Skill Transfer between Industrial Robots by Learning from Demonstration

By

Mengtang Li

Thesis
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, 2016

Nashville, Tennessee

Approved:
Alan Peters, Ph.D.
Mitchell Wilkes, Ph.D.

To my advisor, my friends and my family without whom I am nothing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Industry in America depends on robots[1, 2]. Industrial robots are programmable repetitive motion machines that can move and/or assemble material objects. Currently they are programmed manually by a human operator who, using a device called a ”teach pendant”, guides the robot through the motions necessary to perform a specific task. The motions are recorded and stored as a program that is run each time the task performed. For a different robot to preform the same task, it likewise must be programmed manually. This can take many hours of work. Software that takes the program from one robot and automatically transforms it into a program for another robot would save much time and could enable reuse of diverse robots that would otherwise have to be replaced.

The current state-of-art in industrial robots is similar to that of computing in the 1970s and pre-internet 80s[3]. Then, a number of companies sold their own mainframe computers that were programmed with proprietary languages and that could not communicate with computers from other manufacturers[4]. The advent of the PC and the internet changed that. Over time the hardware has become inexpensive and computers from many different manufacturers can execute the same programs. Industrial robots today have similar limitations - their control systems are proprietary and robots from different manufacturers are unable to run the same programs. Moreover, robots that do not have identical form and dynamics cannot use the same program for the same task. The ability to transfer such programs, abstracted as skills, would expand the utility of industrial robots[5, 6].

1.1 RELATED WORK

The ability to transfer task programs between heterogeneous robots would provide significant cost savings to industry, through reduced programming time and concurrent robot down-time which necessitates halting production, through increasing the usability of older robots, and by enabling cooperative assembly between robots from different manufacturers. However, a industrial task is complex, consisting of many low-level basic motions. Maja Mataric[7, 8] defined basis behaviors which are stable and prototypical interactions that robots or any other creatures use to interact with environment. Basis behaviors are fundamental building blocks to generate high-level behaviors to finish complex tasks such as industrial tasks. Once we are able to transfer those fundamental skills or knowledge, we can transfer complex high-level skills by combining those basic ones.

Researchers have been working on knowledge transfer across heterogeneous robots for over two decades and they have tried to find out the most accurate and most efficient way to do so with respect to certain requirements and constraints. One important part of knowledge transfer is the representation of the robotic knowledge.

One approach is to represent robotic tasks and knowledge symbolically. Abbas and MacDonald adopted topological task graphs[9]. Konidaris et al. used skill trees[10]. This method provides control from a higher level other than motion-based representations, while it assumes the whole system is equipped with enough knowledge about the relationship between physical behaviors and symbols.

A second approach proposed by R. Grupen [11, 12] relies on a control basis defined by potential functions, feedback signals and motor parameters. New sensorimotor skills are learned in this control basis with a reward function R after enough trainings.

Another philosophy is to model knowledge about motion and task. Once those knowledge models have been built, robots are able to not only to learn knowledge from demonstrations but also to apply what they have learned in new environments and to solve new problems. Kruger et al.[13] included object-action complexes to encode task knowledge as a function of motion, and Lyons et al. [14] used a port automata model to accomplish the same thing. Ijspeert et al.[15] adopted a dynamical system representation. A differential equation that encodes an attractor landscape is proposed to generate new trajectory towards goal state by nonlinearly transforming this canonical attractor.

1.2 Problem and Solution

The goal was to design a prototype control architecture for industrial robots which allows multiple heterogeneous robots of different morphologies from different manufacturers to perform the same task without manually programming each robot separately. The idea is to systematically decompose a typical industrial task into a small collection of behaviors that are common to many such tasks. Those basis behaviors may include:

1. Unloaded Manipulator Motions: Reach Toward, Align to Object, Avoid Obstacle, etc.

2. Object-Constrained Motions: Push, Pull, Place, etc.

3. Perceptual Behaviors: Find Colored Object, Estimate Distance, Visual Servoing, etc.

4. Maintenance Behaviors: Return to Home Position, Calibrate Vision, etc.

Although each basis behavior will have a specific program implementation for each robot, the inputs must be identical. For example, the ”align” behavior could have the inputs: object identifier, object grasp or manipulation point, 3D position in the workspace, object pose. The implementation of the behavior on each robot will differ in code and in motion parameters. The common inputs enable machine independent skill description. The outputs of the behaviors are likewise to be identical. This goes beyond data transfer to the physical result of a behavior. The results of the two robots’ actions should match at the end of each behavior. This is not absolutely necessary since the task outcome is paramount. However, we are taking this as a design decision: basis behaviors are such that both their inputs and their outcomes are identical across robots.

the systematic approach is straightforward:

1. First, we used visual servo control. During last few years, cameras that provide depth information have become available. Kinect is a good example for it and we use it for 3D reconstruction[16–18] since it is inexpensive and is supported by many open source libraries. OpenCV library[19] and Point Cloud Library (PCL)[20] allow us to write a C++ program to reconstruct the 3D world. To guide the robot's end-effector to a target object, we used color feature detection. Once this is done, the program is able to calculate the 3D coordinates of the target object with respect to the Kinect's reference frame.

2. The next procedure is to calibrate the industrial robotic arm, a Yaskawa Motoman HP3JC, to compute the transformation relationship between the Kinect frame and the robot frame. Kinect frame is the reference frame where we can acquire object's 3D information. However, it is more straightforward and easier to control the robot in its own frame. The teach pendant (shown in Figure 1.1) is a handheld device for programming motions of a robot. We used it to program a calibration job guides the robot arm to a sequence of positions. Off-the-shelf software like MotoSDK, provided by Yaskawa Motoman, enabled us to accurately acquire the joint angles of the robotic manipulator[21, 22]. This makes it easier for us to get the end-effector's 3D coordinates with respect to robot frame since once we have joint angles we only need to do the forward kinematics. In order to acquire the end-effector's 3D coordinate in the Kinect frame, we stick a small red marker on the top of the end-effector. Then the C++ program we mentioned earlier returns the corresponding 3D coordinates. Once we obtain 2 groups of coordinates, the problem becomes a well-known rigid body movement problem. There exist plenty of algorithms to help us solve the problem[23, 24]. We used the Singular Value Decomposition method[25].

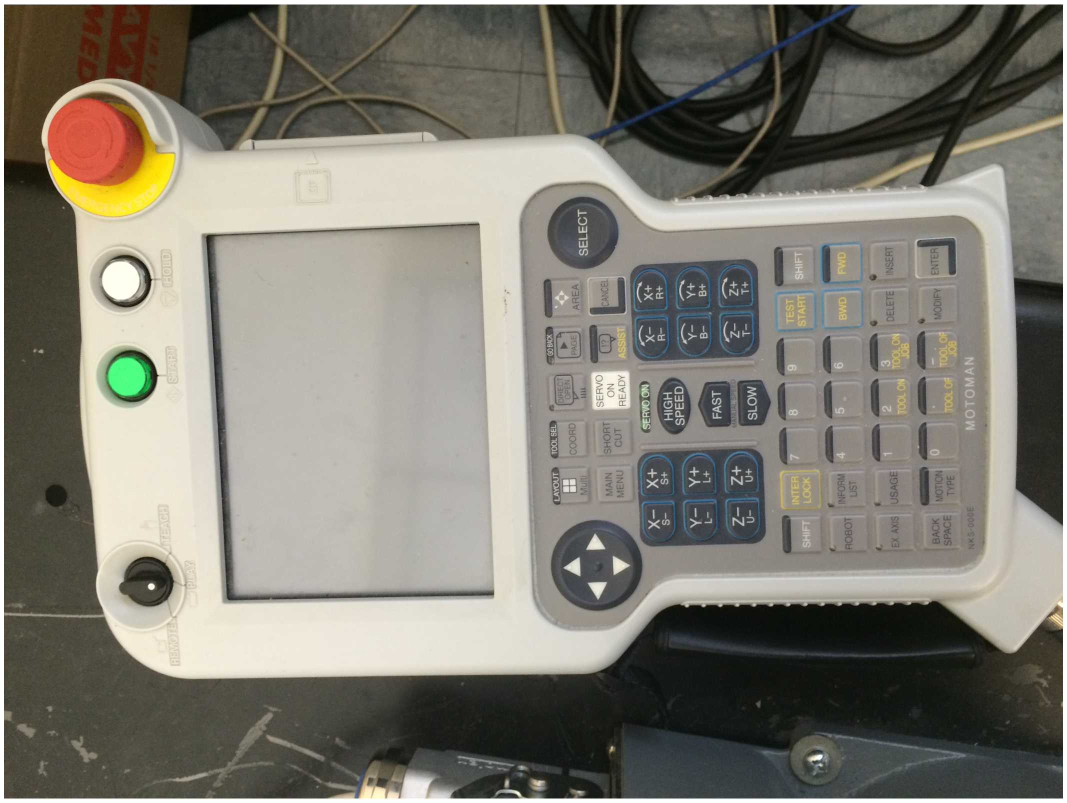


Figure 1.1: Teach Pendant of Yaskawa Motoman HP3JC

3. Finally, we uesd Ijspeert's method to transfer the robotic knowledge by learning attractor landscapes[26, 27]. Dynamic movement primitives (DMPs) are a mathematical method to represent motion primitives which can be combined to generate complex movements. We used DMPs to model attractor behaviors of the nonlinear robotic systems. A non-linear differential equation is constructed to represent a single movement which is performed by one of the robotic manipulators and can be learned by another one. Three articulated industrial manipulators (Yaskawa Motoman HP3JC, Universal Robot UR5 and Rethink Robotics Baxter), were chosen by us as experimental platforms. HP3JC and UR5 both have 6 joints while Baxter has 7 joints. Two assembly tasks were selected for analysis and implementation.

1.3 Software and Hardware

The work was done mainly on a PC using the operating system Ubuntu 14.04 LTS Trusty Tahr[28]. We used the Kinect 1[29] to obtain 3D data. The visual servo system program was written in C++ using the libraries OpenCV 3.1[30] and PCL 1.7.2[31]. PCL allows us to process point cloud data and OpenCV allows us to process image data. Matlab 2015a academic license[32] helps us solve the rigid body transformation problem and do all the computation part of the skill transfer. ROS Indigo[33] was used to communicate between robots and computers. It provides interprocess communications via a publisher/subscriber model. The 3D information message listener and robot driver programs are written for ROS in Python 2.7[34]. Yaskawa Motoman HP3JC[35], Universal Robot UR5[36] and Rethink Robotics Baxter[37] were the three experimental platforms. Since we do not have a real UR5 or a real Baxter, we used Gazebo 7.0.0[38] to simulate them within ROS.

Chapter 2

SYSTEM DESCRIPTION

Since we used three different industrial manipulators as the experimental platforms with which to devise our robotic skill transfer framework, we needed a middleware framework that would enable us to send control information to each of the robots. The Robot Operating System (ROS) [39] provides this. ROS is an efficient collection of tools and libraries that makes the process of designing and controlling different robots easier.

The visual servo system was another important module of the work. We used it to guide the HP3JC. There are two main approaches to visual servoing: Position Based Visual Servo (PBVS)[40–42] and Image Based Visual Servo (IBVS)[43–48]. Generally speaking, PBVS uses visual data to reconstruct the 3D world and allows researchers to design control algorithms in Cartesian space. IBVS requires the design of Jacobians for joint control based on image features acquired directly from a vision systems. we chose the PBVS approach since it is more straightforward to implement. Moreover it is more intuitive to control a robot in Cartesian space than by image features in a vision system.

In this chapter, we introduce the system and describe specs of the robots and visual systems.

