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.

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 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, Inverse Kinematic (IK) solver has been pre-programmed in most robots, 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 are firstly trained from the demonstration using DMPs framework as well as LWR. Then, regard the weights as classifiers 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. Moveover, the task-oriented movements can always be successfully stabilized even though there are some spatial scaling and transformation as well as time scaling.

Contents

List of Figures

List of Abbreviations

DMP(s)	.....	Dynamic movement primitive(s)
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 became 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 without worrying about instability.

In this thesis, the DMPs framework is utilized for imitation learning. The input is a trajectory of motion generated by a human using some specific controllers such as a joystick, which includes both translational and rotational information. Then, the weights for the given trajectory are trained (DeWolf, 2013a). Next, substitute initial state, goal state of the output system and the weights trained from input into DMPs for regenerating the output trajectory. Ultimately, the output is a trajectory of a movement that can manage to do the same task as the demonstration trajectory. Even though the initial states and goal states can be not identical in input and output systems, similar tasks can still be done by the same set of weights.

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 coordinates and orientations 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 simulation is 50 millisecond. The strategy can undertake some spatial scaling or transformation of the initial or goal states. In addition to space scale or transformation, the task-oriented motion can be also scaled in time, which may provide smoother and 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 is consist 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 rhythmic motion model, limit cycle are added into the discrete model (Ude et al., 2010). Since DMPs is sensitive to time and can be stabilized 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 of 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 have shown its flexibility and robustness for allowing to learn obstacle avoidance algorithm (Park et al., 2008) and 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 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 specific task: (Schaal et al., 2003)

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

where x is the state vector, t is the time, u is the control vector, α is the parameter vector which is specific to the control policy π.

Based on equation 2.1, it can indicate that a mapping which is from time, state

vectors and parameter vectors to control vectors exists.

Likewise, a dynamic system can generally be written as a differential equation:

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

which is also related to time, state vectors and parameter vectors.

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

Generally, all data extracted from desired trajectories is sets of points and they are discrete. Although most robotic trajectories are continuous in our real life, they are extracted discretely when communicating between hardware and software. The extracted trajectories are first-order and second-order derivable after interpolations and approximations (DeWolf, 2013b). Likewise, in this work, discrete DMPs model is implemented (Schaal et al., 2003). The corresponding equations are (DeWolf, 2013a):

$$\begin{aligned} \ddot{y}(t) &= \alpha_{\dot{y}}(\alpha_y(g - y(t)) - \dot{y}(t)) + f \\ \dot{x}(t) &= -\alpha_x x(t) \end{aligned} \tag{2.3}$$

In equation 2.3, y is the desired system state, x is the canonical dynamic system state, t is the time, g is the goal state of the desired system, α_x, α_y and $\alpha_{\dot{y}}$ are gain

terms, as well as f is the nonlinear force term defined as:(DeWolf, 2013a)

$$\begin{aligned} f(x(t), g) &= \frac{\sum_{i=1}^{N} \psi_i(t) w_i}{\sum_{i=1}^{N} \psi_i(t)} x(t)(g - y_0) \\ \psi_i(t) &= e^{-h_i(x(t)-c_i)^2} \end{aligned} \tag{2.4}$$

where y_0 is the initial state of the desired system, ψ_i is the i_{th} basis function which is a Gaussian function, c_i and h_i are mean value and variance-related coefficient of the i_{th} Gaussian function, N is the number of basis functions.

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

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

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 function can be activated evenly in time. 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:

$$\begin{aligned} c_i &= e^{-\alpha_x T_i} \\ T_i &= \frac{i}{N_{bfs}} T \end{aligned} \tag{2.6}$$

where N_{bfs} is the number of basis functions and T is the total running time of the system.

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

$$h_i = \frac{N^{\frac{3}{2}}}{\alpha_x \cdot c_i} \tag{2.7}$$

We can scale the results in time via changing the time difference dt between each two generated dynamic system states. Likewise, the result can also be scaled in space via choosing different initial and goal state positions.

2.1.3 Invariant Property

If two dynamic systems can be represented by:

$$\begin{aligned} \dot{a} &= K_1(a) \\ \dot{b} &= K_2(b) \end{aligned} \tag{2.8}$$

where K_1 and K_2 are corresponding mappings of the two systems.

Likewise, if it exists an orientation preserving homeomorphism:

$$\begin{aligned} \begin{bmatrix} a & \dot{a} \end{bmatrix} &\xrightarrow{K_h} \begin{bmatrix} b & \dot{b} \end{bmatrix} \\ \begin{bmatrix} b & \dot{b} \end{bmatrix} &\xrightarrow{{K_h}^{-1}} \begin{bmatrix} a & \dot{a} \end{bmatrix} \end{aligned} \tag{2.9}$$

where K_h stands for a mapping.

Based on equation 2.8 and 2.9, we can say that these two dynamic systems have topological equivalence. 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.

2.1.4 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).

The input is the trajectory of the demonstration and it includes both coordinate

and orientation information. Using DMPs, the framework's weights of original trajectory can be trained. According to the weights trained, a trajectory can be regenerated with same motion done.

The framework is the discrete DMPs model (Schaal et al., 2003), which is shown in equation 2.3 and 2.4. Denote the demonstration trajectory coordinate information as y_{des}, corresponding DMPs can be writen as:

$$\begin{aligned} \ddot{y}_{des}(t) &= \alpha_{\dot{y}}(\alpha_y(g - y_{des}(t)) - \dot{y}_{des}(t)) + f_{des} \\ \dot{x}(t) &= -\alpha_x x(t) \\ f_{des}(x(t), g) &= \frac{\sum_{i=1}^{N_{bfs}} \psi_i(t) w_i}{\sum_{i=1}^{N_{bfs}} \psi_i(t)} x(t)(g - y_0) \\ \psi_i(t) &= e^{-h_i(x(t)-c_i)^2} \end{aligned} \tag{2.10}$$

where x is the canonical dynamic system state, t is the time, g is the goal state of the demonstration system, α_x, α_y and $\alpha_{\dot{y}}$ are gain terms, f_{des} is the nonlinear force term calculated from the demonstration trajectory, y_0 is the initial state of the demonstration system, ψ_i is the i_{th} basis function which is also a Gaussian function, c_i and h_i are mean value and variance-related coefficient of the i_{th} Gaussian function which can be calculated from equation 2.6 and 2.7 , N_{bfs} is the number of basis functions, w_i is the weight corresponding to i_{th} basis function.

From the first equation in equations 2.10, we can calculate the forcing term:

$$f_{des} = \ddot{y}_{des}(t) - \alpha_{\dot{y}}(\alpha_y(g - y_{des}(t)) - \dot{y}_{des}(t)) \tag{2.11}$$

Based on the solution of LWR which will be discussed in chapter 2.2 , the weights can be expressed by:

$$w_i = \frac{s^T \Psi_i f_{des}}{s^T \Psi_i s} \tag{2.12}$$

where

$$s = \begin{bmatrix} x(t_0)(g - y_0) \\ \vdots \\ x(t_n)(g - y_0) \end{bmatrix}$$
$$\Psi_i = \begin{bmatrix} \psi_i(t_0) & \cdots & 0 & 0 \\ \vdots & \psi_i(t_1) & \ddots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \psi_i(t_n) \end{bmatrix} \tag{2.13}$$

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

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 f_{out} using the same canonical dynamic system x as equation 2.10:

$$\begin{aligned} f_{out}(x(t), g) &= \frac{\sum_{i=1}^{N_{bfs}} \psi_i(t) w_i}{\sum_{i=1}^{N_{bfs}} \psi_i(t)} x(t)(g - y_0) \\ \psi_i(t) &= e^{-h_i(x(t) - c_i)^2} \end{aligned} \tag{2.14}$$

Then, work out the acceleration of the generated trajectory:

$$\ddot{y}_{out}(t) = \alpha_{\dot{y}}(\alpha_y(g - y_{out}(t)) - \dot{y}_{out}(t)) + f_{out} \tag{2.15}$$

where y_{out} is the output system state.

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

$$\begin{aligned} i_{th}\ point\ velocity &= \dot{y}_{out}(t_i) = \dot{y}_{out}(t_{i-1}) + \ddot{y}_{out}(t_i) \cdot dt \\ i_{th}\ point\ displacement &= y_{out}(t_i) = y_{out}(t_{i-1}) + \dot{y}_{out}(t_i) \cdot dt \end{aligned} \tag{2.16}$$

where dt is the time difference between each two points in the generated trajectory.

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

2.1.5 DMPs in Rotational Motion

The implementation of DMPs on rotation is similar to the translation one. Nevertheless, the mathematical strategy 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 center point of rotation can be found when implementing the algorithm described in 2.1.4.

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

$$\begin{aligned}
\dot{\omega}_{des}(t) &= \alpha_{\dot{R}}[\alpha_R \gamma_{des}(t) - \omega_{des}(t)] + f_{Rdes}(x_R(t), R_g) \\
\gamma_{des}(t) &= \log\left(R_g R_{des}(t)^T\right) \\
[\omega_{des}(t)]_\times &= \begin{bmatrix} 0 & -\omega_{xdes}(t) & \omega_{ydes}(t) \\ \omega_{zdes}(t) & 0 & -\omega_{xdes}(t) \\ -\omega_{ydes}(t) & \omega_{xdes}(t) & 0 \end{bmatrix} \\
\dot{R}_{des}(t) &= [\omega_{des}(t)]_\times R_{des}(t) \\
\dot{x}_R(t) &= -\alpha_{xR} x_R(t) \\
\gamma_{des}(t) &= \omega_{des}(t)
\end{aligned} \tag{2.17}$$

where some mathematical bases can be found in appendix A, ω_{des} is the scaled angular velocity of rotational movements, ω_{xdes}, ω_{ydes} and ω_{zdes} are scaled angular velocity components along x,y and z axis, R_{des} is the rotation matrix of the demonstration, and R_g is the rotation matrices of the demonstration goal state, t is time, $\alpha_{\dot{R}}$, α_R and α_{xR} are gain terms, f_{Rdes} is the nonlinear term, $\gamma_{des}(t)$ is a coefficient correlated to

the rotation matrix, which is calculated by equation A.12.

In equation 2.17, the nonlinear term can be calculated from following equations: (Ude et al., 2014)

$$\begin{aligned} f_{Rdes} &= \frac{\sum_{i=1}^{N_{dbfs}} \psi_i^o(t) w_i^o}{\sum_{i=1}^{N_{dbfs}} \psi_i^o(t)} x(t) D_{init} \quad \in \mathbb{R}^3 \\ D_{init} &= diag(\gamma_{des}(t_0)) = diag(\log(R_g R_{des0}^T)) \quad \in \mathbb{R}^{3\times 3} \\ \psi_i^o(t) &= e^{-h_i^o (x(t)-c_i^o)^2} \end{aligned} \tag{2.18}$$

where D_{init} is a diagonal matrix in terms of initial orientation information, R_{des0} is the rotation matrix of the initial demonstration state, N_{dbfs} is the number of basis functions, ψ_i^o is the i_{th} basis function which is also a Gaussian function, c_i^o and h_i^o are mean value and variance-related coefficient of the i_{th} Gaussian function, $w_i^o \in \mathbb{R}^{1\times 3}$ is the weight corresponding to i_{th} basis function.

The expression and relationship of x_R , c_i^o and h_i^o are identical to equation 2.5 , 2.6 and 2.7:

$$\begin{aligned} x_R(t) &= e^{-\alpha_{xR} t} \\ T_{io} &= \frac{i}{N_{dbfs}} T_o \\ c_i^o &= e^{-\alpha_{xR} T_{io}} \\ h_i^o &= \frac{{N_{dbfs}}^{\frac{3}{2}}}{\alpha_{xR} \cdot c_i^o} \end{aligned} \tag{2.19}$$

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

$$w_i^o = \left(s_o^T \Psi_i^o s_o\right)^{-1} s_o^T \Psi_i^o f_{Rdes} \tag{2.20}$$

where

$$
\begin{aligned}
s_o &= \begin{bmatrix} x_R(t_0)\gamma_{des}(t_0)^T \\ \vdots \\ x_R(t_n)\gamma_{des}(t_n)^T \end{bmatrix} \\
&= \begin{bmatrix} x_R(t_0)(\log(R_g R_{des}^T(t_0)))^T \\ \vdots \\ x_R(t_n)(\log(R_g R_{des}^T(t_n)))^T \end{bmatrix} \quad \in \mathbb{R}^{3\times N_{dpts}} \\
\Psi_i^o &= \begin{bmatrix} \psi_i^o(t_0) & \cdots & 0 & 0 \\ \vdots & \psi_i^o(t_1) & \ddots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \psi_i^o(t_n) \end{bmatrix}
\end{aligned}
\tag{2.21}
$$

where t_0 , t_1 , ... t_n indicates the time of different demonstration trajectory points, N_{dpts} is the number of demonstration points.

