Master's Thesis

IMPLEMENTATION OF WEIGHTED DAMPED LEAST SQUARES INVERSE KINEMATICS FOR ADAPTIVE ROBOT PROGRAM CALIBRATION

Bucyeye Shema Mireille

Department of Mechanical Engineering

Graduate School of UNIST

2020

Master's Thesis

IMPLEMENTATION OF WEIGHTED DAMPED LEAST SQUARES INVERSE KINEMATICS FOR ADAPTIVE ROBOT PROGRAM CALIBRATION

Bucyeye Shema Mireille

Department of Mechanical Engineering

Graduate School of UNIST

2020

IMPLEMENTATION OF WEIGHTED DAMPED LEAST SQUARES INVERSE KINEMATICS FOR ADAPTIVE ROBOT PROGRAM CALIBRATION

Bucyeye Shema Mireille

Department of Mechanical Engineering

Graduate School of UNIST

Implementation of Weighted Damped Least Squares Inverse Kinematics for Adaptive Robot Program Calibration

A thesis/dissertation
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Bucyeye Shema Mireille

06/12/2020

Approved by

Namhun Kim

Advisor

Namhun Kim

ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Implementation of Weighted Damped Least Squares Inverse Kinematics for Adaptive Robot Program Calibration

Bucyeye Shema Mireille

This certifies that the thesis/dissertation of Bucyeye Shema Mireille is approved.

06/12/2020

Namhun Kim

Advisor: Namhun Kim

[illegible]

Duck Young Kim

[illegible]

Sang Hoon Kang

Abstract

Offline programming has gained popularity over online programming because it allows changing the robot program without stopping production, but it still falls short in terms of accuracy. Robot programs generated using offline programming need to be calibrated to determine the real positions of the robot and its peripherals in the workcell before being uploaded to the real system, because dimensional variations between the virtual and the real workcell may cause robot tool center point (TCP) position errors.

Currently to calibrate the robot workcell, robot TCP positions of strategic locations in the real workcell are measured using the teach pendant or sensors such vision systems and laser sensors, and they are later used to modify the corresponding nominal values in the virtual workcell, to generate a new robot program.

This thesis proposes a method to regenerate the robot program that is feasible in the real robot workcell. The objectives of this thesis are outlined as follows: (1) Computing the transformation matrix between the virtual and real work object frame. This is done by applying a linear regression algorithm to two sets of matching position data points, one measured in the virtual workcell and the other measured in the real workcell. (2) Path feasibility check and path regeneration, closed loop inverse kinematics is used for this task and the robot's manipulability index is used to determine how close a robot is to a singularity and subsequently move the robot away from the singularity. The proposed method was verified using the Neuromeka Indy 7 and the simulations were done using MATLAB/SIMULINK.

Contents

List of Figures

ABBREVIATIONS

Notation	Description
$\boldsymbol{X}$	Matrix (bold, italic, and capital)
X	Vector (italic and capital)
$(\boldsymbol{X})_{i,:} = X_{i,:}$	*i*th row of matrix X
$(\boldsymbol{X})_{:,j} = X_{:,j}$	*j*th column of matrix X
$(\boldsymbol{X})_{i,j} = x_{i,j}$	Matrix element
N	Number of path points
$^{V}\boldsymbol{PM}$	Original path matrix (virtual workbench frame) of size N× 6
$^{R}\boldsymbol{PM}$	New path matrix (real workbench frame) of size N× 6
$^{R}\boldsymbol{T}_{V}$	Error compensation matrix
M	Number of maximum iterations
Q_i	Joint configuration vector at the *i*th path point $(^{R}\boldsymbol{PM})_{i,:}$
$\boldsymbol{S}$	End effector pose matrix
$\boldsymbol{S_j}$	End effector pose matrix at *j*th instance
P	End effector pose vector
P_i	End effector pose vector at the *i*th path point
ΔP	End effector pose increment
ΔQ	Joint configuration increment
$\dot{P}$	End effector velocity
$\dot{Q}$	Joint velocity
$\boldsymbol{J}$	Jacobian matrix
W	Manipulability index
K	Gain
μ	Damping factor to reduce joint velocities in the vicinity of singularities
μ_0	Damping scale variable in the vicinity of singularities
ε	Small positive number that denotes tolerable error

CHAPTER 1 INTRODUCTION

1.1 Background

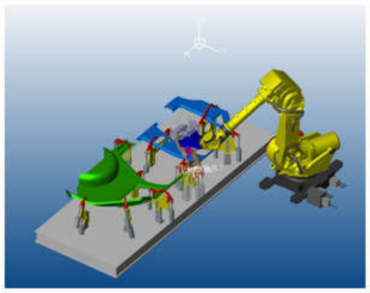
Industrial robot manipulators are crucial parts of the automated manufacturing industry, they increase production quality and productivity, improve reconfigurability and flexibility, and can be used for a high range of applications. Their common use relies on programming the robot end effector with high accuracy. Usually, robot manipulators have a higher repeatability than accuracy, which is acceptable for mass production, but as manufacturing has shifted from mass production to mass customization, where manufacturing cells are frequently reconfigured, the accuracy of robot manipulators need to be improved.

Robot programming methods can be divided into two categories: online programming and offline programming (OLP). With online programming, the operator uses a teach pendant to maneuver the robot to the desired positions in the real workcell, and the related joint angles are registered at each desired position to generate a robot program. Online programming has the following shortcomings: the process is time-consuming and tedious, the program generated by teach pendant is unique to the robot and thus cannot be copied to a robot performing a similar task, and the quality of the produced robot program depends upon the operator's skills.

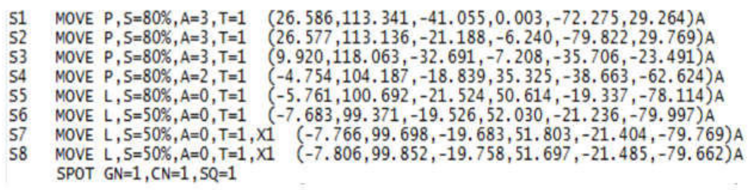
With offline programming, the robot is programmed in a virtual environment (a robot workcell is represented by a 3D CAD model) i.e. programming is done in the absence of the physical robot; and the program is later transferred to the physical robot. This is convenient in that there is no production downtime when the robot is being programmed as opposed to online programming, where the robot must be out of service for programming, and the robot program quality is consistent since it no longer relies on the skills of the robot operator.

Figure 1.1 Online programming using teach pendant

In general, multiple robot workcells perform identical tasks by using the same offline robot program (e.g. spot-welding robots in an automotive body-shop), which makes offline programming a good choice because the same robot program will be copied to the identical robot workcells. Commercial OLP packages include: eZRobotics (DMWorks), Delfoi Robotics, Delmia IGRIP, Siemens RobCAD, ABB RobotStudio, RoboDK, and Roboguide (Fanuc).



Virtual robot workcell



```
S1    MOVE P,S=80%,A=3,T=1  (26.586,113.341,-41.055,0.003,-72.275,29.264)A
S2    MOVE P,S=80%,A=3,T=1  (26.577,113.136,-21.188,-6.240,-79.822,29.769)A
S3    MOVE P,S=80%,A=3,T=1  (9.920,118.063,-32.691,-7.208,-35.706,-23.491)A
S4    MOVE P,S=80%,A=2,T=1  (-4.754,104.187,-18.839,35.325,-38.663,-62.624)A
S5    MOVE L,S=80%,A=0,T=1  (-5.761,100.692,-21.524,50.614,-19.337,-78.114)A
S6    MOVE L,S=50%,A=0,T=1  (-7.683,99.371,-19.526,52.030,-21.236,-79.997)A
S7    MOVE L,S=50%,A=0,T=1,X1  (-7.766,99.698,-19.683,51.803,-21.404,-79.769)A
S8    MOVE L,S=50%,A=0,T=1,X1  (-7.806,99.852,-19.758,51.697,-21.485,-79.662)A
      SPOT GN=1,CN=1,SQ=1
```

Example of a robot program

Figure 1.2 Offline programming

Figure 1.3 shows an example of a robot workcell with its reference frames and the corresponding frame transformations, where:

- {W} World coordinate frame is usually placed in a strategic position in the workcell where it will be convenient to measure the location of the robot and its peripherals. In a simple workcell with few equipment, it can be made to coincide with the robot base coordinate frame.

- {B} Robot base coordinate frame is located at the base of the robot manipulator, otherwise known as link zero. The robot manipulator's forward kinematics (the end effector pose) are defined relative to this frame. The robot base coordinate frame is defined with respect to the world coordinate frame.
- {E_0} End effector_0 coordinate frame whose origin is located at the last link's endpoint and is defined relative to the robot base coordinate.
- {E} End effector coordinate frame whose origin is placed at the tool center point (TCP) and is defined relative to the end effector _0 coordinate frame.
- {Tg} Target coordinate frame defines the location of an object with respect to the workbench coordinate frame.
- {O} Work Object coordinate frame is usually described relative to the world coordinate frame. It is preferable to define the robot path in this coordinate frame. Usually the end effector poses are programmed in this frame.

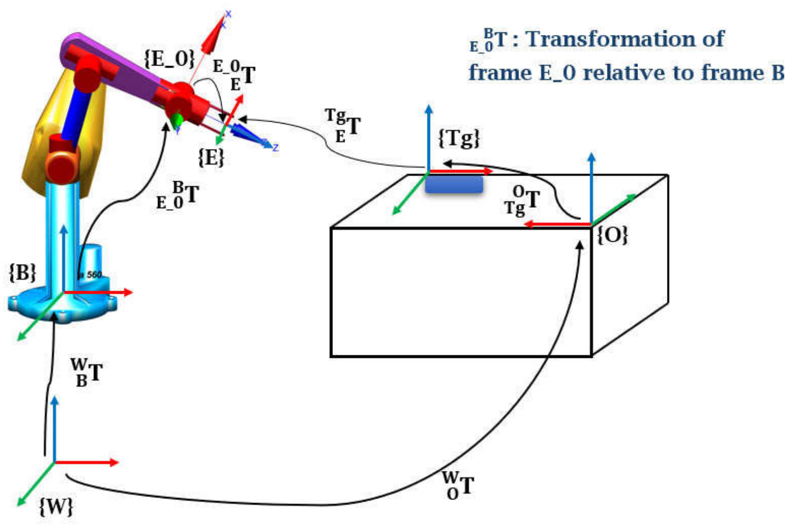


Figure 1.3 Workcell coordinate frames

Equation (1.1) denotes the relationship between the workcell coordinate frames.

$$ {}^{Tg}_{E}T = \left({}^{W}_{O}T\,{}^{O}_{Tg}T\right)^{-1}\,{}^{W}_{B}T\,{}^{B}_{E\_0}T\,{}^{E\_0}_{E}T \qquad (1.1)$$

1.2 Motivation

Offline programming has gained popularity over online programming because it allows changing the robot program without stopping production, but it still falls short in terms of accuracy. Robot programs generated using OLP need to be calibrated to determine the true positions of the robot and its peripherals in the real workcell before being uploaded to the real system, because dimensional variations between the virtual and the real workcell may cause robot tool center point (TCP) position errors.

The goal of workcell calibration is to define the true transformations (in the real workcell) between the robot workcell coordinate frames. The calibration process is often time consuming and it requires high accuracy measuring devices, and a skilled robot programmer.

A common method used for calibration is to measure the robot TCP positions of strategic locations in the real workcell using the teach pendant, and subsequently use those measurements to modify the corresponding nominal values in the virtual workcell to generate a new robot program. This method's weakness is that it relies on the operator's skills to move the robot to accurate positions. Methods that incorporates sensors, such as vision systems, and laser sensors in the calibration process also make the corresponding changes to the OLP manually.

For these reasons above, an adaptive robot program calibration method is devised in this thesis to assist the user in the calibration procedure by closely maintaining the shape of the initial robot path in the virtual work object frame.

The first step to calibration is the identification of the error compensation matrix ${}^{R}_{V}\boldsymbol{T}$ that represents the dimensional variations between the virtual and real workcells as can be shown in Figure 1.4.

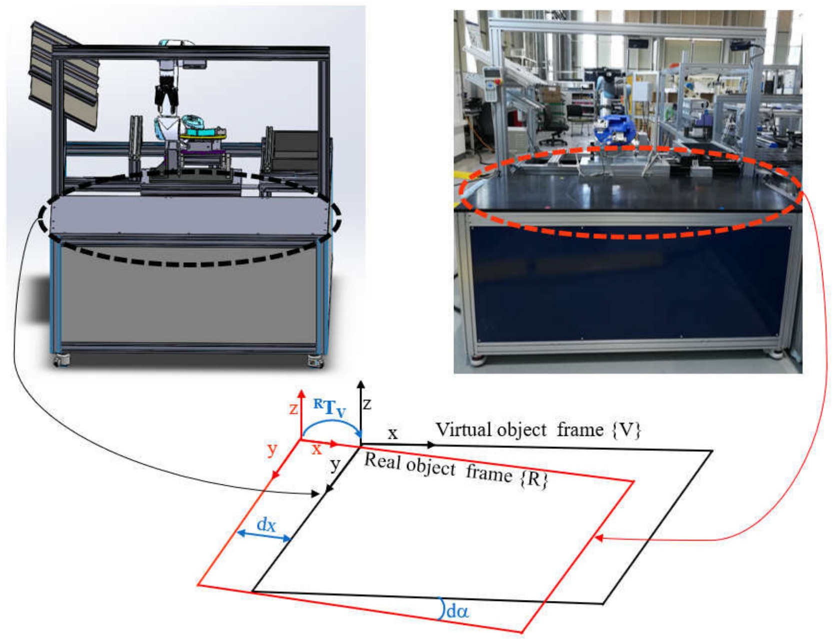


Figure 1.4 Dimensional variations between the virtual and real work object frames

The next step is to regenerate the robot path by considering the dimensional variations obtained in the first step. This is usually done manually, reason why it requires a skilled robot programmer. Thus, the need to automate the path regeneration process as nowadays robot manipulators are common in manufacturing industries.

While regenerating the robot path the approach vector length and angle, (from Figure 1.5 $\overrightarrow{{}^{R}PM_2\,{}^{R}PM_3}$) should be preserved since it is the most important part of the robot task.

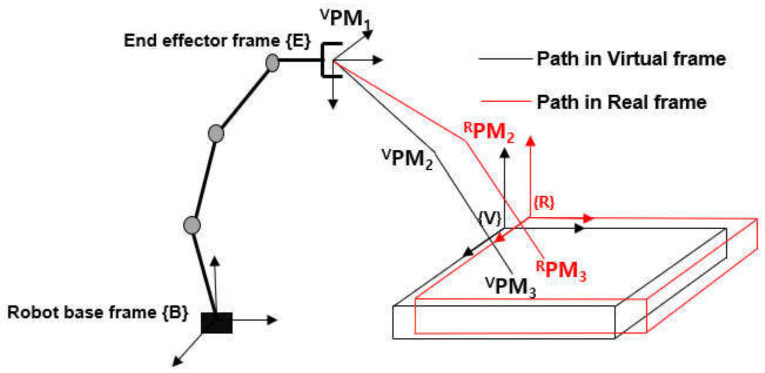


Figure 1.5 Path regeneration

Experimental simulations results from the Neuromeka Indy 7 showed that a dimensional variation of ten millimeters may introduce singularities in the regenerated robot path. In this case a path that avoids the singular regions needs to be regenerated.

1.3 Objective

The purpose of this thesis is to generate an accurate and feasible path in the real work object coordinate frame given a path in the virtual work object frame. This task comprises of two main parts: (1) Transformation matrix (error compensation matrix) computation and (2) Path feasibility check and regeneration.

Transformation Matrix Computation

To compute the transformation matrix that expresses the virtual work object coordinate frame with respect to the real work object frame ${}^{R}_{V}\boldsymbol{T}$, the transformation matrix that relates the real work object frame to the robot base frame ${}^{B}_{\{O\_R\}}\boldsymbol{T}$ is determined first, by applying the linear regression algorithm to two sets of matching position data points. A kinematically calibrated robot manipulator is used to measure the position of data points in the robot base frame ***{B}*** and the data points in the real work object frame are obtained from the workcell CAD model. Since the transformation matrix that describes the virtual work object frame with respect to the robot base frame ${}^{\boldsymbol{B}}_{\{\boldsymbol{O\_V}\}}\boldsymbol{T}$ is extracted from the CAD model, ${}^{\{\boldsymbol{O\_R}\}}_{\{\boldsymbol{O\_V}\}}\boldsymbol{T}$ (in short ${}^{\boldsymbol{R}}_{\boldsymbol{V}}\boldsymbol{T}$) can be computed by compound transformations.

Path Feasibility Check and Regeneration

Once a path in the real work object coordinate frame is obtained using the transformation matrix; the feasibility check will determine whether the path can be uploaded to the real system or whether the path needs to be regenerated. For the feasibility check, closed loop inverse kinematics is used to obtain a feasible path in the joint space by checking for singularities. The robot's manipulability index is used to determine how close a robot is to a singularity and subsequently move the robot away from the singularity.

