Compliant dynamic movement primitives

Jaka Kovše, Adam Hrastnik

Faculty of Electrical Engineering, University of Ljubljana, Tržaška c. 25, 1000 Ljubljana
E- mail: jk4684@student.uni-lj.si
E- mail: ah1223@student.uni-lj.si

Abstract

This article introduces mathematical equations for formalization of dynamic movement primitives and shows their use in simulation environment. Different parameters for generating DMPs are described and their effects on adequacy of generated DMPs are shown. Basics of impedance control and torque primitives which together with DMPs constitute compliant dynamic movement primitive are introduced.

Keywords: CDMP, robot, impedance control, compliant

1 Introduction

As robots become more common throughout the industry [5], so do jobs involving human-robot interaction. Additionally, robots are becoming more frequent in household applications, by partaking assistive roles. Due to these reasons, it is essential to provide maximum safety in collaborative and cooperative tasks. Increased safety will not only help with preventing injuries but will also help people accept robots as their cohabitants and/or co-workers.

In this project we explore safe robot control, by utilizing compliant dynamic movement primitives (CDMPs) [2] [3]. All experiments were conducted in simulation on a model of the Kuka LBR iiwa robot [1]. The main goal was to teach the robot using statistical generalization, to apply the appropriate torque for a given task, while remaining compliant. This ensures safety and accuracy.

2 Dynamic movement primitives

To get a better understanding of the behaviour of a nonlinear system, whether it is in a field of neuroscience, economics, or robotics, a nonlinear dynamical model of that system is needed. It has become common practice to model them with systems of coupled nonlinear differential equations. Mostly, the unexpected emergent behaviour of nonlinear systems is investigated, however a goal-directed behaviour is of equal importance and is shown in this article. Modelling that kind of behaviour is rather difficult due to parameter sensitivity of these systems and their lack of analytical predictability. A generic modelling approach to generate nonlinear differential equations to capture an observed behaviour in an attractor landscape is proposed. Its manner is to start with set of linear differential equations and transform them into desired system with prescribed attractor dynamics, using learnable autonomic forcing term [2].

2.1 Model development

Basic idea of this approach is to use a well-understood dynamical system and modulate it with nonlinear terms to achieve desired behaviour. Here, we use a damped spring system:

$$\tau\ddot{y} = \alpha_z(\beta_z(g - y) - \dot{y}) + f(x, \boldsymbol{w}), \quad (1)$$

where $\ddot{y}, \dot{y}, y$ are variables of acceleration, velocity and position which would be converted to motor commands by a controller (they cause nonlinearities in dynamics). We write it in first-order notation:

$$\tau\dot{z} = \alpha_z(\beta_z(g - y) - z) + f(x, \boldsymbol{w}) \quad (2)$$
$$\tau\dot{y} = z \quad (3)$$
$$\tau\dot{x} = -\alpha_x x, \quad (4)$$

where τ represents time constant and α_z, α_x and β_z are positive constants. Variable y describes the motion of one degree of freedom and x is called a phase variable (initially equals 1 and limits to zero in infinity). Equations (1) and (2) represent transformation system and (3) represents a canonical system (replacement of time component). There is also a forcing function $f(x, \boldsymbol{w})$ given in equation (2), which is responsible for system adequately following desired path. For that we need to define forcing term f:

$$f(x) = \frac{\sum_{i=1}^{N} \Psi_i(x) w_i}{\sum_{i=1}^{N} \Psi_i(x)} x(g - y_0) \quad (5)$$

where:

$$\Psi_i(x) = exp\left(-\frac{1}{2\sigma_i^2}(x - c_i)^2\right), \quad (6)$$

σ_i and c_i determine width and center of basis functions and y_0 is the initial state at $t = 0$. Parameters w_i are weight vectors which can be adjusted using learning algorithms to produce complex trajectories before reaching the goal.

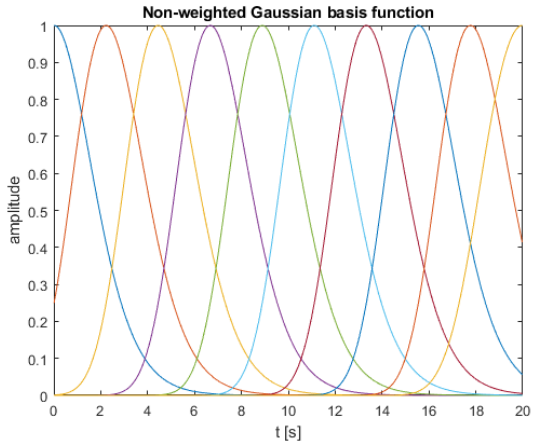

Figure 1: Non-weighted Gaussian basis functions

With use of x term in forcing function we guarantee that the contribution of the forcing term goes to $x \cong 0.13$ over time. Meaning; our system can trace very complex paths but will eventually return to its simpler point attractor dynamics and converge to the target. However, x term does not converge linearly but exponentially which presents us with a problem, because basis functions activate according to x term (activation of basis function at the start is more frequent than at the end).

