2020

Imitation learning with dynamic movement primitives

BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Thesis

IMITATION LEARNING WITH DYNAMIC MOVEMENT PRIMITIVES

by

HAOYING ZHOU

B.S., Beijing Institute of Technology, 2018

Submitted in partial fulfillment of the

requirements for the degree of

Master of Science

2020

Approved by

First Reader

Calin A. Belta, Ph.D.
Professor of Mechanical Engineering
Professor of Systems Engineering
Professor of Electrical and Computer Engineering

Second Reader

Sean B. Andersson, Ph.D.
Professor of Mechanical Engineering
Professor of Systems Engineering

Third Reader

Roberto Tron, Ph.D.
Assistant Professor of Mechanical Engineering
Assistant Professor of Systems Engineering

Acknowledgments

I would like to express my sincere gratitude to my academic advisor, Dr. Calin Belta for his guidance and support. Also, I would like to thank Dr. Xiao Li for his assistance and cooperation.

I am very grateful to Dr. Sean Andersson and Dr. Roberto Tron for serving in my thesis committee and their interests that they have indicated in my research.

I would like to thank Ms. Anna Masland who is the master's program administrator in my department. She has assisted me a lot in scheduling the thesis process.

In addition, I would like to thank Mr. Brendan McDermott, the thesis / dissertation coordinator from Mugar Memorial Library at Boston University, for his assistance in correcting the format of the thesis.

To my family, my friends and the people who have ever supported me, I would express my most profound appreciation for their love and encouragement all the time.

Haoying Zhou
Master's Student
ME Department

IMITATION LEARNING WITH DYNAMIC MOVEMENT PRIMITIVES

HAOYING ZHOU

ABSTRACT

Scientists have been working on making robots act like human beings for decades. Therefore, how to imitate human motion has became a popular academic topic in recent years. Nevertheless, there are infinite trajectories between two points in three-dimensional space. As a result, imitation learning, which is an algorithm of teaching from demonstrations, is utilized for learning human motion. Dynamic Movement Primitives (DMPs) is a framework for learning trajectories from demonstrations. Likewise, DMPs can also learn orientations given rotational movement's data. Also, the simulation is implemented on Robot Baxter which has seven degrees of freedom (DOF) and the Inverse Kinematic (IK) solver has been pre-programmed in the robot, which means that it is able to control a robot system as long as both translational and rotational data are provided. Taking advantage of DMPs, complex motor movements can achieve task-oriented regeneration without parametric adjustment and consideration of instability.

In this work, discrete DMPs is utilized as the framework of the whole system. The sample task is to move the objects into the target area using Robot Baxter which is a robotic arm-hand system. For more effective learning, a weighted learning algorithm called Local Weighted Regression (LWR) is implemented. To achieve the goal, the weights of basis functions are firstly trained from the demonstration using DMPs framework as well as LWR. Then, regard the weights as learning parameters and substitute the weights, desired initial state, desired goal state as well as time-correlated

parameters into a DMPs framework. Ultimately, the translational and rotational data for a new task-specific trajectory is generated. The visualized results are simulated and shown in Virtual Robot Experimentation Platform (VREP). For accomplishing the tasks better, independent DMP is used for each translation or rotation axis. With relatively low computational cost, motions with relatively high complexity can also be achieved. Moreover, the task-oriented movements can always be successfully stabilized even though there are some spatial scaling and transformation as well as time scaling.

Twelve videos are included in supplementary materials of this thesis. The videos mainly describe the simulation results of Robot Baxter shown on Virtual Robot Experimentation Platform (VREP). The specific information can be found in the appendix.

Contents

List of Tables

List of Figures

List of Abbreviations

DMP(s)	.....	Dynamic movement primitive(s)
DOF	.....	Degrees of Freedom
GMM	.....	Gaussian Mixture Model
IK	.....	Inverse Kinematics
LWL	.....	Locally Weighted Learning
LWR	.....	Locally Weighted Regression
ROS	.....	Robot Operation System
VREP	.....	Virtual Robot Experimentation Platform
3D	.....	Three-Dimensional
$\mathbb{R}^3$	.....	the real three-dimensional space

Chapter 1

Introduction

1.1 Problem Definition and Introduction

Scientists have shown their interest in making robots act like human beings for decades. Likewise, how to imitate human motion has become a popular academic topic recent years. Therefore, a trajectory planning framework named Dynamic Movement Primitives (DMPs) was created by Stefan Schaal's Lab at USC in 2002 (Schaal et al., 2003; Schaal, 2006; Ijspeert et al., 2013). This method was motivated by the requirement of representing complex motor movements and it can implement complex motor actions by adjusting few manual parameters and ensuring the stability.

In this thesis, the DMPs framework is utilized for imitation learning. The input is some discrete points sampled from a demonstration trajectory of motion generated by a human using some specific controllers such as a joystick, which includes both translational and rotational information. Firstly, substitute the processed input data (DeWolf, 2013b) into the second order differential equation of the DMPs framework. Then, the weights of the DMPs model's basis functions are trained (DeWolf, 2013a). Next, substitute initial state, goal state of the output system and the weights trained from input system into DMPs for regenerating the output trajectory. Ultimately, the output is also discrete points which consists of a trajectory that can manage to do the same task as the demonstration trajectory. Even though the initial states and goal states may be not identical in input and output systems, similar tasks can still be done by the same set of weights. The simulations are implemented on a seven

DOF robot arm-hand system named Robot Baxter.

In the learning part, a training method called Locally Weighted Regression (LWR) is utilized (Schaal et al., 2002; Gams, 2018). The weights trained in LWR are used for motion regeneration. With translation and scaling in both space and time , the shape of trajectories stays invariant (Schaal et al., 2003).

In this work, both 3D coordinates and orientations of the Robot Baxter's end-effector are considered for learning and the two terms' combination allow further development of human-robot interaction. For rotational motion representation, rotation matrix is utilized. When imitating both translational and rotational movements, the time step of simulations is set to be 50 millisecond. The strategy can undertake some spatial scaling or transformation of the initial or goal states. In addition to space scale or transformation for smoothing, the task-oriented motion can be also scaled in time, which may provide more time-saving solution to some specific tasks. For further development, it can be anticipated to stabilize some disturbances on the trajectories at any time (Theodorou et al., 2010).

For real-life implementation or future development, this thesis's work can be utilized to accomplish some task-oriented motions with demonstrations. For further extension, it may combine with some advanced learning algorithms to achieve higher generality. Likewise, it can be incorporated within some other algorithms such as obstacle-avoiding algorithm to solve some advanced or complicated robot control problems.

1.2 Related Work

DMPs has been developing since it was raised in Schaal's lab and it has become one of the most common-used framework for learning trajectories from demonstrations (Matsubara et al., 2011). This framework is based on a system which consists of

second-order ordinary differential equations with nonlinear forcing term. The nonlinear forcing term can be weight-trained and learned for regenerating task-oriented trajectories. For more complicated movements, limit cycle can be added into the discrete model (Ude et al., 2010). Since DMPs is sensitive to time and can always reach the goal state if the running time is long enough, thus reinforcement learning with time-related reward or some time-related control strategies can be applied to the DMPs framework (Tamosiunaite et al., 2011; Theodorou et al., 2010) .

The framework was firstly applied on the learning algorithm which is to imitate the motion described via only point attractors (Ijspeert et al., 2002; Ijspeert et al., 2003) and further extended to general movements or human motion (Rückert and d'Avella, 2013; Rosado et al., 2014; Nemec and Ude, 2012). To be specific, the framework has been proved effective in robot learning of some human tasks, which includes arm swings (Matsubara et al., 2011), drum playing (Ude et al., 2010), handwriting (Kulvicius et al., 2011) and limb motion learning (Rosado et al., 2014). Likewise, the approach has shown its flexibility and robustness for allowing to implement obstacle avoidance algorithm (Park et al., 2008) or some advanced control strategies (Pervez and Lee, 2018; Chi et al., 2019). In addition to pure point-to-point learning, it has potential on learning from orientations (Ude et al., 2014; Kramberger et al., 2016; Ginesi et al., 2019) and generalization for some tasks (Zhou and Asfour, 2017).

Nevertheless, this work mainly focuses on non-cycled movements of hand-arm system. Considering the difficulty of real-life implementation and the desire to explore the flexibility and availability of discrete models, the discrete DMPs is the specific model discussed in later chapters.

Chapter 2

Basic Theories and Application

2.1 Dynamic Movement Primitives

Dynamic Movement Primitives (DMPs) is a framework for learning a point-to-point trajectory from a demonstration. This framework is to represent a movement trajectory with a group of second-order ordinary differential equations. Then, some learning methods such as Gaussian Mixture Model (GMM) (Pervez and Lee, 2018) can be implemented to the framework for learning some complex motor motion. Ultimately, a canonical dynamical system defined by a first order differential equation shown in equation 2.3 is utilized to converge the trained nonlinear term so that the whole dynamic system can stay bounded. DMPs is motivated by the desire of representing complex motor actions. Furthermore, this whole method can implement complex motor movements with adjusting few manual parameters and without worrying about instability.

2.1.1 Introduction

Generally, the goal of the motion learning can be formulated to a policy of finding a control strategy of the specific tasks which is to move the control objects into the target area: (Schaal et al., 2003)

$$u = \pi(x, t, \alpha_d, N) \tag{2.1}$$

where x is the state vector, t is the time, u is the control vector, α_d is the dynamic parameter vector which is specific to the control policy π, and N is the hyperparameter vector about how to run the model more effective.

Likewise, the dynamic system utilized in the policy shown in equation 2.1 can generally be written as a differential equation:

$$\dot{x} = f(x, t, \alpha_d) \tag{2.2}$$

To be specific, in most applications of robots, the complexity of learning the control policy is reduced because some extra information is provided. The most common situation is that the desired trajectories are given in some demonstrations.

A potential of combining equation 2.1 and 2.2 exists (Schaal et al., 2003) . DMPs is a method that can make the two equation correlated.

2.1.2 Discrete DMPs

Parameter table of discrete DMPs model

Sign	Name
c_k	Center of ψ_k, the mean of Gaussian function
dt	Time length of a single time step
f	Nonlinear term, feedback term
g	Goal state of the desired dynamic system
h_k	Parameter of ψ_k, correlated to the variance of Gaussian function
N_{bfs}	Number of the basis functions
N_{pts}	Number of the discrete points in the desired dynamic system
T	Total running time
t_i	Time of the i_{th} time step in the desired dynamic system
x	Canonical dynamic system state
y	Desired dynamic system state
y_0	Initial state of the desired dynamic system
w_k	Weight of the k_{th} basis function
α_x	Gain term of the canonical dynamic system
α_y, $\alpha_{\dot{y}}$	Gain terms of the desired dynamic system
ψ_k	the k_{th} basis function, which is a Gaussian function

Table 2.1: Parameter Table of discrete DMPs model

Generally, all data extracted from demonstration trajectories is sets of discrete points. Although most robotic trajectories are continuous in our real life, they are

sampled discretely when communicating between hardware and software. The extracted trajectories are first-order and second-order derivable after proper interpolations and approximations (DeWolf, 2013b). Hence, in this work, discrete DMPs model is implemented (Schaal et al., 2003). The corresponding equations are (DeWolf, 2013a):

$$\begin{aligned} \ddot{y}(t_i) &= \alpha_{\dot{y}}(\alpha_y(g - y(t_i)) - \dot{y}(t_i)) + f(t_i) \\ \dot{x}(t_i) &= -\alpha_x x(t_i) \\ dt &= t_i - t_{i-1} \\ T &= N_{pts} * dt \end{aligned} \tag{2.3}$$

In equation 2.3, f is the nonlinear force term for feedback defined as:(DeWolf, 2013a)

$$\begin{aligned} f(x(t_i), g) &= \frac{\sum_{k=1}^{N_{bfs}} \psi_k(t_i) w_k}{\sum_{k=1}^{N_{bfs}} \psi_k(t_i)} x(t_i)(g - y_0) \\ \psi_k(t_i) &= e^{-h_k(x(t_i) - c_k)^2} \end{aligned} \tag{2.4}$$

The canonical dynamic system in 2.3 can be solved by integrating both sides of the second equation:

$$x(t_i) = e^{-\alpha_x t_i} \tag{2.5}$$

It can see that x is nonlinear, then the choice of the basis function's centers and variances needs to be careful and tricky so that the basis functions can be activated evenly in time. Only if basis functions are activate evenly in time, the force term can move along the demonstration trajectory when approaching to the goal state (DeWolf, 2013a). If we choose the centers of the basis functions linearly, according to the expression formula of x, most basis functions are activated significantly when x moves at the beginning, and the activation stretch out when the movement of x

approaches to the end.

To make the activation of basis function even in time, the i_{th} Gaussian function (basis function) center is selected as:

$$\begin{aligned} c_k &= e^{-\alpha_x T_k} \\ T_k &= \frac{k}{N_{bfs}} T \end{aligned} \tag{2.6}$$

Likewise, the i_{th} variance-related coefficient h_i is expressed as:(DeWolf, 2013a)

$$h_k = \frac{{N_{bfs}}^{\frac{3}{2}}}{\alpha_x \cdot c_k} \tag{2.7}$$

When having topological equivalence, DMPs retain their qualitative behaviors if translated or scaled in both space and time. It was proved in Schaal's paper (Schaal et al., 2003). Different DMPs with topological equivalence can be converted to one DMP via simple addition or multiplication to the dynamic equations. This property allows us to imitate different human motions without changing parameters of dynamic systems. To be specific, we can scale the results in time via changing the length of one time step dt. Likewise, the result can also be scaled in space via choosing different initial and goal state positions.

2.1.3 DMPs in Translational Motion

For translation, the model used is almost identical to the model described in section 2.1.2 (Ijspeert et al., 2003).

Denote the input data of translational movements as y_{input}, which is the input system state for DMPs . Then, we can get the nonlinear term f_{input} of the input system from the dynamic equation of the model which is shown in equation 2.3 :

$$f_{input}(t_i) = \ddot{y}_{input}(t_i) - \alpha_{\dot{y}}(\alpha_y(g - y_{input}(t_i)) - \dot{y}_{input}(t_i)) \tag{2.8}$$

Based on the solution of LWR which is discussed in section 2.2 , the weights of the basis functions can be expressed by:

$$w_k = \frac{s^T \Psi_k f_{input}}{s^T \Psi_k s} \tag{2.9}$$

where

$$\begin{aligned} s &= \begin{bmatrix} x(t_0^p)(g - y_0) \\ \vdots \\ x(t_n^p)(g - y_0) \end{bmatrix} \\ \Psi_k &= \begin{bmatrix} \psi_k(t_0^p) & \cdots & 0 & 0 \\ \vdots & \psi_k(t_1^p) & \ddots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \psi_k(t_n^p) \end{bmatrix} \end{aligned} \tag{2.10}$$

where t_0^p, t_1^p ... t_n^p indicates the corresponding time of discrete points extracted from different demonstration translational trajectories.

The weights extracted from the training process are the learning parameters calculated from the demonstration. For generating new trajectories, substitute the weights into DMPs and calculate the nonlinear term of the output system f_{out} using the same canonical dynamic system x as equation 2.3:

$$\begin{aligned} f_{out}(x(t_i), g) &= \frac{\sum_{k=1}^{N_{bfs}} \psi_k(t_i) w_k}{\sum_{k=1}^{N_{bfs}} \psi_k(t_i)} x(t_i)(g - y_0) \\ \psi_k(t_i) &= e^{-h_k(x(t_i) - c_k)^2} \end{aligned} \tag{2.11}$$

Then, using the dynamic equation of discrete DMPs model, work out the acceleration of the generated trajectory which is the second derivative of the output system state:

$$\ddot{y}_{out}(t_i) = \alpha_{\dot{y}}(\alpha_y(g - y_{out}(t_i)) - \dot{y}_{out}(t_i)) + f_{out}(t_i) \tag{2.12}$$

where y_{out} is the output system state.

Ultimately, integrate the accelerations to get the velocity and displacement in global coordinate system:

$$\begin{aligned} i_{th}\ point\ velocity &= \dot{y}_{out}(t_i) = \dot{y}_{out}(t_{i-1}) + \ddot{y}_{out}(t_i) \cdot dt \\ i_{th}\ point\ displacement &= y_{out}(t_i) = y_{out}(t_{i-1}) + \dot{y}_{out}(t_i) \cdot dt \end{aligned} \tag{2.13}$$

In above equations, it is assumed that the output trajectory is generated evenly in time. Also, it is assumed that the initial velocity and acceleration are zero.

2.1.4 DMPs in Rotational Motion

Parameter table for rotational discrete DMPs model

Sign	Name
c_k^o	Center of ψ_k^o, the mean of Gaussian function
D	Diagonal matrix in terms of R_0 and R_g
f_R	Nonlinear term, feedback term
R	Desired system state, rotation matrix
R_g	Rotation matrix at goal state of the desired dynamic system
R_0	Rotation matrix at initial state of the desired dynamic system
h_k^o	Parameter of ψ_k^o, correlated to the variance of Gaussian function
N_{dbfs}	Number of the basis functions
x_R	Canonical dynamic system state
w_k^o	Weight of the k_{th} basis functions
α_{xR}	Gain term of the canonical dynamic system
α_R, $\alpha_{\dot{R}}$	Gain terms of the desired dynamic system
γ	Scaled angular velocity calculated using Rodrigue's formula
ω	Scaled angular velocity of input trajectories, $\omega \in \mathbb{R}^3$
ω_x, ω_y, ω_z	Scaled angular velocity components along x,y and z axis
ψ_k^o	the k_{th} basis function, a Gaussian function

Table 2.2: Parameter Table of rotational discrete DMPs model

For the parameters dt, T, t_i and N_{pts}, we use the same parameters as the translational model, and their definitions can be found from table 2.1.

