Accurate Position and Velocity Control for Trajectories Based on Dynamic Movement Primitives

KeJun Ning, Tomas Kulvicius, Minija Tamosiunaite, and Florentin Wörgötter

Abstract—**This paper presents a novel method for trajectory generation based on dynamic movement primitives (DMPs) treated from a control theoretical perspective. We extended the key ideas from the original DMP formalism by introducing a velocity convergence mechanism in the reformulated system. Theoretical proof is given to guarantee its validity. The new method can deal with complex paths as a whole. Based on this, we can generate smooth trajectories with automatically generated transition zones, satisfy position- and velocity boundary conditions at start and endpoint with high precision, and support multiple via-point applications. Theoretic proof of this method and experiments are presented.**

I. Introduction

FOR all applications of any robotic system, we need a trajectory generating technology to create a time history of relevant variables in state space (joint space or task space). Thus, this is still a fundamental and important research field in robotics.

A common method for joint space trajectory control uses lower order polynomials to provide interpolating points at servo rates between user- specified via-positions [1,2]. Specific boundary conditions can be imposed onto several lower order polynomials to obtain continuous trajectories in joint space. On the other hand, by planning a trajectory in task space, we can completely specify the desired trajectory of the end-effector. A popular approach is to utilize some simple curves (e.g., lines, arcs or parabolas) to define the whole path in task space, depending on the assigned via-points. This way one can also introduce zones in which an interpolation between the two adjacent segments can be performed [3].

With the advent of humanoid and anthropomorphic robots, many new challenges have surfaced [4-7]. The framework of Dynamic Movement Primitives (DMPs), which will serve as the basis for this paper, is an example of this. The DMP formalism was presented by Ijspeert et al [4-6], and their investigations on humanoid robots have led to numerous applications [8-15]. The original DMP algorithm consists of two sets of differential equations; a canonical system, and a transformation system [4-6]. The canonical system is used to represent the phase of the motor process. The transformation system is a basic point attractive system, utilized to generate the desired movement [4-6].

While interesting, the basic DMP formulation has some limitations, which restricts its usefulness for trajectory generation applications. For example, limited by its formulation and structure, the original DMP can be used neither to directly incorporate a target velocity [15] nor a via-point. It always asymptotically converges to the final point and the speed of approaching the target is zero. Kober et al [15] made a modification of the original DMP formulation in order to implement striking (e.g., batting a ball) movements in the middle of a trajectory.

We were inspired by the original DMP applications [4-6] and show here a novel trajectory generating technology based on the key ideas from DMPs. The goal is to present a DMP based trajectory generating technology that provides accurate control over the trajectory including its start- and endpoints in position and velocity space.

The remainder of this article is organized as follows. In Section 2, we present our new solution for trajectory generation. Case studies and experiments are provided in Section 3. Finally, conclusions and application potential are summarized in Section 4.

II. Trajectory Generator Based on DMP

The following requirements are fundamental for a complete and useful trajectory generation technology: accurate adherence to boundary conditions, global smoothness and accuracy. Furthermore, for possible industrial applications, we also need to incorporate the required velocity profile into the trajectory.

A. Architecture of Our Trajectory Generator

Fig. 1 shows the architecture of our novel DMP based trajectory generator, which consists of four key modules. The Second-Order System module provides the trajectory $y(t)$ and $\dot{y}(t)$. The outputs from the Boundary Function

Manuscript received Sep. 13, 2010. This work was supported by PACO_PLUS from the European Commission and by the BMBF BFNT project 3a of the University Göttingen.

KeJun Ning was a Post Doctoral researcher in the Bernstein Center for Computational Neuroscience, Inst. of Physics III, University of Göttingen, 37077 Göttingen, Germany. Now he is with the Corporate Technology, Siemens Ltd., China, 100102, Beijing, China (e-mail: nkj@sjtu.org , nkjchina@hotmail.com).