1.4 Overview

This thesis is organized as follows: chapter one outlines the introduction. Chapter two denotes the literature review. Chapter three explains how to compute the transformation matrix and also elaborates the robot path feasibility check algorithm. Chapter four demonstrates the experimental and verification results. In chapter five, a conclusion and future works are elaborated.

CHAPTER 2

LITERATURE SURVEY

2.1 Robot Programming

The three main approaches to Robot programming as outlined by (Lozano-Perez 1983) are: programming by guiding, robot-level programming, and task-level programming. In programming by guiding, the operator manually moves the robot to the desired positions, using a teach pendant, and the joint coordinates positions are registered and later used to develop a robot program made of a sequence of joint coordinates. This method is fast and straightforward when the robot must perform tasks that involve simple motions and do not require sensory information. However, programming by guiding is limited, when the robot tasks require complex motions and sensory information, like inspection or assembly. Other limitations to be cited are that this type of programming can be quite tedious, when the same task must be repeated at different points and achieving fine positioning accuracy is hard.

Robot-level programming allows the use of computer programming languages to acquire and use sensor data, and to define desired motions. The acquired sensor data such as force and vision can be used to specify the robot motions. The drawback of this method is that it requires the user to have expertise in computer programming, and sensor data analysis.

Task-level programming involves determining target positions of objects instead of determining the motions required to achieve those targets. This method is does not depend on the robot in that the user does not specify paths or motions that are dependent on the robot kinematics.

(Biggs and MacDonald 2003) classified robot programming methods into: manual programming, automatic programming, and software architectures. In manual programming systems, the user generates the robot program by hand. These include text-based and graphical programming language. Since the robot is not present during programming, these methods are often referred to as offline programming systems. Automatic programming systems require little or no direct involvement of the user to develop the robot program. The robot code is created from data input to the system in various ways. These comprise programming by demonstration, learning, and instructive systems. For these systems, programming is done while the robot system is running, so they are also called online programming systems. The importance of software architectures lies in that they provide communication and access to robots.

Offline programming is usually done on a CAD model of the robot and its peripherals; and sometimes there are dimensional variations between the physical system and the CAD model. The robot programs generated in this way need to be calibrated since these dimensional errors will affect the robot's absolute pose accuracy. The need for calibration has impeded the use of OLP in industries. Online programming does not require calibration, because dimensional errors do not affect the robot's repeatability (Duelen and Schröer 1991). Figure 2.1 outlines the process of offline programming (OLP).

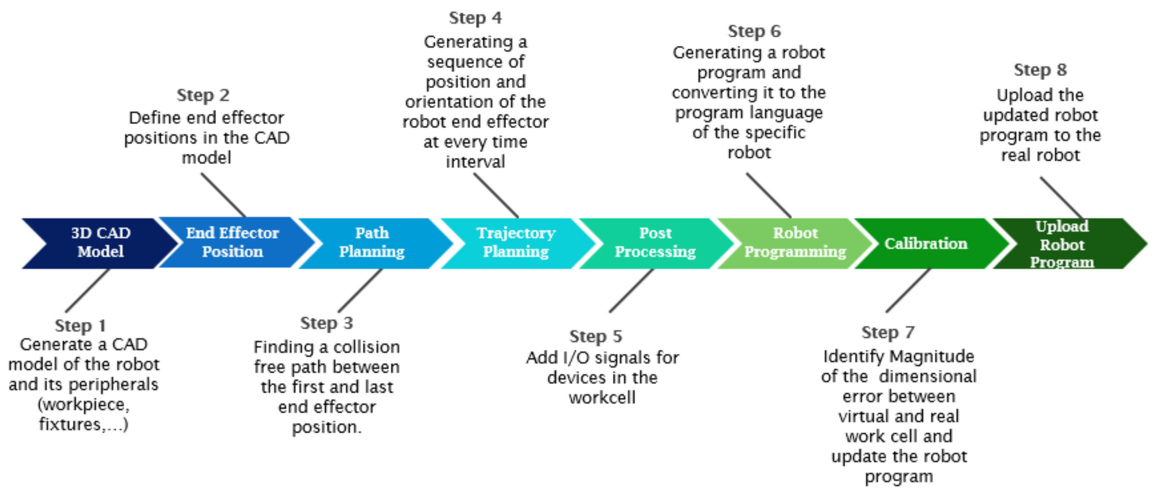

Step 4
Generating a sequence of position and orientation of the robot end effector at every time interval
Step 6
Generating a robot program and converting it to the program language of the specific robot
Step 2
Define end effector positions in the CAD model
Step 8
Upload the updated robot program to the real robot
3D CAD Model
End Effector Position
Path Planning
Trajectory Planning
Post Processing
Robot Programming
Calibration
Upload Robot Program
Step 1
Generate a CAD model of the robot and its peripherals (workpiece, fixtures,...)
Step 3
Finding a collision free path between the first and last end effector position.
Step 5
Add I/O signals for devices in the workcell
Step 7
Identify Magnitude of the dimensional error between virtual and real work cell and update the robot program

Figure 2.1 Offline Programming process (Pan, Polden et al. 2010)

2.2 Kinematic modelling and forward kinematics

A robot manipulator can be modelled as a kinematic chain comprised of rigid bodies called links that are connected to each other by joints. The joints can either be prismatic, for a linear motion, or revolute, for a rotary motion. For a manipulator with 1, …, N joints, link 0 is constrained at the base, whereas the last link (link N) is the robot end effector. The number of joints represents the number of degrees of freedom for a given manipulator. The motion of the robot is derived from the composition of each link's motion with respect to the previous one. To describe the location of each link a coordinate frame is attached to it i.e. frame i to link i. The relative position and orientation of two consecutive links can be obtained using the modified Denavit-Hartenberg (DH) convention by (Craig 2009); the link is defined by two parameters: the link length (a_i) and the link twist (α_i), and the joint is defined by the offset length (Lozano-Perez 1983) and the joint angle (θ_i). The four parameters are called the DH parameters.

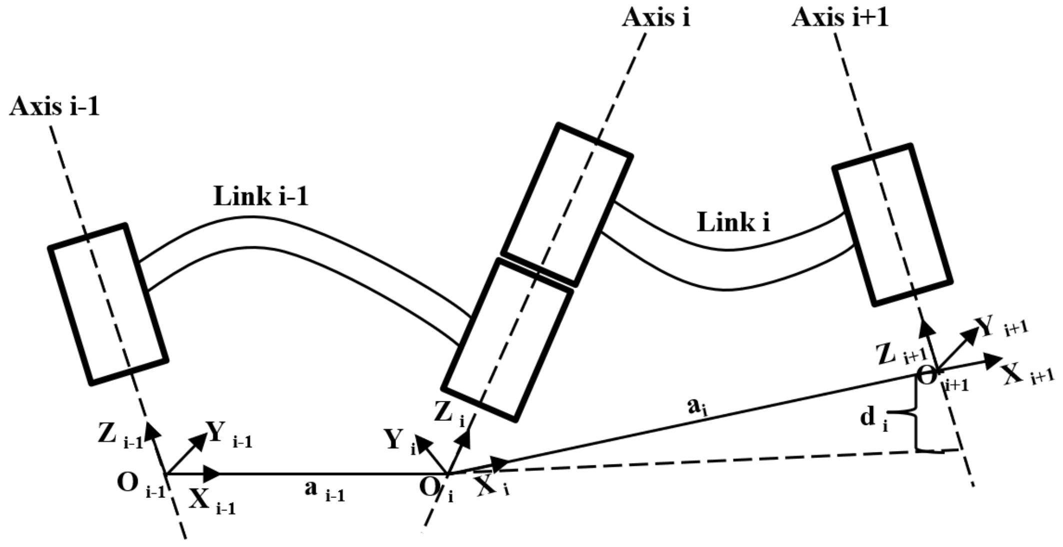


Figure 2.2 Modified Denavit-Hartenberg convention

The coordinate frame attached to each link can be defined as:

- Axis Z_i is coincident with the i^{th} joint axis.
- Axis X_i *is* taken along the common perpendicular to Z_i and Z_{i+1}, in the direction from joint axis i to $i+1$
- The origin O_i is at the intersection of Z_i and the common perpendicular to Z_i and Z_{i+1}
- Axis Y_i is selected to form the right-hand rule

The DH parameters can be defined as

- Link length a_{i-1}: the distance between Z_{i-1} and Z_i along X_{i-1}
- Link twist α_{i-1}: the angle between Z_{i-1} and Z_i about X_{i-1}
- Offset length d_i: the distance between X_{i-1} and X_i along Z_i
- Joint angle θ_i: the angle between X_{i-1} and X_i about Z_i

For a robot manipulator, three parameters are fixed and only one joint variable Q_i, (d_i for prismatic joint and θ_i for revolute joint), changes during robot movement. To describe motion from link i-1 to link i, link transformation ${}^{i-1}_{i}T$ may be expressed as follows starting from link i+1: a rotation α_{i-1} about X_{i-1}, followed by a translation ai-1 along X_{i-1}, then a rotation θi about Z_i, then a translation di along Z_i.

$$ {}^{i-1}_{i}T = R_x(\alpha_{i-1})T_x(a_{i-1})R_z(\theta_i)T_z(d_i) $$
$$ = \begin{bmatrix} C\theta_i & -S\theta_i & 0 & a_{i-1} \\ S\theta_i C\alpha_{i-1} & C\theta_i C\alpha_{i-1} & -S\alpha_{i-1} & -d_i S\alpha_{i-1} \\ S\theta_i S\alpha_{i-1} & C\theta_i S\alpha_{i-1} & C\alpha_{i-1} & d_i C\alpha_{i-1} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1} $$

The transformation that links the base frame {0} to frame {n}, the end effector can be expressed as:

$$ \boldsymbol{{}^{0}_{n}T = {}^{0}_{1}T\,{}^{1}_{2}T \ldots {}^{n-1}_{n}T} \tag{2.2} $$

With the above expression, we can define the manipulator's forward kinematics, which computes the end effector pose *P* (*P* is 6 elements vector, 3 for position [x, y, z] and 3 for orientation [u, v, w]) given the joint configuration vector Q, and can be rewritten as:

$$ P = f(Q), \tag{2.3} $$

where $Q = [Q_1, Q_2, \cdots, Q_n]\,, n{:}\, number\ of\ joints$

A minimum of six joints is required for the robot to achieve any position and orientation in its workspace. For a redundant manipulator, manipulator with more than six joints, the number of joints configuration parameters exceeds that of the end effector configuration parameters. In other words, the robot manipulator's number of degrees of freedom is more than the number of degrees of freedom of its end effector. The focus of this work is on nonredundant robot manipulators.

2.3 Inverse kinematics

The inverse kinematics problem is defined as: given the desired orientation and position of the manipulator's end effector compute the corresponding joint angles to attain the desired result, in other words it computes the inverse of equation 2.3. The inverse kinematics is crucial to the control of robot manipulators; once the robot task is specified each joint is controlled individually, i.e. each robot joint must be known to accomplish the desired task (Manseur 2007).

The complexity of solving inverse kinematics problems depends largely on the geometry of the robot manipulator. Two types of solutions exist for this problem: closed form (analytical) solutions and iterative (numerical) solutions. Analytical solutions exist when robots have fewer joints and when there are pairs of parallel or intersecting joint axes. For any six degrees of freedom (DOF) robot, a closed form solution exists when there are three consecutive joints that are either parallel or perpendicular. Analytical methods provide solutions of joint variables in terms of the end effector cartesian coordinates. Numerical solutions are used when there is no closed form solution or when there is a need to develop a generic inverse kinematics solver for all kinematic configurations (Lenarcic, Bajd et al. 2012). The rest of this section will deal with some of the numerical methods used to solve inverse kinematics.

Numerical or iterative methods are used when an analytical solution cannot be achieved even though for the given set of Cartesian coordinates a joint space solution exists. Unlike, closed form solution, numerical solutions use an initial approximate value of the joint coordinates *Q(0)* to iteratively converge towards a solution. The initial condition affects the convergence rate of the numerical solution and whether a solution will be found; if the initial joint coordinate is far from the target an exact solution may not be obtained. For continuous trajectory tracking, the best initial condition is the joint coordinates in the previous iteration step.

Major drawbacks of these techniques include failure to converge, not obtaining all possible solutions, and computationally more expensive and slower than closed form solutions. The advantage of these methods is that they are generic and thus can be applied to various structures of robot mechanisms.

Let *Q(0)* be the initial joint coordinate, the goal is to find the target joint coordinate Q_t such that:

$$\begin{gathered} P_t - f(Q_t) = 0 \\ \text{or} \\ P_t - f(Q_t) \leq \varepsilon \end{gathered} \tag{2.4}$$

Where ε: a small positive number.

For a converging method, the error decreases and approaches zero after each iteration.

$$\varepsilon(P_0, P_t) \to 0,$$

where $P_0 = f(Q(0))$

The Newton Raphson is one of the numerical methods used to compute inverse kinematics through several iterations (Whitney 1969). Equation (2.3) is highly nonlinear but can be approximated by a linear equation using a Jacobian matrix. The Jacobian is a matrix of joint variables first order partial derivatives relative to the end effector coordinates.

Differentiating equation (2.3) with respect to Q

$$\frac{dP}{dQ} = \boldsymbol{J}(\boldsymbol{Q}), \tag{2.5}$$

where $J(Q)$: the Jacobian of *f* with respect to *Q,*

$$\boldsymbol{J}(\boldsymbol{Q}) = \frac{\partial f}{\partial Q_i},$$

where *i = 1, …, n*

Equation 2.5 can be rewritten as

$$\Delta P = \boldsymbol{J}(\boldsymbol{Q})\Delta Q \tag{2.6}$$

If *J* is a nonsingular (invertible) matrix

$$\Delta Q = \boldsymbol{J}^{-1}(\boldsymbol{Q})\Delta P \tag{2.7}$$

The Jacobian matrix expresses a linear relationship between the end effector and joint velocities

$$\begin{gathered} \dot{P} = \boldsymbol{J}(\boldsymbol{Q})\dot{Q} \\ \dot{Q} = \boldsymbol{J}^{-1}(\boldsymbol{Q})\dot{P} \end{gathered} \tag{2.8}$$

Given *Q (0)* and *P_0*, the joint coordinate *Q_t* for *P_t* can be obtained by the following iteration

At each iteration k

$$\begin{gathered} \Delta P_k = P_t - P_k \\ \Delta Q_k = \boldsymbol{J}^{-1}(\boldsymbol{Q_k})\Delta P_k \\ Q_{k+1} = Q_k + \Delta Q_k \\ P_{k+1} = f(Q_{k+1}), \qquad k = 0,1,2,\ldots \end{gathered} \tag{2.9}$$

The iteration stops when the condition in equation (2.4) is fulfilled.

(Balestrino, De Maria et al. 1984) derived another method to ensure that condition (2.4) is fulfilled.

Expressing the error between P and $f(Q)$ as ε

$$\varepsilon = P - f(Q) \tag{2.10}$$

Differentiating with respect to time

$$\dot{\varepsilon} = \dot{P} - \boldsymbol{J}(\boldsymbol{Q})\dot{Q}$$

From the Lyapunov theory (Khalil and Grizzle 2002) ,choosing K as a positive definite matrix such that

$$\dot{\varepsilon} = -\mathrm{K}\varepsilon$$

ensures that the system is asymptotically stable i.e. that the error ε goes to zero. In that case $\dot{Q}$ can be described as

$$\dot{Q} = \boldsymbol{J^{-1}}(\boldsymbol{Q})(\dot{P} + \mathrm{K}(P - f(Q))) \tag{2.11}$$

Figure 2.3 illustrates equation (2.11)

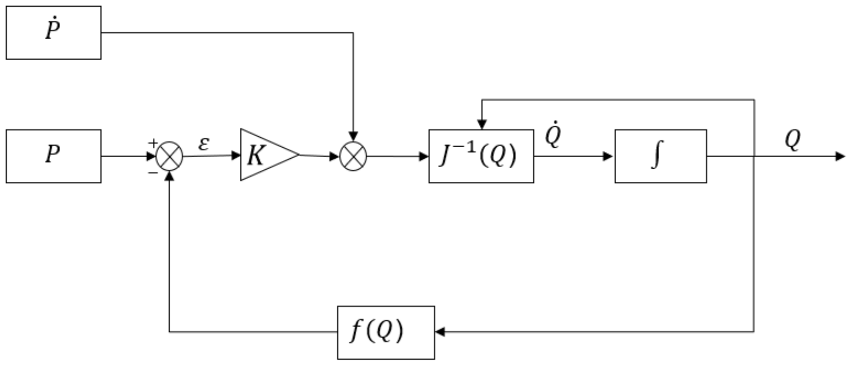


Figure 2.3 Inverse Jacobian block scheme

The solution to equation (2.9) is not unique and usually depends on the initial condition of the joint coordinates. As mentioned above, the Jacobian should be invertible to achieve a solution; but even if this is the Jacobian may perform poorly near singularities. At singularities, the Jacobian matrix loses rank, which means that the robot loses the ability to move the end effector in one or more directions even though the joints are moving. Different researchers have developed methods to avoid this problem that include the pseudoinverse, the Jacobian transpose and the singularity robust inverse (Nakamura and Hanafusa 1986, Nakamura, Hanafusa et al. 1987, Sciavicco and Siciliano 2012).

