ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

INVERSE DYNAMICS CONTROL OF A HUMANOID ROBOT ARM

M.Sc. THESIS

Oğuzhan CEBE

Department of Mechatronics Engineering

Mechatronics Engineering Programme

JANUARY 2017

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

INVERSE DYNAMICS CONTROL OF A HUMANOID ROBOT ARM

M.Sc. THESIS

Oğuzhan CEBE
518151019

Department of Mechatronics Engineering

Mechatronics Engineering Programme

Thesis Advisor: Prof. Dr. Şeniz ERTUĞRUL

JANUARY 2017

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

İNSANSI BİR ROBOT KOLUNUN TERS DİNAMİK KONTROLÜ

YÜKSEK LİSANS TEZİ

Oğuzhan CEBE
518151019

Mekatronik Mühendisliği Anabilim Dalı

Mekatronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Şeniz ERTUĞRUL

OCAK 2017

Oğuzhan CEBE, a M.Sc. student of ITU Graduate School of Science Engineering and Technology student ID 518151019, successfully defended the thesis entitled "INVERSE DYNAMICS CONTROL OF A HUMANOID ROBOT ARM", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Şeniz ERTUĞRUL** İstanbul Technical University	.....
Jury Members :	**Doç. Dr. Zeki Yağız BAYRAKTAROĞLU** İstanbul Technical University	.....
	Yard. Doç. Dr. Janset DAŞDEMİR Yıldız Technical University	.....

Date of Submission : 31 November 2016
Date of Defense : 2 January 2017

To my family,

FOREWORD

I thank to my advisor Şeniz Ertuğrul for her support throughout my thesis. I am grateful to Ömer Faruk Argın, Cihat Bora Yiğit, Gökçe Burak Tağlıoğlu and Ömür Baç for their guidance and enduring my endless questions. Finally, I thank to my family for always being there for me.

January 2017

Oğuzhan Cebe
Mechanical Engineer

TABLE OF CONTENTS

Page

ABBREVIATIONS

A: Transformation matrix

R: Rotation matrix

p: Translation vector

ϕ: Orientation vector

X: Pose vector

J: Jacobian matrix

J_P: Translational part of Jacobian matrix

J_O: Rotational part of Jacobian matrix

$\dot{p}$: Linear velocity

ω: Angular velocity

$\ddot{p}$: Linear acceleration

$\dot{\omega}$: Angular acceleration

B(q): Mass matrix

C(q, $\dot{q}$): Coriolis and Centrifugal terms

g(q): Gravitational terms

r(t): Trajectory timing function

D: Trajectory linear distance

$\boldsymbol{a}$: Resolved acceleration

y*:* Control input of task-space inverse dynamics controller

T: Orientation velocity to angular velocity transformation matrix

$\dot{\boldsymbol{J}}\dot{\boldsymbol{q}}$: Jacobian derivative term

LIST OF TABLES

LIST OF FIGURES

INVERSE DYNAMICS CONTROL OF A HUMANOID ROBOT ARM

SUMMARY

Nowadays, humanoid robot technology is studied extensively around the world. These humanoid robots can reach the places that wheeled robots cannot and can perform complicated tasks with their two arm manipulators.

Experimental studies are being conducted for humanoid robots. Interacting with environment, cooperation with humans and executing human-like motions for various tasks are the key objectives of these studies.

Control of the manipulators of humanoid robots require a dynamic model based control for fast movement cases. As the manipulator is supposed to move in a cluttered environment, task space control inverse dynamics control is a suitable control policy for this scenario, where the motion of the end-effector can be predicted during the execution of the desired trajectory.

The humanoid robots consist of a high number of actuators and sensors. To control the robot, sensor processing, motion planning and actuator control need to be done simultaneously. Thus, the software of these robots consist of multi-processes and scheduling to handle this problem.

ROS (Robot Operating System) is an open-source operating system that has software libraries and tools for such robotic applications. It offers both simulator and hardware interface, alongside state-of-art algorithms. A software that runs on ROS consists of multiple processes called ‘nodes’. Each node handles a different task, runs at a specified frequency and communicate with each other. This architecture eases the programming and enables use of open-source libraries in separate nodes in a plug-and-play way.

Gazebo is an open-source dynamic simulation environment that enables the simulation of many type of robots such as full body humanoids with various sensors and environment interaction. Gazebo has interface with several platforms, including ROS and it offers several dynamic engines and number of transmissions, but not all of them are supported by ROS at the moment.

ITECH Arm is a six degrees of freedom humanoid robot arm built in Mechanical Engineering Automatic Control Laboratory of İstanbul Technical University. The purpose of this thesis is creating a generic kinematics and dynamics library for the Automatic Control Laboratory, writing the software packages using this library for the ROS integration of the robot arm and finally implementing task space inverse dynamics control of ITECH Arm in Gazebo simulation environment.

This thesis consists of six chapters. In the first chapter, purpose of the thesis, literature review and mechanical-software properties of ITECH Arm manipulator will be mentioned.
In second chapter, kinematic and dynamic modelling of a robot manipulator is presented. The geometric, kinematic and dynamic models of ITECH Arm manipulator are derived.

In chapter three, Robot Operating System is introduced. The software architecture and capabilities of ROS are mentioned. Integration steps of ITECH Arm to ROS environment and interfacing ROS and Gazebo simulation environment are described.

In fourth chapter, trajectory generation for robot manipulators is mentioned. Several robot control methods are discussed. Implementation of task space trajectory tracking with inverse dynamics control algorithm on ITECH Arm is described.

In chapter five, simulation results of circular trajectory for with and without payload cases using various gain sets and controller frequencies are presented. Also, as an example task, a pick and place scenario results are appended. The simulation results are discussed.

In sixth and the last chapter, all the work done in the thesis is summarized and suggestions for future works are presented.

İNSANSI ROBOT KOLUNUN TERS DİNAMİK İLE KONTROLÜ

ÖZET

Günümüzde insansı robot teknolojisi dünyada yaygın olarak çalışılmaktadır. Bu insansı robotlar tekerlekli robotların ulaşamayacağı yerlere ulaşabilmekte ve iki kolları ile karmaşık görevleri yerine getirebilmektedir. Bu özellikleri, onları arama kurtarma ve insanlarla birlikte çalışma gibi senaryolarda vazgeçilmez kılar.

İnsansı robotlar ile ilgili kapsamlı deneysel çalışmalar yapılmaktadır. Çevre ile etkileşime girmek, insanlarla iş birliği ve insansı hareketler yapmak, bu çalışmaların ana hedeflerindendir.

Uzuv kontrolü, robota hızlı harekete imkan sağlamak için dinamik model tabanlı bir kontrol gerektirmektedir. Kol, engeller içeren bir çevrede çalışacağı için görev uzayında ters dinamik kontrol, bu senaryo için uygun görülmüştür. Ters dinamik kontrolünde, kontrol sinyali olarak robotun karar verilmiş ivmesi kullanılır. Bu ivme, ters dinamik modeline beslenerek eklemler için gereken kuvvetler bulunur. Görev uzayında yörünge takibinde hatalar, yine kartezyen koordinat sisteminde tanımlanır. Bu sayede eyleyicideki toplam hata, eklem uzayındaki kontrole göre daha düşük olur. Ayrıca bu yöntemde uç eyleyicinin yörünge boyunca hareketi tahmin edilebilmektedir, böylece engeller içeren çevrede hareket planlaması kolaylaşır. Görev uzayında yapılan bu kontrolde ters kinematik hesaplanması için sözde ters jakobiyen kullanılmıştır.

İnsansı robotlar bir çok eyleyici ve sensörden oluşur. Robotu kontrol etmek için aynı anda sensor bilgilerini değerlendirmek, hareketi planlamak ve eyleyicileri denetlemek gerekmektedir. Bu yüzden bu robotların yazılımlarında çoklu işlemler ve zamanlayıcılar kullanılır.

ROS (Robot İşletim Sistemi), bahsedilen uygulamalarda kullanılabilecek kütüphane ve araçları barındıran bir açık kaynaklı işletim sistemidir. Simulasyon ve donanım arayüzünün yanında gelişmiş algoritmalar sunar. ROS üzerinde koşan bir yazılım, nod adı verilen bir çok işlemden oluşur. Her bir nod, belirli frekanslarda farklı görevleri yerine getirir ve diğer nodlarla iletişime geçer. Bu yapı programlamayı kolaylaştırır ve açık kaynak kütüphaneleri nod olarak eklenmesini sağlayarak sisteme hızlı kuruylan modüler bir yapı kazandırır. ROS, şimdiden bir çok endüstriyel ve enstitü robotunu desteklemekte ve artık robotikte bir standart olarak görülmektedir. Yazılımında C++ ve Python kullanılabilmekte ve bu iki dilde yazılan kod parçaları, aynı anda birbiriyle haberleşerek koşabilmektedir.

Gazebo, dört pervaneli helikopter, manipülatör, sürü robotiği ve tam-vücut insansı robotlar gibi bir çok robotu, çeşitli sensörler ve çevresel etkileşimle birlikte simüle edebilen bir açık kaynaklı dinamik simülasyon ortamıdır. Gazebo, ROS da dahil olmak üzere çeşitli platformlarla arayüze sahiptir. İçinde çok sayıda eklenebilir obje barındırır ve SDF formatında hazırlanan bütün objeler eklenebilir. Gerekli eklenti

programları kullanılarak, ROS'un desteklediği URDF formatını SDF'ye çevirerek çalıştırabilir. Henüz ROS ortamında desteklenmese de birden fazla dinamik motoru ve aktarım elemanı sunar.

ITECH Kolu, İstanbul Teknik Üniversitesi Makina Mühendisliği Otomatik Kontrol Laboratuarı'nda üretilmiş altı serbestlik dereceli bir insansı robot koludur. Robot, Maxon firmasına ait fırçasız doğru akım motorlarla tahrik edilmekte ve aktarım elemanı olarak harmonik dişliler kullanılmaktadır. Bu tezin amacı, Otomatik Kontrol laboratuarı için kapsamlı bir kinematik ve dinamik kütüphanesi yaratmak, bu kütüphanenin ROS ile kullanılabilmesi için gerekli yazılım paketlerini yasmak ve sonunda ITECH kolunun görev uzayında ters dinamik kontrolünü Gazebo simulasyon ortamında uygulamaktır.

Bu tez altı bölümden oluşmaktadır. İlk bölümde tezin amacı, literatür taraması ve ITECH Robot Kolu'nun koşacağı işletim sistemi, motor güç ve limitleri, aktarım elemanları gibi mekanik-yazılımsal özelliklerinden bahsedilmiştir.

İkinci bölümde bir robot kolun geometrik, kinematik ve dinamik modellenmesi anlatılmıştır. Seçilen mevcut kinematik ve dinamik çözümlerin, alternatiflerine göre yapılan işlem sayısı bakımından üstünlüklerinden bahsedilmiştir. ITECH Robot Kolu'nun geometrik, kinematik ve dinamik modeli türetilmiştir.

Üçüncü bölümde Robot İşletim Sistemi tanıtılmıştır. ROS'un yazılım mimarisinden ve imkanlarından bahsedilmiştir. ROS'un ve Gazebo'nun neden tercih edildiği ve ilerideki çalışmalarda, bu çalışmada hazırlanan yazılımların gerçek robotta nasıl kullanılabileceği anlatılmıştur. ITECH Kolu'nun ROS ortamına entegrasyonu ve ROS ile Gazebo dinamik simulasyon ortamının arayüzünün oluşturulma basamakları tarif edilmiştir.

Dördüncü bölümde robot kollarında yörünge oluşturulmasından ve bu yörüngeye ait zamanlama fonksiyonlarından bahsedilmiştir. Robot kollarının kontrol metodları tartışılmıştır. Merkezi olmayan ve merkezi kontrol algoritmalarına değinilmiştir. Görev uzayında yörünge takibi ve ITECH Kolu'nda ters dinamik kontrol algoritmasının uygulanması anlatılmıştır.

Beşinci bölümde noktadan noktaya ve çember yörüngeye ait yüklü ve yüksüz durumlarda, çeşitli kazanç ve kontrolcü frekanslarında simülasyon sonuçları verilmiştir. Kontrolcünün performansını test etmek amacıyla bu simulasyonlar 1m/s hızında yapılmıştır. Yüklü durumda robot yükten habersiz olduğu ve bu ek kütle modele dahil edilmediği için sisteme bir bozucu olarak etki etmiştir.Ayrıca örnek bir görev olarak al-yerleştir senaryosu sonuçları eklenmiştir. Simülasyon sonuçları irdelenmiştir.

Altıncı ve son bölümde tez boyunca yapılan çalışmalar özetlenmiştir. İleride dinamik algoritmaların geliştirilmesi için simulasyon ortamı seçimi ve gerçek robot üzerinde yapılacak çalışmalarda kullanılabilecek haberleşme teknolojileri için tavsiyelerde bulunulmuştur.

Ekler bölümünde robotun geometrik ve kütle özelliklerinin yanı sıra, bu tez için yazılan nesne tabanlı Python kütüphanesinin sınıfları ve bu sınıflara ait fonksiyonların

kullanımı verilmiştir. Bu kütüphane, bütün tek zincir seri robot kollarına uygun olduğu için ITECH Kolu'nda yapılacak serbestlik derecesi, eksen değişikliği gibi mekanik değişimler, birkaç satır kod ile bu tezdeki kodu kullanarak yeni robot koluna uygulanabilir. Ayrıca, tezde sözde ters jakobiyen yöntemi ile ters kinematik kullanıldığı için, serbestlik derecesi altıdan farklı olan robot kolları da bu kütüphaneyle yaratılacak kodla kontrol edilebilir. İnsan kolu gibi serbestlik derecesi altıdan büyük robotlar için fazlalık çözünürlüğünün eklenmesi gerekmektedir.

1. INTRODUCTION

Robotics is a multidisciplinary field of study based on electronic, control, mechanical and computer engineering and it requires a good understanding of physics, mathematics and control theory to study on problems of robotics.

Robots exist in many forms, such as unmanned vehicles, manipulators, humanoid robots. In this thesis, the arm of a humanoid robot is studied, which is a six degrees of freedom serial manipulator with non-spherical wrist.

Human arm is actually consists of seven degrees of freedom revolute joint with spherical wrist, but many humanoid robots consist of six or less revolute joints which are capable of executing many daily tasks, with the sacrifice of redundancy in task space. The main concerns in humanoid robot arms is being capable of human-like motions, while ensuring the safety required for working with humans.

Controlling a six degrees of serial manipulator generally requires model based approaches, where a mathematical representation of robot kinematics and dynamics are derived. The control algorithm is based on this kinematic and dynamic terms and defined task.

1.1 Purpose of Thesis

Purpose of this thesis is generating a control algorithm for ITECH Manipulator. A side task is making this algorithm generic by making it capable of controlling any n degrees of freedom serial manipulators.

Execution of human-like tasks or assisting humans require working in clustered environments, thus, it is needed to define the motion in task space to predict the robot hand motion during whole trajectory. Considering the model of the robot, a proper

control algorithm has to be selected and applied to robot manipulator by taking hand motion and robot structure into consideration.

1.2 Literature Review

Control and simulation of robot manipulators has been studied for a long time. The simplest control algorithm is independent joint control, a decentralized control where each joint are controlled with a PID, without using the dynamic and kinematic model. In centralized control, PD Control with Gravity Compensation is the most basic form, where the control law includes nonlinear coupling terms. In this method, a linear PD feedback plus gravity compensation torques are used in the control law. In [1] a Newton-Euler formulation based computation is proposed. In this work, the inertial, coriolis and centrifugal forces and the resulting joint torques are computed efficiently online.

The simulation of the robot manipulator is the first stage for controlling an actual robot. There are a number of robot simulators that simulates the robot and its environment by solving the related dynamic equations. MATLAB-Simulink is a well known platform for simulating dynamical systems and designing controllers. Powerful toolboxes for MATLAB exist,such as Drake, which is a layer built on top of the MATLAB-Simulink engine that allows the user to define structured dynamical system [2]. It provides a number of tools for analysis and controller design which take advantage of the system structure. A commonly used simulator is Gazebo, a multi-robot simulator for outdoor environments. It supports multiple physics engines (ODE, Bullet, DART) and, thanks to its modular and plugin-based structure, can be extended with new features [3]. Another simulator is V-Rep, which has a development environment based on a distributed control architecture: each object/model can be individually controlled via an embedded script, a plugin, a ROS node, a remote API client [4].

1.3 ITECH Humanoid Robot Manipulator

ITECH is a humanoid robot manipulator designed and constructed in System Dynamics and Control Laboratory of Istanbul Technical University Mechanical

Engineering faculty. It consists of six brushless DC Maxon motors, harmonic drives and an aluminium-steel body. A view of ITECH Arm is given in (Figure 1.1).

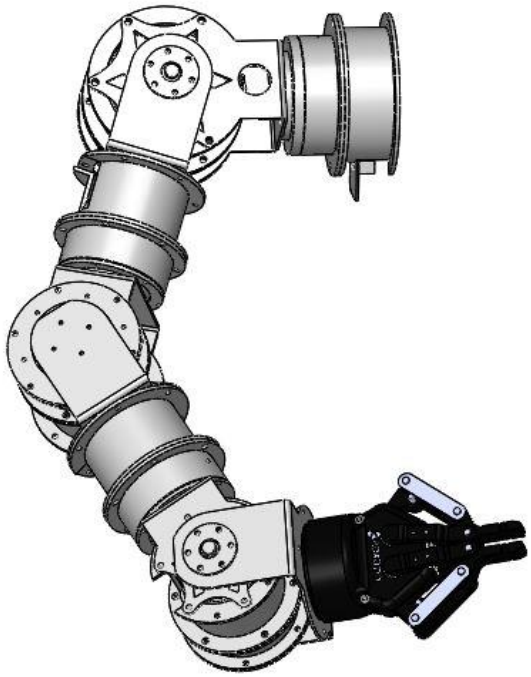


Figure 1.1: ITECH Humanoid Robot Manipulator

1.3.1 Mechanical Properties

The mechanical properties of ITECH Arm manipulator are given in Table 1.1 [5].