Tomas Kulvicius is currently a Post Doctoral researcher in Bernstein Center for Computational Neuroscience, Inst. of Physics III, University of Göttingen, 37077 Göttingen, Germany (e-mail: tomas@physik3.gwdg.de).

Minija Tamosiunaite is with the Department of Informatics, Vytautas Magnus University, Vileikos 8, LT-44404 Kaunas, Lithuania (e-mail: m.tamosiunaite@if.vdu.lt).

Florentin Wörgötter is with the Bernstein Center for Computational Neuroscience, Inst. of Physics III, University of Göttingen, 37077 Göttingen, Germany (e-mail: worgott@physik3.gwdg.de).

Generator $r(t)$, and Modulation Function $f(t)$, are fed into the Second-Order System. The Boundary Function Generator imposes the boundary conditions onto the system. The Modulation Function, consisting of a Gaussian Kernel Based Approximating Function and a Suppressing Window Function, will affect the Second-Order System's response as a forcing input. The Gaussian Kernel Based Approximating Function contains a weight vector, and these weights need to be trained in order to code the information brought from the sample trajectory $y^*(t)$, provided by the Sample Trajectory Generator. After training, the Modulation Function module will contain the main information obtained from $y^*(t)$, and the sample trajectory will not be used anymore.

The architecture shown in Fig. 1 is for one Degree-of-Freedom (DOF). We can employ *N* copies of this architecture in parallel for an N-DOF application.

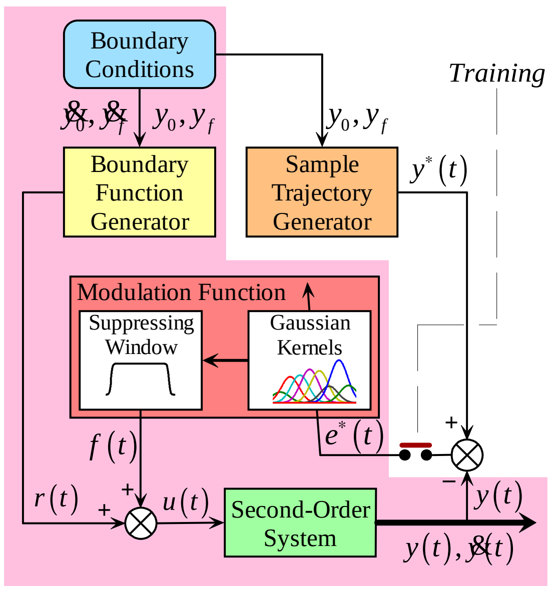


Fig. 1. The architecture of our DMP based trajectory generator. After training, only the pink-shaded components will be used.

B. Second-Order System Module

In the original DMP formalism and the following studies [4-6, 8-15], parameters have not been directly related to the physical meaning of a second-order system. In the current paper, we adopt a standard formulation to make the system easier to understand and analyze from a control theory perspective.

Fig. 2 shows the block diagram of the Second-Order System module employed in this paper, expressed in Laplace space. Its forward channel is

$$G(s) = \frac{\omega_n^2}{s(s+2\zeta\omega_n)}, \tag{1}$$

and its feedback channel is $H(s)=1$. Let $U(s)=R(s)+F(s)$ and $Y(s)$ denote the input and output respectively, then the transfer function is as follows,

$$\frac{Y(s)}{U(s)} = \frac{\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}. \tag{2}$$

This is a standard second-order system, where the constants ζ and ω_n are the damping coefficient and the natural undamped frequency of the system, respectively. When $\zeta=1$, the system has a double pole at $-\omega_n$, and this results in a critically damped response [16].

The model shown in Eq. (2) is well-studied in control theory. Its steady-state error can be calculated by the final-value theorem [16], as follows,

$$e_{ss} = \lim_{t\to\infty} e(t) = \lim_{s\to 0} \frac{sU(s)}{1+G(s)}. \tag{3}$$

Furthermore, we can express the error's change rate as

$$\dot{e}_{ss} = \lim_{t\to\infty} \dot{e}(t) = \lim_{s\to 0} \frac{s^2U(s)}{1+G(s)}. \tag{4}$$

Based on Eqs. (3-4), we can prove the asymptotic convergence of Ijspeert's original DMPs. Eqs. (3-4) also provide the foundation for including boundary conditions in the architecture shown in Fig. 1. We will discuss this issue in the following sections.

