Boston University
College of Engineering

Master Thesis Proposal

Imitation Learning From Human Motions With Dynamic Movement Primitives

by

Haoying Zhou

Thesis Advisor/First Reader: **Calin Belta**
Second Reader: **Sean Andersson**
Third Reader: **Roberto Tron**

June, 2019

Contents

Chapter 1

Problem Formalization

1.1 Problem Definition

My idea is to imitate some actions given some data sets extracted from real human motions; for instance, grasp a bottle and hold it up. For further usage, it can replace human for doing something dangerous.

Taking an example, somebody needs to move a bottle from one place to another place; however, the bottle is in a room with toxic gas which people cannot go in. At this time, this person can collect some data outside the room, and utilize the robotic arm to imitate his motions.

The essential point of the project is that human moves something from one point to another point, and we can record the data of hand's coordinates and orientations in many points, then we generate new trajectories made up of points based on a specific model. We want the two trajectories as similar as possible so that it can imitate the motions which the person giving the data sets tend to do as well as possible

To avoid tuning parameters manually and worrying about instabilities, I will use a model named as dynamic movement primitives.

Compared to the past work [1], I will use the human motions' data sets rather than assisting the robotic arm to accomplish some specific motions and collecting the data sets for later implement.

1.2 Basic Theories

For the control theory or model, dynamic movement primitives will be utilized. And for optimization, I will use locally weighted regression.

The input will be vectors which include coordinates, rotation angles with corresponding time. The output will be coordinates and rotation matrices with corresponding time. All data will be in SE(3) ($\mathbb{R}^3$ or $\mathbb{R}^{3\times3}$). The output can also be written in $\mathbb{R}^{4\times4}$ form as $\begin{bmatrix} R & p \\ 0 & 1 \end{bmatrix}$ where $R \in \mathbb{R}^{3\times3}$ and $p \in \mathbb{R}^3$.

We assume collected data are close enough and continuous so that we can calculate the derivatives and second derivatives of each point.

More details of theories will be shown in next chapter.

1.3 Future Difficulties

The difficulties of the project is included but limited in:

1. Potential problems when extracting data from human motions, for example, trajectory differences among people.
2. Inverse kinematic solver simplification or development.
3. Noise in data.

OBJECTIVES?

Chapter 2

Theories and Background

This chapter mainly discuss the theories I will utilize when solving the problem. It includes two main parts: dynamic movement primitives and locally weighted regression.

2.1 Dynamic Movement Primitives

Dynamic movement primitives (DMPs) is a method of trajectory control or planning from Stefan Schaal's lab. They were presented ~~way~~ in 2003[2], and then updated in 2013 by Auke Ljspeert[1]. This work was motivated by the desire to find a way to represent complex motor actions that can be flexibly adjusted without manual parameter tuning or having to worry about instability.

Formally, if dynamic system one obeys $\dot{a} = K_1(a)$ and system two yields $\dot{b} = K_2(b)$, then the existence of an orientation preserving homeomorphism K_h: $\begin{bmatrix} a & \dot{a} \end{bmatrix} \xrightarrow{K_h} \begin{bmatrix} b & \dot{b} \end{bmatrix}$ and $\begin{bmatrix} a & \dot{a} \end{bmatrix} \xleftarrow{{K_h}^{-1}} \begin{bmatrix} b & \dot{b} \end{bmatrix}$ prove topological equivalence. It allows us to t[illegible] the average from some similar motions for getting more general results.

When having topological equivalence, DMPs retain their qualitative behavior if translated and if scaled in both space and time. And for this project, I will only use discrete DMPs model.

2.1.1 Positions with DMPs

The basic equation for positions are shown following:[1]

$$\ddot{y} = \tau^2[\alpha_y(\beta_y(g - y) - \dot{y}) + f]$$

$$\dot{x} = \tau(-\alpha_x x)$$

In above equations, α_y, β_y and α_x are parameters which we need to self-determine. And y is the state variable as well as the output; x is the diminishing term, which has an initial value of 1 and approaches to 0 when time approaches to infinity. g means the goal state; τ is the temporal scaling term, for higher accuracy τ will take the value 1.0 generally.

Actually, we can solve the equation of x, and can get the expression of x as:

$$x(t) = e^{-\tau\alpha_x t}$$

To be specific, x and t are all discrete, it will has n points, which are $t_0, \cdots, t_n$ and $x_0, \cdots, x_n$. So, n is the amount of time-steps.

