Skill Transfer between Humans and Robots Based on Dynamic Movement Primitives and Sparse Autoencoder

By

Mingqi Li

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Electrical Engineering

May, 2017

Nashville, Tennessee

Approved:

Richard Alan Peters, Ph.D.

Kazuhiko Kawamura, Ph.D.

To my advisor, partners and my family.

Thanks for your helps.

ACKNOWLEDGMENTS

The data used in this thesis was obtained from mocap.cs.cmu.edu.

The database was created with funding from NSF EIA-0196217.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

INTRODUCTION

With the rapid development in industry, robots are not only applied in manufacturing. Robotics companies are increasingly concentrating on the service business [10, 23]. Many of the robots have humanoid characteristics and have features of human behaviours [21, 8]. For a simple example, the behaviors of picking up a cup on the desk then putting it on the ground and taking a bowl in the cabinet then placing it on the table will look different. However, we can classify these two series behaviours as *picking & placing*. Dynamic Movement Primitive (DMP) are mathematical models that can be used to generalize such tasks.

Nowadays, robots are not same as personal computers whose operating systems are limited to a few such as Windows, Mac OS and Linux. There are thousands of different robots on the market. Many of them need to complete the same set of tasks [27]. How to transfer behaviours between robots efficiently is the primary objective of this thesis. The idea is to decompose a task into a sequence of behaviors – simple sensory coupled actions – that can be defined for each robot. Different robots can perform the same task by sequencing the same behaviors. Firstly, we can calibrate robot actions with human actions through a series calibration behaviours. We can also calibrate two arbitrary robots with this method. Then, a sparse autoencoder is used to produce the library of basis behaviours which can be maintained easily. The behavior library can be implemented on different robots so that the DMP algorithm can transfer skills between them

The behavior library for each robot depends on its kinematics. Part of our work was to use kinematics to decide whether the robot can achieve the target we expect.

1.1 Related Work

This work used the Microsoft Kinect. In [16, 34], the Kinect SDK and programming skills were described. Programmers at Microsoft wrote a program to detect human skeletons which are the base of our code to detect joints. Livingston, Mark A et al. [22] prove the good performance of Kinect in skeleton detecting.

In the section on forward kinematics, we use DH-parameters invented by Jaques Denavit and Richard Hartenburg [12], which model a robot's links and joints. Khalil, Wisama et al. [18], Siciliano, Bruno et al. [28] and Craig, John J [6] used a modified DH-parameters, but the core idea is same [35].

Inverse kinematics (IK) algorithms can be characterised as analytical and numerical. The analytical method can be used only with robots that have 6 or fewer degrees of freedom (DOF) [30]. Moreover, analytical IK is specific to a given robot model. So it cannot be generalised to an arbitrary robot. The numerical method is more straightforward and is more easily generalized. There are three kinds of numerical algorithms which have been proved to solve the inverse kinematics problem. The first type is based on the Newton-Raphson method [3, 7]. The primary problem of this method is when the Jacobian matrix is singular, the algorithm will be unstable. Some modified algorithms have been developed to overcome this disadvantage [4, 31, 32]. The second type is based on optimization techniques. The core idea is to solve an equivalent minimization problem [9, 4, 11]. The biggest problem for algorithms of this type is the computational complexity of optimisation [33]. The last type is based on heuristic direct search techniques, but has higher computational than the second type.

Ijspeert, Nakanishi, & Schaal describe DMP in [14, 15]. In [13], Ijspeert et al. add an online learning model to DMP, which can make a robot avoid obstacles when reproducing behaviours. Pastor et al. illustrate in detail how to extend DMP into multiple DOF robots [26]. Petar Kormushev et al. control a robot's end effector's direction with DMP combined with EM-based reinforcement learning [19].

Sparse Autoencoder is a technique based on an artificial neural network for unsupervised learning [37]. It calculates a set of bases to represent the original data. Andrew Ng describes the structure and use of the algorithm in his lecture notes [25].

In entire process of generating behaviours, Huan Tan and Kazuhiko Kawamura proposed a framework to enable robots systematically learn how to integrate perception, tasks and behavior from experience in social settings [29].

1.2 Problem & Solution

The problem we need to solve can be described into 3 parts:

1. What robots were used and how were they to be controlled? In this thesis, we mainly operate Yaskawa Motoman HP3JC and Rethink Robotics Baxter. The Robot Operating System was used to build the connection between computer and robots. Scripts can be executed through ROS to operate robots. Within ROS, robotic kinematics were used to achieve the movement of robots. Forward kinematics solve the problem how to control robots with every joint's angle. Inverse kinematic solve the inverse problem which is how to compute every joint's angle to move the end effector to a specific pose (position and orientation). In forward kinematics, DH-parameters were used to build the robot model. For inverse kinematics, both the CCD method and the Jacobian method were used.

2. How to record human behaviours and transfer them to robots? Two ways were used to acquire human motion behaviours. The Kinect system was used to record a researcher's hand trajectory. And we used data from the CMU graph lab motion capture database. Before transferring behaviours, applied Scaling Iteritive Closet Point (SICP) to calibrate robot and human to make the behaviours correspond. After that we used the DMP algorithm to transfer the skill. DMP can calculate a similar trajectory to a different target.

3. How to increase the efficiency of DMP? DMP is a time-consuming process, especially complex trajectory, because it computes parametric kernels to make the trajectory match the details of the original trajectory as close as possible. We use Sparse Autoen-

coder to construct a dictionary of behaviors and use DMP to calculate every segment of the dictionary. Behaviors in the dictionary are simpler than the entire trajectory we need to imitate. They form a basis set of DMP trajectories from which more complicated tasks can be constructed. So, we can use fewer kernels to fit the basis and achieve better performance. After that, we can combine the DMP trajectories of the bases in the dictionary to rebuild the trajectory. That is a linear superimposed process and cost less time than directly applying DMP to the original trajectory. Rather than full trajectories, we just need to maintain the DMP of the set of bases. Moreover, this method can smooth behaviours and filter out unwanted jitter from the original observed trajectories, which are from human or electronic sample process.

Chapter 2

PLATFORM DESCRIPTION

In this chapter, three main aspects will be introduced. At first, we illustrate two robots we used in this research. Then a system which helps us operate robot will be discussed. After that, two methods are applied to collect experimental data. The first method is using Kinect to sample behaviours. The second method is to use data from the CMU human motion capture database.

2.1 Robot Platform

In this thesis research, two robot systems were used. HP3JC has fewer DOFs. So it imitates human behaviour stiffly. Baxter has higher DOFs; its arms are designed more like people.

2.1.1 Yaskawa Motoman HP3JC

Yaskawa Motoman is a Japanese robotics company well known for its industrial manipulators [36]. HP3JC is a compact, high speed and high accuracy 6 DOFs robotic arm [20], which is shown in Figure 2.1[1]. HP3JC can be controlled by its teach pendant. There is one switch to adjust mode to control the robotic arm. Three modes can be chosen:

"*remote*" mode allows operators to control it with ROS through the network.

"*play*" mode can execute scripts which is stored in the pendant. The robotic arm can redo motions according to every joint's velocities, movement time and angles. Scripts can be created by ROS or by the third mode "*teach*".

In "*teach*" mode, we can use the button to control the velocity, direction and processing time for robot's every joint.

[1] https://www.codeproject.com/Articles/317974/KinectDepthSmoothing

Figure 2.1: Yaskawa Motoman HP3JC

2.1.2 Rethink Robotics Baxter

Baxter is an industrial robot built by Rethink Robotics [17]. Baxter is a two-armed robot with an animated face, which is shown in Figure 2.2[2]. Both arms are 7 DOFs which can be operated to control arm's position. In this thesis research, the only right arm will be used.

In Figure 2.3[3], lengths between every joint are illustrated in (b); joint names are showed in (c) and (d).

Every joint's constraints are showed in Table 2.1[4].

2.2 Robot Operating System

ROS (Robot Operating system) which was originally developed by the Stanford Artificial Intelligence Laboratory in 2007 is a software system for robot programming, simulation, and control [39]. ROS provides services like low-level communication and distributed control.

[2]http://www.hizook.com/blog/2012/09/18/baxter-robot-rethink-robotics-finally-unveiled
[3]http://sdk.rethinkrobotics.com/wiki/Hardware_Specifications
[4]http://sdk.rethinkrobotics.com/wiki/Hardware_Specifications

Figure 2.2: Rethink Robotics Baxter

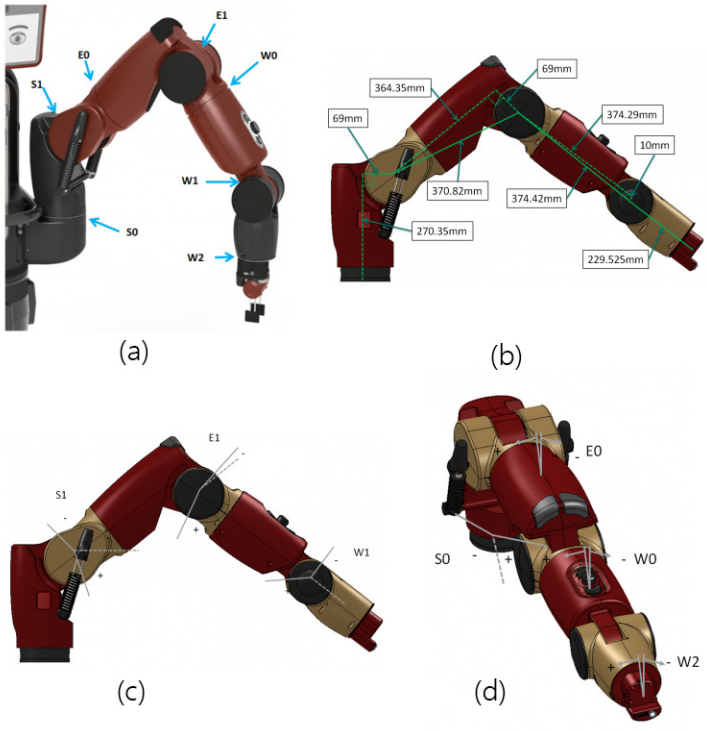


Figure 2.3: Baxter Hardware Specifications

Table 2.1: Baxter joints constraints

Joint Name	Min limit (rad)	Max limit (rad)
S0	-1.7016	+1.7016
S1	-2.147	+1.047
E0	-3.0541	+3.0541
E1	-0.05	+2.618
W0	-3.059	+3.059
W1	-1.5707	+2.094
W2	-3.059	+3.059

The core of ROS comprises four parts [2]:

1. *Node*. Nodes are executable files, which can be compiled with many advanced languages. A ROS system typically combines a number of different nodes. This method of construction makes the entire system easier to observe.

2. *Message*. Communications between different nodes are based on transfer messages.

3. *Topic*. A topic is a named bus or communications channel. Messages are transferred through topics using a "Publish & Subscribe" protocol. A node can publish messages to a topic. Other nodes can subscribe to specific messages from topics. In addition, publishers and subscribers don't notice the existence of each other, which makes the entire system easier to maintain.

4. *Service*. A service is another means of communication. Services allow nodes to send a request and receive a response.

In this thesis, ROS is used to operate the Motoman robot arm and Baxter. Figure 2.4 shows the simulator of Baxter.