2.1 YASKAWA MOTOMAN HP3JC

Since we have a Yaskawa Motoman HP3JC in our lab, we used it as the primary platform on which to learn a set of motions. The HP3JC is a small yet compact, speedy and flexible industrial manipulator that can be mounted on the floor, ceiling or even wall. It only needs a little working space. Although it was designed as a welding robot and for the handling of lightweight materials, its characteristics make it ideal for lab research and

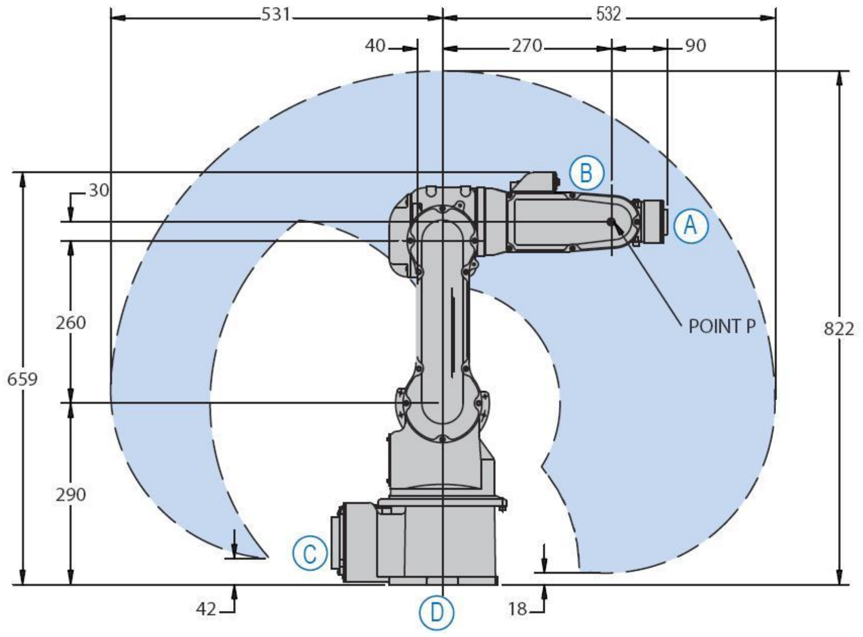


Figure 2.1: Yaskawa Motoman HP3JC
(http://www.used-robots.com/images/robots/original/hp3jc-side.jpg)

education. In our set-up the HP3JC is mounted on box which places the robot's base 55 cm above the ground. Figure 2.1 shows the metric specifications of the Yaskawa Motoman HP3JC.

2.2 Universal Robot UR5

A ROS Gazebo simulation of a Universal Robot UR5 was chosen as another experimental platform that we used. Like the HP3JC, the UR5 is a lightweight, flexible and collaborative robot. It has a wide working space with radius up to 850mm which is larger than the PH3JC. We used the simulated UR5 in Gazebo as a the ”student robot” to learn skills from the ”teacher robot” HP3JC. Figure 2.2 shows the metric specifications of the Universal Robot UR5.

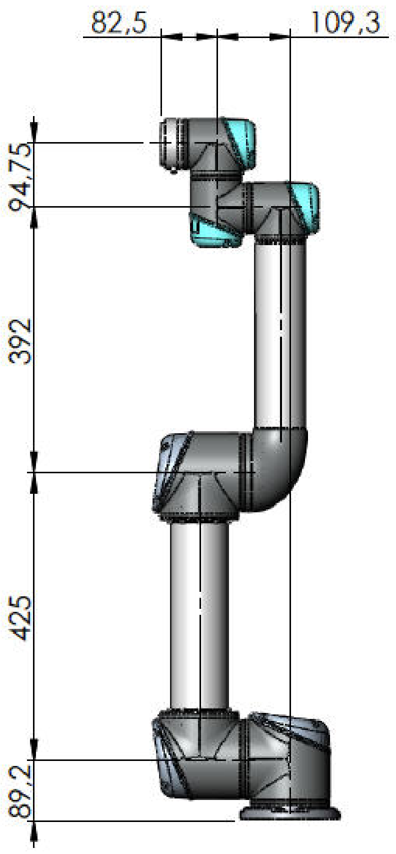


Figure 2.2: Universal Robot UR5 (http://www.zacobria.com/universal-robots-zacobria-forum-hints-tips-how-to/script-client-server-example/)

2.3 Rethink Robotics Baxter

A ROS Gazebo simulation Rethink Robotics Baxter was the third experimental platform. Baxter has been integrated by many companies in their factories across North America. Baxter enables an important test of our results because each of its two arms has 7 joints while the former two robots only have 6 joints. The DH parameters are the same for each arm. For our experiment, we used only one arm. Figure 2.3 shows the metric specifications of the Rethink Robotics Baxter.

2.4 Forward Kinematics

Forward kinematics uses the kinematic equations of a robot to calculate the position and orientation of the robot's end-effector given specified joint angles. In other words, forward kinematics translates the joint space to Cartesian space. To calculate the forward kinematics and inverse kinematics, we used the famous Denavit Hartenberg parameters (also known as DH parameters). The DH convention represents each individual homogeneous transformation of each corresponding joint as the product of four basic transformations[49].

$$\begin{aligned} T_i &= \mathrm{Rot}_{z,\theta_i}\mathrm{Trans}_{z,d_i}\mathrm{Trans}_{x,a_i}\mathrm{Rot}_{x,\alpha_i} \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.1}$$

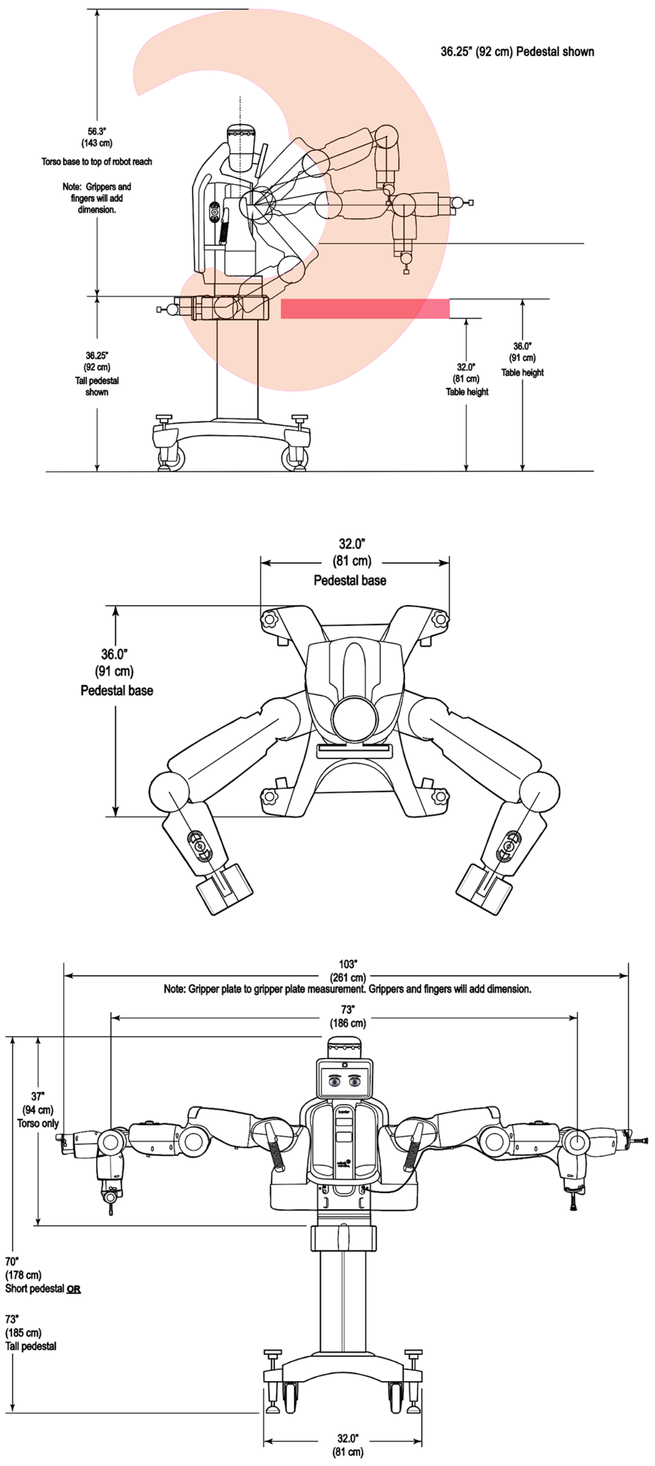

36.25" (92 cm) Pedestal shown
56.3" (143 cm) Torso base to top of robot reach
Note: Grippers and fingers will add dimension.
36.25" (92 cm) Tall pedestal shown
32.0" (81 cm) Table height
36.0" (91 cm) Table height
32.0" (81 cm) Pedestal base
36.0" (91 cm) Pedestal base
103" (261 cm)
Note: Gripper plate to gripper plate measurement. Grippers and fingers will add dimension.
73" (186 cm)
37" (94 cm) Torso only
70" (178 cm) Short pedestal OR
73" (185 cm) Tall pedestal
32.0" (81 cm)

Figure 2.3: Rethink Robotics Baxter (http://sdk.rethinkrobotics.com/wiki/Workspace_Guidelines)

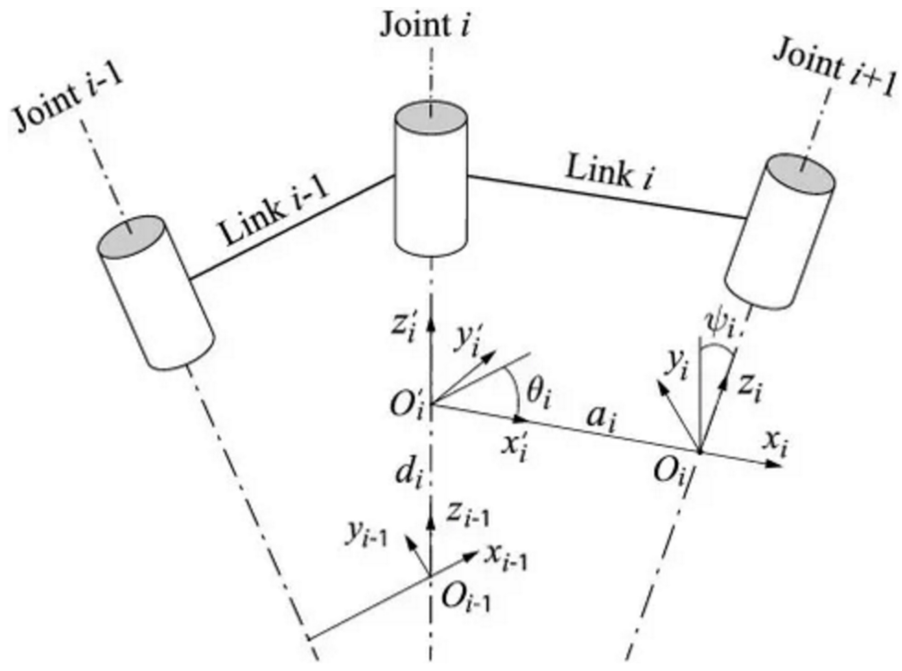


Figure 2.4: Denavit Hartenberg Parameters
(https://www.quora.com/Robotics/What-is-the-best-resthece-to-understand-Denavit%E2%80%93Hartenberg-parameters)

The fthe key parameters of DH parameters are (see Figure 2.4):

- a_i: link length: distance along x_i from the intersection of the x_i and z_{i-1} axes to o_i
- α_i: link twist: the angle from z_{i-1} to z_i measured about x_i
- d_i: link offset: distance along z_{i-1} from o_{i-1} to the intersection of the x_i and z_{i-1} axes
- θ_i: joint angle: the angle from x_{i-1} and x_i measured about z_{i-1}

There is no need to do the matrix multiplication every time. Use the final resulting matrix of Eq 2.1 for every link. Table 2.1 shows the DH parameters of the Yaskawa Motoman HP3JC. Table 2.2 shows that of Universal Robot UR5 and table 2.3 shows that of Rethink Robotics Baxter. Note that since we have mounted a JR3 force and torque sensor (45 mm height) on the top of the end-effector, the link length of link 6 of Yaskawa Motoman HP3JC is modified. Plugging the corresponding DH values to Eq 2.1 for every link can we get 6 (7 for Baxter) homogeneous transformation matrices $T_1 \sim T_6$ with respect to every joint angle θ_i. Then we multiply those 6 matrices.