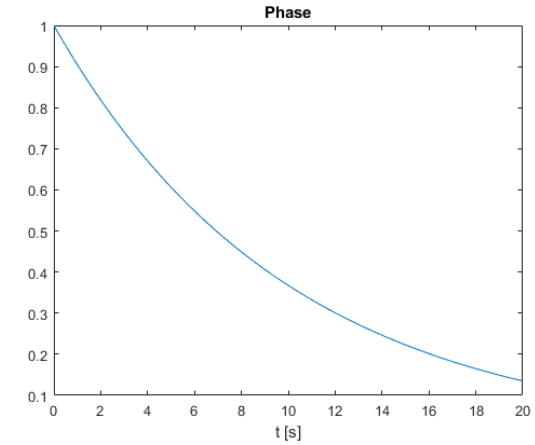

Figure 2: DMP phase

A solution is to determine variance of a basis function that is equal to number of basis functions divided by the centre of that basis function. Term $(g - y_0)$ in forcing function is responsible for scaling of computed trajectory. It scales the activation of basis functions relative to the distance to the target. It compensates if weights are too weak (new goal moved further away) or too strong (new goal closer than original one). Like spatial also temporal scaling is possible, meaning, ability to perform same path at different speeds, which is made possible with inclusion of the term τ. If τ is set between 0 and 1, the system will perform a movement slower and if τ is set higher than 1 the movement will be performed faster [3].

With described equations it is possible for a system to follow a desired path with temporal and spatial scalability, however, it is desired for the system to be able to learn a demonstrated path.

Once obtained, desired path is then differentiated twice to get acceleration trajectory (7), from which the effect of the base point attractor is removed (8):

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) \tag{7}$$

$$\boldsymbol{f_d} = \ddot{\boldsymbol{y}}_d - \alpha_y(\beta_y(g - \boldsymbol{y}) - \dot{\boldsymbol{y}}) \tag{8}$$

With the use of locally weighted regression the weights over basis functions are calculated, so that the forcing term matches the desired trajectory $\boldsymbol{f_d}$:

$$w_i = \frac{s^T \Psi_i \boldsymbol{f_d}}{s^T \Psi_i s} \tag{9}$$

where:

$$s = \begin{bmatrix} x_{t0}(g - y_0) \\ \vdots \\ x_{tN}(g - y_0) \end{bmatrix}, \Psi_i = \begin{bmatrix} \Psi_i(t_0) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \Psi_i(t_N) \end{bmatrix}$$

The path has been learned, however, the ability to satisfactory replicate it using DMPs is dependent on its complexity and number of basis functions used. From that statement we can conclude, that highly nonlinear parts of the path are more densely placed and the basis functions being used are narrower; the more linear the area of the path is, the fewer and wider basis functions being used are. Method that can determine the complexity of the path and distribution of basis functions is described in another article [4].

3 Impedance control

Compliancy is achieved, by utilizing impedance control. It allows us to determine how the robot reacts upon external forces. Compared to a PD regulator, the impedance control offers compensation for dynamic nonlinear properties of a joint, which allow each joint to act as in ideal conditions, with second-order dynamics.

3.1 Mass-spring-damper model

$$F(t) = m\ddot{x}(t) + b\dot{x}(t) + kx(t) \tag{10}$$

Mechanical impedance is described by a relatively simple mass-spring-damper model (10) [6], which is a widely used second-order model, that attempts to describe how an object reacts upon an external force. The $\ddot{x}, \dot{x}, x$ variables represent acceleration, velocity, and displacement that are the result of an external force being applied. Parameters m, b, k denote the mass, damping coefficient, and stiffness, respectively.

From a robotics standpoint, the mass represents a mass of a joint, the stiffness represents how "elastic" the joint is, while the damping coefficient controls the resistance of a joint. By lowering the stiffness and damping, we achieve compliancy. The more compliant the system, the "softer" and safer the robot is and vice versa. However, with more compliant systems, the response is also slower.