The first equation looks similar as a PD controller, and f is an nonlinear force term which is correlated to the input data, which is define as:

$$f(x, g) = \frac{\sum_{i=1}^{N} \psi_i w_i}{\sum_{i=1}^{N} \psi_i} x(g - y_0)$$

In this equation, the other parameters remain their meanings from above equations. y_0 is the initial state, which can be directly achieved from given data; N is the number of basis functions and it need to be self-determined, which is related to the accuracy of imitation.

ψ_i, the basis function, is a Gaussian function defined as:

$$\psi_i(t) = e^{-h_i(x(t)-c_i)^2}$$

where t is time, c_i is the center of the Gaussian function and h_i is a coefficient correlated to the variance. And c_i is the diminishing term, which means $c_i = x_i$.

When choosing $h_i s$, we need to consider about the activation of $\psi_i s$ since it is like superposition of multiple Gaussian functions. Thus, we want the $\psi_i s$ to be identical in shape.

Thus, for properly activating Gaussian functions, the relationship of c_i and h_i is:

$$h_i = \frac{N^{\frac{3}{2}}}{\alpha_x \cdot c_i}$$

The relationship is extracted from some code which is confirmed to work.[3]

The most important as well as complicated parameter in f is the weights w_i, it determines the accuracy of imitation, which means the shape of the curve. It can be optimized by locally weighted regression, which will be shown in next chapter.

According to the conclusion from Schaal's paper[4], the formula of weights is given as:

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

where s and Ψ_i are defined as:

$$s = \begin{bmatrix} x(t_0)(g - y_0) \\ \vdots \\ x(t_n)(g - y_0) \end{bmatrix}, \Psi_i = \begin{bmatrix} \psi_i(t_0) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \psi_i(t_n) \end{bmatrix}$$

And f_d is the desired force term which is calculated from DMPs model formula:

$$f_d = \frac{1}{\tau^2} \ddot{y}_d - [\alpha_y(\beta_y(g - y_d) - \dot{y}_d)]$$

where y_d is the input data and its derivatives and second derivatives are able to calculate according to the assumption in section 1.2.

2.1.2 Orientations with DMPs

The orientation imitation and position imitation are extremely similar. However, there are some slight differences when implementing DMPs on orientations.

Firstly, the input will be series of Euler angles: αs, βs and γs. We need to convert them into desired rotation matrices R_d for later calculations:

$$R_d = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix}$$

Then R_d will be the input to DMPs system.

The basic equations for orientation imitation with DMPs are:[5]

$$\tau\dot{\eta} = \alpha_z[\beta_z \log(R_g R^T) - \eta] + f_0(x)$$
$$\tau\dot{R} = [\eta]_\times R$$
$$\eta = \tau\omega$$
$$\dot{x} = \tau(-\alpha_x x)$$

In above equations, $\omega \in \mathbb{R}^3$, which is the scale angular velocity along three axes, R_g is the goal state. And η is the state variable, x is the diminishing term as section 2.1.1, f_0 is the force term. R is the output which means the rotation matrices. α_z, β_z and α_x are coefficients we need to choose. Also, τ will take 1.0 as section 2.1.1. In addition, $[\eta]_\times \in \mathbb{R}^{3\times3}$ which is defined as:[6]

$$[\eta]_\times = \begin{bmatrix} 0 & -\eta_x & \eta_y \\ \eta_z & 0 & -\eta_x \\ -\eta_y & \eta_x & 0 \end{bmatrix}$$

And the force term can be defined as section 2.1.1:

$$f(x, R_g) = \frac{\sum_{i=1}^{N} \psi_i w_i}{\sum_{i=1}^{N} \psi_i} x \log(R_g {R_0}^T)$$

where R_0 is the initial state, and $w_i \in \mathbb{R}^{3\times3}$ which is slight different from the equation in section 2.1.1. Also, ψ_i is defined as:

$$\psi_i(t) = e^{-h_i(x(t)-c_i)^2}$$

where t is time, c_i is the center of the Gaussian function and h_i is a coefficient correlated to the variance. And c_i is the diminishing term, which means $c_i = x_i$. And the relationship of c_i and h_i is the same as section 2.1.1.