Table 1.1: Mechanical Properties of ITECH Humanoid Manipulator

joint	Motor	Harmonic Drive	Torque (Nm)	Range(rad)
1	EC90	CPL-20	71.3	- π ..+ π
2	EC90	CPL-20	71.3	- 2π/3 ..+ 2π/3
3	EC60	CPL-17	34.8	- 2π/3 ..+ 2π/3
4	EC60	CPL-17	34.8	- π ..+ π
5	EC60	CPL-17	34.8	- 2π/3 ..+ 2π/3
6	EC60	CPL-17	34.8	- π ..+ π

The actuation properties of ITECH Arm manipulator are given in table Table 1.2 [5].

Table 1.2: Actuation properties of ITECH Humanoid Manipulator

Specifications	EC90	EC60
Power	90 Watt	100 Watt
Voltage	24V	24V
Moment	444 mNm	283 mNm
Max Velocity	5000 rpm	6000 rpm
Mass	0.61 kg	0.48 kg
Encoder	$1024\frac{At\beta m}{N}$	$1024\frac{At\beta m}{N}$

1.3.2 Software Properties

ROS, an open-source, meta-operating system for robots, is the platform planned to control the humanoid robot. Thus, the controller design and simulations of the arm is developed in ROS environment and by using a 3rd party dynamic simulation environment, GAZEBO [6].

2. MODELLING OF ROBOT

Modelling of the robot generally consists of two parts; kinematic modelling and dynamic modelling. In kinematic modelling, the relationship between the joint positions, velocities, accelerations and link positions, velocities and accelerations are considered. In dynamic modelling, the relationship between the joint forces and robot motion is considered under internal and external forces.

2.1 Geometric Model of Robot

In geometric model, the robot link frames are computed with respect to the corresponding joint positions. In forward geometric model, the goal is computing the pose of the end effector from the joint positions of the robot, whereas in inverse geometric model, the joint positions of the robot is computed with respect to the corresponding end effector pose, which is not evaluated in this work due to the task space control laws.

2.1.1 Forward geometric model of robot

The serial manipulators, the links of the robot is connected to its adjacent links with one joint, which is the case in ITECH manipulator. In the notation used in this work, the base link is considered as Link 0 and fixed with respect to the world frame. The first joint, which connects Link 0 to Link 1 is Joint 1. Thus, the robot consists of n joints and n+1 links. In our case, 7 links and 6 joints. The last link is referred as ‘end effector’.

In manipulators with single degree of freedom joints consist of prismatic and revolute joints. The joint variable in prismatic joints is linear displacement, while it is angular displacement in revolute joints.

$$q_i = \begin{matrix} d_i \text{ for prismatic joints} \\ \theta_i \text{ for prismatic joints} \end{matrix} \quad \textbf{(2.1)}$$

2.1.1.1 Link parameters and link coordinate systems

In this section, we assign coordinate frames to each link. We are following the Denavit-Hartenberg convention in this work. The joint and link vectors are given in (Figure 2.1) [7]. The frame assignment is described below [8]:

- The z_i axis is aligned with the joint axis (i+1). The direction of rotation is arbitrary.
- The x_i- axis is defined along the common normal between the joint axis i and joint axis (i+1) and points from the joint axes i to the joint axis (i+1). In case the joint axes are parallel, the x_i- axis can be chosen arbitrarily, while being perpendicular to the two joint axes.
- The y_i-axis is assigned by the right-hand rule.

The assignment of coordinate frames according to Denavit-Hartenberg convention is described below:

1. As mentioned above, the base link is numbered as link 0 and the links are numbered from 0 to i+1 joints. The first joint is numbered as joint 1 and joint i connects link i to link i+1.
2. The common normal are drawn between the adjacent joint axes.
3. The base coordinate system is assigned in such a way that z_0-axis is aligned with the first joint axis. The x_0-axis is perpendicular to z_0-axis.
4. The last coordinate system is assigned in such a way that its x-axis is perpendicular to last joint axis.
5. The assignment of coordinate frames are described below:
    - The z_i-axis is aligned with the joint axis (i+1)
    - The x_i-axis is assigned along the common normal between the joint axis i and joint axis (i+1). In case of parallel joint axes, the x_i-axis is chosen arbitrarily while being perpendicular to the two joint axes. For intersecting joint axes, x_i-axis can be assigned arbitrarily while being the cross product of vectors z_i and z_{i+1}.

6. The link parameters and joint variables a_i, α_i, θ_i and d_i are determined.

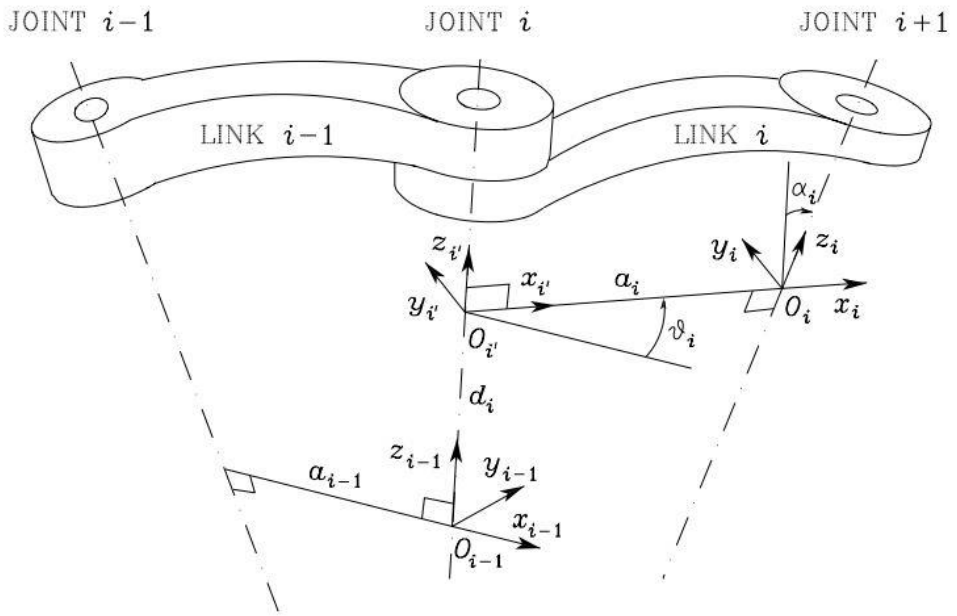


Figure 2.1: Denavit-Hartenberg kinematic parameters

2.1.1.2 Denativ-Hartenberg Transformation Matrices

The coordinate frames assigned in the previous chapter are used to generate the Denavit-Hartenberg transformation matrices. The parameters of denavit-hartenberg is described below [8]:

1. The coordinate system (i+1) is translated along the z_{i-1} along d_i distance in order to coincide origin of this coordinate system with z_{i-1} axis.
2. The translated coordinate system is rotated about the z_{i-1} for θ_i angle in order to align the axis x_i and x_{i-1}.
3. The rotated coordinate system is translated along x_i-axis for a_i distance in order to coincide the origin of coordinate system (i-1) with coordinate system i.
4. The translated coordinate system is rotated about the x_i-axis for α_i angle in order to coincide the two coordinate systems.

The homogenous transformation matrices corresponding to the transformations and rotations given above are:

$$\mathrm{A(z,d)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.2)}$$

$$\mathrm{A(z,\theta)} = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.3)}$$

$$\mathrm{A(x,a)} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.4)}$$

$$\mathrm{A(x,\alpha)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.5)}$$

The resulting transformation is calculated from the products of the given homogenous transformation matrices.

$${}^{i-1}_{i}A = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i & a_i c\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.6)}$$

Thus, the forward geometric model of the robot is

$${}^{0}_{n}A = {}^{0}_{1}A^{1}_{2}A \ldots {}^{n-1}_{n}A \quad \textbf{(2.7)}$$

The ${}^{0}_{n}A$ matrix represents the position and orientation of the manipulator.

$${}^{0}_{n}A = \begin{bmatrix} & {}^{0}_{n}R(q) & & {}^{0}_{n}p(q) \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2.8)}$$

The rotation matrix ${}^{i-1}_{i}R$ is

$$ {}^{i-1}_{i}R = \begin{bmatrix} c\theta_i & -c\alpha_i s\theta_i & s\alpha_i s\theta_i \\ s\theta_i & c\alpha_i c\theta_i & -s\alpha_i c\theta_i \\ 0 & s\alpha_i & c\alpha_i \end{bmatrix} \quad \textbf{(2.9)} $$

The translation vector is ${}^{i-1}_{i}p$ is

$$ {}^{i-1}_{i}p = \begin{bmatrix} a_i c\theta_i \\ a_i s\theta_i \\ d_i \end{bmatrix} \quad \textbf{(2.10)} $$

2.1.2 Position and orientation of manipulator

Position analysis of the manipulator is straightforward as it is represented directly as Cartesian coordinates in the robot transformation matrix ${}^{0}_{n}A$.

$$ \begin{bmatrix} x \\ y \\ z \end{bmatrix} = {}^{0}_{n}p(q) $$

In order to get [3x1] orientation from the rotation matrix ${}^{0}_{n}R(q)$ of the robot transformation matrix ${}^{0}_{n}A$, different representations exist.

2.1.2.1 Euler Angles

In Euler Angles representation, the rotation of a rigid body is expressed at three successive rotations by angles ϕ, θ and ψ. There are 24 different sets of Euler Angles. ZYZ representation is given in (Figure 2.2) [9]. For this representation,

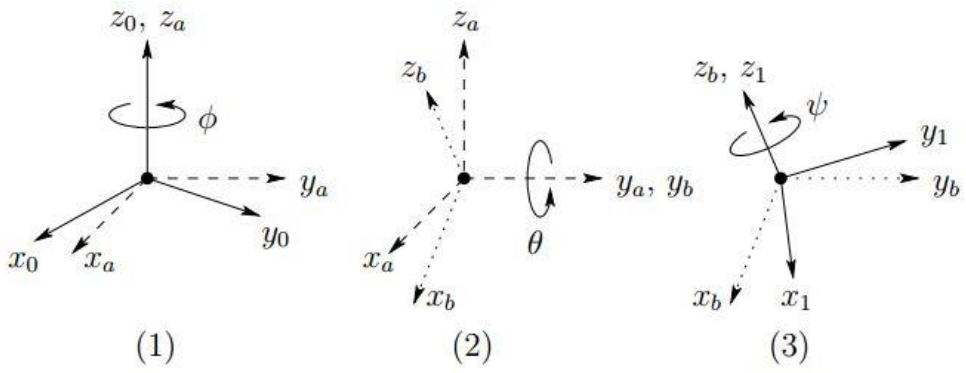


Figure 2.2: ZXZ Euler Angles Representation

The rotation sequence is as follows:

1. Rotation by an angle ϕ in z axis
2. Rotation by an angle ϑ in y axis
3. Rotation by an angle ψ in z axis

The rotation matrices of rotations in each axis are

$$\mathrm{R}(\mathrm{z},\phi) = \begin{bmatrix} c\phi & -s\phi & 0 \\ s\phi & c\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

$$\mathrm{R}(\mathrm{y},\vartheta) = \begin{bmatrix} c\vartheta & 0 & s\vartheta \\ 0 & 1 & 0 \\ -s\vartheta & 0 & c\vartheta \end{bmatrix} \tag{2.12}$$

$$\mathrm{R}(\mathrm{z},\psi) = \begin{bmatrix} c\varphi & -s\varphi & 0 \\ s\varphi & c\varphi & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.13}$$

$$\begin{aligned} R &= \mathrm{R}(\mathrm{z},\phi)\mathrm{R}(\mathrm{y},\vartheta)R(z,\psi) \\ &= \begin{bmatrix} c\phi c\vartheta s\psi - s\phi s\psi & c\phi s\vartheta s\psi - s\phi c\psi & c\phi s\vartheta \\ s\phi c\vartheta c\psi + c\phi s\psi & -s\phi c\vartheta c\psi + c\phi s\psi & s\phi s\vartheta \\ -s\vartheta c\psi & s\vartheta s\psi & c\vartheta \end{bmatrix} \end{aligned} \tag{2.14}$$

$$\begin{aligned} \vartheta &= arctan2\left(R_{33}, \sqrt{1 - R^2{}_{32}}\right) \\ \psi &= arctan2(R_{13}, R_{23}) \\ \phi &= arctan2(-R_{31}, R_{32}) \end{aligned} \tag{2.15}$$

2.1.2.2 Angle Axis

In angle axis representation, the orientation is expressed as a rotation of a given angle about a unit vector.

Given the unit vector r and angle ϑ, the rotation matrix is

$$\mathrm{R}(\vartheta, \mathrm{r}) =$$

$$= \begin{bmatrix} r_x^2(1-c\vartheta)+c\vartheta & r_x r_y(1-c\vartheta)-r_z s\vartheta & r_x r_z(1-c\vartheta)+r_y s\vartheta \\ r_x r_y(1-c\vartheta)+r_z s\vartheta & r_y^2(1-c\vartheta)+c\vartheta & r_y r_z(1-c\vartheta)-r_x s\vartheta \\ r_x r_z(1-c\vartheta)-r_y s\vartheta & r_y r_z(1-c\vartheta)+r_x s\vartheta & r_y^2(1-c\vartheta)+c\vartheta \end{bmatrix} \quad \textbf{(2.16)}$$

2.2 Kinematic Model of Robot

Kinematic model of a robot is the mapping between the joint space velocities and task space velocities. The matrix used in mapping the joint velocities to Cartesian velocities is Jacobian matrix. For the inverse kinematics, several methods are discussed in the following chapter.

2.2.1 Forward kinematic model of robot

In forward kinematics, the goal is to derive the relationship between the joint velocities and end-effector linear and angular velocities. The linear and angular velocity of the end-effector is given as $\dot{p}_e$ and $\dot{\omega}_e$ respectively. The joint velocities are $\dot{q}$ [7].

$$\dot{p}_e = J_P(q)\dot{q} \quad \textbf{(2.17)}$$

$$\omega_e = J_O(q)\dot{q} \quad \textbf{(2.18)}$$

In (2.17), matrix J_P is the mapping of joint velocities $\dot{q}$ to linear velocities of the end-effector, $\dot{p}_e$ and is a (3 x n) matrix. In (2.18), matrix J_O is the mapping of joint velocities $\dot{q}$ to angular velocities of the end-effector, $\dot{\omega}_e$ and (3 x n) matrix. In manipulator control, the Jacobian and task space velocities are written in a compact form:

$$v_e = \begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix} = J(q)\dot{q} \quad \textbf{(2.19)}$$

The Jacobian is a (6 x n) matrix and in our case, it is a (6 x 6) matrix.

$$\mathrm{J} = \begin{bmatrix} J_P \\ J_O \end{bmatrix} \tag{2.20}$$

The geometric properties of Link i are given in Figure 2.3 [7].

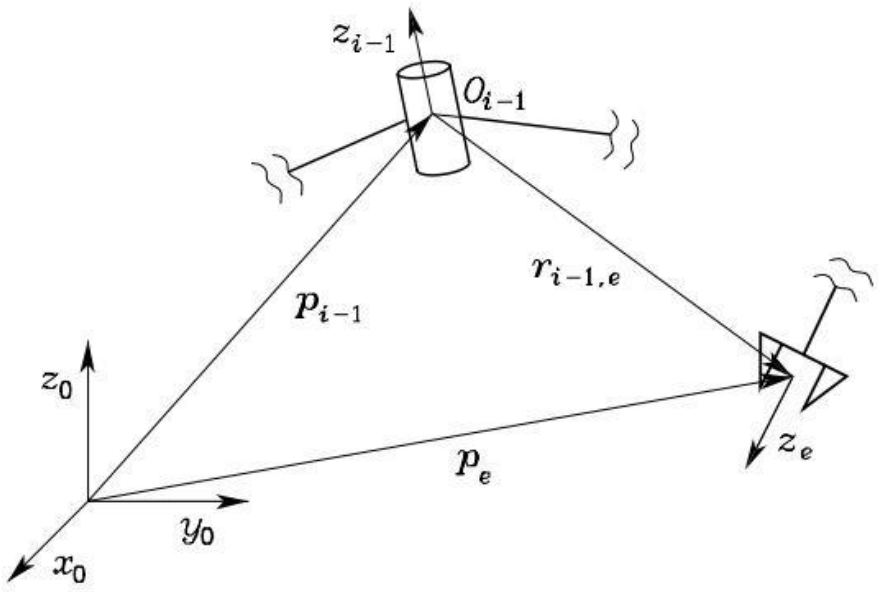


Figure 2.3: Link i of a manipulator

2.2.1.1 Geometric Jacobian

The angular and linear velocities of the links actuated by revolute joints are given respectively [7]:

$$\omega_i = \omega_{i-1,i}\, x\, r_{i-1,i} \tag{2.21}$$

And

$$\dot{p}_i = \dot{p}_{i-1} + \omega_i\, x\, r_{i-1,i} \tag{2.22}$$

Using the equations (2.21) and (2.22), the Jacobian is derived as

$$J_{P_i} = z_{i-1}\, x\, (p_e - p_{i-1}) \tag{2.23}$$

And

$$J_{O_i} = z_{i-1} \tag{2.24}$$

Thus, the Jacobian matrix of the manipulator is

$$\mathrm{J} = \begin{bmatrix} J_{P_1} & \cdots & J_{P_n} \\ J_{O_1} & & J_{O_n} \end{bmatrix} \tag{2.25}$$

The z_{i-1} term is the third column of the rotation matrix ${}^{i-1}_{i}R$.