2.3 Kinect & Kinect SDK

The kinect is a motion sensing input device developed by Microsoft for the Xbox 360, Xbox One video game system and Windows PCs. Two cameras are combined in Kinect; one is a RGB color camera. The other is a depth sensor capture system which uses an infrared emitter and an infrared CMOS camera [38].

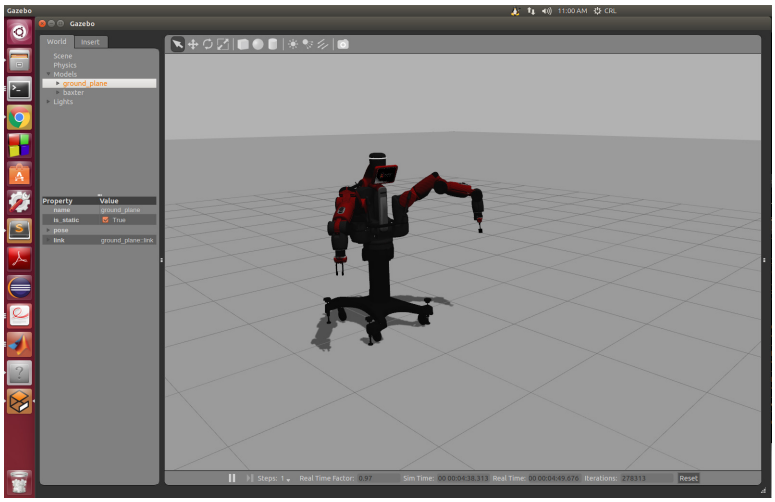
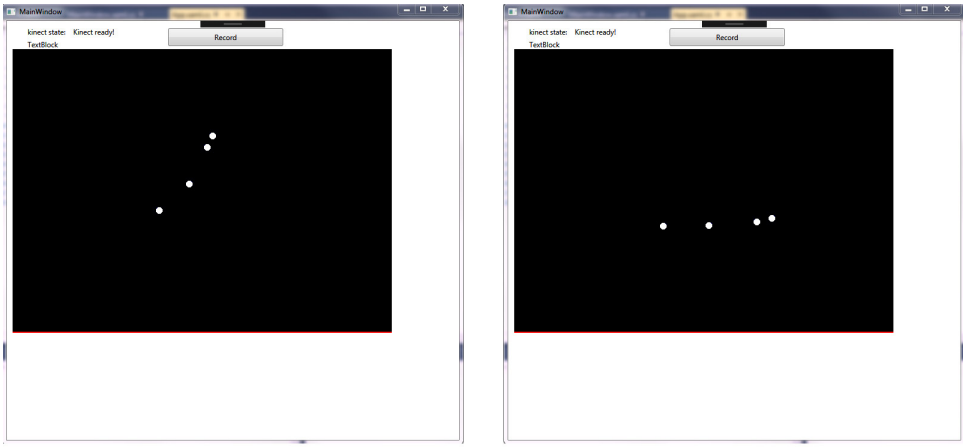


Figure 2.4: Baxter Simulator in ROS



(a) (b)

Figure 2.5: Two sample positions collected by Kinect

The Kinect SDK includes code samples and a help document for developers.

Skeleton Basics which can track and display skeletons for up to two players is one sample from the Developer Toolkit. This sample was modified by us to record the right arm's joints. In Figure 2.5, white points represent the joints' positions. The posture in this figure is upper raise (a) and lateral raise (b). Shoulder, elbow, wrist and hand coordinates are saved as ASCII files.

2.4 CMU Graph Lab Motion Capture Database & ASF/AMC File System

In this thesis, we also used the Motion Capture Database from CMU [1]. In this database researcher use markers to mark 31 different joints and bones of human bodies. Relationships between every bone and joint are defined in an ASF file. The motion data is stored in an AMC file.

The ASF file is a skeleton definition file for humans' body. In an ASF file, important data is *root*, *bonedata* and *hierarchy*[1].

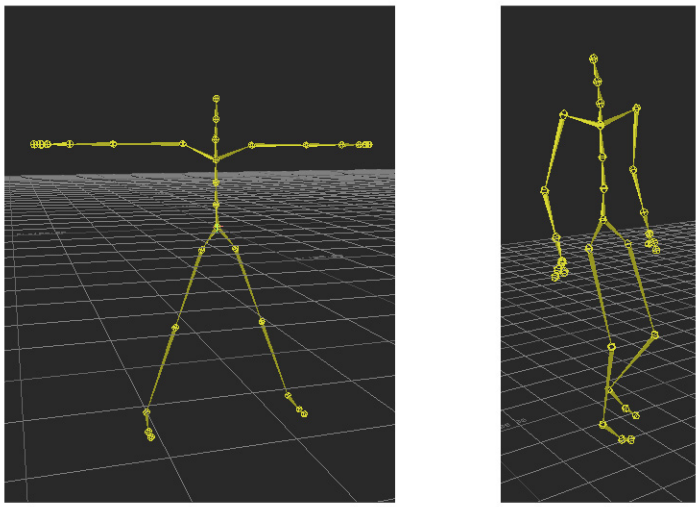
The "*root*" section defines a specific segment to be the root of the skeleton hierarchy of whole skeleton system. The key word "*position*" and "*orientation*" define the start position and the local coordinate system's orientation of the root.

The "*bonedata*" section contains data for every joint and bone. Keywords "*direction*" and "*length*" define the position of segment with respect to a parent segment. The "*axis*" defines a local coordinate system for the specific segment, which is rotated from its parent's coordinate system. "*dof*" and "*limits*" introduce a segment's degrees of freedom and every degree's constraints.

The "*hierarchy*" section defines every segment's parent and children.

The AMC file contains motion data which is defined by rotation angle except for the "*root*" segment. In the "*root*" section, the first three values define the coordinate system's origin. The other three values define the coordinate system's orientation. In other segments, every value represents the rotation angle for different DOFs. Figure 2.6 shows the original posture (a) and walk posture (b) which is one frame of the walking process. The display software we used is *Motion Builder*.

In this thesis, we used the HDM05 database toolkit [24] to extract every joint's coordinate based on a global system. We choose some motion files including wash windows and boxing behaviours from CMU Motion Capture database. Sample frames of these two behaviours are shown in Figure 2.7. Because our robotic arm can imitate one human arm's behaviour, we just use the right arm data.



(a) (b)

Figure 2.6: Two postures defined by ASF and AMC file

(a) Wash windows

(b) Boxing

Figure 2.7: Sample frames of boxing and washing window

Chapter 3

ROBOTIC KINEMATICS

In this chapter, forward and inverse kinematics are described. Forward kinematics is using every joint's rotation angle to control robotic arm movement. Inverse kinematics determine the joint angles necessary for the robot's end effector to remove to a specific pose.

3.1 Forward Kinematics

Forward kinematics are used to describe the end effector coordinate of the robotic arm. We use DH-parameters to find the translation and rotation matrix that describe the relationship between end effector and base.

3.1.1 DH-Parameters

DH-parameters were introduced by Jaques Denavit and Richard Hartenburg [12, 35], as a simple method to model a robot's links and joints. The idea is to assign and rotation a coordinate system to every joint, then connect adjacent joints with a translation vector. By combining each transformation from base to end effector, the entire transformation can be derived.

The first step to calculate the robot's transform matrix is to assign X axis and a Z axis to every joint. The base coordinate system's Z_0 axis is perpendicular to the ground. The X_0 axis and Y_0 axis can be set arbitrarily, but all of the coordinate systems must be right-handed system. For every joint, the Z_i axis is in the direction of the joint rotation axis. The X_i axis is parallel to the common normal of Z_i and Z_{i-1}. The direction is from Z_{i-1} to Z_i. The Y_i axis is defined by the directions of X_i and Z_i.

The second step is to assign four parameters of every joint. The first parameter is θ_i which defines the angle between X_i and X_{i-1}. When X_{i-1} rotates θ_i around Z_{i-1}, X_{i-1} can

be paralleled with X_i. The rotation matrix of this step is

$$\mathrm{Rot}_{z_{i-1}}(\theta_i) = \left[\begin{array}{ccc|c} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The second parameter is d_i which defines the distance between X_{i-1} and X_i. After the first step's rotation, the X_{i-1} and X_i axes are parallel. After translation along the Z_{i-1}, X_{i-1} and X_i will be in the same line. The translation matrix is

$$\mathrm{Trans}_{z_{n-1}}(d_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_n \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The third parameter is a_i which defines the distance between the origin of X_{i-1} and X_i. The translation matrix is

$$\mathrm{Trans}_{x_n}(a_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & a_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The fourth parameter is α_i which defines the angle between Z_{i-1} and Z_i. The rotation matrix is

$$\mathrm{Rot}_{x_n}(\alpha_n) = \left[\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_n & -\sin\alpha_n & 0 \\ 0 & \sin\alpha_n & \cos\alpha_n & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

So, the transfer matrix from the $i-1^{th}$ joint to the i^{th} joint is

$$^{i-1}T_i = \text{Rot}_{z_{i-1}}(\theta_i) \cdot \text{Trans}_{z_{i-1}}(d_i) \cdot \text{Trans}_{x_i}(a_i) \cdot \text{Rot}_{x_i}(\alpha_i)$$

$$^{i-1}T_i = \left[\begin{array}{ccc|c} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & r_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & r_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ \hline 0 & 0 & 0 & 1 \end{array}\right]$$

The entire transform matrix is

$$^{0}T_i = {}^{0}T_1 \cdot {}^{1}T_2 \cdot {}^{2}T_3 \cdots {}^{i-1}T_i \tag{3.1}$$

For Motoman robot, the DH-parameters are showed in Table 3.1.

Table 3.1: Motoman DH-parameters [20]

Joint Number	$\theta(rad)$	$d(mm)$	$a(mm)$	$\alpha(rad)$
1	θ_1	157	0	$-\frac{\pi}{2}$
2	$\theta_2 - \frac{\pi}{2}$	0	260	$\frac{\pi}{2}$
3	θ_3	0	30	$-\frac{\pi}{2}$
4	θ_4	-270	0	$\frac{\pi}{2}$
5	θ_5	0	0	$-\frac{\pi}{2}$
6	θ_6	-135	0	0

For Baxter, the DH-parameters are shown in Table 3.2.