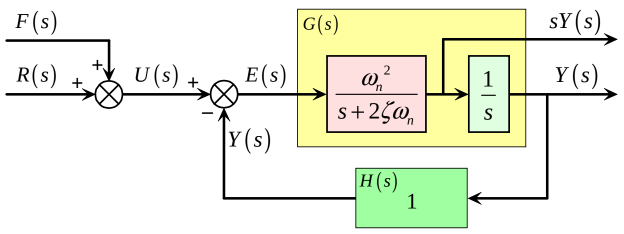


Fig. 2. Block diagrams of the employed Second-Order System in Laplace space.

C. Boundary Function Generator

For a useful trajectory generator, we have to take into account boundary conditions for position and velocity. As mentioned above, polynomials are normally employed to remove discontinuities (velocity and acceleration) between adjacent path segments. We use a third-order polynomial and determine the coefficients, given by y_0, y_f and $\dot{y}_0, \dot{y}_f$ as the positions and velocities at the start and end points respectively.

However, we cannot employ the third-order polynomial alone to construct the Boundary Function Generator shown in Fig. 1. As we know, a type-1 system can not follow a parabolic or a higher order function, because the steady-state error is infinite [16]. This conclusion can be derived from Eq. (3). If $U(t)$ is a ramp function, from Eqs. (3-4) we obtain

$$e_{ss} = V / K_v, \tag{5}$$

and

$$\dot{e}_{ss} = 0, \tag{6}$$

where V is the slope coefficient of $U(t)$, and $K_v = \omega_n / 2\zeta$. Eq. (6) shows that the slope coefficient of the Second-Order System's response will approach that of the input ramp function in the end. Even though Eqs. (5-6) hold only for the steady-state, we can design our system based on this principle with controllable precision. In fact, Eq. (6) is the proof for approaching the assigned velocity and Eq. (5) presents the offset that we can use to reduce the position error. Thus, we can utilize a third-order polynomial extended by a line segment to construct the reference signal $r(t)$. Fig. 3 shows such a case.

Let us denote t_m as the junction moment (the time point where the polynomial segment and the line segment are joined), then the reference signal is defined as follows,

$$r(t) = \begin{cases} a_0 + a_1 t + a_2 t^2 + a_3 t^3 & 0 \le t \le t_m \\ y_e - (t_f - t)\dot{y}_f & t_m < t \le t_f \end{cases} \tag{7}$$

where,

$$y_e = y_f + \delta_e \tag{8}$$

and $\delta_e = (2\zeta / \omega_n)\dot{y}_f$.

In Eq. (7), the polynomial segment $[0, t_m]$ is employed to connect the boundary conditions imposed on the start point, and the linear segment $(t_m, t_f]$ works as a ramp input that provides a convergence reference for the Second-Order System, with both, position and velocity, constraints. The value t_m depends on the temporal properties of the Second-Order System module.

The Boundary Function Generator will guarantee the required system's performance at start and end regions.

D. Modulation Function

For most application tasks, we need to control a robot to move along an assigned path. The Modulation Function $f(t)$ shown in Fig. 1 is used to "force" the generated response to follow the assigned trajectory (i.e., the sample trajectory, see later) at the middle of the path.

As shown in Fig. 1, the input to the Second-Order System is

$$u(t) = r(t) + f(t). \tag{9}$$

The variable $f(t)$ works as the forcing input to this system and we use it to counteract unwanted effects brought by the original reference signal $r(t)$ from the Boundary Function Generator and to reshape the system's output in order to force it to follow the desired path.