2.2.1.2 Analytic Jacobian

Alternatively Jacobian can be generated by using differentiation. Given the forward geometry

$$\mathrm{k(q)} = \begin{bmatrix} p_e \\ \phi \end{bmatrix} \tag{2.26}$$

Where ϕ is the set of Euler angles.

$$\dot{x}_e = \begin{bmatrix} \dot{p}_e \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} J_P(q) \\ J_\phi(q) \end{bmatrix} \dot{\mathrm{q}} = J_A(q)\dot{\mathrm{q}} \tag{2.27}$$

By differentiating the forward geometry with respect to the joint variables

$$J_A(q) = \frac{\partial \mathrm{k(q)}}{\partial q} \tag{2.28}$$

Note that this analytical Jacobian is a symbolic representation that includes trigonometric functions and multiplications. As analytic Jacobian maps the joint velocities to orientation velocity $\dot{\phi}$ in terms Euler angle set given in (2.1.2.1), a transformation is required between angular velocity ω and $\dot{\phi}$.

$$\omega = \mathrm{T}(\phi)\dot{\phi} \tag{2.29}$$

The transformation for XYZ Euler angles representation is

$$\mathrm{T} = \begin{bmatrix} 1 & 0 & s\vartheta \\ 0 & c\phi & -s\phi c\vartheta \\ 0 & s\phi & c\phi s\vartheta \end{bmatrix} \tag{2.30}$$

The relationship between geometric and analytic Jacobian is

$$T_A(\phi) = \begin{bmatrix} I & 0 \\ 0 & \mathrm{T}(\phi) \end{bmatrix} \tag{2.31}$$

$$\mathrm{J} = T_A(\phi)J_A \tag{2.32}$$

2.2.2 Inverse kinematic model of robot

Inverse kinematics of a robot represents the relationship between the task space velocities and joint space velocities. A few different methods exist to compute the inverse kinematics of the manipulator.

2.2.2.1 Inverse Jacobian

In this method, the inverse of the Jacobian is computed directly and the relationship is as follows

$$\dot{\theta} = J^{-1}\dot{p}_e \tag{2.33}$$

As the robot approaches kinematic singularity, the determinant of the Jacobian approaches to zero and when kinematic singularity is reached, the determinant

becomes zero as the Jacobian matrix loses a rank. Thus, this situation yields infinite joint velocities. Several methods exist to handle this problem.

2.2.2.2 Pseudo inverse

This method uses the least-square approximation to solve the non-invertible Jacobian matrix problem. This method enables the use of inverse Jacobian in redundant manipulators. The velocities are calculated as

$$\dot{\theta} = J^T(JJ^T)^{-1}\dot{p}_e \quad \textbf{(2.34)}$$

The matrix $(JJ^T)^{-1}$ is guaranteed to be invertible, yet this method performs poorly in the neighbourhood of singular configurations.

2.2.2.3 Damped least squares

In damped least-squares method, a damping coefficient λ is introduced to the pseudo-inverse method.

$$\dot{\theta} = J^T(JJ^T + \lambda^2 I)^{-1}\dot{p}_e \quad \textbf{(2.35)}$$

Several methods exist to optimize the damping coefficient λ during the manipulation [10].

2.2.2.4 Jacobian transpose

Given the task space error p_e, joint variables q and gain K, the error term in the equation

$$\dot{q} = J^T K p_e \quad \textbf{(2.36)}$$

converges to zero.

2.3 Dynamic Model of Robot

The dynamic model of a robot is crucial in design and motion control of the robot as the required joint torques must be calculated for the execution of motion.
The dynamic model of a robot manipulator is in the form of

$$\mathrm{B(q)}\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau \qquad \textbf{(2.37)}$$

where $\mathrm{B(q)}$ is the Mass Matrix and represents the inertial forces, $C(q,\dot{q})$ represents the Coriolis and Centrifugal forces and $g(q)$ represents the gravitational forces.

2.3.1 Euler-Lagrange dynamics

The Lagrange dynamics uses the energy equations of the robot and is independent of the coordinate frame.
Lagrangian of the manipulator is

$$\mathrm{L} = \mathrm{K} - \mathrm{P} \qquad \textbf{(2.38)}$$

where K is the kinetic and P is the potential energy.
The Lagrange equations are derived as

$$\frac{\mathrm{d}}{\mathrm{dt}}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = \tau_i \qquad \textbf{(2.39)}$$

2.3.2 Newton-Euler dynamics

The Newton-Euler dynamics is a recursive algorithm that is based on the balance of forces in each link of the manipulator. The vectors used in this algorithm are given in (Figure 2.4). In forward recursion, link velocities and accelerations are computed and in backward recursion, joint torques are computed [7].

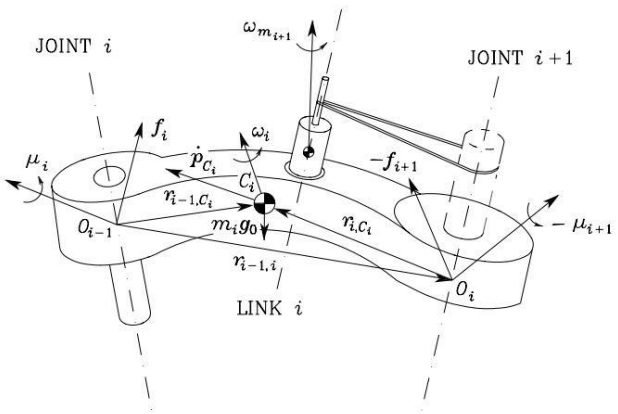


Figure 2.4: Force balance representation on Link i

The recursive Newton-Euler algorithm is as follows

Forward recursion:

$$\omega_i^i = R_{i-1}^i(\omega_{i-1}^{i-1} + \dot{\theta}_i z) \tag{2.40}$$

$$\dot{\omega}_i^i = R_{i-1}^i(\dot{\omega}_{i-1}^{i-1} + \ddot{\theta}_i z + \dot{\theta}_i \omega_{i-1}^{i-1} \text{ x } z) \tag{2.41}$$

$$\ddot{p}_i^i = R_{i-1}^i \ddot{p}_{i-1}^{i-1} + \dot{\omega}_i^i \text{ x } r_{i-1,i}^i + \omega_i^i \text{ x } (\omega_i^i \text{ x } r_{i-1,i}^i) \tag{2.42}$$

$$\ddot{p}_{C_i}^i = \ddot{p}_i^i + \dot{\omega}_i^i \text{ x } r_{i,C_i}^i + \omega_i^i \, x \, (\omega_i^i \text{ x } r_{i,C_i}^i) \tag{2.43}$$

Backward recursion:

$$f_i^i = R_{i+1}^i f_{i+1}^{i+1} + m_i \ddot{p}_{C_i}^i \tag{2.44}$$

$$\mu_i^i = -f_i^i \text{ x } \left(r_{i-1,i}^i + r_{i,C_i}^i\right) + R_{i+1}^i \mu_{i+1}^{i+1} + \left(R_{i+1}^i f_{i+1}^{i+1}\right) \text{x } r_{i,C_i}^i + I_i^i \dot{\omega}_i^i \tag{2.45}$$

$$+\omega_i^i \text{ x } (I_i^i \dot{\omega}_i^i)$$

$$\tau_i = \mu_i^{i^T} R_{i-1}^i z \quad \textbf{(2.46)}$$

$$z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \text{ constant vector} \quad \textbf{(2.47)}$$

$$R_{i-1}^i = R_i^{i-1^T} \quad \textbf{(2.48)}$$

2.4 Modelling of ITECH Manipulator

In order to generate the model of ITECH Manipulator, a generic object-oriented Python library is created. This created library takes the coordinate systems and inertial properties as input and generates the model of the robot and provides the necessary geometric, kinematic and dynamic functions. The code is capable of generating the kinematic and dynamic model of an n degree of freedom single chain manipulator. This code is then uploaded in a GitHub library as an open-source project.

2.4.1 Geometric and kinematic model

The coordinate frames and Denavit-Hartenberg parameters of ITECH Manipulator are assigned using the method described in (2.1.1.1). The coordinate frames are given in Figure 2.5 and the Denavit-Hartenberg parameters θ_i, α_i, d_i and a_i of ITECH Manipulator are given in Table 2.1.

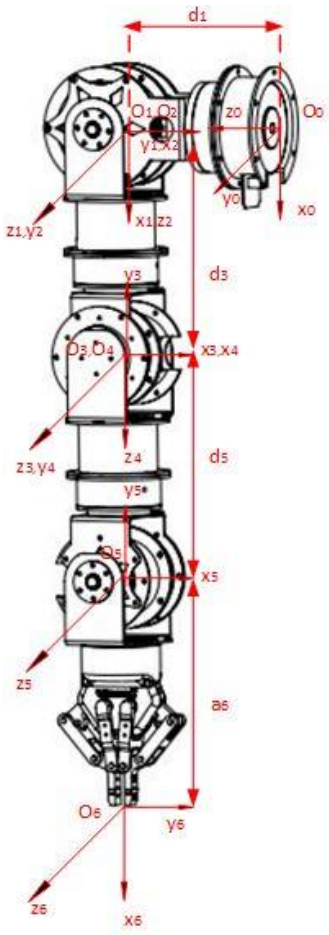


Figure 2.5: Link coordinate systems of ITECH Manipulator

Table 2.1: Denavit-Hartenberg Parameters of ITECH Manipulator

joint	a_i(m)	d_i(m)	α_i(rad)	θ_i(rad)
1	0	0.16496	$-\pi/2$	θ_1
2	0	0	$\pi/2$	$\theta_2 + \pi/2$
3	0	0.219	$-\pi/2$	θ_3
4	0	0	$\pi/2$	θ_4
5	0	0.213	$-\pi/2$	θ_5
6	0.21718	0	0	$\theta_6 - \pi/2$

The transformation matrices ${}^{0}_{1}A, {}^{1}_{2}A, \ldots, {}^{5}_{6}A$ and the robot transformation matrix ${}^{0}_{6}A$ is generated using the formulations (2.6) and (2.7).

The Jacobian of the ITECH Manipulator is generated using the formulations given in (2.2.1). It should be noted that, the terms of Jacobian matrix consist of a large number of coupled elements. Thus, generating a numerical Jacobian using the equations (2.23) and (2.24) online is computationally much more efficient than generating a symbolic Jacobian. In this work, numerical Jacobian is used and computed online.

2.4.2 Dynamic model

The dynamic model of ITECH Manipulator is generated using the Recursive Newton-Euler Algorithm described in 2.3.2. In order to include the gravitational forces, the base linear acceleration term $\ddot{p}_0^0$ is defined as

$$\ddot{p}_0^0 = \begin{bmatrix} -g \\ 0 \\ 0 \end{bmatrix} \tag{2.49}$$

where g is the acceleration of gravity. Other initial terms ω_0^0, $\dot{\omega}_0^0$ are taken as zero vectors. The end-point force and moment terms f_n^n, μ_n^n can be selected as external forces. In this work, they are also taken as zero vectors.

The inertial properties and link centre of gravity vectors are gathered from the SolidWorks model of the robot and given in the Appendix.

3. ROS- ROBOT OPERATING SYSTEM

3.1 General Structure of ROS

ROS is an open-source, meta-operating system for robots. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers [11].

3.1.1 Ros Nodes

Nodes are the process that perform computation. Each ROS node is written using ROS client libraries such as roscpp and rospy. Using client library APIs, we can implement different types of communication methods in ROS nodes. In a robot, there will be many nodes to perform different kinds of tasks. Using the ROS communication methods, it can communicate with each other and exchange data. Ros node network allows to build simple processes rather than a large process with all functionality [12].

3.1.2 Ros Topics

Each message in ROS is transported using named buses called topics. When a node sends a message through a topic, then we can say the node is publishing a topic. When a node receives a message through a topic, then we can say that the node is subscribing to a topic. The publishing node and subscribing node are not aware of each other's existence. The production of information and consumption of it are decoupled. Each topic has a unique name, and any node can access this topic and send data through it as long as they have the right message type.

3.1.3 Ros Messages

Nodes communicate with each other using messages. Messages are simply a data structure containing the typed field, which can hold a set of data and that can be sent to another node. ROS supports standard primitive types (Boolean, Integer, Float etc.) and user defined messages can be generated by using these standards.

3.1.4 Ros Services

The ROS services are a type request/response communication between ROS nodes. One node will send a request and wait until it gets a response from the other. The request/response communication is also using the ROS message description.

3.2 Gazebo

Gazebo is a 3D dynamic simulation environment that can simulate multiple robots simultaneously. Various types of sensors, including camera and LIDAR (Laser Scanning Range Finder) can be simulated in Gazebo. Current version of Gazebo uses Open Dynamics Engine (ODE) in its ROS compatible version, but its further versions, which are planned to be interfaced with ROS allow the use of Featherstone-based engines optimized for joint chains [13].

3.2.1 URDF universal robot description format

URDF is a package that contains XML specifications of a robot, sensors and actuators. Robot is defined as a child-parent relationship. Each joint has a parent and a child link and together they form the chain. ROS uses URDF as its robot description format. Major drawbacks of URDF is that it only supports open chain manipulators and does not include elastic joints.

3.2.2 SDF simulation description format

Similar to URDF, SDF is also an XML format used by Gazebo Simulator. URDF of a robot is converted to SDF by Gazebo for simulation. Though not used by ROS, it supports multiple robots, elastic joints, closed chain manipulators and can store the states of the robot.

3.3 Itech Arm Gazebo-Ros Interface

The procedure of controlling Itech Arm with ROS in Gazebo Simulator is described in the following section. The work done can be directly applied to control the actual Itech Arm manipulator.

3.3.1 Itech Arm description

To generate a Gazebo-Ros interface, initially a robot description package needs to be created. This package consists of the URDF xml file and the robot meshes in STL format. This is a similar procedure to SimMechanics, where an xml file and STL mesh files are generated via SolidWorks export option.

While URDF can be created by hand, using basic shapes known to this XML format, a more practical approach is using SolidWorks URDF exporter.

The joint coordinate frames and rotation axes are specified in SolidWorks assembly according to the axes assigned in section (2.4.1) to export the Itech Arm robot to URDF format. However, in URDF, each joint coordinate frame represents its child link, while in D-H representation, the joint coordinate frames represent its parent link. Thus, for the controller algorithm, the link mass properties should be taken from the SolidWorks robot model using D-H coordinate frames.

After exporting the assembly to URDF, the XML file needs to be modified by adding joint transmissions as Effort Joint Interface, provided by ros-control. These transmissions form an interface between the ROS controllers and the robot or the Gazebo simulator. The ROS controllers for Itech Arm are further explained in section (3.3.2). To communicate with Gazebo simulator, gazebo_ros_control plugin needs to be added to the URDF file.

3.3.2 Itech Arm control

This control package includes a YAML format file that holds the controller types of the joint actuators and a launch file that spawns these controllers using the Controller Manager of ros-control package. These controllers publish torque commands to the robot via Joint Command Interface and the robot publishes its joint position, velocity and torque states to Joint State Interface. Effort controllers are used in Itech Arm, thus, torque commands are published to the Joint Command Interface. The input/output relationship between the URDF, robot controller, simulator and the actual robot is given in figure (Figure 3.1) [14].

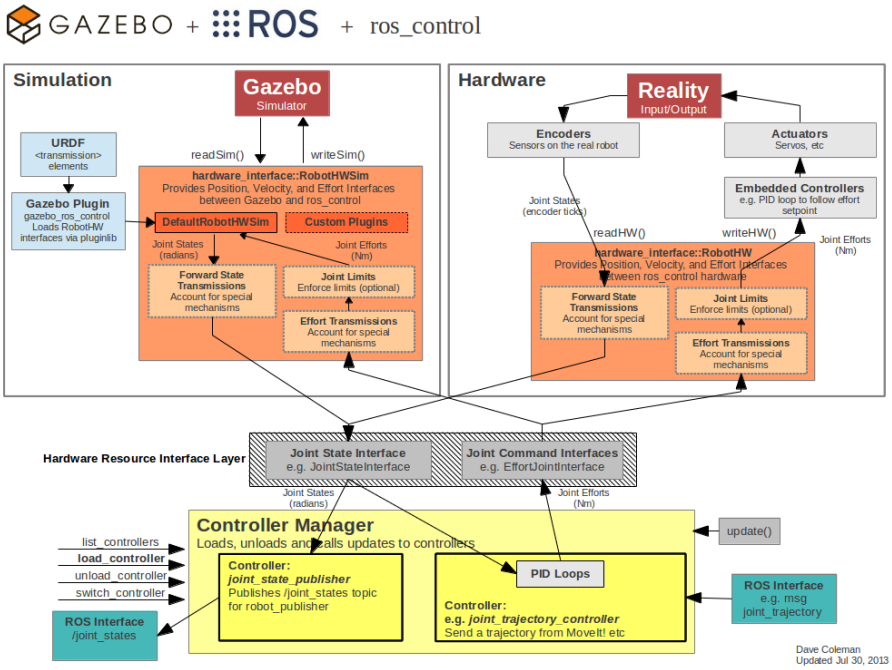


Figure 3.1: Gazebo Ros Interface

3.3.3 Itech Arm gazebo

This package holds the world and launch files of the robot. Gazebo simulator environment is described in the world file, such as objects, light sources and camera pose. The launch file starts Gazebo, loads the URDF of the robot to the simulator, launches the robot control and starts nodes that contains the control algorithms, such

as inverse dynamics controller and robot trajectory. In Itech Arm, a seperate package is created to hold the algorithm nodes as Itech Arm command. The visual of the robot inside the Gazebo simulation environment is given in (Figure 3.2).