After we assign the DH-parameters to the two robots, the end effector's coordinates based on every joint's angle can be calculated. The Motoman's end effector coordinates

Table 3.2: Baxter DH-parameters [20]

Joint Name	θ (rad)	d (mm)	a (mm)	α (rad)
S0	θ_1	270.35	69	$-\frac{\pi}{2}$
S1	$\theta_2 + \frac{\pi}{2}$	0	0	$\frac{\pi}{2}$
E0	θ_3	364.35	69	$-\frac{\pi}{2}$
E1	θ_4	0	0	$\frac{\pi}{2}$
W0	θ_5	374.29	10	$-\frac{\pi}{2}$
W1	θ_6	0	0	$\frac{\pi}{2}$
W2	θ_7	280	0	0

are shown as Equation 3.2, 3.3 and 3.4.

$$\begin{aligned} E_x =& 30cos(\theta_1)sin(\theta_2)sin(\theta_3) - 260cos(\theta_1)sin(\theta_2) - 135sin(\theta_1)sin(\theta_4)sin(\theta_5) - 30\cdot \\ & cos(\theta_1)cos(\theta_2)cos(\theta_3) - 270cos(\theta_1)cos(\theta_2)sin(\theta_3) - 270cos(\theta_1)cos(\theta_3)sin(\theta_2) - \\ & 135cos(\theta_1)cos(\theta_2)cos(\theta_5)sin(\theta_3) - 135cos(\theta_1)cos(\theta_3)cos(\theta_5)sin(\theta_2) + 135cos(\theta_1) \\ & cos(\theta_2)cos(\theta_3)cos(\theta_4)sin(\theta_5) - 135cos(\theta_1)cos(\theta_4)sin(\theta_2)sin(\theta_3)sin(\theta_5) \end{aligned} \tag{3.2}$$

$$\begin{aligned} E_y =& 135cos(\theta_1)sin(\theta_4)sin(\theta_5) - 270cos(\theta_2)sin(\theta_1)sin(\theta_3) - 270cos(\theta_3)sin(\theta_1)sin(\theta_2) - \\ & 260sin(\theta_1)sin(\theta_2) + 30sin(\theta_1)sin(\theta_2)sin(\theta_3) - 30cos(\theta_2)cos(\theta_3)sin(\theta_1) - \\ & 135cos(\theta_2)cos(\theta_5)sin(\theta_1)sin(\theta_3) - 135cos(\theta_3)cos(\theta_5)sin(\theta_1)sin(\theta_2) + \\ & 135cos(\theta_2)cos(\theta_3)cos(\theta_4)sin(\theta_1)sin(\theta_5) - 135cos(\theta_4)sin(\theta_1)sin(\theta_2)sin(\theta_3)sin(\theta_5) \end{aligned} \tag{3.3}$$

$$\begin{aligned} E_z =& 260cos(\theta_2) + 270cos(\theta_2)cos(\theta_3) - 30cos(\theta_2)sin(\theta_3) - 30cos(\theta_3)sin(\theta_2) - \\ & 270sin(\theta_2)sin(\theta_3) - 135cos(\theta_5)sin(\theta_2)sin(\theta_3) + 135cos(\theta_2)cos(\theta_3)cos(\theta_5) + \\ & 135cos(\theta_2)cos(\theta_4)sin(\theta_3)sin(\theta_5) + 135cos(\theta_3)cos(\theta_4)sin(\theta_2)sin(\theta_5) + 157 \end{aligned} \tag{3.4}$$

The Baxter's end effector coordinate is showed as Equation 3.5, 3.6 and 3.7

$$
\begin{aligned}
E_x =& 69cos(\theta_1) + (1822cos(\theta_1)cos(\theta_2))/5 - 69sin(\theta_1)sin(\theta_3) - 10cos(\theta_5)(cos(\theta_4) \\
& sin(\theta_1)sin(\theta_3) + cos(\theta_1)cos(\theta_2)sin(\theta_4) + cos(\theta_1)cos(\theta_3)cos(\theta_4)sin(\theta_2)) - \\
& (459cos(\theta_6)(sin(\theta_1)sin(\theta_3)sin(\theta_4) - cos(\theta_1)cos(\theta_2)cos(\theta_4) + cos(\theta_1)cos(\theta_3) \\
& sin(\theta_2)sin(\theta_4)))/2 - (3743sin(\theta_4)(sin(\theta_1)sin(\theta_3)) + cos(\theta_1)cos(\theta_3)sin(\theta_2)))/10 - \\
& 10sin(\theta_5)(cos(\theta_3)sin(\theta_1) - cos(\theta_1)sin(\theta_2)sin(\theta_3)) - (459sin(\theta_6)(cos(\theta_5)(cos(\theta_4) \\
& sin(\theta_1)sin(\theta_3) + cos(\theta_1)cos(\theta_2)sin(\theta_4) + cos(\theta_1)cos(\theta_3)cos(\theta_4)sin(\theta_2)) + sin(\theta_5) \\
& (cos(\theta_3)sin(\theta_1) - cos(\theta_1)sin(\theta_2)sin(\theta_3))))/2 + (3743cos(\theta_1)cos(\theta_2)cos(\theta_4))/10 - \\
& 69cos(\theta_1)cos(\theta_3)sin(\theta_2)
\end{aligned}
\tag{3.5}
$$

$$
\begin{aligned}
E_y =& 69sin(\theta_1) + (1822cos(\theta_2)sin(\theta_1))/5 + 69cos(\theta_1)sin(\theta_3) - 10cos(\theta_5)(cos(\theta_2)sin(\theta_1) \\
& sin(\theta_4) - cos(\theta_1)cos(\theta_4)sin(\theta_3) + cos(\theta_3)cos(\theta_4)sin(\theta_1)sin(\theta_2)) + (459cos(\theta_6) \\
& (cos(\theta_1)sin(\theta_3)sin(\theta_4) + cos(\theta_2)cos(\theta_4))sin(\theta_1) - cos(\theta_3)sin(\theta_1)sin(\theta_2)sin(\theta_4))) \\
& /2 + (3743sin(\theta_4)(cos(\theta_1)sin(\theta_3) - cos(\theta_3)sin(\theta_1)sin(\theta_2)))/10 + 10sin(\theta_5) \\
& (cos(\theta_1)cos(\theta_3) + sin(\theta_1)sin(\theta_2)sin(\theta_3)) - (459sin(\theta_6)(cos(\theta_5)(cos(\theta_2)sin(\theta_1) \\
& sin(\theta_4) - cos(\theta_1)cos(\theta_4)sin(\theta_3) + cos(\theta_3)cos(\theta_4)sin(\theta_1)sin(\theta_2)) - sin(\theta_5)(cos(\theta_1) \\
& cos(\theta_3) + sin(\theta_1)sin(\theta_2)sin(\theta_3))))/2 - 69cos(\theta_3)sin(\theta_1)sin(\theta_2) + (3743cos(\theta_2) \\
& cos(\theta_4)sin(\theta_1))/10
\end{aligned}
\tag{3.6}
$$

$$
\begin{aligned}
E_z = & 10cos(\theta_2)sin(\theta_3)sin(\theta_5) - 69cos(\theta_2)cos(\theta_3) - (3743cos(\theta_4)sin(\theta_2))/10 - (1822\cdot \\
& sin(\theta_2))/5 + 10cos(\theta_5)sin(\theta_2)sin(\theta_4) - (3743cos(\theta_2)cos(\theta_3)sin(\theta_4))/10 - (459\cdot \\
& cos(\theta_4)cos(\theta_6)sin(\theta_2))/2 - 10cos(\theta_2)cos(\theta_3)cos(\theta_4)cos(\theta_5) - (459cos(\theta_2)cos(\theta_3) \\
& cos(\theta_6)sin(\theta_4))/2 + (459cos(\theta_2)sin(\theta_3)sin(\theta_5)sin(\theta_6))/2 + (459cos(\theta_5)sin(\theta_2) \\
& sin(\theta_4)sin(\theta_6))/2 - (459cos(\theta_2)cos(\theta_3)cos(\theta_4)cos(\theta_5)sin(\theta_6))/2 + 2703/10
\end{aligned}
\tag{3.7}
$$

With the DH-parameters, forward kinematics build a model of the robotic arm.

3.2 Inverse Kinematics

In the forward kinematics, we have joint values:

$$
\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 & \cdots & \theta_n \end{bmatrix}
$$

The end effector coordinates can be calculated as:

$$
\mathbf{e} = \begin{bmatrix} e_1 & e_2 & e_3 \end{bmatrix}
$$

So, forward kinematics give us a function as:

$$
\mathbf{e} = f(\boldsymbol{\theta}) \tag{3.8}
$$

To control the robot's end effector to a specified target, we use $\mathbf{e}$ to calculate $\boldsymbol{\theta}$. The function can be represented as:

$$
\boldsymbol{\theta} = f^{-1}(\mathbf{e}) \tag{3.9}
$$

To solve the inverse kinematics function, we can obtain analytical solutions and nu-

merical solutions. But in a high DOF robotic arm, it will be too complex for an analytical solution. In this thesis research, two iterative methods are used to solve the f^{-1}.

The first algorithm is named as Cyclic Coordinate Descent, which is easy to understand and implement. The second algorithm is named as Jacobian Pseudo-Inverse method which is based on Jacobian matrix and Newton-Raphson method.

3.2.1 Cyclic Coordinate Descent Method

The CCD method is a directional search method. At first, we define the end effector joint as *No.1* joint and the base joint as the *No.N* joint. We can define the No.1 joint as the child and the No.2 joint as the parent, the No.3 joint is the No.2's parent, and so on. The common step of this method can be showed as follow:

1. Calculate the angle *ANG* between the vector of the current end effector's and it's parent joint (which is named as *A*) and the vector of the target end effector's and *A*. Then, rotating the *A* joint through angle *ANG*

2. If after the first step rotation the end effector cannot reach the target, redo the first step for *A*'s parent joint.

3. If after applying first and second step for the base joint the end effector still doesn't reach the target, stop this iteration and start the next iteration (repeat the first and second step).

The Pseudo code of CCD follows as Algorithm 1.

This algorithm which is based on a 2D environment has two problems. In the 2D situation, all joints' rotation axes are parallel. *crossVec*s which is calculated with *curVec*s and *tarVec*s are also parallel to the joints' rotation axes. However, in the 3D situation, the rotation axis which is calculated by $cross(curVec, tarVec)$ is different with the joints' axis under many situations. So the degree we need to rotate is different with the *turnDeg*. This is illustrated in Figure 3.1. Assume *curVec* is $(1,0,1)$ and *tarVec* is $(0,1,1)$, so the angle between these two vectors are α which is $60°$, $corss(curVec, tarVec)$ is $(-1,1,1)$ which is

Algorithm 1 Cyclic Coordinate Descent Pseudo code

```
// Variables
vector jointPos, curEnd, desiredEnd, tarVec, curVec, crossVec, nowTheta, nowQuaternion
double turnDeg
int curJointNum, triesCounter
// End variables
while triesCounter < triesThreshold & distantance(curEnd, desiredEnd) > distanceThreshold do
    // The condition of ending loop is the number of trying is bigger than the threshold or the distance of current end effector is smaller than the threshold.
    jointPos = funJointPos(curJointNum, nowTheta)
    // funJointPos can calculate coordinates of current joint we want to operate. This function use the result of DH-parameters introduced by last section.
    curVec = curEnd − jointPos
    tarVec = desiredEnd − jointPos
    turnDeg = acos(dot(curVec, tarVec))
    // Calculating the desired degree need to rotate.
    if turnDeg > threshold then
        crossVec = cross(curVec, tarVec)
        // Calculating the rotation axis
        Ensuring the rotation direction
        Updating nowTheta
    end if
end while
```

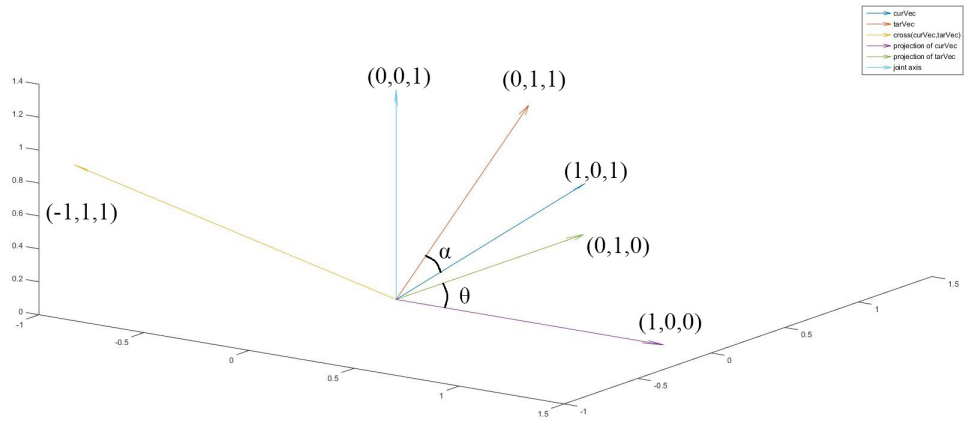


Figure 3.1: The CCD problem in 3D application

default axis of rotation. However, when the joint whose axis is $(0,0,1)$, we cannot rotate θ to make *curVec* and *tarVec*, because of the existence an angle between joint axis with default axis.

