0 & 0 \\ q_1 & \sqrt{q_2^2 - 2q_1} \end{bmatrix} \begin{bmatrix} \xi \\ \dot{\xi} \end{bmatrix} \\ &\quad + \begin{bmatrix} 0 & 1 \\ q_1 & \sqrt{q_2^2 - 2q_1} \end{bmatrix} \begin{bmatrix} \xi_r \\ \dot{\xi}_r \end{bmatrix}. \end{aligned}$$

Appendix B. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.robot.2017.04.006.

References

[1] R.H. Castain, R.P. Paul, An on-line dynamic trajectory generator, Int. J. Robot. Res. 3 (1) (1984) 68–72.

[2] B. Siciliano, L. Sciavicco, L. Villani, G. Oriolo, Robotics: Modelling, Planning and Control, Springer Publishing Company, 2009.

[3] J.A. Ijspeert, J. Nakanishi, S. Schaal, Movement imitation with nonlinear dynamical systems in humanoid robots, in: Proc. 2002 IEEE Int. Conf. Robotics and Automation, 2002, pp. 1398–1403.

[4] J.A. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal, Dynamical movement primitives: Learning attractor models for motor behaviors, Neural Comput. (ISSN: 0899-7667) 25 (2) (2013) 328–373.

[5] J. Kober, K. Mülling, O. Krömer, C.H. Lampert, B. Schölkopf, J. Peters, Movement templates for learning of hitting and batting, in: Proc. 2010 IEEE Int. Conf. Robotics and Automation, 2010, pp. 1–6.

[6] B. Nemec, A. Ude, Action sequencing using dynamic movement primitives, Robotica 30 (5) (2012) 837–846.

[7] K. Ning, T. Kulvicius, M. Tamosiunaite, F. Wörgötter, A novel trajectory generation method for robot control, J. Intell. Robot. Syst. 68 (2) (2012) 165–184.

[8] T. Kulvicius, K.J. Ning, M. Tamosiunaite, F. Wörgötter, Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting, IEEE Trans. Robot. 28 (1) (2012) 145–157.

[9] S.M. Khansari-Zadeh, A. Billard, BM: An iterative method to learn stable nonlinear dynamical systems with gaussian mixture models, in: Proc. 2010 IEEE Int. Conf. Robotics and Automation, 2010, pp. 2381–2388.

[10] S.-M. Khansari-Zadeh, A. Billard, A dynamical system approach to realtime obstacle avoidance, Auton. Robots 32 (2012) 433–454.

[11] A. Paraschos, C. Daniel, J. Peters, G. Neumann, Probabilistic movement primitives, in: C. Burges, L. Bottou, M. Welling, Z. Ghahramani, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 26, Curran Associates, Inc., 2013, pp. 2616–2624.

[12] J.R. Medina, D. Lee, S. Hirche, Risk-sensitive optimal feedback control for haptic assistance, in: Proc. IEEE Int. Conf. Robotics and Automation, 2012.

[13] S. Calinon, D. Bruno, D.G. Caldwell, A task-parameterized probabilistic model with minimal intervention control, in: Proc. IEEE Int. Conf. Robotics and Automation, 2014, pp. 3339–3344.

[14] U.W. Hochstrasser, Orthogonal polynomials, in: Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, New York, Dover, 1972.

[15] M. Steinbach, H. Bock, R. Longman, Time-optimal extension and retraction of robots: Numerical analysis of the switching structure, J. Optim. Theory Appl. 84 (3) (1995) 589–616.

[16] M.C. Steinbach, H.G. Bock, G.V. Kostin, R.W. Longman, Mathematical optimization in robotics: towards automated high-speed motion planning, Surv. Math. Ind. 7 (4) (1997) 303–340.

[17] M.L. Felis, K. Mombaur, H. Kadone, A. Berthoz, Modeling and identification of emotional aspects of locomotion, J. Comput. Sci. 4 (4) (2013) 255–261.

[18] K.J. Aström, R.M. Murray, Feedback Systems: An Introduction for Scientists and Engineers, Princeton University Press, 2008.

[19] A. Ratajczak, Trajectory reproduction and trajectory tracking problem for the nonholonomic systems, Bull. Pol. Acad. Sci. Tech. Sci. 64 (1) (2016) 63–70.