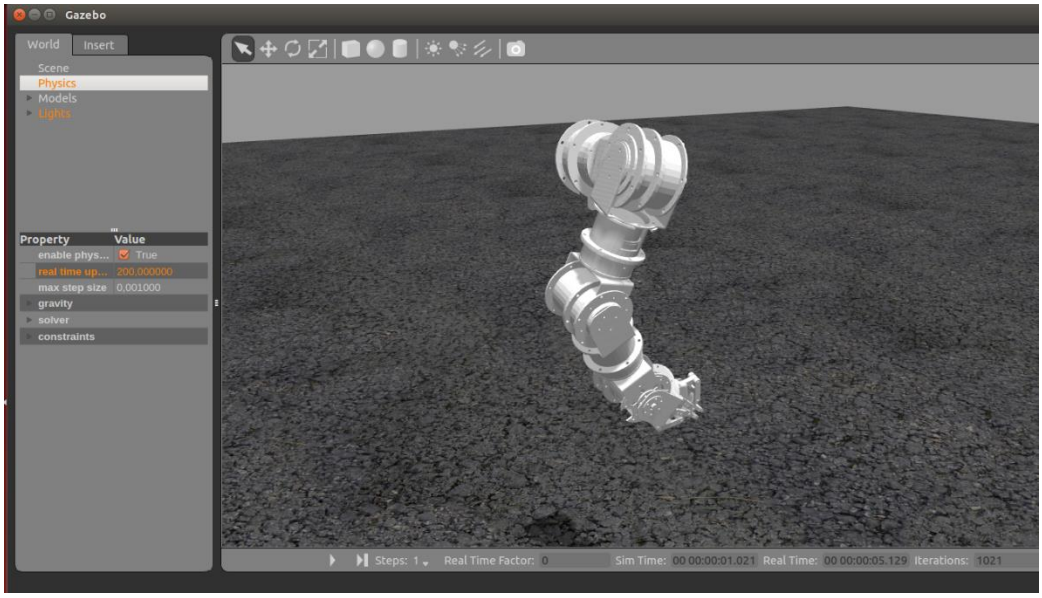


Figure 3.2: Itech Arm in Gazebo simulation environment

3.3.4 Itech Arm command

Itech Arm command is a package created to hold the kinematics and dynamics library, the trajectory generator and the control algorithm node. The Itech Arm control algorithm node subscribes to Joint State Publisher topic. The joint position and velocity states of the robot taken from the Joint State Publisher are used in the inverse dynamics control algorithm. The outputs of the inverse dynamics control algorithm are joint torques and these torque data are published to the joint_effort_controller/command topics. This way, the loop between the dynamic simulation and robot control algorithm is closed.

4. CONTROLLER DESIGN

4.1 Trajectory Generation

In order to have the robot execute a task, a trajectory has to be generated for this specific task, defining the initial and final points and the motion to be followed between these two points.

Let q_i and q_f be the initial and final positions of a joint. The trajectory between these two points is defined by the following equations

$$\mathrm{q(t)} = q_i + r(t)D \tag{4.1}$$

$$\dot{\mathrm{q}}(\mathrm{t}) = \dot{r}(t)D \tag{4.2}$$

In these equations, $D = q_f - q_i$ and $r(t)$ is the interpolation function.

4.1.1 Interpolation functions

Interpolation functions are used in trajectory generation in order to generate time-dependent configurations between the initial and final points. This ensures the continuation of position in the most basic form and can be further improved for velocity and acceleration.

4.1.1.1 Linear Interpolation

Linear Interpolation is the most basic form of interpolation functions. The time dependent function is:

$$\mathrm{q(t)} = q_i + \frac{t}{t_f}D \tag{4.3}$$

4.1.1.2 Cubic spline interpolation

In cubic spline interpolation, the interpolation function is a 3rd degree function, thus ensures the continuity of both velocity and position. Using the position and velocity boundary conditions, the interpolation function is generated as:

$$\mathrm{r(t)} = 3\left(\frac{t}{t_f}\right)^2 - 2\left(\frac{t}{t_f}\right)^3 \tag{4.4}$$

In order to minimize the travel time either acceleration or velocity reaches its limit. Thus the final time for minimum traveling time is defined as:

$$t_f = \begin{cases} \dfrac{3|D|}{2v_{limit}} & \text{if velocity limit } v_{limit} \text{ is reached} \\ \left(\dfrac{6|D|}{a_{limit}}\right)^{1/2} & \text{if acceleration limit } a_{limit} \text{ is reached} \end{cases} \tag{4.5}$$

In order to determine whether velocity or acceleration limit is reached, simply the one yielding the maximum t_f is saturated.

4.1.2 Task space trajectory generation

In task space trajectory generation, the initial and pose of the end effector is determined first and the poses between the initial and final pose are computed in order to synchronize the rotational and translational motions of the end effector.
Let the initial and final pose be

$$X_i = \begin{bmatrix} p_i \\ \phi_i \end{bmatrix} \text{ and } X_f = \begin{bmatrix} p_f \\ \phi_f \end{bmatrix} \tag{4.6}$$

Cartesian position of end effector is p_e and orientation of end effector is ϕ_e, the euler angles (φ,ϑ,ψ).

The position and orientation of trajectory and their time derivatives are

$$\begin{aligned} p_e(t) &= p_i + r(t)(p_f - p_i) \\ \dot{p}_e(t) &= \dot{r}(t)(p_f - p_i) \\ \ddot{p}_e(t) &= \ddot{r}(t)(p_f - p_i) \end{aligned} \tag{4.7}$$

$$\begin{aligned} \phi_e(t) &= \phi_i + r(t)(\phi_f - \phi_i) \\ \dot{\phi}_e(t) &= \dot{r}(t)(\phi_f - \phi_i) \\ \ddot{\phi}_e(t) &= \ddot{r}(t)(\phi_f - \phi_i) \end{aligned} \tag{4.8}$$

4.2 Motion Control

Motion control of a robot manipulator consists of calculating the joint control signals in order to execute the desired motion. The joint level controller may vary, such as position, velocity and torque closed-loop. As robot manipulators that have degrees of freedom equal or higher than 6 have highly coupled nonlinear terms, using the position and/or velocity closed loops in joint level tend to fail. Thus, in this work, torque closed loop control in joint level control is assumed.

4.2.1 Static-Model based control

The static model based control suggests the use of desired task space position error vector as an external vector, which is obtained from the following equation:

$$\tau = J^T(q) F_{ext} \tag{4.9}$$

Which yields the control law:

$$\tau = J^T(q)K_P\vec{e} - K_D\dot{q} + g(q) \quad \textbf{(4.10)}$$

This controller does not include the robot dynamics, thus, it is a relatively simple controller structure and has a naturally compliant behaviour.

4.2.2 Inverse dynamics control

As mentioned in the equation (2.37), the dynamic of the robot manipulator is in the form of

$$\mathrm{B(q)}\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau$$

Rewriting the equation (2.37) as

$$\mathrm{u} = \mathrm{B(q)}y + N(q,\dot{q}) \quad \textbf{(4.11)}$$

Where u is the control vector.

$$N(q,\dot{q}) = C(q,\dot{q})\dot{q} + g(q) \quad \textbf{(4.12)}$$

$$\ddot{q} = y \quad \textbf{(4.13)}$$

In the task space scheme, taking the derivative of

$$\dot{x}_e = J(q)\dot{q}$$

yields

$$\ddot{x}_e = J(q)\ddot{q} + \dot{J}(q,\dot{q})\dot{q} \quad \textbf{(4.14)}$$

The inverse dynamics control law generated for the robot manipulator is

$$\mathrm{y} = J^{-1}(q)(\ddot{x}_d + K_D\dot{\tilde{x}} + K_P\tilde{x} - \dot{J}(q,\dot{q})\dot{q}) \quad \textbf{(4.15)}$$

Where $\dot{\tilde{x}}$ and $\tilde{x}$ are task space pose errors.

The diagram of Inverse Dynamics Control in Task Space is given in (Figure 4.1)

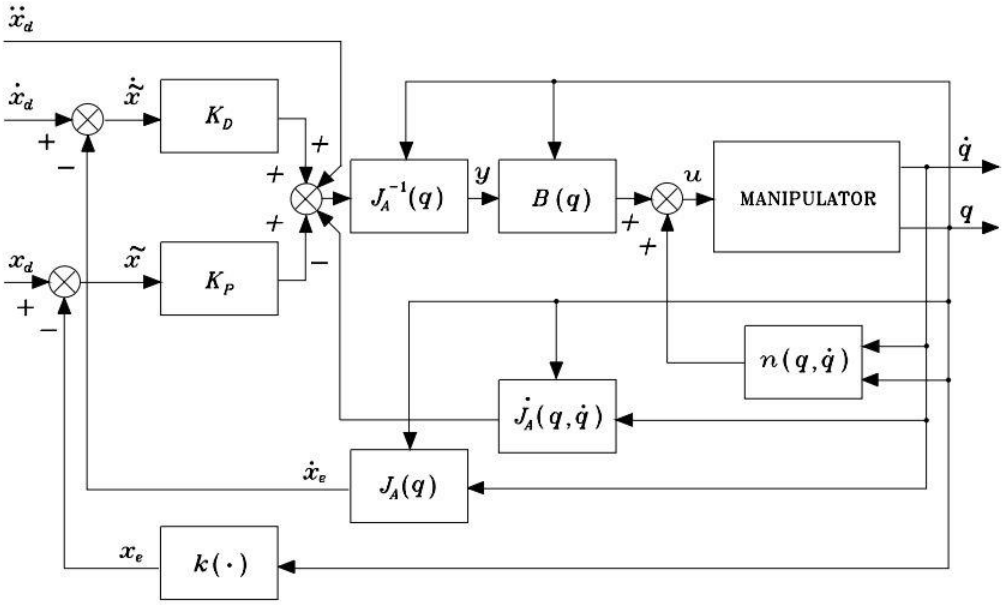


Figure 4.1: Inverse dynamics control in task space

4.2.3 Computation of task space errors

As the control law of inverse dynamics requires the computation of

$$\mathrm{y} = J^{-1}(a - \dot{J}\dot{q}) \quad \textbf{(4.16)}$$

Where a is the resolved acceleration, $\dot{v}_e$

$$a = \begin{bmatrix} a_p \\ a_O \end{bmatrix} \quad \textbf{(4.17)}$$

$$a_p = \ddot{p}_d + K_{D_P}\dot{\tilde{p}} + K_{P_P}\tilde{p} \quad \textbf{(4.18)}$$

The linear position and velocity errors are

$$\tilde{p} = p_d - p_g \tag{4.19}$$

$$\dot{\tilde{p}} = \dot{p}_d - \dot{p}_g \tag{4.20}$$

The desired terms in equations (4.16) and (4.17) are computed from forward geometry and kinematics respectively.

Calculation of a_O term of resolved acceleration is more complicated. Remembering the equation (2.19)

$$v_e = \begin{bmatrix} \dot{p}_e \\ \omega_e \end{bmatrix} = J(q)\dot{q}$$

$$\omega_e = T(\phi_e)\dot{\phi}_e \tag{4.21}$$

$$\dot{\omega}_e = T(\phi_e)\ddot{\phi}_e + \dot{T}(\phi_e, \dot{\phi}_e)\dot{\phi}_e \tag{4.22}$$

The resolved angular acceleration based on Euler angles error can be represented as:

$$a_O = T(\phi_e)\left(\ddot{\phi}_d + K_{D_O}\dot{\tilde{\phi}} + K_{P_O}\tilde{\phi}\right) + \dot{T}(\phi_e, \dot{\phi}_e)\dot{\phi}_e \tag{4.23}$$

4.2.4 Computation of Jacobian Derivative term

$$\ddot{X} = J(q)\ddot{q} + \dot{J}(q, \dot{q})\dot{q} \tag{4.24}$$

$$\dot{J}(q,\dot{q})\dot{q} = \begin{bmatrix} \ddot{p}(\ddot{q}=0) \\ \dot{\omega}(\ddot{q}=0) \end{bmatrix} \quad \textbf{(4.25)}$$

From Newton Euler recursive algorithm, in equations (2.42) and (2.41), $\ddot{p}_n^n$ and $\dot{\omega}_n^n$ calculated respectively.

$$\ddot{p} = {}_0^n R \ddot{p}_n^n \quad \textbf{(4.26)}$$

$$\dot{\omega} = {}_0^n R \dot{\omega}_n^n \quad \textbf{(4.27)}$$

Using the equation (4.25), by setting the joint accelerations zero, the $\dot{J}(q,\dot{q})\dot{q}$ term is calculated recursively [15].

5. SIMULATION RESULTS

The simulation studies are conducted in Gazebo. The robot is started at an initial pose. It is then commanded to go to a specified pose and draw a circle of 10 cm radius. In order to generate significant Coriolis and Centrifugal terms and test the inverse dynamics algorithm, 0.86 m/s velocity and 3.99 m/s^2 acceleration are reached during the motion trajectory. The simulations are done for different gains, controller frequencies and loads. The position and orientation results are both belong to end effector and with respect to the task space coordinate system.

5.1 Performance Criteria of Controller in Simulations

As performance index, integration of the absolute values and maximum errors of the errors in Cartesian and orientation space are used. The integration method is rectangle numerical integration.

$$\text{Integral Absolute Error} = \sum_{t_0}^{t_f} \text{absolute error(t)} * (\text{sampling time}) \quad \textbf{(5.1)}$$

$$\text{Absolute Cartesian Error(t)} = \sqrt{x_{error}(t)^2 + y_{error}(t)^2 + z_{error}(t)^2} \quad \textbf{(5.2)}$$

$$\text{Absolute Orientation Error(t)} = |\phi_{error}(t)| + |\vartheta_{error}(t)| + |\psi_{error}(t)| \quad \textbf{(5.3)}$$

5.2 Simulations without Payload Results

Dynamical simulations are conducted with zero payload using various gain configurations and controller frequencies. The simulation results for KP=1000.0, KD=50.0, 250 Hz are given in from Figure 5.1 to Figure 5.9 and in table from Table 5.1 to Table 5.3.