To solve this problem, the main idea is that when we need to rotate the joint *i*, *curVec* and *tarVec* can be both projected to the plane whose normal is the rotation axis of joint *i* as Figure 3.1. The angle θ of these two projection vectors is the degree that joint *i* need to rotate.

The second problem is that this algorithm doesn't consider the joint's limit. After few steps rotation, joints' angle may bigger or smaller than their limits. So, after every loop, $nowTheta$ must be checked to see if it is beyond the limitation. The angle of the joint must be set to the edge of limitation.

So, CCD method is improved as Algorithm 2.

3.2.2 Jacobian Pseudoinverse Method

Since Equation 3.9 is a non-linear equation, the Jacobian Pseudoinverse method is linearly approximated to reach the solution. The Jacobian matrix *J* is combined with derivatives of end effector's coordinates with respect to joint's angles. The Baxter's Jacobian

Algorithm 2 modified Cyclic Coordinate Descent Pseudo code

```
// Variables
vector jointPos, curEnd, desiredEnd, tarVec, curVec, crossVec, nowTheta, nowQuaternion
double turnDeg
int curJointNum, triesCounter
// End variables
while triesCounter < triesThreshold & distantance(curEnd, desiredEnd) > distanceThreshold do
  [jointPos, jointAxis] = funJointPos(curJointNum, nowTheta)
  curVec = curEnd − jointPos
  tarVec = desiredEnd − jointPos
  projcurVec = projection(curVec, jointAxis)
  projtarVec = projection(tarVec, jointAxis)
  turnDeg = acos(dot(projcurVec, projtarVec))
  if turnDeg > threshold then
    crossVec = cross(curVec, tarVec)
    if dot(crossVec, jointAxis) > 0 then
      // crossVec and jointAxis are in same direction
      if nowTheta + turnDeg > limitation then
        nowTheta = limitation
      else
        nowTheta = nowTheta + turnDeg
      end if
    else
      if nowTheta − turnDeg < limitation then
        nowTheta = limitation
      else
        nowTheta = nowTheta − turnDeg
      end if
    end if
  end if
end while
```

matrix can be presented as

$$J_{Baxtor} = \begin{bmatrix} \frac{\partial e_x}{\partial \theta_1} & \frac{\partial e_x}{\partial \theta_2} & \frac{\partial e_x}{\partial \theta_3} & \frac{\partial e_x}{\partial \theta_4} & \frac{\partial e_x}{\partial \theta_5} & \frac{\partial e_x}{\partial \theta_6} & \frac{\partial e_x}{\partial \theta_7} \\ \frac{\partial e_y}{\partial \theta_1} & \frac{\partial e_y}{\partial \theta_2} & \frac{\partial e_y}{\partial \theta_3} & \frac{\partial e_y}{\partial \theta_4} & \frac{\partial e_y}{\partial \theta_5} & \frac{\partial e_y}{\partial \theta_6} & \frac{\partial e_y}{\partial \theta_7} \\ \frac{\partial e_z}{\partial \theta_1} & \frac{\partial e_z}{\partial \theta_2} & \frac{\partial e_z}{\partial \theta_3} & \frac{\partial e_z}{\partial \theta_4} & \frac{\partial e_z}{\partial \theta_5} & \frac{\partial e_z}{\partial \theta_6} & \frac{\partial e_z}{\partial \theta_7} \end{bmatrix}$$

where e_x, e_y, e_z are the coordinates of end effector; θ_1 to θ_7 are joint's angles.

Because we calculate Baxtor and Motoman's DH-parameters, the coordinates of the end effector can be presented as a polynomial. We can directly take the first order derivative of the coordinate derivative joint angle. Another way to calculate the jacobian matrix numerically is as follow. One column of the Jacobian matrix can be presented as [5]:

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} = \begin{bmatrix} \frac{\partial e_x}{\partial \theta} & \frac{\partial e_y}{\partial \theta} & \frac{\partial e_z}{\partial \theta} \end{bmatrix}^T$$

We can add a small $\Delta\theta$ to θ, which is named as θ'. Therefore, the end effector's coordinate of $\boldsymbol{\theta'}$ is $\boldsymbol{e'}$, so

$$\boldsymbol{\Delta e} = \boldsymbol{e'} - \boldsymbol{e}$$

Now, we have:

$$\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{\theta}} \approx \frac{\boldsymbol{\Delta e}}{\boldsymbol{\Delta \theta}} = \begin{bmatrix} \frac{\Delta e_x}{\Delta \theta} & \frac{\Delta e_y}{\Delta \theta} & \frac{\Delta e_z}{\Delta \theta} \end{bmatrix}^T$$

The velocities of the end effector are:

$$\dot{\boldsymbol{e}} = J(\theta)\dot{\theta}$$

When we add the $\Delta\theta$ to the θ, the change in the end effector positions can be estimated as

$$\boldsymbol{\Delta e} = J\boldsymbol{\Delta \theta} \tag{3.10}$$

In order to determine the value of $\boldsymbol{\Delta\theta}$, we need to solve Equation 3.10 [5]. However, in many cases, this equation cannot be solved uniquely because it is rank deficient. Even if

it is invertible, in many cases, J is nearly singular. Therefore, when J is invertible we can temporarily set

$$\Delta\boldsymbol{\theta} = J^{-1}\Delta\boldsymbol{e} \tag{3.11}$$

Many methods can be applied to solve Equation 3.10 such as the transpose method. The idea is to replace the inverse of J with the transpose of J. In this method:

$$\Delta\boldsymbol{\theta} = \alpha J^{T}\boldsymbol{e} \tag{3.12}$$

where α is a scalar value. α needs to be set appropriately. In the case where α is sufficiently small and bigger than zero, the result of this method can be approximated with the original result.

The other method is named the Pseudoinverse method, which uses the pseudoinverse of J to replace the J^{-1}. This method is most approximate with the original equation. The pseudoinverse is also called the Moore-Penrose inverse which exists for all matrices even if they are rank deficient or not squared. A simple and accurate way to calculate the Moore-Penrose inverse is to use the Singular Value Decomposition (SVD). The Moore-Penrose inverse is shown as Algorithm 3 [5].

Algorithm 3 SVD method to solve Moore-Penrose inverse

$[U,S,V^T] = svd(J)$
$newS = \frac{1}{S}$
$J_{inv} = V^T newS^T U^T$

Because the function of inverse kinematics is non-linear. The Jacobian matrix with specific θ can be only applied near θ. So, the $\Delta\boldsymbol{e}$ should be as small as possible. If the target is too far from the current end effector, we need to subdivide the distance between the current coordinate and the target into small steps. After every update, $\boldsymbol{\theta}$ may be outside the joint limits. So, if one specific joint's angle is out of the limits, it will be set to a neutral value which is equal to half of the sum of positive constraint and the negative constraint.

The algorithm of Jacobian-Pseudoinverse method is illustrated as Algorithm 4.

Algorithm 4 Core of Jacobian-Pseudoinverse method to solve inverse kinematic

Output thetaR
Input intTheta, tarPos, conP, conN // *conP* and *conN* present positive and negative constraints of every joint.
$tries = 1$
$curTheta = intTheta$
$curPos = calPos(curTheta)$
// Calculating current position of specific $\boldsymbol{\theta}$
$err = tarPos - curPos$
while $abs(err) > threshold \& tries < tryThreshold$ **do**
  $J = jacobian(curTheta)$
  $invJ = INVJ(J)$
  $dTheta = invJ * err$
  if $dTheta + curTheta > conP || dtheta + curTheta < conN$ **then**
    $curTheta = (conP + conN)/2$
  else
    $curTheta = curTheta + dTheta$
  end if
  $curPos = calPos(curTheta)$
  $err = tarPos - curPos$
  $tries++$
end while

Chapter 4

ROBOTIC SKILL TRANSFER

With the development of anthropomorphic robots, demand to the robot is not only recording the action but also reproducing behaviours according to the actual environment. The Dynamic Movement Primitive algorithm is designed to solve this problem, which can find a set of environmental parameters to adjust a complex behaviour automatically without intervention from operators. The stability of this algorithm is proved by its inventor [13].

4.1 Dynamic Movement Primitives

The basic idea of DMP is using a dynamical system to simulate the original stable behaviour and add another force to achieve some specific goals such as reach different targets or to avoid obstacles. Specifically, the dynamical system which Ijispeert used in [13] is a spring system which is simple and effective. The equation is showed as:

$$\tau \dot{z} = \alpha_z(\beta_z(g-y) - \dot{y}) + f \tag{4.1}$$

$$\tau \dot{y} = z \tag{4.2}$$

In these two differential equations, τ is considered as time-scaling parameters. α_z and β_z are constants. y and g represent the current position and the goal position. f is the external force which results the desired behaviours. How to define f is an important question. In order to achieve more complex behaviour trajectories, f can be defined as the kernel function:

$$f(x) = \frac{\Sigma_{i=1}^{N} \psi_i(x) w_i}{\Sigma_{i=1}^{N} \psi_i(x)} x(g - y_0) \tag{4.3}$$

$$\psi_i(x) = exp(-\frac{1}{2\sigma_i^2}(x - c_i)^2) \tag{4.4}$$

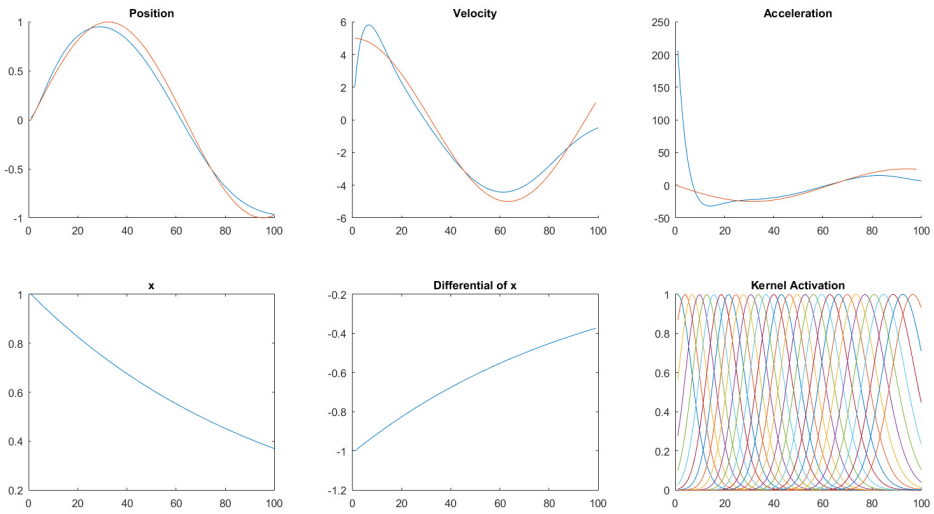


Figure 4.1: A sample of DMP

where w is the weight of ψ, y_0 is the initial position, σ_i and c_i are constants depended on the variance and center of the kernel function. In Function 4.4, ψ is a radial basis function (RBF) but it can be replaced by other kernels. The gaussian RBF kernel which we used in DMP is simple and effective.