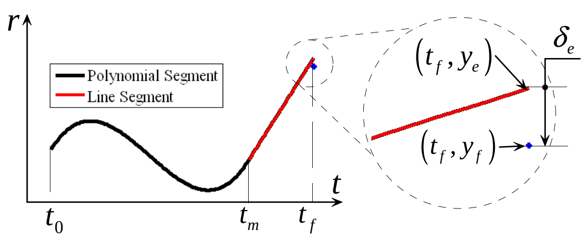


Fig. 3. A solution for building the Boundary Function Generator. A third-order polynomial extended by a line segment can be employed here to construct the reference signal $r(t)$ for the DMP based trajectory generator.

1) Gaussian Kernels Based Approximating Function

The original DMP introduces nonlinear control based on learned feed-forward controllers [4-6]. A nonlinear function based on a group of Gaussian kernels can be trained to smoothly approximate a given sample curve. We utilize this idea to build the Modulation Function for our system.

Let us define the Modulation Function with M kernels,

$$f(t) = \frac{\boldsymbol{\psi}^{\mathbf{T}}\mathbf{W}}{\sum_{i=1}^{M}\psi_i} v(t), \tag{10}$$

where $\mathbf{W} = [w_1, w_2, \cdots, w_i, \cdots, w_M]^T$, w_i is the weight of the Gaussian kernel i and $\boldsymbol{\psi} = [\psi_1, \psi_2, \cdots, \psi_i, \cdots, \psi_M]^T$, with

$$\psi_i = \exp\left(-h_i\left(\frac{t}{T} - c_i\right)^2\right), \tag{11}$$

where T is the time length of the whole process, t is the time, and $v(t)$ is a suppressing window function (see later). Constants c_i and h_i are centers and widths of the Gaussian kernels i, evenly distributed over the range $(0,1)$.

2) Weight Learning

The weight vector $\mathbf{W}$ is trained to match the sample trajectory $y^*(t)$ generated by the Sample Trajectory Generator shown in Fig. 1. We have used a simple and practical learning rule to train the weight vector $\mathbf{W}$, formalized by the following equation:

$$\Delta w_i = \gamma\left(y^*(k) - y(k)\right)v(k) \tag{12}$$

where, γ is the learning rate, $k = c_i T$ defines the center of the *i-th* Gaussian kernel within the time period $[0, T]$. Here, k serves as a phase variable to anchor the Gaussian kernels to the time period T. Regression methods like in [5, 6] can be used instead.

3) Suppressing Window Function

The Suppressing Window Function $v(t)$ in Eq. (10), serves as an "enable" term inside the Modulation Function shown in Fig. 1. It controls kernel influence by suppressing their action near start and endpoints thereby assuring accuracy of the process. When the system is in those regions, we should let the Boundary Function Generator's output $r(t)$ drive the Second-Order System and Eqs. (5-6) will guarantee that the assigned boundary conditions will be successfully reached. We define the Suppressing Window Function by,

$$v(t) = \frac{1}{1+e^{-l_1(t-C_1)}} \times \frac{1}{1+e^{l_2(t-C_2)}}. \tag{13}$$

Eq. (13) is a double-sigmoid function. It is continuous and differentiable and it has a pair of horizontal asymptotes as $t \to \pm\infty$. In Eq. (13), l_1, l_2 are used to set the slopes, and C_1, C_2 to set the inflection points. With suitable parameters' setting, $v(t)$ approaches zero at the start- and end time as close as possible in order to obtain $f(t)=0$. This way, $r(t)$ totally governs the Second-Order System at start- and endpoints and a more precise result is achieved. This kind of transition region is similar to the popular "zone solution" [3], but implicitly described by the parameters of Eq. (13).

E. Sample Trajectory Generator

Planning a trajectory by this method is done in two steps. In the first step the sample trajectory provides geometric path constraints and the speed profile information along the trajectory. Note, when designing the sample trajectory sequence, we do not need to care about the possible initial and final velocities imposed by the boundary conditions (see the signal flow shown in Fig. 1), which simplifies this step dramatically. Also, the sample $\mathbf{y}^*$ is only used during the training process. It is a one-shot process. After training, $f(t)$ will contain all the information required to obtain the resulting trajectory.