Regard the weight as the classifier, regenerate the orientation movement of the demonstration. Firstly, work out the nonlinear forcing term:

$$
f_{Rout} = \frac{\sum_{i=1}^{N_{dbfs}} \psi_i^o(t) w_i^o}{\sum_{i=1}^{N_{dbfs}} \psi_i^o(t)} x(t) D_{init} \tag{2.22}
$$

Then, substitute the nonlinear terms into the model equations 2.17, it can calculate the derivative of the scaled angular velocity which is the scaled angular acceleration :

$$
\begin{aligned}
\dot{\omega}_{out}(t) &= \alpha_{\dot{R}}[\alpha_R \gamma_{out}(t) - \omega_{out}(t)] + f_{Rout}(x_R(t), R_g) \\
\gamma_{out}(t) &= \log(R_g R_{out}(t)^T)
\end{aligned}
\tag{2.23}
$$

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 :

$$\omega_{out}(t_i) = \omega_{out}(t_{i-1}) + \dot{\omega}_{out}(t_i) \cdot dt \tag{2.24}$$

where dt is the time difference between each two points in the generated trajectory. 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:

$$[\omega_{out}(t_i)]_\times = \begin{bmatrix} 0 & -\omega_{xout}(t_i) & \omega_{yout}(t_i) \\ \omega_{zout}(t_i) & 0 & -\omega_{xout}(t_i) \\ -\omega_{yout}(t_i) & \omega_{xout}(t_i) & 0 \end{bmatrix} \tag{2.25}$$

where ω_{xout}, ω_{yout} and ω_{zout} are scaled angular velocity components along x,y and z axis.

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

$$R_{out}(t_i) = R_{out}(t_{i-1} + dt) = e^{dt \cdot [\omega_{out}(t_i)]_\times} R_{t_{i-1}} \tag{2.26}$$

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 comprehen-

sive 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 2.1.4 , we want to choose the weights of basis functions in order that the forcing term function matches the desired trajectory f_des . Therefore, the loss function which is need to be minimized is expressed as:

$$\begin{aligned} J(w_i) &= \sum_t \psi_i(t)(f_{des}(t) - w_i(x(t)(g - y_0)))^2 \\ \psi_i(t) &= e^{-h_i(x(t)-c_i)^2} \end{aligned} \tag{2.27}$$

where w_i is the i_{th} weight of the trajectory, t is the time, f_{des} is the nonlinear forcing term calculated from demonstration DMPs , ψ_i is the i_{th} basis function which is a Gaussian function in equation 2.4, g is the goal state of the demonstration, y_0 is the initial state of the demonstration.

Convert equation 2.27 into a matrix form. Firstly, substitute different ψ_i at different time into a diagonal matrix Ψ:

$$\Psi_i = \begin{bmatrix} \psi_i(t_0) & \cdots & 0 & 0 \\ \vdots & \psi_i(t_1) & \ddots & 0 \\ 0 & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & \psi_i(t_n) \end{bmatrix} \tag{2.28}$$

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:

$$\begin{aligned} s &= \begin{bmatrix} x(t_0)(g - y_0) \\ \vdots \\ x(t_n)(g - y_0) \end{bmatrix} \\ f_d &= \begin{bmatrix} f_{des}(t_0) \\ \vdots \\ f_{des}(t_n) \end{bmatrix} \end{aligned} \tag{2.29}$$

Therefore, the equation 2.27 become:

$$J(w_i) = \Psi_i (f_d - s^T w_i)^T (f_d - s^T w_i) \tag{2.30}$$

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

$$w_i = (s^T \Psi_i s)^{-1} s^T \Psi_i f_d = \frac{s^T \Psi_i f_d}{s^T \Psi_i s} \tag{2.31}$$

2.3 Algorithm

The algorithm of implementation is shown following:

Algorithm 1 Imitation Learning With DMPs

Input:

Coordinate and orientation information with time extracted from the demonstration trajectory.

Output:

Coordinate and orientation 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 2.10 as well as $\alpha_{\dot{R}}$, α_R and α_{xR} in equation 2.17. The corresponding parameters are the same as the ones in equations 2.10 and 2.17. These parameters stay invariant in later procedures.

2: Extract data from the demonstration. The data includes both coordinate and orientation trajectories with time.

3: Choose the number of basis functions N_{bfs} properly.

4: Using the models described in section 2.1.4 and 2.1.5, train the weights of translational and rotational trajectories. Store the weights.

5: Pick out the number of points in the generated trajectories N_{pts} properly.

6: Calculate the time difference of the implementation :

$$dt = \frac{T}{N_{bfs}}$$

where T is the total run time.

7: Substitute the weights, initial and goal state for translational and rotational movements into the models described in section 2.1.4 and 2.1.5.

8: Simulate the results in VREP so that it can see how well the algorithm works.

Some influences of parameters' selection are discussed following: For procedure 1 in above algorithm, the selection of parameters $\alpha_{\dot{y}}$, α_y, $\alpha_{\dot{R}}$ and α_R follows the rules of PD controller parameter selection (Lee et al., 1998). Likewise, α_x and α_{xR} are correlated to the learning rate. The learning rate or gradient should be picked out carefully. Too high or too low both leads to the failure of reaching the minimum. For procedure 2 in above algorithm, if the other parameters are not invariant constants, the higher the N_{bfs}, the better effect the imitation process will achieve. Nevertheless, higher N_{bfs} may lead to relatively high computational cost. For procedure 5 in above algorithm, N_{pts} need to be selected properly. Too high may lead to higher computational cost, too low may lead to failure of task accomplishment. However, some comprehensive effects of multiple parameters can only be evaluated in specific situations, which 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 human doing some task-specific motion, they can do it in extremely different ways. Therefore, precise numerical error analysis has no meaning, what we need to care about is whether the point-to-point DMPs trajectories (Ginesi et al., 2019) can accomplish similar tasks. Likewise, inconspicuous error, which is less than 10% of the length or less than 5 degrees in rotation angles, can be anticipated to achieve the goal when evaluating the model. Furthermore, it is acceptable to has larger error 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. 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.

3.1.1 Motion Generated by Formulated Functions

Translational Motion

According to the theories shown in section 2.1.4, DMPs is implemented on translational movements generated by given formulas.

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

$$d_x(t) = 1.3t$$

$$d_y(t) = 1.3t$$

$$d_z(t) = 1.3t$$

The corresponding parameters in section 2.3 are given, the gain parameters are from (Schaal et al., 2003), the number of basis functions is selected so that it can regenerate the visual shape of the trajectory and the number of generated points is picked randomly as long as the DMPs has enough information to describe the task-oriented movements :

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

$$\alpha_y = 4.0$$

$$\alpha_x = 1.0$$

$$N_{bfs} = 50$$

$$N_{pts} = 500$$

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

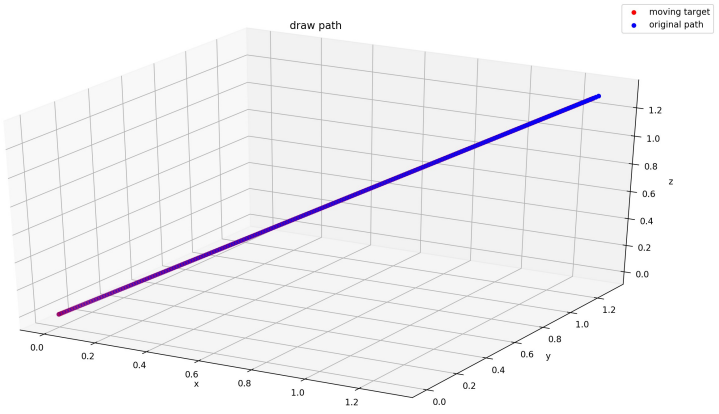


Figure 3·1: Visualized 3D original and DMPs-generated trajectories of linear functions.

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

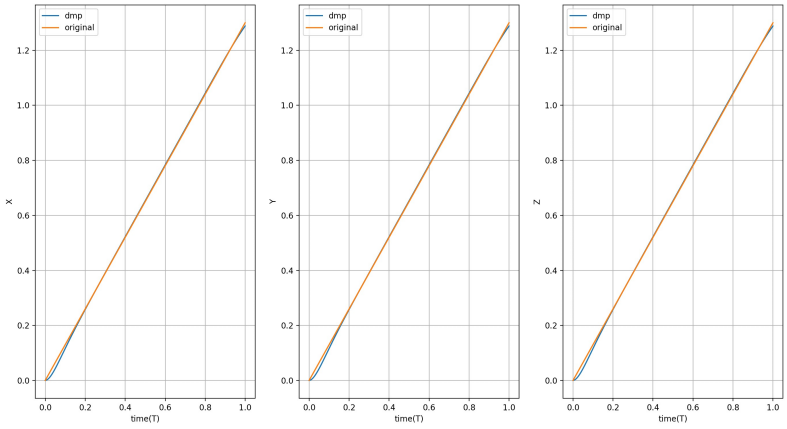


Figure 3·2: Schematic diagram of linear function-generated displacements along x,y,z axis versus time.

Then, we try with some complicated 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 previous principles of parameter selection, the corresponding parameters in section 2.3 are given:

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

$$\alpha_y = 4.0$$

$$\alpha_x = 1.0$$

$$N_{bfs} = 100$$

$$N_{pts} = 500$$

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

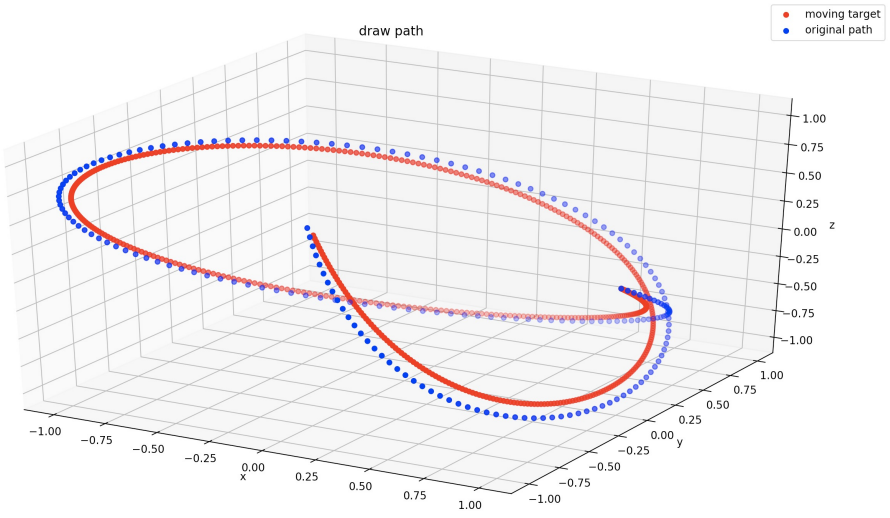


Figure 3·3: Visualized 3D original and DMPs-generated trajectories of sine or cosine functions.

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

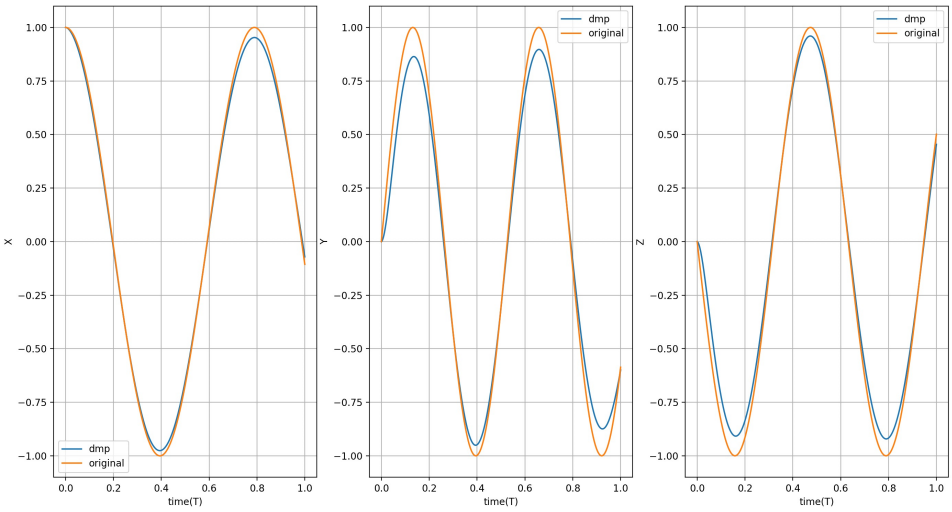


Figure 3·4: Schematic diagram of sine or cosine function-generated displacements along x,y,z axis versus time.