The implementation of DMPs on rotation is similar to the translation one. Nevertheless, the mathematical method of calculating rotation matrix and its derivatives is different from the Euclidean ones, which is discussed in appendix A specifically. Therefore, the model is slight different from the model shown in section 2.1.2. Furthermore, the model of orientations does not have to be intuitive because the rotation

centers is the position of the end-effector at the same time, which can be obtained when implementing the algorithm described in 2.1.3.

The model of DMPs for rotation is based on equation 2.3 (Ude et al., 2014), and it can be expressed as:

$$\begin{aligned}
\dot{\omega}(t_i) &= \alpha_{\dot{R}}[\alpha_R \gamma(t_i) - \omega(t_i)] + f_R(t_i) \\
\gamma(t_i) &= \log\left(R_g R(t_i)^T\right) \quad \in \mathbb{R}^3 \\
[\omega(t_i)]_\times &= \begin{bmatrix} 0 & -\omega_x(t_i) & \omega_y(t_i) \\ \omega_z(t_i) & 0 & -\omega_x(t_i) \\ -\omega_y(t_i) & \omega_x(t_i) & 0 \end{bmatrix} \\
\dot{R}(t_i) &= [\omega(t_i)]_\times R(t_i) \\
\dot{x}_R(t_i) &= -\alpha_{xR} x_R(t_i)
\end{aligned} \tag{2.14}$$

where the calculation of logarithmic function and rotation matrix derivative can be found in appendix section A.2.

In equation 2.14, the nonlinear term can be calculated from following equations: (Ude et al., 2014)

$$\begin{aligned}
f_R(x_R(t_i), R_g) &= \frac{\sum_{k=1}^{N_{dbfs}} \psi_k^o(t_i) w_k^o}{\sum_{k=1}^{N_{dbfs}} \psi_k^o(t_i)} x_R(t_i) D \quad \in \mathbb{R}^3 \\
D &= diag(\gamma(t_0)) = diag(\log\left(R_g {R_0}^T\right)) \quad \in \mathbb{R}^{3\times 3} \\
\psi_k^o(t_i) &= e^{-h_k^o (x_R(t_i) - c_k^o)^2}
\end{aligned} \tag{2.15}$$

After replacing N_{bfs} with N_{dbfs} and replacing α_x with α_{xR}, the expression and relationship of x_R , c_k^o and h_k^o are identical to equation 2.5 , 2.6 and 2.7.

Using the model which is shown in equation 2.14, given the input data which is rotation matrix R_{input} calculated from appendix section A.1 defined Euler angles, the

nonlinear term of the input system f_{Rinput} can be expressed as:

$$\begin{aligned} f_{Rinput}(t_i) &= \dot{\omega}_{input}(t_i) - \alpha_{\dot{R}}[\alpha_R \gamma_{input}(t_i) - \omega_{input}(t_i)] \\ \gamma_{input}(t_i) &= \log\left(R_{ginput} R_{input}(t_i)^T\right) \end{aligned} \tag{2.16}$$

where γ_{input} is scaled angular velocity of the input system calculated using Rodrigue's formula and ω_{input} is scaled angular velocity extracted from the demonstration trajectory of the input system.

Based on the derivation in section 2.2.2, we can solve the weights of basis functions :

$$w_k^o = \left(s_o^T \Psi_k^o s_o\right)^{-1} s_o^T \Psi_k^o f_{Rinput} \tag{2.17}$$

where

$$\begin{aligned} s_o &= \begin{bmatrix} x_R(t_0^o)\gamma_{input}(t_0)^T \\ \vdots \\ x_R(t_n^o)\gamma_{input}(t_n)^T \end{bmatrix} \\ &= \begin{bmatrix} x_R(t_0^o)(\log\left(R_g {R_{input}}^T(t_0^o)\right))^T \\ \vdots \\ x_R(t_n^o)(\log\left(R_g {R_{input}}^T(t_n^o)\right))^T \end{bmatrix} \\ \Psi_k^o &= \begin{bmatrix} \psi_k^o(t_0^o) & \cdots & 0 & 0 \\ \vdots & \psi_k^o(t_1^o) & \ddots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \psi_k^o(t_n^o) \end{bmatrix} \end{aligned} \tag{2.18}$$

where t_0^o , t_1^o , ... t_n^o indicates the time of different discrete points extracted from demonstration orientation .

Regard the weight as the learning term, regenerate the orientation movement of the demonstration. Firstly, work out the nonlinear forcing term of the output system f_{Rout}:

$$f_{Rout}(t_i) = \frac{\sum_{k=1}^{N_{dbfs}} \psi_k^o(t_i) w_k^o}{\sum_{k=1}^{N_{dbfs}} \psi_i^o(t_i)} x_R(t_i) D \tag{2.19}$$

Then, substitute the nonlinear terms into the model equations 2.14, it can calculate the derivative of the scaled angular velocity which is the scaled angular acceleration :

$$\begin{aligned} \dot{\omega}_{out}(t) &= \alpha_{\dot{R}}[\alpha_R \gamma_{out}(t) - \omega_{out}(t)] + f_{Rout}(x_R(t), R_g) \\ \gamma_{out}(t) &= \log\left(R_g R_{out}(t)^T\right) \end{aligned} \tag{2.20}$$

where ω_{out} is the output scaled angular velocity, R_{out} is the output rotation matrix which can be worked out through equation A.5, $\gamma_{out}(t)$ is a coefficient correlated to the output rotation matrix, which is calculated by equation A.12 .

Likewise, integrate the scaled angular acceleration to calculate the scaled angular velocity of the i_{th} orientation demonstration point :

$$\omega_{out}(t_i) = \omega_{out}(t_{i-1}) + \dot{\omega}_{out}(t_i) \cdot dt \tag{2.21}$$

In above equation, it is assumed that the output trajectory of orientation is generated evenly in time. Likewise, it is assumed that the initial scaled angular velocity and acceleration are zero.

And the skew symmetry matrix related to scale angular velocity of the i_{th} point can be expressed as:

$$[\omega_{out}(t_i)]_\times = \begin{bmatrix} 0 & -\omega_{xout}(t_i) & \omega_{yout}(t_i) \\ \omega_{zout}(t_i) & 0 & -\omega_{xout}(t_i) \\ -\omega_{yout}(t_i) & \omega_{xout}(t_i) & 0 \end{bmatrix} \tag{2.22}$$

Therefore, the rotation matrix for next timestep can be calculated via:

$$R_{out}(t_i) = R_{out}(t_{i-1} + dt) = e^{dt \cdot [\omega_{out}(t_i)]_\times} R_{t_{i-1}} \tag{2.23}$$

The output rotation matrices can be the eventual output. If the robotic system cannot accept the rotation matrices, it can be converted into Euler angles through

equation $A.7$,$A.8$ and $A.9$.

Nevertheless, the model of rotational movements is slight different from the translational one. According to equation A.5, the comprehensive rotation matrix is calculated from multiplication of three rotation matrix of components along three axes. Likewise, in the implementation of the algorithm, three component rotation matrices are imported into DMPs at the same time, learn and regenerate corresponding output rotation matrix. Ultimately, multiple the three matrices to work out the comprehensive rotation matrix. This modification allows different DMPs implementations on rotation matrix around different axes. For simpler rotational movements, we can use smaller constant on number of basis functions, which can reduce the computational cost. For more comprehensive motion, we do not have to decompose the rotation matrix and implement the orientation imitation learning algorithm regularly.

In addition, quaternion can also be used for controlling rotational movements (Ude et al., 2014). The whole algorithm of quaternion's are similar to the rotation matrix one. The rotation matrix is used in the algorithm for testing its availability and generality.

2.2 Locally Weighted Regression

2.2.1 Introduction

Locally weighted regression (LWR) is one method of locally weighted learning (LWL). Locally weighted learning (LWL) is a class of techniques from nonparametric statics that provides useful representation and training algorithms for learning about complex motor movements in autonomous adaptive control of robotic systems. It is shown that LWR algorithm has been successfully implemented in real-time learning of complex robot tasks (Schaal et al., 2002).

Locally weighted regression (LWR) is a memory-based learning algorithm. Its

training is very fast, which only needs to add new training data to the memory. For prediction, LWR only need a query point and the training points in memory. Furthermore, it can provide sufficiently good local approximation for the motion trajectory.

2.2.2 Application and derivation

For the implementation in this thesis, LWR algorithm with locally linear models is utilized. In this section, only the LWR solution derivation of DMPs translational movements is shown as a demonstration. Likewise, the derviation of DMPs rotational movement is very similar and not shown in the section.

Based on model described in section 2.1.3 , we want to choose the weights of basis functions in order that the forcing term function matches the nonlinear term f_{input} of the input system . Therefore, the loss function which is need to be minimized is expressed as:

$$\begin{aligned} J(w_k) &= \sum_{t_i} \psi_k(t_i)(f(t_i) - w_k(x(t_i)(g - y_0)))^2 \\ \psi_k(t_i) &= e^{-h_k(x(t_)-c_k)^2} \end{aligned} \tag{2.24}$$

where the parameters meaning can be found in table 2.1.

Convert equation 2.24 into a matrix form. Firstly, substitute different ψ_k values at different time into a diagonal matrix Ψ_k:

$$\Psi_k = \begin{bmatrix} \psi_k(t_0) & \cdots & 0 & 0 \\ \vdots & \psi_k(t_1) & \ddots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \psi_k(t_n) \end{bmatrix} \tag{2.25}$$

where t_0, t_1 ... t_n indicates the corresponding time of different demonstration trajectory points.

Then, set up column vectors s and f_d for $x(t)(g - y_0))$ and f_{des}term:

$$\begin{aligned} s &= \begin{bmatrix} x(t_0)(g - y_0) \\ \vdots \\ x(t_n)(g - y_0) \end{bmatrix} \\ f_d &= \begin{bmatrix} f(t_0) \\ \vdots \\ f(t_n) \end{bmatrix} \end{aligned} \tag{2.26}$$

Therefore, the equation 2.24 become:

$$J(w_k) = (f_d - s^T w_k)^T \Psi_k (f_d - s^T w_k) \tag{2.27}$$

Ultimately, using least squares method, we can solve the weight:

$$w_k = (s^T \Psi_k s)^{-1} s^T \Psi_k f_d = \frac{s^T \Psi_k f_d}{s^T \Psi_k s} \tag{2.28}$$

2.3 Algorithm

The algorithm of implementation is shown following:

Algorithm 1 Imitation Learning With DMPs

Input:

Discrete translational and rotational motion information with time sampled from the demonstration trajectory.

Output:

Discrete translational and rotational motion information with time generated by DMPs which can accomplish the same or similar tasks as the demonstration.

1: Select parameters $\alpha_{\dot{y}}$, α_y, and α_x in equation 2.3 as well as $\alpha_{\dot{R}}$, α_R and α_{xR} in equation 2.14. The corresponding parameters are the same as the ones in equations 2.3 and 2.14. These parameters stay invariant in later procedures.

2: Extract data from the demonstration discretely. The data consists of both translational and rotational trajectories with time.
3: Choose the number of basis functions N_{bfs} properly.
4: Using the models described in section 2.1.3 and 2.1.4, train the basis functions' weights of translational and rotational trajectories. Store the weights.
5: Pick out the number of points in the generated trajectories N_{pts} properly.
6: Substitute the trained weights of basis functions, initial and goal state of translational and rotational movements into the models described in section 2.1.3 and 2.1.4 to generated results.
7: Simulate the results in VREP so that it can see the qualitative behaviors of algorithm directly.

For the gain parameters α_d of dynamic systems in the DMPs models, we set them to be (Schaal et al., 2003; Ude et al., 2014):

$$\alpha_{\dot{y}} = \alpha_{\dot{R}} = 25.0$$

$$\alpha_y = \alpha_R = 4.0$$

$$\alpha_x = \alpha_{xR} = 1.0$$

The above parameters' values can be modified following the rules of PD controller parameter selection (Lee et al., 1998) . The hyperparameters N, which are N_{bfs}, N_{dbfs} and N_{pts}, influence the qualitative behaviors of the imitation learning. Moreover, the hyperparameters which are the learning parameters need be picked out properly. Too small values may lead to the failure of reaching the minimum and too high ones may lead to the high computational cost. Generally, given the values of learning

parameters N_{bfs}, N_{dbfs} and N_{pts} in modest ranges, if the other parameters are invariant constants, with the learning parameters increasing, the demonstrate trajectories and the DMPs-generated ones have more similar qualitative behaviors. Ultimately, after picking randomly from modest ranges of N_{bfs}, N_{dbfs} and N_{pts}, the qualitative behaviors in specific situations are shown in detail in chapter 3.

Chapter 3

Simulation and Result

3.1 Model Evaluation

This section is to evaluate the capability of the model and algorithm described in section 2.3 when imitating different motions. Nevertheless, the goal of the model and algorithm is to regenerated task-oriented trajectories rather than identical trajectories. Also, when doing some task-specific motion, people can do it in extremely different ways. Therefore, the evaluations mainly focus on qualitative behaviors rather than precise numerical error analysis. What we need to care about is whether the point-to-point DMPs trajectories (Ginesi et al., 2019) can accomplish similar tasks and small errors do not mean the framework and models effective necessarily. To evaluate the qualitative results, we define that the errors of model evaluation are acceptable if they are less than 5% of the length of the range of translation or less than 5 degrees in rotation angles. Furthermore, it is acceptable to has larger errors around the beginning or end of the trajectories because compared to the points in the middle part of the trajectory, the beginning and end points have no prior or posterior points which means that they have less information for learning. This larger errors may be shown in following figures. Moreover, the defaulted initial and goal states of the DMPs-generated trajectories are the same as the demonstration ones. Also, the original trajectories represent the demonstration trajectories.

Here are the principles of selecting parameters: the gain parameters are chosen as the same ones shown in section 2.3. The hyperparameters are self-defined: the

number of basis functions is selected randomly from a modest range so that the DMPs-generated trajectory would have a shape similar to the shape of input system's trajectory. At that time, the shape of trajectory consists of discrete points in defined as the shape of path connecting all discrete points. For translational movements in 3D space, we will use point cloud to describe the shape. Furthermore, the number of generated points is picked randomly as long as the discrete DMPs-generated trajectory can provide enough information for a complete description of the motion.

3.1.1 Motion Generated by Formulated Functions

Translational Motion

According to the theories shown in section 2.1.3, DMPs is implemented on translational movements generated by given formulas. The input trajectory consists of the discrete points of position generated using the given functions. Also, the output trajectory is made up of discrete points.

Firstly, start from an extremely simple motion, which is a straight line defined by linear functions :

$$d_x(t) = 1.3t$$

$$d_y(t) = 1.3t$$

$$d_z(t) = 1.3t$$

Following the principles of parameter selection defined in the beginning part of chapter 3, and the corresponding hyperparameters are given:

$$N_{bfs} = 50$$

$$N_{pts} = 500$$

For analysing the results, the original and generated trajectories can be drawn:

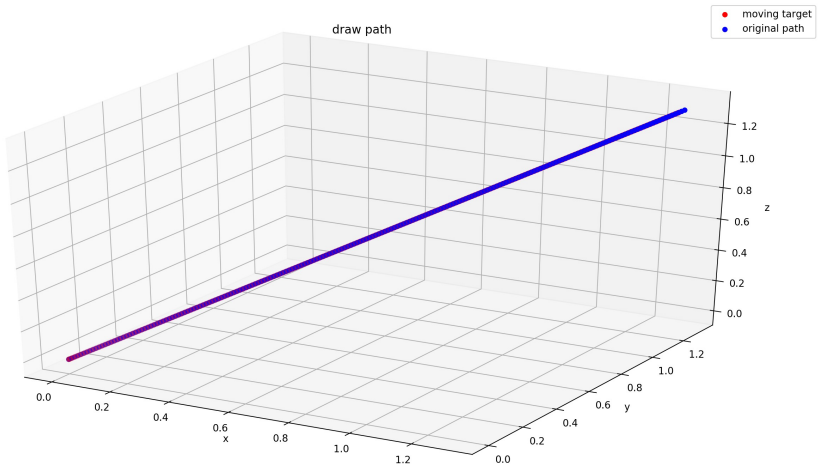


Figure 3·1: Visualized 3D original and DMPs-generated trajectories of linear functions.

To evaluate the results, plot the figures of displacements along three axes versus time:

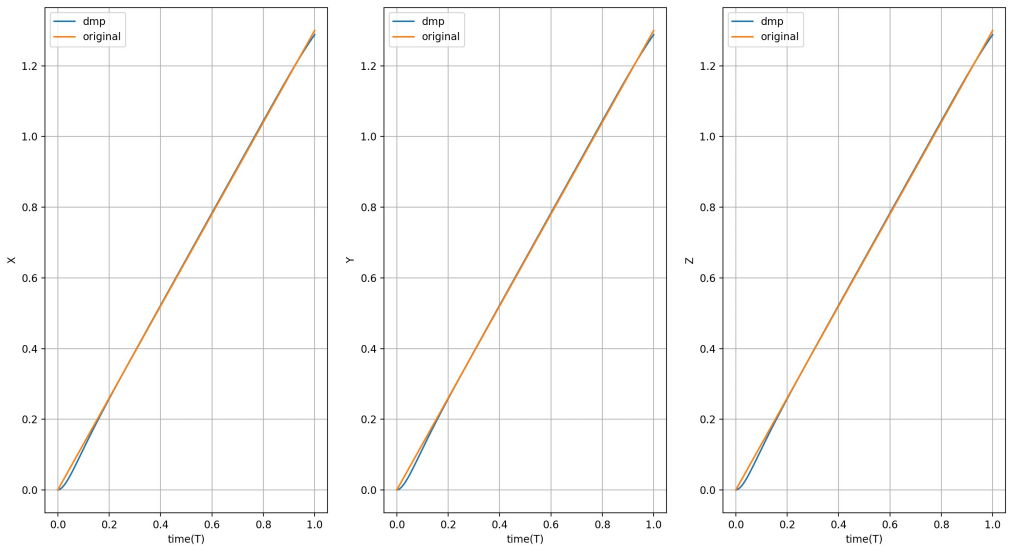


Figure 3·2: Schematic diagram of linear function-generated displacements along x,y,z axis versus time.