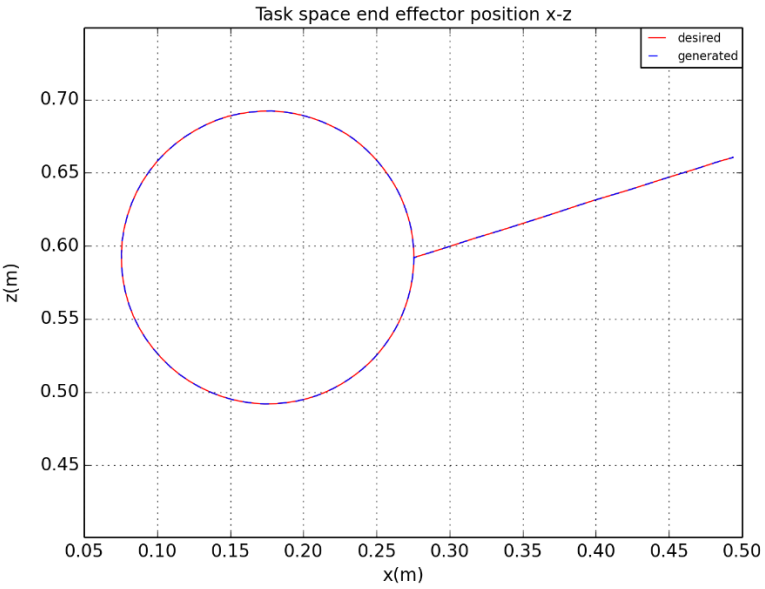


Figure 5.1: End effector x-z position

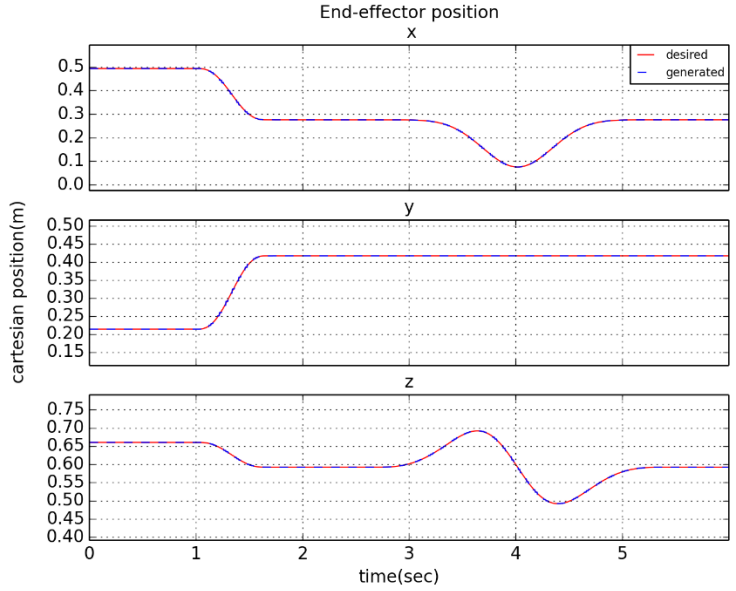


Figure 5.2: End effector cartesian position

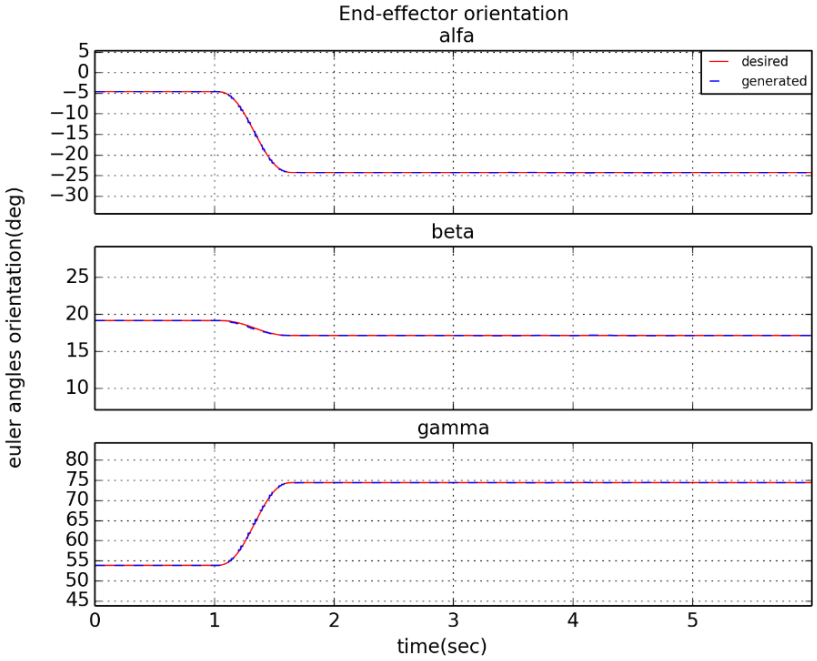


Figure 5.3: End effector orientation

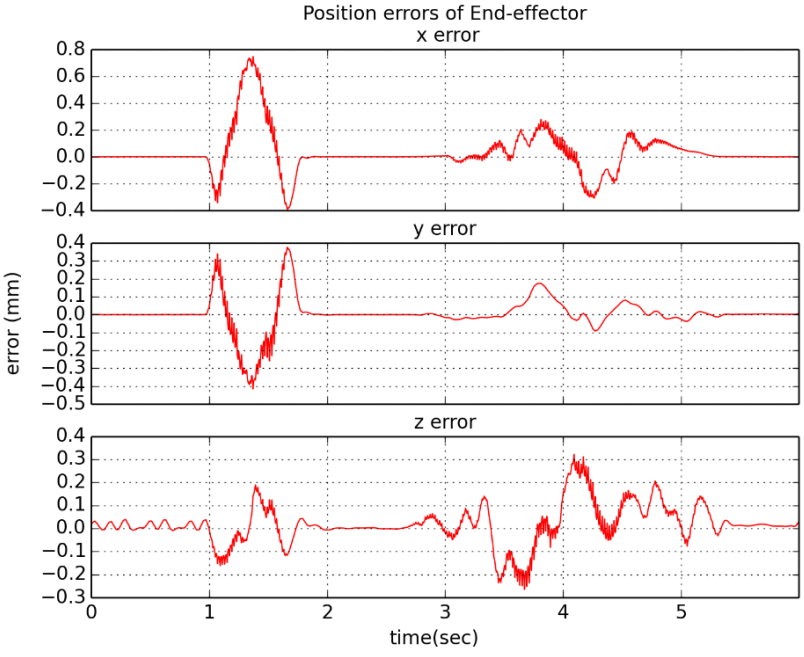


Figure 5.4: End effector position error

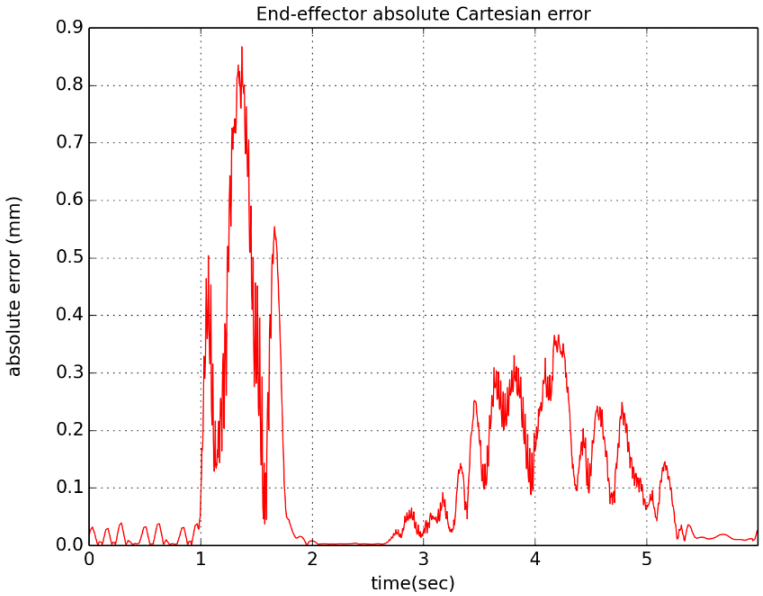


Figure 5.5: End effector position absolute error

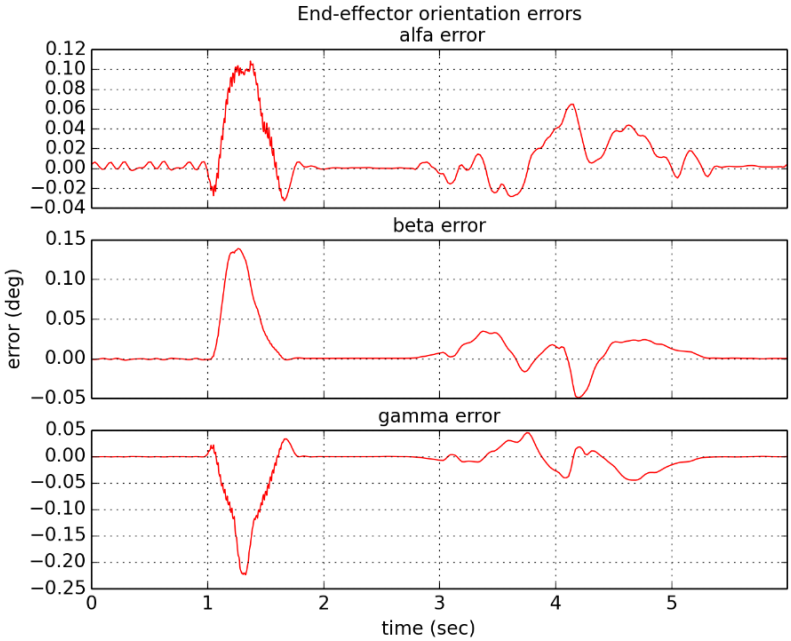


Figure 5.6: End effector orientation error

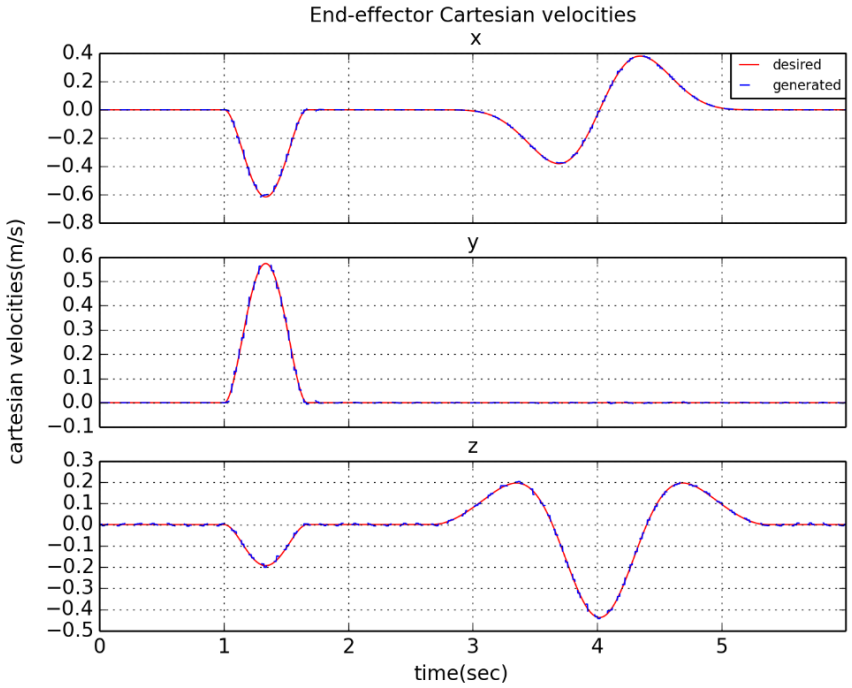


Figure 5.7: End effector cartesian velocities

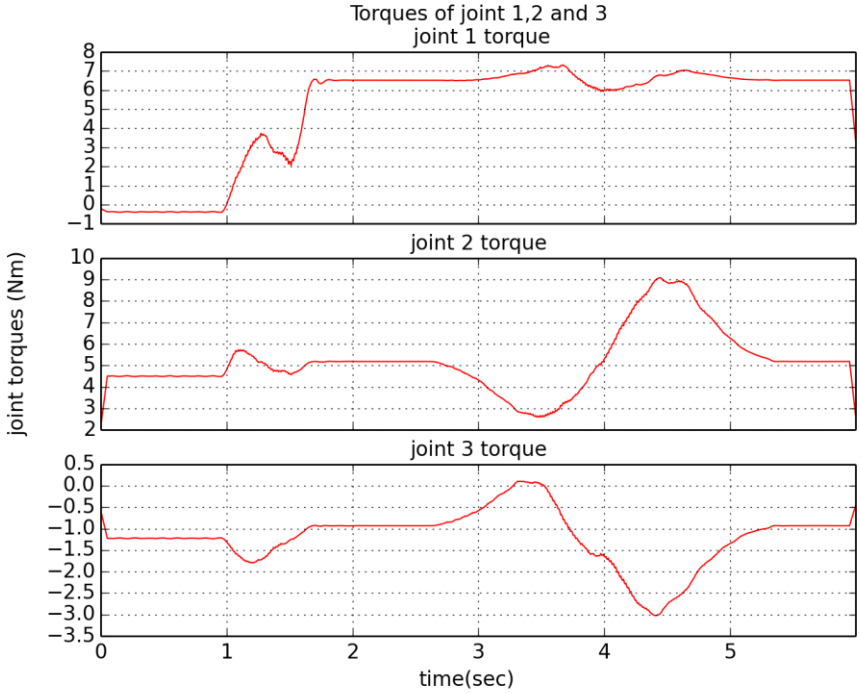


Figure 5.8: Torques of the robot joints 1,2 and 3

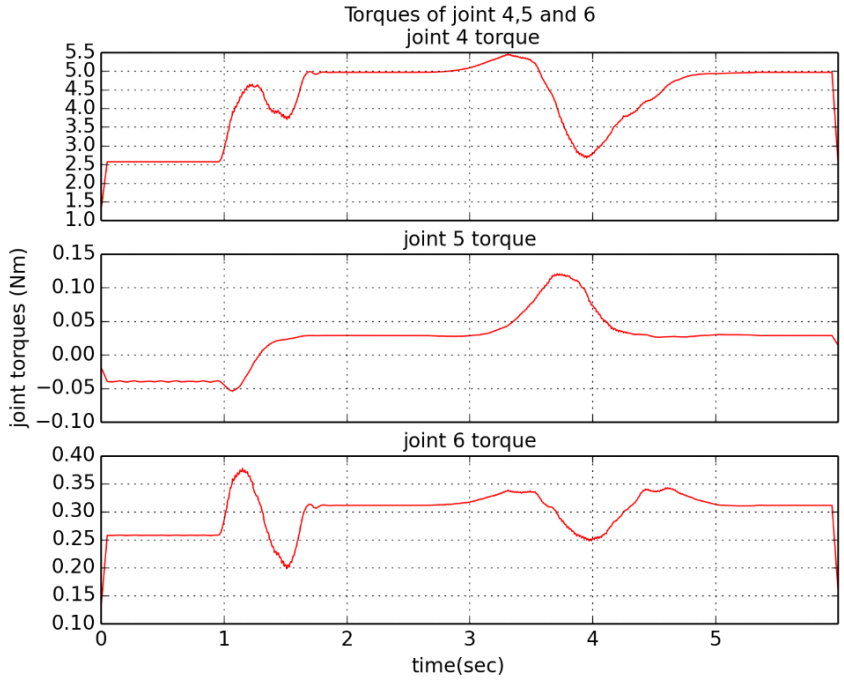
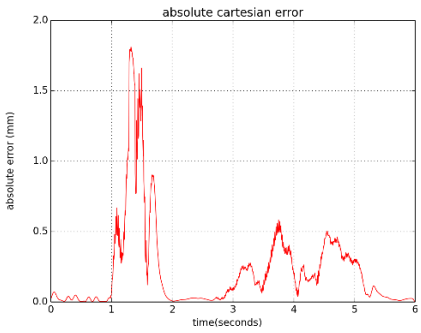


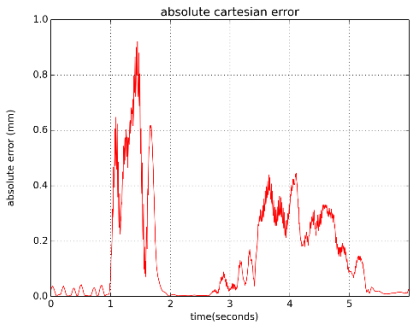
Figure 5.9: Torques of the robot joints 5,6,7

Table 5.1: Errors of ITECH Arm, 250 Hz controller frequency

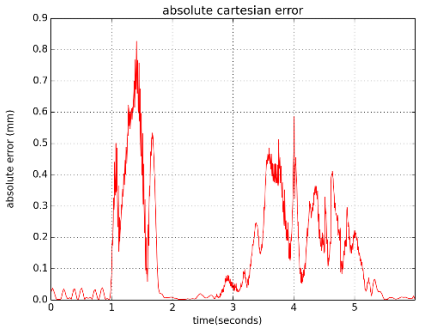
Position and Velocity Gains at 250 Hz Controller Frequency	Integral Absolute Cartesian Error	Maximum Absolute Cartesian Error	Integral Absolute Orientation Error	Maximum Orientation Error
KP=600.0, KD=50.0	4.9997	1.8122	0.5623	0.2417
KP=800.0, KD=50.0	4.9228	0.9208	0.4842	0.1965
KP=900.0, KD=50.0	4.8944	0.8263	0.4691	0.1836
KP=900.0, KD=25.0	4.9808	1.2594	0.5235	0.2218
KP=900.0, KD=75.0	4.9105	0.9421	0.4627	0.1531
KP=1000.0, KD=50.0	4.9935	1.0856	0.4372	0.1571
KP=1200.0, KD=50.0	4.9086	1.2423	0.4358	0.1313
KP=1500.0, KD=50.0	4.9514	1.3380	0.4218	0.1164
KP=2000.0, KD=50.0	5.0827	0.6766	0.5144	0.1179


absolute cartesian error
absolute error (mm)
time(seconds)

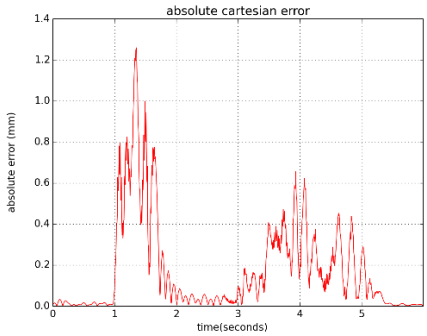
a)


absolute cartesian error
absolute error (mm)
time(seconds)

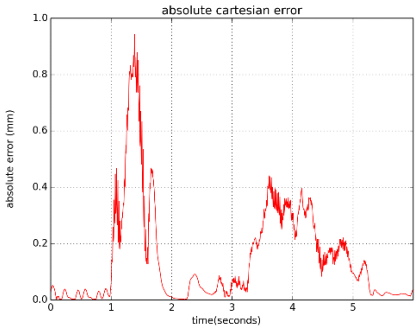
b)


absolute cartesian error
absolute error (mm)
time(seconds)

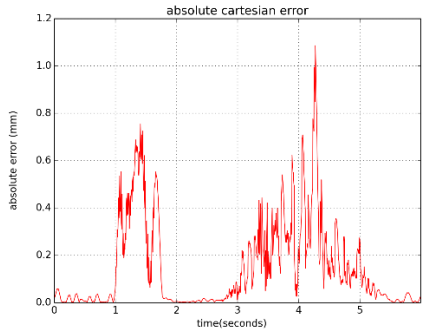
c)


absolute cartesian error
absolute error (mm)
time(seconds)

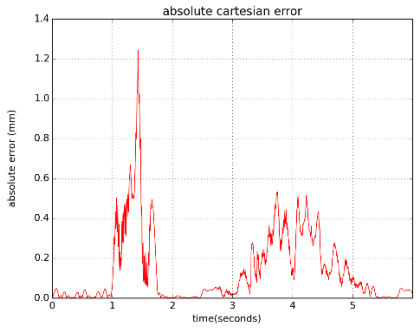
d)


absolute cartesian error
absolute error (mm)
time(seconds)

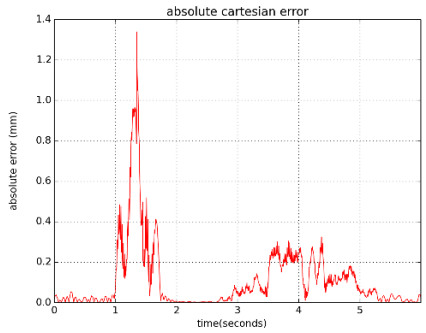
e)


absolute cartesian error
absolute error (mm)
time(seconds)

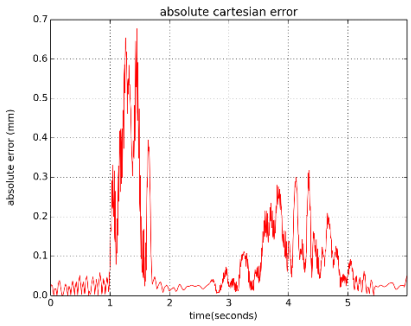
f)


absolute cartesian error
absolute error (mm)
time(seconds)

g)


absolute cartesian error
absolute error (mm)
time(seconds)

h)



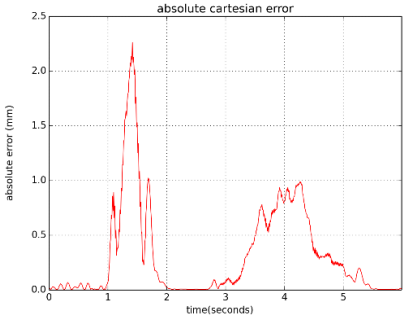
i)

Figure 5.10: End effector absolute position errors

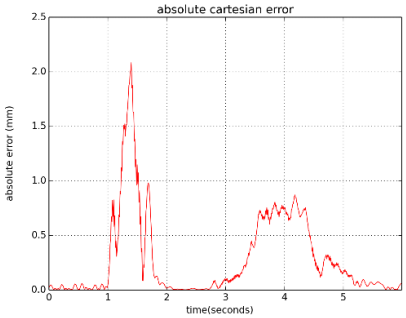
a) KP=600.0, KD=50.0, b) KP=800.0, KD=50.0, c) KP=900.0, KD=50.0, d) KP=900.0, KD=25.0, e) KP=900.0, KD=75.0, f) KP=1000.0, KD=50.0, g) KP=1200.0, KD=50.0, h) KP=2000.0, KD=50.0 at 250 Hz

Table 5.2: Errors of ITECH manipulator,100 Hz controller frequency

Position and Velocity Gains at 100 Hz Controller Frequency	Integral Absolute Cartesian Error	Maximum Absolute Cartesian Error	Integral Absolute Orientation Error	Maximum Orientation Error
KP=600.0, KD=50.0	5.0953	2.2604	0.7249	0.3602
KP=700.0, KD=50.0	5.1176	2.0833	0.6346	0.3341
KP=800.0, KD=25.0	5.8569	2.1440	0.9423	0.3538
KP=800.0, KD=50.0	5.1462	2.0063	0.5932	0.2690
KP=800.0, KD=75.0	6.3147	2.1931	1.0645	0.2455
KP=900.0, KD=50.0	5.1673	1.9925	0.5970	0.2563
KP=2000.0, KD=50.0	Unstable	-	-	-



a)



b)

