Graduate School ETD Form 9
(Revised 01/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Garrett Scot Yesmunt

Entitled DESIGN, ANALYSIS, AND SIMULATION OF A HUMANOID ROBOTIC ARM APPLIED TO CATCHING

For the degree of Master of Science in Mechanical Engineering

Is approved by the final examining committee:

Tamer Wasfy

Hazim El-Mounayri

Ali Razban

To the best of my knowledge and as understood by the student in the *Thesis/Dissertation Agreement. Publication Delay, and Certification/Disclaimer (Graduate School Form 32)*, this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

Approved by Major Professor(s): Tamer Wasfy

Approved by: Jie Chen 04/25/2014

Head of the Department Graduate Program Date

DESIGN, ANALYSIS, AND SIMULATION OF A HUMANOID ROBOTIC ARM APPLIED TO CATCHING

A Thesis

Submitted to the Faculty

of

Purdue University

by

Garrett Scot Yesmunt

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Mechanical Engineering

May 2014

Purdue University

Indianapolis, Indiana

This thesis is dedicated to my Family.

ACKNOWLEDGMENTS

First and foremost, I would like to thank Dr. Tamer Wasfy for his guidance and instruction. Over the years, his expertise has helped me to develop my own foundation of understanding engineering analysis and problem solving. He taught technical skills which enabled me to explore new areas of engineering.

Dr. Hazim El-Mounayri has given me solid support through several years of classes and research projects. I am thankful for his support and for the time I have had to work with him.

I would like to thank Dr. Ali Razban. His encouragement in the area of robotics and control systems was invaluable. His class on robotics equipped me to further study this area.

Finally, I would like to thank Lukas Bearden and Nathan Fitzpatrick for their solid support and commitment. I will always remember their true friendship and encouragement through the last several years. I have and will continue to enjoy the ride with you.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Yesmunt, Garrett Scot. M.S.M.E., Purdue University, May 2014. Design, Analysis, and Simulation of a Humanoid Robotic Arm Applied to Catching. Major Professor: Tamer Wasfy.

There have been many endeavors to design humanoid robots that have human characteristics such as dexterity, autonomy and intelligence. Humanoid robots are intended to cooperate with humans and perform useful work that humans can perform. The main advantage of humanoid robots over other machines is that they are flexible and multi-purpose. In this thesis, a human-like robotic arm is designed and used in a task which is typically performed by humans, namely, catching a ball. The robotic arm was designed to closely resemble a human arm, based on anthropometric studies. A rigid multibody dynamics software was used to create a virtual model of the robotic arm, perform experiments, and collect data. The inverse kinematics of the robotic arm was solved using a Newton-Raphson numerical method with a numerically calculated Jacobian. The system was validated by testing its ability to find a kinematic solution for the catch position and successfully catch the ball within the robot's workspace. The tests were conducted by throwing the ball such that its path intersects different target points within the robot's workspace. The method used for determining the catch location consists of finding the intersection of the ball's trajectory with a virtual catch plane. The hand orientation was set so that the normal vector to the palm of the hand is parallel to the trajectory of the ball at the intersection point and a vector perpendicular to this normal vector remains in a constant orientation during the catch.

It was found that this catch orientation approach was reliable within a 0.35 x 0.4 meter window in the robot's workspace. For all tests within this window, the robotic arm successfully caught and dropped the ball in a bin. Also, for the tests within this window, the maximum position and orientation (Euler angle) tracking errors were 13.6 mm and 4.3°, respectively. The average position and orientation tracking errors were 3.5 mm and 0.3°, respectively. The work presented in this study can be applied to humanoid robots in industrial assembly lines and hazardous environment recovery tasks, amongst other applications.

1. INTRODUCTION

Modern robotics is a widely studied field which has existed for many decades. Robots have been used in many different applications, such as autonomous vehicles, robot-assisted surgery, and manufacturing robots, and the list of applications grows with time. Robotics has given rise to many opportunities across all industries. While visions of futuristic robots are common and celebrated, most robotics research attempts have failed to produce machines which rival these visions of highly sophisticated and autonomous machines. However, significant progress has been made over the last two decades.

One segment studied within the robotics research community is humanoid robotics. The objective of humanoid robotics is to develop human-like machines which assist and protect people. Several research groups are involved in humanoid mechanics and control systems, and there are several differing approaches. These designs exist with varying performances related to dexterity, speed, and cost.

This study is an investigation into a specific aspect of humanoid robotics, namely, catching flying objects. In order to explain the background and context of this study, a brief overview of the history of humanoid robotics is included. After this, the objective of this study is explained. Finally, an overview of the approach presented in this work is outlined.

1.1 Humanoid Robotics History and Motivation

Human innovation and vision brought the idea of a robot to life long before the means to physically create one existed. In fact, the word "robot" was first introduced in a play in 1920 by Karel Capek where robots were not a mechanical creation, but a chemical and biological creation. Capek's play was *Rossum's Universal Robots*, and the robots were distinctly humanoid and existed for the purpose of helping humans. This idea has remained with people ever since: one could have a servant which was not a person and did not require the same consideration as a human being. Many attempts were made to realize this idea, which eventually led to the current electromechanical concept.

A humanoid robot is a system of robotic manipulators and controllers which resembles the human body. In recent decades, there have been many humanoid robots represented in science fiction novels and films, as well as prototypes built by corporations and labs. The fundamental reason that humans desire robotic mechanisms is to make their lives easier. The implementation of such a device would raise the quality of life, leaving the hazardous or repetitive jobs to robots. This would allow for the development of better technology to improve the standard of living.

It is useful to consider the manifestations of humanoid robotics in film and literature for the purpose of acquiring a vision for what this technology can eventually become. One of the most widely known humanoid robots in film is *C-3PO*, the chattering, golden-colored robot, in the 1977 film *Star Wars IV: A New Hope* directed by George Lucas. One of the most striking characteristics of this character is his autonomy. He does not require direct instructions to function. There are countless other examples of autonomous behaviour, and this concept is widely understood in today's world.

This autonomous existence is the ultimate goal of modern day humanoid robotics. This is because autonomous mechanisms require less work to operate and allow for human attention to be devoted to more complex tasks. However, robots are not capable of spontaneous thoughts, commands, or actions with today's technology. Every event and action in a robotic system must be triggered by some type of event, whether internal or external to the system. While spontaneity can be mimicked using algorithms with random functions, independent decision making is not currently possible. This superior manifestation of humanoid robotics can only be found in science fiction literature and films, such as Alex Proyas' 2004 film, *I, Robot* (Figure 1.1), or the 1984 film *Terminator*, by James Cameron.

Figure 1.1. Scene from Alex Proyas' 2004 film, *I, Robot*

A major discrepancy between science fiction films and today's robots is the appearance and performance of robotic movement. None of the current humanoid robot prototypes can operate with human-like agility. This gap is closing as research labs develop new systems and as supporting technology progresses.

Honda has made an investment in this field and has one of the most sophisticated humanoid robots, called ASIMO (Figure 1.2) [1]. The development of this robot started in 1986 with biped locomotion studies. As time went on, more functionality was added until Honda had a complete humanoid robotic package which could be used to study human-robot interaction. Honda continues to improve on this humanoid platform and is still a pioneer in the field of humanoid robotics.

Another significant organization which has invested in humanoid robotics is NASA. Development of human robots designed to support space exploration is ongoing and advanced prototypes, such as the Robonaut [2], have been tested on the International Space Station. Bipedal locomotion has not been the main focus of this system since its application is in space where such ability is not practical. Therefore this is not a full humanoid system. However, significant research has been done on handling

Figure 1.2. Honda's Humanoid Robot, ASIMO

dexterity and tool use ability, since the main purpose of this robot is to perform spacewalks for repairs on the space station.

Humanoid robotics is a field which continues to grow. A division of the United States Department of Defense, DARPA, has given incentives through the "Robotics Challenge" to research labs and corporations to develop autonomous robots which can perform tasks in human-engineered surroundings [3]. Starting in 2012 and ending in 2014, participating universities and labs are developing competing systems which could be used in dangerous situations in order to reduce the risk to humans. Many of these systems have human characteristics because of the requirement that they perform tasks originally done by people.

This field has a significant future with many invested parties in government and industry. The technology will continue to advance, and more concepts and ideas will change how future robotic systems will look and behave. While there may be several reasons that this research is interesting and profitable to people, the primary motivation behind this may be that humans can relate to objects which are human-like. These systems are designed to interface and behave like people. The interaction between people and robots is another large area of research. Currently, humanoid robots are most widely used in space exploration, military applications, and prosthetics development.

1.2 Literature Review

Humanoid robotics are recent objects of attention in the world of research, but there is already substantial work in this field. Studies related to the work presented in this paper are outlined in the next few sections.

1.2.1 Human-Robot Interaction

Some research has focused on the place of robotics in the lives of people and the interaction between robotics and humans. This type of research is helpful because it gives researchers and developers a vision for how robotics can be used and the effects that these types of robots could have on the future. Understanding how humans and robots will interact is important because if people cannot accept robotic behaviour and learn to work with them, the technology itself does not matter because it will never be used. In one study, the history, identifying characteristics, and areas of application of humanoid robotics are described [4]. The characteristics listed are bipedal locomotion, perception, human-robot interaction, dextrous manipulation, and adaptive behaviour. This paper explains that real-world applications of humanoid robotics are few because their capabilities are limited. However, the paper lists applications such as technology demonstrations, space missions, manufacturing, household, and robot competitions.

1.2.2 Robotic Inverse Kinematics

The main problem in robotic arm control is solving the inverse kinematics. This has been studied in depth for both general and specific robotic configurations. For systems where no closed-form (analytical) solutions exist, numerical solution strategies are used. There are several strategies such as the Secant method, the Newton-Raphson method, and optimization methods. Since the Newton-Raphson method is used in this project, the following primarily includes studies which use the Newton-Raphson method.

In a 2011 study, the Newton-Raphson method was applied to calculating the inverse kinematics of a six degree-of-freedom robotic arm [5]. This study used this control algorithm on a real welding robot arm to prove that its motion had good precision. An important note about this approach is that it uses the psuedo-inverse of the Jacobian matrix to solve for the joint angles, and that the singularity problem was avoided using the singular value decomposition (SVD) of the Householder matrix.

An effort has also been made to develop a generalized solution of inverse kinematics of robots. A study was published which presented a solution to inverse kinematics of robots with an arbritrary number of degrees-of-freedom [6]. The solution is iterative and uses a modified Newton-Raphson algorithm when the number of degrees-of-freedom is greater than six, because the Jacobian is not square and the inverse or psuedo-inverse cannot be found. In this paper, the inverse kinematics are solbed using this method for a seven degree-of-freedom robot.

While iterative numerical methods which use first derivatives, such as Newton-Raphson or Predictor-Corrector, are fast, they are not stable because they cannot converge around singular positions. Research has been done in this area by modifying the Newton-Raphson and Predictor-Corrector so that solutions can be found at singular positions [7].

In addition to general inverse kinematics methods, methods specific to grasping moving objects have been studied. In 1993, the instance of tracking an object outside the workspace was studied [8]. The primary effort in this study is to move the manipulator as close as possible to the object which is not yet within the workspace. For instances which did not have closed form solutions, the Newton-Raphson numerical method was used to solve the non-linear system of kinematic equations.

1.2.3 Humanoid Robotic Arms

Several studies have focused on designing a humanoid robotic arm. One study developed an analytic solution for the inverse kinematics of a seven degree-of-freedom humanoid robotic arm [9]. This study also focused on developing human-like motion.

In a study presented in 2005, particular attention was given to mimicking the human arm with a cable-driven robotic arm [10]. Specific aspects of this study was displacement analysis and cable tension analysis. Optimization techniques were used to solve the inverse kinematics. Another study in cable-driven robotic arms focused on kinematic and workspace analysis [11]. This study included a simulation of the humanoid robotic arm to test the developed algorithms.