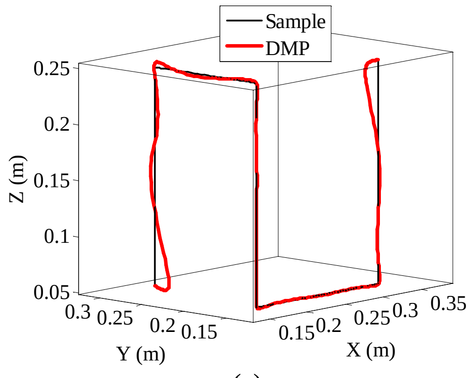
In the second step the boundary conditions are imposed and the final smooth result is obtained through the interaction of $f(t)$ and $r(t)$.

Now we describe how to define for step 1 the Sample Trajectory Generator. It provides a sample trajectory sequence $\mathbf{y}^*(t)$:

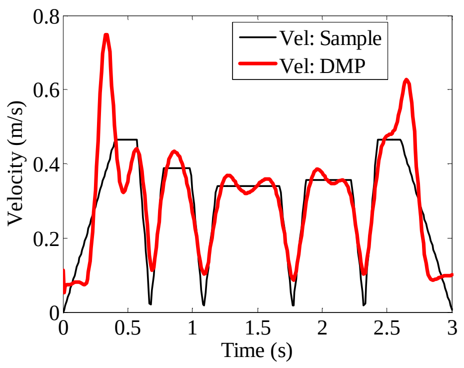
$$\mathbf{y}^*(t) = \left[y^*(0), y^*(\tau), \cdots, y^*(k\tau), \cdots, y^*(L\tau)\right]^T, \tag{14}$$

where, τ is the sampling time period, and $L\tau = T$ is the total desired time to complete the path. We can acquire it by imitation (teaching and recording), or generate it by a simple path planner routine. Without loss of generality, in this paper, we focus on the latter.

To generate a velocity profile for a sample trajectory, the Linear Segments with Parabolic Blends (LSPB) [17] method can be employed. For applications with many via-points, the easiest solution is to set up independent LSPB trajectories between adjacent via points and then to connect them one by one to form the whole sample trajectory sequence shown as Eq. (14). In fact, the discontinuous variation in the acceleration which comes from the LSPB will be filtered out by the trained Modulation Function. In the following section, we will give several application examples.



(a)



(b)

Fig. 4. A multi-segment trajectory generating case, with two-end none-zero velocity constraints. (a) Position plot, sample and the generated position trajectory; (b) Velocity plot, sample and the generated position trajectory.

III. Application Cases

Our DMP based trajectory generator has the potential to deal with very complicated application cases.

Fig. 4 shows an application case in Cartesian space. It is a 3D multi-segment path. Velocity vectors are assigned to start- and end-points. Three trajectory generators work in parallel to yield X, Y and Z components. The sample trajectory was planned by the LSPB method, and without velocity constraint consideration. One can see that our method generates a trajectory that satisfies the assigned boundary condition. As shown in Fig. 4 (b), the generated result "slows down" and closely passes these sharp corners (via-points) autonomously. The method presented in this paper generates smooth transition on its own.

In the case shown in Fig. 5, a manipulator needs to move along a rectangle path. The difference between cases shown