Link	θ_i	d_i	a_i	α_i
1	$\theta_1 + 0°$	1.57	0	$-90°$
2	$\theta_2 - 90°$	0	2.60	$90°$
3	$\theta_3 + 0°$	0	0.30	$-90°$
4	$\theta_4 + 0°$	-2.70	0	$90°$
5	$\theta_5 + 0°$	0	0	$-90°$
6	$\theta_6 + 0°$	-1.35	0	$0°$

Table 2.1: DH parameters for Yaskawa Motoman HP3JC

Link	θ_i	d_i	a_i	α_i
1	$\theta_1 + 0°$	0.892	0	$90°$
2	$\theta_2 + 0°$	0	-4.250	$0°$
3	$\theta_3 + 0°$	0	-3.923	$0°$
4	$\theta_4 + 0°$	1.092	0	$90°$
5	$\theta_5 + 0°$	0.947	0	$-90°$
6	$\theta_6 + 0°$	0.823	0	$0°$

Table 2.2: DH parameters for Universal Robot UR5

Link	θ_i	d_i	a_i	α_i
1	$\theta_1 + 0°$	2.7035	0.690	$-90°$
2	$\theta_2 + 0°$	0	0	$90°$
3	$\theta_3 + 0°$	3.6435	0.690	$-90°$
4	$\theta_4 + 0°$	0	0	$90°$
5	$\theta_5 + 0°$	3.7429	0.100	$-90°$
6	$\theta_6 + 0°$	0	0	$90°$
7	$\theta_7 + 0°$	2.800	0	$0°$

Table 2.3: DH parameters for Universal Robot Rethink Robotics Baxter

$$H_b^a = T_1(\theta_1)T_2(\theta_2)T_3(\theta_3)T_4(\theta_4)T_5(\theta_5)T_6(\theta_6) = \begin{bmatrix} \text{Rotation}_{3x3} & \text{Translation}_{3x1} \\ 0 & 1 \end{bmatrix} \tag{2.2}$$

The resulting matrix H_b^a is a 4 by 4 matrix. It represents the relationship between origin base and end-effector. The sub matrix Rotation is a 3 by 3 matrix contains orientation information while vector Translation is a 3 by 1 column vector whose elements are the end-effector's 3D coordinates x, y, z. So far, we have found a way to obtain the end-effector's coordinates and orientation given the values of joint variables.

2.5 KINECT VISION SYSTEM

Kinect becomes more and more popular among researchers because it is equipped with an infrared laser depth sensor which can acquire image 3D data under any light conditions. Kinect releases the burden of calibrating a pair of stereopsis cameras upon researchers since this procedure is time consuming and sensitive to any slight physical disturbance. Using the open source PCL, we are able to obtain 3D data returned from the Kinect in the form of a point cloud[50]. The point cloud is a group of points. Each point has its own 3D coordinate. Additionally, it can also contain RGB color information. Figure 2.5 is an exemple perspective from Kinect using PCL.

Specifically, we use the following functions:

- *opencv* :: *CvScalar* to create a HSV vector
- *opencv* :: *cvCvtColor* to convert RGB image into HSV image
- *opencv* :: *cvInRanges* to filter images not in a particular range
- *opencv* :: *cvSmooth* to smooth images
- *pcl* :: *PointXYZRGBA* to store point cloud data
- *pcl* :: *OpenNIGrabber* and *pcl* :: *visualization* :: *CloudViewe*r to create a OpenNI point cloud viewer
- *ros* :: *Publisher* to publish message through ROS

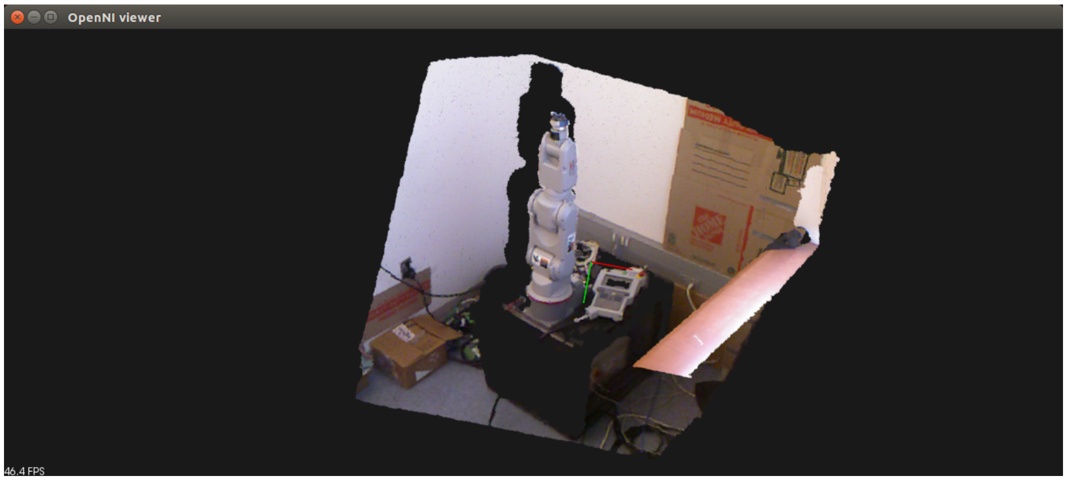


Figure 2.5: A point cloud obtained from Kinect using PCL

2.6 Working System

We placed the robot arm at a place where its work space is big enough so there won't be any accidental collision. Next, we placed the Kinect at a place where its field of view is big enough to see all possible movements of the robotic arm (minimum 1.4 meters or 4.5 feet[51]). The system includes a Yaskawa Motoman HP3JC robotic manipulator, a NXC100 controller, a Kinect visual system and a PC and is shown in Figure 2.6.

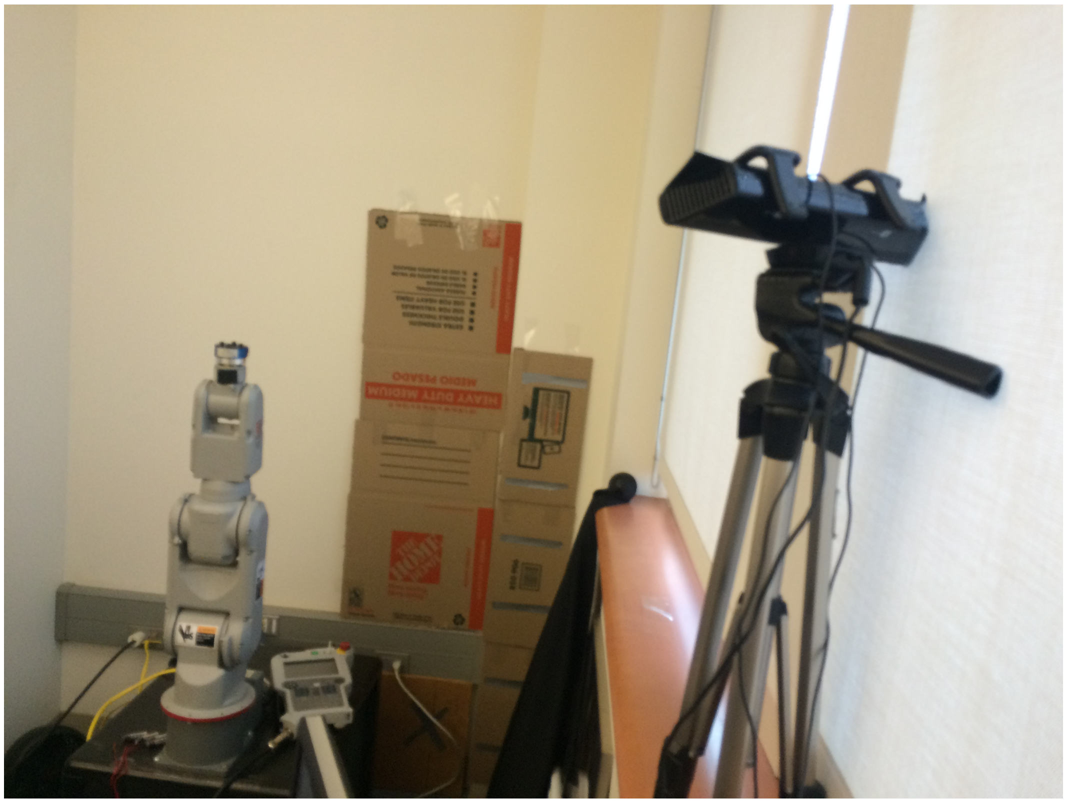


Figure 2.6: System Setup

Chapter 3

System Setup

In this chapter, we describe how to use the Kinect visual system to find a colored object then to use it to complete the transformation relationship between the Kinect frame and the robot frame. After this procedure, the system is set up and good to go for the robotic knowledge transfer.

3.1 Color Object Detection

To detect a color target like a red marker, the approach is straightforward and simple. We use it because computer vision is not our focus. The approach flowchart is shown in Figure 3.1. First of all, we convert the color space from RGB to HSV. The reason is that HSV color space separates luma from chroma which means HSV separates the intensity information from color information. Often, the RGB color space is much noisier than HSV color space. OpenCV provides us with built-in function to accomplish this task. Next, we specify the desired HSV values and use them to filter the original point cloud. Once this is done, the resulting point cloud contains only those points whose HSV values are in the range of the specified color. Then, we just need to calculate the average 3D coordinates of

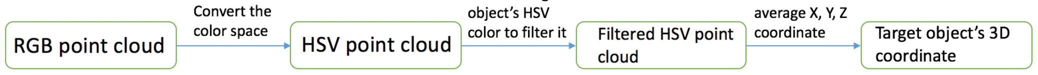

RGB point cloud
Convert the color space
HSV point cloud
Use our target object's HSV color to filter it
Filtered HSV point cloud
Calculate the average X, Y, Z coordinate
Target object's 3D coordinate

Figure 3.1: Color Detection Flowchart

all points. That yields the target object's 3D position.

$$\begin{cases} x = \frac{1}{N}\sum_{i=1}^{i=N} x_i \\ y = \frac{1}{N}\sum_{i=1}^{i=N} y_i \\ z = \frac{1}{N}\sum_{i=1}^{i=N} z_i \end{cases} \tag{3.1}$$

Additionally,since outliers will bias an average, we use a median filter over all the points in a 5 by 5 by 5, 3D cubic area. Then a ROS channel is selected to publish the 3D position information so that the robot can read it.

3.2 Robot Calibration

This section describes the computation of the transform relationship between the robotic frame and the vision frame. To acquire the target's 3D coordinates in the Kinect frame, we wrote a C++ program that uses OpenCV and PCL. To position the target in the robot's frame, we use the teach pendant to program a sequence of movements. Figure 3.2 shows this procedure. Table 3.1 shows the end-effector's 3D coordinates in Kinect frame. Since the Kinect-based vision system can estimate the position of a color object, we stick a red marker on the top of the end-effector of the robot arm. MotoSDK released by Yaskawa Motoman can acquire the joint angles of each motion step. Figure 3.3 shows the user-interface of this software. Once that is done, the next step is to use Eq 2.2 to calculate the end-effector's 3D coordinates in the robot frame.

	step1	step2	step3	step4	step5	step6	step7	step8	step9	step10
x	-219.7	-358.8	-285.3	-37.2	236.3	61.0	111.5	71.6	-13.4	-105.1
y	335.1	174.4	25.9	-60.0	30.8	267.2	337.6	409.9	432.0	414.1
z	1243.7	1189.7	1094.5	995.0	1032.5	1073.0	1116.8	1216.1	1243.7	1243.7

Table 3.1: 3D coordinates in Kinect Frame(mm)

Table 3.2 shows the joint angle variables used for robot calibration.