Then, we try with some periodic functions such as cosine or sine functions:

$$d_x(t) = \cos(0.08t)$$

$$d_y(t) = \sin(0.12t)$$

$$d_z(t) = -\sin(0.10t)$$

Following the principles of parameter selection defined in the beginning part of chapter 3, the corresponding hyperparameters are given:

$$N_{bfs} = 100$$

$$N_{pts} = 500$$

For analysing the results, the original and generated trajectories can be drawn:

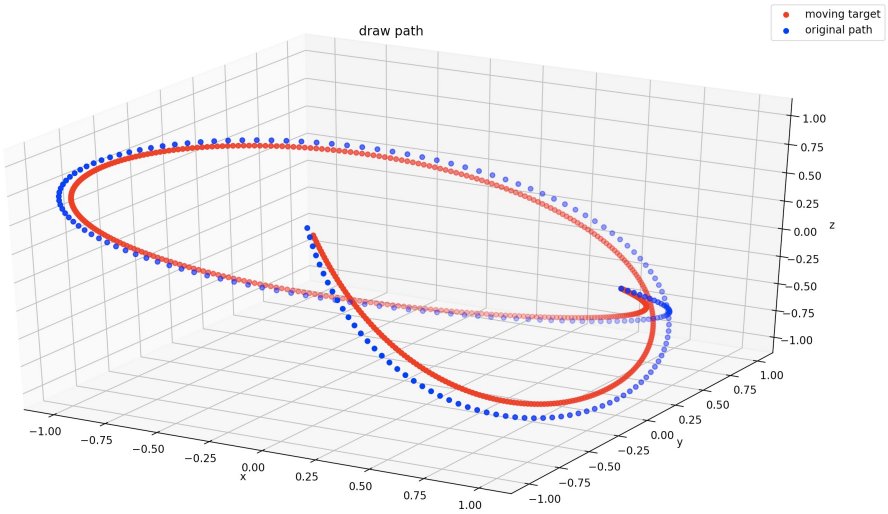


Figure 3·3: Visualized 3D original and DMPs-generated trajectories of sine or cosine functions.

To evaluate the results, plot the figures of displacements along three axes versus time:

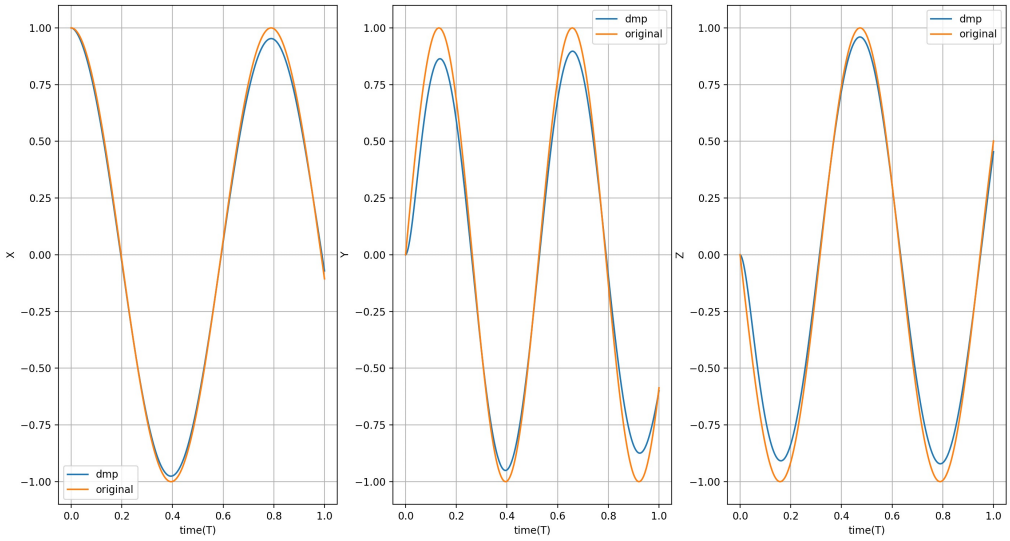


Figure 3·4: Schematic diagram of sine or cosine function-generated displacements along x,y,z axis versus time.

Ultimately, try with some nonperiodic and nonlinear functions:

$$d_x(t) = 0.2\cos(0.08t) + 0.9e^{-0.02t} + 0.16t^2$$

$$d_y(t) = 0.3\sin(0.12t) + 0.85e^{-0.02t} + 0.2t^2$$

$$d_z(t) = 0.4\sin(0.10t) + 0.8e^{-0.02t} + 0.24t^2$$

Following previous principles of parameter selection, the corresponding hyperparameters in section 2.3 are given:

$$N_{bfs} = 100$$

$$N_{pts} = 500$$

For analysing the results, the original and generated trajectories can be drawn:

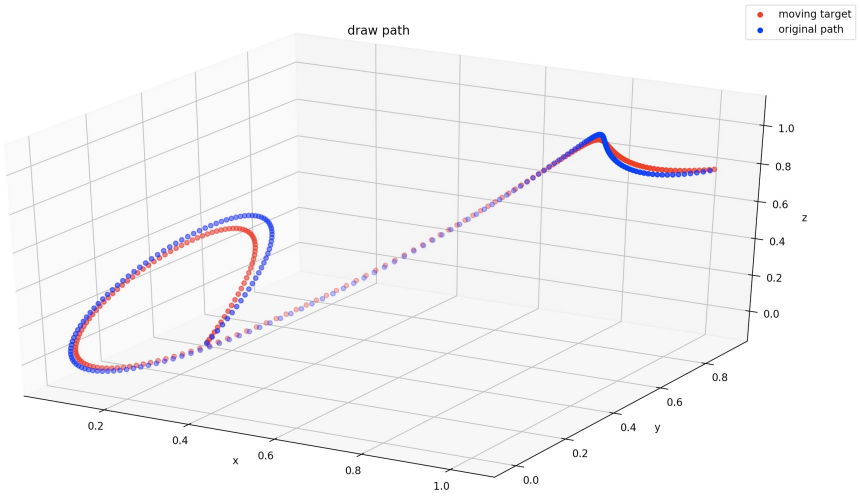


Figure 3·5: Visualized 3D original and DMPs-generated trajectories of the nonperiodic and nonlinear function.

To evaluate the results, plot the figures of displacements along three axes versus time:

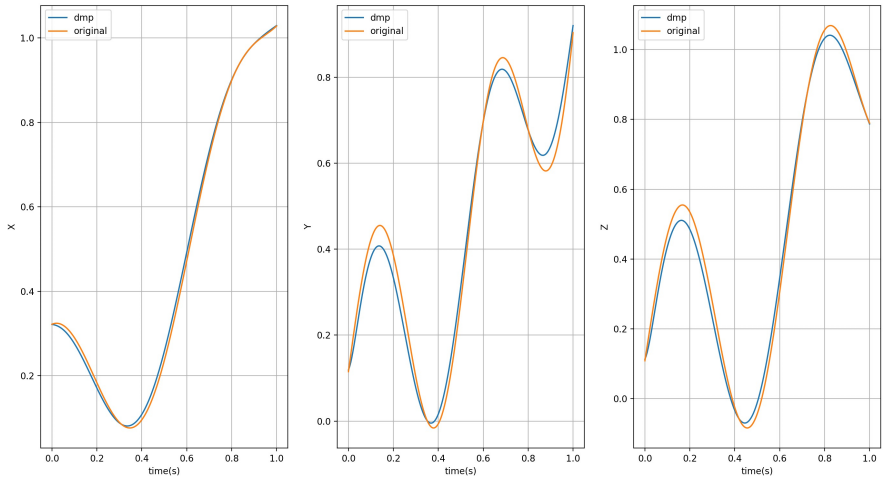
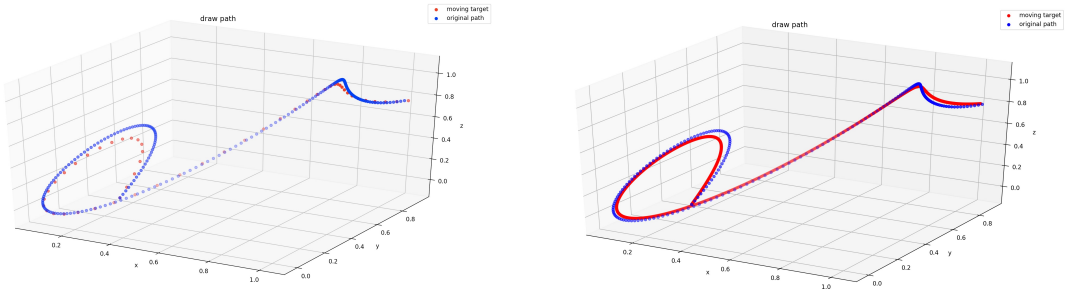


Figure 3·6: Schematic diagram of the nonperiodic and nonlinear function-generated displacements along x,y,z axis versus time.

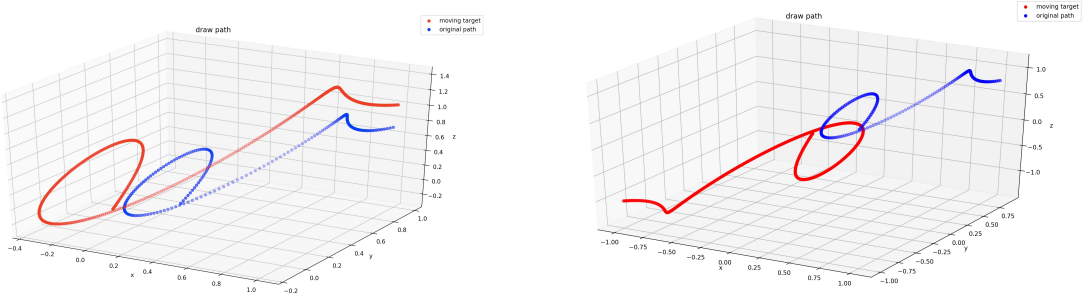
According to above figures, it can anticipated that the demonstration trajectories

consisted of function-generated discrete points and DMPs-generated ones would have similar qualitative behaviors. For the function-generated demonstration trajectories, we would be able to regenerate the trajectories which can accomplish the same or similar tasks. Furthermore, the trajectories can be scaled or translated in time or space without losing its features, which is described in section 2.1.2 :



(a) (b)

Figure 3·7: Nonperiodic and nonlinear function-generated trajectories scaled in time : (a) change $N_{pts} = 200$; and (b) change $N_{pts} = 1000$.



(a) (b)

Figure 3·8: Nonperiodic and nonlinear function-generated trajectories scaled and translated in space : (a) set initial position to be (0,0,0) and goal position to be (1,1,1); and (b) set initial position to be (0,0,0) and goal position to be (-1,-1,-1).

Likewise, it can be scaled both in time and in space :

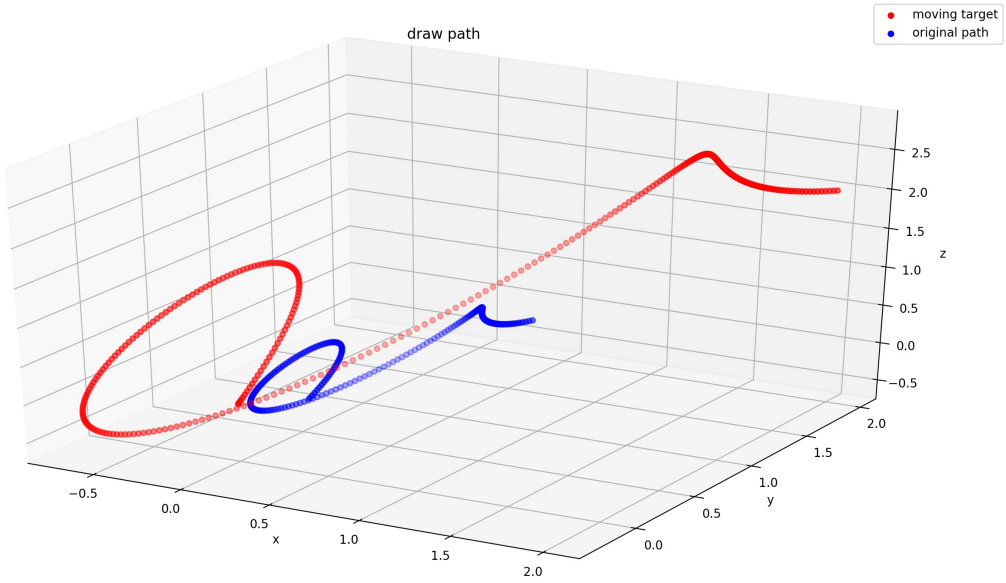


Figure 3·9: Nonperiodic and nonlinear function-generated trajectories scaled and translated both in space and in time : set initial position to be (0,0,0), goal position to be (2,2,2) and $N_{pts} = 300$.

Rotational Motion

For representing the rotation movement , rotation matrix is chosen. For displaying results more directly, we may need to convert generated rotation matrices into Euler angles using equations A.7, A.8 and A.9 . Using the functions in above part of section 3.1.1, we can generate the discrete points of Euler angles and convert them into the rotation matrices. Ultimately, the results are represented in rotation matrix form. Correlated equations can be found in section A.1.

Firstly, using linear functions to generate Euler angles:

$$\alpha(t) = 1.3t$$

$$\beta(t) = 1.3t$$

$$\gamma(t) = 1.3t$$

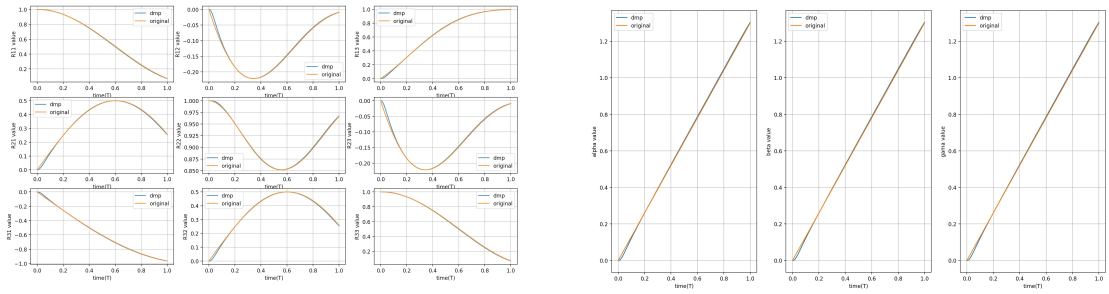
where Euler angles α, β and γ are defined in section A.1

Following previous principles of parameter selection, the corresponding hyperparameters in section 2.3 are given:

$$N_{dbfs} = 100$$

$$N_{dpts} = 200$$

For evaluating the results, the information of the original and generated rotation versus time are shown following:



(a) (b)

Figure 3·10: Linear function-generated orientation: (a) nine entries of the rotation matrix versus time; and (b) Euler angles versus time.

Then, try with some more periodic functions, such as sine or cosine functions to generate Euler angles:

$$\alpha(t) = \cos(0.08t)$$

$$\beta(t) = \sin(0.12t)$$

$$\gamma(t) = -\sin(0.10t)$$

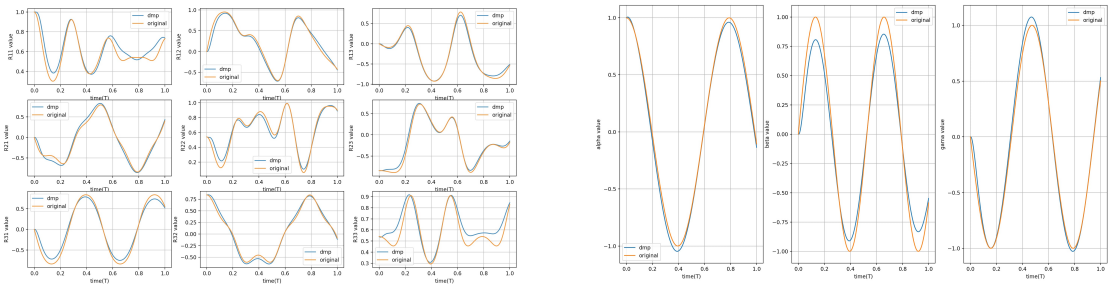
Following previous principles of parameter selection, the corresponding hyperpa-

rameters in section 2.3 are given:

$$N_{dbfs} = 100$$

$$N_{dpts} = 200$$

For evaluating the results, the information of the original and generated rotation versus time are shown following:



(a) (b)

Figure 3·11: Sine or cosine function-generated orientation: (a) nine entries of the rotation matrix versus time; and (b) Euler angles versus time.