To optimize it, except for dimension different, the equation is the same as section 2.1.1:

$$w_i = \frac{s^T \Psi_i f_{0d}}{s^T \Psi_i s}$$

where s and Ψ_i are defined as:

$$s = \begin{bmatrix} x(t_0)\log\left(R_g {R_0}^T\right) \\ \vdots \\ x(t_n)\log\left(R_g {R_0}^T\right) \end{bmatrix}, \Psi_i = \begin{bmatrix} \psi_i(t_0) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \psi_i(t_n) \end{bmatrix}$$

And f_{0d} is the desired force term which is calculated from DMPs model formula:

$$f_{0d} = \tau\dot{\eta} - \alpha_z[\beta_z \log\left(R_g {R_d}^T\right) - \eta]$$

where η and its derivatives can be calculated from Euler angles according to the assumption in section 1.2.

Also, it is necessary to clarify that $\log\left(R_g R^T\right) \in \mathbb{R}^{3\times3}$, it can be calculate as:[7]

$$\log A = \begin{cases} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, & R = I \\ \omega = \theta n, & otherwise \end{cases}$$

In this equation, it shows general calculation, we have:

$$\theta = \arccos\left(\frac{trace(A) - 1}{2}\right), n = \frac{1}{2\sin\theta}\begin{bmatrix} A_{32} - A_{23} \\ A_{13} - A_{31} \\ A_{21} - A_{12} \end{bmatrix}$$

where A_{ij} is corresponding entry in A matrix. And we can substitute $R_g R^T$ into A.

When $\sin\theta = 0$, the above formula cannot work out the solution. It can have numerically stable formula in the book[7]. However, it is a discontinuity in the logarithmic map. It means a boundary where the logarithmic map switches from positive to negative rotation angles.

2.2 Locally Weighted Regression

Locally weighted regression (LWR) [8] is a class of function approximation techniques, where a prediction is done by using an approximated local model around the current point of interest

It can overcome the disadvantage of global method: sometimes no parameter can provide a sufficient good approximation.

~~The computational costs for some tasks are extremely high. Local algorithm can avoid the high computational costs.~~

2.2.1 Problem Definition

Generally, in an optimization problem, the question can be denoted as:

$$y = K_f(x) + \epsilon$$

where $K_f(x)$ is the input, y is the output and ϵ is the noise.

The general cost of LWR which need to be minimized is defined as:

$$cost = \sum_{i=1}^{n} C_{wi}(x_q)(y_i - x_i\beta_q)^2$$

where $C_{wi}(\cdot)$ is the weight function; and x_q is the query point, also named as the point of interest, which means the point where we want the prediction. y_i and x_i are output and input. β_q is the coefficient that we want to calculate.

The main idea of the problem is to minimize the total distance to desired points.

To be specific, according to the model equations in section 2.1.1, the problem converts to minimize the cost:

$$\sum_{t} \psi_i(t)(f_d(t) - w_i(x(t)(g - y_0)))^2$$

2.2.2 Basic Algorithm

Using corresponding variable names, the whole algorithm can be written as:[9]
Give:
(1) a set of training points, as known as input.
Prediction:
(1) Build matrix $s = \begin{bmatrix} x_{t0}(g-y_0) & \cdots & x_{tn}(g-y_0) \end{bmatrix}$
(2) Build vector $f_d = \begin{bmatrix} f_d(t_0) & \cdots & f_d(t_n) \end{bmatrix}^T$
(3) calculate Gaussian function $\psi_i s$ and generate diagonal matrix Ψ_i:

$$\Psi_i = \begin{bmatrix} \psi_i(t_0) & \cdots & 0 \\ 0 & \ddots & 0 \\ 0 & \cdots & \psi_i(t_n) \end{bmatrix}$$

(4) Calculate Regression coefficient:

$$w_i = (s^T \Psi_i s)^{-1} s^T \Psi_i f_d = \frac{s^T \Psi_i f_d}{s^T \Psi_i s}$$

Chapter 3

Completed Work

3.1 Coordinate Imitation Completed

I have successfully completed coordinate imitation utilizing package "pydmp"[10] with some modification in Python:

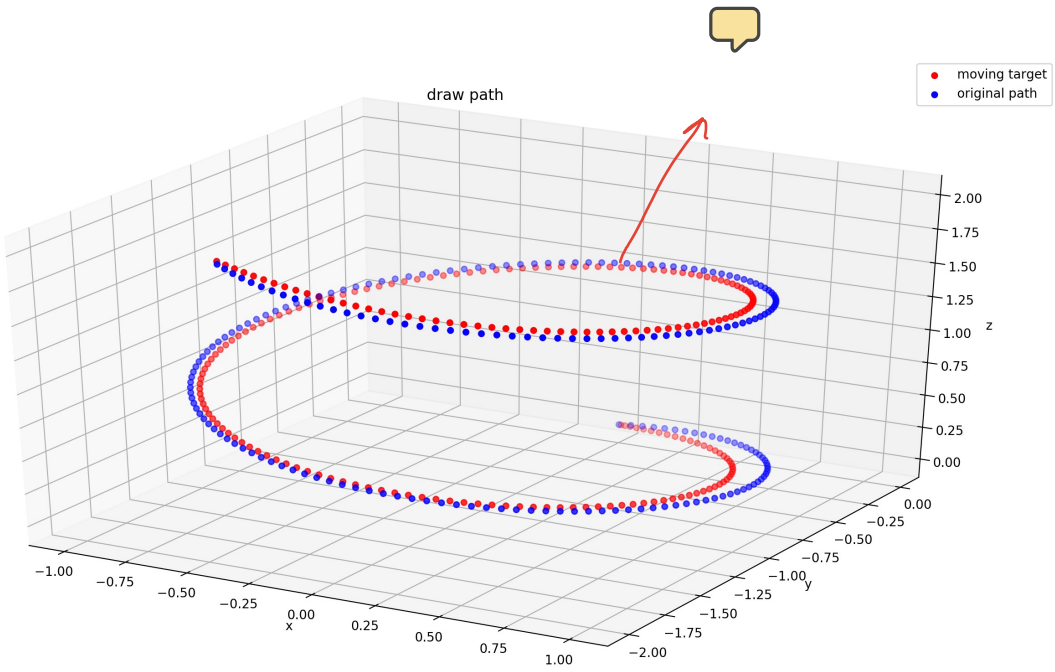

Figure 3.1: Original and imitated path in 3D space

At this time, coefficient are:

$$\alpha_x = 1.0$$

$$\tau = 1$$

$$\alpha_y = 25.0$$

$$\beta_y = 4.0$$

$$unit\,time(dt) = 0.005s$$

$$N = 100$$

And the accuracy can be shown as following:

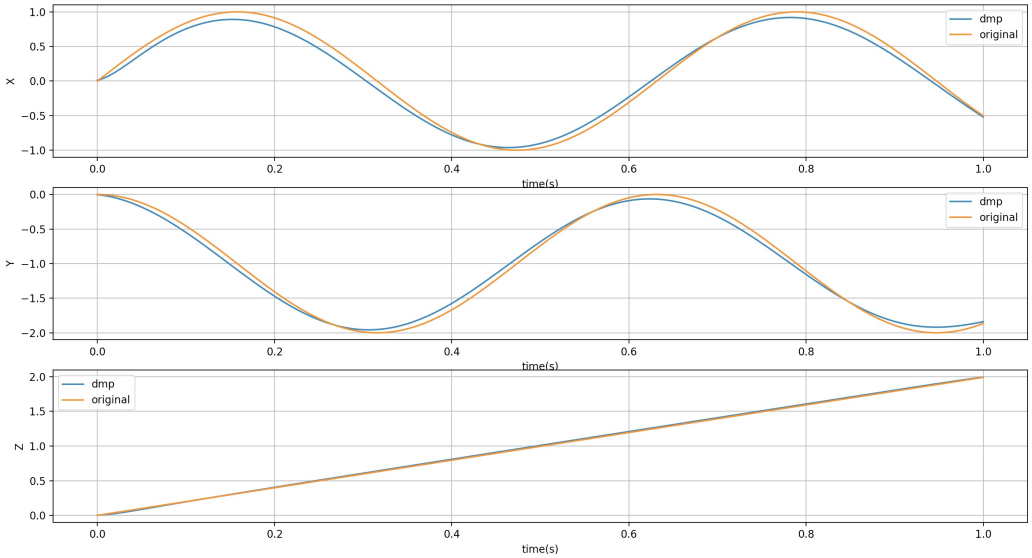

Figure 3.2: Coordinates versus time in three axes

3.2 Working on orientation imitation

I did finish main code of orientation imitation on August 2019, but the accuracy need to improve significantly.