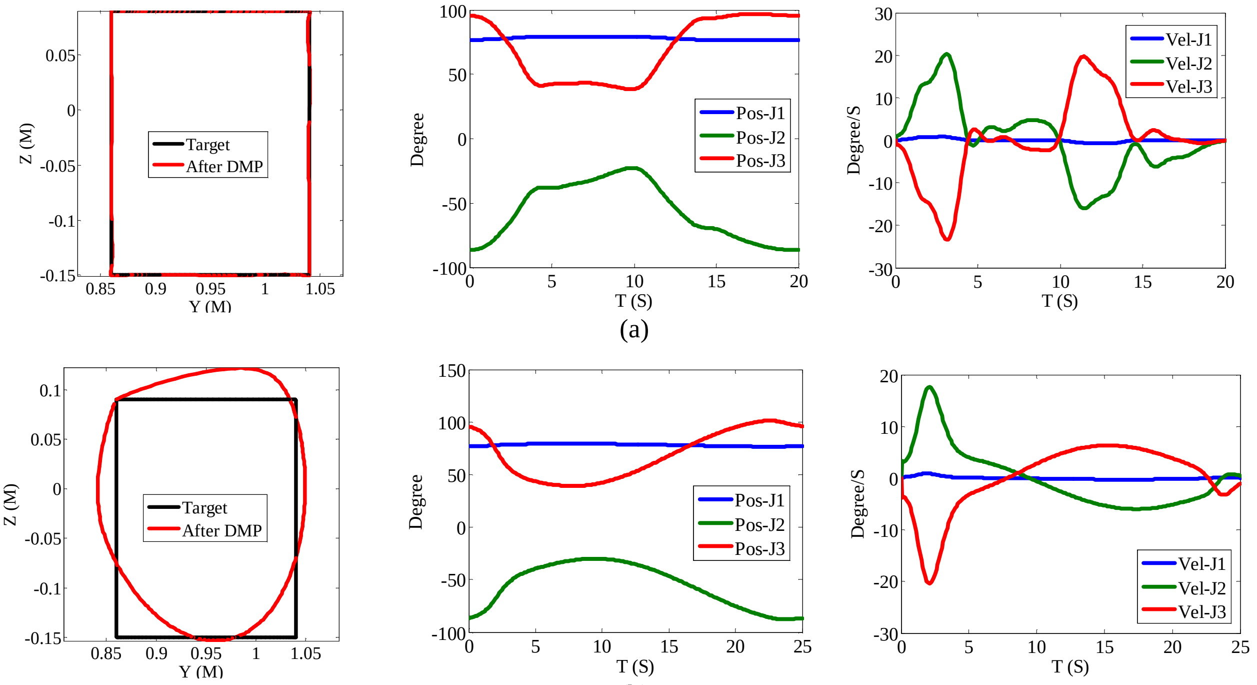


Fig. 5. A multi-segment trajectory generating and comparison case, with two-end none-zero velocity constraints. (a) More and narrower Gaussian kernels: $M = 40$ and $h = 400$; (b) Fewer and wider Gaussian kernels: $M = 4$, and $h = 10$.