Ultimately, using some nonperiodic and nonlinear functions to generate Euler angles:

$$\alpha(t) = 0.2\cos(0.08t) + 0.9e^{-0.02t} + 0.16t^2$$

$$\beta(t) = 0.3\sin(0.12t) + 0.85e^{-0.02t} + 0.2t^2$$

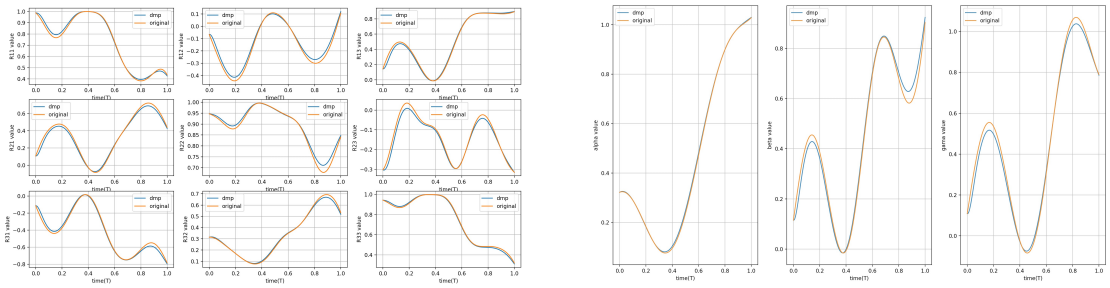
$$\gamma(t) = 0.4\sin(0.10t) + 0.8e^{-0.02t} + 0.24t^2$$

Following previous principles of parameter selection, the corresponding hyperparameters in section 2.3 are given:

$$N_{dbfs} = 100$$

$$N_{dpts} = 200$$

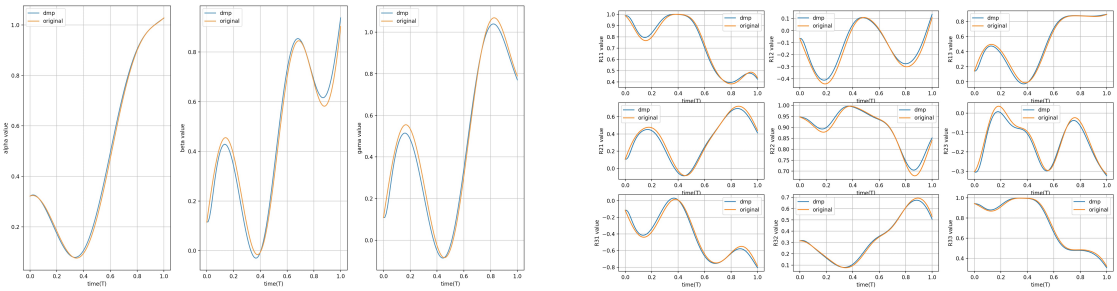
For evaluating the results, the information of the original and generated rotation versus time are shown following:



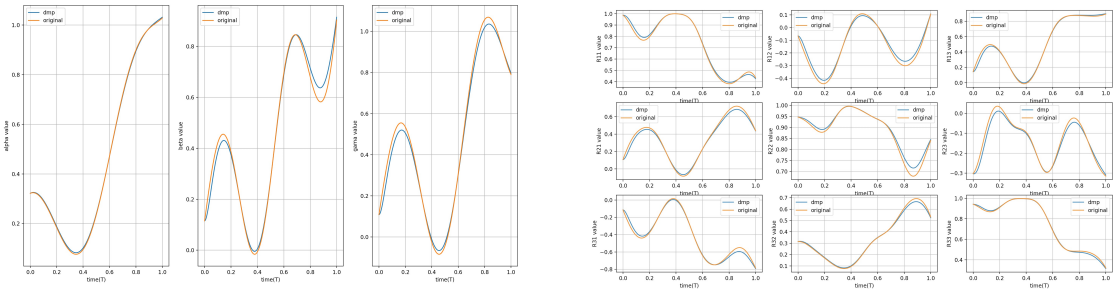
(a) (b)

Figure 3·12: Nonperiodic and nonlinear function-generated orientation: (a) nine entries of the rotation matrix versus time; and (b) Euler angles versus time.

According to above results, it can anticipate that the imitation algorithm can work for most function-generated rotation movements since the errors are acceptable. The generated trajectories are almost overlaid to the original data, which means that they can manage to accomplish the same or similar tasks. The slightly larger errors around the beginning or ending points can be explained with lack of necessary information. Similar to section 3.1.1, the orientation also has potential to retain their qualitative behaviors if numerically modified or scaled in space or time:

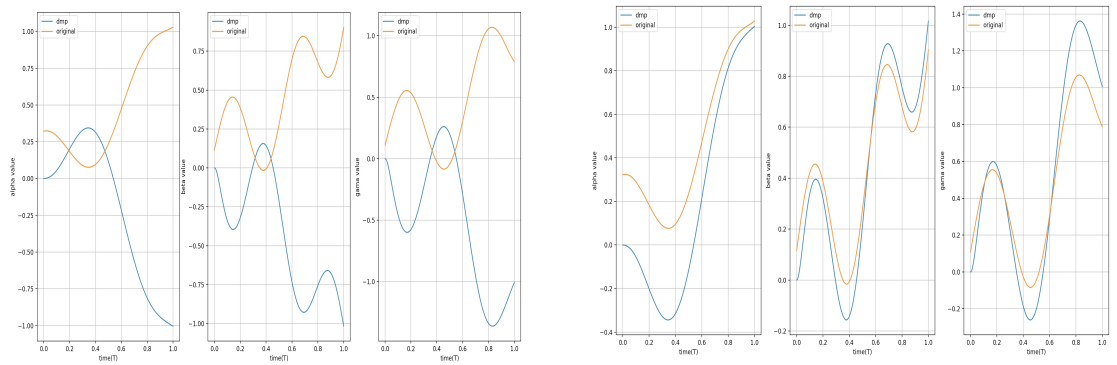


(a) (b)



(c) (d)

Figure 3·13: Nonperiodic and nonlinear function-generated orientation scaled in time : (a) nine entries of the rotation matrix versus time when $N_{dpts} = 100$; (b) Euler angles versus time when $N_{dpts} = 100$; (c) nine entries of the rotation matrix versus time when $N_{dpts} = 1000$;and (d) Euler angles versus time when $N_{dpts} = 1000$.

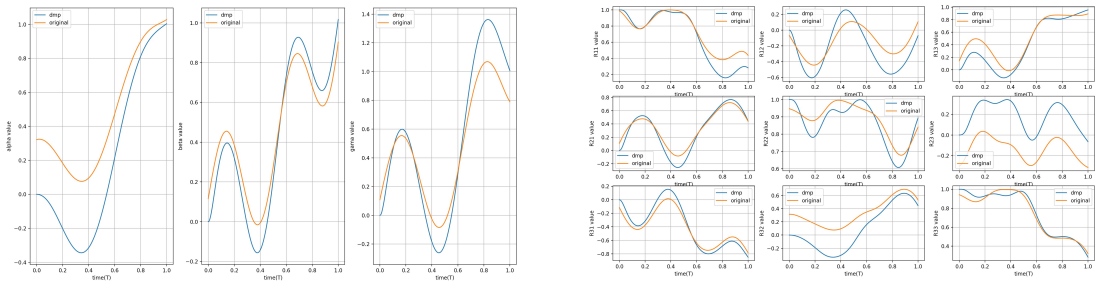


(a) (b)

Figure 3·14: Nonperiodic and nonlinear function-generated orientation scaled and translated in space, set initial Euler angles to be (0,0,0) : (a) nine entries of the rotation matrix versus time when setting goal Euler angles to be (-1,-1,-1) ; and (b) Euler angles versus time when setting goal Euler angles to be (1,1,1).

The rotation matrix are not shown since it is kind of meaningless when initial and goal states are change significantly.

Likewise, it also works when scaling or translating in both space and time:



(a) (b)

Figure 3·15: Nonperiodic and nonlinear function-generated orientation, set initial Euler angles to be (0,0,0) , goal Euler angles to be (1,1,1) and $N_{dpts} = 500$: (a) nine entries of the rotation matrix versus time; and (b) Euler angles versus time.

3.1.2 Motion Generated by PC Mouse Implementation

According to the results given in section 3.1, most function generated trajectories are able to be successfully regenerated using DMPs. Also, almost real-life trajectories are derivable. Therefore, we can utilize the PC mouse combining with VREP to visualize the qualitative behaviors of some real-life random movements. We can also evaluate the DMPs models' availability at the same time.

In VREP, the translational motion is controlled by given coordinates and the rotational motion can controlled by given Euler angles, rotation matrix or quaternion. The robot Baxter is used for data extraction and DMPs implementation. The end-effector is the control object. Meanwhile, my output which consists of both translational and rotational information will be executed using the IK solver built in the model. For rotational movements, quaternion is utilized for execution and Euler angles are utilized for more direct display. Following is a figure of the robot Baxter shown is VERP:

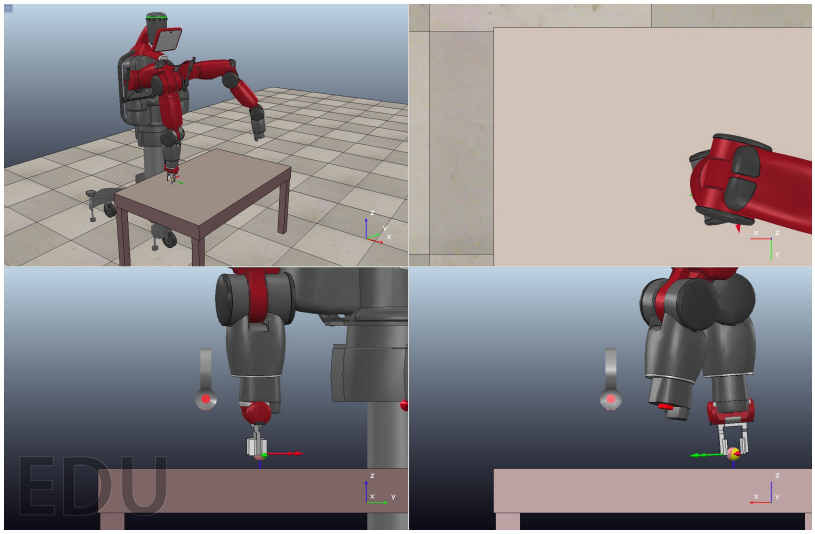
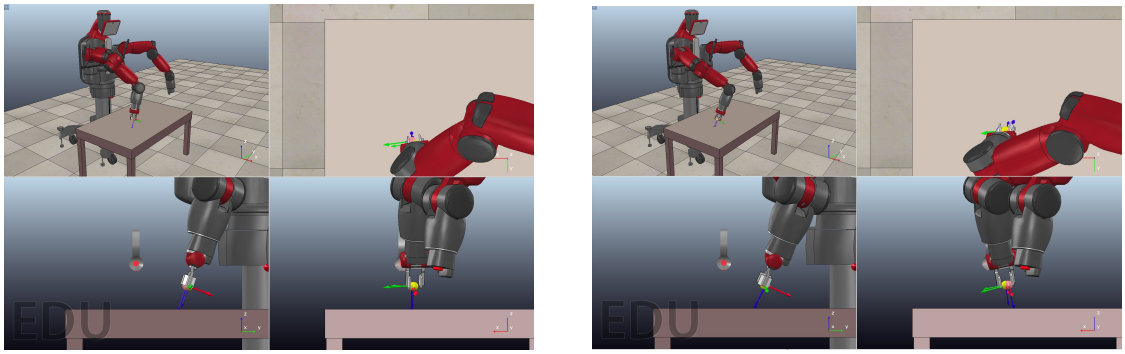


Figure 3·16: Robot Baxter with end-effector's translational and rotational information shown .

Given a motion generated by PC mouse, extract data and substitute in the algorithm described in section 2.3, generate movement calculated using DMPs. Corresponding videos can be found following:



(a) (b)

Figure 3·17: Videos of the sample motion: (a) original movement generated by PC mouse; and (b) DMPs-generated movement.

Following previous principles of parameter selection, the corresponding hyperpa-

rameters in section2.3 are:

$$N_{bfs} = 800$$

$$N_{dbfs} = 200$$

$$N_{pts} = 500$$

Substitute extracted data into DMPs, the trajectories can be obtained :

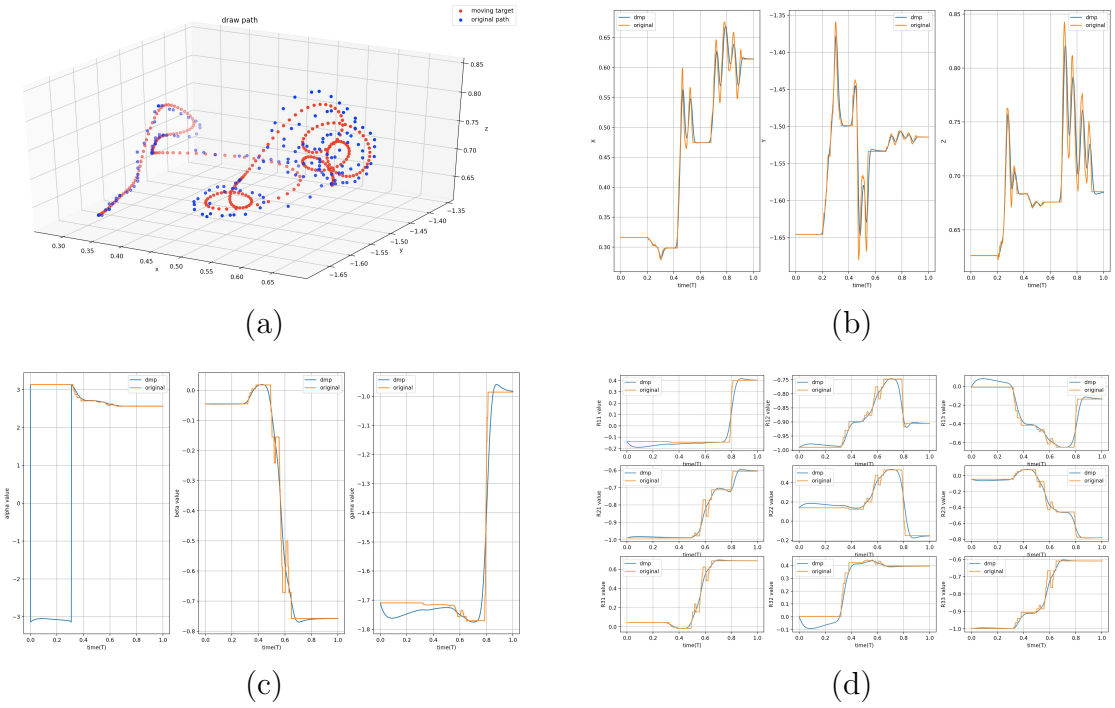
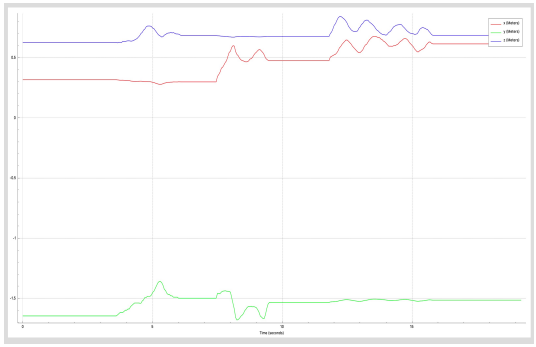


Figure 3·18: The translational and rotational information of the PC mouse-generated movement: (a) 3D visible trajectories; (b) displacements versus time along x,y and z axis; (c) Euler angles versus time; and (d) nine rotation matrix entries versus time.

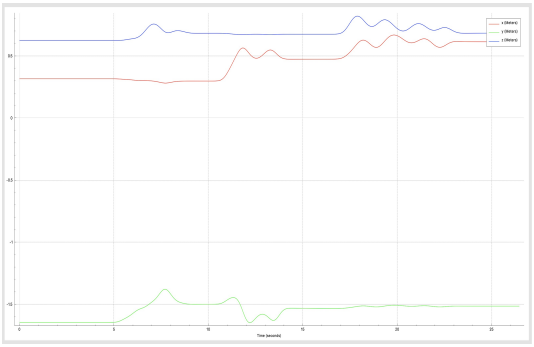
We can observe that the errors of figures 3·18 (c) and (d) are larger than the other ones at the beginning part and ending part. This is because compared to the points in the middle part of the trajectory, the beginning and ending points have no prior or posterior points which means that they have restricted information for

learning. Nevertheless, the errors seems acceptable, we can evaluate the qualitative behaviors in imitation learning of task-oriented movements. In addition, the Euler angles are calculated from the rotation matrices and their definition can be found in appendix section A.1. For the first Euler angle, its range is $(-\pi, \pi]$. And it may lead to some misunderstanding when representing the Euler angles, for instance, the difference between $-\pi + 0.1$ and $\pi - 0.1$ should be 0.2 rather than $2\pi - 0.2$. Therefore, the error of the Euler angle is acceptable since it is actually fluctuating around pi. To evaluate the qualitative behaviors of the random movements in detail, we may need some videos which are shown in figure 3·18 to visualize the movements.

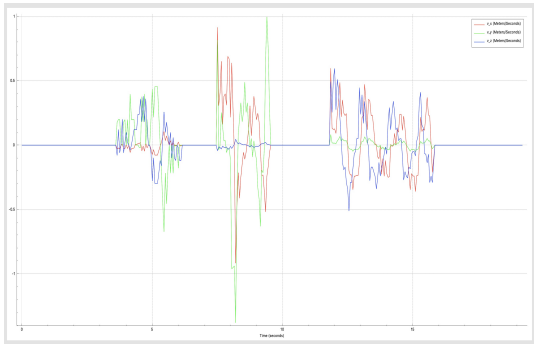
In addition, according to the figures 3·18 , the algorithm can be anticipated to work for this random movement's imitation learning in simulation. For testing availability in real world, we can get the real-time data from VREP monitor and plot the figures of coordinates, velocities and Euler angles versus time are plotted:



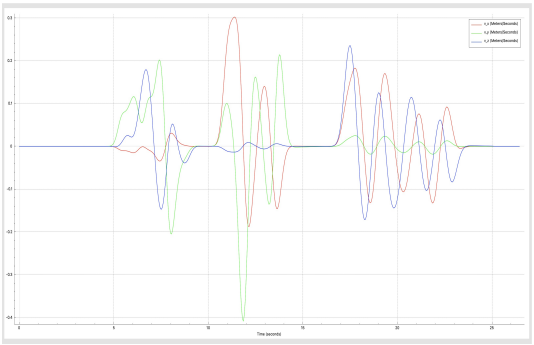
(a)



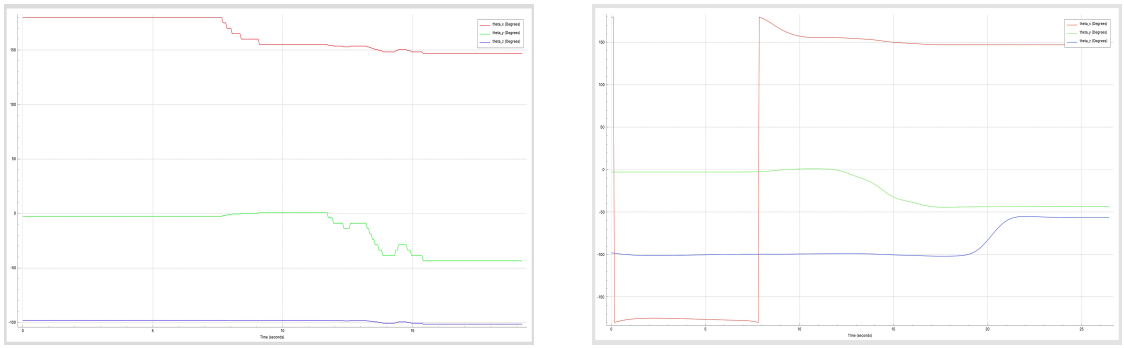
(b)



(c)



(d)



(e) (f)

Figure 3·19: Data of the PC mouse motion from real-time monitor in VREP : (a) 3D coordinates of original movement generated by joystick; (b) 3D coordinates of DMPs-generated movement; (c)scaled velocities of original movement generated by joystick; (d)scaled velocities of DMPs-generated movement; (e) Euler angles of original movement generated by joystick; and (f) Euler angles of DMPs-generated movement.