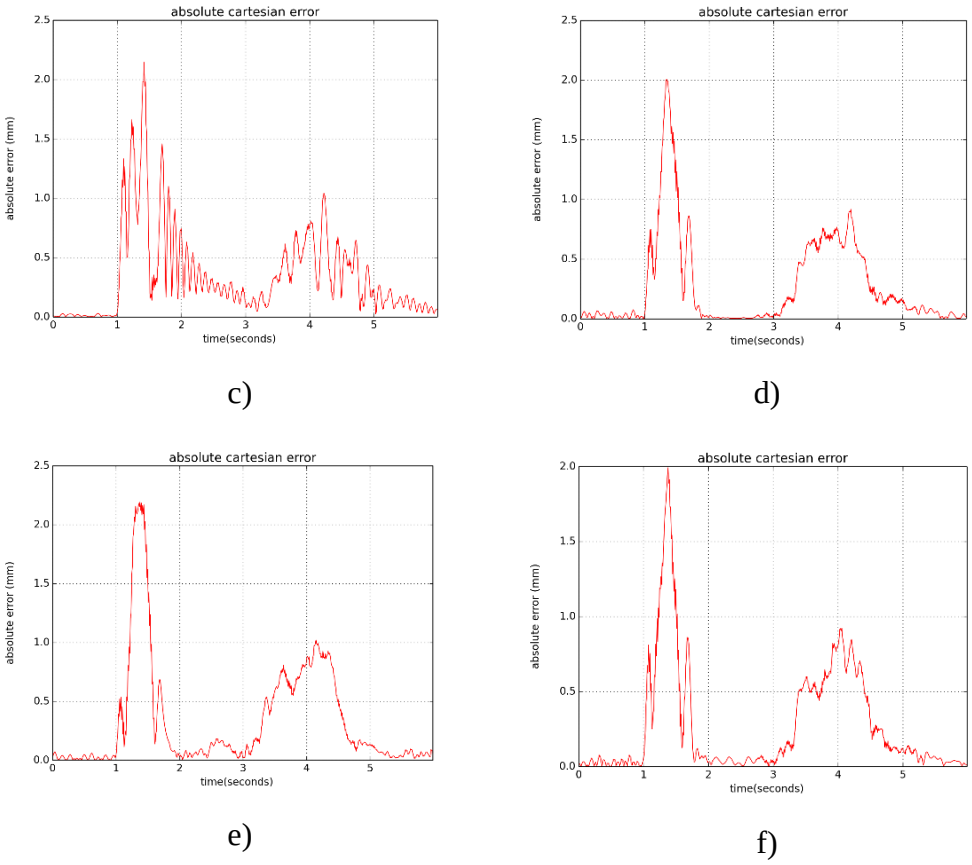


Figure 5.11: End effector absolute position errors

a) KP=600.0, KD=50.0, b) KP=700.0, KD=50.0, c) KP=800.0, KD=25.0, d) KP=800.0, KD=50.0, e) KP=800.0, KD=75.0, f) KP=900.0, KD=50.0 at 100 Hz

Table 5.3: Errors of ITECH manipulator, 50 Hz controller frequency

Position and Velocity Gains at 50 Hz Controller Frequency	Integral Absolute Cartesian Error	Maximum Absolute Cartesian Error	Integral Absolute Orientation Error	Maximum Orientation Error
KP=100.0, KD=10.0	16.6974	15.4748	6.3882	2.9924
KP=200.0, KD=10.0	15.7356	12.5877	6.1531	2.9984
KP=200.0, KD=25.0	5.5180	4.6929	2.1253	1.2157
KP=200.0, KD=50.0	Unstable	-	-	-
KP=300.0, KD=10.0	Unstable	-	-	-

5.2.1 Discussion

From the tables and figures, it is clearly seen that, higher controller frequencies yield better results. Although the maximum errors change for the same gain set in different simulation trials due to the non-real time operating system, it can still give idea on the performance of the controller. However, the error integral is a more reliable performance criteria than the maximum error. In 250 Hz, gain sets KP=900.0, KD=75.0 and KP=1200.0, KD=50.0 give the best results. For position feedback, setting the KP and KD 900.0 and 50.0 and for orientation feedback, 1200.0 and 75.0 would be the best choice for the controller. Using the same method, best choices of gain sets for the controller in different frequencies can be generated.

5.3 Simulations with Unknown Payload Results

Dynamic simulations with unknown payload carried by the end effector of the robot are conducted. The simulation results for KP=1200, KD=50 and 250 Hz controller frequency are given in from (Figure 5.12) to (Figure 5.20) and from Table 5.4 to Table 5.6.