As previously mentioned before, once the joint angles are known, we can apply Eq 2.1

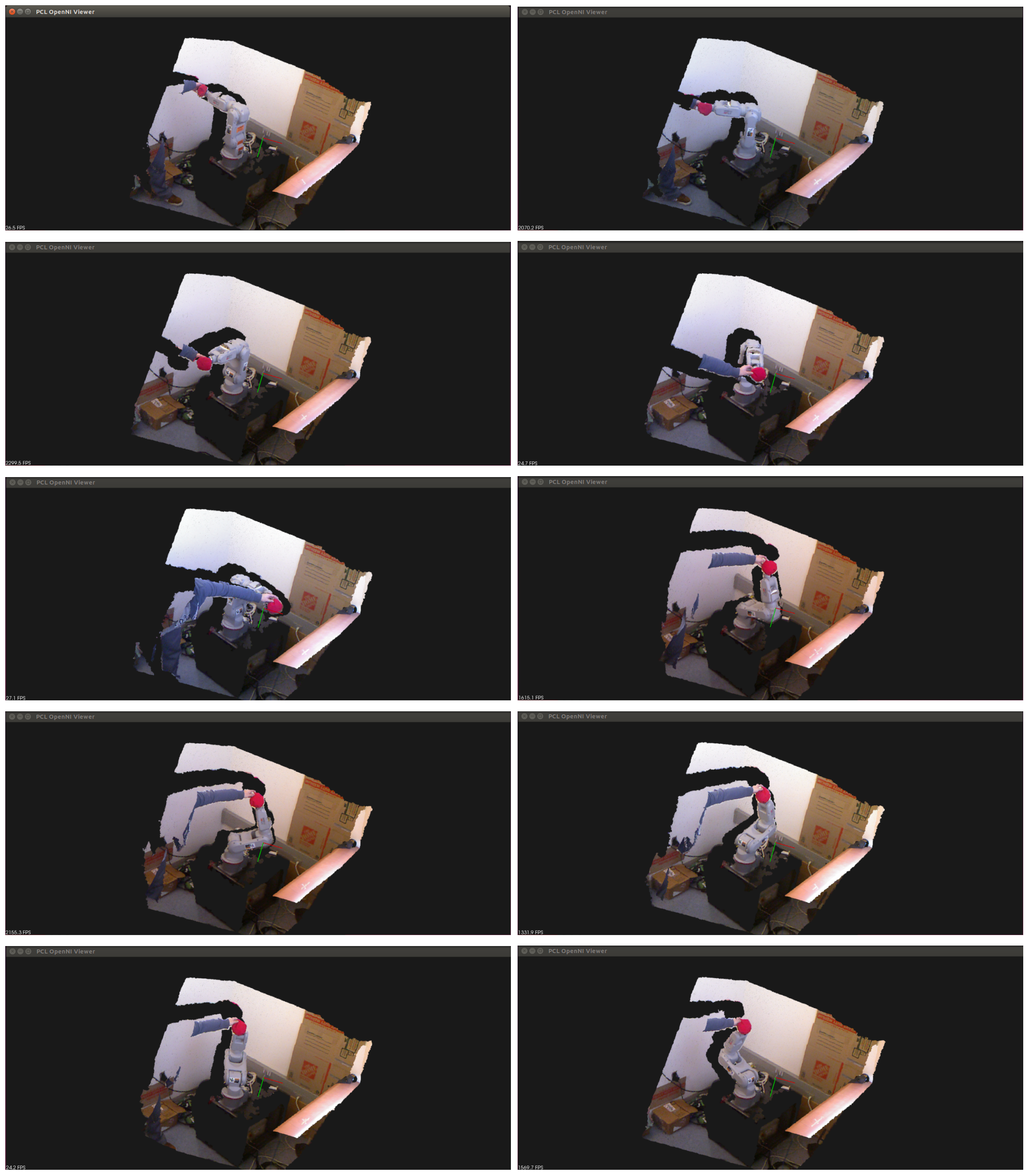
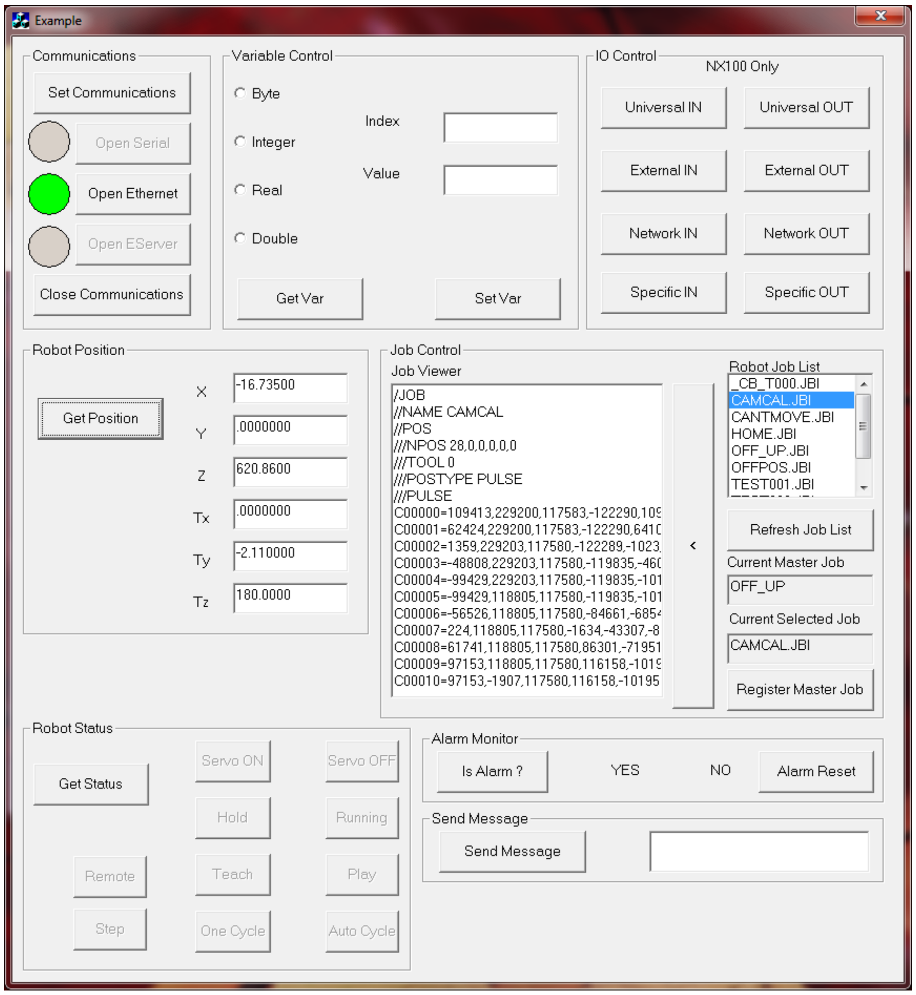


Figure 3.2: Sequence of Movements Used to Calibrate the Robot



X	-16.73500
Y	.0000000
Z	620.8600
Tx	.0000000
Ty	-2.110000
Tz	180.0000

```
/JOB
//NAME CAMCAL
//POS
///NPOS 28,0,0,0,0,0
///TOOL 0
///POSTYPE PULSE
///PULSE
C00000=109413,229200,117583,-122290,109
C00001=62424,229200,117583,-122290,6410
C00002=1359,229203,117580,-122289,-1023,
C00003=-48808,229203,117580,-119835,-460
C00004=-99429,229203,117580,-119835,-101
C00005=-99429,118805,117580,-119835,-101
C00006=-56526,118805,117580,-84661,-6854
C00007=224,118805,117580,-1634,-43307,-8
C00008=61741,118805,117580,86301,-71951
C00009=97153,118805,117580,116158,-1019
C00010=97153,-1907,117580,116158,-10195
```

Robot Job List:
- _CB_T000.JBI
- CAMCAL.JBI
- CANTMOVE.JBI
- HOME.JBI
- OFF_UP.JBI
- OFFPOS.JBI
- TEST001.JBI

Current Master Job: OFF_UP

Current Selected Job: CAMCAL.JBI

Figure 3.3: Motoman SDK Software Use Interface

θ_i	step1	step2	step3	step4	step5	step6	step7	step8	step9	step10
θ_1	-90	-45	0	45	90	-90	-45	0	45	90
θ_2	0	0	0	0	0	-70	-70	-70	-70	-70
θ_3	0	0	0	0	0	0	0	0	0	0
θ_4	0	0	0	0	0	0	0	0	0	0
θ_5	0	0	0	0	0	0	0	0	0	0
θ_6	0	0	0	0	0	0	0	0	0	0

Table 3.2: Joint Angle Variable (degree)

and Eq 2.2 to compute the target's 3D coordinates. We plug the joint angles into Eq 2.2 and multiply the 6 matrices. Table 3.3 shows the corresponding 3D coordinates in the robot frame.

	step1	step2	step3	step4	step5	step6	step7	step8	step9	step10
x	0.000	2.864	4.050	2.864	0.000	0.000	-0.947	-1.340	-0.947	0.000
y	-4.050	-2.864	0.000	2.864	4.050	1.340	0.947	0.000	-0.947	-1.340
z	4.470	4.470	4.470	4.470	4.470	6.368	6.368	6.368	6.368	6.368

Table 3.3: 3D coordinates in Robot Frame(dm)

3.3 RIGID BODY TRANSFORMATION

Now, we have two groups of coordinates correspond to the same points (say $\{x_1, x_2, ..., x_n, x \in \Re^3\}$ and $\{y_1, y_2, ..., y_n, y \in \Re^3\}$). We need to find a rotation matrix R and a translation vector T to map x_i to y_i, $n = 1, 2, ..., n$. Note that measurement errors are unavoidable. So the mapping is not exact. The problem becomes:

$$\min_{R \in \Omega, d} \sum_{i=1}^{n} \| Rx_i + d - y_i \|^2 \tag{3.2}$$

where

$$\Omega = \{R \mid R^T R = RR^T = I_3; det(R) = 1\} \tag{3.3}$$

We used a Singular Value Decomposition method to solve this rigid body transformation problem[52].

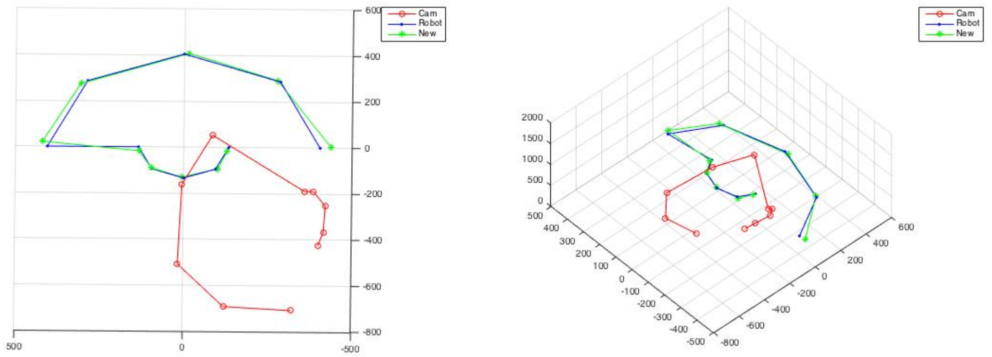


Figure 3.4: Results of Rigid Body Transformation

Algorithm 1 using SVD method to solve problem (3.2) with constraint (3.3)

1. $\bar{x} = \frac{1}{n}\sum_{i=1}^{i=n} x_i$, $\bar{y} = \frac{1}{n}\sum_{i=1}^{i=n} y_i$, $x_i \in \Re^3$, $y_i \in \Re^3$
2. $A = [x_1 - \bar{x}, ..., x_n - \bar{x}]$, $B = [y_1 - \bar{y}, ..., y_n - \bar{y}]$
3. $C = BA^T$
4. $USV^T = C$, Singular Value Decomposition of C
5. $R = Udiag(1, 1, det(UV^T))V^T$, this is the rotation matrix
6. $d = \bar{y} - R\bar{x}$, this is the translation vector

We used Matlab to solve this problem. The coordinates in Table 3.1 are x_i and the coordinates in Table 3.3 are the y_i. The results were as follows and shown in Figure 3.4.

$$R = \begin{bmatrix} -0.4448 & 0.5112 & -0.7354 \\ 0.8956 & 0.2449 & -0.3715 \\ -0.0098 & -0.8238 & -0.5667 \end{bmatrix}, d = \begin{bmatrix} 1065.9 \\ 886.8 \\ 1124 \end{bmatrix} \tag{3.4}$$

$$RobotCoord = R \times KinectCoord + T \tag{3.5}$$

With the rotation matrix R and translation vector T, we are able to send 3D position command to the Yaskawa Motoman HP3JC within its own frame. First, the C++ visual servo program calculates the target's 3D coordinates, *KinectCoord*, in its own frame and publishes them on a ROS channel. Then, the NXC100 robot controller subscribes to the same channel and uses the rotation matrix R and translation vector T to calculate the target's 3D coordinates, *RobotCoord*, in the robot frame.

Chapter 4

Robotic Knowledge Transfer

Learning from demonstration is the cornerstone of human education. Here, we use the similar method to transfer robotic knowledge across heterogeneous robotic platforms.

There exist some major challenges before we are able to fully transfer the knowledge[53].

1. Correspondence: The teacher robot and the student robot are heterogeneous and their degree of freedom (DoF) or links may not match.

2. Generalization: The student robot must have the ability to generate new knowledge from what teacher robot has taught since teacher robot cannot demonstrate all possible movements.

We used Ijspeert's approach[15, 26, 27] to transfer robotic knowledge by learning attractor landscapes. Dynamical movement primitives are a mathematical method to represent motion primitives which can be combined to generate complex movements. DMPs are useful for robotics because. A non-linear differential equation is set up to represent the joint trajectory of a single movement what was performed by one of the robotic manipulators. The differential equation and its learned attractor are then used to drive a different robot. We have chosen three articulated industrial manipulators (Yaskawa Motoman HP3JC, Universal Robot UR5 and Rethink Robotics Baxter) as experimental platforms. Because we don't have a real UR5 or Baxter, we use Gazebo to simulate them.