An in-depth study on a humanoid robotic arm focused on developing a simulation which included kinematic analysis and control system [12]. The focus in this case was developing a control architecture which is biologically inspired.

One study which included humanoid robotic arm development was actually focused on developing a full humanoid robot, called WABIAN-2 [13]. While the primary application of this robot was walking, a full robotic arm was developed and the design is presented in this work.

1.2.4 Robotic Catching

There have been studies which focus on robots catching objects, and some of these systems are quite successful. However, they are not often designed to be humanoid, and they often use pre-existing parts and software packages. One study involved a six degree-of-freedom robot, with the requirement that the system had closed form analytical kinematics and dynamics [14]. This allows for the inverse kinematics to be directly solved without numerical methods or iterations. This simplifies the calculations and constrains the design. Successful catching with a real, physical robot was achieved in this study.

A paper in 1989 describes how course and fine tuning of a manipulator might be handled in order to accommodate catching of a moving object [15]. This study divided the problem into two parts: path planning and path tracking. This study focuses on a two degree-of-freedom planar robot. The unique aspect of this study is that the object has a highly nonlinear path, so the system cannot use any past data to determine a future location of the object.

A similar study to the one above was published in 2005 [16]. This study proposes a method of intercepting moving objects. The main aspects of this study revolve around velocity and torque constraints, as well as matching the manipulator's position and velocity to the object's position and velocity to allow for smooth grasping. This study focuses on the high-level trajectory planning for the interception of objects whose motion may be known or possibly one with random acceleration.

In a study about trajectory and path planning, a four degree-of-freedom robot is used to experiment with ball catching [17]. This research focuses on matching the trajectory of the object with the manipulator in real-time. This system could achieve consistent tracking of balls where the typical distance between the ball and manipulator during the tracking period ranged from zero to five centimeters. The method used in this study for catching was an observer and second-order filter along with an error estimator.

Studies by DLR involve a robotic arm which successfully catches flying balls [18] [19] [20]. These studies involve real systems which have simulated as well as physical test results. The method used for solving the inverse kinematics was nonlinear optimization. In one study, a seven degree-of-freedom robot is used to catch a ball with a success rate of more than 80%. The approach used for performing a catch while under joint, speed, and work cell limitations was a unified nonlinear optimization problem with nonlinear constraints [18]. The year after this study was published, another study was published by the same group describing how this functionality had been built onto a mobile robot [19]. Also, since the mobile robot has two of these arms, it can simultaneously catch two balls.

1.3 Objective

The main purpose of this thesis is to advance human-like capability in robotic arms. Specifically, the objectives are (1) design a robotic arm, (2) develop a technique to solve the inverse kinematic model, (3) build a rigid body dynamics model, (4) develop a control system for the robotic arm, and (5) apply the robotic arm to catching a ball. While the system can be easily used for a variety of other applications, this project applies the arm to one specific scenario which is usually only performed by humans, namely, catching a ball. The reason for this application is that most robotics lack human-like behavior, and this is one area is a step towards that goal. The main tasks of this project are:

- Design the geometry of the robotic arm.
- Develop an inverse kinematics solution method.
- Create a rigid body dynamics model.
- Design a control system and determine actuator control parameters.
- Set up the rigid body dynamics simulation and environment parameters.
- Perform catching tests and collect results.

This robotic arm is designed to be robust so that it can be used in other applications. A significant effort in this research is laying the groundwork of a simulation environment for the development of a robotic system. The technique developed for this system can be adapted to other applications or other robots with different geometries. So in addition to completing the aspects listed above, it is also the objective of this project to build a platform with which to advance humanoid robotics.

1.4 Overview of Approach

First of all, the boundaries of the study are defined:

- In order to define a scope which would be achievable in a project of this size, it is assumed that the initial position and velocity of the ball is known using a vision system. The vision system design is not included in this study.
- The rigid body dynamics model and control system script is developed within the multibody dynamics simulation software.

The first step is to design the geometric model and select motor placement. Basic research on human anthropometry defines ranges of sizes for different parts of the arm. The United States military has performed many anthropometric studies and these results are used to define the size of the arm and hand [21]. The arm has six degrees-of-freedom (not including the hand), three rotational and three positional degrees-of-freedom, thus requiring six actuators. The actuator placement is selected in order to give the arm human-like appearance and functionality.

Once the geometric design is complete, the model is imported into the multibody dynamics software. The joints, actuators, and robotic arm's rigid bodies are added to the system and a basic script is written to control the actuators. Denavit-Hartenberg parameters are developed and used to write the forward kinematics equations. The Denavit-Hartenberg approach is applied because it is easy to apply to a system when the actuators are initially at right angles with each other.

The next step is to write an algorithm to calculate the inverse kinematics. This is an essential component of this project. Since the forward kinematic equations are nonlinear, a numerical method (Newton-Raphson with a numerically calculated Jacobian) is implemented to solve for the joint angles, given the trajectory (position and orientation of the end effector). Since the arm has six degrees-of-freedom, only six equations can be used to solve for six joint angles. Since three of these equations pertain to the three-dimensional translational coordinates, only three equations are left for defining the orientation. Therefore, the three-dimensional rotation matrix which defines the end effector orientation is converted to Euler angles. This allows for the system to find a solution which satisfies both the position and complete orientation desired for the end effector.

While this study focuses on how to develop a robotic arm which can catch a ball, it is important to study how this process happens in the human body. Some important factors that were identified in one study [22] were wrist positioning and timing. These are important factors to keep in mind when developing an arm whose ability includes catching balls.

Finally, the catch zone is defined and the experiments are configured. A catch position and orientation are chosen to be dependent on the trajectory of ball. The experiments are run in the simulation, and the results can then be viewed to determine the performance of the arm. In each of these experiments, the robotic arm can be viewed from any angle and the catch rate can be determined.

The work presented here differs from previous work in the following ways:

- This study uses multibody dynamics software to create a high-fidelity virtual model of the robotic arm, perform ball-catching experiments, and collect robotic arm performance data. This allows for low-cost design iterations and performance feedback.

- The inverse kinematics are scripted using the Newton-Raphson method with a numerically calculated Jacobian to numerically solve for the desired joint angles. This approach can be readily applied to complex robotic systems, as opposed to analytic (closed-form) solutions. Also, the real inverse of the Jacobian, not the psuedo-inverse, is used in this technique, allowing for greater accuracy.

- The system closely resembles a human arm, which is based on anthropometric studies made by the United States military.

- The humanoid robotic arm is applied to catching a ball.

While each of these are not completely unique from other studies, the combination of these four elements in a single, comprehensive project make it novel.

2. MULTIBODY DYNAMICS FORMULATION

The multibody dynamics software used in this study is based on equations and theory presented in this chapter. The software is DIS, *Dynamic Interactions Simulator* [23]. This software was developed by Advanced Science and Automation Corporation by Tamer Wasfy and is used in this study to model the rigid body dynamics and the control system.

2.1 Governing Multibody Dynamics Equations

The Newton-Euler equations of motion (translational and rotational) are used as the governing equations of the rigid body dynamics analysis [24]. Assuming that the origin of the rigid body frame is at the body's center of gravity, the Newton-Euler equations of motion can be written as:

$$M\,\ddot{X}0_i = F_{res_i} \tag{2.1}$$

$$I_{ij}\ddot{\theta}_j + \left(\dot{\theta}_k \times \left(I_{jl}\dot{\theta}_l\right)\right)_i = T_{res_i} \tag{2.2}$$

Where:

M: Mass of the rigid body.

$\ddot{X}0_i$: Acceleration vector of center of gravity of body.

F_{res_i}: Force vector acting on rigid body.

I_{ij}: Body mass moment of inertia.

$\ddot{\theta}_j$: Angular acceleration vector of rigid body.

$\dot{\theta}_k$: Angular velocity vector of rigid body.

T_{res_i}: Torque vector acting on rigid body.

2.2 Joint Modeling

The purpose of this section is to give a brief overview of how the joints are modeled in the rigid body kinematics simulation environment.

Joint constraints are of the form:

$$f(\{X\}) = 0 \tag{2.3}$$

A joint is defined by the relation between connection points. This imposes motion constraints between points on rigid bodies.

2.2.1 Connection Points

Each rigid body can have a number of connection points. A connection point is a point on the body where joints can be located. The position of a connection point c on rigid body K with respect to the global inertial reference frame (X_{c_i}) is given by:

$$X_{c_i} = X_{Ki} + R_{Kij} x_{c_j} \tag{2.4}$$

Where:

X_{Ki}: position coordinates of center of gravity of rigid body K with respect to the global inertial reference frame.

R_{Kij}: rotation matrix of rigid body K.

x_{c_j}: position coordinates of point p with respect to body reference frame.

2.2.2 Spherical Joint

Spherical joints are important in this study because they are used in the construction of the end effector and they are also used to construct other types of joints, such as revolute joints. A spherical joint connects two points on two bodies. It constrains those two points such that they have the same translational coordinates relative to

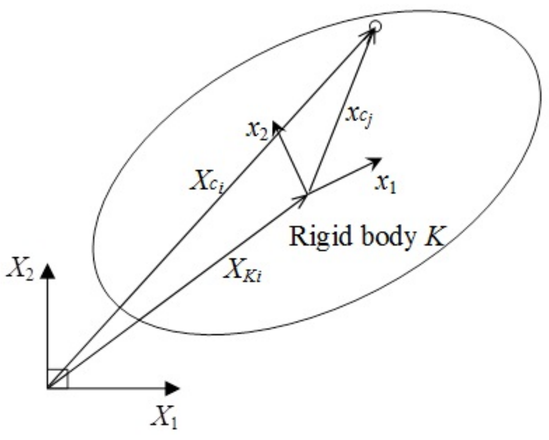


Figure 2.1. Defining a joint by relation to connection points

the global reference frame. Thus, a spherical joint between two connection points is defined as:

$$X^t_{c1_i} = X^t_{c2_i} \tag{2.5}$$

$$X^t_{c1_i} - X^t_{c2_i} = 0 \tag{2.6}$$

$X^t_{c1_i}$ is the global position vector of the first point *c1* and $X^t_{c2_i}$ is the global position vector of the second point *c2*. A spherical joint leaves three relative rotational degrees-of-freedom between the two rigid bodies free and constrains three relative translational degrees-of-freedom.

This constraint is imposed using the penalty technique:

$$F_p = k_p\, d + c_p\ d_i \dot{d}_i \tag{2.7}$$

$$d_i = X^t_{c1_i} - X^t_{c2_i} \tag{2.8}$$

$$\dot{d}_i = \dot{X}^t_{c1_i} - \dot{X}^t_{c2_i} \tag{2.9}$$

$$d = \sqrt{{d_1}^2 + {d_2}^2 + {d_3}^2} \tag{2.10}$$

$$F_{p_i} = F_p\, d_i/d \tag{2.11}$$

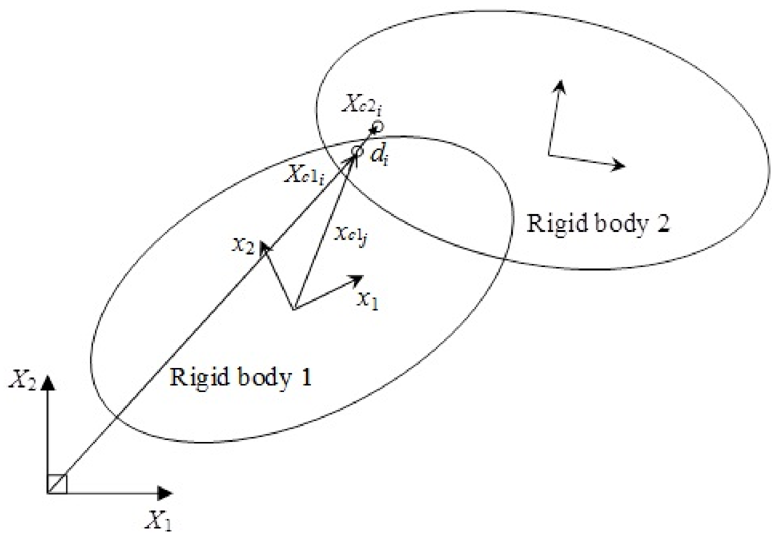


Figure 2.2. Spherical joint

Where:

F_p: penalty force magnitude.

k_p: penalty spring stiffness.

c_p: penalty damping.

d_i: relative displacement vector between points *c1* and *c2*.