2.3.1 Resolved Motion Rate Control

In case we want the robot end effector's motion to be along a straight line in the cartesian space, the resolved motion rate control (RMRC) approach is used. (Whitney 1972)who derived the RMRC defined it as having multiple joints moving simultaneously at different and time-varying rates to attain a desired end effector motion in the cartesian space. Given the start and target pose, a smooth trajectory in the cartesian space is generated; since robots are controlled at the joint level, it is necessary to determine the corresponding joint motions. With RMRC, this is achieved by computing the Cartesian coordinates of the end effector at each instant (P_k) and from that obtaining the joint coordinates using equation (2.8). Figure 2.4 shows the process of RMRC.

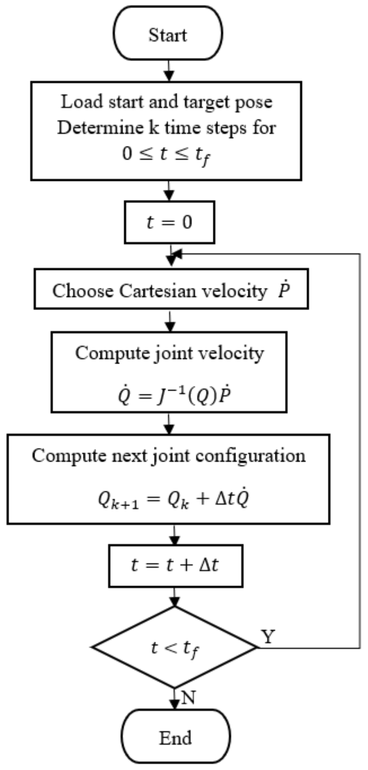

Start
Load start and target pose
Determine k time steps for
$0 \leq t \leq t_f$
$t = 0$
Choose Cartesian velocity $\dot{P}$
Compute joint velocity
$\dot{Q} = J^{-1}(Q)\dot{P}$
Compute next joint configuration
$Q_{k+1} = Q_k + \Delta t\dot{Q}$
$t = t + \Delta t$
$t < t_f$
Y
N
End

Figure 2.4 RMRC Algorithm

(Liu, Lei et al. 2016) used resolved motion control to control a humanoid robot during coordinated manipulation like lifting and placing a load. The authors developed three types of control methods at the acceleration, velocity, and position level.

(O'Neil, Chen et al. 1997) enhanced the RMRC by a including a second order condition, that if satisfied provides a solution in the neighborhood singularity. Their method also provided a measure of the recovery rate of the manipulability index around singularities.

2.3.2 Jacobian Transpose

The Jacobian transpose offers an intuitive solution of the inverse kinematics based on the Lyapunov theorem for nonlinear systems (Khalaji and Moosavian 2015). With the Jacobian Transpose, a relationship between $\dot{q}$ and ΔP that drives the error to zero, is possible. Using the Jacobian transpose, equation (2.8) can be expressed as:

$$\dot{Q} = \mathrm{K}\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{Q})\Delta P \tag{2.12}$$

where K is a positive definite matrix

The validity of equation (2.12) was proven in (Sciavicco and Siciliano 2012) by choosing a positive definite Lyapunov function of the form

$$V = \frac{1}{2}(P_t - P)^T \mathrm{K}(P_t - P)$$

Differentiating V with respect to time

$$\dot{V} = (P_t - P)^T \mathrm{K}\dot{P}_t - (P_t - P)^T \mathrm{K}\dot{P}$$

Considering equation (2.8)

$$\dot{V} = (P_t - P)^T \mathrm{K}\dot{P}_t - (P_t - P)^T \mathrm{K}\boldsymbol{J}(\boldsymbol{Q})\dot{Q}$$

By choosing joint velocities as

$$\dot{Q} = \boldsymbol{J^T}(\boldsymbol{Q})\mathrm{K}\Delta P$$

$\dot{V}$ becomes

$$\dot{V} = (P_t - P)^T \mathrm{K}\dot{P}_t - (P_t - P)^T \mathrm{K}\boldsymbol{J}(\boldsymbol{Q})\boldsymbol{J^T}(\boldsymbol{Q})\mathrm{K}\Delta P$$

For the case of a constant target ($\dot{P}_t = 0$), $\dot{V}$ is a negative definite function given that the Jacobian matrix has full rank, and from (Khalil and Grizzle 2002) it implies that the system is asymptotically stable i.e. the error converges to zero.

(Buss 2004) suggested a method to update matrix K that aims at minimizing the updated value of ΔP

$$\mathrm{K} = \frac{\langle \Delta P, \boldsymbol{JJ^T}\Delta P\rangle}{\langle \boldsymbol{JJ^T}\Delta P, \boldsymbol{JJ^T}\Delta P\rangle},$$

where $\langle\cdot\rangle$ is the dot product.

The Jacobian transpose is computationally inexpensive and does not involve inverting the Jacobian. Its limitations include slow convergence and the solution to equation (2.12) degenerates when KΔP is in the null space of $\boldsymbol{J^T}(\boldsymbol{Q})$.

2.3.3 Jacobian Pseudoinverse

The pseudoinverse offers a least square solution to equation (2.7). Unlike the inverse, which is only defined for nonsingular and square matrices, the pseudoinverse is defined for all matrices. It is mostly used to determine the inverse kinematics solutions for redundant manipulators. These solutions are not unique, hence the need to impose an optimization function $g(Q)$ that is a Euclidean norm of the joint increment vector Δq (Šoch and Lórencz 2005).

From (Klein and Huang 1983, Buss 2004), the pseudoinverse sets to find the joint incremental ΔQ that minimizes

$$\|\boldsymbol{J}\Delta Q - \Delta P\|^2 \tag{2.13}$$

To compute the value of ΔQ that minimizes equation (2.13), we differentiate equation (2.13) with respect to ΔQ and equate to zero to obtain

$$\boldsymbol{J}^{\boldsymbol{T}}\boldsymbol{J}\Delta Q = \boldsymbol{J}^{\boldsymbol{T}}\Delta P$$

The pseudoinverse is then defined as

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{\boldsymbol{T}}\left(\boldsymbol{J}^{\boldsymbol{T}}\boldsymbol{J}\right)^{\boldsymbol{-1}}$$

Thus ΔQ can be obtained by

$$\Delta Q = \boldsymbol{J}^{\dagger}(\boldsymbol{Q})\Delta P \tag{2.14}$$

A singular value decomposition of J expresses J as

$$\boldsymbol{J} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}} = \sum_{\mathrm{i}=1}^{\mathrm{r}} \sigma_{\mathrm{i}}\mathrm{u}_{\mathrm{i}}\mathrm{v}_{\mathrm{i}}^{\mathrm{T}} \tag{2.15}$$

where *r* is the rank of the Jacobian matrix

The pseudoinverse is expressed as

$$\boldsymbol{J}^{\dagger} = \boldsymbol{V}\boldsymbol{\Sigma}^{\dagger}\boldsymbol{U}^{\boldsymbol{T}} = \sum_{i=1}^{r} \frac{1}{\sigma_i} v_i u_i^T \tag{2.16}$$

For a square, invertible, full rank matrix the pseudoinverse is equal to the inverse.

$$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{\boldsymbol{-1}}$$

The pseudoinverse has a good property that makes the matrix $\left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J}\right)$ project onto the null space of J; by including the optimization function $g(Q)$ equation (2.14) is reformulated as:

$$\Delta Q = \boldsymbol{J}^{\dagger}\Delta P + \left(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J}\right)\frac{\partial g(Q)}{\partial Q}$$

Since at singularities the Jacobian loses rank, the Pseudoinverse provides a good solution instead of the inverse; but in the vicinity of singularities, the Jacobian Pseudoinverse has stability issues. Small changes in the Cartesian space will cause large changes in the joint space i.e. high joint speeds.

2.3.4 Damped Least Square Inverse

As mentioned in the previous section, around singularities the pseudoinverse exhibits high joints speeds. This can be corrected by adding a constraint function that damps the high joint velocities to equation (2.13) (Wampler and Leifer 1988)

$$\|\boldsymbol{J}\Delta Q - \Delta P\|^2 + \mu^2\|\Delta Q\|^2 \tag{2.17}$$

where μ is a nonzero damping factor.

To compute the value of ΔQ that minimizes equation (2.17), we differentiate equation (2.17) with respect to ΔQ and equate to zero to obtain

$$\left(\boldsymbol{J^T J} - \mu^2 \boldsymbol{I}\right)\Delta Q = \boldsymbol{J^T}\Delta P$$

The damped least square inverse (DLSI) $\boldsymbol{J}^*$can be defined as

$$\boldsymbol{J}^* = \left(\boldsymbol{J^T J} - \mu^2 \boldsymbol{I}\right)^{-1}\boldsymbol{J^T} = \boldsymbol{J^T}\left(\boldsymbol{J J^T} - \mu^2 \boldsymbol{I}\right)^{-1}$$

Thus ΔQ can be defined as

$$\Delta Q = \boldsymbol{J}^*\Delta P \tag{2.18}$$

By singular value decomposition, the DLSI is described as

$$\boldsymbol{J}^* = \boldsymbol{V\Sigma}^*\boldsymbol{U^T} = \sum_{i=1}^{r} \frac{\sigma_i}{\sigma_i^2 + \mu^2} v_i u_i^T \tag{2.19}$$

Various authors have suggested different ways to adjust the damping factor. (Nakamura and Hanafusa 1986) proposed adjusting the damping factor in terms of the manipulability index.

The manipulability index (W) was first introduced by (Yoshikawa 1985); it is the ability of a robot to change the end effector's orientation and position given the joint configuration. The larger the value of this index the greater the range of possible movements at that configuration. At singularities, the value of the manipulability index reduces to zero and the end effector loses one or more degrees of freedom.

The manipulability index can be computed by

$$\mathrm{W} = \sqrt{\det\left(\boldsymbol{J}(\boldsymbol{Q})\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{Q})\right)} = \sigma_1\sigma_2 \ldots \sigma_r \tag{2.20}$$

where σ_i: ith singular value of the Jacobian matrix where r is the rank of $\boldsymbol{J}$

The manipulability ellipsoid for an end effector velocity $\dot{P}$ satisfies

$$\|\dot{Q}\|^2 = \dot{Q}_1^2 + \dot{Q}_2^2 + \cdots + \dot{Q}_n^2 \leq 1$$

and its principal axes are $\sigma_1 u_1, \sigma_2 u_2, \ldots \sigma_r u_r$. The robot has the best mobility in the direction of the largest singular value which is the major axis of the manipulability ellipsoid. The minor axis represents the direction where the end effector has least mobility.

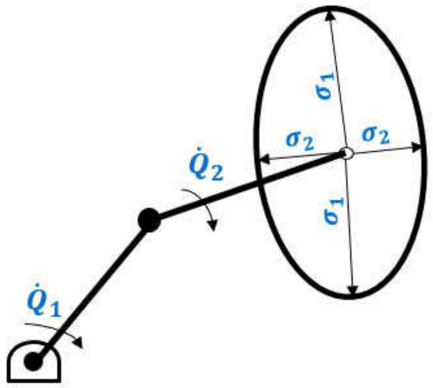


Figure 2.5 Axes of the manipulability ellipsoid

(Nakamura and Hanafusa 1986) proposed a method to adjust the damping factor using the manipulability index. Another method proposed by (Chiaverini, Egeland et al. 1991) adjust the manipulability index in terms of the minimum singular value (σ_{min}).

$$\mu^2 = \begin{cases} \mu_0^2(1 - \sigma_{\min}/\varepsilon)^2 & \text{otherwise} \\ 0 & \text{for } \sigma_{\min} \geq \varepsilon \end{cases}$$

where ε determines the limit of the singularity neighborhood, and μ_0 is the damping factor for $\sigma_{min} = 0$

The damped least square reaches a balance between robustness and accuracy of the desired solution. Since accuracy will often be sacrificed, a weighting matrix $\boldsymbol{\Psi}$ that distinguishes between end effector directions where higher accuracy is required and those where lower accuracy is accepted was introduced by (Chiaverini, Egeland et al. 1991). The pose increment ΔP in equation (2.6) can be rewritten as $\widetilde{\Delta P} = \boldsymbol{\Psi}\Delta P$, substituting this in equation (2.6)

$$\widetilde{\Delta P} = \tilde{\boldsymbol{J}}\,\Delta Q$$

where $\tilde{\boldsymbol{J}} = \boldsymbol{\Psi J}$

Another method developed by (Buss and Kim 2005)assigns a different damping factor to each singular value σ_i from the SVD. In this case, the damping factor not only depends on the current robot configurations but also on the error between the end effector's current and target poses. With this method, each joint is analyzed separately to know how much it is attempting to move the end effector to the target pose; this distance is then compared to the distance the end effector is to move to reach the target. In case the prior is larger than the latter, the motion of the joint is damped. The implementation of this algorithm involves limiting the maximum joint angle increment in each iteration.

2.4 Robot Workcell Calibration

Calibration is needed to reduce or eliminate robot errors without redesigning the robot or reprogramming the OLP. Robot errors can be divided into four classes: dynamic, geometric, system, and thermal errors (Greenway 2000). Dynamic errors come from structural resonance and inertial loadings resulting from robot motion. Geometric errors come from manufacturing tolerances, they affect the end effector orientation and position since the kinematic solutions will be computed with as designed values of the Denavit-Hartenberg (DH) parameters. Inaccuracies in relative orientation and position between the robot and its peripherals also are a source of geometric errors in robot systems. System errors originate from inaccurate calibration, wrong tuning of servos, and sensor inaccuracies. Thermal errors result from the expansion of the robot parts under heat.

Dynamic and thermal errors are hard to compensate for since they are highly nonlinear and cannot be easily modeled. Other errors such as sensor inaccuracies and dynamic errors are ignored as they do not have a considerable impact on the robot operation. Geometric errors are compensated for during calibration.

Model based calibration can be used to increase the robot manipulator's positioning accuracy by way of software such as DynaCal, Calibware, BullsEye, and VISOR Robotics. The kinematic calibration process involves four steps namely: establishing a kinematic model of the robot, measuring planned end effector poses, identifying the real robot kinematic parameters, and compensating for the kinematics parameters for an improved accuracy (Cheng 2008). Usually the Denavit-Hartenberg convention is used to develop the robot's kinematic model.

There are various techniques to calculate true end effector position and orientation if the joints configurations at different known points in a robot workspace are given (Hayati 1983). An external measuring system is required to accurately determine the robot position at these points. Researchers have used measuring systems such as theodolites (Duelen and Schröer 1991), laser tracker (Nubiola and Bonev 2013) and vision system (Du and Zhang 2013) among others.

To obtain the actual DH parameter an equation that relates the actual to the designed parameters has been developed. Given a vector $\rho = [\rho_1, \rho_2 \cdots, \rho_n]$ that represent the DH parameters for an n-joint robot, the actual link transformation ${}^{i\_1}_{i}\boldsymbol{A}$ can be expressed as

$${}^{i\_1}_{i}\boldsymbol{A} = {}^{i\_1}_{i}\boldsymbol{T} + \Delta\boldsymbol{T}_i$$

where $\Delta\boldsymbol{T_i} = \Delta\boldsymbol{T}_i(\Delta\boldsymbol{\rho}_i)$. Thus, the actual end effector transformation ${}^{0}_{n}A$

$${}^{\mathbf{0}}_{n}\boldsymbol{A} = {}^{\mathbf{0}}_{n}\boldsymbol{T} + \Delta\boldsymbol{T}$$

where $\Delta\boldsymbol{T} = \Delta\boldsymbol{T}(\boldsymbol{Q}, \Delta\boldsymbol{\rho})$, $\Delta\rho$ is the DH parameter error vector and $Q = [Q_1, Q_2 \cdots, Q_n]$ is the robot joints vector. ΔT is a highly nonlinear function (Cheng 2008), solved by nonlinear least square methods such as the Levenberg – Marquardt method, to generate actual DH parameters $\rho^* = \rho + \Delta\rho$. Since it is hard to compensate for each DH parameter, because usually the robot's control architecture is closed due to safety issues, compensations are made to the joint positions using inverse kinematics with the determined true DH parameters. Figure shows an offline compensation scheme.

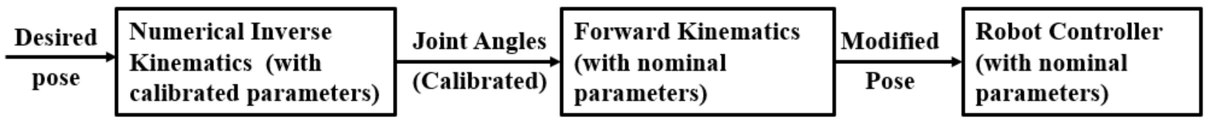

Desired pose
Numerical Inverse Kinematics (with calibrated parameters)
Joint Angles (Calibrated)
Forward Kinematics (with nominal parameters)
Modified Pose
Robot Controller (with nominal parameters)

Figure 2.6 Offline compensation scheme (Tao, Mustafa et al. 2015)

As mentioned above, other geometric errors originate from the relative orientation and position inaccuracies between the robot and its peripherals in the robot workcell. Two methods have been developed methods to minimize or eliminate these errors. The first method is to measure planned robot TCP positions in the real workcell using the teach pendant and subsequently change the corresponding nominal values in the virtual workcell. The second method involves using sensors to measure the actual orientation and positions of workcell elements. The sensor measurements are then utilized to update the offline programs (Lu and Lin 1997, Cheng 2008).