Ultimately, try with some comprehensive 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 parameters in section 2.3 are given:

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

$$\alpha_y = 4.0$$

$$\alpha_x = 1.0$$

$$N_{bfs} = 100$$

$$N_{pts} = 500$$

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

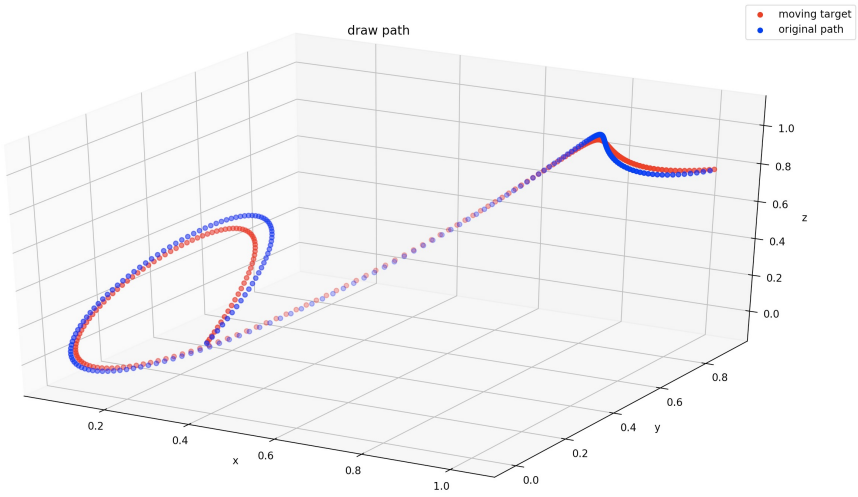


Figure 3·5: Visualized 3D original and DMPs-generated trajectories of comprehensive functions.

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

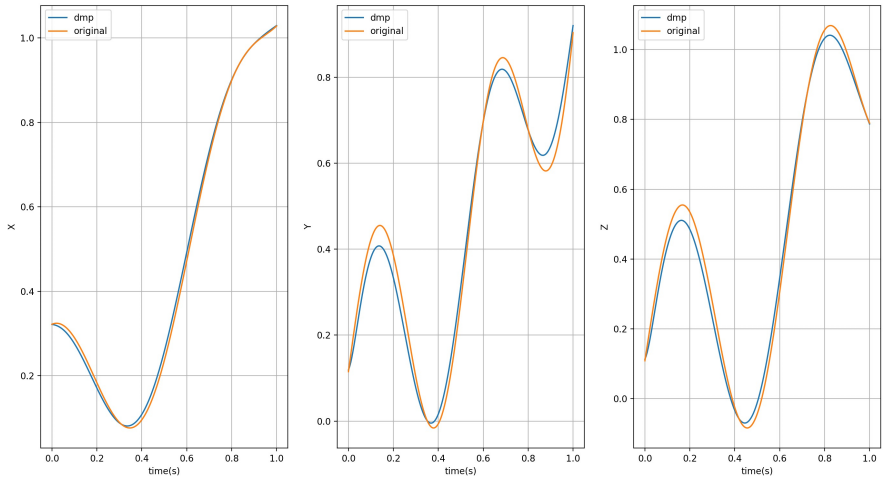
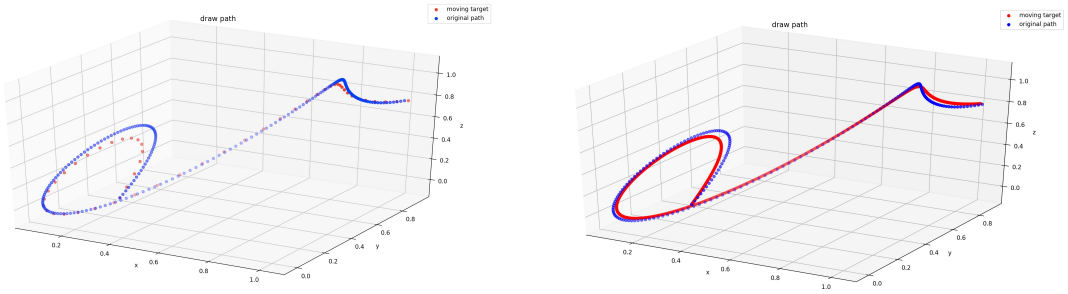


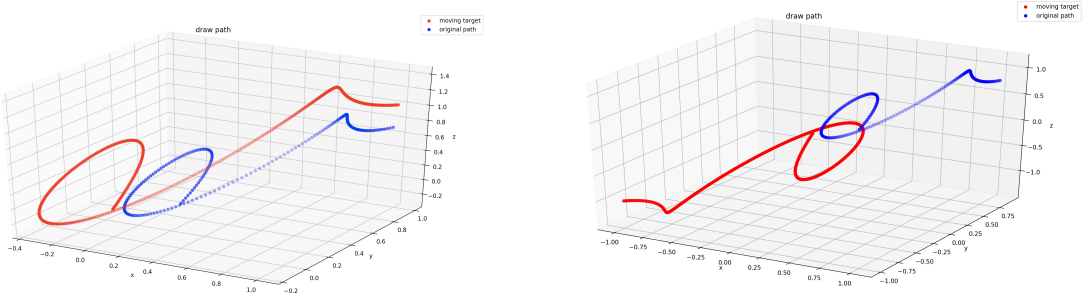
Figure 3·6: Schematic diagram of comprehensive function-generated displacements along x,y,z axis versus time.

According to above figures, it can be anticipated that the model and algorithm will do the task-oriented imitation successfully. For function-generated trajectories, we will 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 2.1.3 :



(a) (b)

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



(a) (b)

Figure 3·8: Comprehensive 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 :

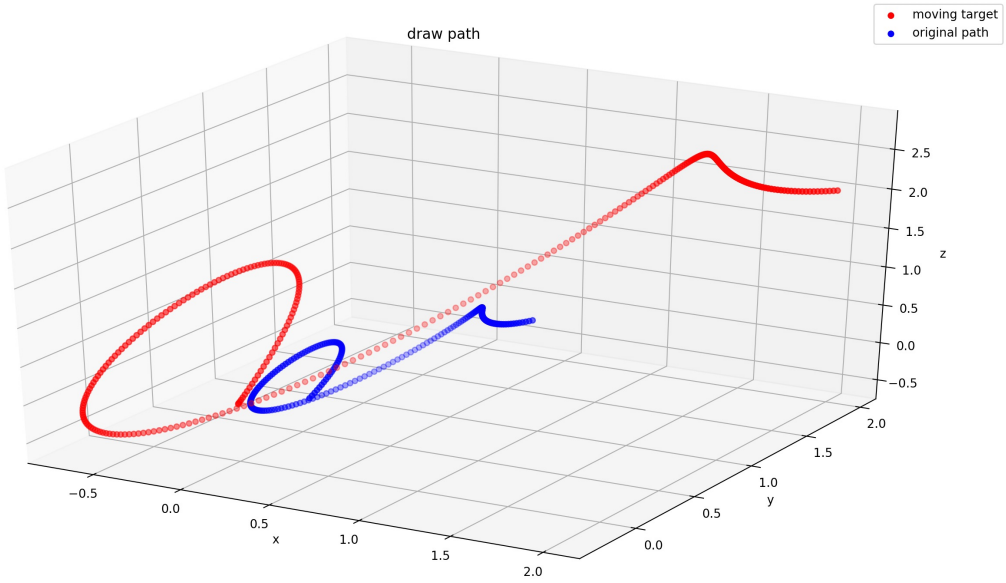


Figure 3·9: Comprehensive function-generated trajectories scaled and translated both in space and in time : set initial position to be (0,0,0), goal position to be (2,2,2) and $N_{pts} = 300$.

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 the upper part of section 3.1.1, evaluate the results of rotational motion. Extracted data are Euler angles, we can convert them into the rotation matrices. Likewise, the results are represented in rotation matrix form. Correspond equations which are utilized 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 α, β and γ are defined in section A.1

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

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

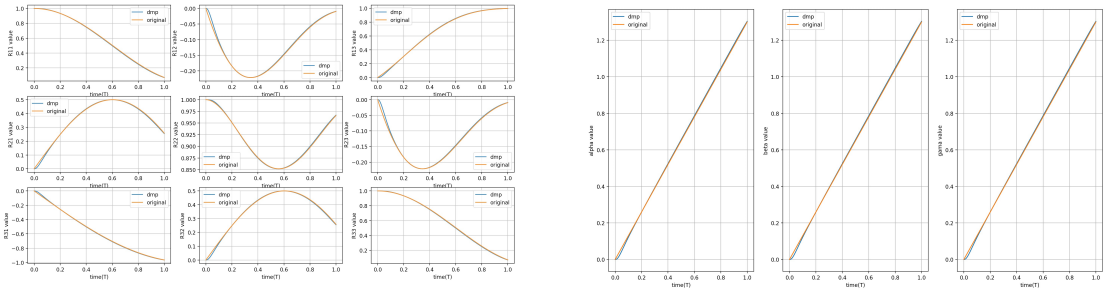
$$\alpha_R = 4.0$$

$$\alpha_{xR} = 1.0$$

$$N_{dbfs} = 100$$

$$N_{dpts} = 200$$

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 complex 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 parameters in section 2.3 are given:

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

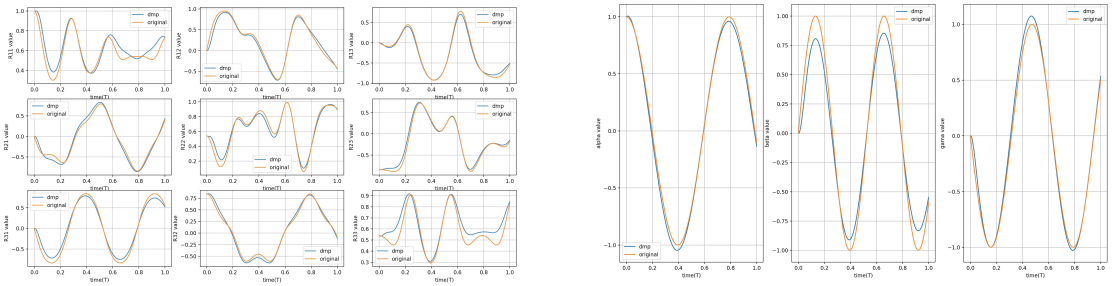
$$\alpha_R = 4.0$$

$$\alpha_{xR} = 1.0$$

$$N_{dbfs} = 100$$

$$N_{dpts} = 200$$

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 comprehensive 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 parameters in section 2.3 are given:

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

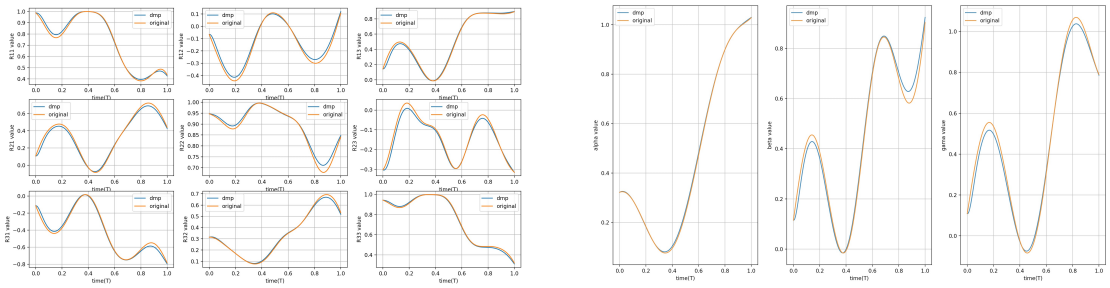
$$\alpha_R = 4.0$$

$$\alpha_{xR} = 1.0$$

$$N_{dbfs} = 100$$

$$N_{dpts} = 200$$

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



(a) (b)

Figure 3·12: Comprehensive 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 not conspicuous. For angle difference, it can directly see that the fluctuations are less than five degrees. The generated trajectories are almost overlaid to the original data, which means that they can manage to accomplish the same or similar tasks. 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:

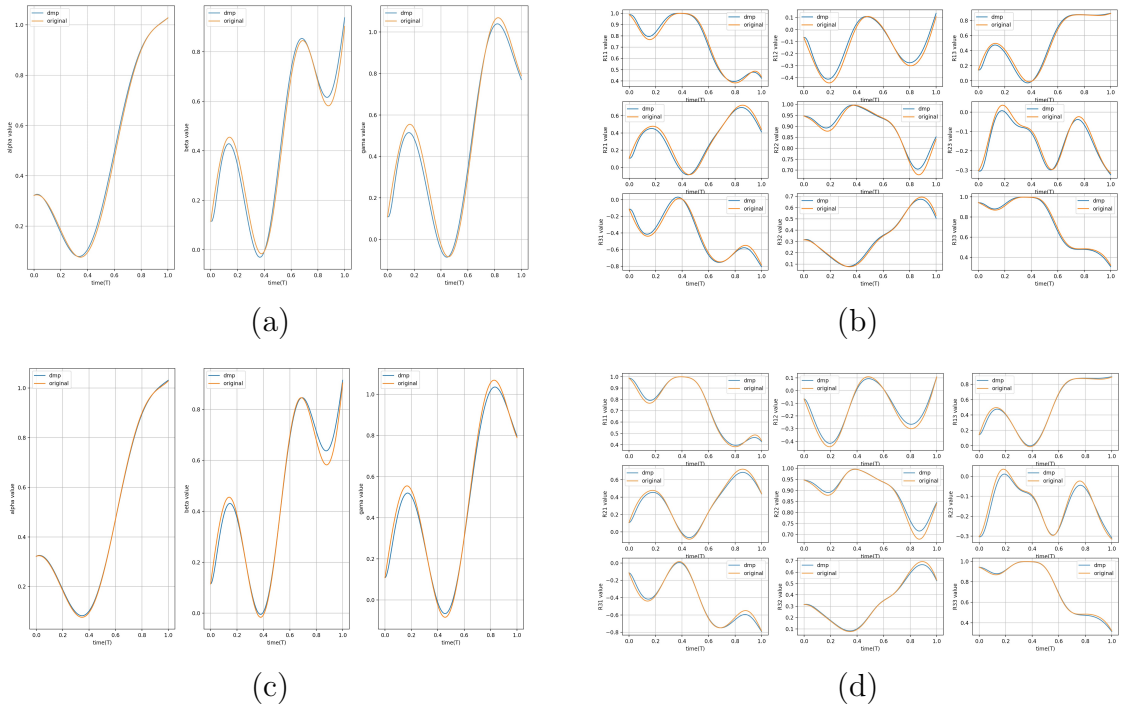
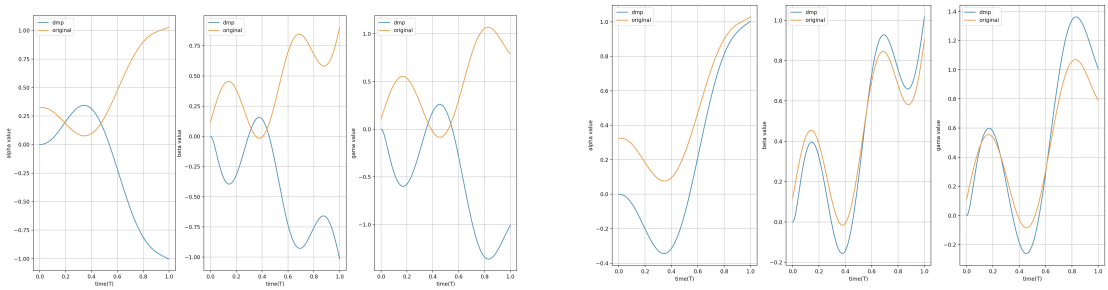


Figure 3·13: Comprehensive function-generated orientation scaled in time : (a) nine entries of the rotation matrix versus time when $N_{dpts} = 100$; (b) Euler angles versus time when $N_{dpts} = 100$; (c) nine entries of the rotation matrix versus time when $N_{dpts} = 1000$;and (b) Euler angles versus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

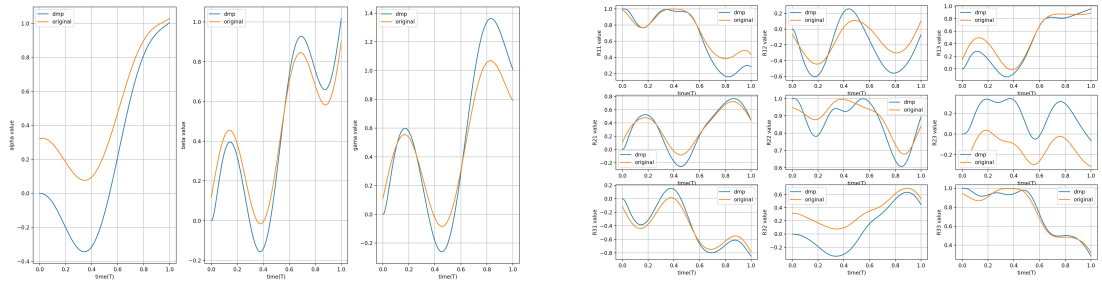


(a) (b)

Figure 3·14: Comprehensive 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: Comprehensive 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. Nevertheless, some real-life

trajectories may not be formulized but are derivable. Therefore, we can utilize the PC mouse combining with VREP to test the availability of some real-life trajectories which may lead to some singularities.

In VREP, the translational motion is controlled by given coordinates and the rotational motion is controlled by given Euler angles. The robot Baxter is used for data extraction and DMPs implementation. The end-effector is the control object. Meanwhile, my output which is consisted 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:

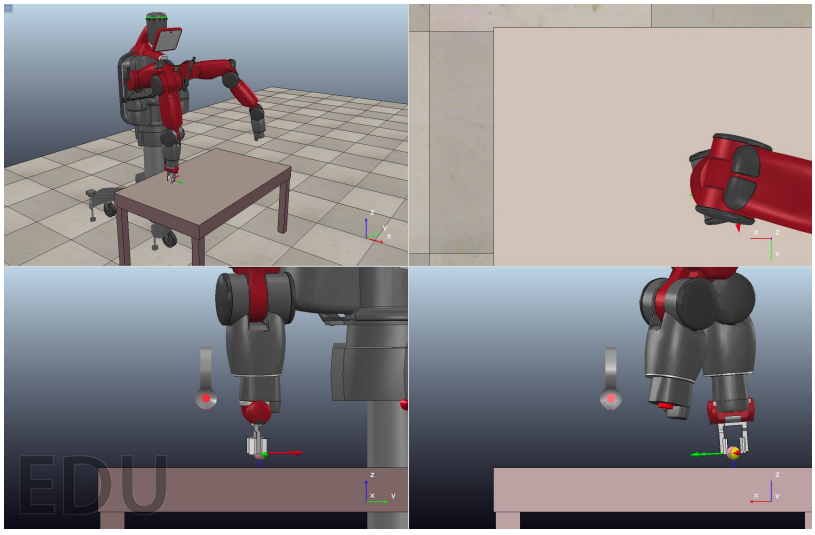
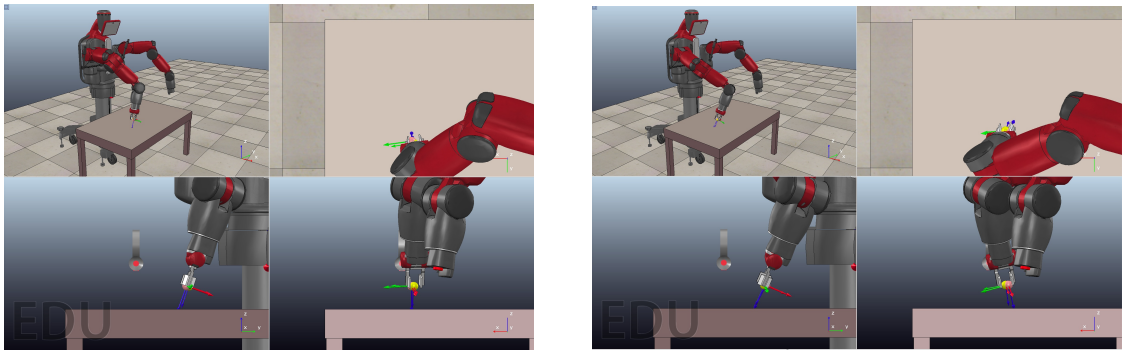


Figure 3·16: Robot Baxter with end-effector's translational and rotational information shown .

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 parameters in section2.3 are:

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

$$\alpha_y = \alpha_R = 4.0$$

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

$$N_{bfs} = 800$$

$$N_{dbfs} = 200$$

$$N_{pts} = N_{dpts} = 500$$

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

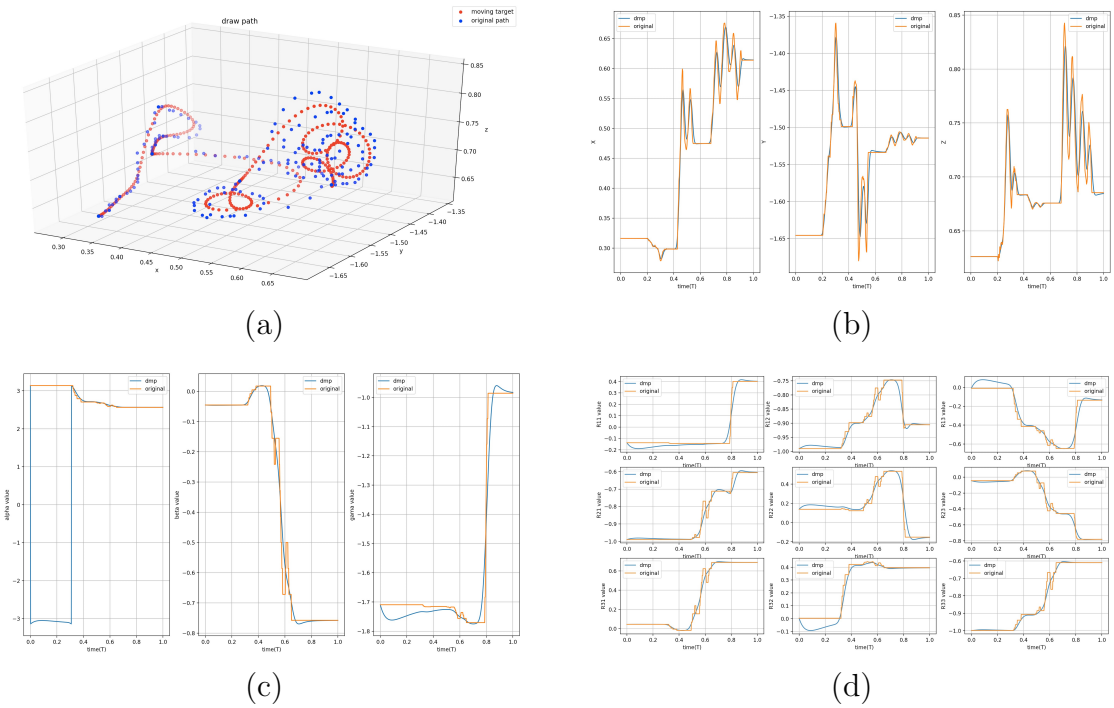
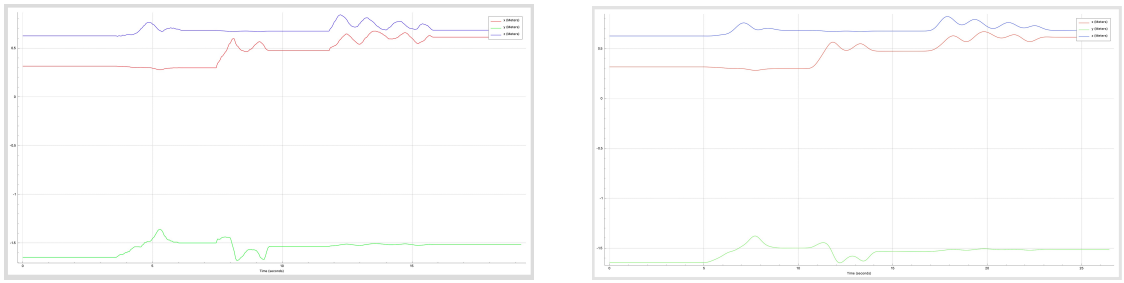


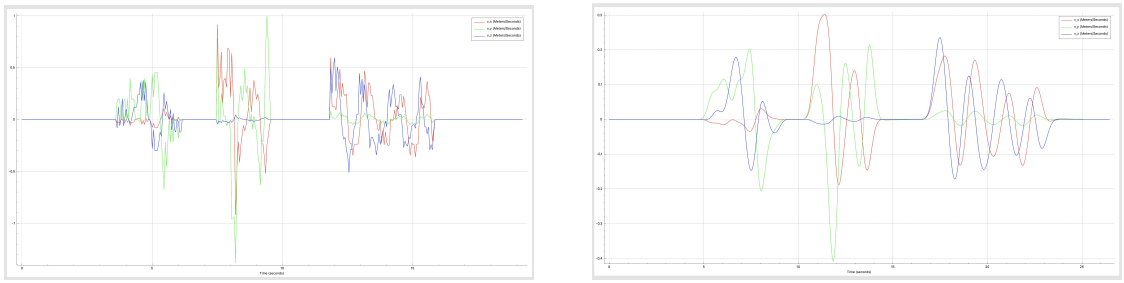
Figure 3·18: The translational and rotational information of the PC mouse-generated movement: (a) 3D visible trajectories; (b) displacements versus time along x,y and z axis; (c) Euler angles versus time; and (d) nine rotation matrix entries versus time.