3.2 Position-based impedance regulation

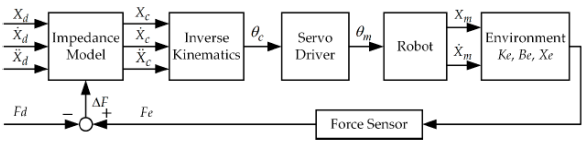

Figure 3: Position-control impedance scheme [6]

The impedance model acts as an input for the inverse kinematics and dynamics [6]. It accepts the requested motion from a trajectory and the corresponding force. Inverse kinematics and dynamics (Figure 3) then calculate the appropriate torques, which are applied to the joint actuators – usually servo motors. Force is then received from the environment through the force sensors, located on the robot, which then feed it back into the impedance model, completing the loop.

4 Encoding a trajectory into DMPs

To familiarize ourselves with DMPs, we were tasked with recording and encoding a trajectory into DMPs and then back. First, we enabled low stiffness to be able to move the robot by hand. Then we moved it to make a trajectory and recorded rotations in joints. That data was used in the encoding process. We also had to specify number of basis functions to be used. Below (figure 4), original and DMP-generated (we used ten basis functions) trajectories for a single joint are shown.

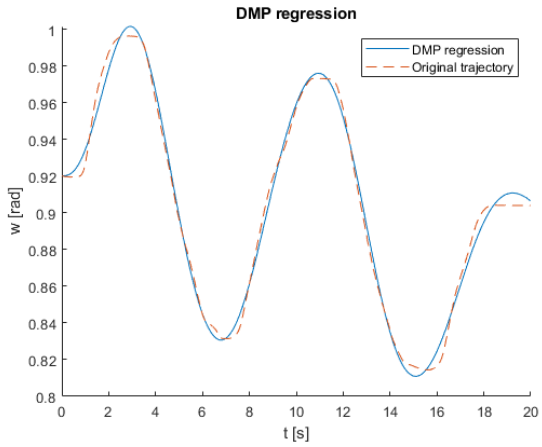

Figure 4: DMP transformation of a single joint

5 Transformational error of DMPs

At its core, a DMP transformation is a lossy or an irreversible compression. It uses inexact approximation to achieve a high compression ratio. Therefore, once the motion is decoded from DMPs, it contains an error from its original form.

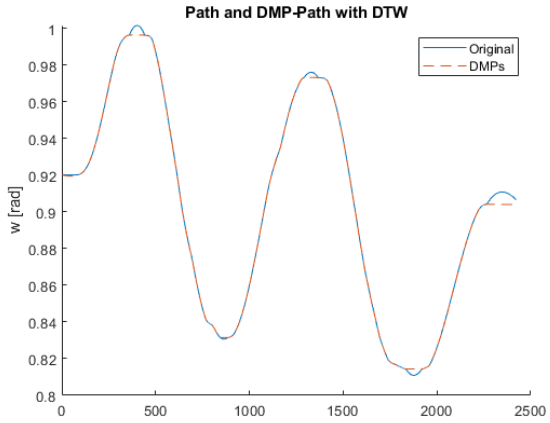

Figure 5: Transformation corrected with dynamic time warping

To properly calculate the error, we utilized dynamic time warping, to remove any phase shift or scaling. The error is calculated as a root mean square of the original and the decoded path.

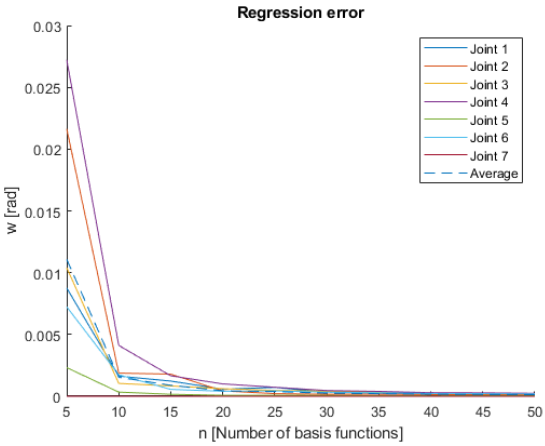

Figure 6: Transformational error, based on the number of basis functions

As described, the higher number of basis functions we use, the better the approximation is. To see just how much the number of basis functions affects the decoding error, we compared the error at different numbers of basis functions (Figure 1). From the results, we can gather that after around 30 basis functions, we start getting diminishing returns, based on the computation time.