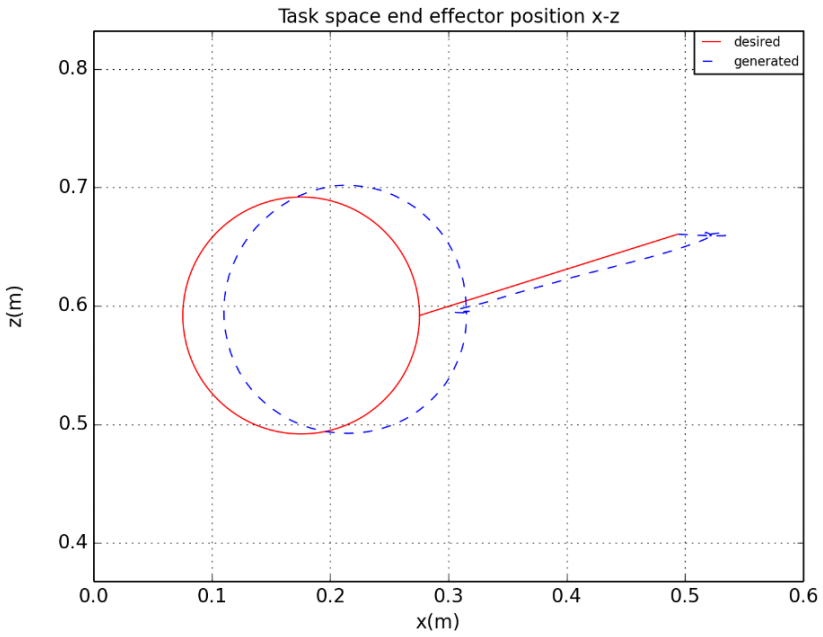


Figure 5.12: End effector position in x-y, 2 kg Payload

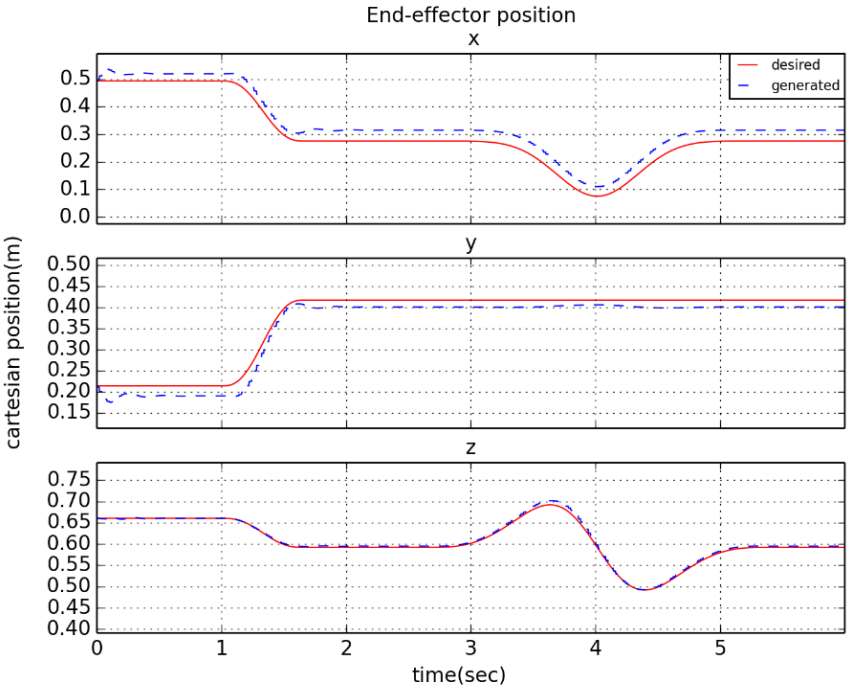


Figure 5.13: End effector position, 2 kg Payload

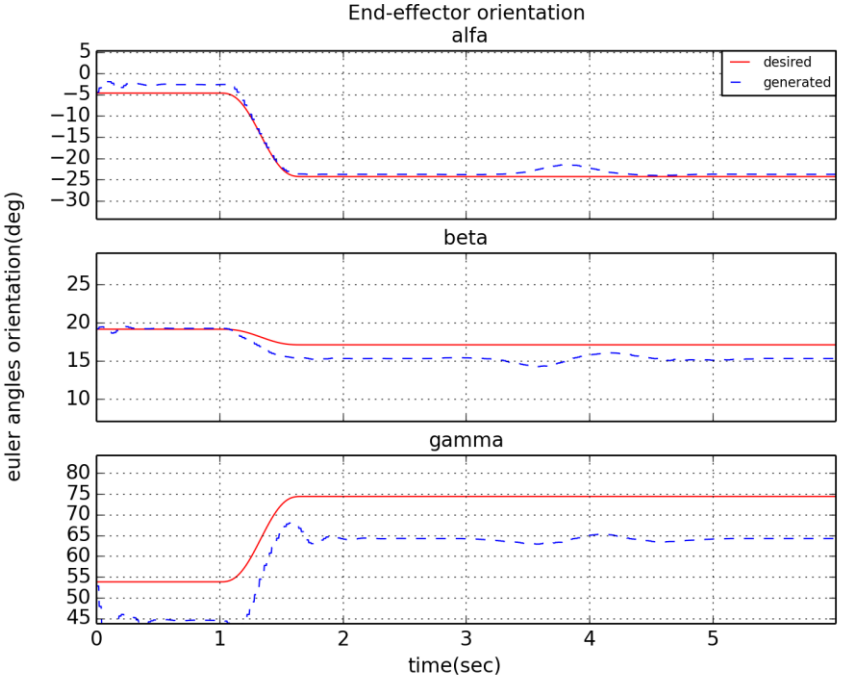


Figure 5.14: End effector orientation, 2 kg Payload

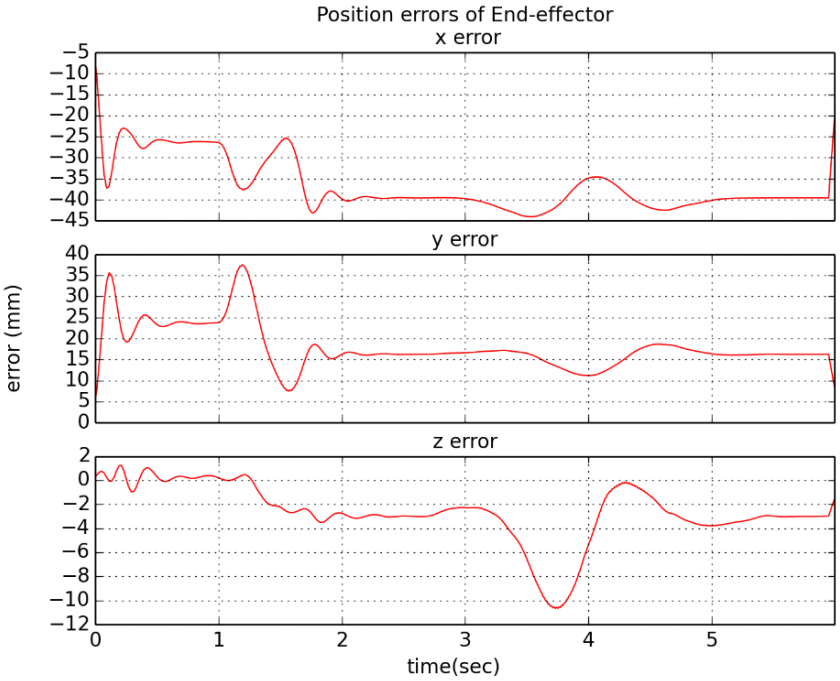


Figure 5.15: End effector position error, 2 kg Payload

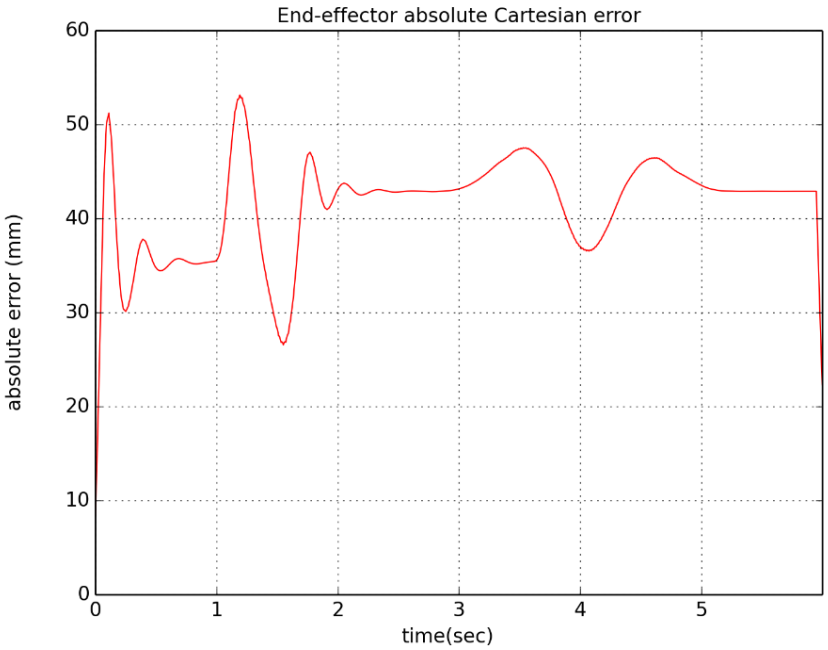


Figure 5.16: End effector absolute position error, 2 kg Payload

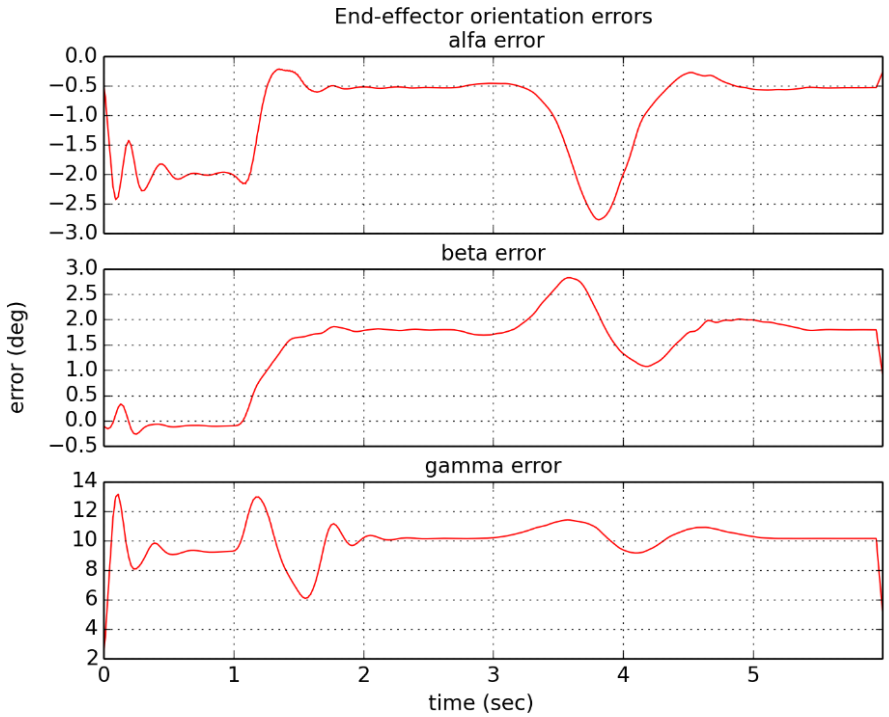


Figure 5.17: End effector orientation error, 2 kg Payload

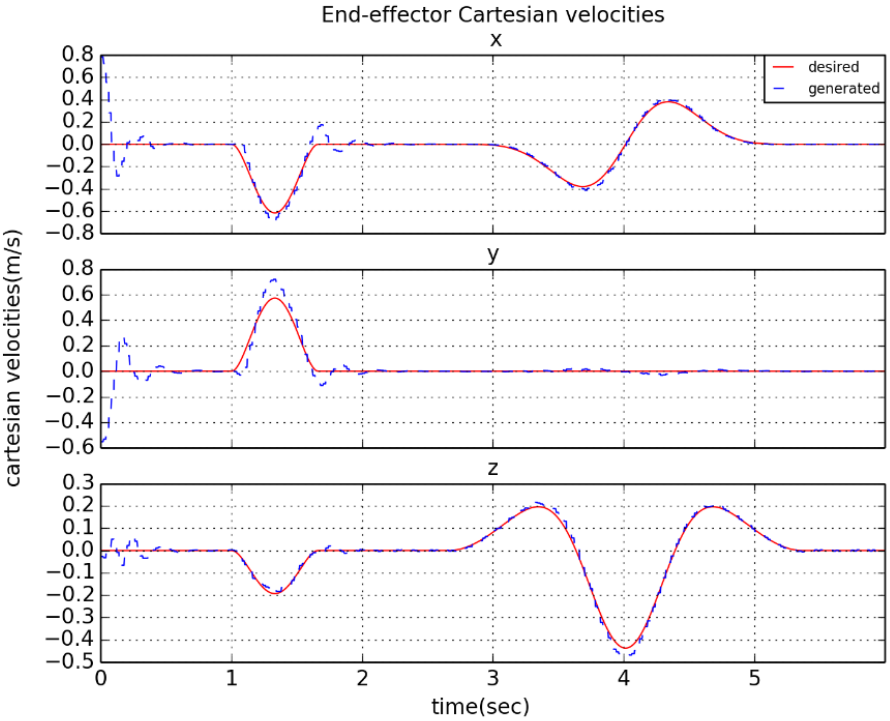


Figure 5.18: End effector cartesian velocities, 2 kg Payload

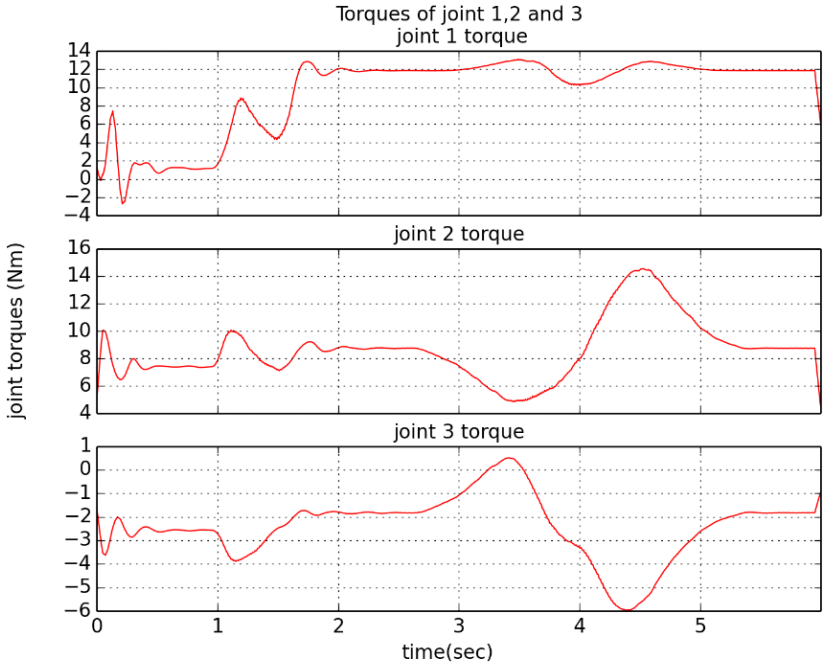


Figure 5.19: Torques of robot joints 1,2 and 3, 2 kg Payload

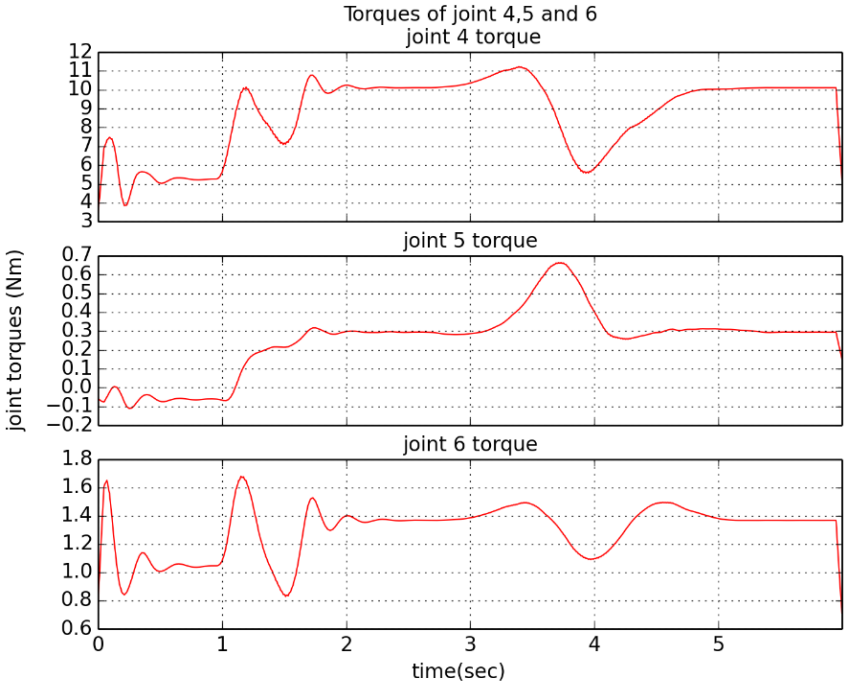


Figure 5.20: Torques of robot joints 4,5 and 6, 2 kg Payload

Table 5.4: Errors of ITECH Arm, 250 Hz controller frequency**,** 2 kg payload

Position and Velocity Gains at 250 Hz Controller Frequency	Integral Absolute Cartesian Error	Maximum Absolute Cartesian Error	Integral Absolute Orientation Error	Maximum Orientation Error
KP=900.0, KD=50.0 with 2000 gr Payload	321.5292	66.5428	96.5249	16.0400
KP=1200.0, KD=50.0 with 2000 gr Payload	248.7201	53.0033	74.7592	13.2540

Table 5.5: Errors of ITECH Arm, 100 Hz controller frequency**,** 2 kg payload

Position and Velocity Gains at 100 Hz Controller Frequency	Integral Absolute Cartesian Error	Maximum Absolute Cartesian Error	Integral Absolute Orientation Error	Maximum Orientation Error
KP=900.0, KD=50.0 with 2000 gr Payload	398.5052	81.01961	119.5790	19.1733

Table 5.6: Errors of ITECH Arm, 50 Hz controller frequency**,** 2 and 1 kg payload

Position and Velocity Gains at 50 Hz Controller Frequency	Integral Absolute Cartesian Error	Maximum Absolute Cartesian Error	Integral Absolute Orientation Error	Maximum Orientation Error
KP=200.0, KD=25.0 with 2000 gr Payload	Unstable	-	-	-
KP=200.0, KD=25.0 with 1000 gr Payload	618.6905	122.8586	184.3349	28.7477

5.3.1 Discussion

When the ‘with and without payload’ cases are compared, the payload reduces the performance of the manipulator. In 250 Hz and 100 Hz, the manipulator could handle the load while in 50 Hz, the system become unstable. From the Table 5.4, it can be seen that higher proportional gain yield significantly better results with the existence of unknown payload as greater effort is produced for the same amount of error and it helped to overcome the weight of the payload. As in the previous case, higher controller frequencies yield better result. It should be noted that, with 1:140 harmonic drive transmission and frictions, the effect of payload would be significantly different than the simulation. The payload simulation is an effort to test the controller with existence of disturbance. In (Figure 5.19) and (Figure 5.20), the joint torques are within the limits given in Table 1.1, thus we can conclude that in such a motion, 2 kg payload can be handled by the ITECH Arm.

5.4 Simulation for Pick and Place Task Results

In this task, ITECH Arm starts at an initial condition, proceeds to object and carries the object to target location. Finally, the robot arm goes to the initial condition, completing the task. The simulation is done with KP=1000.0, KD=50.0 and 250 Hz controller frequency.