In the original equation, external force f is time dependent. In order to allow straightforward coupling of multiple dynamical systems and the coordination of multiple DOF in one dynamical system, the *canonical system* is introduced. This solves the problem of time dependence in the dynamic system. In Figure 4.1, 3 plots in the first line show the position, velocity and acceleration of a specific trajectory which is generated by $sin(5 \cdot step)$. The red line is the original trajectory and the blue line represents the trajectory which is reproduced. In this figure, we just test its ability to recover the original trajectory. In the second line of this figure, the first and second plots show how the canonical system evolves over time. The third plot indicates the activation of the kernel which is a combination of 30 Gaussian kernels comprising the force term.

Another problem is to prove this algorithm can be effective in every application. In the [13], authors use BIBO (Bounded-Input, Bounded Output) stable theory and contraction

theory to prove the system is stable. The stability of this dynamic system means that when the external force f decays to zero, the system will, after a period of time, converge the goal position. Therefore, this property can ensure that the dynamic system will be able to move to the goal with certainty.

The final problem is to prove that original trajectory and reproduced trajectory are corresponding when the actual goal is different. This property is called *invariance*. To prove the *invariance* property of the dynamic system, we first classify its parameters. The first kind of parameters are the constants α_z, β_z and w_i which do not change when we reproduce behaviours. The second kind of parameters' τ and g, do change. To have the invariance we require, when we modify the value of τ and g, the trajectory should not be changed qualitatively. In [13], authors prove that after scaling τ and g, the behaviour of trajectory reproduction is topologically equivalent to the original one.

In Figure 4.2, we can easily observe the invariance of the dynamic system through the experimental results. The original trajectory is generated by $sin(5 \cdot step)$, which is represented by the light blue line. Its goal is -1. When we set goals to 1, 0.67, 0.33, -0.33 and -0.67 respectively, the reproduced result is as shown as this figure. It's easy to observe that all of these trajectories are similar. We can also find that when the goal is mirrored, entire reproduced trajectory will be mirrored to keep the *invariance* property.

We apply DMP to control a robotic arm. Three methods to embed DMP into multiple DOF systems are introduced in [13].

1. Every DOF has its own canonical system. The disadvantage is that every joint has no constraint the from other joints. Therefore, when the robotic arm attempts a complex behaviour, the joints may disturb each other.

2. Constructing coupling terms in canonical systems between each DOF. However, this method will lead to complex calculation in tuning coupling terms.

3. All joints share one canonical system and use their own transformation system. The advantage of this method is that it is easy to maintain and is simple in approach.

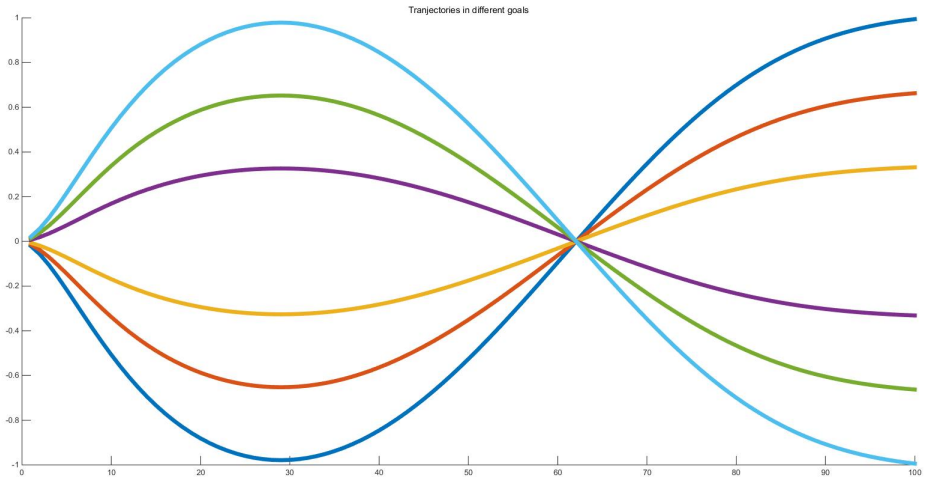


Figure 4.2: Invariance of DMP

The third method. In the implementation of DMP, the key step is to fit w_i. A supervised learning framework was introduced as follows:

1. Constructing the canonical system. The differential equation is:

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

where α_x and τ will be 1 in practice.

2. Extracting the goal g and initial state y_0. g and y_0 can be extracted from the desired trajectory position vector. The first and last element correspond to y_0 and g.

3. Obtaining velocity $\dot{y}$ and acceleration $\ddot{y}$. These can be obtained by the first order and the second order derivatives of position.

4. Ensuring the value of τ. In practice, $\tau = 1.05 \cdot 2\% \cdot V_{max}$, where V_{max} represents the maximum velocity.

5. Accessing the value of f_{target}. After rearranging the equation, we obtain:

$$f_{target} = \tau^2\ddot{y} - \alpha_z(\beta_z(g-y) - \tau\dot{y})$$

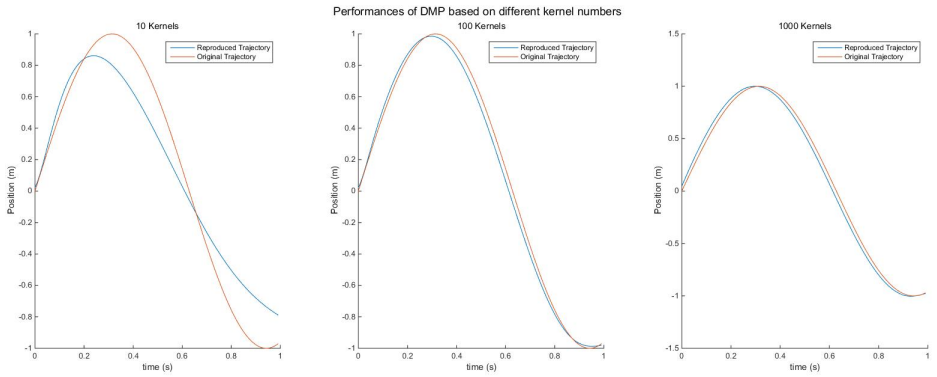


Figure 4.3: Performances for different kernel numbers

where α_z and β_z will be 25 and 6.25

6. Fitting weights w_i. The solution is calculated using the Locally Weighted Regression, which is represented as:

$$w_i = \frac{\boldsymbol{s}^{\boldsymbol{T}} \boldsymbol{\Gamma}_i \boldsymbol{f}_{target}}{\boldsymbol{s}^{\boldsymbol{T}} \boldsymbol{\Gamma}_i \boldsymbol{s}}$$

where

$$\boldsymbol{s} = \begin{bmatrix} x(1)(g-y_0) \\ x(2)(g-y_0) \\ \vdots \\ x(P)(g-y_0) \end{bmatrix}, \boldsymbol{\Gamma}_i = \begin{bmatrix} \Psi_i(1) & 0 & \cdots & 0 \\ 0 & \Psi_i(2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \Psi_i(P) \end{bmatrix}, \boldsymbol{f}_{target} = \begin{bmatrix} f_{target}(1) \\ f_{target}(2) \\ \vdots \\ f_{target}(2) \end{bmatrix}$$

After calculating the w_i, we can reproduce the behaviour with the dynamic system.

The performance of behaviour reproduction will be better if more kernels are used to calculate w_i. Figure 4.3 shows the performance between different kernels. In this figure, the behaviour of reproduction with 1000 kernels is much better than 10 kernels. The use of more kernels in the calculations will consume more time; the performance and the time of calculating are inversely proportional.

Chapter 5

DICTIONARY GENERATION & TRAJECTORY SYNTHESIS

To achieve good performance and better efficiency, we introduce the Sparse Autoencoder [25] into the dynamic system.

It is an artificial neural network that can decompose complex behaviour into a set of simple basis functions. The original behaviour can be recovered by linear superposition of the bases. The error of the reproduced behaviour is within acceptable limits. The set of basis function can be designed as a dictionary which can supply the elements which to combine into an arbitrary behaviour.

Two necessary conditions for constructing the dictionary are over-complete and unrelated. Sparse autoencoder builds a dictionary meets that these two conditions.

5.1 Sparse Autoencoder

Sparse Autoencoder is an unsupervised algorithm. Its most specific feature is that outputs and inputs are same in this network. The task of this network is to use linear combination of elements to represent the original data.

In general, the error of this task can be defined as:

$$J(w,b) = \frac{1}{m}\sum_{j=1}^{m}\left\| x^{(j)} - (\sum_{i=1}^{k} w_i^{(j)}\phi_i + b)\right\|^2 \tag{5.1}$$

where x is the set of original data vectors; ϕ represent the set of bases; w and b denote the weight vectors and constants of the combinations. Our target is to find a set of ϕ to make $J(w,b)$ minimum.

To avoid overfitting and to reduce weight decay amplitude, a weight decay term is added

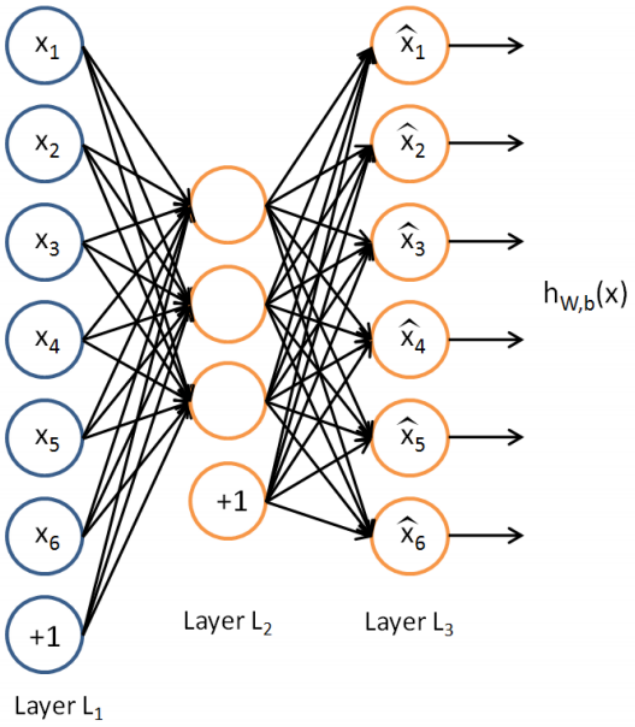


Figure 5.1: Structure of Sparse Autoencoder structure [25]

in to equation 5.1. The modified $J(w,b)$ is:

$$J(w,b) = \frac{1}{m}\sum_{j=1}^{m}\left\|\frac{1}{2}(x^{(j)} - (\sum_{i=1}^{k} w_i^{(j)}\phi_i + b))\right\|^2 + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(w_{ji}^{(l)})^2 \tag{5.2}$$

where $w_{ji}^{(l)}$ represents the weight parameter between unit i in layer l, and unit j in layer $l+1$; s_l is the number of units in layer l; n_l represents total number of layers in this network.