(Lu and Lin 1997) devised an online calibration method that automatically detects and compensates for relative orientation and position errors between the robot and its peripherals by using a vision system, a 3-dimensional force/torque sensor, together with control schemes that involve neural networks.

(Tao, Mustafa et al. 2015) proposed a method that computes the actual relative poses of the robot TCP and the work object frame in the sensor coordinate frame. A sensor fixed on the robot end effector is utilized to acquire the point cloud of the work object and subsequently compensate for the relative pose errors between the TCP and the work object frame. With accurate measurements, the robot TCP is directed to follow the desired robot path in the work object coordinate frame.

(Cai, Gu et al. 2018) developed a robot workcell calibration that utilizes a touch panel as a measurement equipment. A touch panel is fixed to the end effector and a touch panel is attached to work object. Once the touch probe comes in contact with the touch panel, the latter sends a signal to the robot controller by WIFI. The method achieves robot accuracy with a 0.25 mm error. The quality of the robot calibration depends on the resolution and measuring accuracy of the touch panel.

(Liu, Shen et al. 2008) devised a line-based calibration scheme that calibrates the transformation matrix relating the robot base frame to the work object frame. A position sensitive device is used to precisely capture a laser beam originating from a pointer attached to the end effector. Once numerous PSDs are installed on the work object, calibration points in both base and work object frames are generated. Subsequently a least square method and a quaternion algorithm are used to determine the transformation matrix relating the work object frame to the robot base frame.

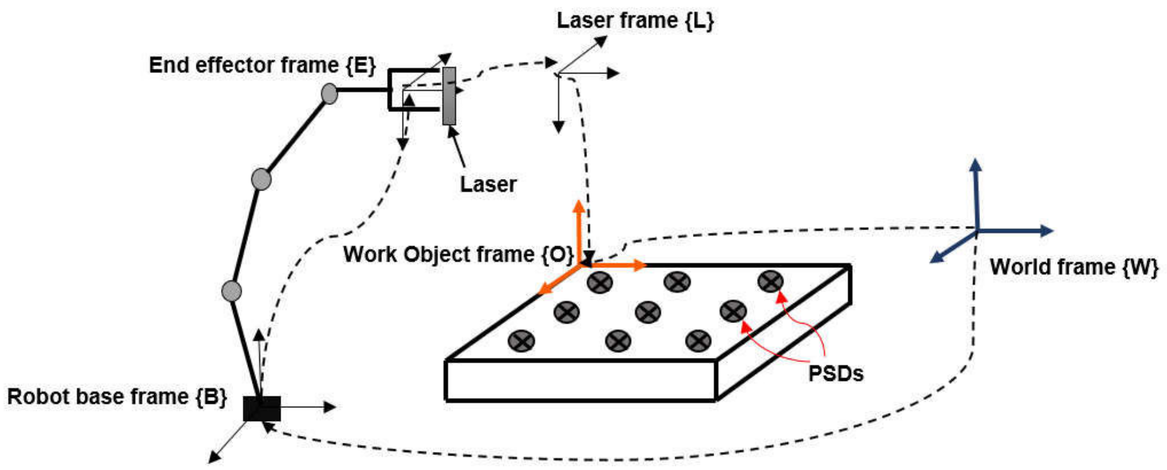


Figure 2.7 Coordinate frames of the calibration scheme

Table 2. 1 Robot workcell calibration methods

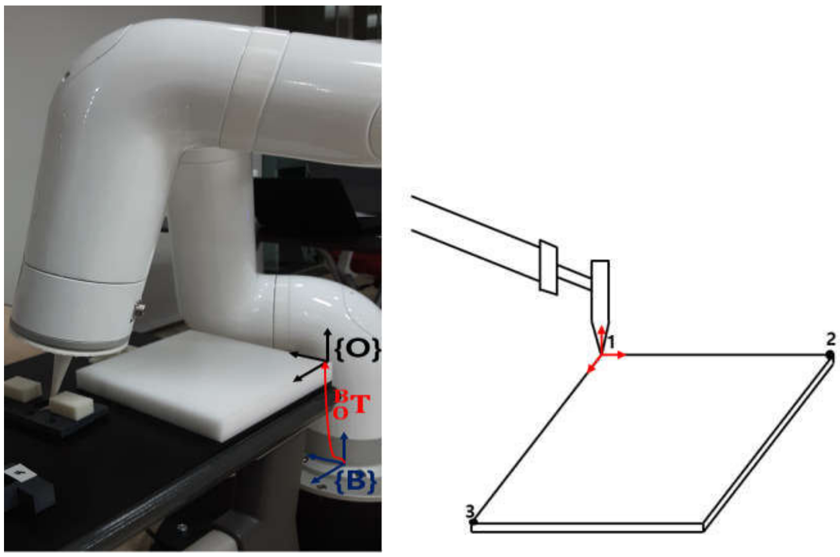
Author	**Measurement Technique**	**Compensation Technique**
(Tao, Mustafa et al. 2015)	Laser scanner, Stereo vision system (mounted on Robot)	Point cloud and least square method
(Gu, Li et al. 2015)	Line touch sensor (mounted on robot), cross beam sensor (mounted on work object)	Planar searching algorithm
(Ge, Gu et al. 2014)	Displacement sensor (mounted on robot), ball and cubes (mounted on work object)	Least square method for ball fitting, plane fitting
(Gan, Sun et al. 2004)	A coordinate measuring touch probe, a sphere	Nonlinear least square optimization
(Liu, Shen et al. 2008)	Laser pointer detector, position sensitive detector (PSD)	Plane fitting algorithm, quaternion based algorithm
(Arai, Maeda et al. 2002)	Charge-coupled device (CCD) camera	Direct linear transformation
(Cai, Gu et al. 2018)	Touch probe, touch panel	Linear least square optimization
(Lu and Lin 1997)	3D force/torque sensor, vision system	Neural networks
(De Smet 2015)	Laser emitter and laser receiver (mounted on the robot or work object)	
(Horváth and Erdős 2017)	Kinect sensor	Point cloud registration
(Schmidt and Wang 2014)	Camera, fiducial markers	Transformation matrix

CHAPTER 3

PATH REGENERATION

3.1 Transformation Matrix Computation

Once the robot manipulator is kinematically calibrated it can be utilized as a measurement tool for the calibration of the work object. Two sets of matching position data points are needed to get the transformation matrix of the work object relative to the robot base. The first set of points is measured with respect to the work object while the other set of points is measured with the robot and the values are read from the teaching pendant.



(a) (b)

Figure 3.1 (a)Transformation matrix of the real work object frame relative to the robot base frame, (b) Points measured in the work object frame

The work object frame is defined by measuring three points as shown in Figure 3.1, choosing point {1} as the origin of the work object frame the other two points are defined relative to this origin. The robot is then used to measure the same points in the base frame. Thus obtaining two data sets that will be used to compute the transformation matrix between the real work object frame and the robot base frame ${}_{\{O\_R\}}^{B}\boldsymbol{T}$.

${}_{\{O\_R\}}^{B}\boldsymbol{T}$ describes the orientation and position between the two frames.

$$ {}_{\{O\_R\}}^{B}\boldsymbol{T} = \begin{bmatrix} n_x & o_x & a_x & r_x \\ n_y & o_y & a_y & r_y \\ n_z & o_z & a_z & r_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.1} $$

A linear regression method applied to the matched data points is used to determine the transformation matrix ${}_{\{O\_R\}}^{B}\boldsymbol{T}$.

$$ \begin{bmatrix} x_B \\ y_B \\ z_B \\ 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & r_x \\ n_y & o_y & a_y & r_y \\ n_z & o_z & a_z & r_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_O \\ y_O \\ z_O \\ 1 \end{bmatrix} \tag{3.2} $$

The corresponding linear equations are

$$ \begin{gathered} x_B = n_x x_O + o_x y_O + a_x z_O + r_x \\ y_B = n_y x_O + o_y y_O + a_y z_O + r_y \\ z_B = n_z x_O + o_z y_O + a_z z_O + r_z \end{gathered} \tag{3.3} $$

For n pairs of matched data points, the square of residuals for the above equations can be determined by

$$ \begin{gathered} R^2 x_B = \sum_{i=1}^{n} [x_{Bi} - (n_x x_{Oi} + o_x y_{Oi} + a_x z_{Oi} + r_x)]^2 \\ R^2 y_B = \sum_{i=1}^{n} \left[y_{Bi} - \left(n_y x_{Oi} + o_y y_{Oi} + a_y z_{Oi} + r_y\right)\right]^2 \\ R^2 z_B = \sum_{i=1}^{n} [z_{Bi} - (n_z x_{Oi} + o_z y_{Oi} + a_z z_{Oi} + r_z)]^2 \end{gathered} \tag{3.4} $$

Since with linear regression we want to minimize the square of residual, the minimum can be obtained by equating the partial derivatives of equation (3.4) to zero.

$$ \frac{\partial R^2 x_B}{\partial n_x} = 0 = -2 \sum_{i=1}^{n} [x_{Bi} - (n_x x_{Oi} + o_x y_{Oi} + a_x z_{Oi} + r_x)] x_{Oi} $$

$$\frac{\partial R^2 x_B}{\partial o_x} = 0 = -2\sum_{i=1}^{n}[x_{Bi} - (n_x x_{Oi} + o_x y_{Oi} + a_x z_{Oi} + r_x)]y_{Oi}$$

$$\frac{\partial R^2 x_B}{\partial a_x} = 0 = -2\sum_{i=1}^{n}[x_{Bi} - (n_x x_{Oi} + o_x y_{Oi} + a_x z_{Oi} + r_x)]z_{Oi}$$

$$\frac{\partial R^2 x_B}{\partial r_x} = 0 = -2\sum_{i=1}^{n}[x_{Bi} - (n_x x_{Oi} + o_x y_{Oi} + a_x z_{Oi} + r_x)]$$

The above process is repeated for row two and three of equation (3.4). Putting the results into matrix form, the values of the transformation matrix can be expressed as

$$\begin{bmatrix} n_x \\ o_x \\ a_x \\ r_x \end{bmatrix} = [\boldsymbol{H}]^{-1} \begin{bmatrix} \sum_{i=1}^{n} x_{Bi} x_{Oi} \\ \sum_{i=1}^{n} x_{Bi} y_{Oi} \\ \sum_{i=1}^{n} x_{Bi} z_{Oi} \\ \sum_{i=1}^{n} x_{Bi} \end{bmatrix} \tag{3.5}$$

$$\begin{bmatrix} n_y \\ o_y \\ a_y \\ r_y \end{bmatrix} = [\boldsymbol{H}]^{-1} \begin{bmatrix} \sum_{i=1}^{n} y_{Bi} x_{Oi} \\ \sum_{i=1}^{n} y_{Bi} y_{Oi} \\ \sum_{i=1}^{n} y_{Bi} z_{Oi} \\ \sum_{i=1}^{n} y_{Bi} \end{bmatrix} \tag{3.6}$$

$$\begin{bmatrix} n_z \\ o_z \\ a_z \\ r_z \end{bmatrix} = [\boldsymbol{H}]^{-1} \begin{bmatrix} \sum_{i=1}^{n} z_{Bi} x_{Oi} \\ \sum_{i=1}^{n} z_{Bi} y_{Oi} \\ \sum_{i=1}^{n} z_{Bi} z_{Oi} \\ \sum_{i=1}^{n} z_{Bi} \end{bmatrix} \tag{3.7}$$

where

$$\boldsymbol{H} = \begin{bmatrix} \sum(x_{Oi}^2) & \sum(x_{Oi}y_{Oi}) & \sum(x_{Oi}z_{Oi}) & \sum(x_{Oi}) \\ \sum(x_{Oi}y_{Oi}) & \sum(y_{Oi}^2) & \sum(y_{Oi}z_{Oi}) & \sum(y_{Oi}) \\ \sum(x_{Oi}z_{Oi}) & \sum(y_{Oi}z_{Oi}) & \sum(z_{Oi}^2) & \sum(z_{Oi}) \\ \sum(x_{Oi}) & \sum(y_{Oi}) & \sum(z_{Oi}) & n \end{bmatrix}$$

The transformation matrix between the robot base frame and the virtual work object frame ${}^{\boldsymbol{B}}_{\{\boldsymbol{O\_V}\}}\boldsymbol{T}$ is obtained from the CAD model, to obtain the transformation matrix of the real work object relative to virtual work object frames ${}^{\{\boldsymbol{O\_R}\}}_{\{\boldsymbol{O\_V}\}}\boldsymbol{T}$ (in short ${}^{\boldsymbol{R}}_{\boldsymbol{V}}\boldsymbol{T}$) can be computed by compound transformations.

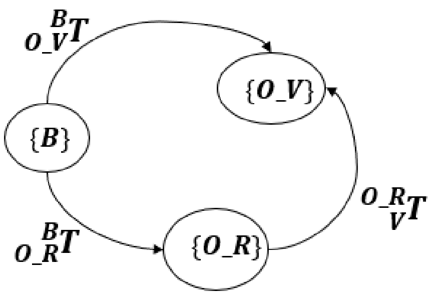


Figure 3.2 Compound transformations

$$ {}^{\boldsymbol{R}}_{\boldsymbol{V}}\boldsymbol{T} = \left({}^{\boldsymbol{B}}_{\boldsymbol{R}}\boldsymbol{T}\right)^{-1} {}^{\boldsymbol{B}}_{\boldsymbol{V}}\boldsymbol{T} $$

Thus, given a path in the virtual work object frame ${}^{V}\boldsymbol{PM}$, a corresponding path in the real work object ${}^{\boldsymbol{R}}\boldsymbol{PM}$ can be obtained by equation (3.8).

$$ {}^{R}\boldsymbol{PM} = {}^{R}_{V}\boldsymbol{T}^{V}\boldsymbol{PM} \tag{3.8}$$

3.2 Path Feasibility Check and Regeneration

Once the path in the real work object frame is determined, we need to check the feasibility of that path in terms of singularity. In case the path is not feasible, a new approximate path must be regenerated.

Given the joint configuration vector Q of a robot manipulator whose ith element Q_i is the joint angle for the ith joint of the robot manipulator, the forward kinematics finds the pose vector P that denotes the orientation and position of the effector. For an n joint robot manipulator, $Q = [\mathrm{Q_1}, \mathrm{Q_2}, \dots \mathrm{Q_n}]$. For a 6 degrees of freedom task the vector P has six elements: three for position and three for orientation $P = [\mathrm{r_x}, \mathrm{r_y}, \mathrm{r_z}, \mathrm{u}, \mathrm{v}, \mathrm{w}]$.

$$P = f(Q) \tag{3.9}$$

The inverse kinematics on the other hand finds the joint configuration required to reach a given target pose. Inverse kinematics equations are harder to derive than the forward kinematics as *f* is a highly nonlinear function.

$$Q = f^{-1}(P) \tag{3.10}$$

For a given end effector pose, there are more than one solutions for inverse kinematics.

Two types of solutions exist for inverse kinematics: closed-form (analytical) solutions and numerical solutions. Analytical solutions are unique to a given robot manipulator and they do not work for all robot manipulators, and as the number of joints increases the solutions become increasingly challenging to solve. For a six-joint manipulator, a closed form exists only when they are consecutive joints that either parallel or intersecting including wrist portioned robot manipulator such as the Puma 560 whose last three joint axes intersect at one point. Numerical solutions are general, and they can be applied to any type of robot manipulator. The drawbacks with these techniques are that they only give a single solution for a given initial condition when there may exist multiple solutions, and with an improper initial condition the method may fail to converge. The method in this section assumes a six joint robot manipulator with a six degrees of freedom task space and a six joint robot manipulator. The most common numerical method used is the Newton's method.

To derive the Newton's method, Taylor's expansion of equation (3.9) for a target pose P_t with initial condition Q_0 can be expressed as

$$P_t = f(Q_t) = f(Q_0) + \left.\frac{\partial f}{\partial Q}\right|_{Q_0} (Q_t - Q_0) + h.o.t$$

Letting the Jacobian matrix $\boldsymbol{J}(\boldsymbol{Q_0}) = \left.\frac{\partial f}{\partial Q}\right|_{Q_0}$ and dropping the higher order terms h.o.t

$$\Delta P = \boldsymbol{J}(\boldsymbol{Q_0})\Delta Q \tag{3.11}$$

where $\Delta Q = (Q_t - Q_0)$ and $\Delta P = P_t - f(Q_0)$

If the Jacobian matrix is nonsingular

$$\Delta Q = \boldsymbol{J^{-1}}(\boldsymbol{Q_0})\big(P_t - f(Q_0)\big) \tag{3.12}$$

Algorithm 1.1

```
1: Input target pose $\boldsymbol{P_t}$, initial condition $\boldsymbol{Q_0}$, maximum number of iterations M, set j = 0 and $\varepsilon > 0$
2: For $\mathbf{j = 1{:}M}$
3:     Define $\mathbf{e} = \boldsymbol{P_t} - \boldsymbol{f}(\boldsymbol{Q_0})$, While $\|\mathbf{e}\| > \boldsymbol{\varepsilon}$
4:     $\Delta\boldsymbol{Q} = \boldsymbol{J}^{-1}(\boldsymbol{Q_j})\mathbf{e}$
5:     Set $\boldsymbol{Q_{j+1}} = \boldsymbol{Q_j} + \Delta\boldsymbol{Q}$
6:     $\boldsymbol{Q_0} = \boldsymbol{Q_{j+1}}$
7: End For
```

If the Jacobian is singular i.e. $\det(\boldsymbol{J}) = 0$, equation (3.12) impossible. This is known as the robot manipulator's kinematic singularity, where the robot manipulator cannot produce motion in one or more degrees of freedom of the end effector. The robot manipulator in this case operates as if at least one degree of freedom has been lost.