According to the above results, it can see that DMPs has plenty of potential to accomplish this random motion's imitation learning in real life.

Nevertheless, for some motion, PC mouse may not successfully complete the real-life task due to the high accelerations of mouse movements. Therefore, Joystick controller becomes an alternative since it can complete relatively complicated tasks with smaller acceleration. More detailed discussion is shown in 3.2.

3.2 Implementation on Baxter Robot

All movements in section 3.2 are generated by joystick controller. The control targets in this section are cubes, which is sized as height of 4.5 cm, length of 4.5 cm and width of 2.8 cm. A mapping is built to convert the electric signal extracted from joystick controllers into the differences of coordinates and Euler angles every time step . Data are extracted from VREP via ROS joy package. Following is an example of extracting data from a complex motion discussed in section 3.2.2:

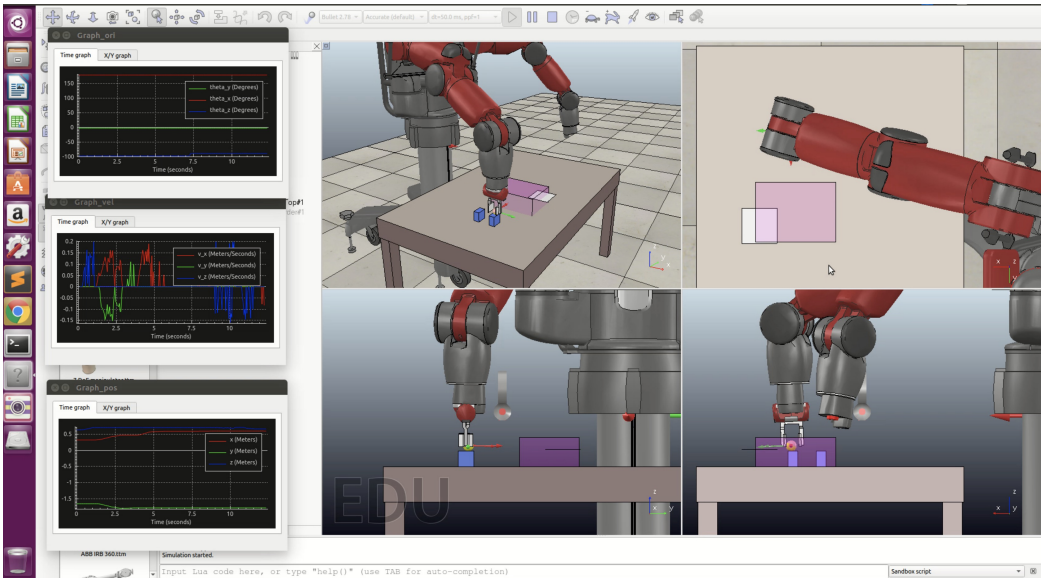


Figure 3·20: Sample video of extracting data from a complex motion.

The simulator's physics engine is based on Newton's laws of motion and it is sensitive to forces, inertia and materials. As mentioned before, the task completeness is the only thing we care about because people can accomplish similar tasks in extremely different ways. Furthermore, the time step of the simulation is set to be 50 milliseconds. Compared to the data extraction, the implementation generally take shorter time. Also, it is assumed that the joystick is controlled manually by an inexperienced person when extracting data. It may lead to some visible spikes when extracting the data discretely from the demonstration trajectories. The results of velocities and accelerations are smoother because they are interpolated (DeWolf, 2013b) and trained in weighted learning algorithm.

3.2.1 Simple Motion

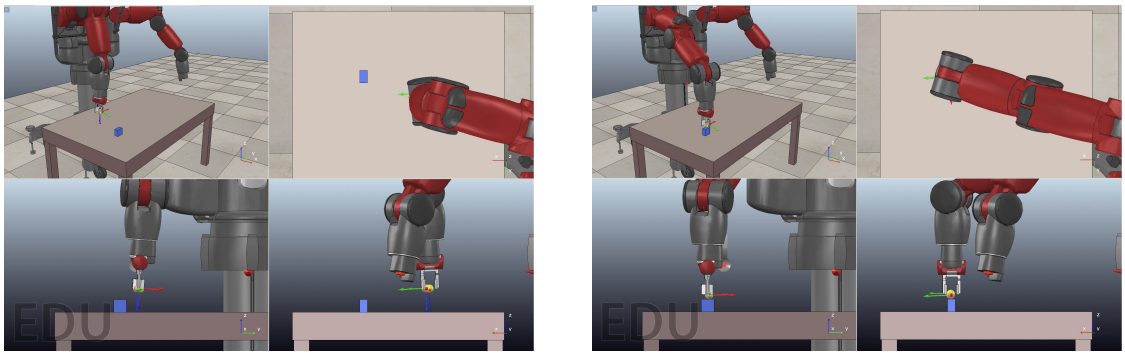
For simple motion, the task is to make the robot arm go and ready to grasp an object denoting the position and orientation of end-effector as the goal state. Following the principles of parameter selection, the corresponding hyperparameters in section2.3

are:

$$N_{bfs} = 50$$

$$N_{dbfs} = 50$$

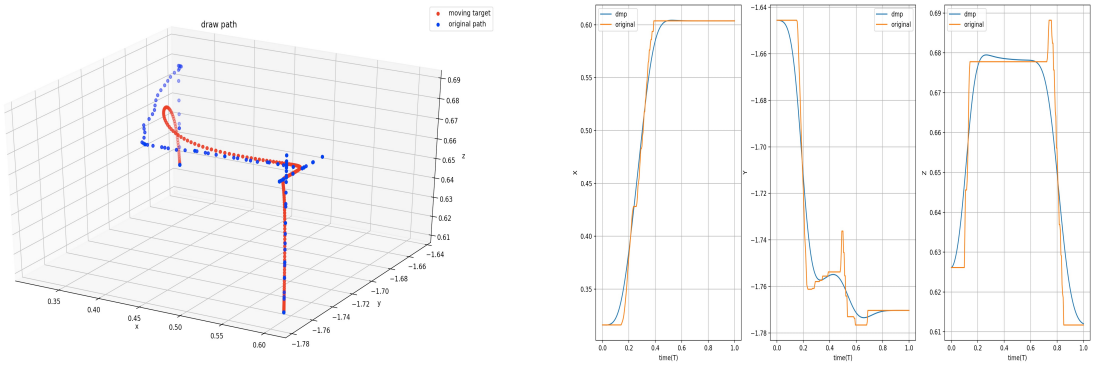
$$N_{pts} = 100$$



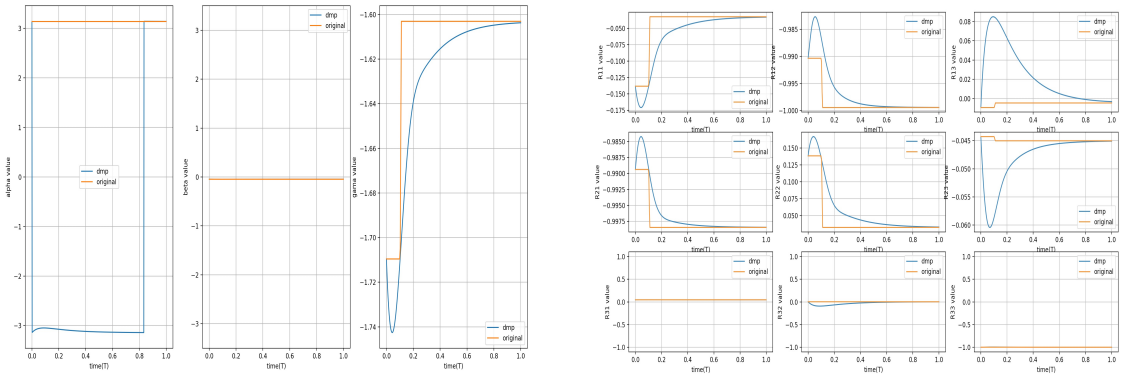
(a) (b)

Figure 3·21: Videos of the simple motion case : (a) original movement generated by joystick ; and (b) DMPs-generated movement.

Plotting the figures of translational and rotational trajectories, we can see how DMPs accomplish the learning process and recognize the error roughly:



(a) (b)

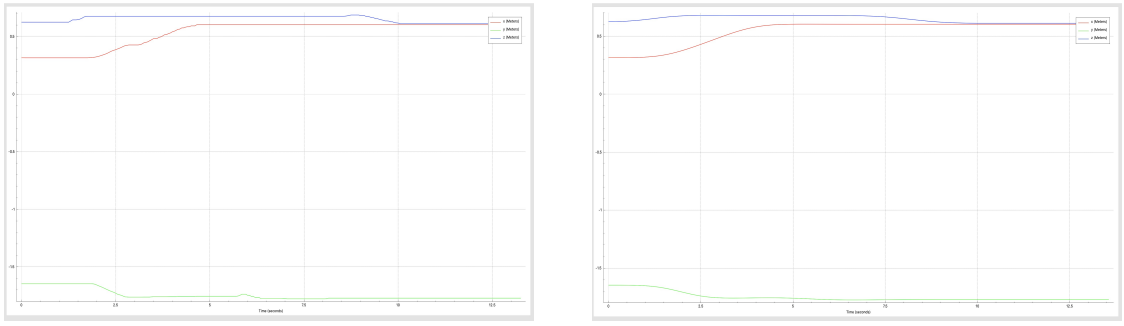


(c) (d)

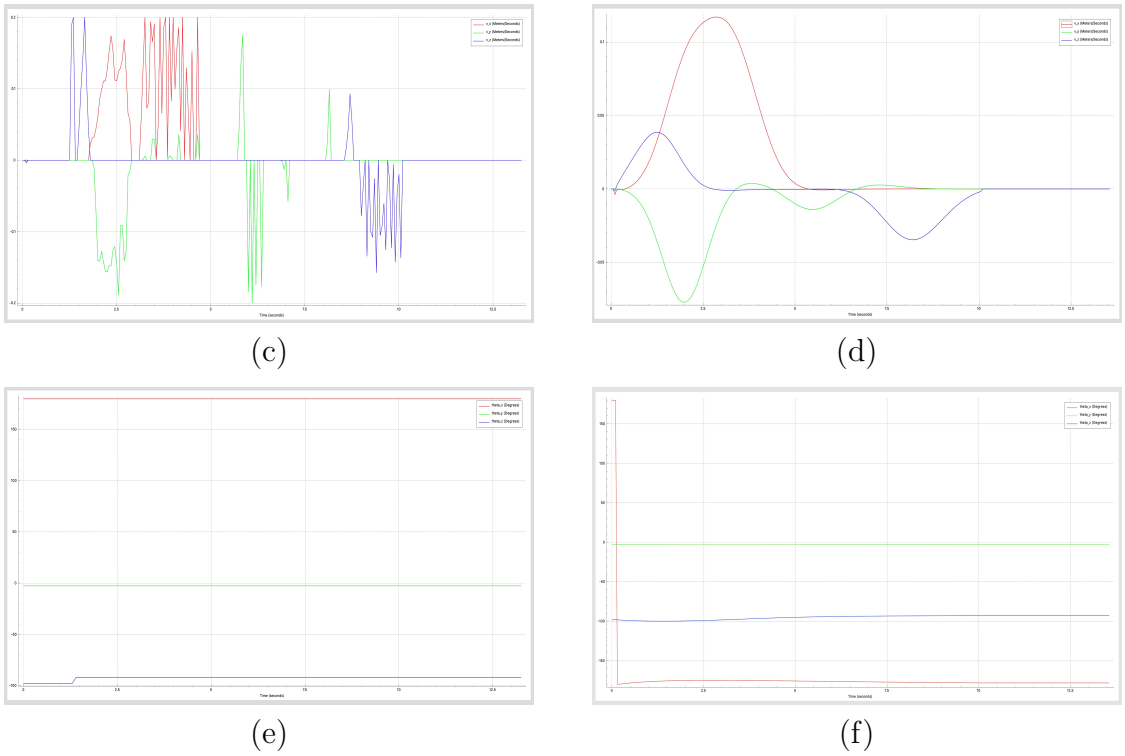
Figure 3·22: The translational and rotational information of the simple motion case: (a) 3D visible trajectories; (b) displacements versus time along x,y and z axis; (c) Euler angles versus time; and (d) nine rotation matrix entries versus time.

Although it seems that it has relatively large errors in figure 3·22 (b), given the videos shown in figure 3·21 , the DMPs-generated trajectory manages to imitate this simple joystick-generated human motion. The larger errors are acceptable and it has the same reasons as the errors in figure 3·18. Also, rather than the random movement discussed in section 3.1.2 , this movement has a conspicuous task which is to reach the position of an object and the orientation ready for grabbing the object.

To test the availability in real world, we can get the real-time data from VREP monitor and draw the figures of coordinates, velocities and Euler angles versus time :



(a) (b)



(c) (d)

(e) (f)

Figure 3·23: Data of the simple motion case from real-time monitor in VREP : (a) 3D coordinates of original movement generated by joystick; (b) 3D coordinates of DMPs-generated movement; (c)scaled velocities of original movement generated by joystick; (d)scaled velocities of DMPs-generated movement; (e) Euler angles of original movement generated by joystick; and (f) Euler angles of DMPs-generated movement.

According to the above results, it can see that DMPs has plenty of potential to accomplish this simple task-oriented imitation learning in real life. The Euler angles in figure 3·23 may looks extremely different because of the Euler angle representation's drawback, nevertheless, the rotation matrix shown in figure 3·22 indicates availability.

3.2.2 Complex Motion

In this part, there are two cases of complex motions to be analyzed. The second one has higher complexity and longer running time compared to the first one. If we can divide a complicated motion into several simple movements, we would always be

able to solve the imitation learning problem using the algorithm shown in section 3.2.1. Nevertheless, sometimes we cannot divide one motion when some important information are dismissed. For instance, for the motion which includes grasping and moving the object, if the position and orientation of object is not given, we cannot separate the movement. Therefore, we regard a complex trajectory as an independent trajectory and do not divide the trajectory into several parts for retaining its integrity sometimes.

The first case is to move one object into the target area. Specifically, the object in the VREP simulation is the cube described before.

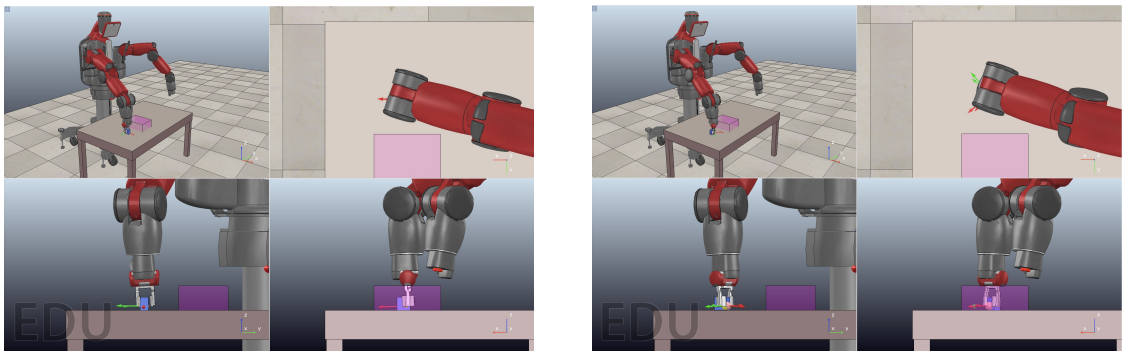
Following previous principles of parameter selection, the corresponding hyperparameters in section 2.3 are:

$$N_{bfs} = 50$$

$$N_{dbfs} = 200$$

$$N_{pts} = 200$$

For evaluating the results of the task-oriented imitation learning directly , the VREP simulations of original and DMPs-generated trajectories are shown following:



(a) (b)

Figure 3·24: Videos of the complex motion case 1 : (a) original movement generated by joystick ; and (b) DMPs-generated movement.