Another problem is how to prove the sparsity of our dictionary. In order to solve this problem, a sparse term is designed to modify the original error function.

$$KL(\rho||\hat{\rho}_j) = \rho log\frac{\rho}{\hat{\rho}_j} + (1-\rho)log\frac{1-\rho}{1-\hat{\rho}_j} \tag{5.3}$$

where ρ is the desired level of sparsity; $\hat{\rho}_j$ represents the average activation of hidden unit j.

When this sparsity is combined with the regularization term and the error defined above, the cost function is defined as:

$$J_{sparse}(w,b) = J(w,b) + \beta \sum_{j=1}^{m} KL(\rho||\hat{\rho}_j) \tag{5.4}$$

where β is the weight of the sparsity penalty term. To find the minimum solution of the cost function, a Back-propagation method [25] is used to learn the basis functions.

5.2 Process of Dictionary Generation & Trajectory Synthesis

After introducing the technique we used in dictionary generation and trajectory synthesis. Steps of this process will be described as followed.

The first step is applying random joint's angles to the robot models and human's model, aims to access numerous random trajectories which can cover almost all of behaviours in daily life.

Secondly, employing Back-propagation method to extract a set of over-complete bases from numerous data which is described in step one, which is showed as all segments in $LayerL_2$ in Figure 5.1. This step is called *Dictionary Generation*.

After generated dictionary, LASSO (Least Absolute Shrinkage and Selection Operator) [41] can be employed to solve the problem of calculating $w_i^{(j)}$ and b.

Chapter 6

EXPERIMENTAL RESULTS & ANALYSIS

6.1 Experiments for Inverse Kinematics

In this section, we test the performance of two methods for calculating each joint's angular position trajectory. The performance test is divided into two parts.

At first, we test whether these two methods can calculate the correct joints' angles to achieve specific targets. The platform we used was Baxter. We derive 5 points in Baxter's end effector working space randomly, which is showed as Figure 6.1 (a). Then we interpolate 95 points into these 5 points to make them smooth. So now we have 100 points in this trajectory which is shown in Figure 6.1 (b). 10,000 trajectories are prepared this way to form a test database. Our test strategy is to use the two methods to calculate the set of joint's corresponding to positions. So in one trajectory, we can obtain 100 sets of joint angles. Then we use forward kinematics to compute the end effector's position. If the average position error (distance between the original position and the calculated position) is less than 0.001 mm, we claim that the method is effective. Both methods' iteration number is set to 10000. The number of successful trials and the average error is showed as Table 6.1

Table 6.1: CCD and Jacobian Pseudo-inverse accuracy

Method	Effective Number	Average Error (mm)
CCD	9986	0.001087
Jacobian Pseudo-inverse	10000	0.000856

We can find that CCD method is not 100% effective. The CCD method may, under some circumstances, try to put the arm into an impossible position. A simple example in the 2D environment can explain this problem, which is also applicable in the 3D environment.

At first, given a 2-DOFs chain model where both DOFs are started with no rotation, we

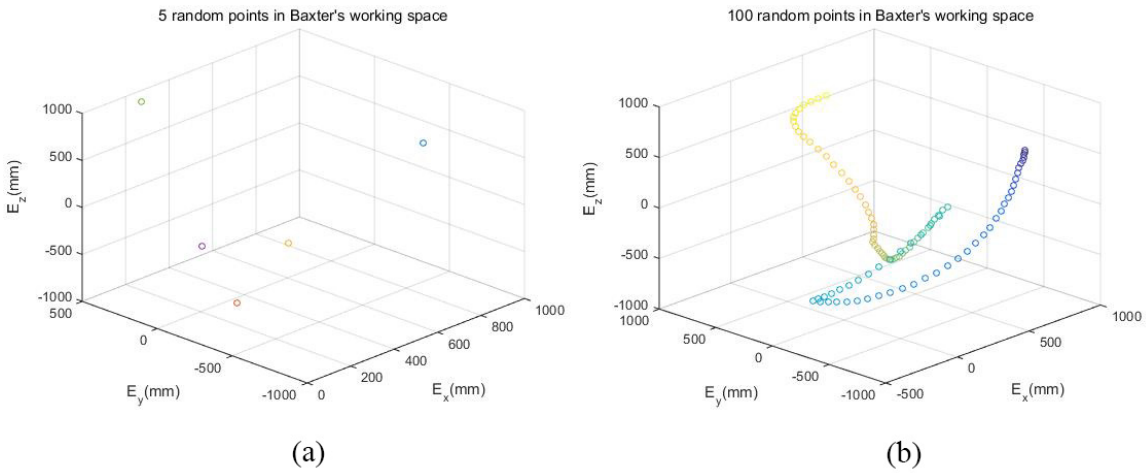


Figure 6.1: 5 and 100 points in Baxter's working space

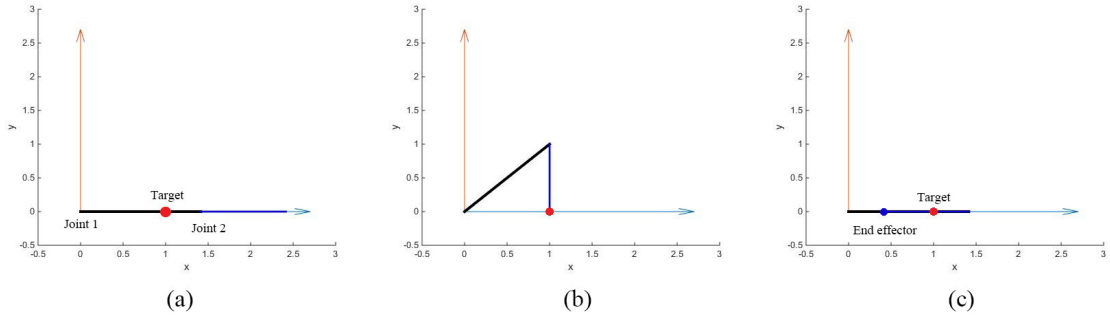


Figure 6.2: CCD's problem in practice

place the target at the position shown in Figure 6.2 (a). If it were to perform correctly, a valid solution would be as shown in Figure 6.2 (b). However, as we described in the CCD section, at first, joint 2 would be rotated 180° to the position which is shown in (c) where link 2 is on top of link 1. Note that both, *curVec* and *tarVec* are in line with the arm. Then, the algorithm, tries to rotate joint 1 to keep *curVec* and *tarVec* in one line. However, these two vectors were already in one line. So, joint 1 stays in its position (rotates 0 degrees). In sequence, joint 2 tries to rotate 180 degrees. Then, the end effector would be stacked in the position shown in (c). The end effector is not only not at the goal position but also the algorithm has tried to put the arm into an impossible configuration. That is the reason why CCD has a small possibility of failing when calculating the inverse kinematics.

After the accuracy test, we compute the average time, which represents the computational efficiency. Because the CCD method will fail in some specific situations, we took the average time over those trials which CCD should calculate successfully. The average computation time is shown in Table 6.2

Table 6.2: CCD and Jacobian Pseudo-inverse time in calculating

Method	Time in Calculation (s)
CCD	0.002754
Jacobian Pseudo-inverse	0.014950

We found that the time required by the Jacobian method was 5 times that of CCD's. In the Jacobian method, the step of solving the pseudoinverse of the jacobian matrix cost more than 80% of the total time. So, we have to notice that applying pseudoinverse to solve Equation 3.10 decreases efficient of the Jacobian method.

6.2 Experiments of DMP

In this section, we test the performance of the DMP algorithm. 200 frames of hand positions were sampled from the CMU database. These 200 frames included the hand positions of a person washing a window as shown in Figure 2.7 (a). At first, we reproduced this trajectory with the DMP algorithm. Figure 6.3, 3 plots in every line show the original trajectory (red line) and the reproduced trajectory (blue line) with different numbers of kernels. The reproduced trajectory was a better fit to the original trajectory when applying more kernels in DMP.

Figure 6.4 displays the original trajectory (red line) and the reproduced trajectory (blue line) with 100 kernels. After reproduced by DMP, the trajectory. Glitches in the sampling process were ignored by DMP to produce a smooth curve.

The next step was to test the performance of DMP when the goal was changed. Three values were added to the goal in the X, Y and Z directions. The reproduced 3-dimensional trajectory generated by DMP with 100 kernels is shown in Figure 6.5. The synthesized