The Levenberg Marquadt algorithm is used to derive an approximate solution to the Jacobian inverse that is called the damped least square solution (Nakamura and Hanafusa 1986), that allows the solution to equation (3.12) to become continuous. In this way a feasible and continuous solution is guaranteed at the cost of the robot end effector diverging from the desired path.

The damped least square solution satisfies the following optimization equation

$$\min_{\Delta Q} \|\Delta P - \boldsymbol{J}\Delta Q\|^2 + \mu^2 \|\Delta Q\|^2$$

where μ is the damping factor.

In the vicinity of a singularity, the robot manipulator exhibits a jerky motion because a small change in the end effector motion will produce large changes in joint angles. The damping factor serves as a means to reduce these joint angle changes by inducing minimal errors in the end effector motion.

Which yields the following solution

$$\Delta Q = \boldsymbol{J^T}\left(\boldsymbol{JJ^T} + \mu^2 \boldsymbol{I}\right)^{-1} \Delta P \tag{3.13}$$

The damped least square Jacobian $\boldsymbol{J}^*$ is expressed as

$$\boldsymbol{J}^* = \boldsymbol{J^T}\left(\boldsymbol{JJ^T} + \mu^2 \boldsymbol{I}\right)^{-1}$$

For a non-redundant robot manipulator equation (3.13) can be expressed as

$$\Delta Q = (\boldsymbol{J} + \mu^2 \boldsymbol{I})^{-1} \Delta P \tag{3.14}$$

By singular value decomposition of the Jacobian

$$\boldsymbol{J} = \boldsymbol{U \Sigma V^T} = \sum_{\mathrm{i}=1}^{\mathrm{r}} \sigma_\mathrm{i} \mathrm{u}_\mathrm{i} \mathrm{v}_\mathrm{i}^\mathrm{T} \tag{3.15}$$

where $\mathbf{\Sigma}$ is the diagonal matrix of singular values σ_i , $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices, r is the rank of the Jacobian.

By singular value decomposition of the damped least square Jacobian

$$\boldsymbol{J}^* = \boldsymbol{V \Sigma^* U^T} = \sum_{\mathrm{i}=1}^{\mathrm{r}} \frac{\sigma_\mathrm{i}}{\sigma_\mathrm{i}^2 + \mu^2} \mathrm{v}_\mathrm{i} \mathrm{u}_\mathrm{i}^\mathrm{T}$$

The damping factor allows the continuity of the inverse kinematics solution, it can be tuned with respect to the manipulability index of the robot manipulator.

The workspace of a robot manipulator is the set of all poses that are reachable by a specific end effector attached to that robot. The workspace of a robot depends on its number of degrees of freedom, its geometry, i.e. the link length, and the joint motion constraints. For a six-axis robot manipulator, the workspace is a six-dimensional body (Lenarcic, Bajd et al. 2012). If the Jacobian matrix has full rank, the manipulability index will have the same dimension as the workspace.

3.2.1 Manipulability Index

The manipulability index is a value that quantifies the ability of a robot manipulator to change the end effector's orientation and position given the joint configuration. The larger the value of this index the greater the range of possible movements at that configuration. At singularities, the value of the manipulability index reduces to zero and the end effector loses one or more degrees of freedom.

When the joint increments have the same length in all directions, in other words when the vector δQ lies on the surface of an n-dimensional hypersphere

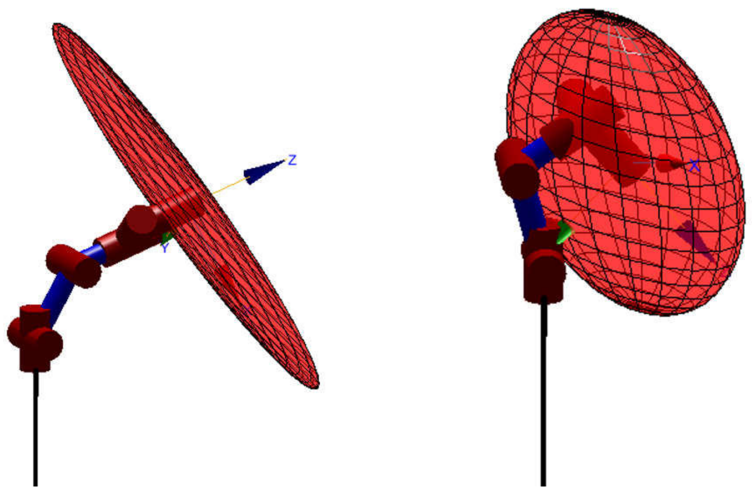
$$\Delta Q^T \Delta Q = 1$$

the end effector increment vector ΔP then lies on the surface of an ellipsoid of m-dimensional end effector coordinate space called the manipulability ellipsoid. If the ellipsoid has an almost spherical shape, i.e. it has uniform radii, then the end effector can move in any cartesian direction. If on the other hand one or more radii have a very small magnitude, it means the end effector cannot move in the directions with the small radii. The radii of the ellipsoid are the singular values of the Jacobian matrix

The manipulability index $W(Q)$ can be computed

$$\mathrm{W(Q)} = \sqrt{det(\boldsymbol{J}(\boldsymbol{Q})\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{Q}))} = \sigma_1 \sigma_2 \cdots \sigma_n \tag{3.16}$$

Figure 3.3 shows examples of manipulability index and manipulability ellipsoid for two joint configurations (a) $Q = [-26, -31 - 41, -36, 0, 0]$ and (b) $Q = [0, 15, -70, 175, 86, 63]$ (in degrees) for the Indy 7. In (a) the ellipsoid's radii are not uniform while in (b) the ellipsoid's radii are more uniform which is why the manipulability index in this case is larger than in (a).



(a) (b)

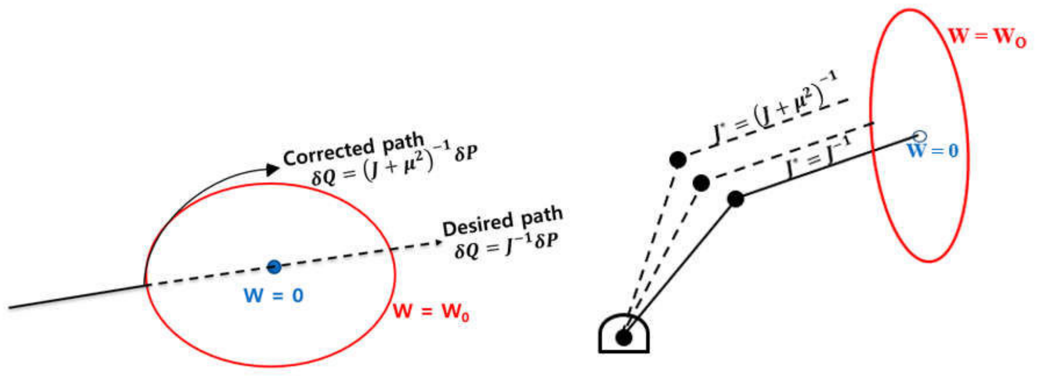
Figure 3.3 Manipulability ellipsoid (a) W(Q) = 0.006 and (b) W(Q) = 0.0248 (Corke 2011)

The damping factor is adjusted with respect to the manipulability index

$$\mu^2 = \begin{cases} {\mu_0}^2(1 - \mathrm{W}/\mathrm{W}_0)^2 & \text{for } \mathrm{W} < \mathrm{W}_0 \\ 0 & \text{for } \mathrm{W} \geq \mathrm{W}_0 \end{cases} \tag{3.17}$$

where μ_0 is the damping factor at singular points, and W_0 is the limit of manipulability index in the vicinity of singularities. These two values depend on the robot geometry and they are user-defined.

As mentioned above the damped least square offers a continuous solution at the cost the end effector deviating from the desired path. Figure 3.4 shows how the damping factor modifies the path and the robot configuration in a singular region. In (a) the robot path deviates from the desired path due to the damping factor and in (b) the robot manipulator moves away from the singular region because of the damping factor.


Corrected path
$\delta Q = (J + \mu^2)^{-1}\delta P$
Desired path
$\delta Q = J^{-1}\delta P$
W = 0
$\mathrm{W} = \mathrm{W}_0$
$J^* = (J + \mu^2)^{-1}$
$J^* = J^{-1}$
$\mathrm{W} = \mathrm{W}_0$
W = 0

(a) (b)

Figure 3.4 Change in (a) desired path and (b) robot joint configuration, due to the damping factor

3.2.2 Closed Loop Inverse Kinematics

The numerical integration in algorithm 1.1 introduces a tracking error in the inverse kinematics solution. The closed loop inverse kinematics algorithm introduces a feedback gain that is multiplied to the error in order to eliminate the tracking error. The tracking error decreases as feedback gain increases, but there is a limit to this increase. In case the feedback gain surpasses this limit, the system becomes unstable.

Using the damped least square Jacobian inverse $\boldsymbol{J}^*$ in equation (3.12) ΔQ can be expressed as

$$\Delta Q = \boldsymbol{J}^*(\boldsymbol{Q_0})\mathrm{K}\big(P_t - f(Q_0)\big) \tag{3.18}$$

where K is the feedback gain

In algorithm 1.1, step 4 then becomes

$$\Delta Q = \boldsymbol{J}^*(\boldsymbol{Q_0})\mathrm{K}\varepsilon \tag{3.19}$$

Figure 3.5 shows the loop for the inverse kinematics solution

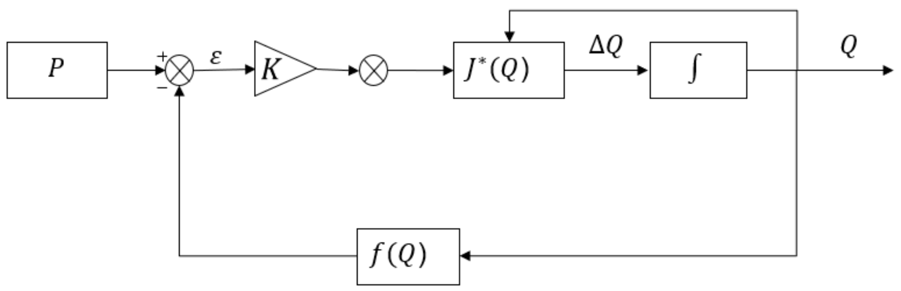


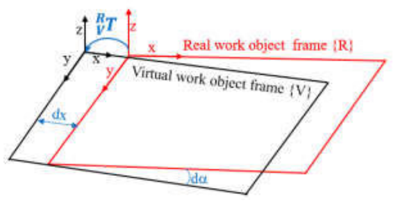
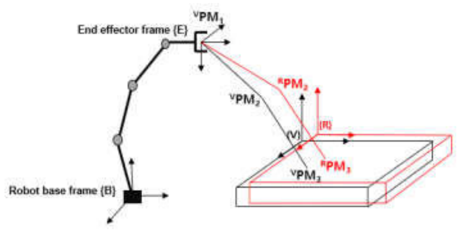
Figure 3.5 Closed loop inverse kinematics

From (Falco and Natale 2011) a limit to the feedback gain was proposed that allows the algorithm to remain stable

$$K = \frac{2}{T_s}$$

where T_s is the sampling time.

3.3 Summary of the Method

Step I: Path transformation	**Input:** • Original path (virtual work object frame): ${}^{V}\boldsymbol{PM}$ • Error compensation matrix: $({}^{R}_{V}\boldsymbol{T})$  ${}^{R}_{V}T$ z y x Real work object frame {R} Virtual work object frame {V} dx dα  End effector frame {E} ${}^{V}PM_1$ ${}^{V}PM_2$ ${}^{R}PM_2$ {R} {V} ${}^{R}PM_3$ ${}^{V}PM_3$ Robot base frame {B} **Output:** New path matrix (real workbench frame): ${}^{R}\boldsymbol{PM} = {}^{R}_{V}\boldsymbol{T}\,{}^{V}\boldsymbol{PM}$
Step II: Feasibility check and path regeneration	**Input:** • ${}^{R}\boldsymbol{PM}$**:** new path matrix • Q_1**:** joint angle vector at the first path point $({}^{R}PM_{1,:})$ • **S** ← f **(**Q_1**):** forward kinematics for Q_1 • Extract pose (position + orientation) vector P from pose matrix S (transform the rotation matrix into roll, pitch, and yaw angles) Rotation matrix

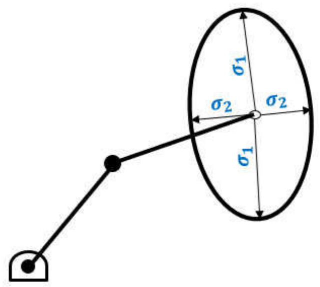
$$S = \begin{bmatrix} \boxed{\begin{matrix} n_x & o_x & a_x \\ n_y & o_y & a_y \\ n_z & o_z & a_z \end{matrix}} & \boxed{\begin{matrix} x \\ y \\ z \end{matrix}} \\ \begin{matrix} 0 & \quad 0 & \quad 0 \end{matrix} & 1 \end{bmatrix}$$

Position vector

$$P = [x, y, z, u, v, w]^T$$

- **J ($\boldsymbol{Q_1}$,):** Jacobian matrix at joint configuration $Q_{1,:}$
- $W = \sqrt{\det(\boldsymbol{J}(\boldsymbol{Q_i})\boldsymbol{J}^T(\boldsymbol{Q_i}))} = \sigma_1\sigma_2 \ldots \sigma_r$

  where σ_i: ith singular value of the Jacobian matrix where r = rank of $\boldsymbol{J}$
- W_0: limit of manipulability index in the vicinity of singularities



Manipulability ellipsoid that allows to visualize how close a robot is to being singular. In a singular position the singular values are zero.

Algorithm 1: Path execution

Input: $^R\boldsymbol{PM}$ (new path matrix by transformation matrix), Q_1 (joint angles vector at first path point ($^RPM_{1,:}$)),

```
1: for i = 1 : N do
2:   while j < M do
3:      compute the forward kinematics matrix $\boldsymbol{S_j} \leftarrow f(Q_i)$
4:      extract the pose (position + orientation) vector $P_{i+1}^c$ from pose matrix $\boldsymbol{S_j}$
5:      compute the pose difference $\Delta P \leftarrow P_{i+1} - P_{i+1}^c$
6:      $\boldsymbol{J}^*(\boldsymbol{Q_{j,:}}) \leftarrow$ Jacobian_matrix
7:      compute the joint configuration increment $\Delta Q \leftarrow \boldsymbol{J}^*(\boldsymbol{Q_i})\Delta P$
8:      compute the next joint configuration $Q_{i+1}^c \leftarrow Q_i + \Delta Q$
9:      compute the forward kinematics matrix $\boldsymbol{S_{j+1}} \leftarrow f(Q_{i+1}^c)$
10:     extract the pose (position + orientation) vector $P_{i+1}^c$ from pose matrix $\boldsymbol{S_{j+1}}$
11:     update $\Delta P \leftarrow P_{i+1} - P_{i+1}^c$
12:     if $\Delta \boldsymbol{P} < \boldsymbol{\varepsilon}$
13:       break
14:     else j ← j + 1
```

```
15:       end if
16:   end while
17: end for

Algorithm 2: Jacobian_matrix
Input: $Q_{j,:}$ (Joint angles configuration at instance j)
1: compute the Jacobian matrix at iteration j ← $\boldsymbol{J}\left(\boldsymbol{Q}_j\right)$
2:  compute the manipulability index  W ← $\sqrt{\det\left(\boldsymbol{J}(\boldsymbol{Q}_j)\boldsymbol{J}^{\boldsymbol{T}}(\boldsymbol{Q}_j)\right)}$
3: if $\mathbf{W} < \mathbf{W_0}$
4:   compute damping factor $\mu^2 \leftarrow {\mu_0}^2\left(1-\frac{W}{W_0}\right)^2$
5: else $\boldsymbol{\mu} \leftarrow \mathbf{0}$
6: end if
6: $\boldsymbol{J}^*(\boldsymbol{Q}_{j,:}) \leftarrow \boldsymbol{J}^{\boldsymbol{T}}\left(\boldsymbol{J}\boldsymbol{J}^{\boldsymbol{T}}-\mu^2\boldsymbol{I}\right)^{-1}$
```

3.4 Simulink Model

The above algorithm was implemented in MATLAB/SIMULINK. The closed loop inverse kinematics algorithm was implemented using SIMULINK as shown in Figure 3.6.