$\dot{d}_i$: relative velocity vector between points *c1* and *c2*.

$\dot{X}^t_{c1_i}$: global velocity vector for point *c1*.

$\dot{X}^t_{c2_i}$: global velocity vector for point *c2*.

F_{p_i}: penalty reaction force on connection point *c1*.

The joint penalty force is applied on the two connection points in opposite directions. This force tries to make points $X^t_{c1_i}$ and $X^t_{c2_i}$ coincident, i.e. it tries to impose the joint constraint $X^t_{c1_i} = X^t_{c2_i}$ or $d_i = X^t_{c1_i} - X^t_{c2_i} = 0$.

Each joint force is transferred to the center of gravity of the corresponding rigid body (center of the body frame) using:

$$F_i = F_{p_i} \tag{2.12}$$

$$T_i = -(x_{c1_i} \times R_{ji} F_{p_i}) \tag{2.13}$$

Where:

F_i: the force at the c.g. of the rigid body.

T_i: moment on the rigid body.

x_{c1_i}: position of the point relative to the rigid body's frame.

R_{ji}: rigid body rotation matrix.

2.2.3 Revolute Joint

Most joints in the robotic arm are revolute joints. A revolute joints connects two rigid bodies. These are modeled by placing two spherical joints along a line. This constrains the three translational degrees-of-freedom between the two rigid bodies and two rotational degrees-of-freedom. This allows for a single free rotational degree-of-freedom (see Figure 2.3).

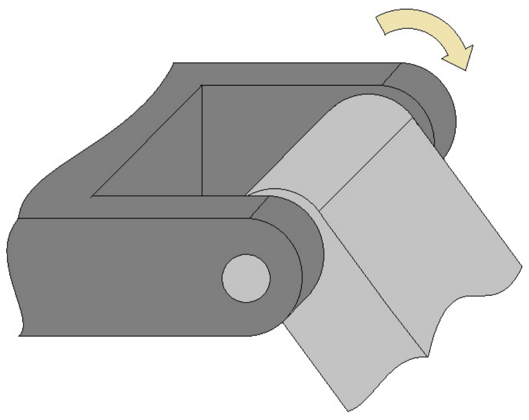


Figure 2.3. Revolute joint

2.2.4 Cylindrical Point Joint

The cylindrical point joints are important because they are used to model the cylindrical joint, which is used in the robotic arm. A cylindrical point joint connects a linear curve on one rigid body to a point on another rigid body. It constrains the point to move on the linear curve. Typically this curve is a line but it can be any three-dimensional curve.

The cylindrical point joint constrains two translational degrees-of-freedom between the two bodies and leaves one translational and three rotational degrees-of-freedom free.

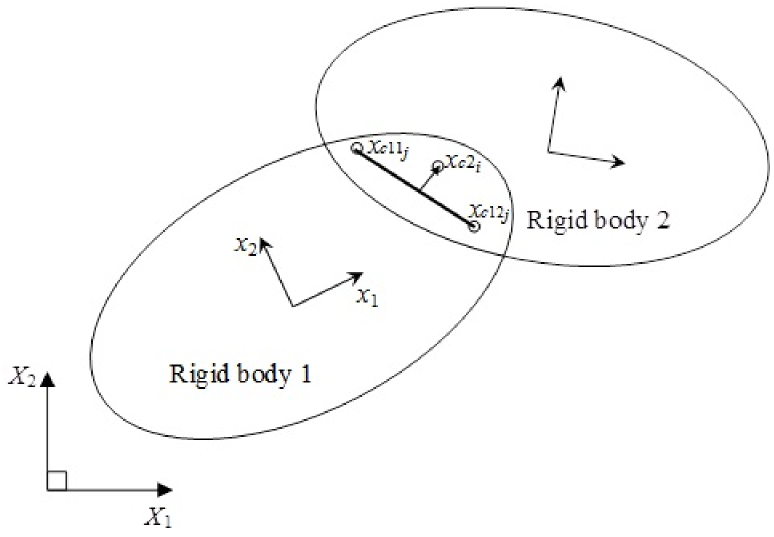


Figure 2.4. Cylindrical point joint definition

The equation of a line on the first rigid body is given by:

$$X^t_{c1_i} = (1 - s)X^t_{c11_i} + sX^t_{c12_i} \tag{2.14}$$

The parameter s indicates a position along that line with a value from 0 to 1. A cylindrical point joint connects a point on that line to a fixed point on the second rigid body:

$$X^t_{c1_i} = X^t_{c2_i} \tag{2.15}$$

$$\Rightarrow (1-s)X^t_{c11_i} + sX^t_{c12_i} = X^t_{c2_i} \tag{2.16}$$

$$\Rightarrow \big((1-s)X^t_{c11_i} + sX^t_{c12_i}\big) - X^t_{c2_i} = 0 \tag{2.17}$$

This constraint is imposed using the penalty technique:

$$F_p = k_p\, d + c_p\, d_i\, \dot{d}_i \tag{2.18}$$

$$d_i = \big((1-s)X^t_{c11_i} + sX^t_{c12_i}\big) - Xc2^t_i \tag{2.19}$$

$$\dot{d}_i = \Big((1-s)\dot{X}c11^t_i + s\dot{X}c12^t_i\Big) - \dot{X}c2^t_i \tag{2.20}$$

$$d = \sqrt{{d_1}^2 + {d_2}^2 + {d_3}^2} \tag{2.21}$$

$$F_{p_i} = F_p\, d_i/d \tag{2.22}$$

$$d_i\dot{d}_i = d_1\dot{d}_1 + d_2\dot{d}_2 + d_3\dot{d}_3 \tag{2.23}$$

Where:

F_p: penalty force magnitude.

k_p: penalty spring stiffness.

c_p: penalty damping.

d_i: relative displacement vector between points *c1* and *c2*.

$\dot{d}_i$: relative velocity vector between points *c1* and *c2*.

$\dot{X}^t_{c1_i}$: global velocity vector for point *c1*.

$\dot{X}^t_{c2_i}$: global velocity vector for point *c2*.

F_{p_i}: penalty reaction force on connection point *c1*.

The parameter s is chosen such that:

$$d_i v_i = 0 \text{ (vector } \vec{d} \text{ is normal to vector } \vec{v}) \tag{2.24}$$

$$d_i v_i = d_1 v_1 + d_2 v_2 + d_3 v_3 \tag{2.25}$$

Where $v_i = X^t_{c12_i} - X^t_{c11_i}$ is the vector along the cylindrical joint line.

$$\left(\left((1-s)X^t_{c11_i} + sX^t_{c12_i}\right) - X^t_{c2_i}\right) v_i = 0 \tag{2.26}$$

$$\begin{split}\left((1-s)X^t_{c11_1}v_1 + sX^t_{c12_1}v_1\right) - X^t_{c2_1}v_1 + \left((1-s)X^t_{c11_2}v_2 + sX^t_{c12_2}v_2\right) \\ - X^t_{c2_2}v_2 + \left((1-s)X^t_{c11_3}v_3 + sX^t_{c12_3}v_3\right) - X^t_{c2_3}v_3 = 0\end{split} \tag{2.27}$$

$$\begin{split}s\left(X^t_{c12_1}v_1 + X^t_{c12_2}v_2 + X^t_{c12_3}v_3 - X^t_{c11_1}v_1 - X^t_{c11_2}v_2 - X^t_{c11_3}v_3\right) \\ + X^t_{c11_1}v_1 + X^t_{c11_2}v_2 + X^t_{c11_3}v_3 - X^t_{c2_1}v_1 - X^t_{c2_2}v_2 - X^t_{c2_3}v_3 = 0\end{split} \tag{2.28}$$

$$s = -\frac{X^t_{c11_1}v_1 + X^t_{c11_2}v_2 + X^t_{c11_3}v_3 - X^t_{c2_1}v_1 - X^t_{c2_2}v_2 - X^t_{c2_3}v_3}{X^t_{c12_1}v_1 + X^t_{c12_2}v_2 + X^t_{c12_3}v_3 - X^t_{c11_1}v_1 - X^t_{c11_2}v_2 - X^t_{c11_3}v_3} \tag{2.29}$$

If $s < 0$ then $s = 0$ and if $s > 1$ then $s = 1$. Each joint force is transferred to the center of gravity of the corresponding rigid body (center of the body frame) using equations (2.12) and (2.13).

2.2.5 Cylindrical Joint

Cylindrical joints are used to model the linear actuator components found in the end effector. A cylindrical joint between two rigid bodies can be modeled using a cylindrical point joint by placing two points on the second body rather than one point. Thus, the cylindrical joints connect a line on the first rigid body to two points on the second rigid body. The two points are constrained to move on the line. This constrains two translational degrees-of-freedom and two rotational degrees-of-freedom between the two bodies and leaves one translational and one rotational degree-of-freedom free (see Figure 2.5).

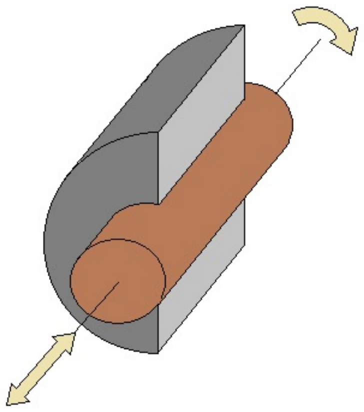


Figure 2.5. Cylindrical joint

2.3 Contact Modeling

Contact modeling is used in this study to calculate the interactions between the robot's end effector (hand) and the ball. This section outlines the method for modeling contact between rigid bodies, and this is based on previous work by Wasfy [25] [26].

Contact constraints are of the form:

$$f(\{X\}) \geq 0 \tag{2.30}$$

The penalty technique is used to impose the normal contact constraints between points on one rigid body and a surface on a second rigid body. The first step is to find the position and velocity of each contact point.

$$X^t_{c_i} = X^t_{Ki} + R^t_{Kij} x_{cj} \tag{2.31}$$

$$\dot{X}^t_{c_i} = \dot{X}^t_{Ki} + R^t_{Kij} \left(\{\dot{\theta}^t_K\} \times \{x_c\} \right)_j \tag{2.32}$$

Where:

x_{cj}: coordinates of contact point c with respect to local frame of first body.

$X_{c_i}^{t}$: coordinates of contact point c with respect to global frame at time t.

X_{Ki}^{t}: coordinates first body frame center with respect to global frame at time t.

R_{Kij}^{t}: Rotation matrix of the first rigid body at time t.

$\dot{X}_{c_i}^{t}$: velocity components of contact point c with respect to global reference frame at time t.

$\dot{X}_{Ki}^{t}$: velocity components of center of first body frame with respect to global frame at time t.

$\dot{\theta}_{Kk}^{t}$: angular velocity components of the first body at time t.

The total contact force (*normal contact force + tangential friction force*), F_c, at each contact point is transferred as a force and a moment to the center of the rigid body using equations (2.12) and (2.13). The negative of this force is transferred to the second body using the same equations.

2.3.1 Penalty Contact Model

The penalty technique is used for imposing the normal contact constraint. Using this technique, a normal reaction force (F_{normal}) is generated when a point penetrates a contact surface. The magnitude of the normal reaction force is proportional to the penetration distance and to the velocity of penetration:

$$F_{normal} = A\, k_p\, d + A \begin{cases} c_p\, \dot{d} & \dot{d} \geq 0 \\ s_p\, c_p\, \dot{d} & \dot{d} < 0 \end{cases} \tag{2.33}$$

$$\dot{d} = v_i\, n_i \tag{2.34}$$

$$vn_i = \dot{d}\, n_i \tag{2.35}$$

Where:

A: area of the rectangle associated with the contact point.

k_p and c_p: penalty stiffness and damping coefficient per unit area.

d: closest distance between the contact point and the contact surface.