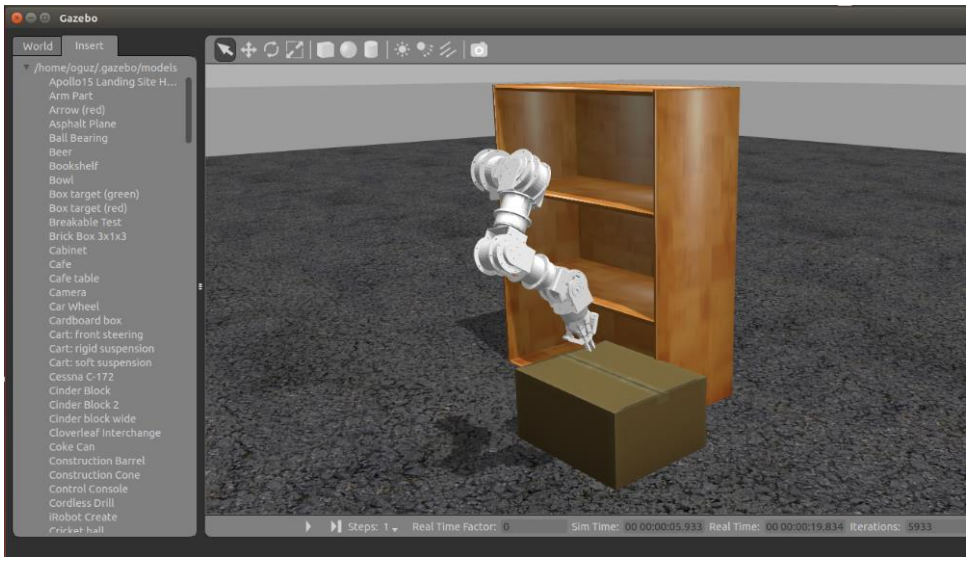


Figure 5.21: Screenshot of pick and place in Gazebo

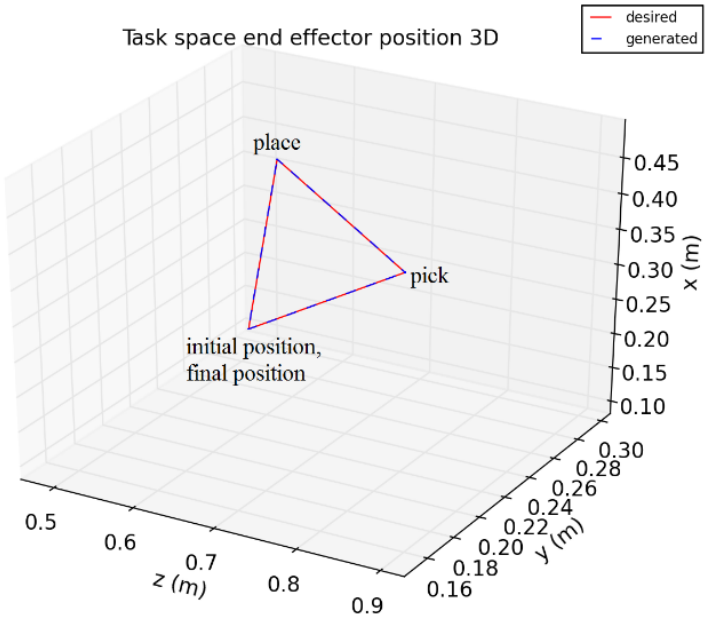


Figure 5.22: End effector position in x-y-z, pick and place

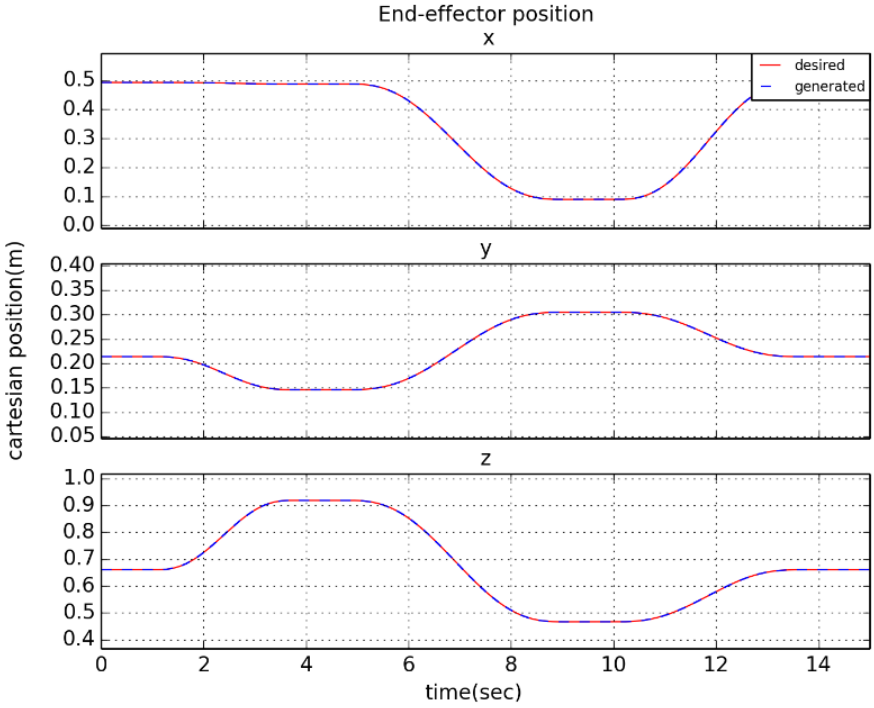


Figure 5.23: End effector position, pick and place

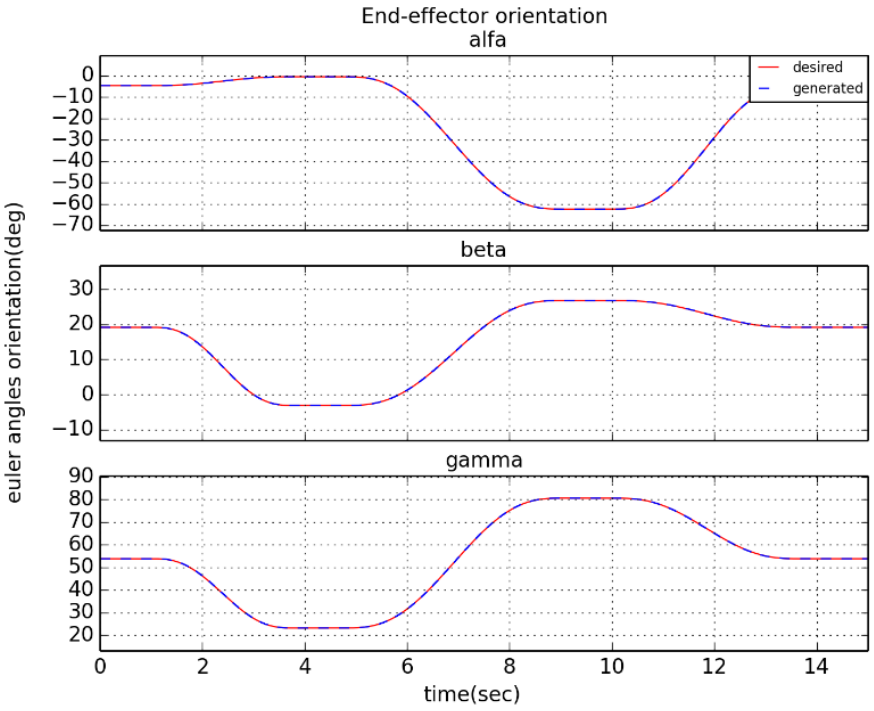


Figure 5.24: End effector orientation, pick and place

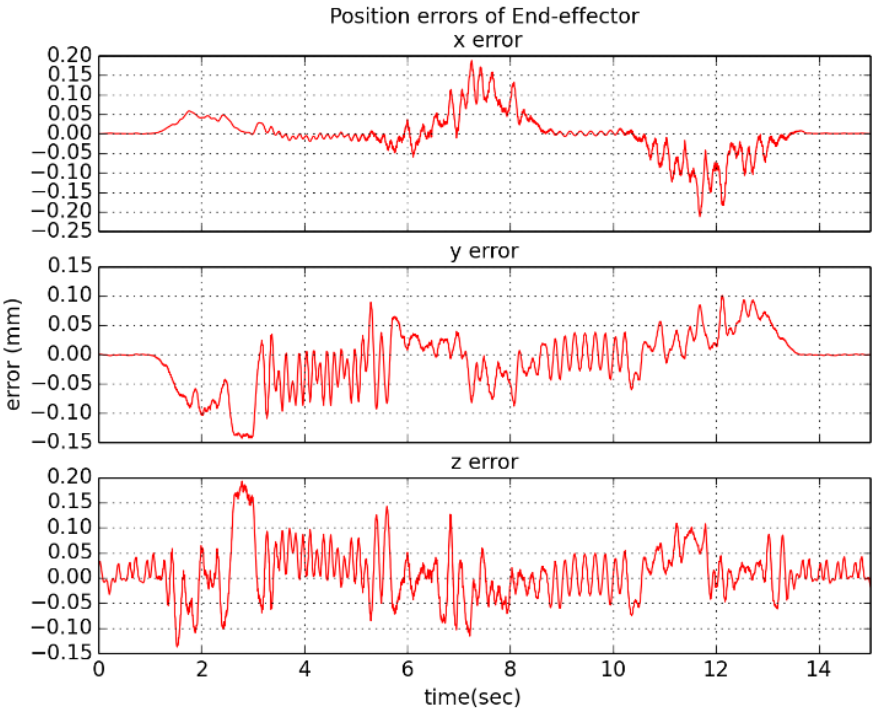


Figure 5.25: End effector position error, pick and place

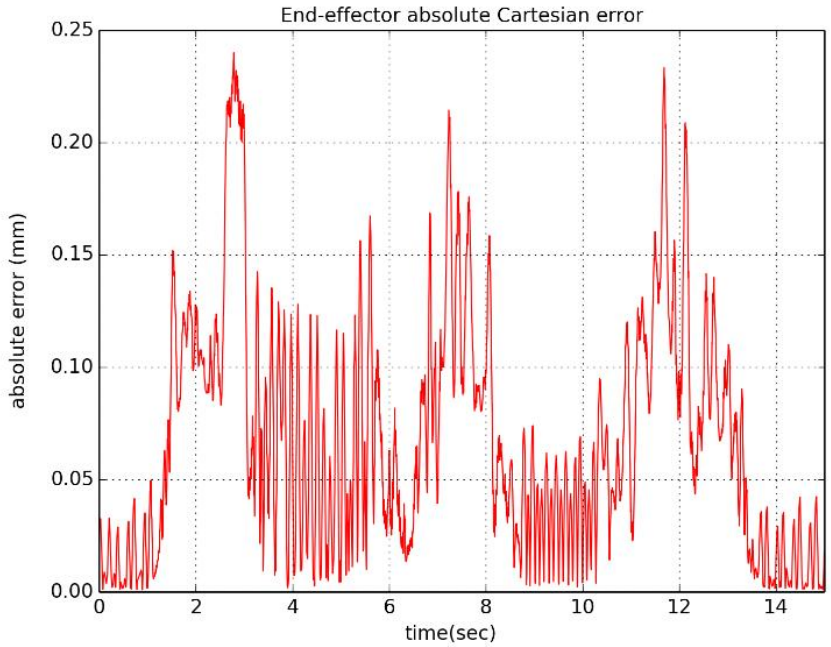


Figure 5.26: End effector absolute cartesian error, pick and place

Figure 5.27: End effector orientation error, pick and place

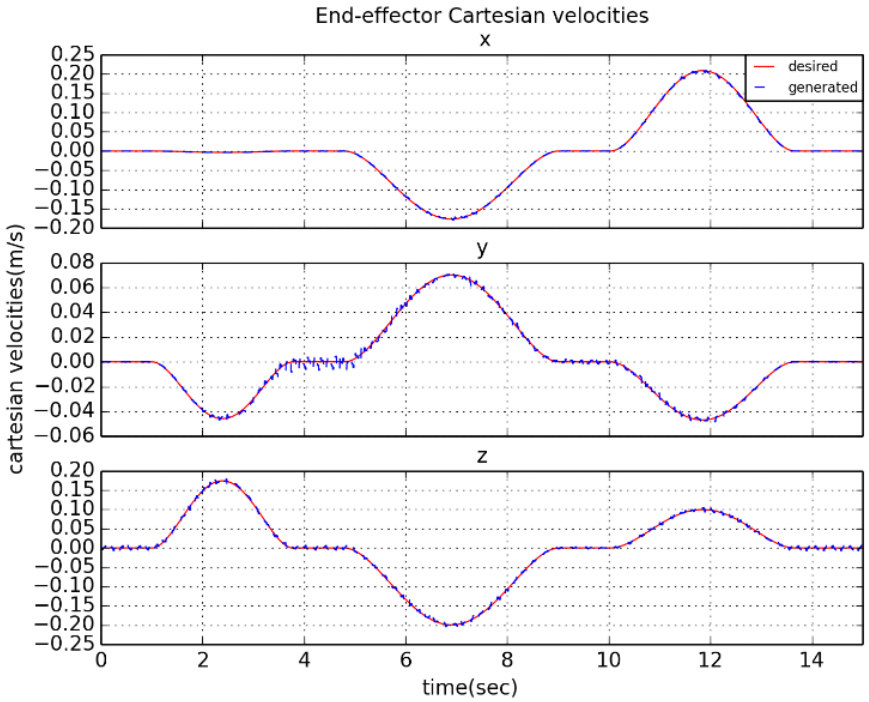


Figure 5.28: End effector cartesian velocities, pick and place

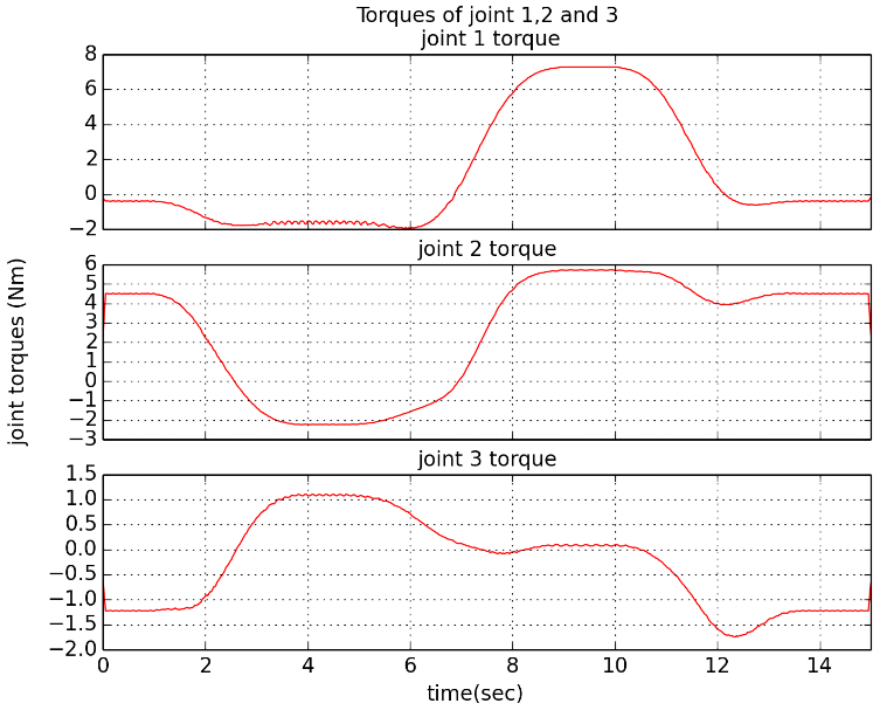


Figure 5.29: Torques of robot joints 1,2 and 3, pick and place

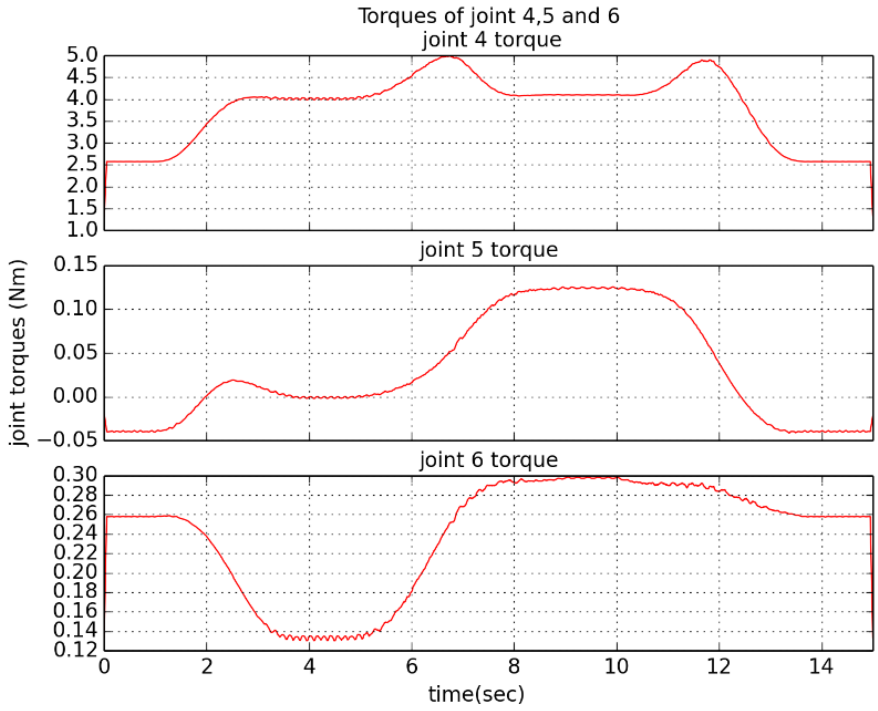


Figure 5.30: Torques of robot joints 4,5 and 6, pick and place

Table 5.7: Errors of ITECH Arm, 250 Hz controller frequency**,** pick and place

Position and Velocity Gains at 250 Hz Controller Frequency	Integral Absolute Cartesian Error	Maximum Absolute Cartesian Error	Integral Absolute Orientation Error	Maximum Orientation Error
KP=1000.0, KD=50.0	7.156	0.240	1.691	0.072

5.4.1 Discussion

In the pick and place task, the maximum velocity and acceleration of the manipulator are decreased to improve the accuracy. The maximum errors are significantly lower than the previous scenarios. Considering the operation time, integral error is also lower for unit time period. Due to the controller frequency, there are differences between the velocity in the controller loop and the actual velocity in time *t*. This causes differences between the forces acting on joint such as Coriolis and centrifugal forces. When the accelerations are lower, this difference decreases, improving the performance in terms of the selected criteria.

6. CONCLUSION

The purpose of this thesis was to integrate the ITECH 6-DOF Humanoid Manipulator to ROS and Gazebo environment, write a generic object oriented kinematics & dynamics library and implement task space inverse dynamics control algorithm in dynamic simulation environment. The control algorithm code that is written in python and used in Gazebo simulator can be directly implemented to the real robot that runs ROS.

In kinematics, numerical geometric Jacobian and numerical transformation and rotation matrices are used. In dynamics, Newton-Euler formulation is preferred due to its computational efficiency compared to Euler-Lagrange formulation [15].

Dynamic simulations are conducted in Gazebo with gazebo-ros control interface. In different controller frequencies, various pose and velocity gains are tested in an effort to find a suitable gain set with respect to pose error. As the controller and simulator are both run in a non-real time operating system and communicating over a node network, there were small differences in the results of the same gain sets and controller frequencies. Matlab SimMechanics software platform, which offers various transmission properties such as friction and elastic joints that does not exist in Gazebo yet, can be a better alternative in developing controller algorithms. SimMechanics. This platform also delivers more precise results. One can also generate his own simulation environment using Rigid Body Dynamics algorithms [16] and define custom friction models.

In simulations, it is observed that the controller frequency dramatically effects the control performance. Higher controller frequencies were stable for wider range of gain sets and more robust to disturbances such as payloads carried by end effector of the robot and yielded better performance in terms of integral absolute pose error. It is concluded that a fast communication plays a critical role in maintaining high controller

frequencies in a real robotic system. While CAN Bus communication has been used in industrial manipulators for a long time [17], EtherCAT network communication provides data rates higher than 100 Mbits/s [18] and is currently used in robot platforms such as KUKA youBot [19] and Justin Robot [20]. This communication network technology can be implemented on ITECH robot.

With integration of a force/torque sensor on the wrist of the robot, mass of the payload can be estimated and the inverse dynamics control law can be updated for this additional mass. As ITECH is planned to be a humanoid manipulator, control architectures such as impedance control can be implemented for better human-robot interaction.

REFERENCES

[1] J. S. Luh , M. W. Walker and R. P. C. Paul, “On-Line Computational Scheme for,” *Journal of Dynamic Systems, Measurement, and Control ,* 1980.

[2] R. Tedrake, «Underactuated Robotics,» 2016. [Çevrimiçi]. Available: http://underactuated.csail.mit.edu/underactuated.html?chapter=26. [Erişildi: 15 11 2016].

[3] S. Ivaldi, J. Peters, V. Padois ve F. Nori, «Tools for simulating humanoid robot dynamics: a survey,» *IEEE-RAS International Conference on Humanoid Robots ,* 2014.

[4] «Features: Coppelia Robotics,» Coppelia Robotics, [Çevrimiçi]. Available: http://www.coppeliarobotics.com/features.html. [Erişildi: 20 11 2016].

[5] O. Kaya, «INSANSI ROBOT KOLUNUN OPT˙IMIZASYONU VE DINAMIK ANALIZI,» Istanbul Technical University, Institute of Science, 2015.

[6] D. Thomas, «ROS Documentation,» Open Source Robotics Foundation, 2014. [Çevrimiçi]. Available: http://wiki.ros.org/ROS/Introduction. [Erişildi: 2016].

[7] B. Siciliano, L. Sciavicco, L. Villani ve G. Oriolo, Robotics: Modeling, Planning and Control, Springer, 2009.

[8] L.-W. Tsai, Robot Analysis: The Mechanics of Serial and Parallel Manipulators, Wiley, 1999.

[9] M. W. Spong, S. Hutchinson ve M. Vidyasagar, Robot Modeling and Control, John Wiley & Sons, 2006.

[10] S. Chiaverini, B. Sicilliano ve O. Egeland, «Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator,» *IEEE Transactions on Control Systems Technology ,* cilt 2, no. 2, Jun 1994.

[11] «ROS Documentation,» Willow Garage, [Çevrimiçi]. Available: http://wiki.ros.org/ROS/Introduction. [Erişildi: 22 11 2016].

[12] J. M. O’Kane, A Gentle Introduction to ROS, Jason Matthew O’Kane, 2016.

[13] Gazebosim, «New Feature Highlight: Multiple Physics Engines,» Open Source Robotics Foundation, 2014. [Çevrimiçi]. Available: http://gazebosim.org/blog/feature_physics. [Erişildi: 25 9 2016].

[14] D. Coleman, «Tutorials:Ros Control,» Gazebosim, 2013. [Çevrimiçi]. Available: http://gazebosim.org/tutorials/?tut=ros_control.

[15] W. Khalil ve E. Dombre, Modeling, Identification and Control of Robots, Elsevier, 2004.

[16] R. Featherstone, Rigid Body Dynamics Algorithms, Springer, 2014.

[17] KUKA, «KUKA Robotics,» KUKA Robotics, 2000. [Çevrimiçi]. Available: http://www.kuka-robotics.com/en/pressevents/productnews/print/NP_000508_Truetzschler.htm. [Erişildi: 5 10 2016].

[18] EtherCAT, «Ethercat,» EtherCAT, [Çevrimiçi]. Available: EtherCAT. [Erişildi: 6 10 2016].

[19] KUKA, «KUKA Healthcare,» KUKA Robotics, [Çevrimiçi]. Available: http://www.kuka-healthcare.com/NR/rdonlyres/4833867A-D24F-410A-9AE4-300FBF671DFD/0/youBot_datenblatt_web_0514.pdf. [Erişildi: 5 10 2016].

[20] PC-Control, «ethercat,» DLR, 2010. [Çevrimiçi]. Available: https://www.ethercat.org/download/documents/pcc_0210_dlr_e.pdf. [Erişildi: 8 10 2016].

APPENDICES

APPENDIX A: Geometric properties of Itech Arm

APPENDIX B: Mass properties of Itech Arm

APPENDIX C: Itech Arm Software Library classes and functions

APPENDIX A

The length of links (m):

Base: 0.16496

Arm: 0.219

Forearm: 0.213

Hand: 0.21718

In this thesis, Robotiq 2-Finger Adaptive Gripper is used as hand. The properties would change once an end effector is determined and mounted. The transformation matrices of the robot are generated inside the kinematics algorithm of ITECH Arm, using D-H parameters.

APPENDIX B

Masses of the links (kg):

Mass 1: 1.7621
Mass 2: 1.3727
Mass 3: 1.2878
Mass 4: 1.2701
Mass 5: 1.2878
Mass 6: 0.5465

Link centre of mass vectors for the corresponding link coordinate frames (mm):

$$CoM = [CoM_x \quad CoM_y \quad CoM_z]$$

$$CoM_1 = [0.00 \quad 12.20 \quad 0.30]$$
$$CoM_2 = [0.37 \quad -2.26 \quad 84.80]$$
$$CoM_3 = [-0.40 \quad 8.97 \quad -2.26]$$
$$CoM_4 = [0.35 \quad 3.40 \quad 83.38]$$
$$CoM_5 = [0.40 \quad 8.97 \quad 2.26]$$
$$CoM_6 = [-154.06 \quad 0.0 \quad -7.36]$$

Link inertia matrices for the corresponding link coordinate frames (gr.mm^2)

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

$$I_1 = \begin{bmatrix} 3525915 & 444 & 34 \\ 444 & 2019491 & 4861 \\ 34 & -48461 & 3776165 \end{bmatrix}$$

$$I_2 = \begin{bmatrix} 3924797 & 4315 & -1241 \\ 4315 & 3371160 & 274684 \\ -1241 & 274684 & 1643140 \end{bmatrix}$$

$$I_3 = \begin{bmatrix} 1964360 & 4451 & 14951 \\ 4451 & 1316183 & 8007 \\ 14951 & 8007 & 1695363 \end{bmatrix}$$

$$I_4 = \begin{bmatrix} 3305476 & -9407 & -3142 \\ -9407 & 2824665 & -361337 \\ -3142 & -361337 & 1478336 \end{bmatrix}$$

$$I_5 = \begin{bmatrix} 1964360 & -4451 & 14951 \\ -4451 & 1316183 & -8007 \\ 14951 & -8007 & 1695363 \end{bmatrix}$$

$$I_5 = \begin{bmatrix} 810649 & 68 & 274614 \\ 68 & 2392078 & 22 \\ 274614 & 22 & 2015807 \end{bmatrix}$$

APPENDIX C

Itech Arm kinematic functions library

Functions:

get_rotation (rotation vector, angle):

Arguments: A unit vector and an angle

Returns: A 3x3 rotation matrix with the unit vector rotated by the angle

transformation_to_pose (Transformation):

Arguments: 4x4 transformation matrix

Returns: 6x1 pose of the robot

Itech Arm kinematics and dynamics library

Classes:

Link (self, DH, r_centre_of_mass, mass, inertia):

Link class arguments: Denavit-Hartenberg parameters, vector of centre of mass, link mass, inertia matrix

Link class functions:

rotation (self):

Returns: 3x3 rotation matrix of the link for corresponding angle attribute of link object

inverse_rotation(self):

Returns: 3x3 inverse rotation matrix of the link for corresponding angle and D-H attributes of the link object

A(self):

Returns: 4x4 transformation matrix of the link for corresponding angle and D-H attributes of the link object

z_axis(self):

Returns: 3x1 z vector of the link for corresponding D-H attribute of the link object

Robot(self, links):

Robot class arguments: Link objects generated from Link class

Robot class functions:

get_forward_geometry(self):

Returns: 4x4 transformation matrix of the robot

get_pose_vector(self):

Returns: 6x1 pose vector of the robot

get_orientation_T(self):

Returns: 3x3 orientation velocity to task space angular velocity transformation matrix

get_orientation_T(self):

Returns: 3x3 derivative of orientation velocity to task space angular velocity transformation matrix

get_jacobian(self):

Returns: The 6xn Jacobian matrix of the robot

get_torques(self):

Returns: The array of torques of the robot from the corresponding joint positions, velocities and accelerations attributes

get_controller_torques(self, control signal):

Arguments: control signal for the inverse dynamics controller

Returns: The array of torques of the robot from the corresponding joint positions, velocities attributes and the control signal

get_jacobian_derivative(self):

Returns: The numerical Jacobian derivative of the robot

CURRICULUM VITAE

Name-Surname :Oğuzhan Cebe

Date and Place of Birth :24.07.1992, İZMİR TURKEY

E-Mail : oguzhancebe@gmail.com

EDUCATION:

- **B.Sc.** : 2014, İzmir Institute of Technology, Faculty of Engineering, Mechanical Engineering

EXPERIENCES

- 2015-2016, Robotics and Control Engineer at BAÇ Engineering, Machinery and Automation

PUBLICATIONS

- Aykut Beke, Mustafa Saraoğlu, Gürtaç Kadem, Oğuzhan Cebe, Ramazan Gökay, Volkan Sezer "Vehicle Platoon Control: Consensus Based Approach" Turkey Automatic Control National Committee, 2016