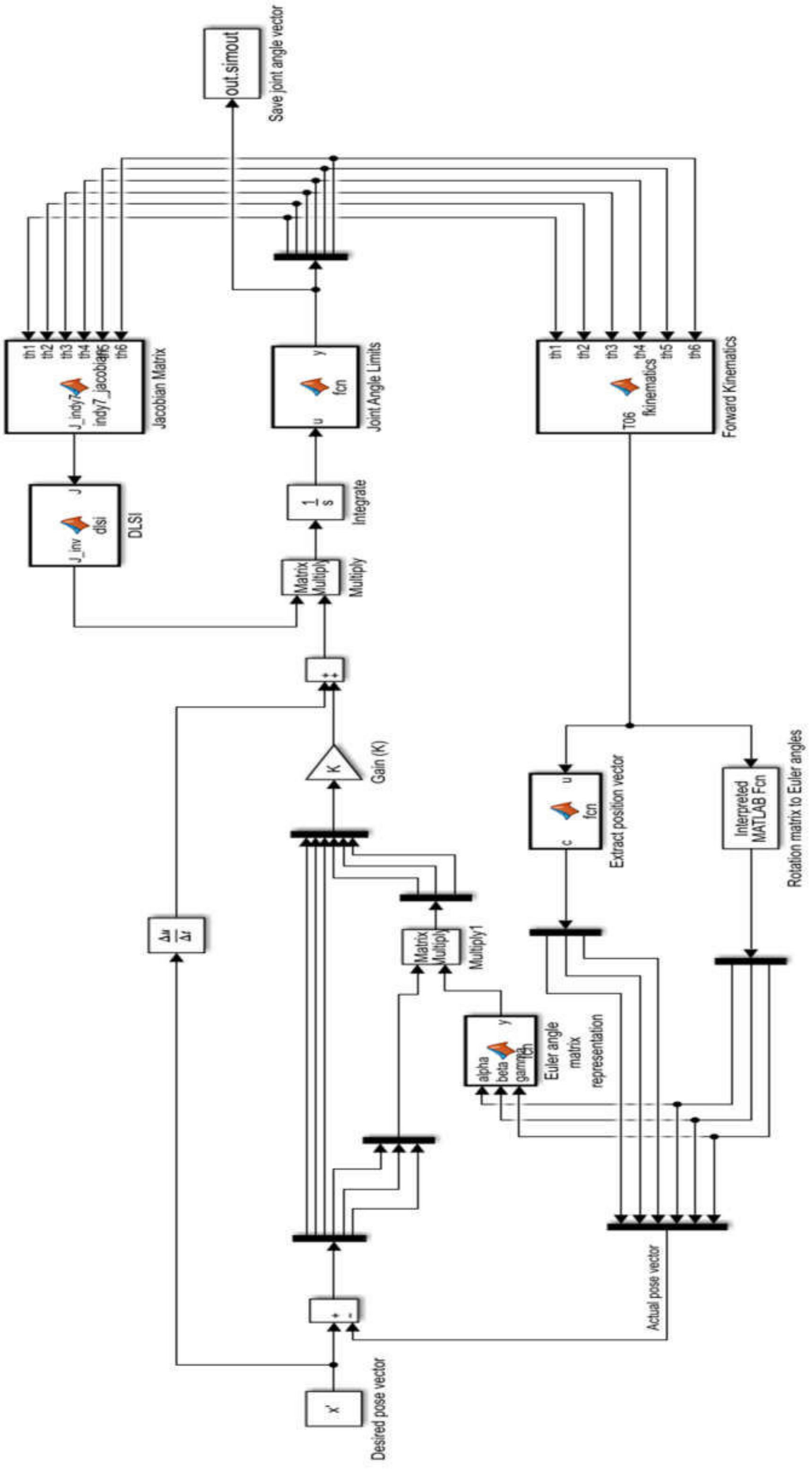


Figure 3.6 Implementation of the algorithm in SIMULINK

3.5 Example

This example provides simulation experiments to demonstrate how the above algorithm works. Given an initial path in the virtual frame to move the cube (purple) from pose 1 to pose 2 (in Figure 3.7). ${}^{V}\boldsymbol{PM}$ path in the virtual frame (position in m).

	X (m)	Y (m)	Z (m)
1	0.65	-0.3	0.2
2	0.65	-0.253	0.308
3	0.362	-0.544	0.14
4	0.055	-0.434	-0.148

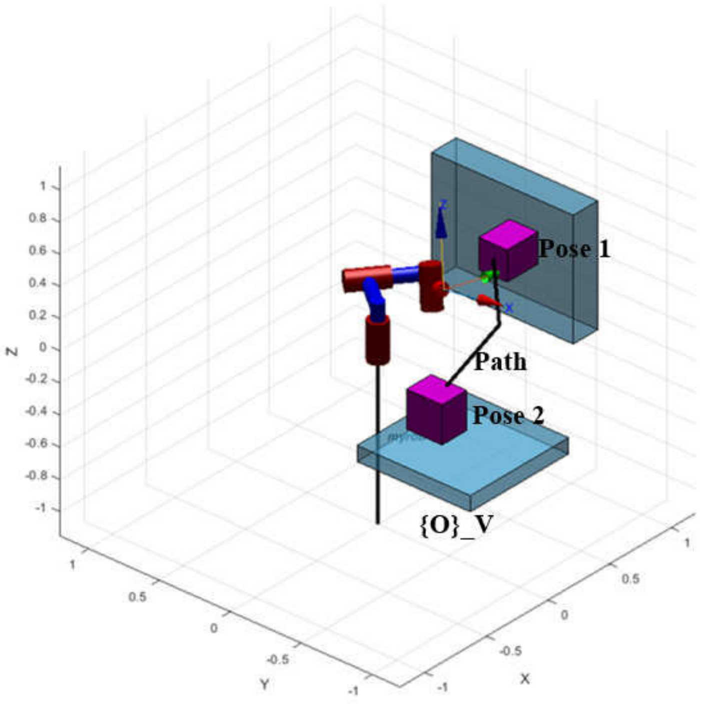


Figure 3.7 Robot workcell in virtual frame

Assuming a transformation matrix ${}^{R}_{V}\boldsymbol{T}$ that quantifies the dimensional variations between the virtual and real work object

$${}^{R}_{V}\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0.1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.15 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From equation we can obtain the path in the real frame (position in m)

	X (m)	Y (m)	Z (m)
1	0.53	-0.45	0.2
2	0.53	-0.403	0.308
3	0.242	-0.694	0.14
4	-0.065	-0.584	-0.148

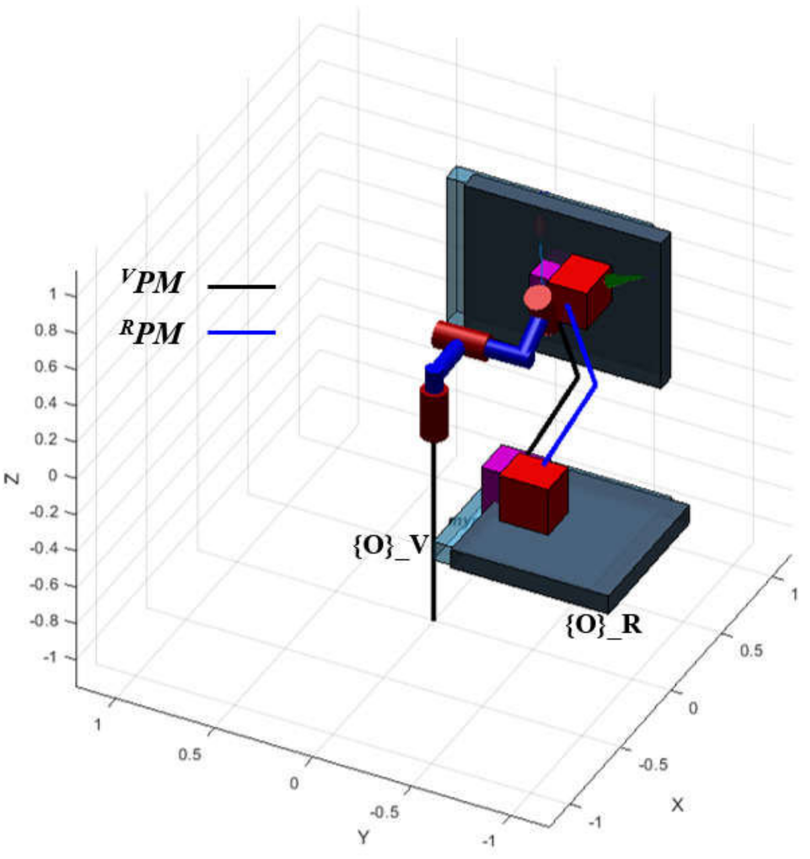


Figure 3.8 Robot workcell showing both the virtual and real frame

The limit for the manipulability index $W_o = 0.001$ and $\mu_o = 0.1$ were used to check the feasibility of the path in the real work object frame. Table 3.1 and 3.2 show joint angle values for $^{V}\boldsymbol{PM}$ and $^{R}\boldsymbol{PM}$**,** respectively. From Figure 3.9 the desired path $^{R}\boldsymbol{PM}$ and the regenerated feasible path are plotted. The error between the two paths is $|e| = 0.003$ and it is shown in Figure 3.10.

Table 3.1 Joint angles for ${}^{V}\boldsymbol{PM}$ (in degrees)

	Q_1	Q_2	Q_3
1	0	0	0
2	3.6	18	-3.7
3	-36.02	-10.86	-46.83
4	-41.7	-85.2	-7

Table 3.2 Joint angles for ${}^{R}\boldsymbol{PM}$ (in degrees)

	Q_1	Q_2	Q_3
1	-18.03	0	-21.17
2	-15.25	18.84	-32.35
3	-42	-10	29.3
4	-48.7	-57.5	-25.7

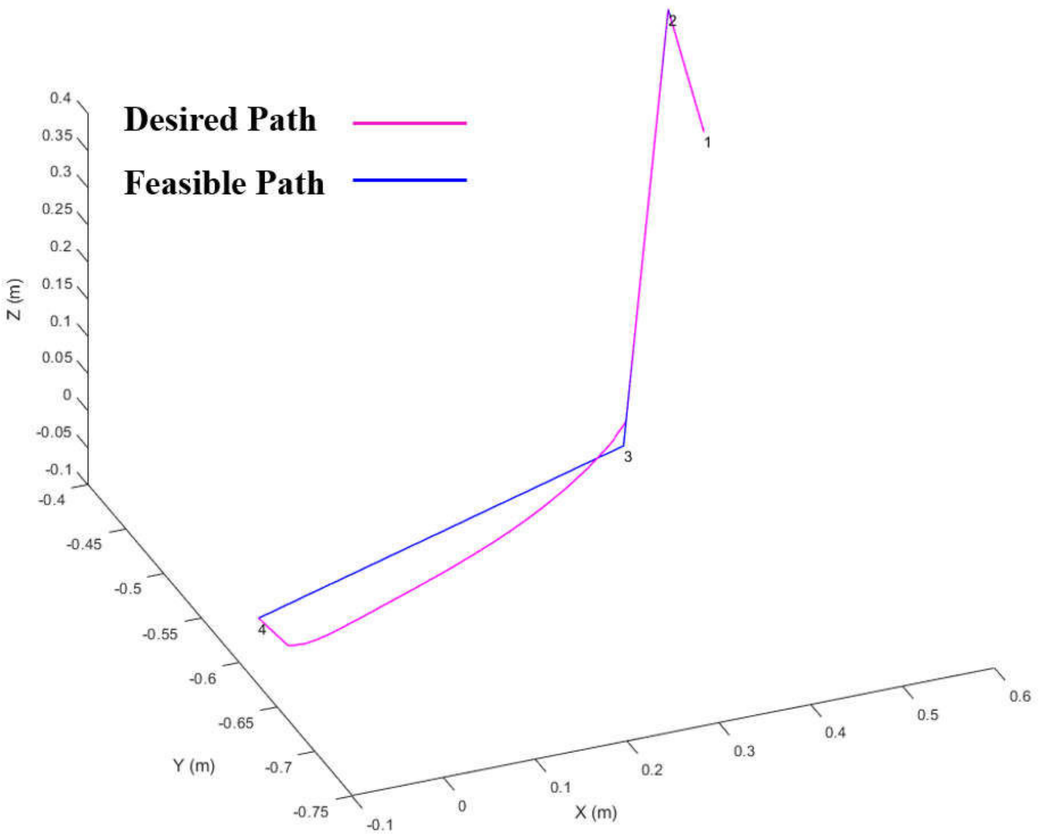


Figure 3.9 Desired robot path vs feasible robot path

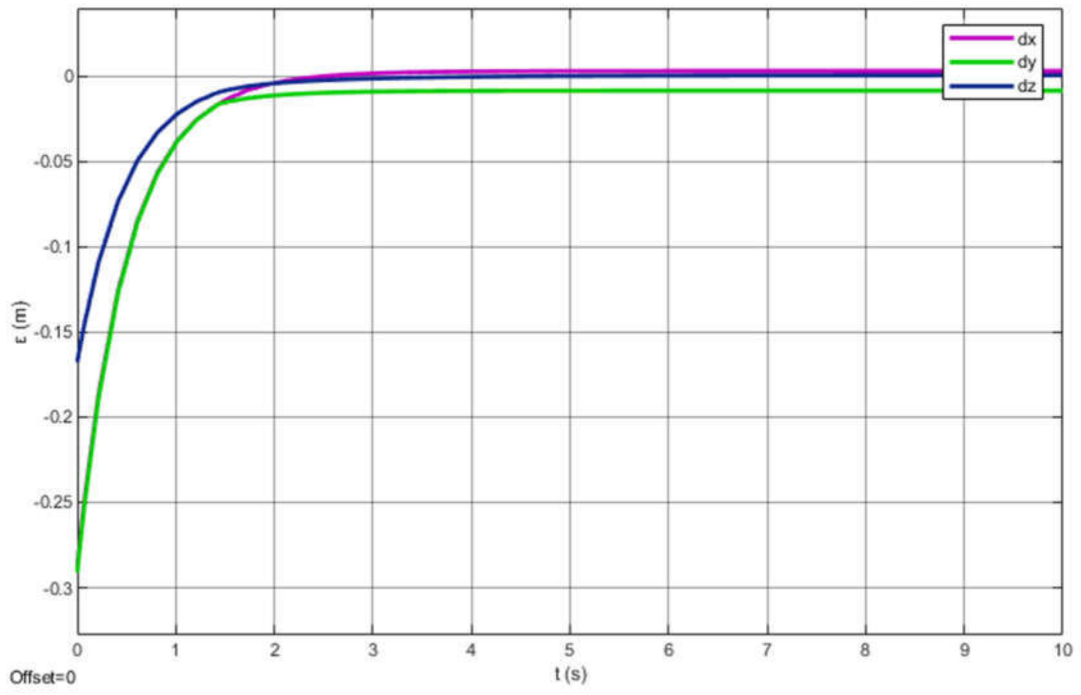


Figure 3.10 Error between the regenerated path and the desired path

CHAPTER 4

CASE STUDY

4.1 Experimental Setup

The Neuromeka Indy 7 whose modified Denavit Hartenberg parameters are shown in Figure 4.2 was used for the experiments. To determine the transformation matrix that relates the robot base to the work object frame two sets of data are acquired, one in the robot base frame using the robot teach pendant and another in the work object frame. Table 4.1 shows the two data sets used to obtain the transformation matrix that describes the work object frame with respect to the robot base frame ${}_{\boldsymbol{O}\_\{\boldsymbol{R}\}}^{\boldsymbol{B}}\boldsymbol{T}$, the points in the work object frame are measured with respect to the origin as shown in Figure 4.3 (a)**.** The TCP of the end effector shown in Figure 4.1 after calibration is 101 mm.