$\dot{d}$: signed time rate of change of d.

s_p: separation damping factor between 0 and 1 which determines the amount of sticking between the contact point and the contact surface at the contact point (leaving the body).

$\vec{n}$: unit vector normal to the surface.

$\vec{v}_i$: the relative velocity vector between the contact point and the contact surface.

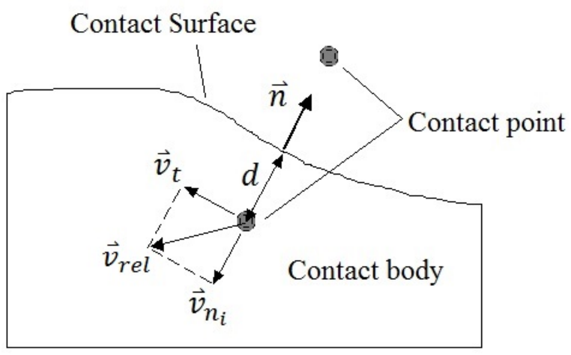


Figure 2.6. Contact surface and contact node [25] [26]

The normal contact force vector is given by:

$$F_{n_i} = n_i \, F_{normal} \tag{2.36}$$

The total force on the node generated due to the frictional contact between the point and surface is given by:

$$F_{point_i} = F_{t_i} + F_{n_i} \tag{2.37}$$

2.3.2 Friction

An asperity-spring friction model is used to model joint and contact friction. Friction is modeled using a piece-wise linear velocity-dependent approximate Coulomb friction element in parallel with a variable anchor point spring.

Asperity-Friction Theory

If the contact surfaces between objects is viewed at a microscopic scale, a surface that may appear smooth actually has asperities that can range from one molecule high to several thousand molecules high. When the two surfaces are in contact, the asperities interlock between the two surfaces.

When a tangential force is applied to the block, the normal contact forces between the asperities on both surfaces prevent motion of the block. Since the block is not moving, this means that the friction force is equal to the tangential force.

However, if the magnitude of the tangential force is more than the product of the friction coefficient (μ) and the normal reaction force N, the asperities will either break or deform to allow motion of the block.

If the normal force, N, is increased, then the surface asperities interlock more tightly. In this case, a larger tangential force is required to break or deform the asperities to allow for the block to move.

This explanation assumes that other contributing forces such as chemical bonding or electrical interactions, are negligible.

Asperity-Friction Model

This model approximates asperity friction where friction forces between two rough surfaces in contact arise due to the interaction of the surface asperities [25] [26]. F_{t_i} is the tangential friction contact force vector transmitted to the contact body at the contact point. It is given by:

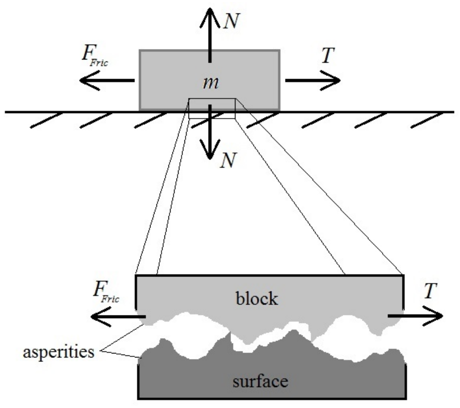


Figure 2.7. Physical interpretation of friction [25] [26]

$$F_{t_i} = F_{tangent} t_i \tag{2.38}$$

The asperity friction model uses the normal force to calculate the tangential friction force ($F_{tangent}$). When two surfaces are in static contact, the surface asperities act like tangential springs. When a tangential force is applied, the springs elastically deform and pull the surfaces to their original position. If the tangential force is large enough, the surface asperities yield (the springs break) allowing sliding to occur between the two surfaces. The breakaway force is proportional to the normal contact pressure.

When the two surfaces are sliding past each other, the asperities provide resistance to the motion that is a function of the sliding velocity, acceleration, and normal contact pressure.

The figure below shows a schematic diagram of the asperity friction model. It is composed of a simple piece-wise linear velocity-dependent approximate Coulomb friction element (that only includes two linear segments) in parallel with a variable anchor point spring.

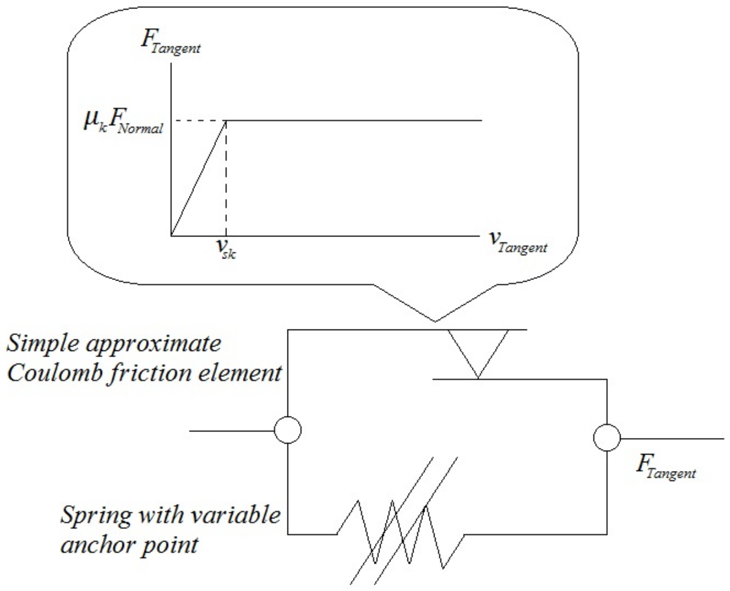

$F_{Tangent}$
$\mu_k F_{Normal}$
v_{sk}
$v_{Tangent}$
Simple approximate Coulomb friction element
$F_{Tangent}$
Spring with variable anchor point

Figure 2.8. Asperity spring friction model [25] [26]

2.3.3 Contact Search

Contact detection is important because it determines when to apply surface interaction forces. Contact is detected between points on a master contact surface and a polygonal surface called the slave contact surface. The contact points of the master contact surface are points on a rigid body. The slave contact surface is a polygonal surface on another rigid body.

Contact between the contact points of the master surface and the polygons of the slave surface is detected using a binary tree contact search algorithm which allows fast contact search [27].

2.4 Rotational Springs

Rotational springs were included in the kinematic model. A rotational spring element connects three points on two rigid bodies. Two of the points are on one rigid body and the remaining point is on the second rigid body.

The rotary spring torque (T) is given by:

$$T = k_t\,(\theta - \theta_o) + c_t\,\dot{\theta} \tag{2.39}$$

Where:

k_t: torsional stiffness of the rotational spring.

c_t: torsional damping of the rotational spring.

θ: angle between the three points.

θ_o: un-stretched angle between the three points.

$\dot{\theta}$: Angular velocity.

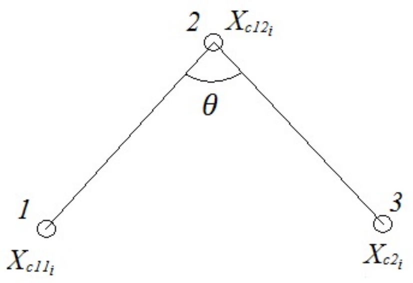


Figure 2.9. Rotational springs

The rotary spring forces are transferred to the rigid body's center of gravity as forces and moments using equations (2.12) and (2.13).

2.5 Actuators

Actuators are a critical component in this study. These components in the model simply generate forces between points on rigid bodies. The two types of actuators used in this study are linear and rotational actuators.

2.5.1 Linear Actuators

A linear actuator connects two points on two rigid bodies. A force, F, is generated by the actuator using a PD controller. This controller uses the following formulas to calculate the force.

$$F = k(l - l_{des}) + c(\dot{l}_i l_i / l - \dot{l}_{des}) \tag{2.40}$$

$$l_i = X^t_{c1_i} - X^t_{c2_i} \tag{2.41}$$

$$\dot{l}_i = \dot{X}^t_{c1_i} - \dot{X}^t_{c2_i} \tag{2.42}$$

$$l = \sqrt{{l_1}^2 + {l_2}^2 + {l_3}^2} \tag{2.43}$$

$$F_i = F l_i / l \tag{2.44}$$

$X^t_{c1_i}$ is the position of the first point on the first body and $X^t_{c2_i}$ is the position of the second point on the second body, both with respect to the global reference frame (Figure 2.10). The controller parameters in Equation 2.40 are k, the proportional gain, and c, the derivative gain. The desired length is l_{des}, and the current length is l. F_i is the actuator force vector.

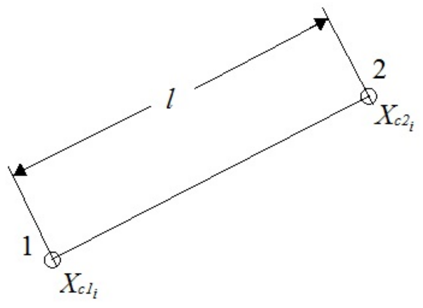


Figure 2.10. Linear Actuator

2.5.2 Rotational Actuators

Similar to a rotational spring, a rotational actuator connects three points on two rigid bodies (Figure 2.9). Two of the points are on one rigid body and the remaining point is on the second rigid body. A torque, T, is generated by the actuator using a PD controller:

$$T = k(\theta - \theta_{des}) + c\,(\dot{\theta} - \dot{\theta}_{des}) \tag{2.45}$$

The current angle of the actuator is θ, and the desired angle is θ_{des}. As in the linear actuator, k is the proportional gain and c is the derivative gain.

2.6 Numerical Integration of the Equations of Motion

If a multibody system consisting of N rigid bodies, then the Newton-Euler equations can be written for each rigid body as:

$$M_K \ddot{X}^t_{Ki} = F^t_{Ki} \tag{2.46}$$

$$I_{Kij}\ddot{\theta}^t_{Kj} = T^t_{Ki} - \left(\dot{\theta}^t_{Kk} \times (I_{Kjl}\dot{\theta}^t_{Kl})\right)_{Ki} \tag{2.47}$$

Where:

t: running time.

K: rigid body number $K = 1$ to N

N: total number of rigid bodies.

M_K: mass of body K.

I_{Kij}: mass moment of inertia of body K.

X^t_{Ki}: position vector of the c.g. of body K.

$\dot{X}^t_{Ki}$: velocity vector of the c.g. of body K.

$\ddot{X}^t_{Ki}$: acceleration vector of the c.g. of body K.

$\dot{\theta}^t_{Kl}$: angular velocity vector of the rigid body K.

$\ddot{\theta}^t_{Kl}$: angular acceleration vector of the rigid body K.

F^t_{Ki}: force vector acting on the rigid body K.

T^t_{Ki}: applied/external moment vector acting on the rigid body K.

The trapezoidal integration rule can be used to integrate each body's translational equations of motion (2.46) in time (Δt is the time step):

$$\dot{X}^t_{Kj} = \dot{X}^{t-\Delta t}_{Kj} + 0.5\,\Delta t\,(\ddot{X}^t_{Kj} + \ddot{X}^{t-\Delta t}_{Kj}) \tag{2.48}$$

$$X^t_{Kj} = X^{t-\Delta t}_{Kj} + 0.5\,\Delta t\,(\dot{X}^t_{Kj} + \dot{X}^{t-\Delta t}_{Kj}) \tag{2.49}$$

Similarly, the trapezoidal rule can be used to integrate each body's rotational equations of motion (2.47) in time:

$$\dot{\theta}^t_{Kj} = \dot{\theta}^{t-\Delta t}_{Kj} + 0.5\,\Delta t\,(\ddot{\theta}^t_{Kj} + \ddot{\theta}^{t-\Delta t}_{Kj}) \tag{2.50}$$

$$\Delta\theta^t_{Kj} = 0.5\,\Delta t\,(\dot{\theta}^t_{Kj} + \dot{\theta}^{t-\Delta t}_{Kj}) \tag{2.51}$$

Where:

$\Delta\theta^t_{Kj}$: incremental rotation vector of body K.