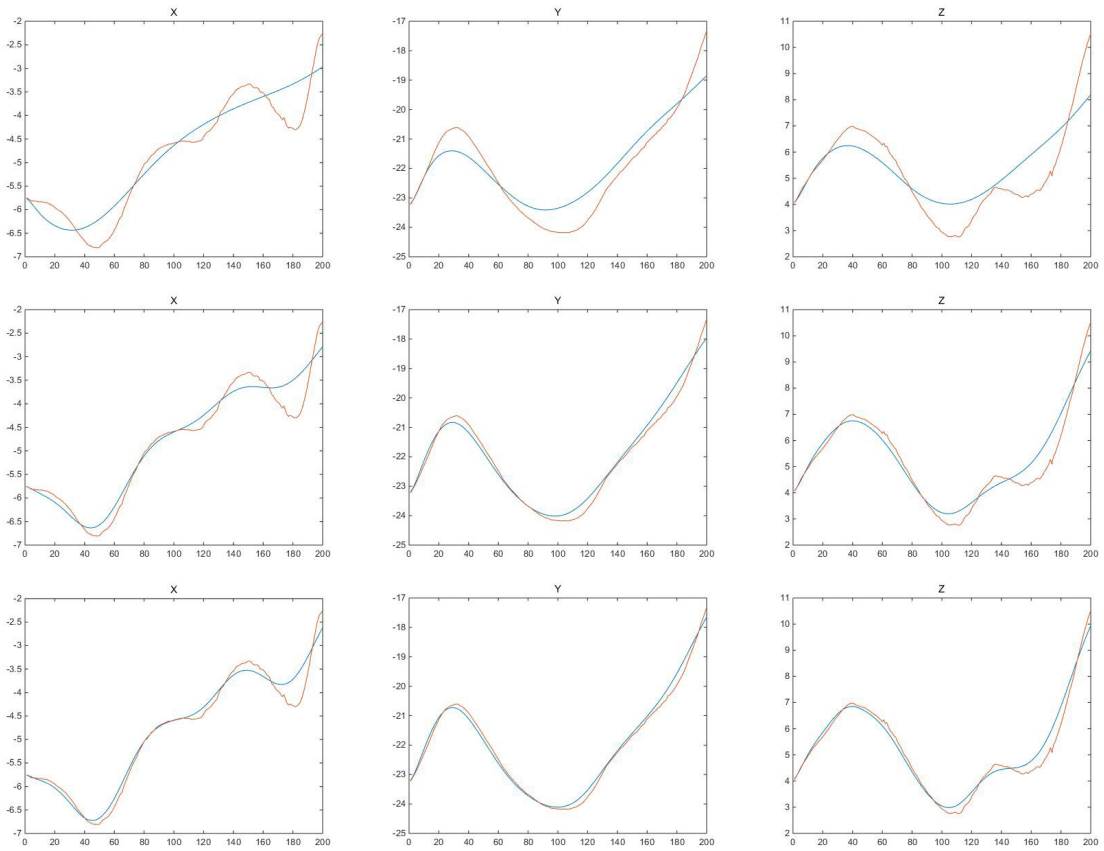


Figure 6.3: Reproduced trajectory of 3 dimensions with DMP in 10, 30, 50 kernels

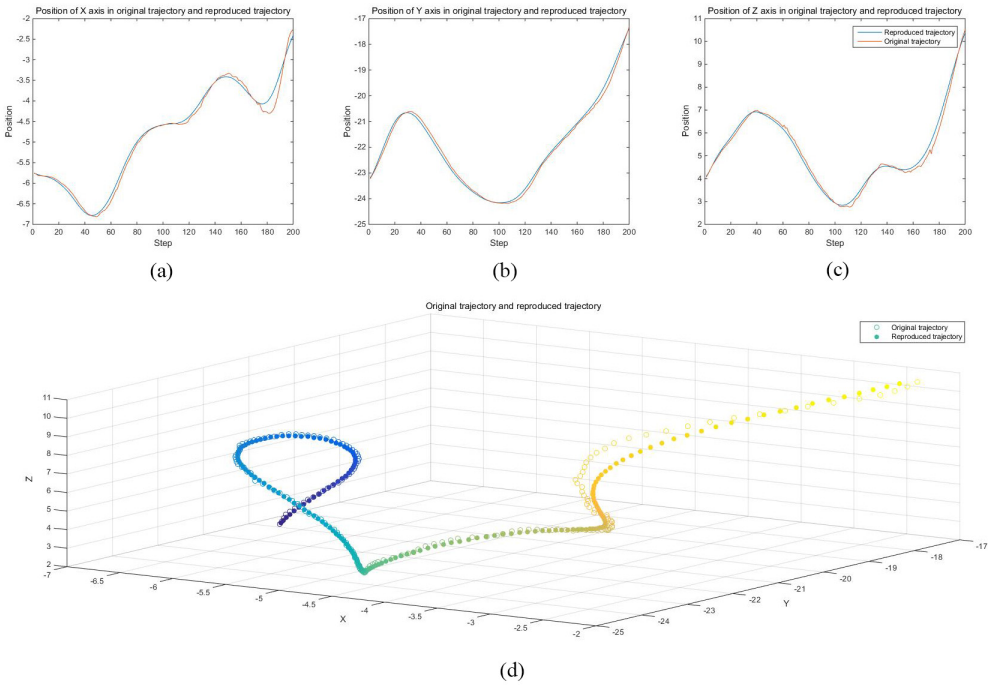


Figure 6.4: Reproduced trajectory of 3 dimensions and 3D plot with DMP in 100 kernels

trajectory kept attributes of the original while reaching different goal positions.

The DMP algorithm requires complex calculations, so it's a time-consuming process. Figure 6.6 shows the times required to calculate DMP with different numbers of kernels. The time has some randomness. However, the trend shows a linear increase in computation time as more kernels are used. In practice, more complex trajectories will need more kernels to calculate. Maintaining a high number of kernels to match a complicated trajectory is a very time-consuming process. That is why we introduced sparse autoencoder to build base for trajectory.

6.3 Experiments for Dictionary Creation & Skill Transfer

In this section, we describe transfer the human motion behaviour to the Motoman robot. At first, we need to build the data bases which describe the human behaviour trajectory. The human behaviour database is built based on the ASF and AMC file systems which the human motion capture information. Since we only need one arm. All joints except the

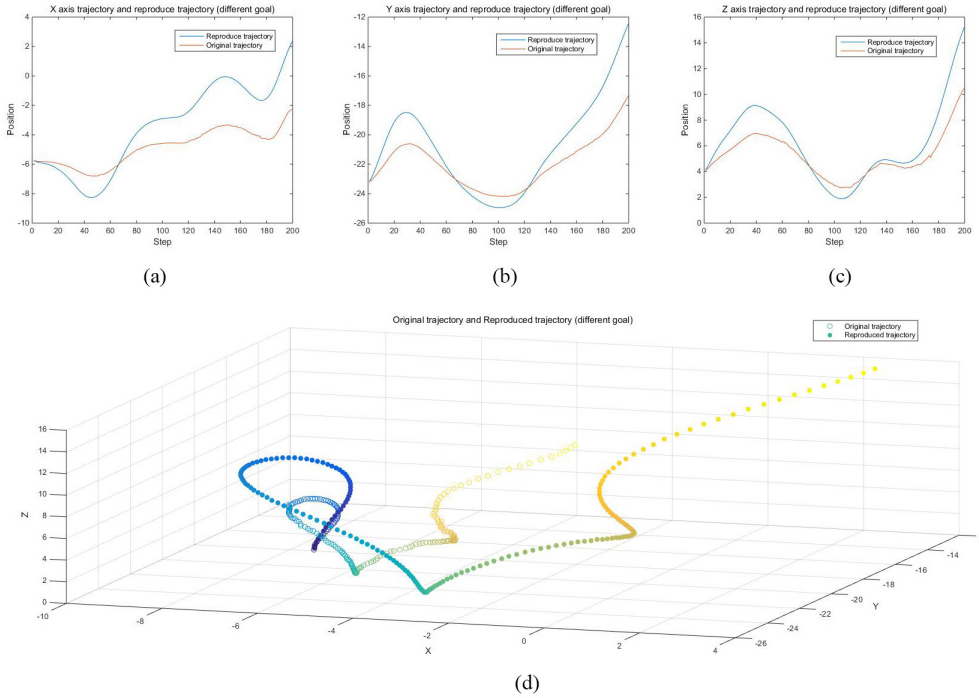


Figure 6.5: Reproduced trajectory of 3 dimensions and 3D plot with DMP in 100 kernels (different goals)

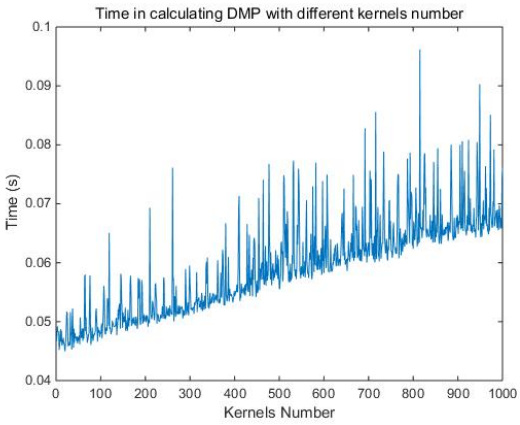


Figure 6.6: Time in calculating DMP with different kernels number

humerus, *radius*, *wrist* and *hand* were set to zero in position and orientation. Initially, we generated 5 sets of random angles as the joints' constraints for four motion segments. Then 95 sets of angles were interpolated between these 5 sets to ensure smoothness of motion 10000 samples we so prepared to build the data base of human trajectories.

Then the robot's behavior must be calibrated to the human motion so that their trajectories coincide. The method we used was the SICP (Scaled Iterative closest point) algorithm [40]. We designed a group of behaviour to calculate scale, rotation matrix and translation vector from human to Baxter. The scale is

$$s = 1.02$$

The rotation matrix is

$$R = \begin{bmatrix} -0.9980 & -0.0450 & -0.0430 \\ 0.0450 & 0.0451 & -0.9980 \\ -0.0430 & -0.9980 & -0.0470 \end{bmatrix}$$

The translation matrix is

$$T = \begin{bmatrix} 7.8653 \\ 15.3739 \\ 166.4076 \end{bmatrix}$$

Then, the Motoman's trajectory was calculated as:

$$P_{Baxter} = s \cdot R \cdot P_{human} + T \tag{6.1}$$

After the calibration process, the set of human data with 10000 groups of random motions is used to generate 25 bases which are shown in Figure 6.7. Then DMP is applied to these bases to reproduce the trajectories. For this experiment, a boxing motion behaviour was selected from the GMU database. By solving the optimisation problem with cost

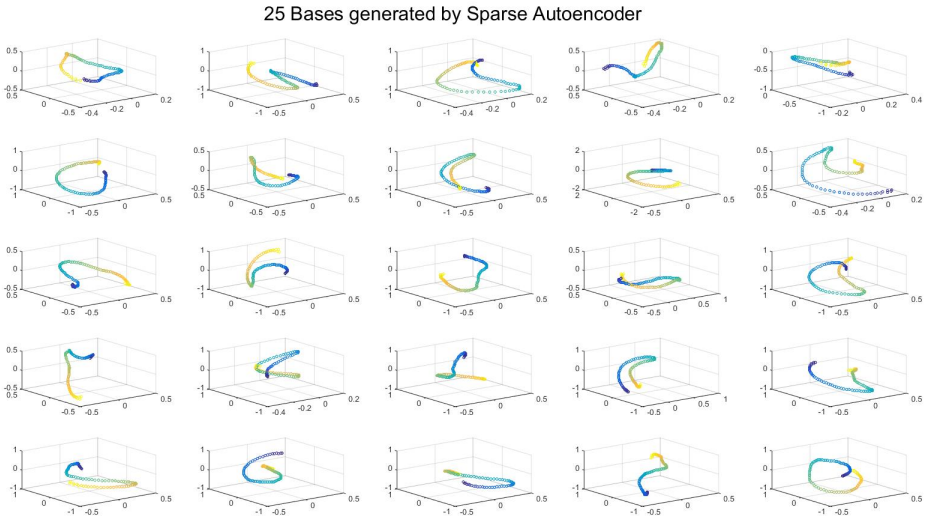


Figure 6.7: 25 bases generated by Sparse Autoencoder

function 5.4, the weighting vector w was generated. Combining w and the bases, we can reproduce the trajectories using DMP and sparse autoencoder. A reproduced trajectory and an original trajectory are shown in Figure 6.8.

Applying this reproduced trajectory with Equation 6.1, it can be transferred to the Motoman's workspace. The new trajectory after translation is shown in Figure 6.9 (a).