Plotting the figures of translational and rotational trajectories, we can see how DMPs accomplish the learning process and recognize the error roughly:

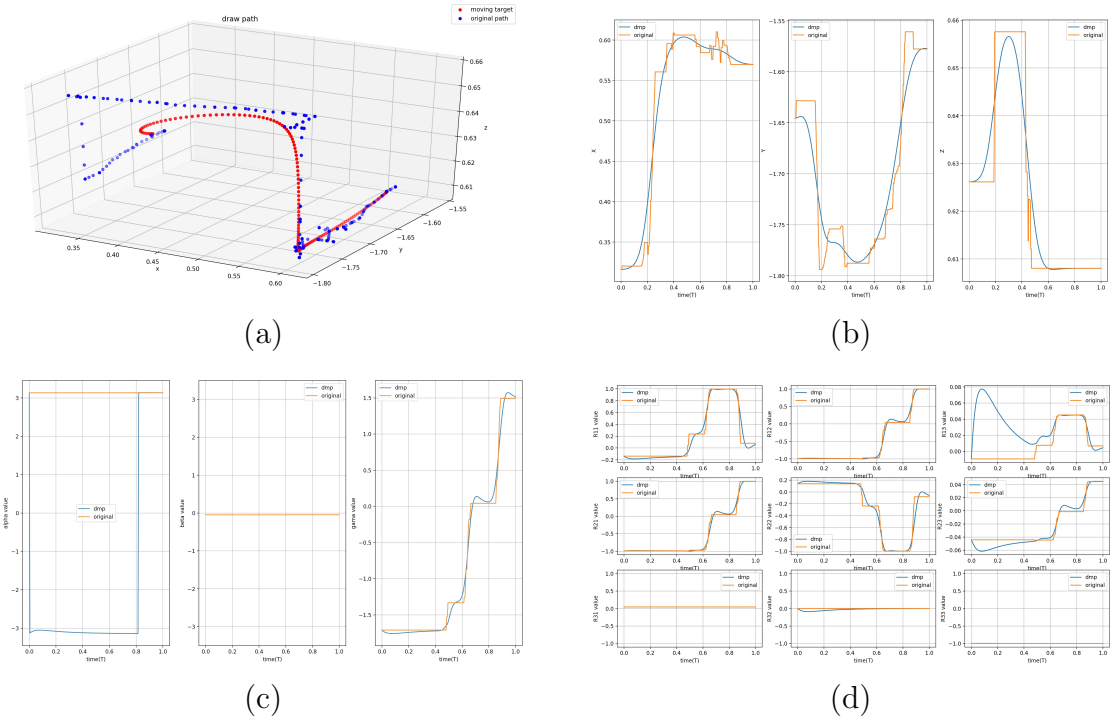
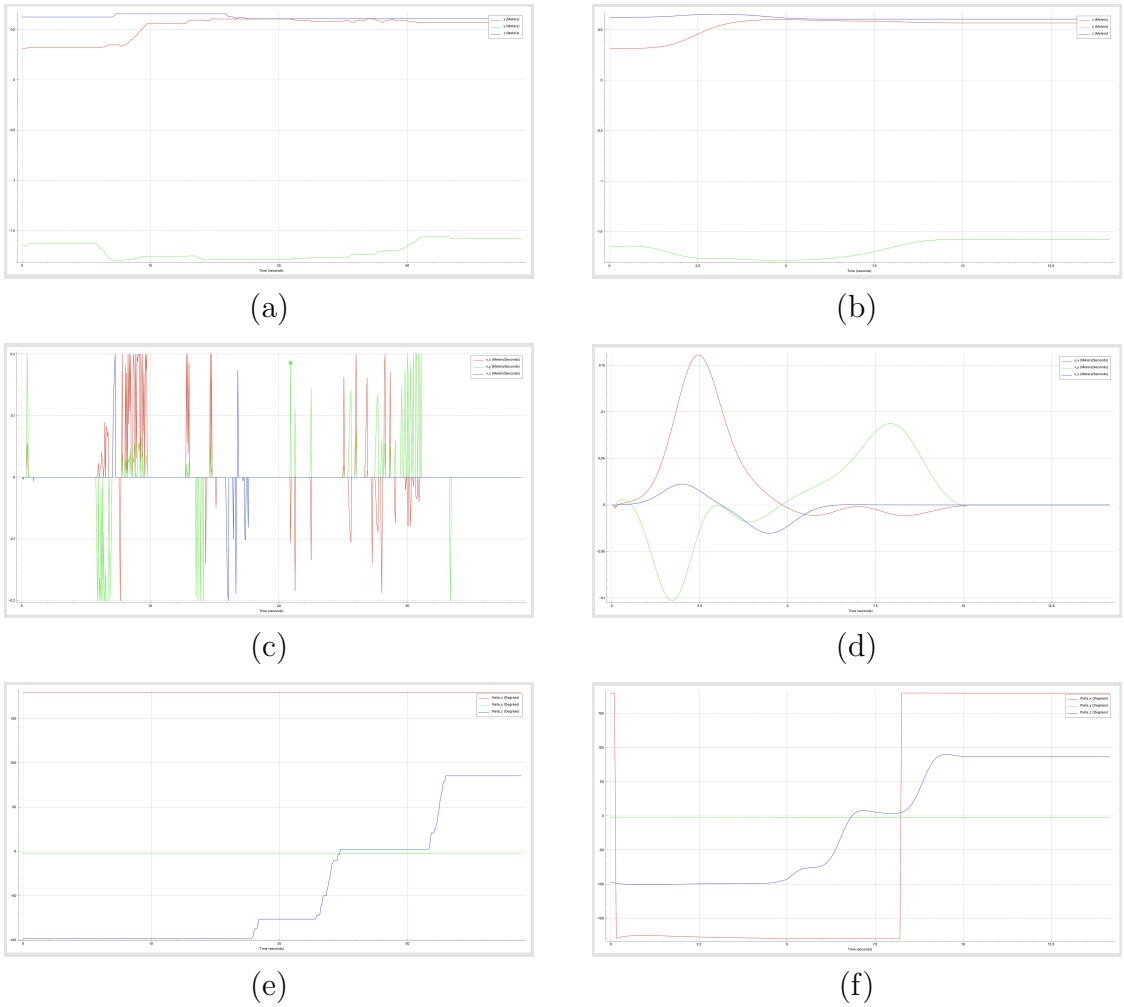


Figure 3·25: The translational and rotational information of the complex motion case 1: (a) 3D visible trajectories; (b) displacements versus time along x,y and z axis; (c) Euler angles versus time; and (d) nine rotation matrix entries versus time.

Although it seems that it has relatively large errors in figure 3·25 (b), given the videos shown in figure 3·24 , the DMPs-generated trajectory manages to imitate this simple joystick-generated human motion. The larger errors are acceptable and it has the same reasons as the errors in figure 3·18. Also, this movement has a conspicuous task which is to moving an object into target area without knowing any information about where the object is.

To test the availability in real world, we can get the real-time data from VREP monitor and draw the figures of coordinates, velocities and Euler angles versus time :



(a) (b) (c) (d) (e) (f)

Figure 3·26: Data of the complex motion case 1 from real-time monitor in VREP : (a) 3D coordinates of original movement generated by joystick; (b) 3D coordinates of DMPs-generated movement; (c)scaled velocities of original movement generated by joystick; (d)scaled velocities of DMPs-generated movement; (e) Euler angles of original movement generated by joystick; and (f) Euler angles of DMPs-generated movement.

According to the above results, it can see that DMPs has plenty of potential to accomplish task-oriented imitation learning in real life. Although the Euler angles in 3·26 may imply that the DMPs-generated movement cannot accomplish the task since it comes across the gimbal lock issue, the rotation matrix shown in figure 3·25

indicates the availability discussed in previous work (Ude et al., 2014).

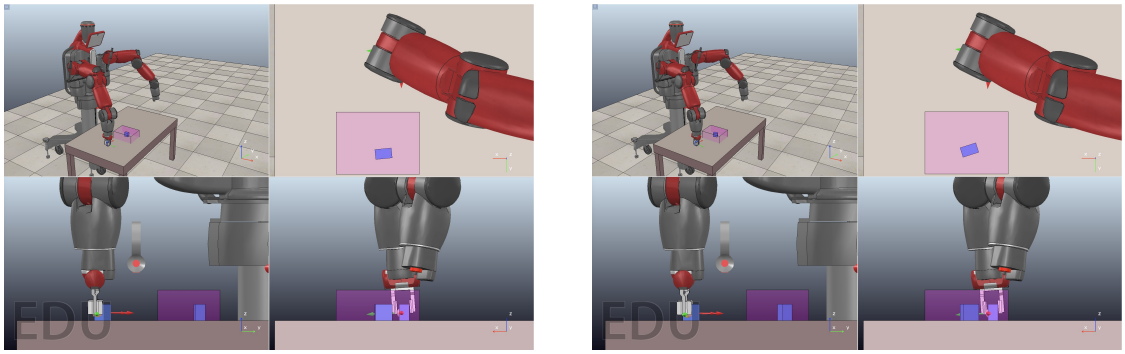
The second case is to move two objects into the target area one by one. Specifically, the objects in the VREP simulation are cubes. This case is to show the availability of DMPs when the trajectories have relatively high complexity. Following previous principles of parameter selection, the corresponding hyperparameters in section 2.3 are:

$$N_{bfs} = 500$$

$$N_{dbfs} = 200$$

$$N_{pts} = 600$$

For evaluating the results of the task-oriented imitation learning directly , the VREP simulations of original and DMPs-generated trajectories are shown following:



(a) (b)

Figure 3·27: Videos of the complex motion case 2: (a) original movement generated by joystick ; and (b) DMPs-generated movement.

Plotting the figures of translational and rotational trajectories, we can see how DMPs accomplish the learning process and recognize the error roughly:

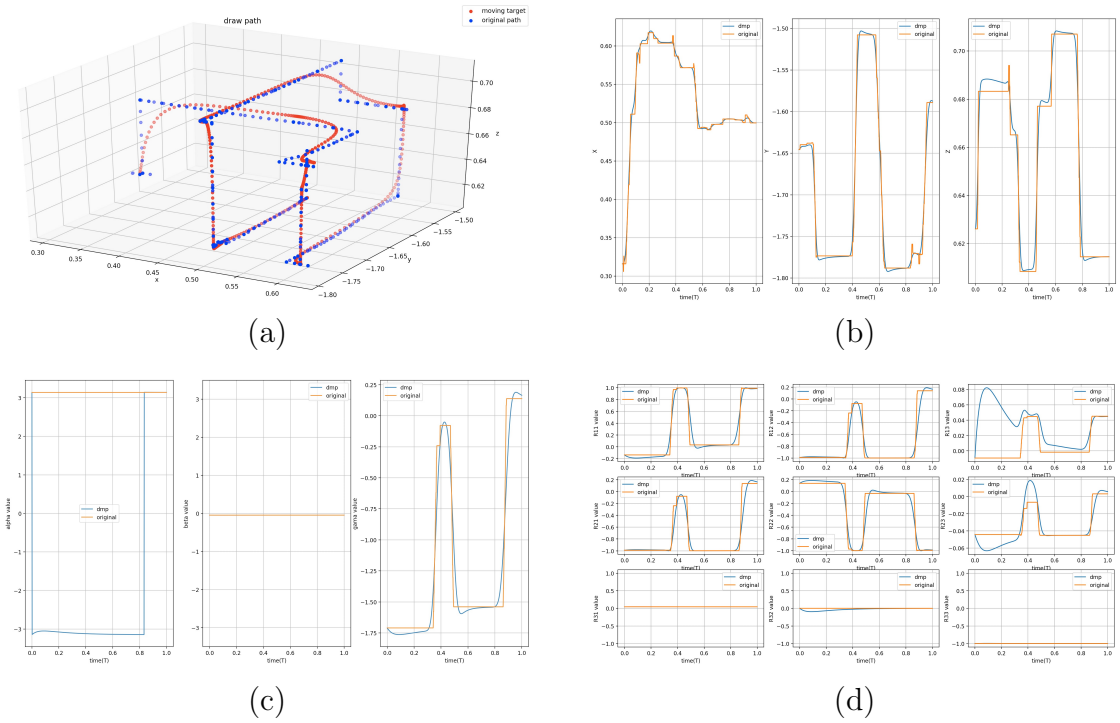
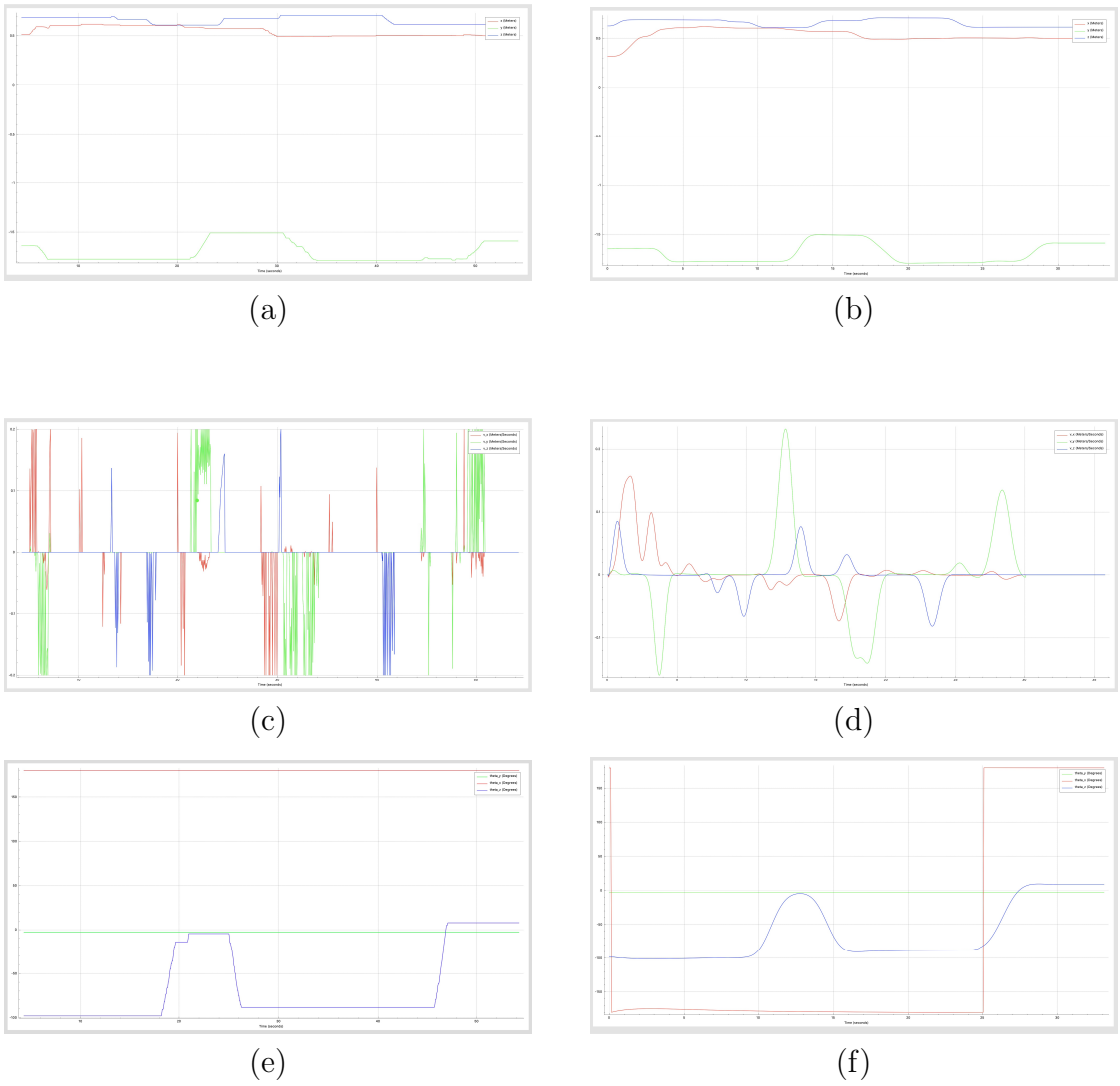


Figure 3·28: The translational and rotational information of the complex motion case 2: (a) 3D visible trajectories; (b) displacements versus time along x,y and z axis; (c) Euler angles versus time; and (d) nine rotation matrix entries versus time.

Although it seems that it has relatively large errors in figure 3·28 (b), given the videos shown in figure 3·27 , the DMPs-generated trajectory manages to imitate this simple joystick-generated human motion. The larger errors are acceptable and it has the same reasons as the errors in figure 3·18. Also, this movement has a conspicuous task which is to moving objects into target area without knowing any information about where the objects are. Compared to the first case, this one has higher complexity and longer running time which indicates the algorithm can be extended to some more complicated case.

To test the availability in real world, we can get the real-time data from VREP monitor and draw the figures of coordinates, velocities and Euler angles versus time :



(a) (b)

(c) (d)

(e) (f)

Figure 3·29: Data of the complex motion case 2 from real-time monitor in VREP : (a) 3D coordinates of original movement generated by joystick; (b) 3D coordinates of DMPs-generated movement; (c)scaled velocities of original movement generated by joystick; (d)scaled velocities of DMPs-generated movement; (e) Euler angles of original movement generated by joystick; and (f) Euler angles of DMPs-generated movement.

According to the above results, it can anticipate that DMPs has potential to accomplish task-oriented imitation learning in real life with relatively high complexity. Nevertheless, it may come across some problem if the orientation changes too sharply,

but it could be overcome by modifying the basis functions (Ginesi et al., 2019) .