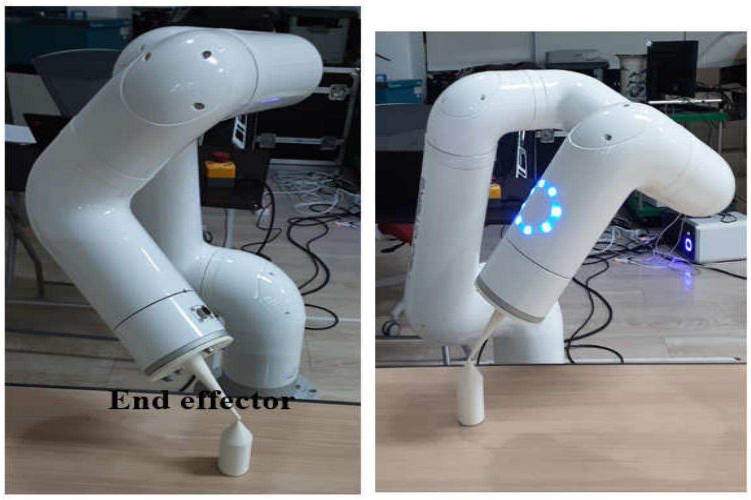


Figure 4.1 Indy 7 with end effector

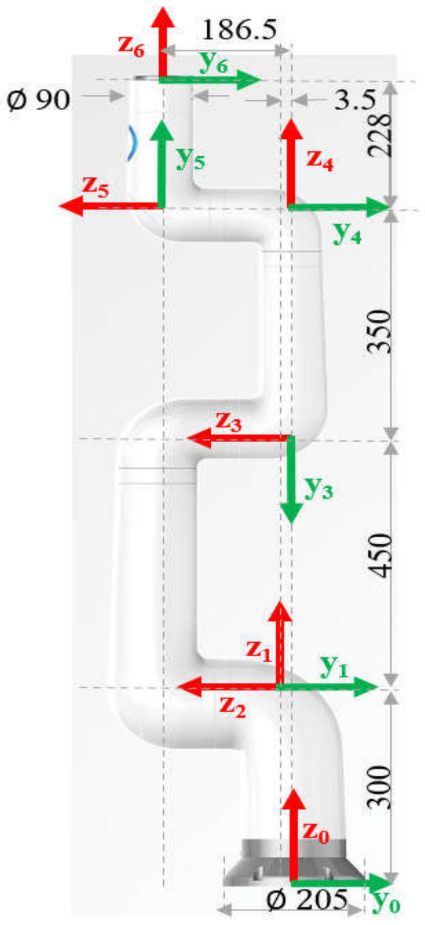


Figure 4.2 Indy 7 modified DH Parameters (dimension in mm and angles in degree)

Modified DH parameters

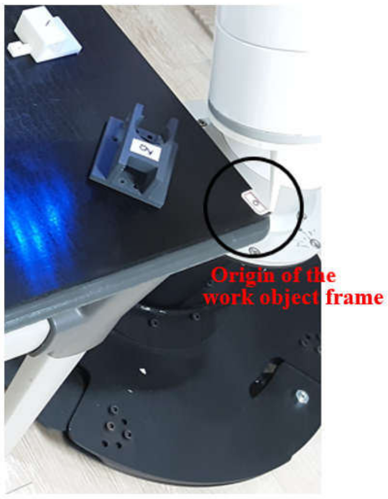
	a_{i+1} (mm)	α_{i+1} (°)	d_i (mm)	θ_i (°)	Joint range (°)
Joint #1	0	0	300	θ_1	-175 to 175
Joint #2	0	90	0	$\theta_2 + 90$	-175 to 175
Joint #3	450	0	3.5	$\theta_3 + 90$	-175 to 175
Joint #4	0	90	350	$\theta_4 + 180$	-175 to 175
Joint #5	0	90	183	θ_5	-175 to 175
Joint #6	0	-90	228	θ_6	-215 to 215

Table 4.1 Transformation matrix data points in mm

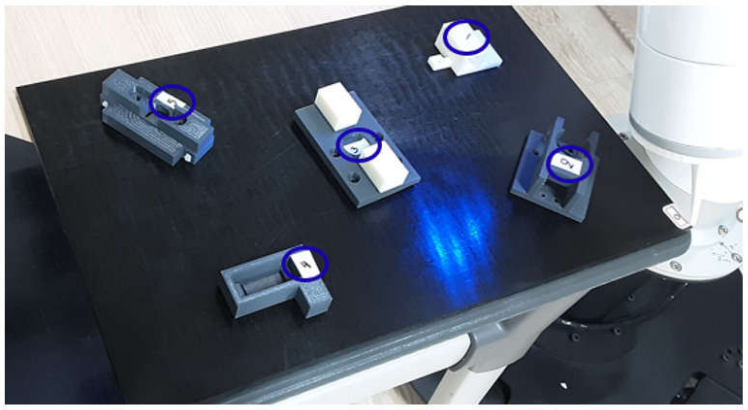
Robot base frame			Work object frame		
X (mm)	**Y (mm)**	**Z (mm)**	**X (mm)**	**Y (mm)**	**Z (mm)**
280.7	230.8	426.5	0	0	0
448.9	415.6	416.6	0	250	0
-15.3	499.3	408	400	0	0
280.9	229.6	405.8	0	0	20.7
405.86	438	440	75	187.5	22.4
310.5	334.7	448	98.7	45.6	26
367.2	469.2	439.35	193.5	130	25

The transformation matrix ${}_{O\_\{R\}}^{B}\boldsymbol{T}$ obtained

$$
{}_{O\_\{R\}}^{B}\boldsymbol{T} = \begin{bmatrix} -0.7411 & 0.6707 & -0.0301 & 281.05 \\ 0.6721 & 0.7407 & -0.0277 & 230.54 \\ 0.0021 & 0.0406 & 0.7849 & 411.99 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$



(a)



(b)

Figure 4.3 Robot work object measurement

To test the algorithm a robot task is shown in Figure 4.3 (b) and its path points in the virtual work object frame are listed in Table 4.2.

Table 4.2 Robot task path points in the virtual work object frame

	X (mm)	Y (mm)	Z (mm)	U (^{O})	V (^{O})	W (^{O})
1	595.95	131.7	1016.3	44	72.83	82.08
2	686.25	18.61	756.17	135.9	66.57	158.1
3	708.23	-151.97	596.92	163.32	54	168
4	592.68	-316.06	318.5	170	25	157.6
5	284.2	419.14	466.46	176.8	4.7	84.4
6	207.5	333.6	430.5	150	5	168
7	92.16	446	392.9	164	68.3	102

Given a transformation matrix that relates the robot base to the virtual work object frames ${}_{O\_\{V\}}^{B}\boldsymbol{T}$

$$ {}_{O\_\{V\}}^{B}\boldsymbol{T} = \begin{bmatrix} -0.7357 & 0.667 & -0.1180 & 277.31 \\ 0.6719 & 0.7413 & 0.0051 & 241.39 \\ -0.0663 & 0.0678 & 0.7802 & 413.65 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The transformation matrix that describes the virtual work object frame relative to the real work object frame ${}_{O\_\{V\}}^{O\_\{R\}}\boldsymbol{T}$ **(**or in short ${}_{V}^{R}\boldsymbol{T}$ **)**

$$ {}_{V}^{R}\boldsymbol{T} = \left({}_{O\{R\}}^{B}\boldsymbol{T}\right)^{-1} {}_{O\_\{V\}}^{B}\boldsymbol{T} $$

$$ {}_{V}^{R}\boldsymbol{T} = \begin{bmatrix} 0.9962 & 0.0030 & 0.0871 & 10.04 \\ 0 & 0.9994 & -0.0349 & 5.6 \\ -0.0872 & 0.0348 & 0.9956 & 1.8 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The path points in the real work object frame are listed in Table 4.3

Table 4.3 Robot task path points in the real work object frame

	X (mm)	Y (mm)	Z (mm)	U (^{O})	V (^{O})	W (^{O})
1	692.64	101.75	966.25	60.11	70.8	97.32
2	759.6	-2.2	695.47	150	67	-70.5
3	767.1	-167.1	529.07	162	48.7	167
4	627.24	-321.4	256.25	170	19.61	157.53
5	335.06	408.2	456	-178	3.2	84.7
6	255.3	324	424	149	0	168
7	137.4	437.6	400.4	174.6	65	111.6

Angles for the path in the real work object frame

	Q_1	Q_2	Q_3	Q_4	Q_5	Q_6
1	15.47	-27.05	-9.848	28.13	-46.88	0.09291
2	22.04	-61.63	-16.75	-72.11	-102.2	94.03
3	-1.63	-33.07	-51.07	12.91	-46.08	-1.351
4	-14.39	-107.5	82.17	9.872	-133.5	14.49
5	70.59	-21.2	-73.91	1.538	-88.56	165.9
6	78.76	9.843	-120.7	-0.3037	-38.21	91.15
7	103	-38.29	-85.83	-15.21	-119.3	168.7

Plotting the desired path and the feasible path in Figure 4.4 and the error between the feasible path and the desired path in

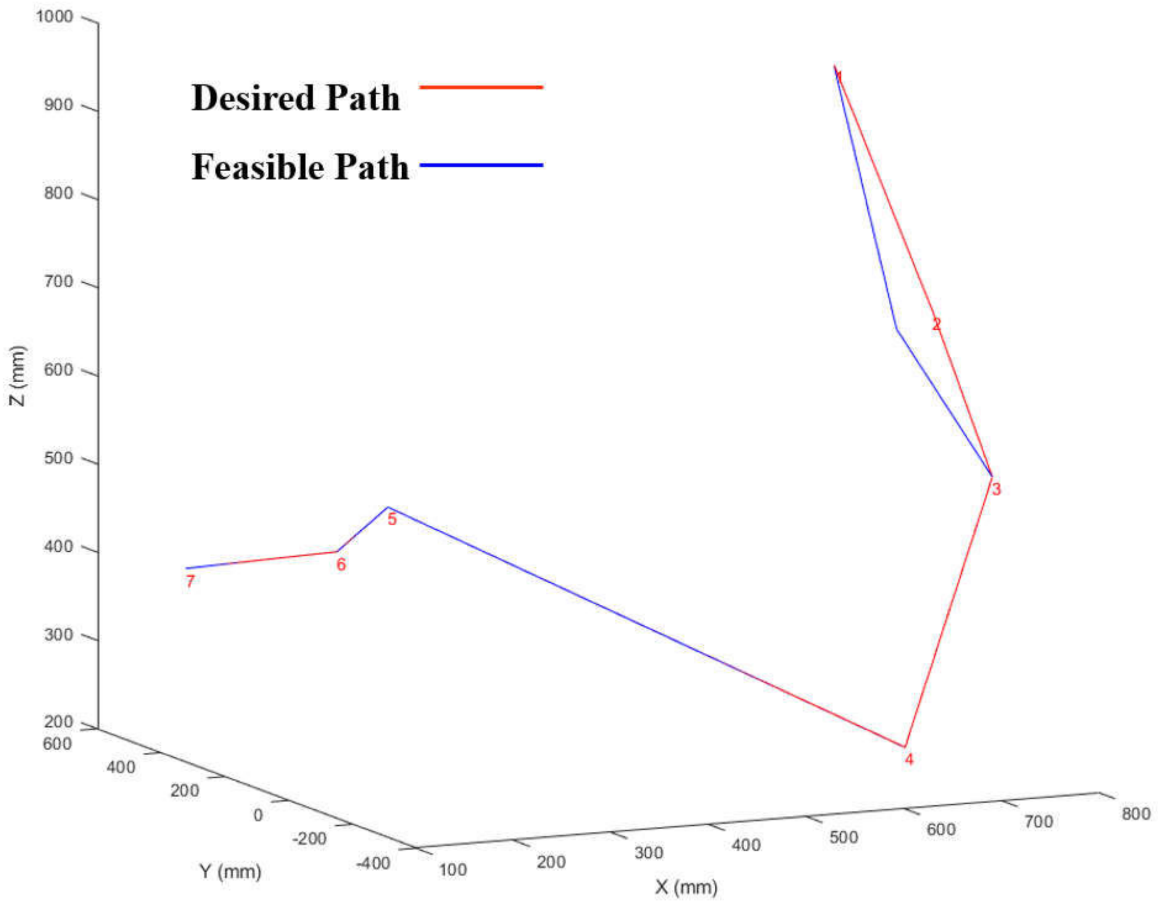


Figure 4.4 Desired robot path and feasible robot path

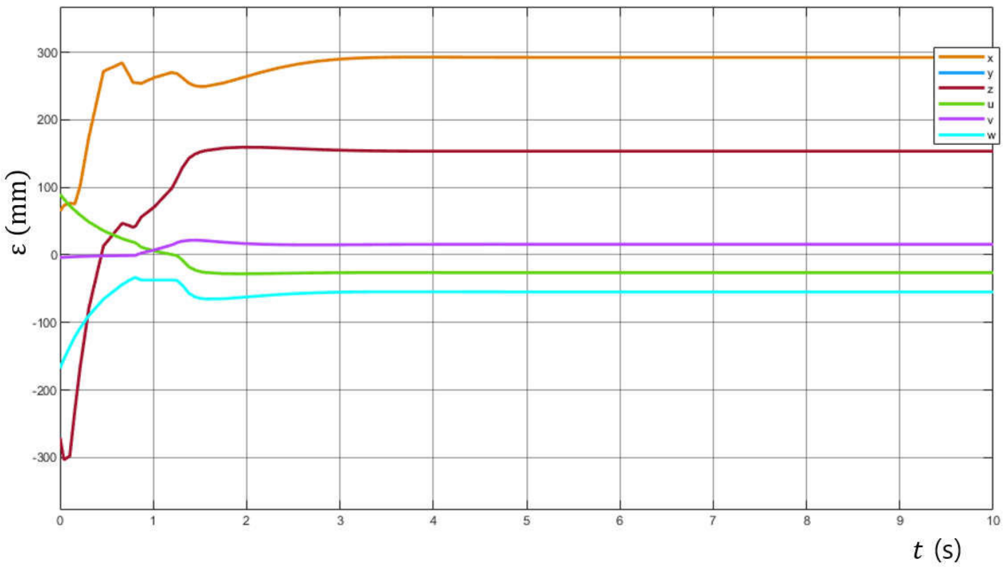


Figure 4.5 Error between the regenerated path and the desired path

4.2 Experiments

To test the feasibility check algorithm a path in the virtual frame is given for the Indy 7 robot manipulator, ${}^{V}\boldsymbol{PM}$.

$\mathrm{W}_0 = 0.01$ and $\mathrm{K}_0 = 0.1$

	X (mm)	Y (mm)	Z (mm)	U (^{O})	V (^{O})	W (^{O})
1	621.75	123.59	981.13	57.2	74.5	93.3
2	612.75	98.01	981.35	57.2	74.5	93.2
3	612.6	98	768.54	57.1	75	93.2
4	557.63	-102.72	621.85	-154	40.5	-125
5	493.63	-238.81	466.46	176.5	19.3	171
6	541.11	-329.5	307.13	-170	0	175
7	605.48	-490.13	98.6	-167	4.5	169

Test I: Transformation matrix ${}^{R}_{V}\boldsymbol{T}$

$${}^{R}_{V}T = \begin{bmatrix} 1 & 0 & 0 & 20 \\ 0 & 0.9848 & -0.1736 & 0 \\ 0 & 0.1736 & 0.9848 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The path in the real frame ${}^{R}\boldsymbol{PM}$

$$ {}^{R}\boldsymbol{PM} = {}^{R}_{V}\boldsymbol{T}\,{}^{V}\boldsymbol{PM} $$

	X (mm)	Y (mm)	Z (mm)	U (^{O})	V (^{O})	W (^{O})
1	621.75	123.59	981.13	57.2	74.5	93.3
2	632.75	-73.8886	983.4603	56	64.5	92
3	632.6	-36.9444	773.8817	56	64.5	92
4	577.63	-209.143	594.5656	-162.6	48.4	-131
5	513.63	-316.182	417.9045	166	17.4	168
6	561.11	-377.827	245.2469	-180	-0.87	175
7	625.48	-499.806	11.99186	-177	2.5	168.4

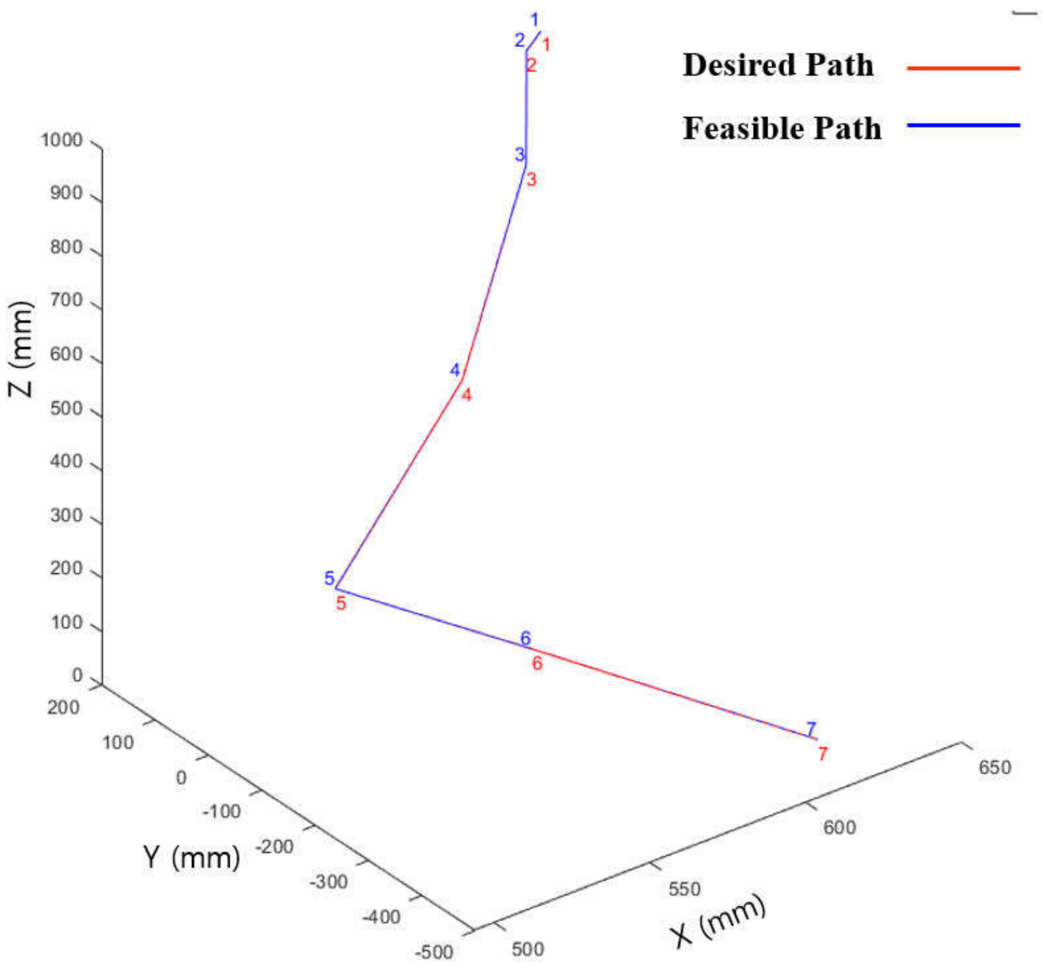


Figure 4.6 Desired robot path and feasible robot path

Since the initial path and the path execution coincide the path has no singular points. The approach vector length (6 to 7) is preserved.