Similarly, we can construct a set of bases for the Motoman and transfer skills from Motoman to Baxter. Parameters of SICP from Motoman to Baxter are:

$$s = 1.4815$$

$$R = \begin{bmatrix} -0.9683 & 0.0159 & -0.2493 \\ 0.0159 & -0.9920 & -0.1252 \\ -0.2493 & -0.1252 & 0.9603 \end{bmatrix}$$

$$T = \begin{bmatrix} -187.8291 \\ -387.4445 \\ 403.7670 \end{bmatrix}$$

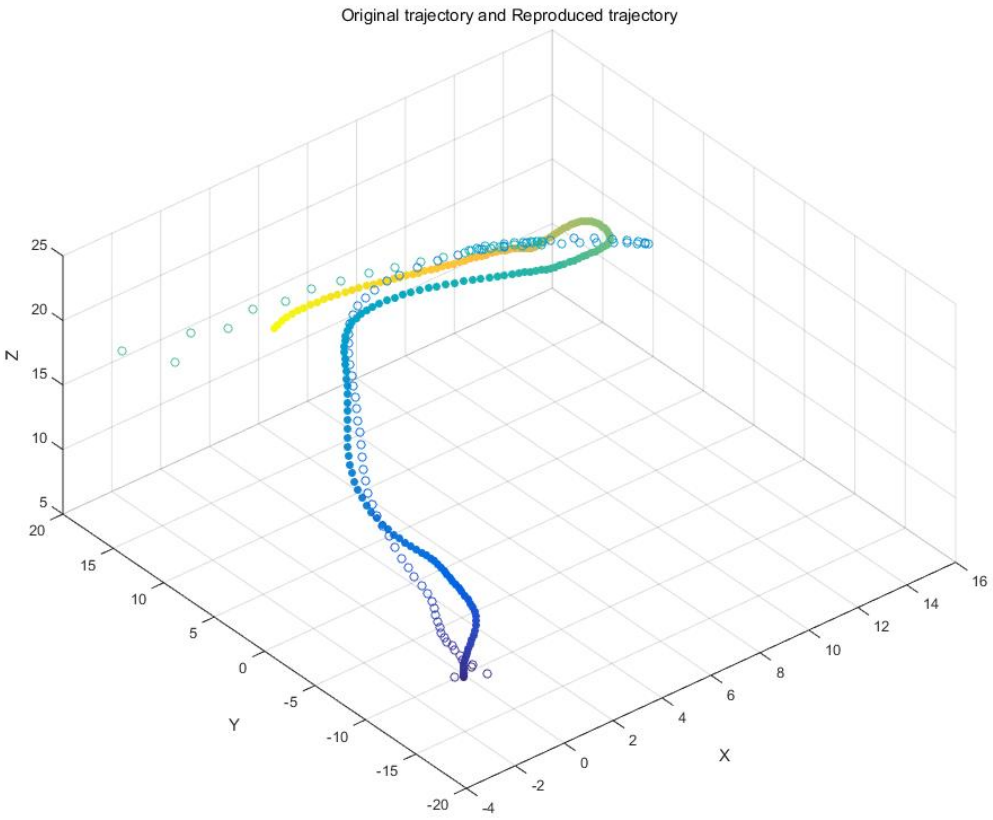


Figure 6.8: The reproduced trajectory by DMP and Sparse Autoencoder and original trajectory

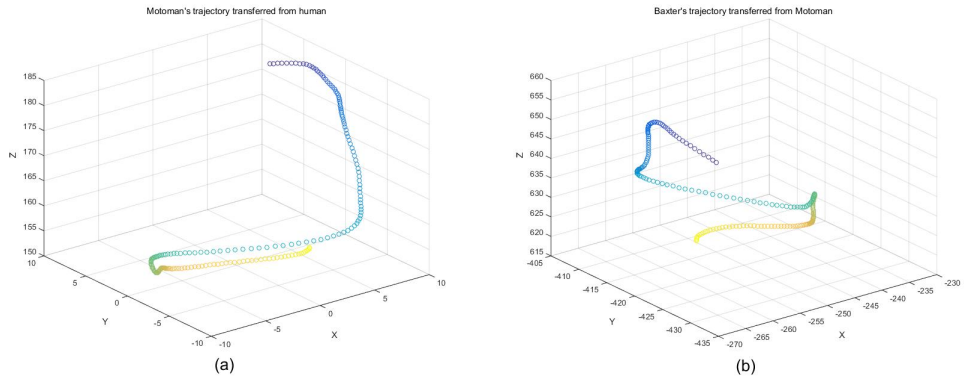


Figure 6.9: Results of trajectory transfer

The trajectory of Baxter transferred from Motoman is shown in Figure 6.9 (b).

Chapter 7

CONCLUSIONS

In this thesis, we achieved three goals. The first one was to implement robot kinematics both forward and inverse. In forward kinematics, the DH-parameters were applied to build the robot model. The end effector's coordinates could be calculated from the robot's joint angles. On the contrary, inverse kinematics was used to calculate every joint's angle from the end effector position. Both approaches were used in the work. CCD is a method based on gradient-descent method. The Jacobian method is based on Newton-Raphson method. In theory, CCD should be more robust than the Jacobian method, because of the instability in calculating the Jacobian matrix inverse. However, we found that the Jacobian method was more stable than the CCD method which was unable to reach some poses. It appears that the pseudoinverse of the Jacobian matrix increased the stability of this method. Even though the Jacobian matrix could be very close to the singular edge, the pseudoinverse could be calculated correctly. On the other hand, CCD could fail in some positions, which was a process locked in the local minimum. This problem was also the most serious problem in optimisation. To improve the performance, moving out of the local minimum would be necessary. A simple idea is to add a small angle to the specific joint to move it out of the bad position.

The second result, was using DMP to transfer a motion trajectory from an original goal to an arbitrary goal while keeping the two trajectories topological similar. Moreover if the data recorded from sensors or camera had jitter. DMP smoothed out the trajectory.

The third goal was to transfer skills from a human or a robot to arbitrary robots. First we constructed a behaviour database from example human or robot motion and decomposed this library to generate a behavior basis, then used DMP to generate new trajectories from the basis set. A different robot could reproduce an equivalent trajectory through linear

superposition of basis functions refined by the ICP process.

We used ASF and AMC file systems to describe human skeleton structures and motion behaviours. There was a significant difference between DH-parameters with the ASF/AMC approach. With DH-parameters, every joint just had one DOF; however, in ASF/AMC system, joints could have up to three DOFs. That was why we could not use DH-parameters to build human arm's model. But, we could use this method to construct models for robotic arms. So, if we use the same approach to build model between human and robots. A new area of research may be opened.

BIBLIOGRAPHY

[1] Asf-amc description. http://research.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html. [Online; accessed 13-Mar.-2017].

[2] Ros tutorial. http://wiki.ros.org/ROS/Tutorials. [Online; accessed 13-Mar.-2017].

[3] Jorge Angeles. On the numerical solution of the inverse kinematic problem. *The International Journal of Robotics Research*, 4(2):21–37, 1985.

[4] JA Apkarian and HW Smith. A new approach to kinematic control of robot manipulators. *Journal of dynamic systems, measurement, and control*, 109:97, 1987.

[5] Samuel R Buss. Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Journal of Robotics and Automation*, 17(1-19):16, 2004.

[6] John J Craig. *Introduction to robotics: mechanics and control*, volume 3. Pearson Prentice Hall Upper Saddle River, 2005.

[7] J Denavit and RS Hartenberg. An iterative method for tie displacement analysis of spatial mechanisms. 1964.

[8] Masahiro Fujita. Digital creatures for future entertainment robotics. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 801–806. IEEE, 2000.

[9] AA Goldenberg and DL Lawrence. A generalized solution to the inverse kinematics of robotic manipulators. *Journal of dynamic systems, measurement, and control*, 107(1):103–106, 1985.

[10] Georg Graetz, Guy Michaels, et al. Robots at work: the impact on productivity and jobs. Technical report, Centre for Economic Performance, LSE, 2015.

[11] Allen Strickland Hall, Ronald Robert Root, and E Sandgren. A dependable method for solving matrix loop equations for the general three-dimensional mechanism. *Journal of Engineering for industry*, 99(3):547–550, 1977.

[12] Richard S Hartenberg and Jacques Denavit. A kinematic notation for lower pair mechanisms based on matrices. *Journal of applied mechanics*, 77(2):215–221, 1955.

[13] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[14] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1398–1403. IEEE, 2002.

[15] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. *Advances in neural information processing systems*, pages 1547–1554, 2003.

[16] Abhijit Jana. *Kinect for windows SDK programming guide*. Packt Publishing Ltd, 2012.

[17] Zhangfeng Ju, Chenguang Yang, and Hongbin Ma. Kinematics modeling and experimental verification of baxter robot. In *Control Conference (CCC), 2014 33rd Chinese*, pages 8518–8523. IEEE, 2014.

[18] Wisama Khalil and Etienne Dombre. *Modeling, identification and control of robots*. Butterworth-Heinemann, 2004.

[19] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 3232–3237. IEEE, 2010.

[20] Mengtang Li. *Skill Transfer between Industrial Robots by Learning from Demonstration*. PhD thesis, Vanderbilt University, 2016.

[21] Patrick Lin, Keith Abney, and George A Bekey. *Robot ethics: the ethical and social implications of robotics*. MIT press, 2011.

[22] Mark A Livingston, Jay Sebastian, Zhuming Ai, and Jonathan W Decker. Performance measurements for the microsoft kinect skeleton. In *Virtual Reality Short Papers and Posters (VRW), 2012 IEEE*, pages 119–120. IEEE, 2012.

[23] Dalia Marin. Globalisation and the rise of the robots. *VoxEU. org*, 15, 2014.

[24] M. Müller, T. Röder, M. Clausen, B. Eberhardt, B. Krüger, and A. Weber. Documentation mocap database hdm05. Technical Report CG-2007-2, Universität Bonn, June 2007.

[25] Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[26] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 763–768. IEEE, 2009.

[27] Christian Schlegel, Andreas Steck, Davide Brugali, and Alois Knoll. Design abstraction and processes in robotics: From code-driven to model-driven engineering. In *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*, pages 324–335. Springer, 2010.

[28] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.

[29] Huan Tan and Kazuhiko Kawamura. Generation of acceptable actions using imitation learning, intention recognition, and cognitive control. In *Robot and Human Interactive Communication (RO-MAN), 2015 24th IEEE International Symposium on*, pages 389–393. IEEE, 2015.

[30] Deepak Tolani, Ambarish Goswami, and Norman I Badler. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical models*, 62(5):353–388, 2000.

[31] Yusheng T Tsai and David E Orin. A strictly convergent real-time solution for inverse kinematics of robot manipulators. *Journal of Field Robotics*, 4(4):477–501, 1987.

[32] Michael Tucker and N Duke Perreira. Generalized inverses for robotic manipulators. *Mechanism and machine theory*, 22(6):507–514, 1987.

[33] L-CT Wang and Chih-Cheng Chen. A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 7(4):489–499, 1991.

[34] Jarrett Webb and James Ashley. *Beginning Kinect Programming with the Microsoft Kinect SDK*. Apress, 2012.

[35] Wikipedia. Denavithartenberg parameters — wikipedia, the free encyclopedia, 2016. [Online; accessed 13-March-2017].

[36] Wikipedia. Motoman — wikipedia, the free encyclopedia, 2016. [Online; accessed 14-March-2017].

[37] Wikipedia. Autoencoder — wikipedia, the free encyclopedia, 2017. [Online; accessed 14-March-2017].

[38] Wikipedia. Kinect — wikipedia, the free encyclopedia, 2017. [Online; accessed 14-March-2017].

[39] Wikipedia. Robot operating system — wikipedia, the free encyclopedia, 2017. [Online; accessed 14-March-2017].

[40] JH Zhu, NN Zheng, ZJ Yuan, SY Du, and L Ma. Robust scaling iterative closest point algorithm with bidirectional distance measurement. *Electronics letters*, 46(24):1604–1605, 2010.

[41] Hui Zou. The adaptive lasso and its oracle properties. *Journal of the American statistical association*, 101(476):1418–1429, 2006.