4.1 Locally Weighted Regression

Most machine learning methods use a single global model to fit all the training data. Such an approach considers all the data to be equivalent. But what if we know there are

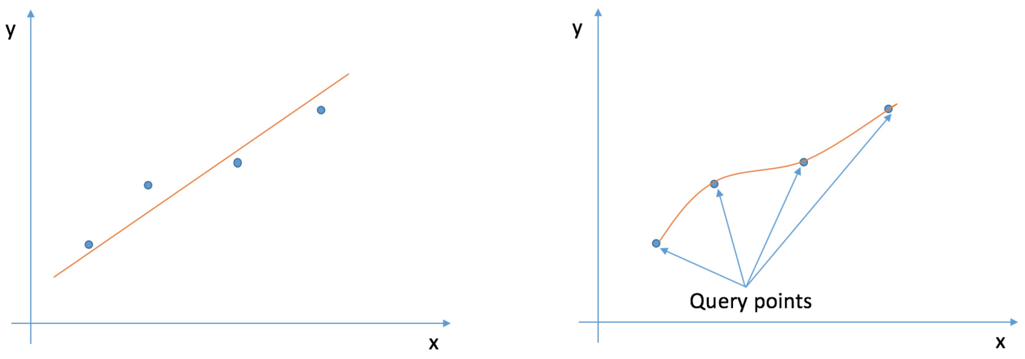


Figure 4.1: Global Model Regression and Locally Weighted Regression

query points of interest? It is more rational to emphasize data that is more similar to the query point and deemphasize data that is not similar to the query point[54]. Figure 4.1 shows a simple example of global model regression and locally weighted regression. While global model regression uses a single model to fit all training data, locally weighted model tries to fit the training data in neighborhood regions around those four query points. The local weighted model emphasizes training data which is closer to the query points.

Many methods exist to calculate the similarity between data points and query points. The simplest and most common one is the Euclidean distance between data point and query point.

$$d(x,q) = \sqrt{\| x - q \|} \tag{4.1}$$

A kernel function $\psi()$ is used to calculate the corresponding weights ω_i for a particular data query point. The simplest and most common one is the Gaussian kernel. The meaning of Gaussian kernel is straightforward and self explanatory. It emphasizes those points near the query point and deemphasizes those points far from the query point exponentially.

$$\psi(d) = e^{-d^2} \tag{4.2}$$

Locally weighted regression is used to calculate those weights ω_i for each kernel function $\psi(s)$. Its fast speed allows real time computation possible and the calculations are independent for each kernel.

4.2 Dynamic Movement Primitives

An analytically well-understood damped spring system was chosen to meet the following requirements mentioned by Ijspeert: (1) it should be a time independent autonomous system; (2) it should be able to work in any dimension; (3) it should not contain too many open parameters to learn; and (4) it should allow real-time computation.

$$\tau\dot{v} = K(g-x) - Dv + (g-x_0)f \tag{4.3}$$

$$\tau\dot{x} = v \tag{4.4}$$

where,
x is the position, x_0 is the initial position;
v is the velocity;
g is the goal position;
τ is a scaling factor;
K is the spring Hook constant;
D is the spring damping term such that they system is critically damped;
f is a non-linear function which can be learned to generate any movement trajectory.

Eq 4.3 and Eq 4.4 are the Transformation System Equations. It is clear and straightforward that if the force term $f = 0$, a unique point attractor $(x, v) = (g, 0)$ is obtained. The force term is defined as follows to generate various point attractors:

$$f(s) = \frac{\sum_{i=1}^{N} \omega_i \psi_i(s)}{\sum_{i=1}^{N} \psi_i(s)} s(g-x_0) \tag{4.5}$$

$$\psi_i(s) = e^{-(s-c_i)^2} \tag{4.6}$$

where,

s is the phase variable, which monotonically decays from 1 towards 0;

$\psi_i(s)$ are kernel functions. c_i are query points;

N is the number of the query points;

ω_i are adjustable weights and are learned from demonstration and are able to generate new desired movement trajectories.

The modulation term s makes the force f effectively vanishes when the goal g is reached. Another modulation term $(g - x_0)$ is used to scale the model due to a change of the movement.

To meet the requirements mentioned earlier before to build a time-independent autonomous system, we introduce a Canonical System Equation:

$$\tau \dot{s} = -\alpha s \tag{4.7}$$

where,

α is constant controlling the canonical system.

The phase variable s gets rid of time dependency and it monotonically decays to 0. $s = 1$ means the start of the time evolution and $s = 0$ means the goal has been reached.

The framework aiming at transferring robotic knowledge by learning from demonstration is as follows:

Algorithm 2 using DMP method to transfer robotic knowledge

1. The teacher shows the student a single demonstration movement and the trajectory is recorded. Especially, $x(t), y(t)$ and $z(t)$ are recorded separately. Their corresponding first derivatives and second derivatives are computed numerically for each time step since we haven't found a way to record velocity or acceleration in real time.

2. Integrate the canonical system equation

$$\tau \dot{s} = -\alpha s \tag{4.8}$$

to obtain the phase variable s to represent the time evolution. s monotonically decreases from 1 to 0.

3. Compute

$$f_{Demo} = \frac{-K(g-x) + D\dot{x} + \tau \ddot{x}}{g - x_0} \tag{4.9}$$

for each time step, where x are the recorded trajectory positions of teacher and x_0 is the initial position, $\dot{x}, \ddot{x}$ are the corresponding velocities and accelerations.

4. Calculate weights ω_i. Locally Weighted Regression (LWR) is used to find weights ω_i for each kernel function $\psi(s)_i$ in f. It minimizes the cost function:

$$J = \sum_{s=1}^{P} \psi_i(s)[f_{Learnt}(s) - \omega_i s(g - x_0)]^2 \tag{4.10}$$

The solution to this weighted linear regression problem is give by Ijspeert[27]:

$$\omega_i = \frac{E^T \Gamma_i f_{demo}}{E^T \Gamma_i E} \tag{4.11}$$

where,

$$E = \begin{bmatrix} 1(g-x_0) \\ \dots \\ s(g-x_0) \\ \dots \\ 0(g-x_0) \end{bmatrix}, \Gamma_i = \begin{bmatrix} \psi_i(1) & & & \\ & \psi_i(2) & & \\ & & \dots & \\ & & & \psi_i(P) \end{bmatrix}, f_{target} = \begin{bmatrix} f_{demo}(1) \\ f_{demo}(2) \\ \dots \\ f_{demo}(P) \end{bmatrix} \tag{4.12}$$

5. Once those weights ω_i are obtained, we can specify the desired initial position x_0 and

goal position x_g and compute

$$f_{Learnt} = \frac{\sum_{i=1}^{N} \omega_i \psi_i(s)}{\sum_{i=1}^{N} \psi_i(s)} s(g - x_0) \tag{4.13}$$

6. After we obtain f_{Learnt}, we can have

$$\ddot{x} = \frac{K(g-x) - D\dot{x} + f_{Learnt}(g - x_0)}{\tau} \tag{4.14}$$

Integrate it once and twice can we have $\dot{x}$ and x for the student robot.

This approach has several advantages. First, because $f(s)$ eventually decays to 0 at the end of a movement, ultimately reaching at the attractor is guranteed. Second, each dimension is independent with each other. We can compute x, y, z separately and then combine the three 1D trajectories into a single 3D trajectory. Third, this approach allows the student robots to not only to learn knowledge from demonstrations but also to apply what they have originally learned into new environments and to plan new trajectories. With the help of Peter Corke's matlab robot tool box[55], we can simulate robots under Matlab. Figure 4.3 shows the ability to generate trajectories to new goals after learning one single demonstration trajectory. We control the simulated HP3JC to move at $(x, y, z) = (5.887, -2.143, 1.566)$ and use this trajectory as a demonstration. Then we apply the algorithm we described above to generate new trajectories with 25 different goal positions.

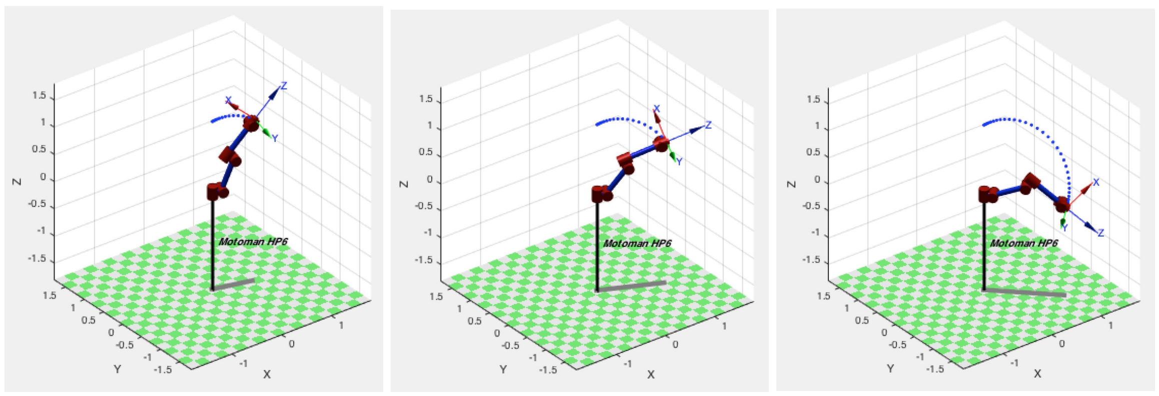


Figure 4.2: HP3JC Matlab Simulation

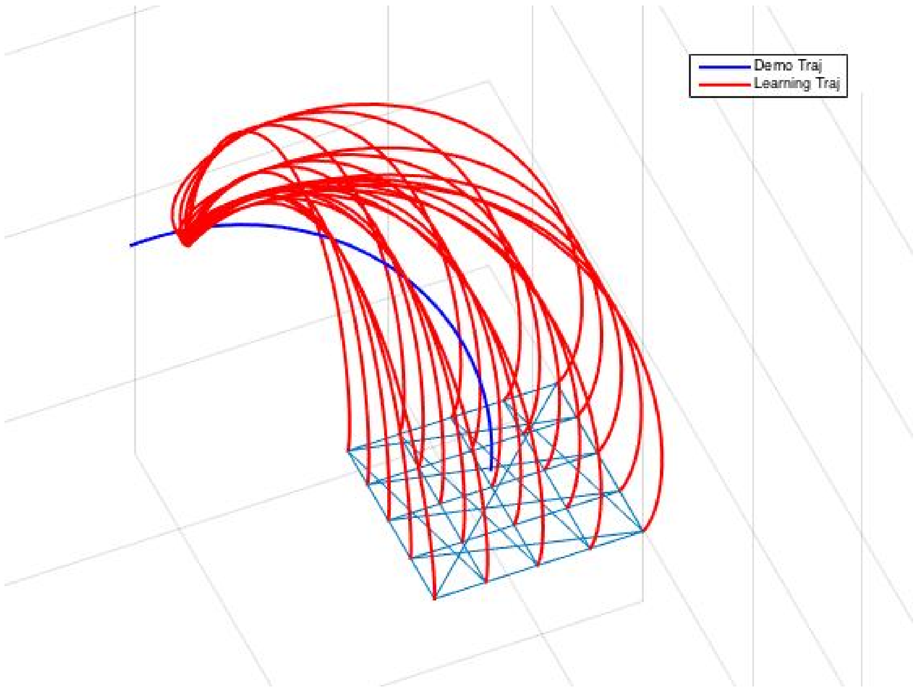


Figure 4.3: Learn from Demonstration (Blue) and Apply to New Goals (Red)

Chapter 5

EXPERIMENTS AND ANALYSIS

Once the experimental platform is setup to work, we test and analyze the system. Since this experimental platform is only a prototype, the main goal is to demonstrate that the framework and systems can actually transfer robotic knowledge from one industrial manipulator to another. Yaskawa Motoman HP3JC is chosen to be the teacher to perform and demonstrate two simple tasks: (1) Reach and Align; (2) Find Colored Object. Universal Robot UR5 and Rethink Robotics Baxter are the student robots to learn from those two tasks. Due to the size of the lab, we do not have enough space for another real industrial manipulator. Instead, we used Gazebo under Ubuntu to create a simulated UR5 and a simulated Baxter to accomplish these jobs. Then the accuracy of the knowledge transfer was evaluated.

5.1 REACH AND ALIGN

To demonstrate the utility and feasibility of the framework, this section tries to transfer a basic assembly task skill: reach and align.

1. First we use the teach pendant of Yaskawa Motoman HP3JC to program a sequence of movements. A sequence programmed by a teach pendant consists of joint angles for each motion step. Figure 5.1 shows those poses of HP3JC.