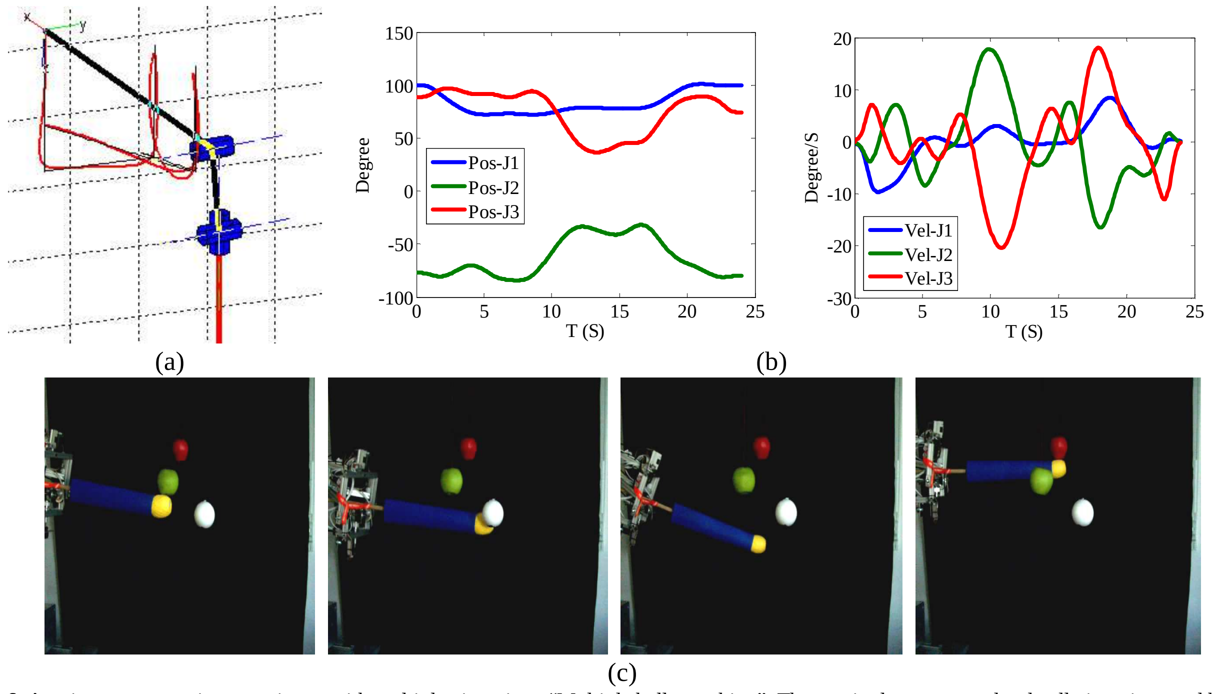


Fig. 6. A trajectory generating experiment with multiple via-points: “Multiple balls touching”. The manipulator passes closely all via-points, and behaves smoothly and naturally. (a) Simulation model. (b) Position and velocity plots of the joints. (c) Snapshots during experiment.

in rows (a) and (b) of Fig. 5 is that they own different Gaussian kernel parameters. If the number of Gaussian kernels, M, in Eq. (10) is bigger, the method can approximate a more complicated trajectory more exactly. A smaller kernel number will produce a smoother more natural trajectory, if intermediate trajectory accuracy is not required.

Fig. 6 shows a trajectory generating experiment with multiple via-points: “Multiple balls touching”. The manipulator passes closely all via-points, and behaves smoothly and naturally. A movie of several basic experiments is shown at [18]. For these cases, the Robotics Toolbox V7.1 [19] was employed to aid our analysis.

All methods presented in this paper have been implemented on a simple robotic manipulator platform (Neuro-Robotics, Sussex) in our laboratory, where not only the key algorithm shown in this paper, but also more low-level driving and interfacing programs for the embedded control system of this platform have been implemented.

IV. Conclusions and Application Potential

In this paper, we presented a novel trajectory generator based on DMPs, formulated from a control theory and robotics viewpoints. Our work is quite different as compared to the original DMPs and the existing extensions [4-6, 8-15].

The technology presented in this paper has some characteristics:

- The generated trajectory is smooth and can be quite complex using as many via-points as desired;
- Transition zones will be created automatically at both ends, different from the traditional ways by sectionalizing and connecting by spline/polynomial [1];
- A required speed profile can be imposed on the trajectory;
- A built-in filter (when ω_n is low, the second-order system indeed acts as a low pass filter) makes the trajectory output being easier to follow by motion control system, as long as the bandwidth of motion control system is higher than the one of our system [16]. The traditional spline-based solution does not have this property [1];
- The path planning stage of a practical task can be simplified greatly, because this design introduces boundary conditions in an independent way.

The often pursued combination of DMPs with imitation learning is also possible with this framework. For such an application, this method has the potential to allow for some dynamic tasks as mentioned above. For instance, a humanoid robot can imitate the movement profile and produce different kinds of end velocities.

In summary, we believe that this novel trajectory generating technology exhibits great flexibility and applicability. Thus, we hope that our work presented in this paper will stimulate further DMP related research and development, and that this novel trajectory generating technology can be an alternative and widely employed not only in humanoid robots, but also in industry.

References

[1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, “Robotics: Modelling, Planning and Control,” Springer, 2009.