According to the figures 3·18 , the algorithm can be anticipated to work for task-oriented imitation learning in simulation. For evaluating availability of real-life implementation, the figures of coordinates, velocities and Euler angles versus time are plotted:



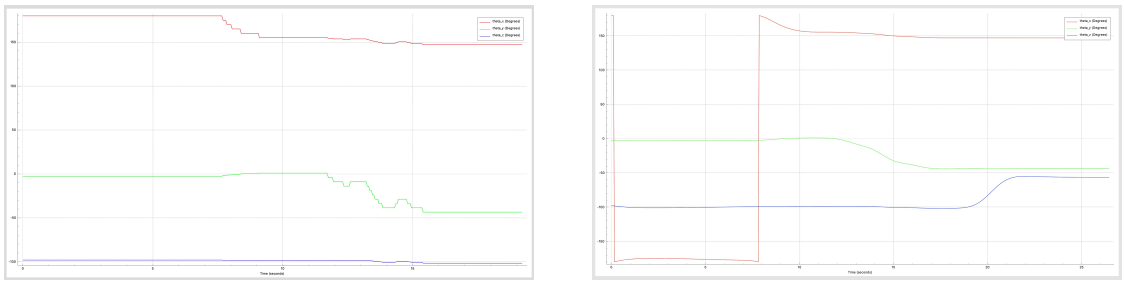
(a) (b)

Figure 3·19: Coordinates of the PC mouse motion: (a) original movement generated by PC mouse; and (b) DMPs-generated movement.



(a) (b)

Figure 3·20: Velocities of the PC mouse motion: (a) original movement generated by PC mouse; and (b) DMPs-generated movement.



(a) (b)

Figure 3·21: Euler Angles of the PC mouse motion: (a) original movement generated by PC mouse; and (b) 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.

3.1.3 Motion Generated by Joystick Implementation

The simulation and result of PC mouse movement indicate the potential of DMPs. Nevertheless, for preciser motion, PC mouse may not successfully complete the task due to the flexibility of mouse movements. Therefore, Joystick controller becomes an alternative since it can complete relatively complicated tasks. 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 have 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 unit time period . 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:

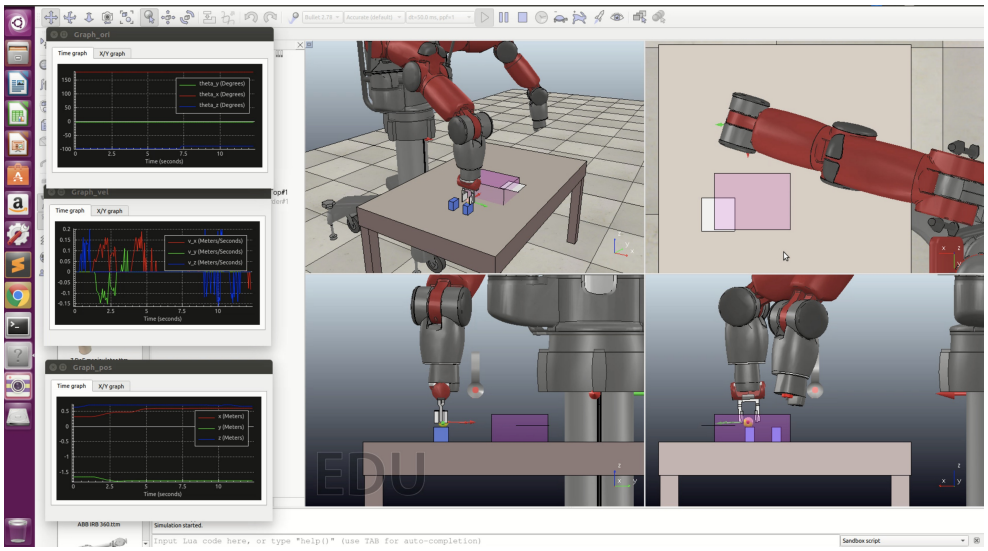


Figure 3·22: Sample video of extracting data from a complex motion.

The simulator's physics engine is based on Newton's laws of motion and it is

sensitive to forces, inertia and materials. Thus, for result or simulation evaluation, 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 50 milliseconds. Compared to the data extraction, the implementation generally take shorter time. Also, the joystick is manually controller when extracting data. It may lead to some visible spikes when extracting the data discretely from the continuous trajectories and we can obverse that the output looks like more smooth as if the data are smoothed out but actually not. The results of velocities and accelerations are smoother because they are interpolated and trained in weighted learning algorithm.

3.2.1 Simple Motion

For simple motion, the task is to let the robot arm go and ready to grasp an object, which is a cube in VREP simulation. Following previous principles of parameter selection, the corresponding parameters in section2.3 are:

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

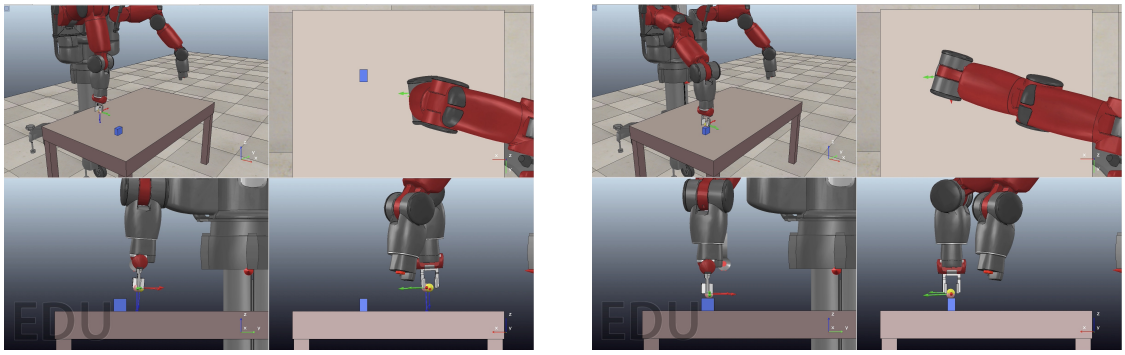
$$\alpha_y = \alpha_R = 4.0$$

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

$$N_{bfs} = 50$$

$$N_{dbfs} = 50$$

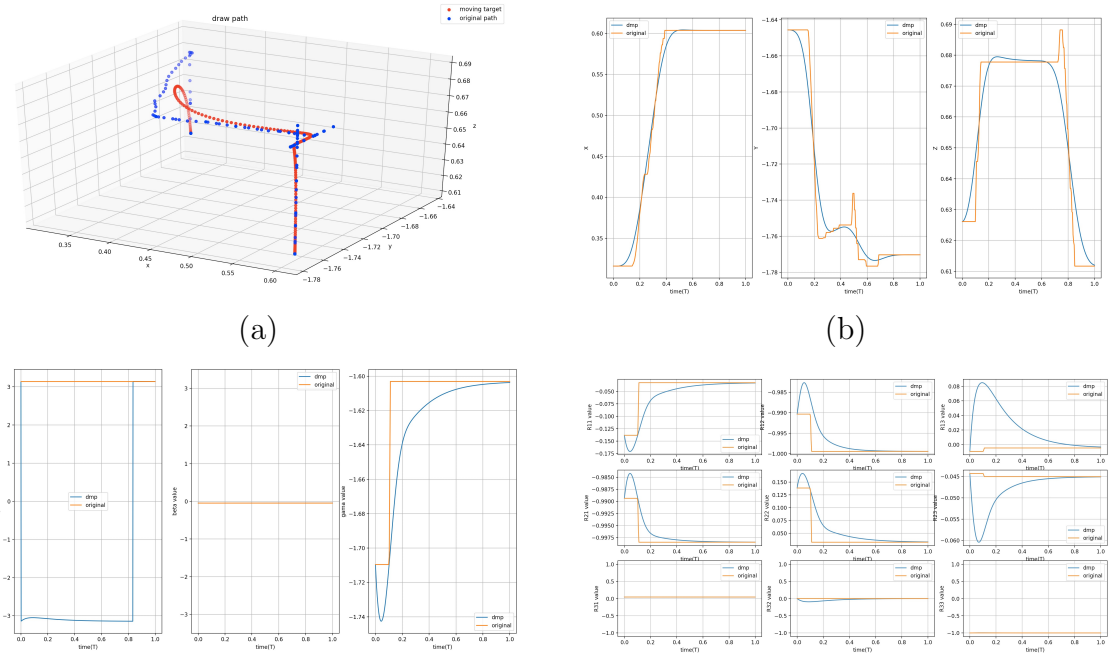
$$N_{pts} = N_{dpts} = 100$$



(a) (b)

Figure 3·23: 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:



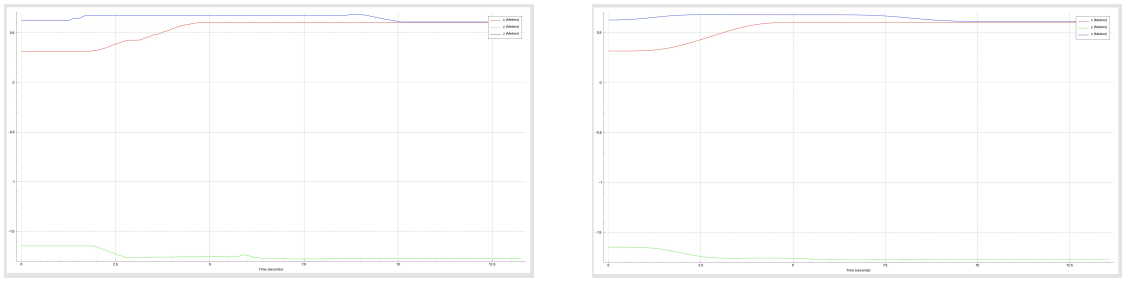
(a) (b)

(a) (b)

Figure 3·24: 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.

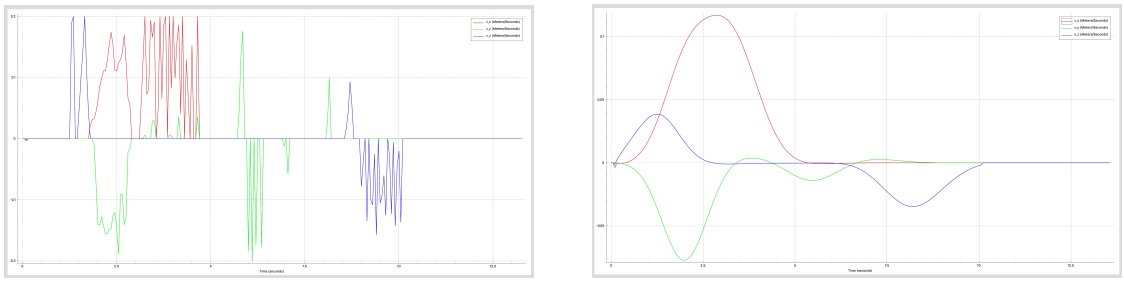
Although it seems that it has relatively large errors in figure 3·24 (b), the smoothed trajectories can be anticipated to regenerate the human motion successfully. The goal of the DMPs are to regenerate task-specific trajectories rather than identical trajectories. Also, for figure 3·24 (c), the angles seem to have large differences; nevertheless, since the Euler angle's range is from $(-\pi, \pi]$ and for direct observation, we convert the rotation matrix into Euler angles. Thinking about the Euler angles, from $-\pi+0.1$ to $\pi - 0.1$ actually can be achieved by rotating 0.2 rads rather than $2\pi - 0.2$ rads in counterclockwise direction.

According to the figures 3·24 , the algorithm would work perfectly for task-oriented imitation learning of the simple motion case. Therefore, we can draw the figures of coordinates, velocities and Euler angles versus time to test the availability of real-life implementation :



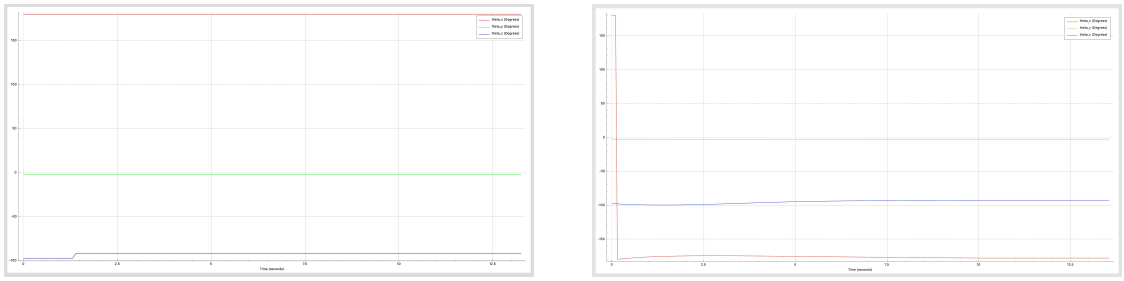
(a) (b)

Figure 3·25: Coordinates of the simple motion case: (a) original movement generated by joystick; and (b) DMPs-generated movement.