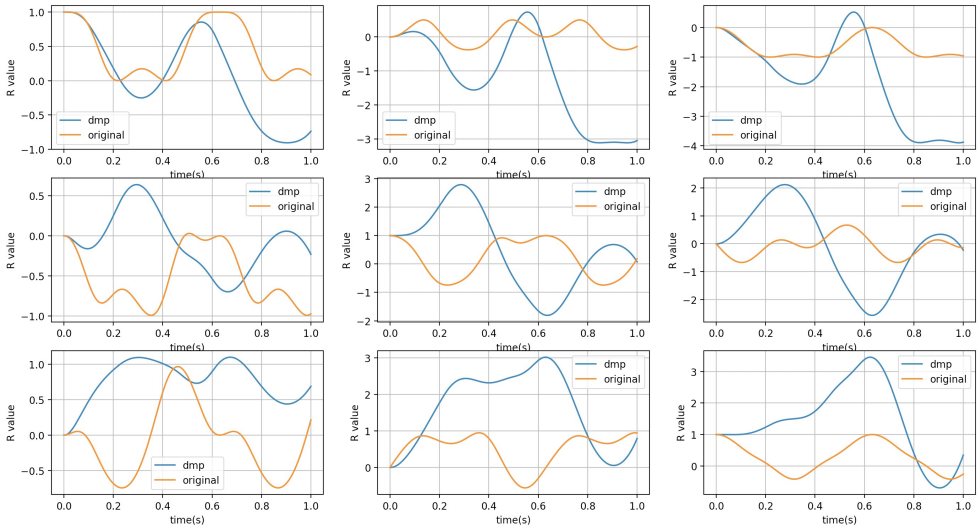

Figure 3.3: R matrix entries versus time

At this time, coefficient are:

$$\alpha_x = 1.0$$

$$\alpha_y = 10.0$$

$$\beta_y = 4.0$$

$$unit\,time(dt) = 0.005s$$

$$N = 100$$

I'm currently working on the development of orientation imitation.
For more information of DMPs, this website[11] may help somehow.

Chapter 4

Time line

Date	Purposes
2019.07.31	Accomplish the position and orientation imitation simulation for at least one trial trajectory
2019.08.31	Improve the accuracy of imitations and build remote API for VREP to import and export data
2019.09.31	Extract data from human motions and solve corresponding problems
2019.10.31	Make the simulation work in VREP with trajectory extracted from human motions
2019.11.30	Make the real robotic arm system work with given human motions
2019.12.31	Accomplish all experiments (no later than 2020.01.29)
2020.03.10	First draft of thesis
2020.03.26	Final draft handed in
2020.04.09	Trail Defense
2020.04.15	Final Defense (date may change)

Bibliography

[1] A. J. Ijspeert, J. Nakanishi, H. Hoffmann, P. Pastor, and S. Schaal, *Dynamical movement primitives: learning attractor models for motor behaviors*, Neural computation **25**, 328–373 (2013).

[2] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. *Control, planning, learning, and imitation with dynamic movement primitives.* In *Workshop on bilateral paradigms on humans and humanoids, IEEE International Conference on Intelligent Robots and Systems*, pages 1–21, (2003).

[3] T. DeWolf. *Discrete Dynamic Movement Primitives Model Code, https://github.com/studywolf/pydmps/blob/master/pydmps/dmp_discrete.py.*

[4] S. Schaal, C. G. Atkeson, and S. Vijayakumar, *Scalable techniques from nonparametric statistics for real time robot learning*, Applied Intelligence **17**, 49–60 (2002).

[5] A. Ude, B. Nemec, T. Petrić, and J. Morimoto. *Orientation in cartesian space dynamic movement primitives.* In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2997–3004. IEEE, (2014).

[6] R. M. Murray, *A mathematical introduction to robotic manipulation*, CRC press (2017).

[7] N. Ayache, *Artificial vision for mobile robots: stereo vision and multisensory perception*, MIT Press (1991).

[8] P. Englert. *Locally weighted learning.* In *Seminar Class on Autonomous Learning Systems*, (2012).

[9] A. Gams. *Generalization, Locally Weighted Regression, Gaussian Process Regression.*

[10] T. DeWolf. *Dynamic Movement Primitives Model in Python, https://github.com/studywolf/pydmps/blob/master/pydmps.*

[11] T. DeWolf. *Dynamic Movement Primitives Model, https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/.*