[2] R. H. Castain, and R. P. Paul, “An On-Line Dynamic Trajectory Generator,” *Int. J. Robotics Research*, vol. 3, no. 1, pp. 68-72, 1984.

[3] M. Nystrom, and M. Norrlof,. “Path generation for industrial robots,” Technical Report LiTH-ISY-R-2529, Department of Electrical Engineering, Linkoping University, 2003.

[4] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert, “Learning Movement Primitives,” in *Proc. Int. Symp. Robotics Research*, Siena, pp. 561-572, 2003.

[5] A. J. Ijspeert, J. Nakanishi, and S. Schaal, “Movement imitation with nonlinear dynamical systems in humanoid robots,” in Proc. IEEE Int. Conf. Robotics and Automation, Washington, DC, pp. 1398-1403, 2002.

[6] S. Schaal, P. Mohajerian, and A., J. Ijspeert, “Dynamics systems vs. optimal control - a unifying view,” *Progress in Brain Research*, vol. 165, no.1, pp. 425-445, 2007.

[7] C. G. Atkeson, J.G. Hale, F. Pollick, M. Riley, S. Kotosaka, S. Schaal, T. Shibata, G. Tevatia, A. Ude, S. Vijayakumar, E. Kawato, and M. Kawato, “Using humanoid robots to study human behavior,” *IEEE Intelligent Systems*, vol. 15, no. 4. pp. 46-56, 2000.

[8] S. Strachan, R. Murray-Smith, I. Oakley, and J. Angesleva, “Dynamic primitives for gestural interaction,” in *Proc. 6th Int. Symp. Mobile Human-Computer Interaction*, Glasgow, pp. 325-330, 2004.

[9] A. Ijspeert, J. Nakanishi, and S. Schaal, “Learning attractor landscapes for learning motor primitives,” Advances in Neural Information Processing Systems, S. Becker, S. Thrun, and K. Obermayer, Eds., vol. 15, pp. 1547-1554, 2003.

[10] D. H. Park, H. Hoffmann, P. Pastor, and S. Schaal, “Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields,” in *Proc. 8th IEEE-RAS Int. Conf. Humanoid Robots*, Daejeon, pp. 91-98, 2008.

[11] H. Hoffmann, P. Pastor, D. H. Park, and S. Schaal, “Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance,” in *Proc. IEEE Int. Conf. Robotics and Automation*, Kobe, pp. 2587-2592, 2009.

[12] A. Gams, and A. Ude, “Generalization of example movements with dynamic systems,” in *Proc. 9th IEEE-RAS Int. Conf. Humanoid Robots*, Paris, pp. 28-33, 2009.

[13] D. Pongas, A. Billard and S. Schaal, “Rapid synchronization and accurate phase-locking of rhythmic motor primitives,” in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Edmonton, pp. 2911-2916, 2005.

[14] J. Kober, and J. Peters, “Learning new basic Movements for Robotics,” in *Proc. Autonome Mobile Systeme*, Karlsruhe, pp. 105-112, 2009.

[15] J. Kober, K. Mulling, O. Kromer, C. H. Lampert, B. Scholkopf, and J. Peters, “Movement Templates for Learning of Hitting and Batting,” in *Proc. ICRA IEEE Int. Conf. Robotics and Automation*, Anchorage, pp. 853-858, 2010.

[16] R. C. Dorf, and R. H. Bishop, “Modern Control Systems,” 8th Edition, Addison-Wesley, 1998.

[17] M. W. Spong, S. Hutchinson, and M. Vidyasagar, “Robot Dynamics and Control,” 2nd Edition, John Wiley & Sons, 2004.

[18] K. Ning, and F. Wörgötter, “A Novel Trajectory Generator Based on Dynamic Movement Primitives. Simple Application - 1,” http://www.youtube.com/watch?v=N84xD1H-G3A

[19] P. I. Corke, “A robotics toolbox for MATLAB,” *IEEE Robotics and Automation Magazine*, vol. 3, pp. 24-32, 1996.