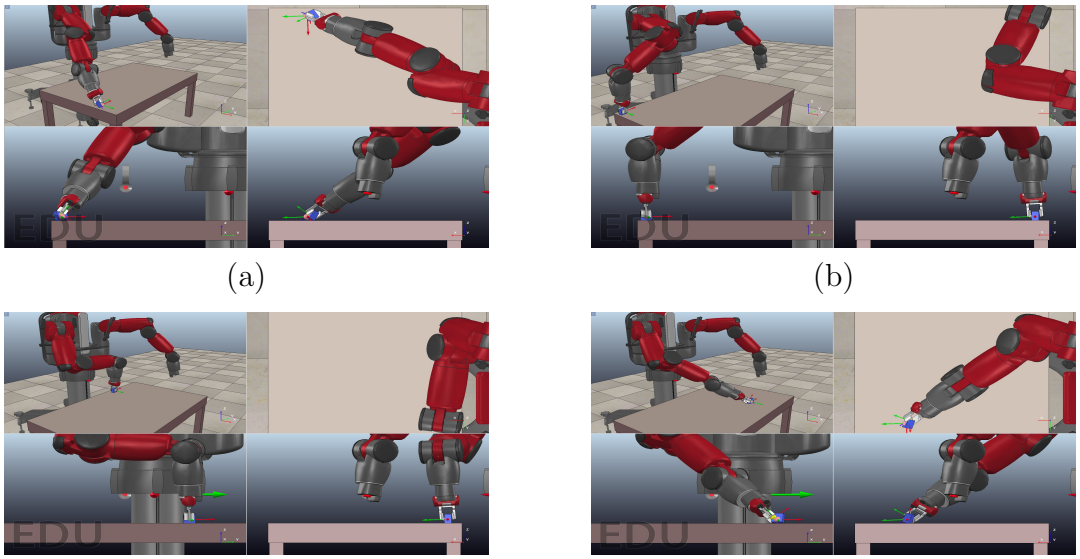
3.2.3 Motions with Modification

The parameters of the algorithm in section 3.1 for different motions retain the same value. In this section, we would like to test the capability of resisting outside disturbance.

Simple Motion with Modification

The modifications are based on data extracted from the simple motion case shown in figure 3·21. To show its flexibility, the goal state becomes a self-defined parameter.

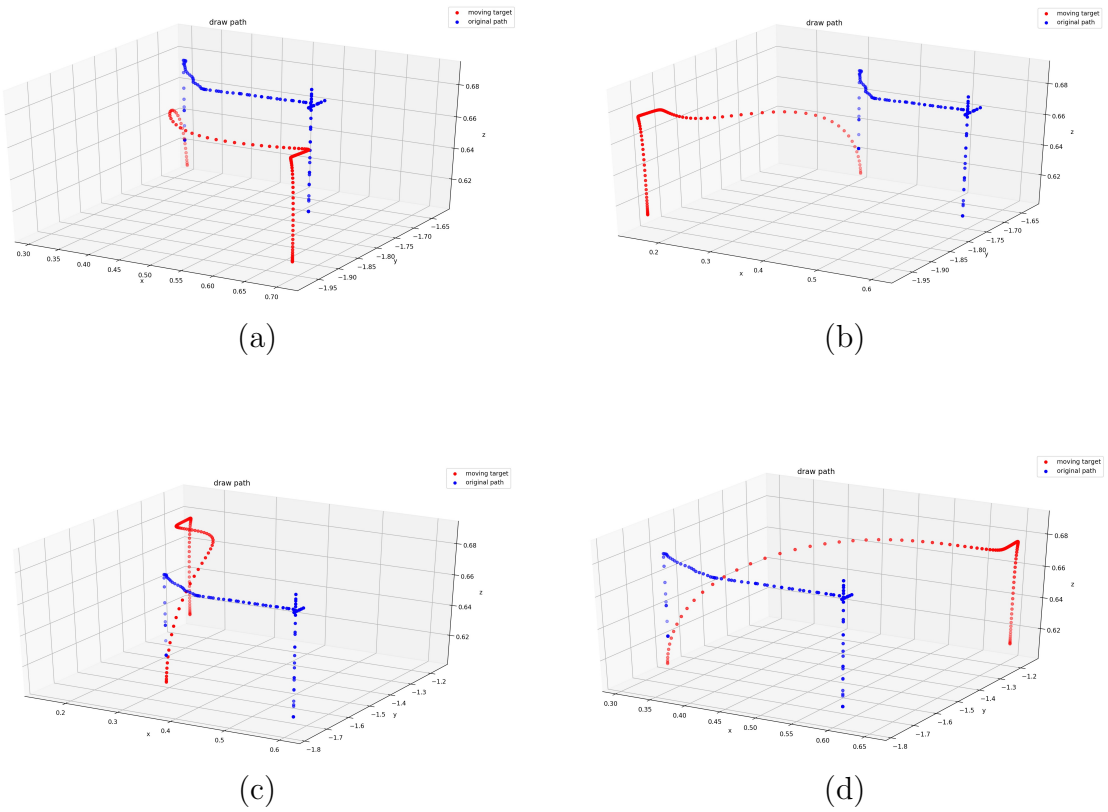
For gripper's movements, since the object is on a platform which is a table. Therefore, any differences in vertical direction cannot be made because of the gravity. As long as the robot Baxter does not have self-collision and the control targets have no collision with the grippers when moving, the object can be placed in any arbitrary position of the robot Baxter's work space:



(a) (b) (c) (d)

Figure 3·30: Modified motions of the simple motion case at: (a) corner 1; (b) corner 2; (c) corner 3; and (d) corner 4.

The visible 3D trajectories of above figures are drawn:

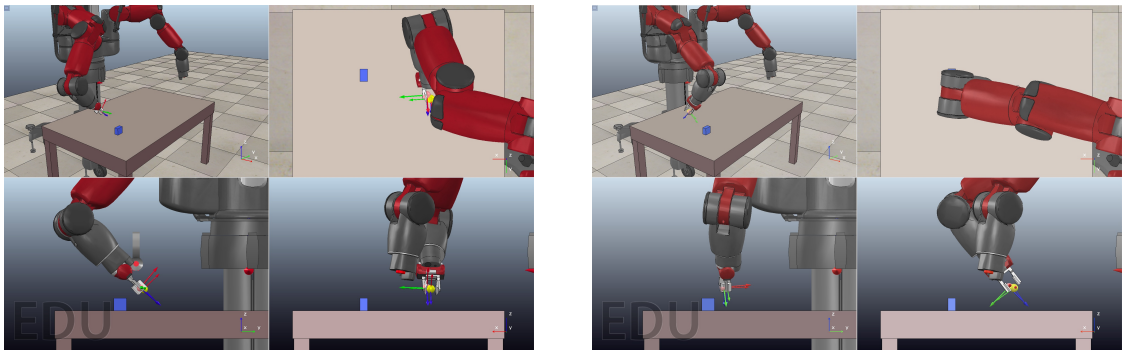


(a) (b)

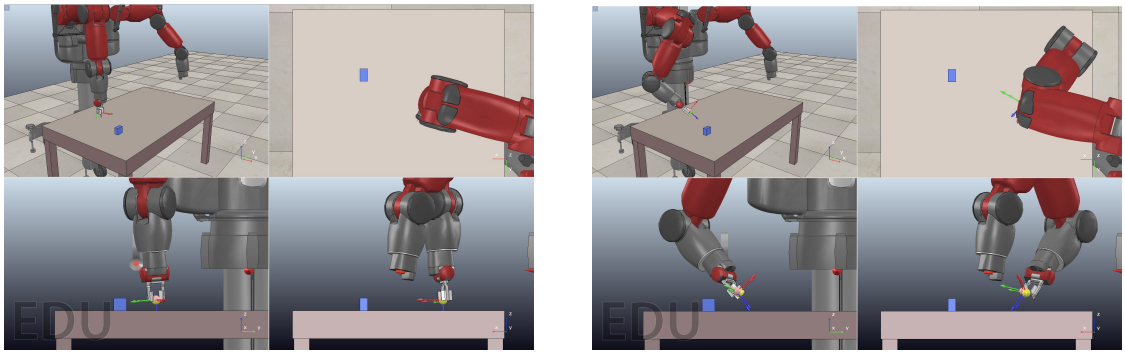
(c) (d)

Figure 3·31: 3D trajectories of the simple motion case at: (a) corner 1; (b) corner 2; (c) corner 3; and (d) corner 4.

Similarly, the orientations of the simple motion can also be changed via modifying the initial state:



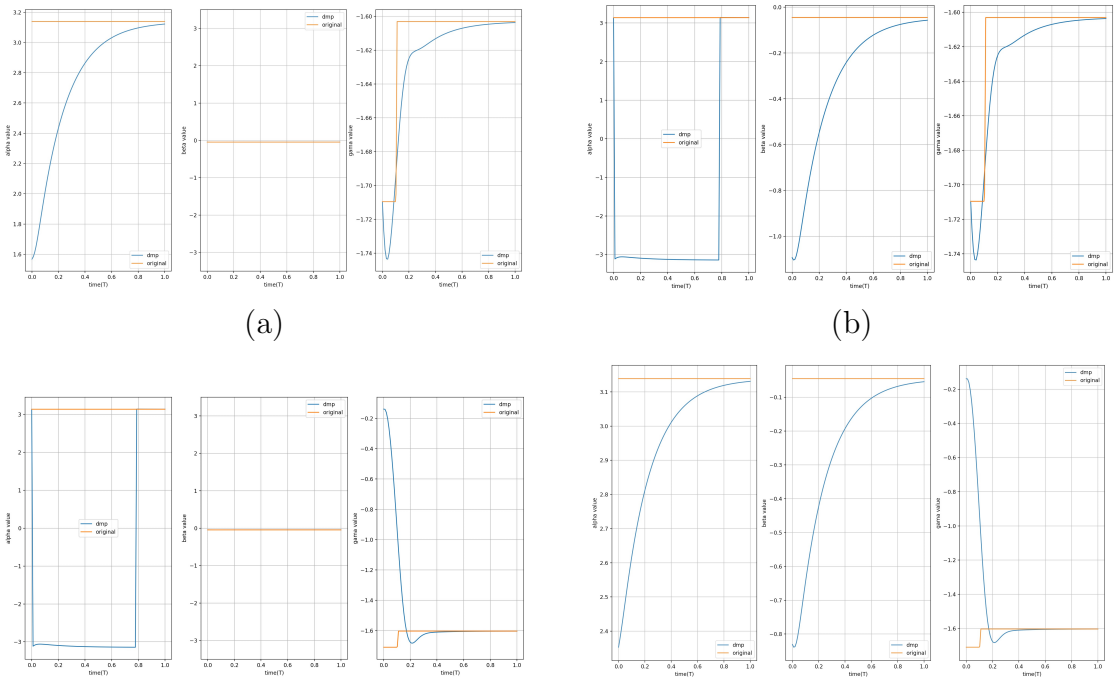
(a) (b)



(c) (d)

Figure 3·32: Orientations of the simple motion after modification at: (a) case 1; (b) case 2; (c) case 3; and (d) case 4.

Corresponding Euler angles of the cases shown in figure 3·32 are:



(a) (b)

(c) (d)

Figure 3·33: Euler angles of the simple motion after modification at: (a) case 1; (b) case 2; (c) case 3; and (d) case 4.

Here is how DMPs response to modifications of both position and orientation:

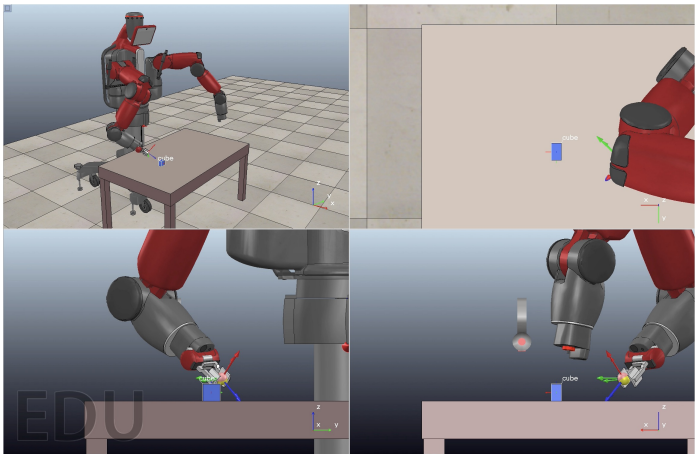
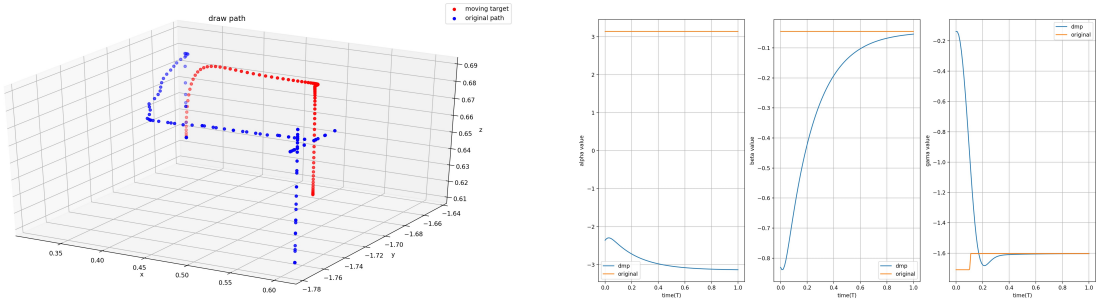


Figure 3·34: Video of modified simple motion.

The detailed translational and rotational information are shown:



(a) (b)

Figure 3·35: Translational and rotational information of modified simple motion: (a) 3D trajectories; and (b) Euler angles.

Given the simulations and results, we would say that DMPs also have potential to remain stability when encountering some unexpected disturbances in some simple movements.

Complex Motion with Modification

The modifications are based on the data extracted the complex motion case 1 and 2 shown in figure 3·24 and 3·27. To display its flexibility, the initial state is self-defined.

For complex motion case 1, the video is shown:

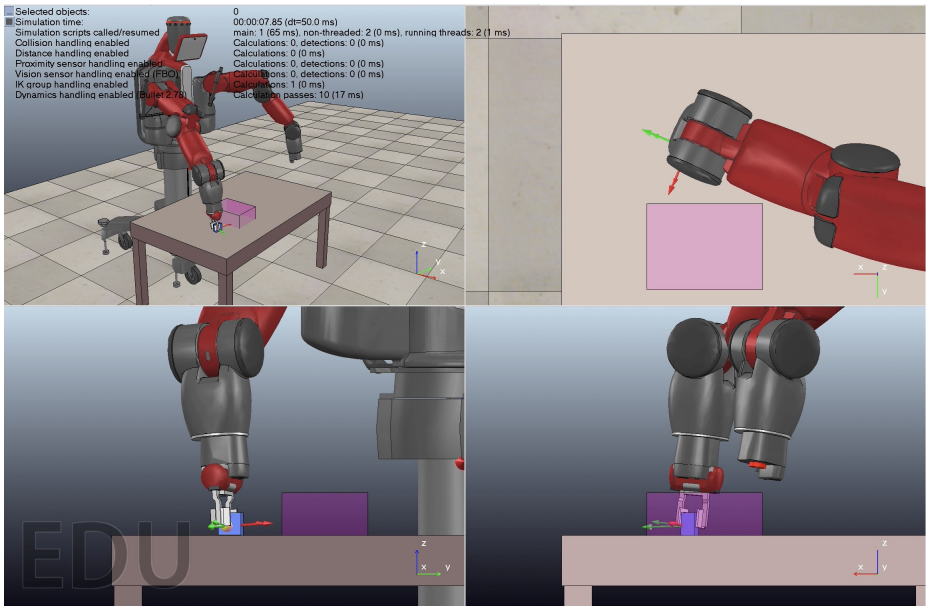
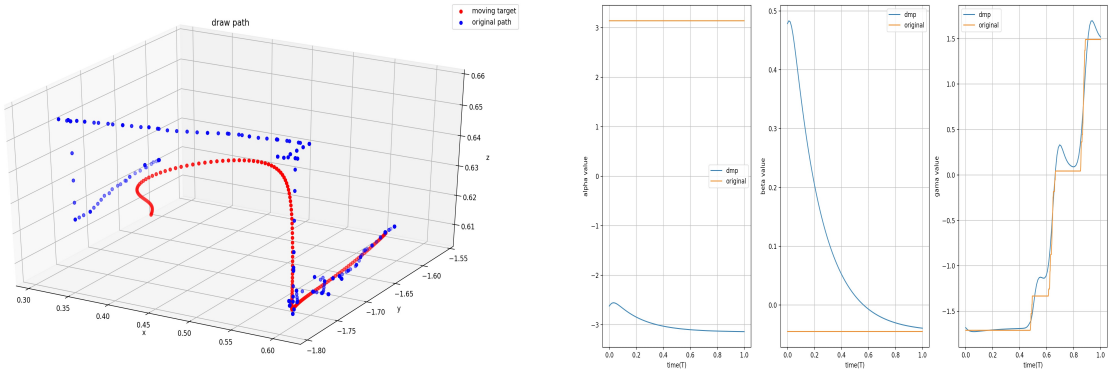


Figure 3·36: Video of modified complex motion case 1.

The visible trajectories and Euler angles of rotation are:



(a) (b)

Figure 3·37: Translational and rotational information of modified complex motion case 1: (a) 3D trajectories; and (b) Euler angles.