Then, the rigid body rotation matrix can be calculated at time t as:

$$R^t_{Kij} = R^{t-\Delta t}_{Kik}\,R_{Kkj}(\Delta\theta^t_{Kl}) \tag{2.52}$$

2.6.1 Solution Procedure

The solution fields for modeling multibody systems are defined for the rigid bodies. Note that a rigid body is modeled as one node. These solutions fields are:

- Global translational position vectors (X^t_{Ki})
- Global translational velocity vectors ($\dot{X}^t_{Ki}$)

- Global translational acceleration vectors ($\ddot{X}^t_{Ki}$)
- Global to local rotation matrices (R^t_{Kij})
- Local body rotational velocity vectors ($\dot{\theta}^t_{Kj}$)
- Local body rotational acceleration vectors ($\ddot{\theta}^t_{Kj}$)

Where:

$K = 1$ to N

N: total number of rigid bodies.

An explicit predictor-corrector solution procedure will be used based on the trapezoidal integration rule for time. The explicit time integration solution procedure predicts the time evolution of the above response quantities.

1. Set $t = 0$ and set the initial conditions for all the rigid bodies (nodes). Those are:

$$X^0_{Ki}, \quad \dot{X}^0_{Ki}, \quad \ddot{X}^0_{Ki}, \quad R^0_{Kij}, \quad \dot{\theta}^0_{Kj}, \quad \ddot{\theta}^0_{Kj}$$

2. Increment time: $t = t + \Delta t$

3. Set the nodal values at the last time step to be equal to the current nodal values for all solution fields.

$$X^t_{Ki} = X^{t-\Delta t}_{Ki} \qquad \dot{X}^t_{Ki} = \dot{X}^{t-\Delta t}_{Ki} \qquad \ddot{X}^t_{Ki} = \ddot{X}^{t-\Delta t}_{Ki}$$
$$R^t_{Kij} = R^{t-\Delta t}_{Kij} \qquad \dot{\theta}^t_{Kj} = \dot{\theta}^{t-\Delta t}_{Kj} \qquad \ddot{\theta}^t_{Kj} = \ddot{\theta}^{t-\Delta t}_{Kj}$$

4. Perform 2 iterations (a predictor iteration and a corrector iteration) of the following steps.

   (a) Initialize the nodal forces and moments to zero.

$$F^t_{Ki} = 0 \qquad T^t_{Ki} = 0$$

(b) Calculate the nodal forces and moments produced by all elements, joints and contact forces and assemble those forces and moments to F^t_{Ki} and T^t_{Ki}.

(c) Calculate $\ddot{X}^t_{Ki}$ and $\ddot{\theta}^t_{Kj}$ using Equations (2.46) and (2.47) respectively:

$$\ddot{X}^t_{Ki} = F^t_{Ki}/M_K$$
$$\ddot{\theta}^t_{Kj} = I^{-1}_{Kij}\left(T^t_{Ki} - \left(\dot{\theta}^t_{Kk} \times (I_{Kjl}\dot{\theta}^t_{Kl})\right)_{Ki}\right)$$

(d) Use the trapezoidal time integration formulas (equations (2.48), (2.49), (2.50), and (2.51)) to calculate $\dot{X}^t_{Ki}$, X^t_{Ki}, $\dot{\theta}^t_{Kj}$ and R^t_{Kij}:

$$\dot{X}^t_{Kj} = \dot{X}^{t-\Delta t}_{Kj} + 0.5\,\Delta t\,(\ddot{X}^t_{Kj} + \ddot{X}^{t-\Delta t}_{Kj})$$

$$X^t_{Kj} = X^{t-\Delta t}_{Kj} + 0.5\,\Delta t\,(\dot{X}^t_{Kj} + \dot{X}^{t-\Delta t}_{Kj})$$

$$\dot{\theta}^t_{Kj} = \dot{\theta}^{t-\Delta t}_{Kj} + 0.5\,\Delta t\,(\ddot{\theta}^t_{Kj} + \ddot{\theta}^{t-\Delta t}_{Kj})$$

$$\Delta\theta^t_{Kj} = 0.5\,\Delta t\,(\dot{\theta}^t_{Kj} + \dot{\theta}^{t-\Delta t}_{Kj})$$

$$R^t_{Kij} = R^{t-\Delta t}_{Kik}\,R_{Kkj}(\Delta\theta^t_{Kl})$$

(e) Execute the prescribed motion constraints which set the nodal value(s) to prescribed values.

5. Go to step 2.

3. ROBOT DESIGN

The initial step in this project was to develop the design of the robotic arm. This exercise in design included an iterative process of developing a concept, refining the details, and deciding on a finalized design. The reason for developing a new geometric design was to demonstrate the entire development process of the humanoid robotic system and its control system. However, it should be noted that the control system is not specific to this design. A different design strategy could be chosen and the control system algorithm would be the same.

3.1 Design Goal

While this study focuses on the control system for a humanoid robotic arm, the general design of the arm geometry was also included. The primary goal with this robotic manipulator was to design it to be as similar as possible to the human arm. The arm was designed to be close to the same size and functionality of a human arm, and weight was also a consideration. Sample human arm measurements were taken from the military handbook *Anthropometry of U.S. Military Personnel*, released in 1991 [21].

Detailed actuator selection was not a focus of this thesis. Instead, the actuators were roughly estimated from initial research of motors.

3.2 Design Requirements

Requirements of the design center around two aspects of the system. The first is that this design is distinctly humanoid. The second is that this arm is used to catch balls. These aspects drive the requirements of the robotic arm. The requirements are as follows:

- The arm (not including the thumb and fingers) must have six degrees-of-freedom to allow for independent control of position and orientation of the end effector.
- The system must be similar in size to a human arm.
- The hand must be capable of grasping (grasp a ball).

These main requirements then drive the decisions in the design process. Major design decisions are outlined in the final design section below.

3.3 Design Overview

The scope of the robotic arm design was narrowed to only include a general solution. The point of this solution was to develop a model which roughly included the main ideas behind the kinematics of the arm in order to demonstrate the control system. A rendering of the final design is shown in Figure 3.1.

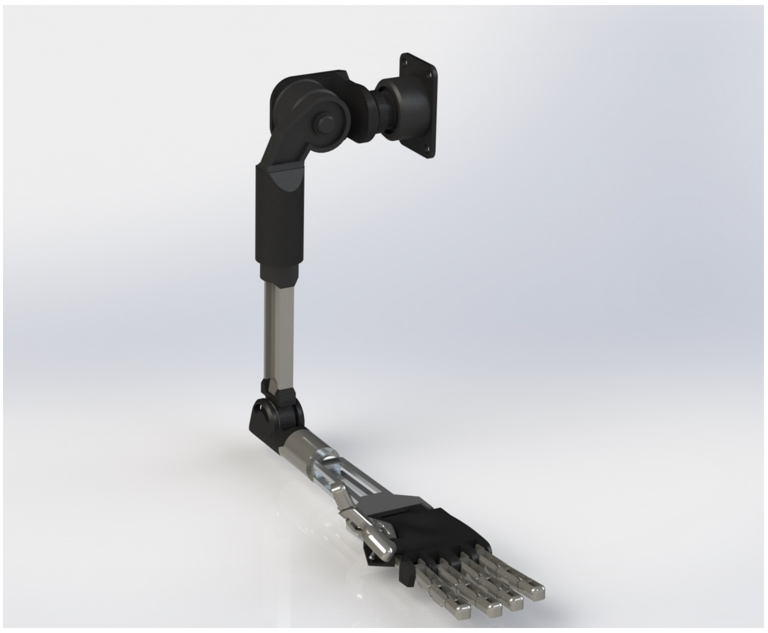


Figure 3.1. Final design

Overall, there are six main actuators which control the robotic arm, not including the end effector actuators. With each part of the design of this arm, the form and function of the human arm was considered. While compromises have to be made to accommodate today's technology, significant effort was made to obtain correct proportions and size for this design.

A critical aspect of this design is the hand subassembly. This specific subassembly directly affects the output of the study. This is because the contact simulation uses the surface of the hand and fingers to calculate the interaction with the ball. A close-up view of the hand is shown in Figure 3.2 and a grasp position is shown in Figure 3.3.

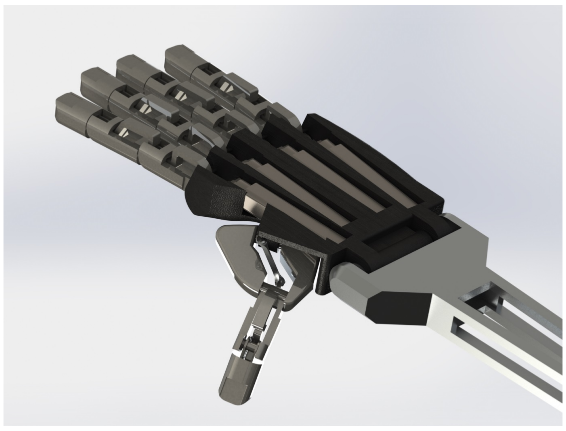


Figure 3.2. Detailed view of hand design

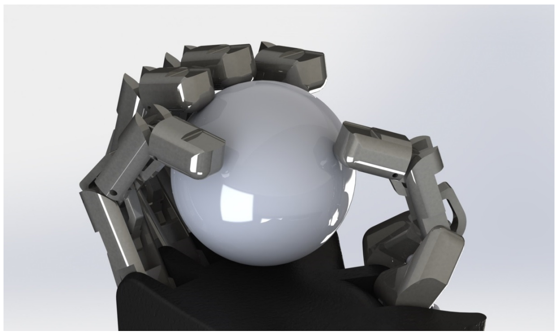


Figure 3.3. Grasp position

The design process is explained in the next section.

3.4 Final Design

The first task in the design process was to gather measurements of a human arm. The primary resource was a study completed by the United States military [21]. This resource contains averages of measurements of different parts of the human body, including the arm and hand.

The following figure identifies measurements comparing anthropometric data with the robot arm. Table 3.1 outlines the approximate dimensions that were chosen for the robot arm based on sample averages.

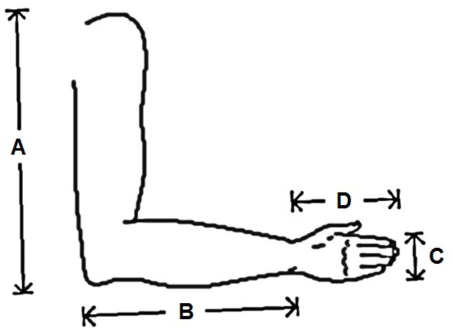


Figure 3.4. Comparison Measurements

Table 3.1. Comparing Arm Measurements

Measurement	U.S. Army Average (mm)	Robotic Arm (mm)
A	340.8	333
B	290.3	251
C	90.4	93
D	193.8	210

Once the basic dimensions were determined, the placement of the actuators had to be decided. In order to reduce the complexity of the system, only inline motors were used for the six major joints. This also allows for the joint angles to be acquired directly through the motor's encoders.

It is apparent from the observation of a human arm that three degrees-of-freedom are required at the shoulder. These three degrees-of-freedom were mimicked in the model. While the first two actuators have intersecting axes, the third actuator's axis does not intersect the previous actuator's axis. This allows the arm to reach across the torso of a full body robot, and it also produces a form similar to a human shoulder.