6 Torque primitives

While very similar to DMPs, torque primitives (TPs) represent an encoded torque, instead of movement. The main difference is in that the movement is represented as second order system and then encoded with quadratic regression, whereas torque primitives are directly

encoded with quadratic regression. The initial idea behind using TPs, was to utilize them in statistical generalization, to teach a robot the necessary torques for a given load. Their usage is necessary, so the trajectory is described with as little information as possible – which accelerates learning.

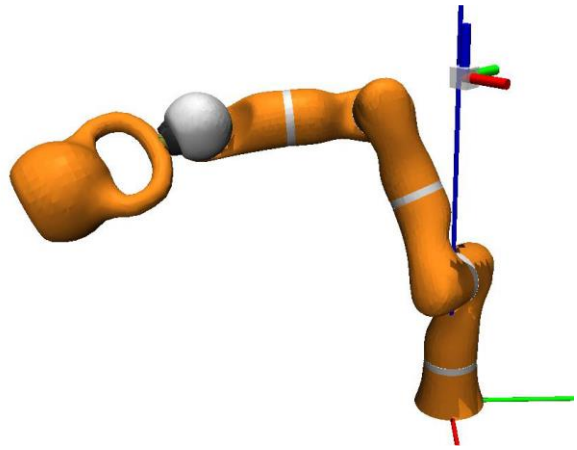

Figure 7: Kuka LBR iiwa [1] simulation with an attached 1 kg load

Ordinarily, heavier loads on compliant systems require higher torques, for the end-effector to follow the trajectory properly. Producing only required torques would insure, that the trajectory would be followed accurately, while maintaining optimal safety; meaning, if an impact would occur, it would be minimal.

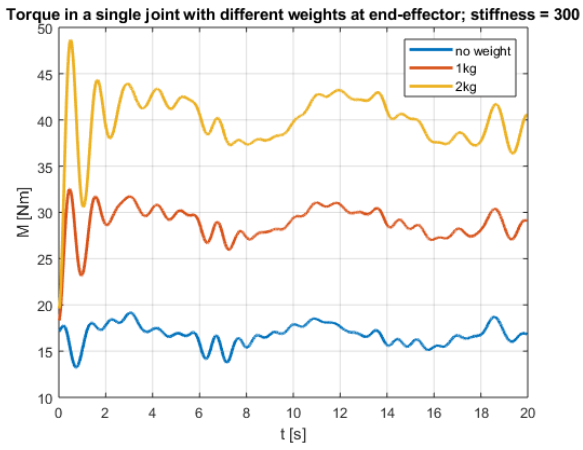

Figure 8: Torques at unloaded and different loaded conditions

To teach the robot, to apply appropriate torques on different loads, we must first measure torques in unloaded conditions, as well as in different loaded conditions. Collected data will serve as a training set for statistical generalization.

7 Conclusion

Due to the recent COVID-19 pandemic, we were forced to move our physical exercises to a simulation. This caused quite a few issues while developing the project. The issues impacted our progress considerably, which caused a noticeable lag and a lot of spent time on debugging. Because of this reason, we did not quite reach the goal of applying appropriate torque reinforcement learning to the system. Nevertheless, we were quite close, as we managed to correctly encode torques as torque primitives and therefore left out only the statistical generalization part.

If we were to have additional time for working on this project, we would firstly implement the beforementioned statistical generalization. Additionally, we would compare it against another learning strategy, such as reinforcement learning [7], to determine the most efficient learning method.

References

[1] Kuka LBR iiwa, https://www.kuka.com/en-de/products/robot-systems/industrial-robots/lbr-iiwa, accessed 16.5.2020

[2] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, S. Schaal: Dynamical Movement Primitives: Learning Attractor Models for Motor Behaviors, Massachusetts Institute of Technology, 2013

[3] Studywolf, Dynamic motion primitives the basics, https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/, accessed 16.5.2020

[4] Stefan Schaal: Dynamic Movement Primitives- A Framework for Motor Control in Humans and Humanoid Robotics, University of South Carolina, Los Angeles, 2002.

[5] International federation of Robotics press conference, 2016

[6] Chao Li, Zhi Zhang, Guihua Xia, Xinru Xie and Qidan Zhu: Efficient Force Control Learning System for Industrial Robots Based on Variable Impedance Control, 2018

[7] P. Kormushev, S. Calinon and D. G. Caldwell, "Robot motor skill coordination with EM-based Reinforcement Learning", 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taipei, 2010