(a) (b)

Figure 3·26: Velocities of the simple motion case: (a) original movement generated by joystick; and (b) DMPs-generated movement.



(a) (b)

Figure 3·27: Euler Angles of the simple motion case: (a) original movement generated by joystick; and (b) 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 3·27 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 3·24 indicates the algorithm's 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 compared to the first one. Although it would be better if we can divide a complicated motion into several simple movements and it can anticipate that

the accuracy will improve after separation, sometimes we cannot divide one motion when some important real-time information are not given. For instance, for the motion which includes reaching to an object and drag the object, if the position and orientation of object is not given, we cannot separate the movement. Therefore, we regard every trajectory as one continuous trajectory and do not divide the trajectory into several parts for retaining its integrity.

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

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

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

$$\alpha_y = \alpha_R = 4.0$$

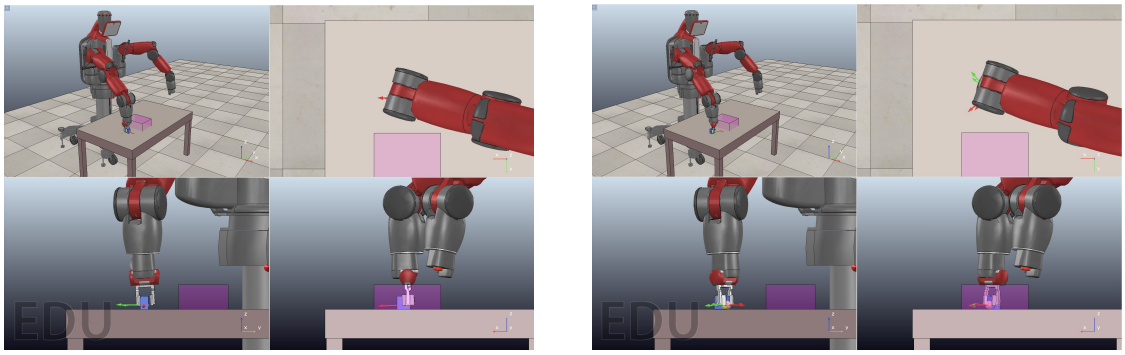
$$\alpha_x = \alpha_{xR} = 1.0$$

$$N_{bfs} = 50$$

$$N_{dbfs} = 200$$

$$N_{pts} = N_{dpts} = 200$$

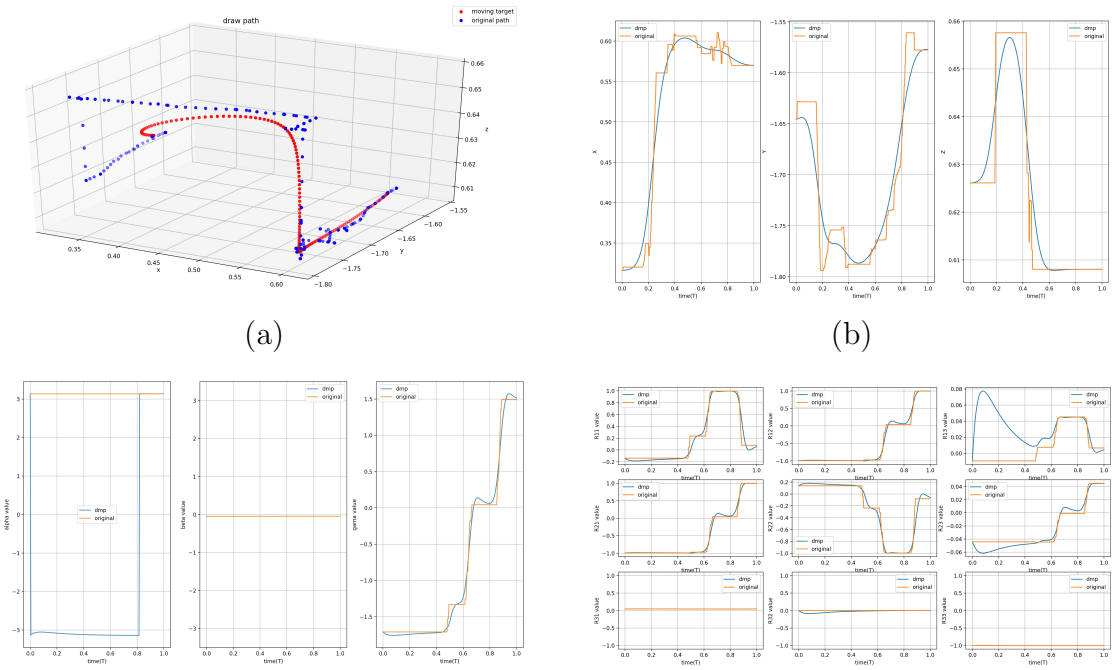
For evaluating the results of the task-oriented imitation learning directly , the VREP simulations of original and DMPs-generated trajectories are shown following:



(a) (b)

Figure 3·28: 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:

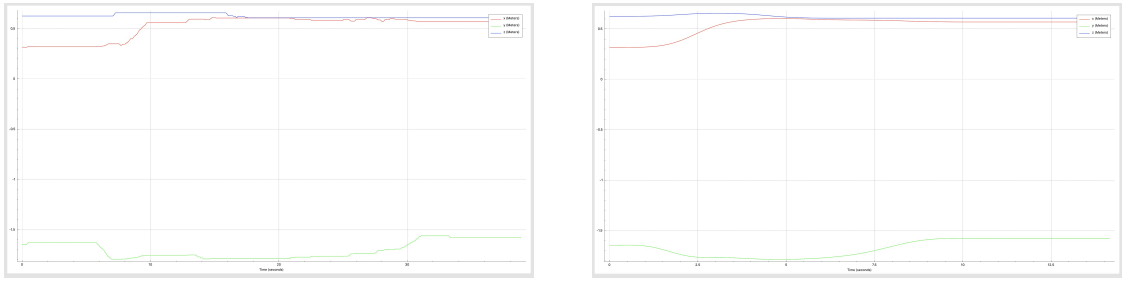


(a) (b)

(a) (b)

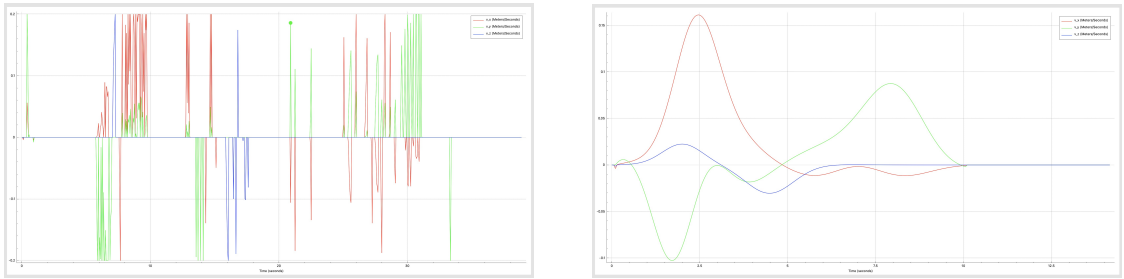
Figure 3·29: The translational and rotational information of the complex motion case 1: (a) 3D visible trajectories; (b) displacements versus time along x,y and z axis; (c) Euler angles versus time; and (d) nine rotation matrix entries versus time.

According to the figures 3·29 , the algorithm would work perfectly for task-oriented imitation learning of the complex motion case 1. Therefore, we can draw the figures of coordinates, velocities and Euler angles versus time to test the availability of real-life implementation :



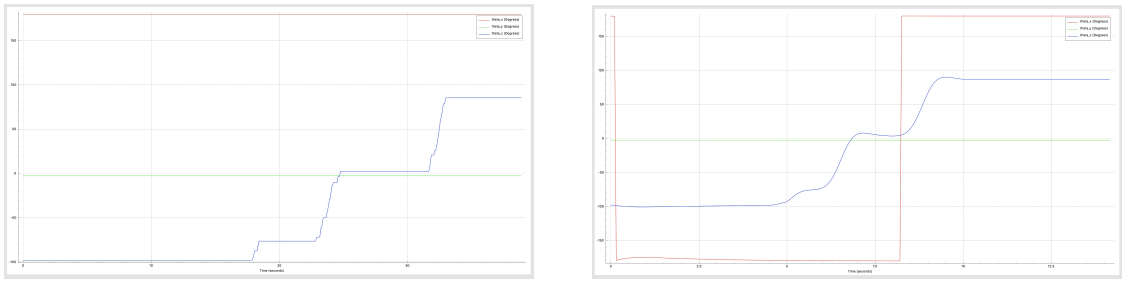
(a) (b)

Figure 3·30: Coordinates of the complex motion case 1: (a) original movement generated by joystick; and (b) DMPs-generated movement.



(a) (b)

Figure 3·31: Velocities of the complex motion case 1: (a) original movement generated by joystick; and (b) DMPs-generated movement.



(a) (b)

Figure 3·32: Euler Angles of the complex motion case 1: (a) original movement generated by joystick; and (b) 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 3·32 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 3·29 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 parameters in section 2.3 are:

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

$$\alpha_y = \alpha_R = 4.0$$

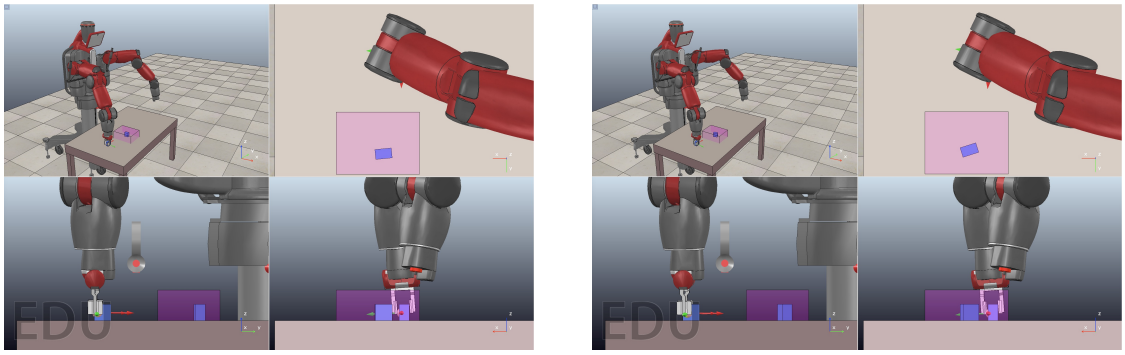
$$\alpha_x = \alpha_{xR} = 1.0$$

$$N_{bfs} = 500$$

$$N_{dbfs} = 200$$

$$N_{pts} = N_{dpts} = 600$$

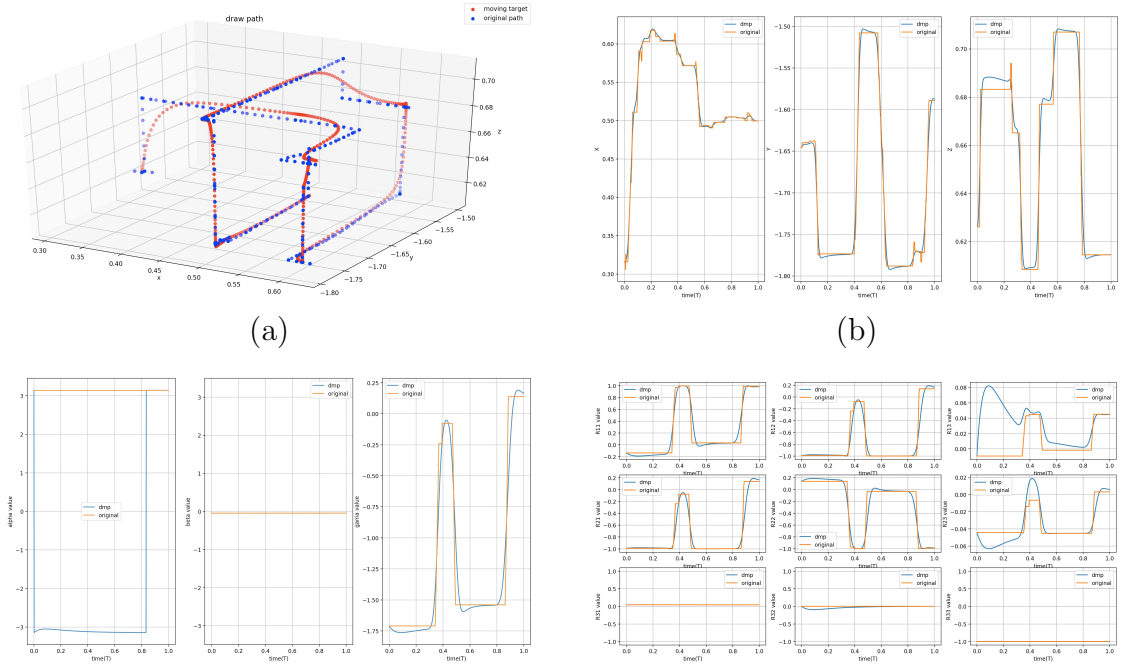
For evaluating the results of the task-oriented imitation learning directly , the VREP simulations of original and DMPs-generated trajectories are shown following:



(a) (b)

Figure 3·33: 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:

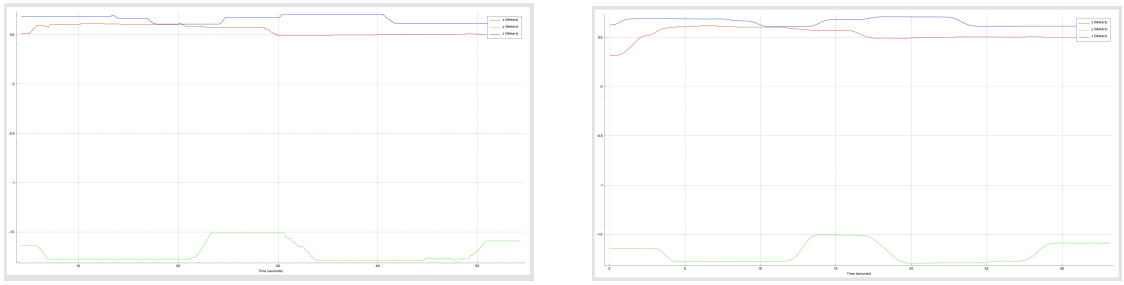


(a) (b)

(a) (b)

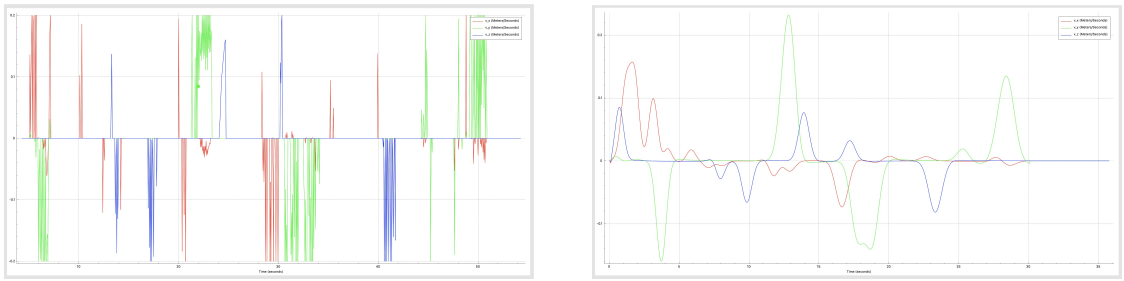
Figure 3·34: The translational and rotational information of the complex motion case 2: (a) 3D visible trajectories; (b) displacements versus time along x,y and z axis; (c) Euler angles versus time; and (d) nine rotation matrix entries versus time.

According to the figures 3·34 , the algorithm would work perfectly for task-oriented imitation learning of the complex motion case 2. Therefore, we can draw the figures of coordinates, velocities and Euler angles versus time to test the availability of real-life implementation :



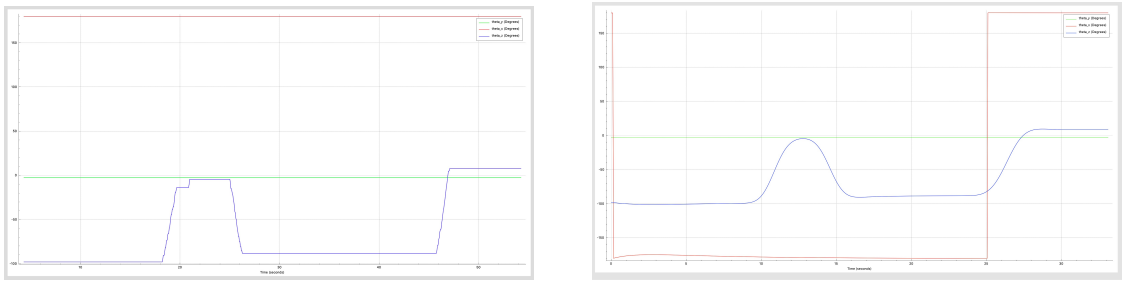
(a) (b)

Figure 3·35: Coordinates of the complex motion case 2: (a) original movement generated by joystick; and (b) DMPs-generated movement.



(a) (b)

Figure 3·36: Velocities of the complex motion case 2: (a) original movement generated by joystick; and (b) DMPs-generated movement.



(a) (b)

Figure 3·37: Euler Angles of the complex motion case 2: (a) original movement generated by joystick; and (b) DMPs-generated movement.

According to the above results, it can anticipate that DMPs has potential to accomplish task-oriented imitation learning in real life. Although the Euler angles in 3·37 may display a huge error or gap, this problem can be avoided if the representation of rotation is rotation matrix or quaternion (Ude et al., 2014). The rotation matrix shown in figure 3·33 indicates the availability. 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

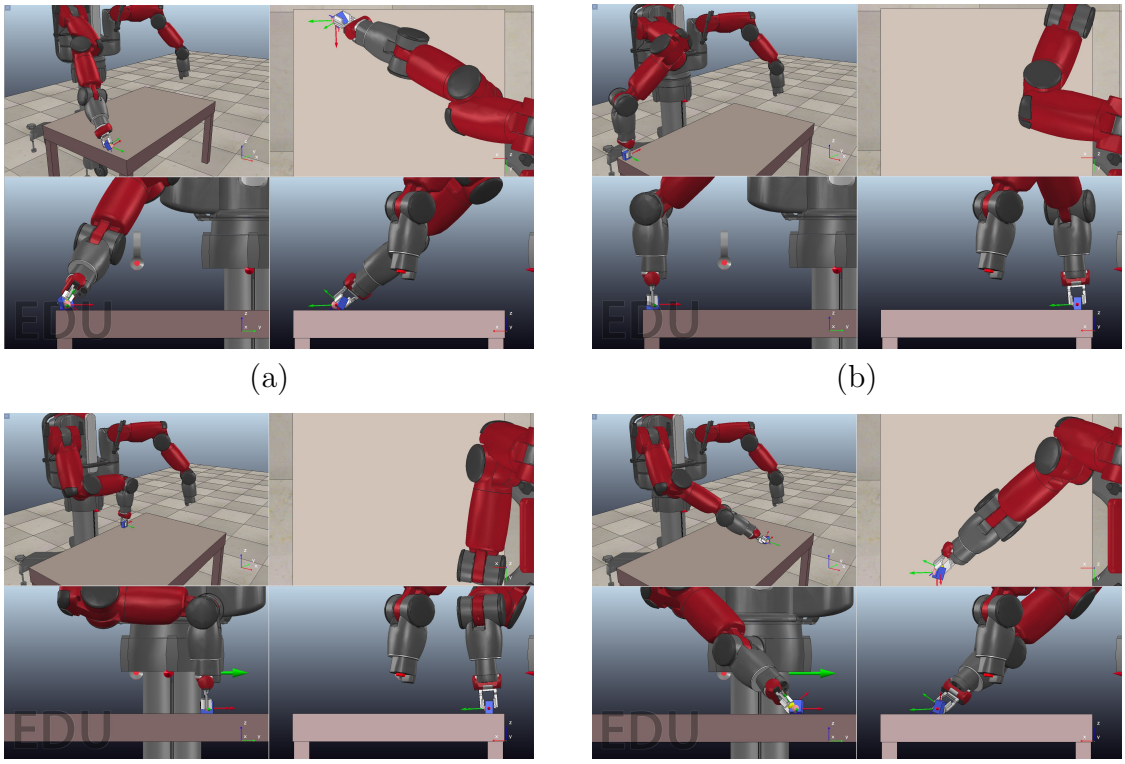
The parameters of the algorithm in section 2.3 for different motions are same as the ones in section 3.2.

Simple Motion with Modification

The modifications are based on data extracted from the simple motion case shown in figure 3·23. 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:

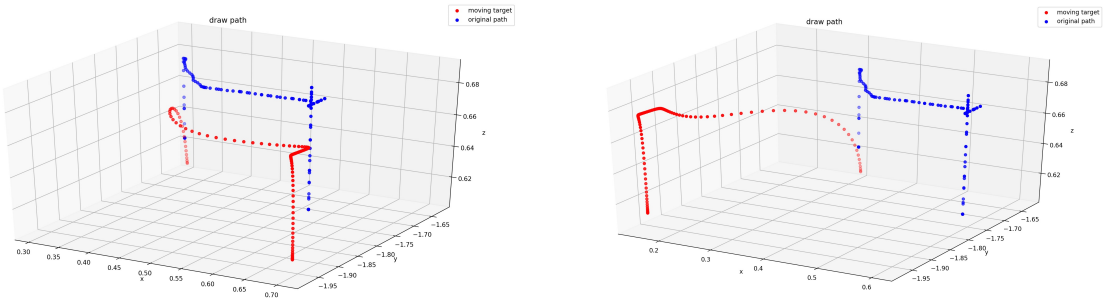


(a) (b)

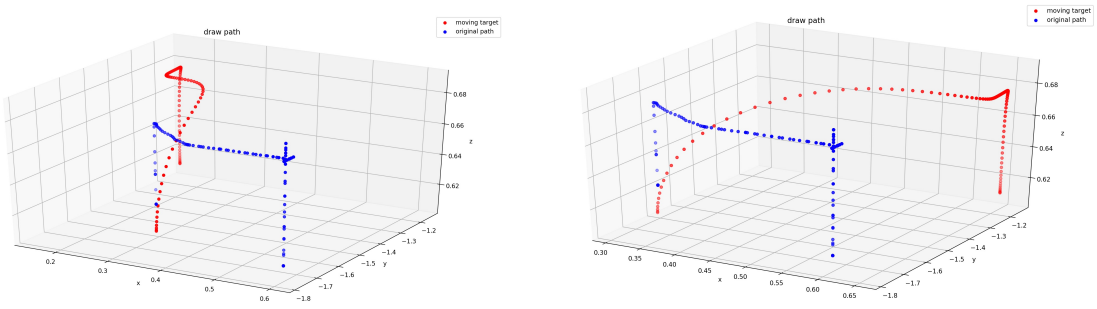
(c) (d)

Figure 3·38: 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:



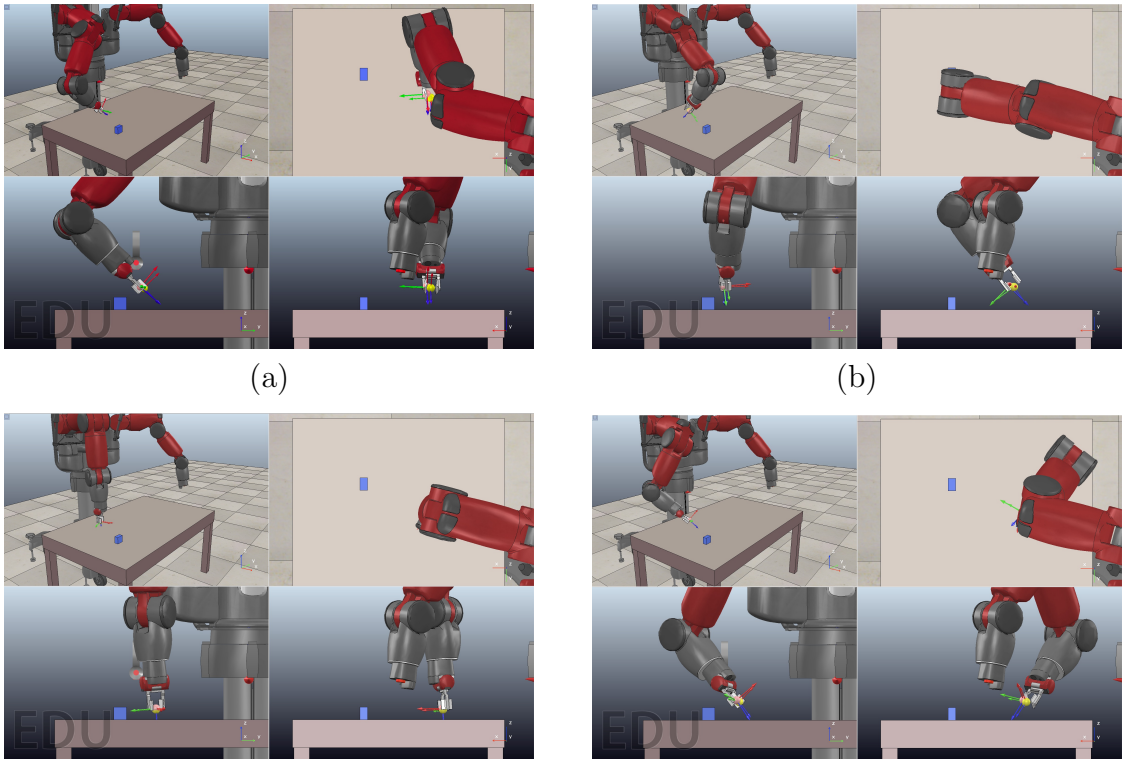
(a) (b)