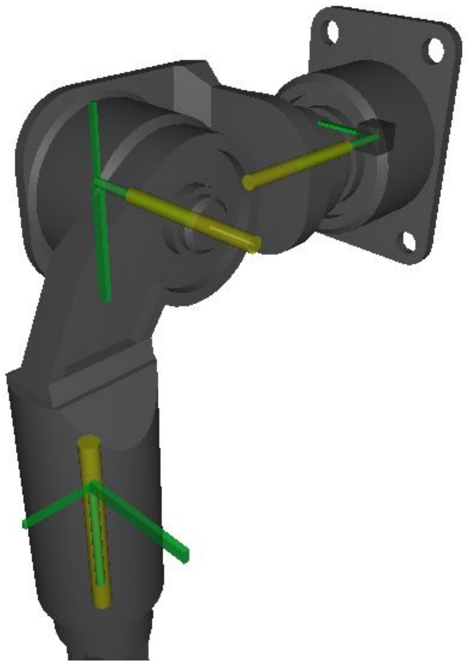


Figure 3.5. Shoulder Joints

The placement of the elbow joint and forearm can also be determined by observation. There are two degrees-of-freedom near the elbow location. Figure 3.6 shows where the elbow and upper forearm joints were placed.

A single wrist joint is the last joint in the arm's kinematic chain. This is powered by a rotational actuator. The end effector in this case also has actuated parts which are not part of the control system's inverse kinematic calculation. These actuated parts are the fingers and thumb. Figure 3.7 shows the wrist joint along with the joints and actuators for the fingers and thumb.

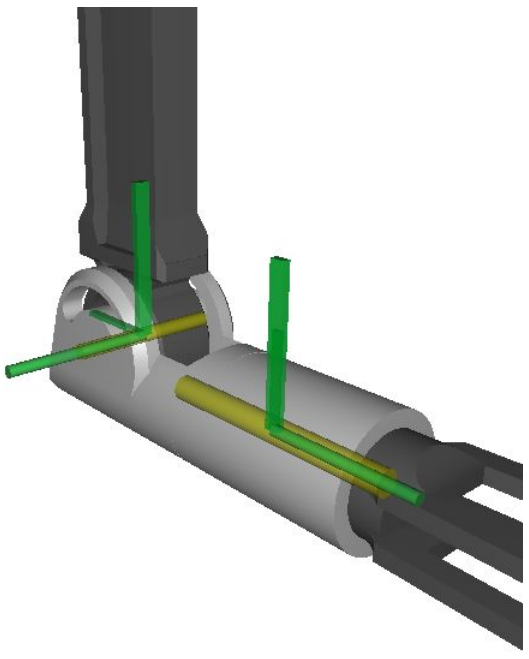
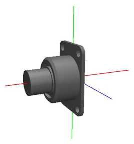
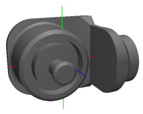
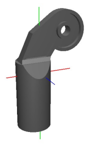
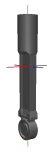
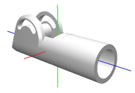
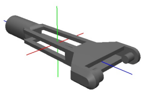
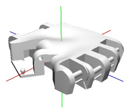
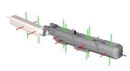
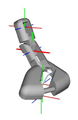


Figure 3.6. Elbow and Forearm Joints

Since the only action required by the hand in this study is grasping, a dependent linkage system and a single actuator for each digit are used (Figure 3.8). This technique has been used on multiple systems before this study, with the most prominent use being on the NASA Robonaut [28]. This allows for a more direct method for controlling grasp when independent control of each joint in the finger is not needed. In this case, linear actuators are used to manipulate the fingers, and a rotational actuator is used for the thumb (Figure 3.9).

All structural components were designed to be made from an aluminum alloy, because weight is a major consideration. The motor masses were estimated based on size and initial research. A CAD package was used to create geometric models of all parts and also to calculate the mass and moments of inertia. These properties were then imported into the multibody dynamics simulation software for each component. In Table 3.2, motor masses were combined with their respective links, so the total mass is the mass of the complete system.

Table 3.2. Masses of Robotic Arm Components

Component	Mass (kg)	I_{xx} $(kg\,m^2)$	I_{yy}	I_{zz}	Figure
Shoulder Link 1	1.4	0.1	0.1	0.1	
Shoulder Link 2	1.4	0.000703	0.00124	0.00132	
Shoulder Link 3	1.3	0.00235	0.000893	0.0029	
Upper Arm	1.2	0.000219	0.00783	0.0079	
Elbow Link	0.3	8.86e-005	0.000382	0.000384	
Forearm	0.3	0.000105	0.00116	0.00124	
Palm	0.4	0.00026	0.00045	0.00068	
Finger (4)	0.075	-	-	-	
Thumb	0.068	-	-	-	
Total	6.667				

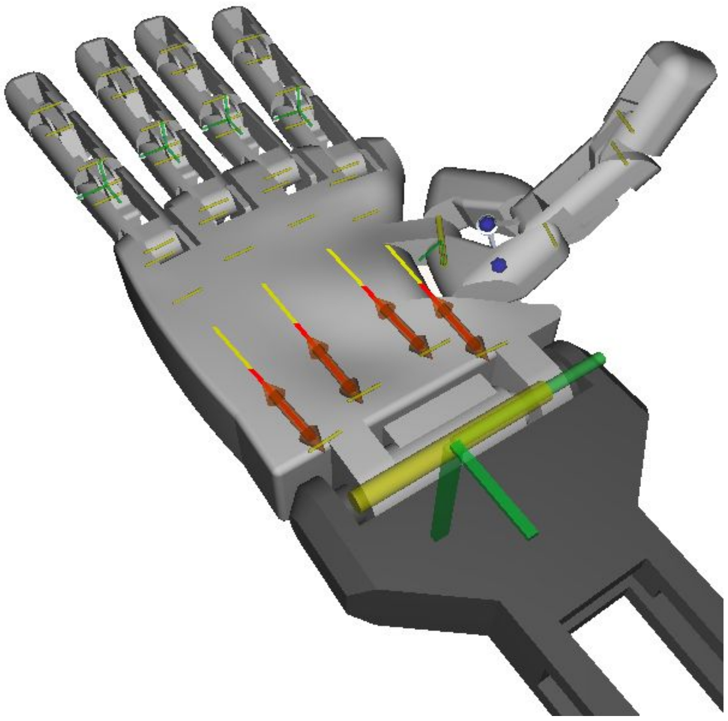


Figure 3.7. Wrist and Hand Joints

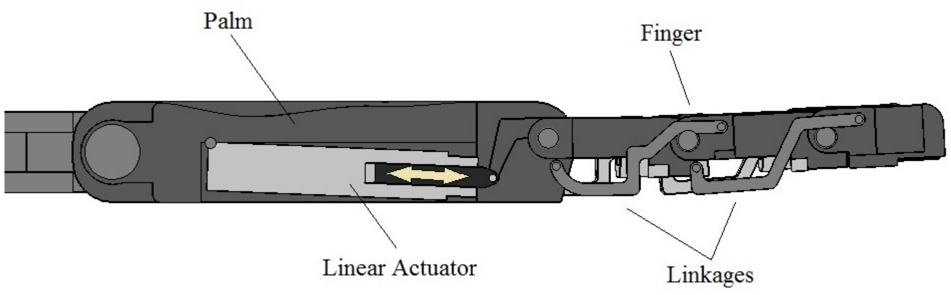


Figure 3.8. Actuated Finger Linkage System

At the end of this task, the critical geometry of the robotic arm had been designed. This data was then available to be imported into the multibody dynamics software.

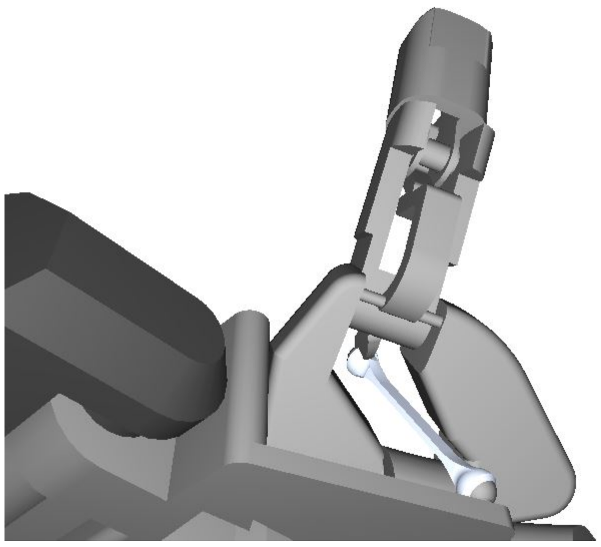


Figure 3.9. Actuated Thumb Linkage System

4. ROBOT MODEL

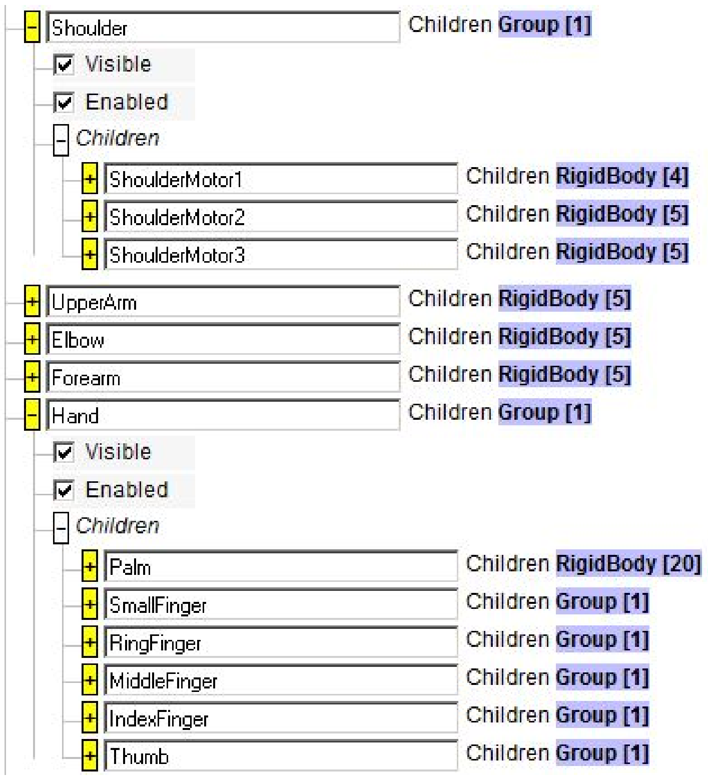
This section discusses the setup of the robot model within the multibody dynamics software as well as the simulation parameters. This involves placing joints and actuators in the correct locations and defining rigid bodies. The output of this step is a fully defined kinematic chain of rigid bodies.

4.1 Robot Arm Setup

In order to build a kinematic model of the robot arm, each component was imported into the simulation package as 3D objects. These objects were then placed in space as an assembly and joints were added which connect each of these components together. The hierarchy tree of the model is shown in Figure 4.1. The assembly was tested by running a simulation with the model under gravity. With no actuators, the links would simply fall and it could be observed whether or not the joints were oriented properly. The joints of the entire arm are shown in Figure 4.2.

Once the joints were placed, the actuators were added. All actuators are rotational except for the finger actuators. These actuators are controlled by PD controllers in a script within the simulation package. Initial gains were added and the system was simulated under an earth-gravity acceleration. The proportional gains were then fine-tuned by either increasing or reducing them to the point where the arm had just enough torque to oppose an earth-gravity acceleration environment with minimal movement. Then damping gains were added so that when the actuators moved to a desired angle, the actuator was minimally overdamped. This gave desirable movement without overshoot or oscillation during initial experiments.