For complex motion case 2, the video is shown:

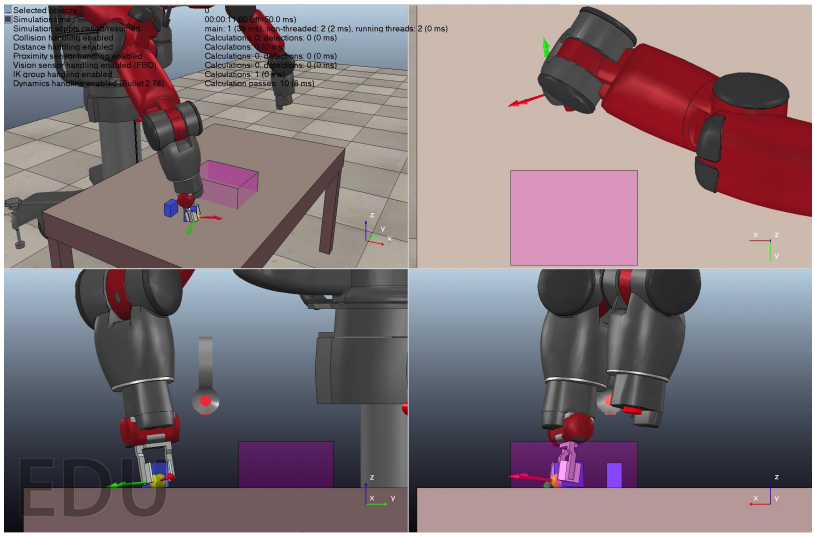
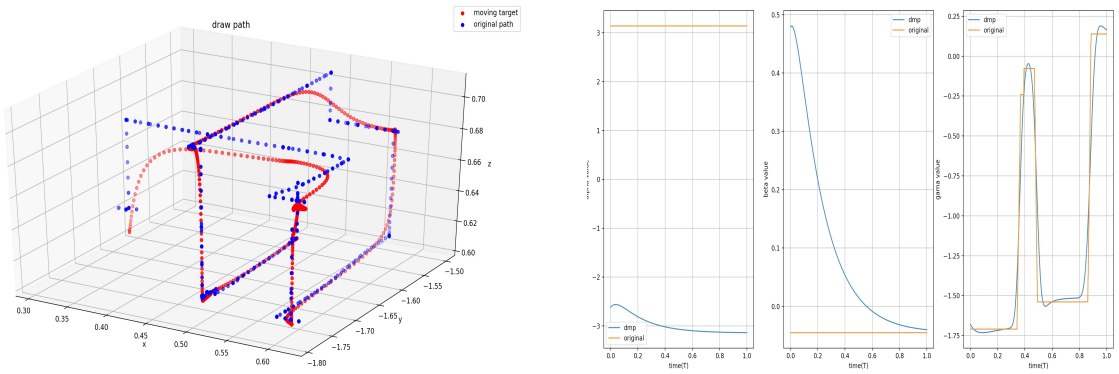


Figure 3·38: Video of modified complex motion case 2.

The visible trajectories and Euler angles of rotation are:



(a) (b)

Figure 3·39: Translational and rotational information of modified complex motion case 2: (a) 3D trajectories; and (b) Euler angles.

Combing the above two simulations and results, the fluctuation of initial coordinates are up to 50% of the cube's length, height and width. And they work for both cases. Also, the angles can fluctuate within 5 degrees for orientations. For the components which are barely changed, the fluctuation can be up to 30 degrees with full task completeness.

In addition to the difference coordinates or Euler angles, DMPs models are also very sensitive to time, too short simulation time, which means that N_{pts} is too small, may lead to failure. Likewise, if the system has enough long running time locally when it comes across any outside disturbances, the system can always manage to reach the goal state eventually. Increasing larger number of generated points locally is an available method of improving the DMPs resistance to disturbance (DeWolf, 2013b).

Chapter 4

Conclusion and Future Development

4.1 Conclusion

With relative high tolerance, all task-oriented imitation learning achieves its goals. For simple movements such as grasping, the control targets can be put in almost the whole work space. For complex motions shown in figure 3·36 and 3·38, the initial state or the goal state can be transformed or scaled. Furthermore, according to the result shown in figure 3·36, if we can divide high complexity motion into several lower complexity motions and combine them later, we can accomplish extremely complicated task with better qualitative behaviors and higher flexibility. Moreover, according to the data of real-time monitor of coordinates, velocities and Euler angles, the real-life implementations are anticipated to be successful.

4.2 Future Development

In addition to non-gripper movements, the gripper algorithm has been designed but force feedback system is required for further implementation. Furthermore, it can imply that using gripper to grasp the objects will be more effective than just dragging with friction forces.

The future of DMPs can be highly anticipated. According to recent papers (Ginesi et al., 2019), the kernel or the basis function of DMPs can still be developed. Likewise, LWR can be replaced by some other learning algorithm such as compli-

cated deep learning or reinforcement learning with time-related reward. In addition, for better real-life implementation, we can design a feedback system for better and more stable motor performance. Also, we can add some time-correlated optimization algorithms into DMPs for better stabilization. For further extension, it may combine with some advanced learning algorithms such as generative adversarial network (GAN) to achieve higher generality. Likewise, it can be incorporated within some other algorithms such as obstacle-avoiding algorithm to solve some advanced or complicated robot control problems. Moreover, it may be able to develop some advanced nonparametric control strategies based on DMPs. And this is because if we have enough much data for training, we may build a mapping from the weights to different control policies more generally.

Appendix A

Rotation Matrix, Exponential Map and Video description

A.1 Euler Angle and Rotation Matrix

For representing rotational movements in $\mathbb{R}^3$, one available method is the rotation matrix R. The rotational motion can be expressed uniquely in SO(3) :

$$SO(3) = \left\{ R \in \mathbb{R}^{3\times 3} \ , \ R^T R = I \ , \ det(R) = 1 \right\} \tag{A.1}$$

Another available method is the Euler angles. Euler angles are three angles for describing the orientation of a rigid body with respect to a fixed coordinate system. The three Euler angles are denoted as α, β and γ, and they represent rotation around x, y and z axis . Their definitions are following:

- α represents the rotation angle of a rotation around the z axis .
- β represents the rotation angle of a rotation around the y axis .
- γ represents the rotation angle of a rotation around the x axis .

The ranges of Euler angles are :

$$\alpha \in (-\pi, \pi]$$
$$\beta \in [-\frac{\pi}{2}, \frac{\pi}{2}]$$
$$\gamma \in (-\pi, \pi]$$

A.1.1 Euler Angle to Rotation Matrix

Given the Euler angles α , β and γ, the rotation matrices can be worked out.

For rotation around z axis :

$$R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

For rotation around y axis :

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{A.3}$$

For rotation around x axis :

$$R_x(\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \tag{A.4}$$

Ultimately, the total rotation matrix is calculated as:

$$\begin{aligned} R(\alpha,\beta,\gamma) &= R_z(\alpha)\cdot R_y(\beta)\cdot R_x(\gamma) \\ &= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \\ &= \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \sin\alpha\sin\gamma + \cos\alpha\sin\beta\cos\gamma \\ \sin\alpha\cos\beta & \cos\alpha\cos\gamma + \sin\alpha\sin\beta\sin\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \end{aligned} \tag{A.5}$$

A.1.2 Rotation Matrix to Euler Angle

Given $R \in SO(3)$ is the rotation matrix of a rotation:

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{A.6}$$

According to equations A.5, we can solve the Euler angles :

CASE 1: $\beta \in (-\frac{\pi}{2}, \frac{\pi}{2})$, $\cos\beta \neq 0$

$$\begin{aligned} \alpha &= \arctan 2(R_{21}, R_{11}) \\ \beta &= \arcsin(-R_{31}) \\ \gamma &= \arctan 2(R_{32}, R_{33}) \end{aligned} \tag{A.7}$$

CASE 2: $\beta = \frac{\pi}{2}$, $\cos\beta = 0$ and $\sin\beta = 1$

$$\begin{aligned} \beta &= \frac{\pi}{2} \\ \gamma - \alpha &= \arctan 2(-R_{23}, R_{22}) \end{aligned} \tag{A.8}$$

CASE 3: $\beta = -\frac{\pi}{2}$, $\cos\beta = 0$ and $\sin\beta = -1$

$$\begin{aligned} \beta &= -\frac{\pi}{2} \\ \gamma + \alpha &= \arctan 2(-R_{23}, R_{22}) \end{aligned} \tag{A.9}$$

In above equations, the function $arctan2()$ is defined as:

$$\arctan 2(y,x) = \begin{cases} \arctan(\frac{y}{x}) & x > 0 \\ \arctan(\frac{y}{x}) + \pi & x < 0, y \geq 0 \\ \arctan(\frac{y}{x}) - \pi & x < 0, y < 0 \\ +\frac{\pi}{2} & x = 0, y > 0 \\ -\frac{\pi}{2} & x = 0, y < 0 \\ \text{undefined} & x = 0, y = 0 \end{cases}$$

A.2 Rotation Matrix and Exponential Map

Every orientation can be represented via a unique $R \in SO(3)$. Therefore, any orientation trajectory can be written as a function correlated to time:

$$R(t) \in SO(3), \quad 0 \leq t \leq T \tag{A.10}$$

On one side, the rotational motion of any point $p \in \mathbb{R}^3$ attached to a robot's end effector is given by a function $p(t)$. The expressions of the function and its derivative are (Ude et al., 2014):

$$\begin{aligned} p(t) &= R(t)p_0 \\ \dot{p(t)} &= \dot{R(t)}p_0 = \dot{R(t)}R(t)^{-1}p(t) = \dot{R(t)}R(t)^T p(t) \end{aligned} \tag{A.11}$$

where p_0 is the initial coordinate of the point.

On the other side, according to the definition of angular velocity ω, it has:

$$\begin{aligned} \dot{p(t)} &= \omega(t) \times p(t) = [\omega(t)]_\times p(t) \\ [\omega(t)]_\times &= \begin{bmatrix} 0 & -\omega_x(t) & \omega_y(t) \\ \omega_z(t) & 0 & -\omega_x(t) \\ -\omega_y(t) & \omega_x(t) & 0 \end{bmatrix} \end{aligned} \tag{A.12}$$

where ω_x , ω_y and ω_z are scaled angular velocity components along x,y and z axis.

Combined equation A.11 and A.12, we obtain the following equation :

$$[\omega(t)]_\times = [\omega]_\times = \dot{R}R^T = \dot{R(t)}R(t)^T \tag{A.13}$$

If ω is constant, then equation A.12 can be solved analytically :

$$p(t) = e^{t[\omega]_\times}p_0 = e^{\theta(t)\frac{[\omega]_\times}{\|\omega\|}}p_0 \tag{A.14}$$

where $\theta(t) = t\|\omega\|$ denotes the rotation angle of time t.

According to Rodrigue's formula (Murray et al., 1994), it can get the exponential map :

$$e^{t[\omega]_\times} = I + \sin\theta\frac{[\omega]_\times}{\|\omega\|} + (1-\cos\theta)\frac{[\omega]_\times^2}{\|\omega\|^2} \tag{A.15}$$

Likewise, it can be proved that $e^{t[\omega]_\times} \in SO(3)$ and for any given $R \in SO(3)$ there exists $\omega \in \mathbb{R}^3$ so that $R = e^{[\omega]_\times}$.

If we limit the domain of ω to $0 \leq \|\omega\| \leq \pi$, the solution of ω is unique. Therefore,

the logarithmic map is:

$$\omega = \log(R) = \begin{cases} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, & R = I \\ \phi n, & otherwise \end{cases} \tag{A.16}$$

where R is the rotation matrix of the rotational movement, and ϕ and n are expressed by:

$$\begin{aligned} \phi &= \arccos\left(\frac{trace(R) - 1}{2}\right) \\ n &= \frac{1}{2\sin\phi} \begin{bmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{12} \end{bmatrix} \end{aligned} \tag{A.17}$$

In above equation, R_{ij} denotes the i_{th} row and j_{th} column entry of R.

When $\sin\theta = 0$, the above formula cannot work out the solution since it is gimbal lock. It can have numerically stable formula in the book (Ayache, 1991). However, it is a discontinuity in the logarithmic map. It means a boundary where the logarithmic map switches from positive to negative rotation angles.

Since $e^{\log(R)} = R$, it can obtain that:

$$e^{\log(R_2 {R_1}^T)} R_1 = R_2 {R_1}^T R_1 = R_2 \tag{A.18}$$

where $R_1, R_2 \in SO(3)$ are two rotation matrices of corresponding rotational motion.

Thus, according to equation A.14, the difference vector $\omega = \log(R_2 {R_1}^T)$ is the angular velocity from the rotational movement represented by rotation matrix R_1 to the rotational movement represented by rotation matrix R_2.

A.3 Video Description

The detailed information of the videos are shown following:

1. ***PC_original.mov*** ..... Figure 3·17 (a)

This is a screen record of the running VREP interface on laptop with MacOs. The video describes the demonstration trajectory of the random PC mouse movement.

2. ***PC_DMP.mov*** ..... Figure 3·17 (b)

   This is a screen record of the running VREP interface on laptop with MacOs. The video describes the DMPs-generated trajectory of the random PC mouse movement.

3. ***DMP_extract.mov*** ..... Figure 3·20

   This is a screen record of the running VREP interface on laptop with Ubuntu 16.0.0. The video describes the real-time data extraction process using a manually controlled joystick. Although the joystick is not shown in the video, the joystick is a Xbox joystick controller.

4. ***simplemotion_original.mov*** ..... Figure 3·21 (a)

   This is a screen record of the running VREP interface on laptop with MacOs. The video describes the demonstration trajectory of the simple motion case.

5. ***simplemotion_dmp.mov*** ..... Figure 3·21 (b)

   This is a screen record of the running VREP interface on laptop with MacOs. The video describes the DMPs-generated trajectory of the simple motion case.

6. ***complex1motion_original.mov*** ..... Figure 3·24 (a)

   This is a screen record of the running VREP interface on laptop with MacOs. The video describes the demonstration trajectory of the complex motion case 1.

7. ***complex1motion_dmp.mov*** ..... Figure 3·24 (b)

   This is a screen record of the running VREP interface on laptop with MacOs.

The video describes the DMPs-generated trajectory the complex motion case 1.

8. ***complex2motion_original.mov*** ..... Figure 3·27 (a)
   This is a screen record of the running VREP interface on laptop with MacOs. The video describes the demonstration trajectory of the complex motion case 1.

9. ***complex2motion_dmp.mov*** ..... Figure 3·27 (b)
   This is a screen record of the running VREP interface on laptop with MacOs. The video describes the DMPs-generated trajectory the complex motion case 1.

10. ***simplemotion_modify.mov*** ..... Figure 3·34
    This is a screen record of the running VREP interface on laptop with MacOs. The video describes the DMPs-generated trajectory of the modified simple motion case.

11. ***complexmotion_1_modify.mov*** ..... Figure 3·36
    This is a screen record of the running VREP interface on laptop with MacOs. The video describes the DMPs-generated trajectory of the modified complex motion case 1.

12. ***complexmotion_2_modify.mov*** ..... Figure 3·38
    This is a screen record of the running VREP interface on laptop with MacOs. The video describes the DMPs-generated trajectory of the modified complex motion case 2.

References

Ayache, N. (1991). *Artificial Vision for Mobile Robots: Stereo Vision and Multisensory Perception.* Cambridge, Mass.: MIT Press.

Chi, M., Yao, Y., Liu, Y., and Zhong, M. (2019). Learning, generalization, and obstacle avoidance with dynamic movement primitives and dynamic potential fields. *Applied Sciences*, 9(8):1535.

DeWolf, T. (2013a). Dynamic movement primitives part 1: The basics — studywolf. `https://studywolf.wordpress.com/2013/11/16/dynamic-movement-primitives-part-1-the-basics/`.

DeWolf, T. (2013b). Dynamic movement primitives part 2: Controlling end-effector trajectories — studywolf. `https://studywolf.wordpress.com/2013/12/05/dynamic-movement-primitives-part-2-controlling-a-system-and-comparison-with-direct-trajectory-control/`.

Gams, A. (2018). Generalization , Locally Weighted Regression , Gaussian Process Regression. `http://abr.ijs.si/upload/1523530180-Generalization.pdf`.

Ginesi, M., Sansonetto, N., and Fiorini, P. (2019). Overcoming some drawbacks of dynamic movement primitives. *arXiv preprint arXiv:1908.10608*.

Ijspeert, A. J., Nakanishi, J., Hoffmann, H., Pastor, P., and Schaal, S. (2013). Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2002). Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 2, pages 1398–1403. IEEE.

Ijspeert, A. J., Nakanishi, J., and Schaal, S. (2003). Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554.

Kramberger, A., Gams, A., Nemec, B., and Ude, A. (2016). Generalization of orientational motion in unit quaternion space. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*, pages 808–813. IEEE.

Kulvicius, T., Ning, K., Tamosiunaite, M., and Worgötter, F. (2011). Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics*, 28(1):145–157.

Lee, Y., Park, S., Lee, M., and Brosilow, C. (1998). PID controller tuning for desired closed-loop responses for si/so systems. *Aiche journal*, 44(1):106–115.

Matsubara, T., Hyon, S.-H., and Morimoto, J. (2011). Learning parametric dynamic movement primitives from multiple demonstrations. *Neural networks*, 24(5):493–500.

Murray, R. M., Li, Z., and Sastry, S. S. (1994). *Grasp statics. In: A Mathematical Introduction to Robotic Manipulation.* Boca Raton, FL: CRC.

Nemec, B. and Ude, A. (2012). Action sequencing using dynamic movement primitives. *Robotica*, 30(5):837–846.

Park, D.-H., Hoffmann, H., Pastor, P., and Schaal, S. (2008). Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Humanoids 2008-8th IEEE-RAS International Conference on Humanoid Robots*, pages 91–98. IEEE.

Pervez, A. and Lee, D. (2018). Learning task-parameterized dynamic movement primitives using mixture of gmms. *Intelligent Service Robotics*, 11(1):61–78.

Rosado, J., Silva, F., and Santos, V. (2014). Motion generalization with dynamic primitives. In *Mobile Service Robotics*, pages 215–222. World Scientific.

Rückert, E. and d'Avella, A. (2013). Learned parametrized dynamic movement primitives with shared synergies for controlling robotic and musculoskeletal systems. *Frontiers in computational neuroscience*, 7:138.

Schaal, S. (2006). Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer.

Schaal, S., Atkeson, C. G., and Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Applied Intelligence*, 17(1):49–60.

Schaal, S., Peters, J., Nakanishi, J., and Ijspeert, A. J. (2003). 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 (IROS 2003)*, pages 1–21.

Tamosiunaite, M., Nemec, B., Ude, A., and Wörgötter, F. (2011). Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems*, 59(11):910–922.

Theodorou, E., Buchli, J., and Schaal, S. (2010). Reinforcement learning of motor skills in high dimensions: A path integral approach. In *2010 IEEE International Conference on Robotics and Automation*, pages 2397–2403.

Ude, A., Gams, A., Asfour, T., and Morimoto, J. (2010). Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, 26(5):800–815.

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

Zhou, Y. and Asfour, T. (2017). Task-oriented generalization of dynamic movement primitive. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3202–3209. IEEE.

CURRICULUM VITAE