2. Second, these movements are learned by Universal Robot UR5 using algorithm described in Section 4.2 and added into the motion library.

3. Finally, based on starting position x_0, y_0, z_0 and orientation with goal position x_g, y_g, z_g and orientation, a sequence of dynamic movement primitives are selected to guide

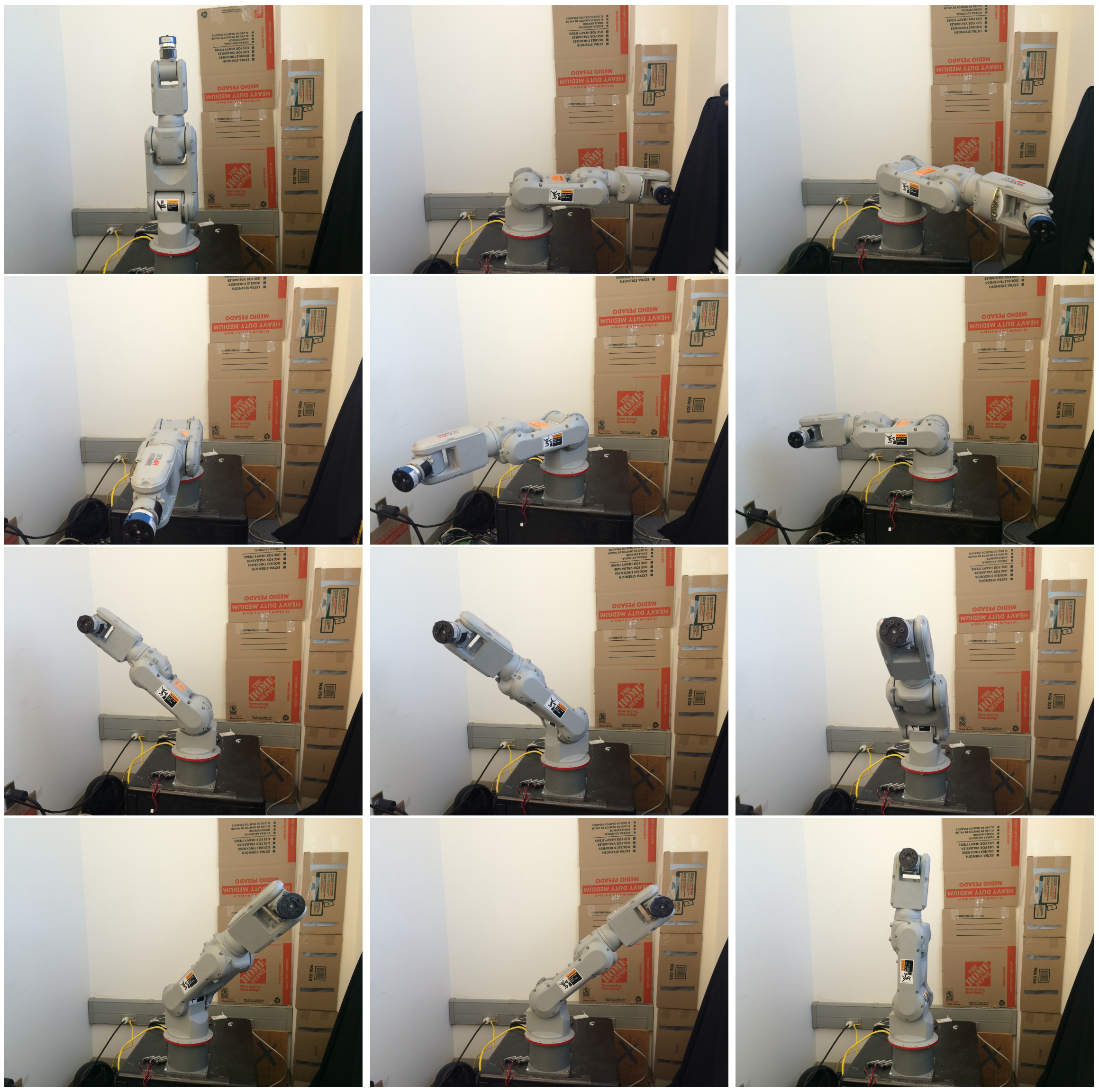


Figure 5.1: Reach and Align, HP3JC

the UR5 to the target position and orientation. The end-effector's orientation is represented in the form of rotation R submatrix in Eq 3.2. Figure 5.2. shows those poses of UR5.

In this experiment, HP3JC moved to a series of positions with its end-effector facing forward all the time. We used the MotoSDK software to obtain its corresponding joint variables and did the forward kinematics to calculate those movement primitives. Then UR5 learned from the demonstration and tried to accomplish the same task using similar

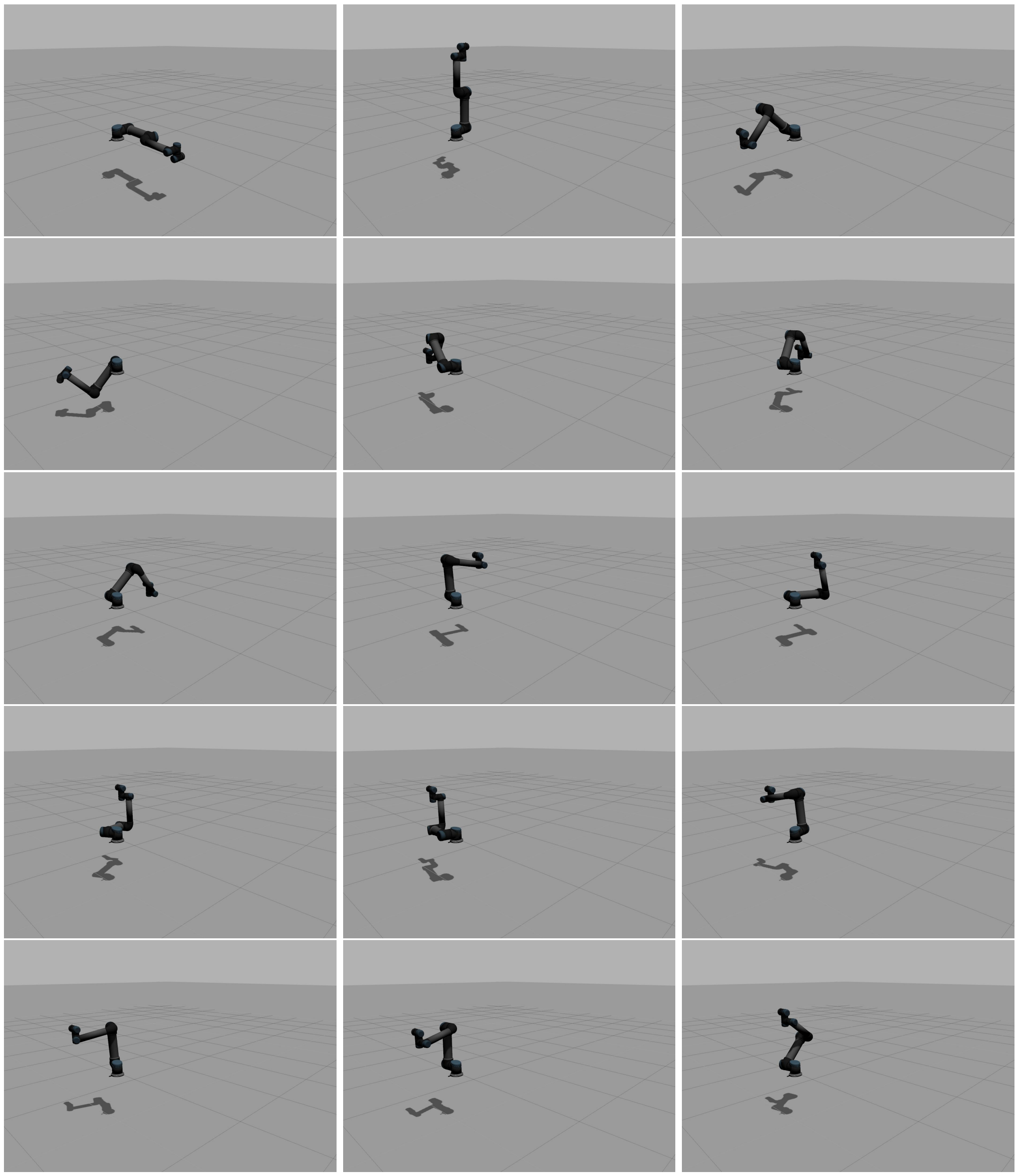


Figure 5.2: Reach and Align, UR5

path trajectory. Table 5.1 shows the testing results. We computed the orientation error and distance error between HP3JC and UR5.

$$orientation\ error = \sum_i \sum_j |R_g - R| \tag{5.1}$$

$$distance\ error = \sqrt{|x_g - x|^2 + |y_g - y|^2 + |z_g - z|^2} \tag{5.2}$$

	x_g	x	y_g	y	z_g	z	dis error	ori error
step1	0.568	0.4702	5.249	5.6468	1.4115	0.8781	0.6726	0
step2	4.301	4.3884	4.371	4.2632	1.331	1.2032	0.1887	0
step3	6.652	6.6673	0.164	0.056	1.304	1.2984	0.1092	0
step4	5.170	5.1087	-3.739	-3.8115	1.334	1.3351	0.095	0
step5	1.397	1.3039	-5.264	-5.2708	1.435	1.4376	0.0934	0
step6	1.384	1.349	-3.804	-3.7443	5.216	5.293	0.1035	0
step7	3.773	3.8253	-2.953	-2.9255	5.24	5.2542	0.0608	0
step8	5.190	5.1985	0.036	0.1161	5.255	5.255	0.0805	0
step9	3.521	3.4564	3.183	3.2328	5.206	5.2154	0.0821	0
step10	1.519	1.475	3.788	3.7902	5.162	5.1619	0.0441	0

Table 5.1: Error between Desired Goal Position/Orientation and Actual Position/Orientation

From the above results, we can tell that the framework works pretty well for the UR5 to reach at the same positions with same orientations as HP3JC. Although the distance error is high at the first step, it monotonically decreases along the whole path trajectory. The orientation error equals to 0 all the time since once we specify the initial orientation and goal orientation, the sequence of dynamic movement primitives will assign the same orientation for all steps.

5.2 Find Colored Object

This section analyzes another important task to evaluate the framework. Since many industrial manipulators utilize visual servo systems, we transfers the skill of reaching colored

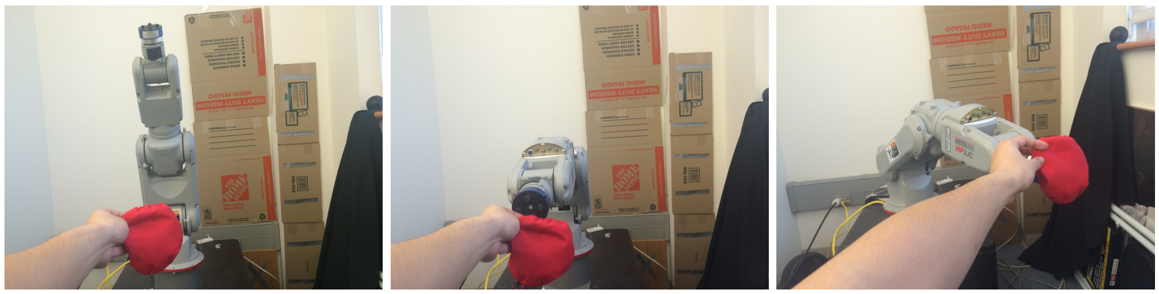


Figure 5.3: Colored Object Reaching, HP3JC

object and the accuracy of the transfer is evaluated.

1. First we use Yaskawa Motoman HP3JC to perform a colored object reaching task with the Kinect visual servo system. The Kinect servo system computes the colored object's 3D coordinates and publishes them on a channel via ROS. Then NXC100 controller subscribes to that channel to acquire the 3D information and calculates the inverse kinematics to guide the robot to reach that colored object. This step is shown in Figure 5.3. Figure 5.4 shows the mechanism of the robot communication.

2. Second, these movements are learned by Universal Robot UR5/Rethink Robotics Baxter and added into the motion library.

3. Third, we create a simulated UR5/Baxter controller to subscribe the 3D coordinate information on the same channel.

4. Finally, based on the initial position x_0, y_0, z_0 and colored object's goal position x_g, y_g, z_g, a sequence of dynamic movement primitives are selected to guide the UR5/Baxter to the target position. Figure 5.5 shows one sequence of movements of the student robot UR5. Figure 5.6 shows one sequence of movements of the student Baxter.