[20] S.M. Khansari-Zadeh, A. Billard, Learning stable non-linear dynamical systems with gaussian mixture models, IEEE Trans. Robot. 27 (2011) 943–957.

[21] M. Mitchell, An Introduction to Genetic Algorithms (Complex Adaptive Systems), A Bradford Book, ISBN: 9780262631853, 1998.

[22] R.W. Holsapple, R. Venkataraman, D. Doman, New, fast numerical method for solving two-point boundary-value problems, J. Guid. Control Dyn. 27 (2) (2004) 301–304.

[23] M. Tamosiunaite, B. Nemec, A. Ude, F. Wörgötter, Learning to pour combining goal and shape learning for dynamic movement primitives, Robot. Auton. Syst. 59 (11) (2011) 910–922.

[24] A. Nemec, R. Vuga, A. Ude, Efficient sensorimotor learning from multiple demonstrations, Adv. Robot. 27 (13) (2013) 1023–1031.

[25] W.S. Levine, The control handbook, in: The Electrical Engineering Handbook Series, CRC Press New York, Boca Raton (Fl.), ISBN: 0-8493-8570-9, 1996.

[26] J.-B. He, Q.-G. Wang, T.-H. Lee, PI/PID controller tuning via LQR approach, in: Decision and Control, 1998. Proceedings of the 37th IEEE Conference on, vol. 1, 1998, pp. 1177–1182.

[27] N. Kumari, A.N. Jha, Automatic generation control using LQR based PI controller for multi area interconnected power system, Adv. Electron. Electr. Eng. 4 (2) (2014).

[28] A. Jose, C. Augustine, S.M. Malola, K. Chacko, et al., Performance study of PID controller and LQR technique for inverted pendulum, World J. Eng. Technol. 3 (02) (2015) 76.
[29] I.D. Cowling, W.J. F., A.K. Cooke, Optimal trajectory planning and LQR control for a quadrotor UAV, in: UKACC International Conference on Control, 2006.
[30] M. Egerstedt, C.F. Martin, Optimal trajectory planning and smoothing splines, Automatica 37 (7) (2001) 1057–1064.
[31] M. Vukobratović, B. Borovac, Zero-Moment Point - Thirty Five Years of its Life, Int. J. Hum. Rob. 1 (1) (2005) 157–173.
[32] N. Hogan, Impedance control: An approach to manipulation, in: American Control Conference, 1984, 1984, pp. 304–313.
[33] C. Ott, R. Mukherjee, Y. Nakamura, Unified impedance and admittance control, in: Proc. 2010 IEEE Int. Conf. Robotics and Automation, 2010, pp. 554–561.

Sebastian Herzog studied computational neuroscience and mathematics at the University of Göttingen, Germany. His current research interests are robotics, artificial intelligence, machine learning, nonlinear dynamics, self-organization, computational fluid dynamics, quantum mechanics, information theory and data assimilation.

Florentin Wörgötter has studied biology and mathematics at the University of Düsseldorf, Germany. He received the Ph.D. degree for work on the visual cortex from the University of Essen, Germany, in 1988. From 1988 to 1990, he was engaged in computational studies with the California Institute of Technology, Pasadena, CA, USA. Between 1990 and 2000, he was a Researcher at the University of Bochum, Germany, where he was investigating the experimental and computational neuroscience of the visual system. From 2000 to 2005, he was a Professor of computational neuroscience with the Psychology Department, University of Stirling, U.K., where his interests strongly turned towards Learning in Neurons. Since July 2005, he has been the Head of the Computational Neuroscience Department at the Bernstein Center for Computational Neuroscience, Inst. Physics 3, University of Göttingen, Germany. His current research interests include information processing in closed-loop perceptionaction systems, sensory processing, motor control, and learning/plasticity, which are tested in different robotic implementations.

Tomas Kulvicius received his Ph.D. degree in Computer Science (2010) from the University of Göttingen, Germany. In his Ph.D. thesis he investigated development of receptive fields in closed loop learning systems. From 2010 to 2015, he was a Researcher at the University of Göttingen where he worked on trajectory generation and motion control for robotic manipulators. From 2015 to 2017, he was appointed as an Assistant Professor at the Centre for BioRobotics, University of Southern Denmark. Currently he is a Research Assistant at the University of Göttingen, Germany. His research interests include modelling of closed-loop behavioural systems, robotics, artificial intelligence, machine learning algorithms, movement generation and trajectory planning.