The hand was a complicated model for a few reasons. The first reason is that there are several revolute joints within each finger. Also, the linear actuators have a



```
- Shoulder                Children Group [1]
    ☑ Visible
    ☑ Enabled
    - Children
        + ShoulderMotor1      Children RigidBody [4]
        + ShoulderMotor2      Children RigidBody [5]
        + ShoulderMotor3      Children RigidBody [5]
+ UpperArm                Children RigidBody [5]
+ Elbow                   Children RigidBody [5]
+ Forearm                 Children RigidBody [5]
- Hand                    Children Group [1]
    ☑ Visible
    ☑ Enabled
    - Children
        + Palm                Children RigidBody [20]
        + SmallFinger         Children Group [1]
        + RingFinger          Children Group [1]
        + MiddleFinger        Children Group [1]
        + IndexFinger         Children Group [1]
        + Thumb               Children Group [1]
```

Figure 4.1. Robotic Arm Model Hierarchy

large amount of leverage to control the arm, so this required fine-tuning in order to get desirable movement. Finally, since the hand would experience high acceleration as the arm moved to a position, the fingers could easily be affected. This acceleration would produce undesirable movement in the fingers, and this would sometimes cause the finger to bend into a locked position. In order to avoid this, a rotational spring was added in the middle joint of each finger to force them to move in the correct range. Also, the gains of the linear actuator which drive each finger were set such that floppy movement during high acceleration was avoided. The model hierarchy for one of the fingers is shown in Figure 4.3. The other fingers were similar.

Another important aspect of setting up the model was specifying contact surfaces. The entire hand was designated to be a contact surface. The mesh came directly from the geometric model, shown in Figure 4.4.

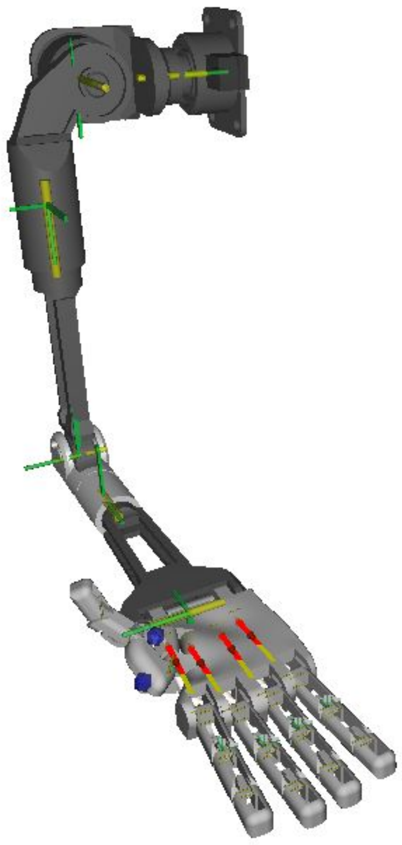
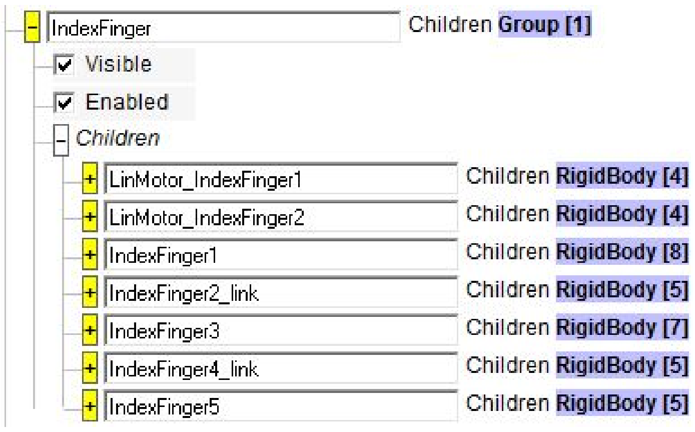


Figure 4.2. Entire Arm with Joints



IndexFinger Children Group [1]
Visible
Enabled
Children
LinMotor_IndexFinger1 Children RigidBody [4]
LinMotor_IndexFinger2 Children RigidBody [4]
IndexFinger1 Children RigidBody [8]
IndexFinger2_link Children RigidBody [5]
IndexFinger3 Children RigidBody [7]
IndexFinger4_link Children RigidBody [5]
IndexFinger5 Children RigidBody [5]

Figure 4.3. Index Finger Model Hierarchy

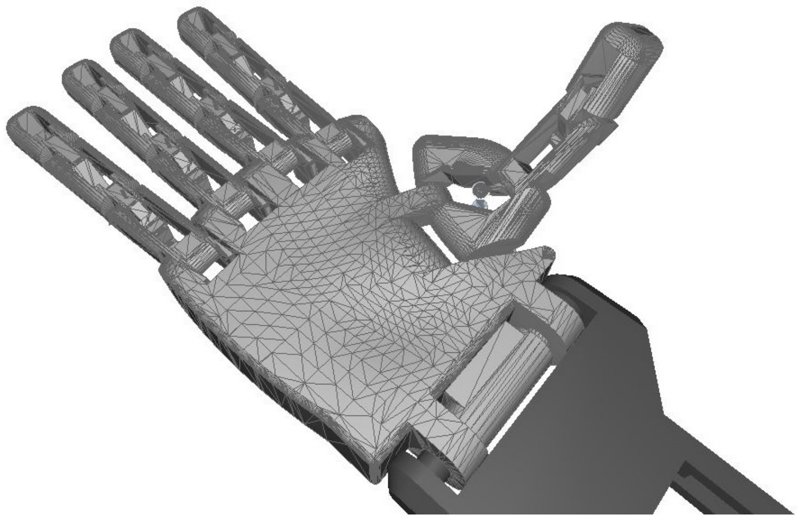


Figure 4.4. Contact Mesh

4.2 Environment Parameters

The rigid body simulation was set up to calculate the interaction of the rigid bodies. A gravity field was added to the simulation in the y-direction. The time step for the simulation was selected based on the desired accuracy of the joints and to accomodate the controller gains. Once the model was set up, experiments were run to determine which time steps were necessary for this model. First, a very small time step was selected, and the simulation was run. The time step was then incremented and tested until the solver produced a different solution or failed. The time step finally decided upon was 0.00002 seconds.

The output of the simulations could be replayed as an animation which showed the interactions of the rigid bodies over time. The values of the actuator torques and forces and end effector orientation and position of any link can be plotted over time. In this study, the primary method of viewing the results was to play the animation of the robot. Then, by observation, it could be seen whether or not the simulation was set up correctly and whether or not the ball was caught by the arm. Actuator torques were then plotted.

In order to present the results and interpret the movement, extra geometries were added to the model as points of reference. A room was built around the robot arm, and a mounting place for the robot arm was added, as shown in Figure 4.5.

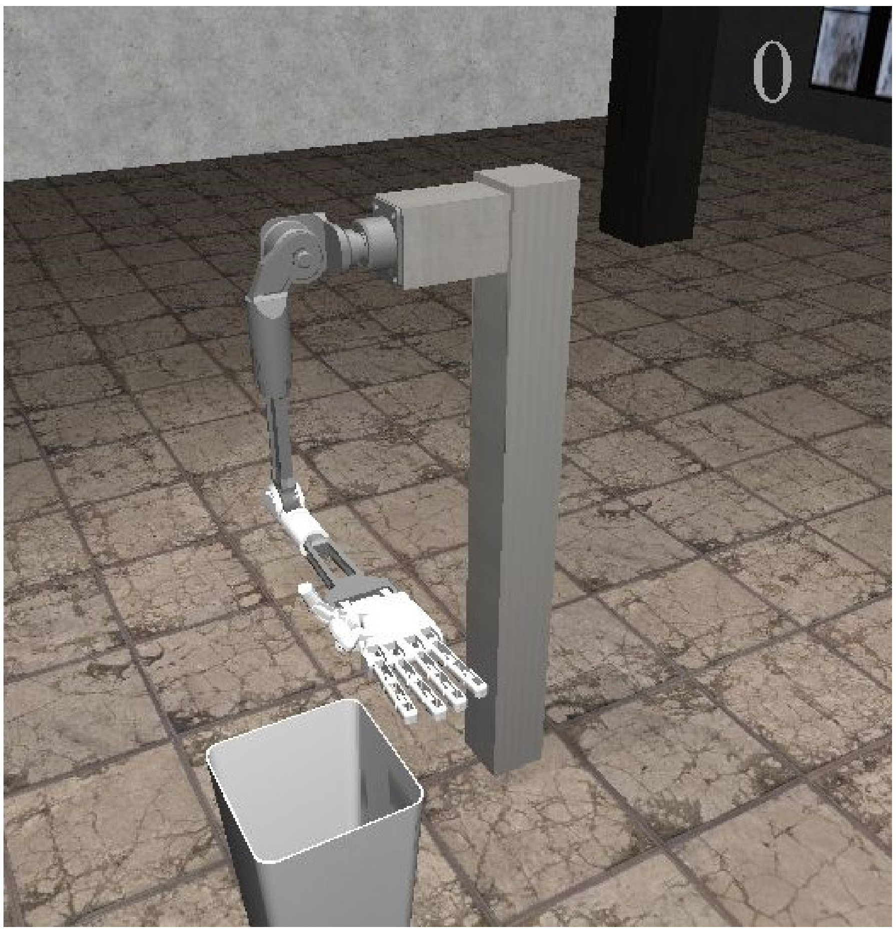


Figure 4.5. Simulation Environment

5. CONTROL STRATEGY

5.1 Robotic Arm Kinematics

The robot kinematics come directly from the geometric design and joint placement. An important output of this design is the joint offsets and distances. In this study, the Denavit-Hartenberg approach is used to develop the forward kinematic equations. The first step in this approach is to assign axes to each joint based on this pattern:

- The z-axis is along the joint axis.
- The x-axis is parallel to the common normal between the current and previous joint axis.
- The y-axis is determined by making a right-hand coordinate system based on the other axes.

This approach requires that specific offsets, distances, and angles are identified based on the previously determined coordinate systems which allow the robot arm to be correctly modeled. These parameters are:

- a: length of the common normal
- α: angle about common normal, from old z-axis to new z-axis
- d: offset along previous z-axis to common normal
- θ: angle about previous z-axis, from old x-axis to new x-axis

The coordinate systems were chosen for this robot arm and the parameters were determined and are listed in Table 5.1.

It may be noticed that there are more rows in the Denavit-Hartenberg parameters than there are joints. This is due to the geometry of the arm. When there are offsets

Table 5.1. Denavit-Hartenberg Parameters

Link	a	α	d	θ
-	0	$90°$	0	$90°$
1	0	$90°$	0.12	θ_1
2	0.09	$0°$	0.0075	θ_2
-	0.0474	$-90°$	0	$-90°$
3	0	$-90°$	0.231	θ_3
4	0.012	$0°$	0	θ_4
-	0.061	$0°$	0	$90°$
-	0	$-90°$	0	$-90°$
5	0	$90°$	0.19	θ_5
6	0.095	$-90°$	0	θ_6

in more than one direction from one joint to the next, an extra joint with a fixed angle was used to accommodate the additional offset.

5.1.1 Forward Kinematics

Once all the parameters were gathered, the equations for calculating the forward kinematics could then be found. The Denavit-Hartenberg parameters were then used to calculate the transformation matrix from the previous to the current joint using the following equation.

$$[T] = \begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & a\cos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & a\sin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

The above transformation matrix is calculated for each set of Denavit-Hartenberg parameters, and all of these are multiplied together to get a complete transformation matrix from the global coordinate system to the end effector's local coordinate system. However, it should be noted that there are six variables within this matrix. These six variables are the joint angles of each actuator. The final transformation matrix gives the position of the end effector's coordinate system (the top three value of the last column) and the rotation matrix (the top three rows of the first three columns). This calculation describes the forward kinematics of the system. Given the six joint angles of the system, the orientation and position of the end effector can be found.

5.1.2 Inverse Kinematics

The forward kinematics equations have been found in matrix form. The transformation matrix is a four-by-four matrix, but the last row is not useful. This leaves 12 values which are the rotation matrix (3x3) and the position components (3x1).