In this experiment, step 1 was done for only once which means the teacher robot Yaskawa Motoman HP3JC demonstrates the reaching colored object task once. Then step $2 \sim 4$ were repeated and each time a new different goal position was specified to the student

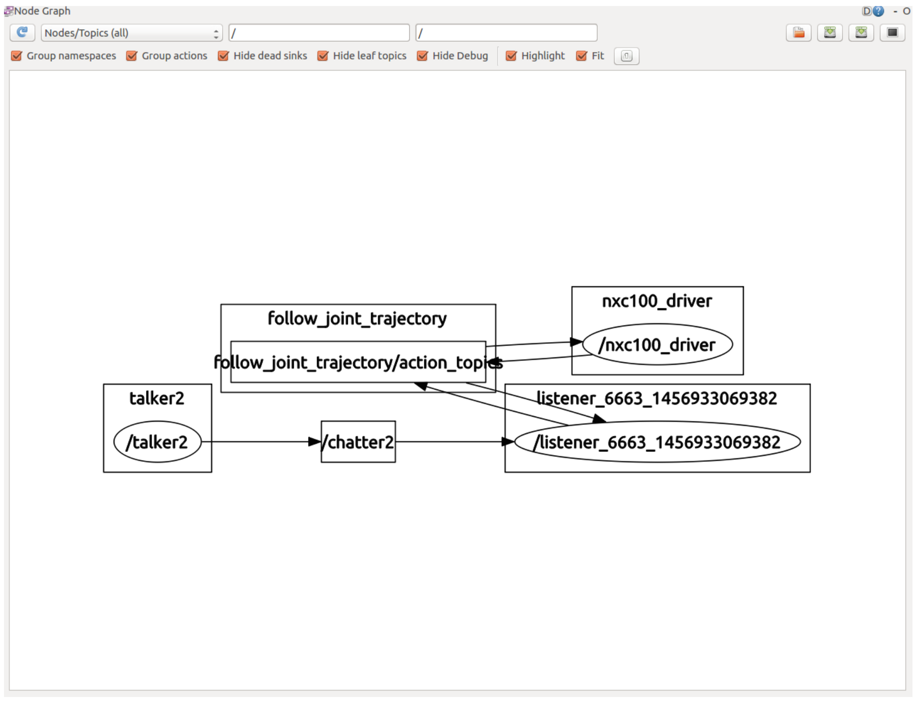


Figure 5.4: Mechanism of the Robot Communication via ROS

robot UR5/Baxter to see how well it can apply what it has learned to solve new challenges. Table 5.2 and table 5.3 show the corresponding testing results of UR5 and Baxter. We computed the 3D mismatch errors between the assigned goal positions and the final real positions.

$$error = \sqrt{|x_g - x|^2 + |y_g - y|^2 + |z_g - z|^2} \tag{5.3}$$

From table 5.2 we can tell that the approach works with some errors. It transfers the colored object reaching skill from the Yaskawa Motoman HP3JC to Universal Robot UR5. The UR5 learns the path which HP3JC used to reach the target and furthermore it uses a similar path to reach the colored object placed at different places.

From table 5.3 we can also tell that the approach transfers the colored object reaching skill from the Yaskawa Motoman HP3JC to Rethink Robotics Baxter with some error. However, we did notice that sometime the student robot Baxter failed to execute a motion sequence in the experiment. The reason behind this is that though we generate a trajectory

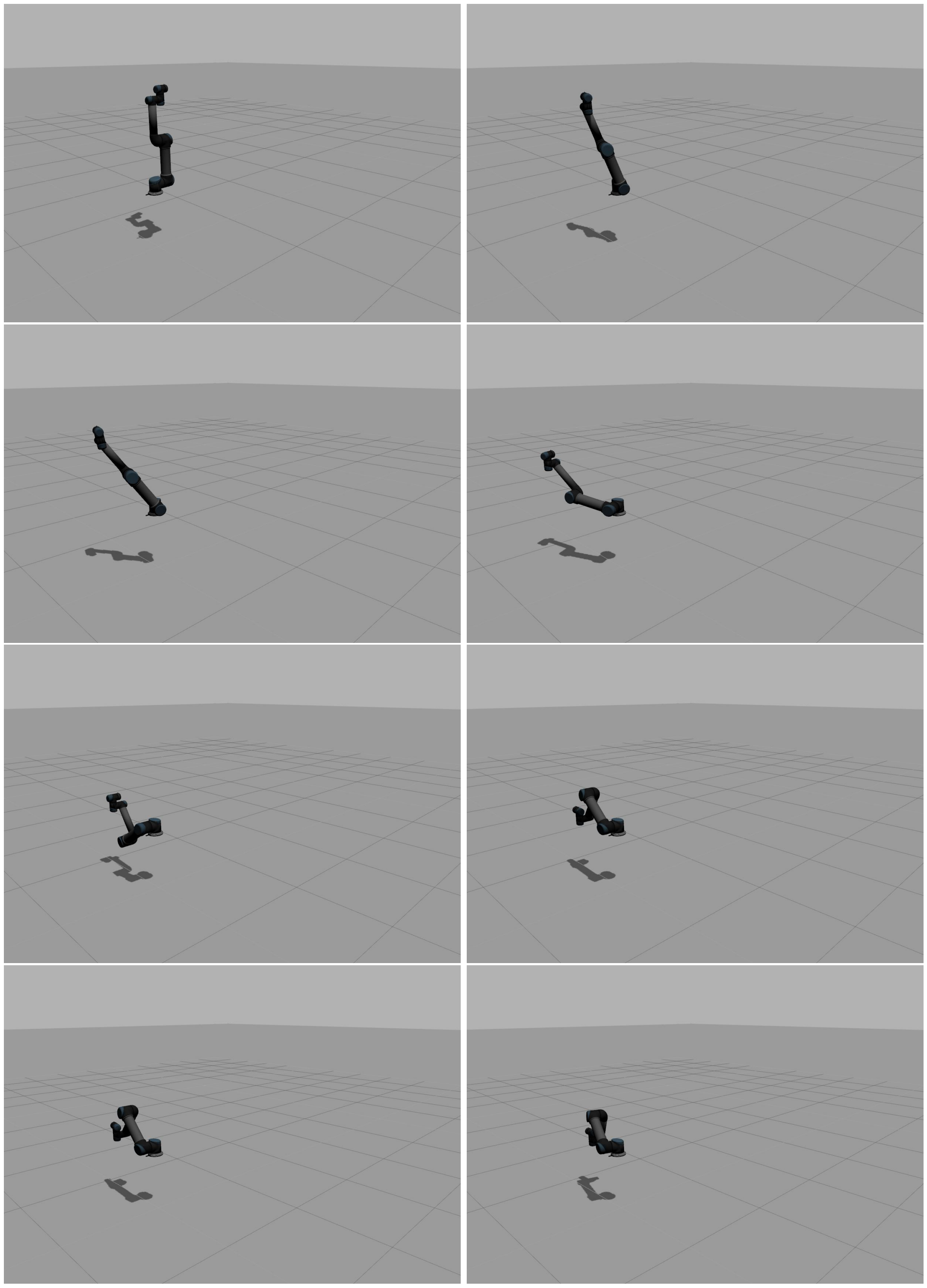


Figure 5.5: Colored Object Reaching, UR5

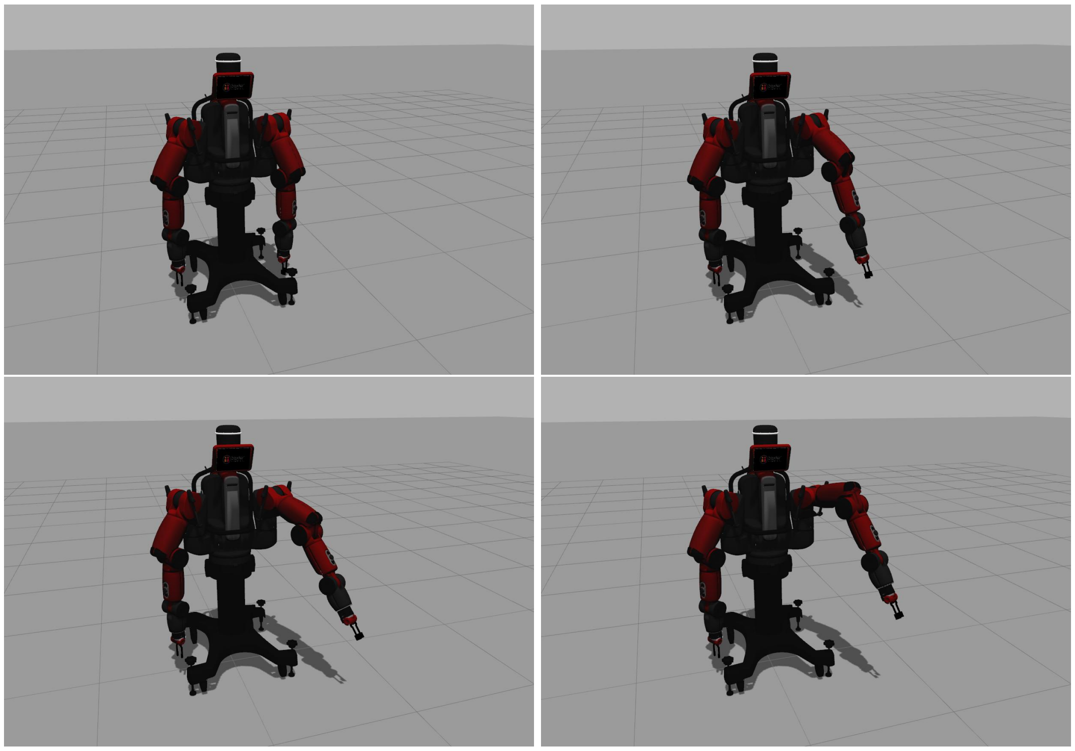


Figure 5.6: Colored Object Reaching, Baxter

for Baxter to reach the object, it cannot rotate some of its joints to some degree due to geometry constraints(e.g. the elbow can only bend inside). The inverse kinematic solver doesn't take these geometry constraints into account.

	x_g	x	y_g	y	z_g	z	error
demo	6.437	6.437	-0.187	-0.187	1.709	1.709	0
Test1	3.8622	3.9498	-0.1122	-0.0798	1.0254	0.5405	0.4938
Test2	3.8622	3.9498	-0.1122	-0.0798	1.7090	1.2678	0.4510
Test3	3.8622	3.9498	-0.1122	-0.0798	2.3926	1.9940	0.4094
Test4	3.8622	3.9498	-0.1870	-0.1541	1.0254	0.5405	0.4939
Test5	3.8622	3.9498	-0.1870	-0.1541	1.7090	1.2678	0.4510
Test6	3.8622	3.9498	-0.1870	-0.1541	2.3926	1.9940	0.4094
Test7	3.8622	3.9498	-0.2618	-0.2286	1.0254	0.5405	0.4939
Test8	3.8622	3.9498	-0.2618	-0.2286	1.7090	1.2678	0.4510
Test9	3.8622	3.9498	-0.2618	-0.2286	2.3926	1.9940	0.4095
Test10	6.4370	6.4951	-0.1122	-0.0798	1.0254	0.5405	0.4895
Test11	6.4370	6.4951	-0.1122	-0.0798	1.7090	1.2678	0.4462
Test12	6.4370	6.4951	-0.1122	-0.0798	2.3926	1.9940	0.4041
Test13	6.4370	6.4951	-0.1870	-0.1541	1.0254	0.5405	0.4895
Test14	6.4370	6.4951	-0.1870	-0.1541	1.7090	1.2678	0.4462
Test15	6.4370	6.4951	-0.1870	-0.1541	2.3926	1.9940	0.4042
Test16	6.4370	6.4951	-0.2618	-0.2286	1.0254	0.5405	0.4895
Test17	6.4370	6.4951	-0.2618	-0.2286	1.7090	1.2678	0.4462
Test18	6.4370	6.4951	-0.2618	-0.2286	2.3926	1.9940	0.4042
Test19	9.0118	8.9730	-0.1122	-0.0798	1.0254	0.5405	0.4878
Test20	9.0118	8.9730	-0.1122	-0.0798	1.7090	1.2678	0.4443
Test21	9.0118	8.9730	-0.1122	-0.0798	2.3926	1.9940	0.4021
Test22	9.0118	8.9730	-0.1870	-0.1541	1.0254	0.5405	0.4878
Test23	9.0118	8.9730	-0.1870	-0.1541	1.7090	1.2678	0.4444
Test24	9.0118	8.9730	-0.1870	-0.1541	2.3926	1.9940	0.4021
Test25	9.0118	8.9730	-0.2618	-0.2286	1.0254	0.5405	0.4878
Test26	9.0118	8.9730	-0.2618	-0.2286	1.7090	1.2678	0.4444
Test27	9.0118	8.9730	-0.2618	-0.2286	2.3926	1.9940	0.4021