Test II: Transformation matrix ${}^{R}_{V}\boldsymbol{T}$

$$ {}^{R}_{V}T = \begin{bmatrix} 1 & 0 & 0 & 20 \\ 0 & 0.866 & -0.5 & 0 \\ 0 & 0.5 & 0.866 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

The path in the real frame ${}^{R}\boldsymbol{PM}$

X (mm)	Y (mm)	Z (mm)	U (^{O})	V (^{O})	W (^{O})
621.75	123.59	981.13	57.2	74.5	93.3
612.8	-385.8	898.9	56	64.5	92
612.6	-279.4	714.6	56	64.5	92
557.6	-379.9	487.2	-162.6	48.4	-131
493.6	-419.6	284.8	166	17.4	168
541.1	-418.9	101.2	-180	-0.87	175
605.5	-453.8	-159.7	-177	2.5	168.4

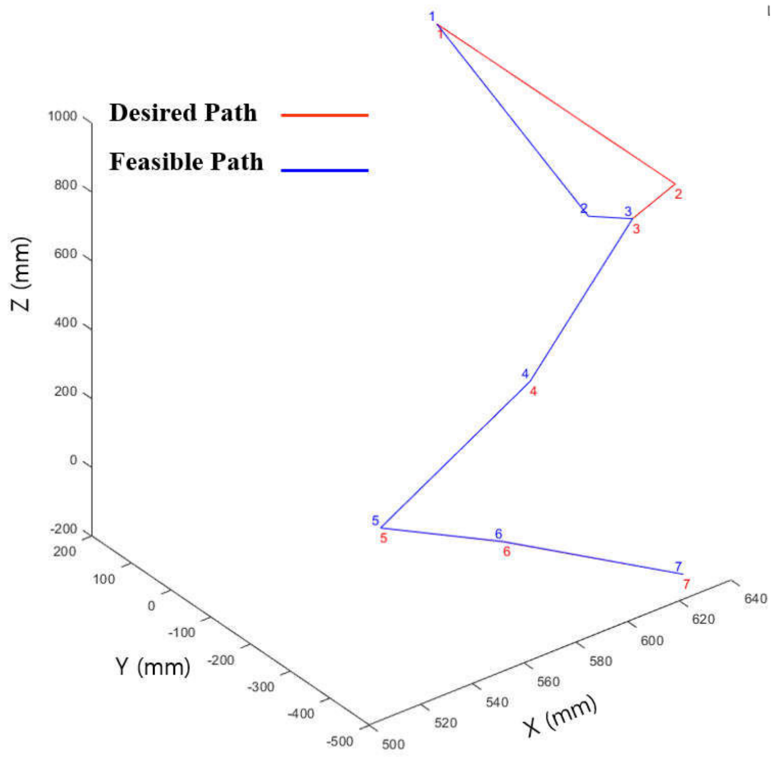


Figure 4.7 Desired robot path vs feasible robot path

The singular point in red (2) is approximated as shown in blue (2). The approach vector length (6 to 7) is preserved. The error between the desired and feasible path is plotted in Figure 4.8.

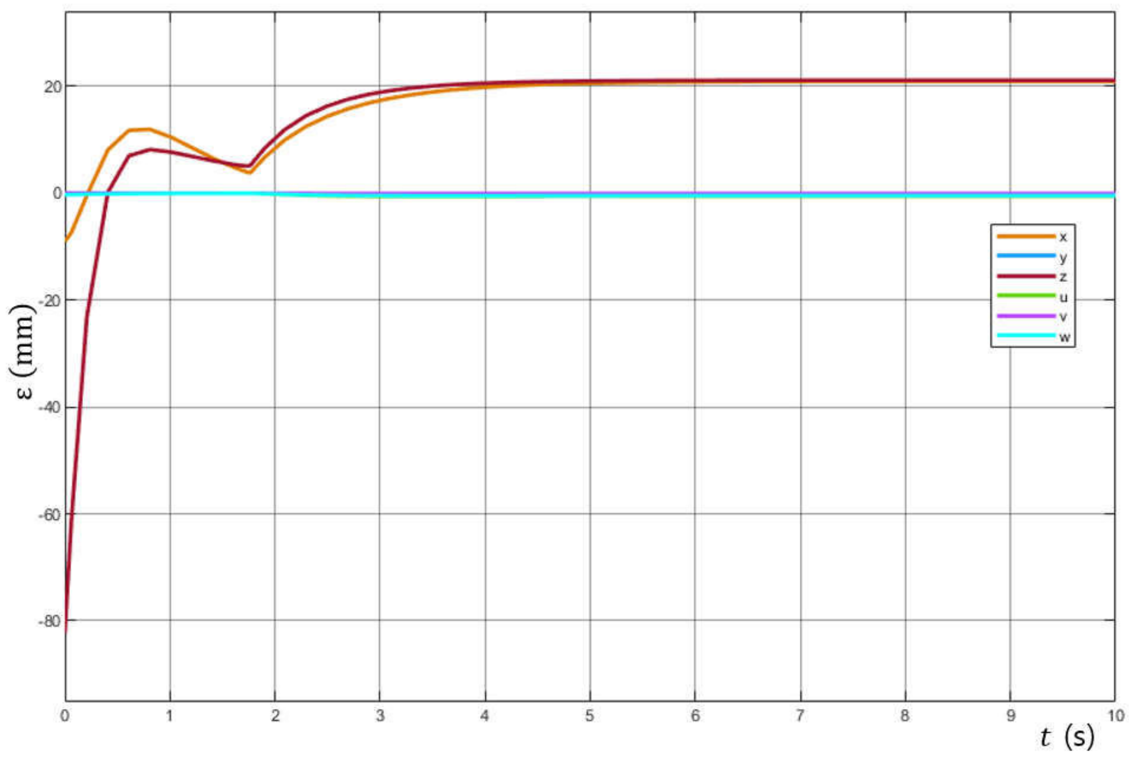


Figure 4.8 Error between the regenerated path and the desired path

CHAPTER 5 CONCLUSION AND FUTURE RESEARCH

5.1 Conclusion

Offline programming provides users with the ability to reprogram the robot without stopping the production process. Despite the popularity of offline programming, accuracy remains an issue in this field due to dimensional differences that occur in the real robot workcell, thus the need for calibrating the robot program. Calibration is required for the robot program in order to determine the real positions of the robot and its peripherals within the workcell. This process must be completed before the program is uploaded to the real system, since the dimensional variations between real and virtual workcells can result in critical TCP position errors.

In this thesis an adaptive robot program calibration method that accounts for the dimensional variations that occur in the real work object frame has been developed. The review of current workcell calibration methods includes methods that identify dimensional variations in the real work object either using the robot as a measurement tool or other measurement sensors such as laser and vision. To compensate for the dimensional variations, the true values are then loaded in the offline programming software to regenerate a new robot program.

The method developed in this thesis, comprises of two main steps, first we compute the transformation matrix between the robot base and the real work object frame in order to regenerate a robot program in the real work object frame. Then we check the feasibility of the regenerated path in terms of singularities and subsequently regenerate a new path in case the robot encounters a singular region while executing the desired path.

Since in this work, we considered a case of a six joint robot manipulator it is difficult to avoid singularities and still reach the target points. By using the manipulability index as a measure of how close the robot manipulator is to a singular region, the singularity can be avoided but at the cost of the end effector deviating from the desired path.

To prove the validity of the proposed method a six joint robot manipulator, the Neuromeka Indy 7 was used, and the algorithms were developed in MATLAB/SIMULINK. The experimental results showed that the end effector pose error introduced by the damping factor is not bounded, which may lead to the robot manipulator missing its target pose while avoiding a singularity.

5.2 Future Research

To avoid singularities, the closed loop inverse kinematics incorporates a damping factor that acts on the Jacobian in the vicinity of singularities. The damping factor in turn depends on two user defined factors namely: μ_0, the damping factor at singular points, and W_0 is the limit of manipulability index in the vicinity of singularities. Since the damping factor provides a compromise between singularity avoidance and minimizing errors in the end effector pose, it is imperative to have a proper method to determine μ_0 and W_0.

As mentioned in the conclusion, the error in the end effector pose caused by the damping factor is unbounded, which is a major setback to the developed method. A robot program calibration method that ensures that robot reaches its target poses without singularities is needed.

Real industrial robots' applications may include obstacles of the workspace of the robot manipulator. In future works a method to calibrate the robot program by incorporating obstacle avoidance in the path regeneration algorithm, will be studied.

REFERENCES

Arai, T., Y. Maeda, H. Kikuchi and M. J. C. A. Sugi (2002). "Automated calibration of robot coordinates for reconfigurable assembly systems." CIRP Annals **51**(1): 5-8.

Balestrino, A., G. De Maria and L. Sciavicco (1984). "Robust control of robotic manipulators." IFAC Proceedings Volumes **17**(2): 2435-2440.

Biggs, G. and B. MacDonald (2003). A survey of robot programming systems. Proceedings of the Australasian conference on robotics and automation: 1-3

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

Buss, S. R. and J.-S. Kim (2005). "Selectively damped least squares for inverse kinematics." Journal of Graphics tools **10**(3): 37-49.

Cai, Y., H. Gu, C. Li and H. Liu (2018). "Easy industrial robot cell coordinates calibration with touch panel." Robotics Computer-Integrated Manufacturing **50**: 276-285.

Cheng, F. S. (2008). "Calibration of robot reference frames for enhanced robot positioning accuracy." Robot Manipulators: 95-112.

Chiaverini, S., O. Egeland and R. Kanestrom (1991). Achieving user-defined accuracy with damped least-squares inverse kinematics. Fifth International Conference on Advanced Robotics' Robots in Unstructured Environments, IEEE: 672-677

Corke, P. (2011). "Robotics Vision and Control." Springer Tracts in Advanced Robotics **118**: 235-237.

Craig, J. J. (2009). Introduction to robotics: mechanics and control, 3/E, Pearson Education India: 65-67

De Smet, P. (2015). Robot-cell calibration, Google Patents.

Du, G. and P. Zhang (2013). "Online robot calibration based on vision measurement." Robotics and Computer-Integrated Manufacturing **29**(6): 484-492.

Duelen, G. and K. Schröer (1991). "Robot calibration—method and results." Robotics Computer-Integrated Manufacturing **8**(4): 223-231.

Falco, P. and C. Natale (2011). "On the stability of closed-loop inverse kinematics algorithms for redundant robots." IEEE Transactions on Robotics **27**(4): 780-784.

Gan, Z., Y. Sun and Q. Tang (2004). In-process relative robot workcell calibration, Google Patents.

Ge, J., H. Gu, L. Qi and Q. Li (2014). An automatic industrial robot cell calibration method. ISR/Robotik 2014; 41st International Symposium on Robotics, VDE.

Greenway, B. (2000). "Robot accuracy." Industrial Robot: An International Journal **27**(4): 257-265.

Gu, H., Q. Li and J. Li (2015). Quick robot cell calibration for small part assembly. The 14th IFToMM World Congress, Taipei.

Hayati, S. A. (1983). Robot arm geometric link parameter estimation. The 22nd IEEE Conference on Decision and Control, IEEE: 1477-1483

Horváth, G. and G. Erdős (2017). "Point cloud based robot cell calibration." CIRP Annals **66**(1): 145-148.

Khalaji, A. K. and S. A. A. Moosavian (2015). "Modified transpose Jacobian control of a tractor-trailer wheeled robot." Journal of Mechanical Science and Technology **29**(9): 3961-3969.

Khalil, H. K. and J. W. Grizzle (2002). Nonlinear systems, Prentice hall Upper Saddle River, NJ.

Klein, C. A. and C.-H. Huang (1983). "Review of pseudoinverse control for use with kinematically redundant manipulators." IEEE Transactions on Systems, Man, and Cybernetics(2): 245-250.

Lenarcic, J., T. Bajd and M. M. Stanišić (2012). Robot mechanisms, Springer Science & Business Media.

Liu, T., Y. Lei, L. Han, W. Xu and H. Zou (2016). "Coordinated resolved motion control of dual-arm manipulators with closed chain." International Journal of Advanced Robotic Systems **13**(3): 80.

Liu, Y., Y. Shen, N. Xi, R. Yang, X. Li, G. Zhang and T. A. Fuhlbrigge (2008). Rapid robot/workcell calibration using line-based approach. 2008 IEEE International Conference on Automation Science and Engineering, IEEE: 510-515

Lozano-Perez, T. (1983). "Robot programming." Proceedings of the IEEE **71**(7): 821-841.

Lu, T.-F. and G. Lin (1997). "An on-line relative position and orientation error calibration methodology for workcell robot operations." Robotics Computer-Integrated Manufacturing **13**(2): 89-99.

Manseur, R. (2007). Robot modeling and kinematics, Firewall Media.

Nakamura, Y. and H. Hanafusa (1986). "Inverse kinematic solutions with singularity robustness for robot manipulator control." Journal of Dynamic systems, Measurement, and Control **108**(3): 163-171.

Nakamura, Y., H. Hanafusa and T. Yoshikawa (1987). "Task-priority based redundancy control of robot manipulators." The International Journal of Robotics Research **6**(2): 3-15.

Nubiola, A. and I. A. Bonev (2013). "Absolute calibration of an ABB IRB 1600 robot using a laser tracker." Robotics and Computer-Integrated Manufacturing **29**(1): 236-245.

O'Neil, K. A., Y.-C. Chen and J. Seng (1997). "Removing singularities of resolved motion rate control of mechanisms, including self-motion." IEEE Transactions on Robotics Automation **13**(5): 741-751.

Pan, Z., J. Polden, N. Larkin, S. Van Duin and J. Norrish (2010). Recent progress on programming methods for industrial robots. ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics), VDE: 619-626

Schmidt, B. and L. Wang (2014). "Automatic work objects calibration via a global–local camera system." Robotics and Computer-Integrated Manufacturing **30**(6): 678-683.

Sciavicco, L. and B. Siciliano (2012). Modelling and control of robot manipulators, Springer Science & Business Media: 106-109

Šoch, M. and R. Lórencz (2005). "Solving inverse kinematics–a new approach to the extended Jacobian technique." Acta Polytechnica Journal of Advanced Engineering **45**(2): 22-26.

Tao, P., S. Mustafa, G. Yang and M. Tomizuka (2015). "Robot work cell calibration and error compensation." Handbook of Manufacturing Engineering Technology. London: Springer: 1995-2034.

Wampler, C. and L. Leifer (1988). "Applications of damped least-squares methods to resolved-rate and resolved-acceleration control of manipulators." Journal of Dynamic systems, Measurement, and Control **110**(1): 31-38.

Whitney, D. E. (1969). "Resolved motion rate control of manipulators and human prostheses." IEEE Transactions on man-machine systems **10**(2): 47-53.

Whitney, D. E. (1972). "The mathematics of coordinated control of prosthetic arms and manipulators." Journal of Dynamic systems, Measurement, and Control **94**(4): 303-309.

Yoshikawa, T. (1985). "Manipulability of robotic mechanisms." The International Journal of Robotics Research **4**(2): 3-9.

ACKNOWLEDGMENT

My deepest gratitude goes to Prof Duck Young Kim. Thank you for giving me the opportunity to learn and advance in my career. I have learnt a lot under your guidance. You have been very patient with me in my learning process, taught me how to be a good researcher, and most importantly taught me the importance of hard work and consistency. I believe that the past two years have been a very defining stage in my career as a researcher.

I would also like to express my sincere gratitude to Prof Namhun Kim and Prof Sang Hoon Kang for agreeing to be my defense committee members. Their insight and guidance were highly appreciated.

My sincere thanks to my Smart Factory laboratory colleagues. They have helped me navigate life in South Korea. They have been great friends and always there for me whenever I needed any help, whether in academic endeavors or with life in general. I have enjoyed our pizza and barbecue outings and all the time we spent together.

Last but not least I deeply thank my parents for their unending love and support in everything I do. My daughter Merveille you are a constant source of inspiration. To my siblings, you believe in me and always inspire me to be the best that I can be. And to Hana my best friend, I appreciate your love and support. This journey has indeed been much easier with you by my side.