(c) (d)

Figure 3·39: 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 3·40: 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 3·40 are:

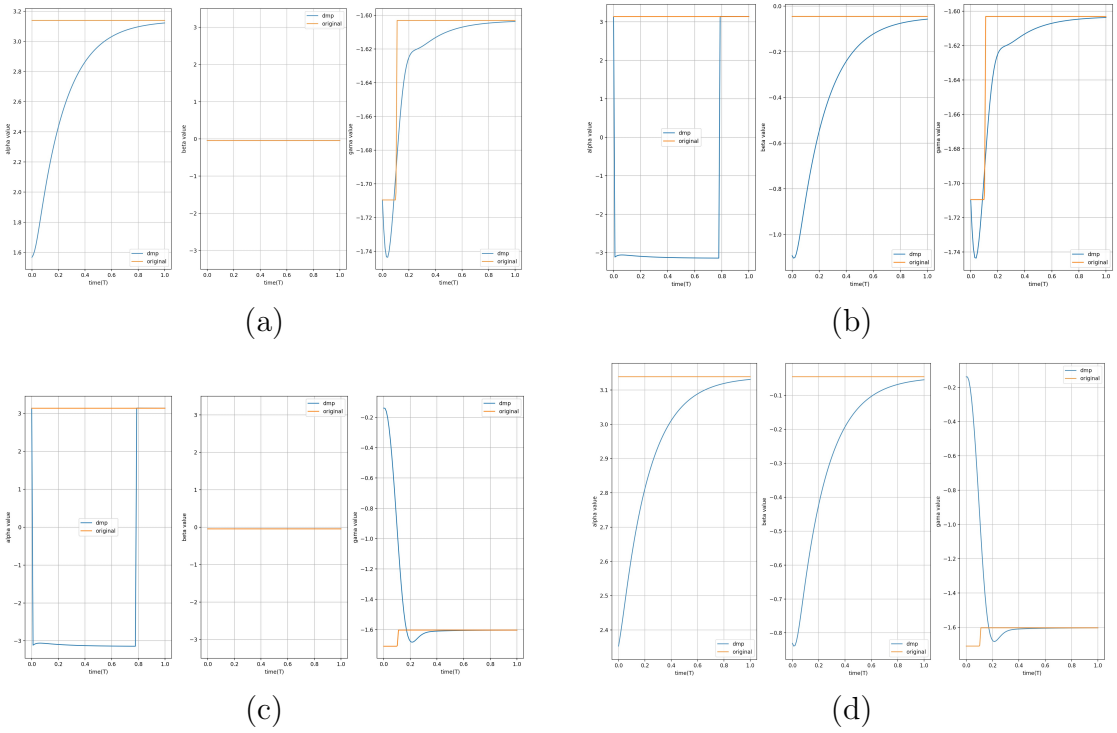


Figure 3·41: Euler angles of the simple motion after modification at: (a) case 1; (b) case 2; (c) case 3; and (d) case 4.

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

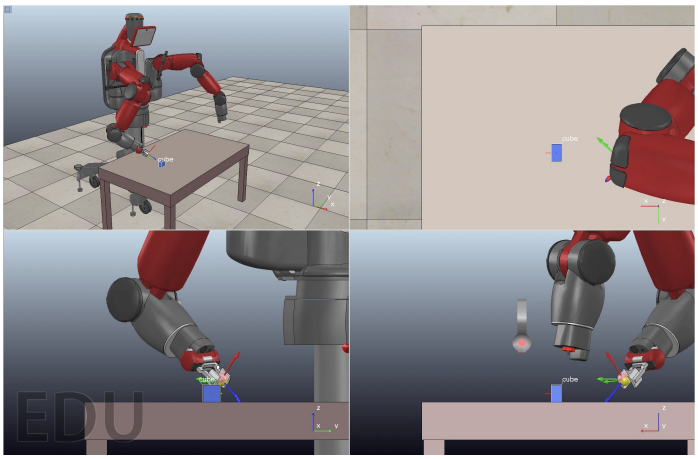
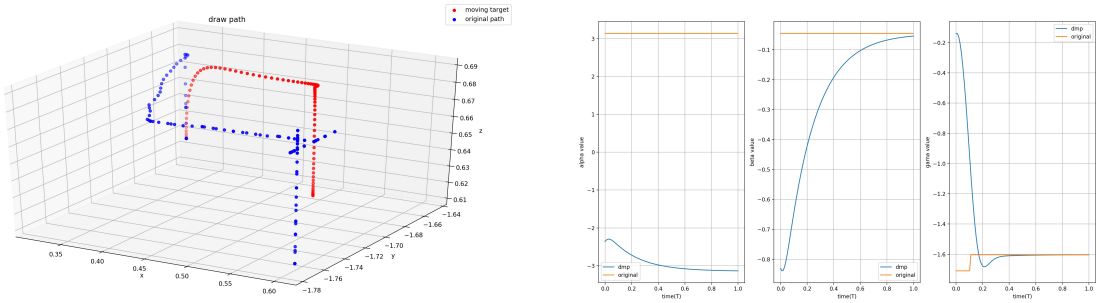


Figure 3·42: Video of modified simple motion.

The detailed translational and rotational information are shown:



(a) (b)

Figure 3·43: 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 3·28 and 3·33. To display its flexibility, the initial state is self-defined.

For complex motion case 1, the video is shown:

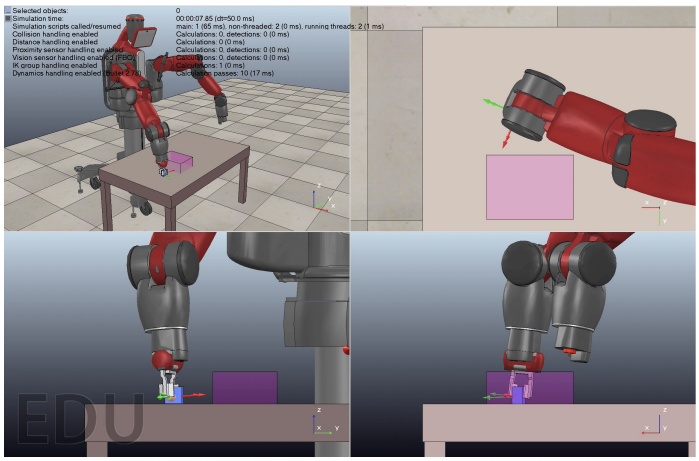
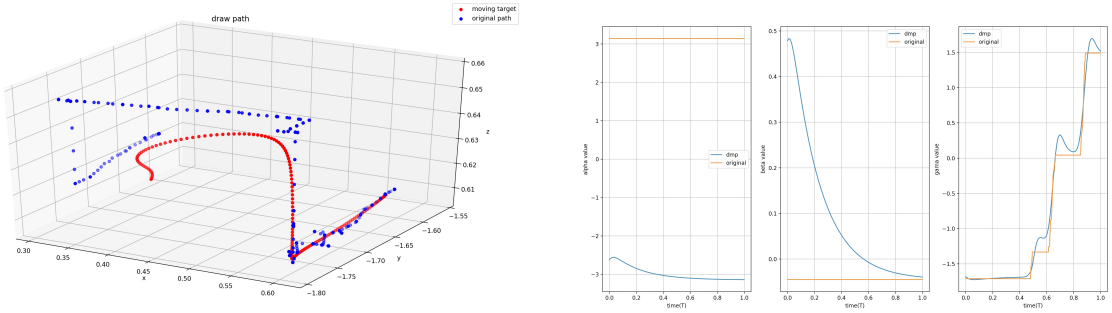


Figure 3·44: Video of modified complex motion case 1.

The visible trajectories and Euler angles of rotation are:



(a) (b)

Figure 3·45: 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:

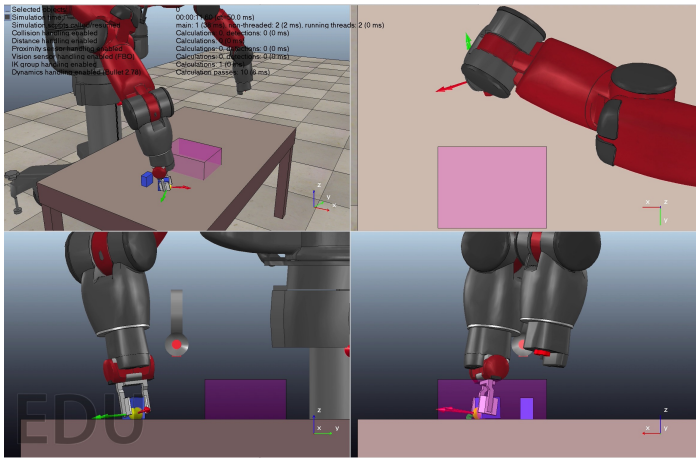
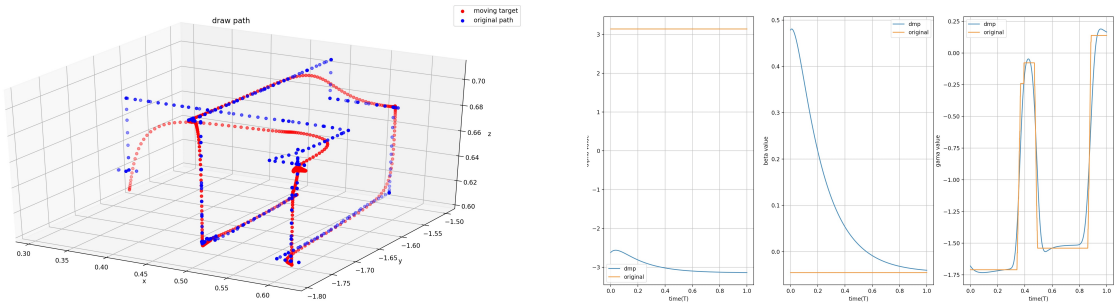


Figure 3·46: Video of modified complex motion case 2.

The visible trajectories and Euler angles of rotation are:



(a) (b)

Figure 3·47: 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 increment of initial coordinates are 50% of the cube's length, height and width. And they work for both cases. Also, the angles can vary around 3 degrees for sensitive rotations and more than 30 degrees for the stabilization of angles. In addition to the difference coordinates or Euler angles, DMPs are also very sensitive to time, too short simulation time may lead to failure. Likewise, if we give the system enough long time to stabilize, the system will reach to be stable eventually. Therefore, larger number of generated points generally increases the accuracy and higher number of basis functions can improve the similarity between the original trajectories and DMPs-generated ones.

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 3·44 and 3·46, the initial state or the goal state can be transformed or scaled. Furthermore, according to the result shown in figure 3·44, if we can divide high complexity motion into several lower complexity motions and combine them later, we are able to accomplish extremely complicated task with high accuracy. Moreover, according to the coordinates, velocities and Euler angles, the real-life implementations are supposed 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 complicated 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 and Exponential Map

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.

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

Haoying Zhou

1079 Commonwealth Avenue
Boston, MA 02215
(857) 800 5488
zhouhaoying96@outlook.com

BU Robotics Lab
Boston University
Boston, MA 02118
jackzhy@bu.edu

Academic Training:

05/2020(expected)	M.S. Boston University, Boston, MA; Mechanical Engineering
06/2018	B.S. Beijing Institute of Technology, Beijing, China; Mechanical Engineering

Master's Research:

Title: Imitation Learning With Dynamic Movement Primitives
Thesis advisor: Calin A. Belta, Ph.D.
Defense date: April 10th, 2020
Summary: In this work, discrete DMPs is utilized as the framework of the whole system. For more effective learning, a weighted learning algorithm called Local Weighted Regression (LWR) is implemented. To achieve the goal, the weights are firstly trained using DMPs framework as well as LWR. Then, regard the weights as classifiers 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. Moveover, the task-oriented movements can always be successfully stabilized even though there are some spatial scaling and transformation as well as time scaling.