Table 5.2: Error between Desired Goal Position and Actual Position of UR5

	x_g	x	y_g	y	z_g	z	error
demo	6.437	6.437	-0.187	-0.187	1.709	1.709	0
Test1	9.743	9.6873	-0.500	-0.5251	5.826	5.9763	0.1622
Test2	9.743	9.6873	-0.500	-0.5251	6.326	6.4929	0.1777
Test3	9.743	9.6018	-0.500	-0.5185	6.826	6.9496	0.1885
Test4	9.743	9.6873	0.000	0.000	5.826	5.9763	0.1603
Test5	9.743	9.6873	0.000	0.000	6.326	6.4929	0.1759
Test6	9.743	9.6145	0.000	0.000	6.826	6.9583	0.1845
Test7	9.743	9.6873	0.500	0.5181	5.826	5.9763	0.1613
Test8	9.743	9.6873	0.500	0.5181	6.326	6.4929	0.1768
Test9	9.743	9.6022	0.500	0.5117	6.826	6.9498	0.1879
Test10	10.243	10.1770	-0.500	-0.5225	5.826	5.9586	0.1498
Test11	10.243	9.9779	-0.5000	-0.5084	6.326	6.3609	0.2675
Test12	10.243	9.7732	-0.500	-0.4941	6.826	6.7300	0.4796
Test13	10.243	10.1915	0.000	0.000	5.826	5.9656	0.1488
Test14	10.243	9.9910	0.000	0.000	6.326	6.3681	0.2555
Test15	10.243	9.7850	0.000	0.000	6.826	6.7373	0.4666
Test16	10.243	10.1774	0.500	0.5156	5.826	5.9587	0.1489
Test17	10.243	9.9783	0.500	0.5017	6.326	6.3611	0.2671
Test18	10.243	9.7735	0.500	0.4876	6.826	6.7301	0.4794
Test19	10.743	10.2803	-0.500	-0.4951	5.826	5.7722	0.4658
Test20	10.743	10.1015	-0.500	-0.4835	6.326	6.1653	0.6615
Test21	10.743	9.9152	-0.500	-0.4715	6.826	6.5296	0.8797
Test22	10.743	10.2934	0.000	0.000	5.826	5.7779	0.4521
Test23	10.743	10.1135	0.000	0.000	6.326	6.1714	0.6482
Test24	10.743	9.9262	0.000	0.000	6.826	6.5357	0.8669
Test25	10.743	10.2807	0.500	0.4885	5.826	5.7724	0.4656
Test26	10.743	10.1019	0.500	0.4771	6.326	6.1655	0.6613
Test27	10.743	9.9155	0.500	0.4653	6.826	6.5297	0.8796

Table 5.3: Error between Desired Goal Position and Actual Position of Baxter

Chapter 6

CONCLUSION

We set up a working system consisting of a teacher robot Yaskawa Motoman HP3JC, a NXC100 controller, a Kinect visual system and a PC. We chose Universal Robot UR5 (6 joints) and Rethink Robotics Baxter (7 joints) to be the student robots to learn from demonstrations performed by the HP3JC. The testing results show that the framework works well with some errors for both UR5 and Baxter. This leads us to conclude that the framework is capable of transferring robotic skills across heterogeneous robotic platforms with different degree of freedom.

However, we did notice that there were still some drawbacks of this prototype framework. The mismatch errors are a little big than expectation. Those errors were mainly introduced when we calculated the inverse kinematics given the trajectories. Future work should improve the accuracy of skill transfer. Also, while we were trying to transfer skills from HP3JC to Baxter, even though we generated a trajectory for Baxter, Baxter sometime failed to rotate some joints due to its geometry constraints. So Baxter sometimes cannot use a similar trajectory its teacher used to accomplish a task. Solving inverse kinematics with special geometry constraints should be another focus in future work.

BIBLIOGRAPHY

[1] Michael Stanton-Geddes and Dennis Fravel. U.s. manufacturing companies are global leaders in industrial robot consumption. *USITC Executive Briefings on Trade May 2014*, May 2014.

[2] Georg Graetz and Guy Michaels. Robots at work. *CEP Discussion Paper*, 1335:52, March 2015.

[3] Bill Gates. A robot in every home. *Science American*, 2007.

[4] Herman Bruyninckx. Robotics software: The future should be open. *IEEE Robotics and Automation Magazine*, pages 9–11, March 2008.

[5] Jacob Huckaby and Henrik I. Christensen. A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics. *in Proceedings of the Eighth International Cognitive Robotics Workshop*, July 2012.

[6] Jacob Huckaby and Henrik I. Christensen. Toward a knowledge transfer framework for process abstraction in manufacturing robotics. *in Proceedings of International Conference on Machine Learning 2013 Workshop: Theoretically Grounded Transfer Learning*, June 2013.

[7] Maja Mataric. Designing and understanding adaptive group behavior. *Adaptive Behavior*, pages 51–80, December 1995.

[8] Maja Mataric. Behavior-based control: Examples from navigation, learning, and group behavior. *Journal of Experimental and Theoretical Artificial Intelligence, Sepcial issue on Software Architectures for Physical Agents*, 9(2-3):323–336, 1997.

[9] T. Abbas and B.A. MacDonald. Generalizing topological task graphs from multiple symbolic demonstrations in programming by demonstrations (pbd) processes.

Robotics and Automation (ICRA), 2011, IEEE International Conference, pages 3816–3821, May 2011.

[10] Kuindersma Scott Grupen et al Konidaris, George. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 2011.

[11] Shiraj Sen Stephen Hart and Roderic Grupen. Generalization and transfer in robot control. *In proceedings of the Eighth International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, July 2008.

[12] Stephen Hart and Roderic Grupen. Learning generalizable control programs. *IEEE Transactions on Autonomous Mental Development. Special Issue on Representations and Architectures for Cognitive Systems*, July 2010.

[13] Geib C. Piater J. et al Kruger, N. Object-action complexes: Grounded abstrations of sensory-motor processes. *Robotics and Autonomous Systems*, 59(10):740–757, 2011.

[14] D.M. Lyons and M.A. Arbib. A formal model of computation for sensory-based robotics. *Robotics and Automation, IEEE Transactions*, 5(3):280–293, 1989.

[15] Nakanishi J. Shibata T. et al Ijspeert, A.J. Nonlinear dynamical systems for imitation with humanoid robots. *IEEE International Conference on Humanoid Robots*, pages 219–226, 2001.

[16] Y. Wang and Y. Jia. A fusion framework of stereo vision and kinect for high-quality dense depth maps. *Computer Vision-ACCV 2012 Workshop*, pages 109–120, 2013.

[17] O. Yilmaz and F. Karakus. Stereo and kinect fusion for continuous 3d reconstruction and visual odometry. *In Electronics, Computer and Computation (ICECCO), 2013 International Conference*, pages 115–118, 2013.

[18] Cohen S. Price B. et al Somanath, G. Stereo+kinect for high resolution stereo correspondences. *3D Vision-3DV 2013, 2013 International Conference*, pages 9–16, 2013.

[19] G. Bradsk and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, Inc., 2008.

[20] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). *In IEEE International Conference on Robotics and Automation (ICRA)*, May 2011.

[21] Motoman Inc., 805 Liberty Lane, West Carrollton, OH 45449. *Motoman NX100 Controller Ethernet Server Function Manual*, May 2007.

[22] Motoman Inc., 805 Liberty Lane, West Carrollton, OH 45449. *MotoCom SDK Function Manual*, May 2007.

[23] D.W. Eggert A. Lorusso and R.B. Fisher. A comparison of four algorithms for estimating 3-d rigid transformations. Technical report 737, Department of Artificial Intelligence, University of Edinburgh, Edinburgh, Scotland, 1995.

[24] R.B. Fisher D.W. Eggert, A. Lorusso. Estimating 3-d rigid body transformations: A comparison of four major algorithms. *Machine Vision and Applications*, 9(5):272–290, 1997.

[25] T.S. Huang K.S. Arun and S.D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Patt. Anal. Machine Intell.*, 9(698-700), 1987.

[26] J. Nakanishi A. J. Ijspeert and S. Schaal. Learning attractor landscapes for learning motor primitives. *in Advances in Neural Information Processing Systems (NIPS)*, 15:1547–1554, 2003.

[27] H. Hoffmann et al A. J. Ijspeert, J. Nakanishi. Dynamical movement primitives: Learning attractor models for motor behaviors. *Neural Computation*, 2012.

[28] http://releases.ubuntu.com/14.04/.

[29] https://dev.windows.com/en-us/kinect.

[30] http://opencv.org/.

[31] http://pointclouds.org/.

[32] http://www.mathworks.com/products/matlab/?refresh=true.

[33] http://wiki.ros.org/indigo.

[34] https://www.python.org/download/releases/2.7/.

[35] https://www.robots.com/motoman/hp3jc.

[36] http://www.universal-robots.com/products/ur5-robot/.

[37] www.rethinkrobotics.com/baxter.

[38] http://gazebosim.org/.

[39] Conley K. Gerkey B. et al Quigley, M. Ros: An open-source robot operating system. *In IEEE International Conference on Robotics and Automation (ICRA)*, 3:5, 2009.

[40] J. Gallice P. Martinet and D. Khadraoui. Vision based control law using 3d visual features. *in World Automation Congress, Robotics and Manufacturing Systems (WAC '96)*, 3:497–502, 1996.

[41] C. C. W. Hulls W. J. Wilson and G. S. Bell. Relative end-effector control using cartesian position based visual servoing. *IEEE Transactions on Robotics and Automation*, 12(5):684–696, 1996.

[42] A. P. Dani N. R. Gans and W. E. Dixon. Visual servoing to an arbitrary pose with respect to an object given a single known length. *in Proceedings of the American Control Conference (ACC '08)*, pages 1261–1267, June 2008.

[43] A. C. Sanderson L. E. Weiss and C. P. Neuman. Dynamic visual servo control of robots: An adaptive image-based approach. *in Proceedings of the IEEE International Conference on Robotics and Automantion*, pages 662–668, 1985.

[44] J. T. Feddema and O. R. Mitchell. Vision-guided servoing with feature-baded trajectory generation. *IEEE Transactions on Robotics and Automation*, 5(5):691–700, 1989.

[45] T. Ebine et al K. Hashimoto, T. Kimoto. Manipulator control with image-based visual servo. *in Proceedings of the 1991 IEEE International Conference on Robotics and Automantion*, pages 2267–2271, April 1991.

[46] Z. Qi and J. E. McInroy. Improved image based visual servoing with parallel robot. *Journal of Intelligent and Robotic Systems*, 53(4):359–379, 2008.

[47] N. Guenard et al O. Bourquardez, R. Mahony. Image-based visual servo control of the translation kinematics of a quadrotor aerial vehicle. *IEEE Transaction on Robotics*, 25(3):743–749, 2009.

[48] N. Iqbal et al U. Khan, I. Jan. Uncalibrated eye-in-hand visual servoing: An lmi approach. *Industrial Robot*, 38(2):130–138, 2011.

[49] Saeed B. Niku. *Introduction to Robotics: Analysis, Systems, Applications*. Prentics Hall, July 2001.

[50] Ronald H. Huesman Arkadiusz Sitek and Grant T. Gullberg. Tomographic reconstruction using an adaptive tetrahedral mesh defined by a point cloud. *IEEE Transaction on Medical Imaging*, 25(9), 2006.

[51] Microsoft Corporation, One Microsoft Way, Redmond, WA 98052. *Kinect Sensor*, 2010.

[52] Inge Soderkvist. Using svd for some fitting problems. Technical report, Lulea University of Technology, 971 87 Lulea, Sweden.

[53] Asfour T. Pastor P., Hoffmann H. and Schaal S. Learning and generalization of motor

skills by learning from demonstration. *In International Converence on Robotics and Automation*, pages 763–768, 2009.

[54] A W Moore C G Atkeson and S Schaal. Locally weight learning. *Artificial Intelligence Review*, (11-73), November 1997.

[55] P. I. Corke. *Robotics, Vision and Control*. Springer, 2011.