The next step is to find a way to solve for the six joint angles given a desired orientation and position of the robot. However, since there are only six variables, the joint angles, only six equations can be used to solve the inverse kinematics. Also, since the functions have many trigonometric functions, they are nolinear and difficult to solve.

With 12 possible equations and only six can actually be used, the next task is to determine which values are most important. The initial approach was to use the three positional components and the three components of the vector normal to the face of the hand. However, after initial testing of this approach, it was found that the hand would have an arbitrary rotation about the normal axis and this was not a reliable approach. The solution to this problem is to convert the rotation matrix into Euler angles. This compacts the orientation information from nine values into three. Then the six equations used to solve for the joint angles are the three Euler angles and the three positional components.

Converting the rotation matrix to Euler angles is a straightforward process. First, the order of rotation is chosen. In this case, the order is chosen to be rotation about z, x, and then the z-axis again. Then the Euler angles are found using the following equations.

$$\phi = \arctan 2(A_{31},\, A_{32}) \tag{5.2}$$

$$\theta = \arccos\left(A_{33}\right) \tag{5.3}$$

$$\psi = -\arctan 2(A_{13},\, A_{23}) \tag{5.4}$$

This formula only works when θ is between 0 and π. When $A_{33} = 0$, an alternative set of equations are used to calculate the Euler angles:

$$\phi = -\pi/2 \tag{5.5}$$

$$\theta = \pi/2 \tag{5.6}$$

$$\psi = \phi + \arctan 2(A_{12},\, A_{13}) \tag{5.7}$$

Once the six functions were decided upon (see Figure 5.1), the next step is to find a way to solve for the variables in these functions, specifically the joint angles. As mentioned earlier, these functions are nonlinear, so numerical methods are used to find a solution. Several methods could be used in this case, but the Newton-Raphson method was chosen because it is easy to implement and it converges quickly in this application.

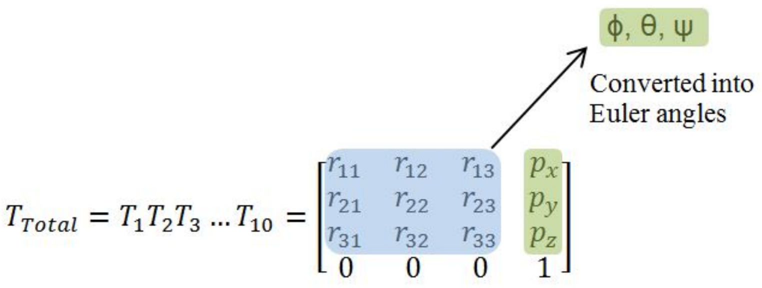

ϕ, θ, ψ
Converted into Euler angles
$T_{Total} = T_1T_2T_3 \ldots T_{10} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Figure 5.1. Six Functions

First, each equation is written in the form:

$$f(\theta_1,\, \theta_2,\, \theta_3,\, \theta_4,\, \theta_5,\, \theta_6) = 0 \tag{5.8}$$

Referencing Figure 5.1, the functions would become:

$$f_1 = \phi_{current} - \phi_{desired} = 0 \tag{5.9}$$

$$f_2 = \theta_{current} - \theta_{desired} = 0 \tag{5.10}$$

$$f_3 = \psi_{current} - \psi_{desired} = 0 \tag{5.11}$$

$$f_4 = p_{x_{current}} - p_{x_{desired}} = 0 \tag{5.12}$$

$$f_5 = p_{y_{current}} - p_{y_{desired}} = 0 \tag{5.13}$$

$$f_6 = p_{z_{current}} - p_{z_{desired}} = 0 \tag{5.14}$$

Then, the roots are found for each of the equations using the Newton-Raphson method. The Newton-Raphson method is a numerical solution which uses the derivative to converge on roots of an equation. However, in this application, the Jacobian is calculated numerically (with finite differences). The method is outlined below.

1. Start with an initial set of joint angles $\{\theta\}$.
2. Calculate the equations, $\{f\} = \{f(\{\theta\})\}$.
3. Stop if $\{f\} < \textit{tolerance}$. This means $\{\theta\}$ are satisfactory joint angles.
4. Calculate the 6x6 Jacobian matrix numerically using θ and $(\theta + \delta\theta)$.
5. Solve: $\{\Delta\theta\} = [J]^{-1}\{f(\{\theta\})\}$
6. Calculate $\{\theta\} = \{\theta\} + \{\Delta\theta\}$.
7. Go to step 2.

The tolerance in step 3 for both position and orientation components is 0.00025. The numerical Jacobian in step 4 is calculated using the following equation.

$$[J] = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \frac{\partial f_1}{\partial \theta_2} & \cdots & \frac{\partial f_1}{\partial \theta_6} \\ \frac{\partial f_2}{\partial \theta_1} & \frac{\partial f_2}{\partial \theta_2} & \cdots & \frac{\partial f_2}{\partial \theta_6} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_6}{\partial \theta_1} & \frac{\partial f_6}{\partial \theta_2} & \cdots & \frac{\partial f_6}{\partial \theta_6} \end{bmatrix} \tag{5.15}$$

Specifically, the Jacobian components are calculated as shown in the following examples.

$$\frac{\partial f_1}{\partial \theta_1} = \frac{f_1(\theta_1 + \delta\theta,\, \theta_2,\, \theta_3,\, \theta_4,\, \theta_5,\, \theta_6) - f_1(\theta_1,\, \theta_2,\, \theta_3,\, \theta_4,\, \theta_5,\, \theta_6)}{\delta\theta} \tag{5.16}$$

$$\frac{\partial f_1}{\partial \theta_2} = \frac{f_1(\theta_1,\, \theta_2 + \delta\theta,\, \theta_3,\, \theta_4,\, \theta_5,\, \theta_6) - f_1(\theta_1,\, \theta_2,\, \theta_3,\, \theta_4,\, \theta_5,\, \theta_6)}{\delta\theta} \tag{5.17}$$

$$\frac{\partial f_2}{\partial \theta_1} = \frac{f_2(\theta_1 + \delta\theta,\, \theta_2,\, \theta_3,\, \theta_4,\, \theta_5,\, \theta_6) - f_2(\theta_1,\, \theta_2,\, \theta_3,\, \theta_4,\, \theta_5,\, \theta_6)}{\delta\theta} \tag{5.18}$$

The value used for $\delta\theta$ in this study is 0.12 radians. The rest of the components are calculated in the same way.

The inverse of the Jacobian in step six is found one column at a time using Gauss elimination with partial pivoting. This process uses the fact that when $[K] = [J]^{-1}$, $[J][K] = [I]$. $[K]$ is solved for by breaking it into separate columns. The first column of $[K]$ is found by solving:

$$[J][K_1] = \begin{Bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{Bmatrix} \tag{5.19}$$

Similarly, the second column can be solved:

$$[J][K_2] = \begin{Bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{Bmatrix} \tag{5.20}$$

The rest of the columns of the inverse are solved similarly.

This method was tested by selecting a desired orientation and position, and producing a set of joint angles using the above algorithm. Then these joint angles were entered into the forward kinematics equations to see if the orienation and position equaled the desired values.

The Newton-Raphson method (Jacobian calculated using finite differences) is used in this study to solve the inverse kinematics equations. This means that the joint angles can be found for the six actuators which satisfy a desired orientation and position of the end effector.

5.2 Robotic Arm Control

Once an algorithm for solving the inverse kinematics is found, the robot's motion control is developed. The problem is determining how to move the robotic arm from one position and orientation to another. The setup for the robotic arm in the simulation environment is outlined below.

5.2.1 Robotic Arm Setup

The arm is loaded into the multibody dynamics software with an initial configuration. The upper arm is vertical, the forearm is horizontal, the palm is facing upward, and the fingers and thumb are opened. The ball to be caught is initially at a distance of approximately 3.1 meters from the base of the robotic arm. It is given an initial velocity to simulate being thrown. The final gains used for controlling the actuators are listed in Table 5.2.

Table 5.2. Actuator Gains

Actuator	Name	P ($\frac{N}{rad}$)	D ($\frac{N\,s}{rad}$)
1	Shoulder 1	1200	5
2	Shoulder 2	1000	3
3	Shoulder 3	1500	5
4	Elbow	1000	3
5	Forearm	600	2
6	Wrist	500	2

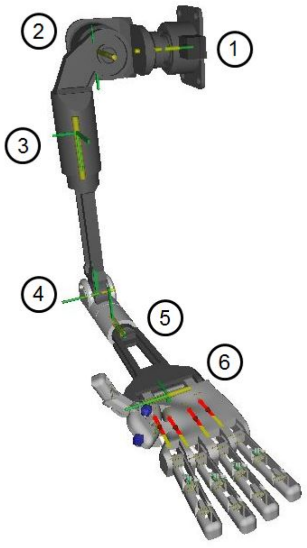


Figure 5.2. Numbered Actuators

5.2.2 Determining the Catch Position

The first step in the algorithm is to read the initial velocity vector from the ball. Then, the intersection of the ball's parabolic trajectory and a predetermined catch plane is found. A single catch plane is used in this study, and it is set to be 0.35 meters from the base of the robotic arm (see Figure 5.3).

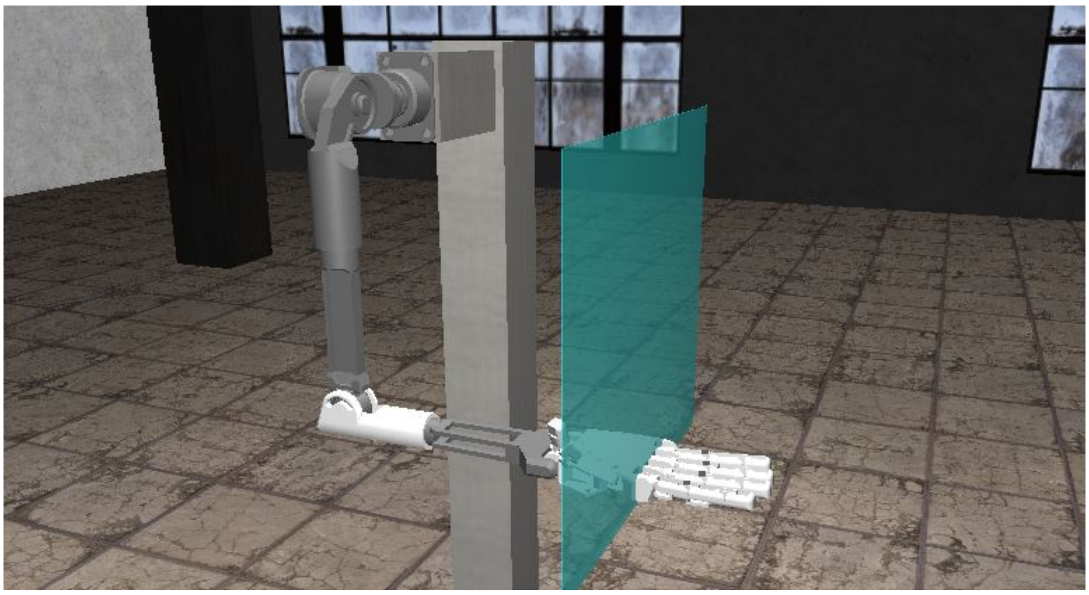


Figure 5.3. Catch Plane

This intersection point is the desired position for the end effector of the robotic arm. The desired orientation is so that the local vector normal to the palm is tangent to the ball's trajectory at the intersection point. Also, the local vector which points to the side of the palm is set to stay pointed in the same initial direction (at start of the arm's motion) during the catch. This fully constrains the orientation of the hand (Figure 5.4).

Once the hand is in the catch position the fingers and thumb are programmed to close just before the ball impacts the hand. This time is chosen using the time calculated at which the trajectory of the ball reaches the catch plane, minus a small offset which is experimentally tuned. In this simulation, the fingers are given the close signal 0.02 seconds before the intersection time. This allows the fingers and thumb to close around the ball just as it enters the palm of the hand.

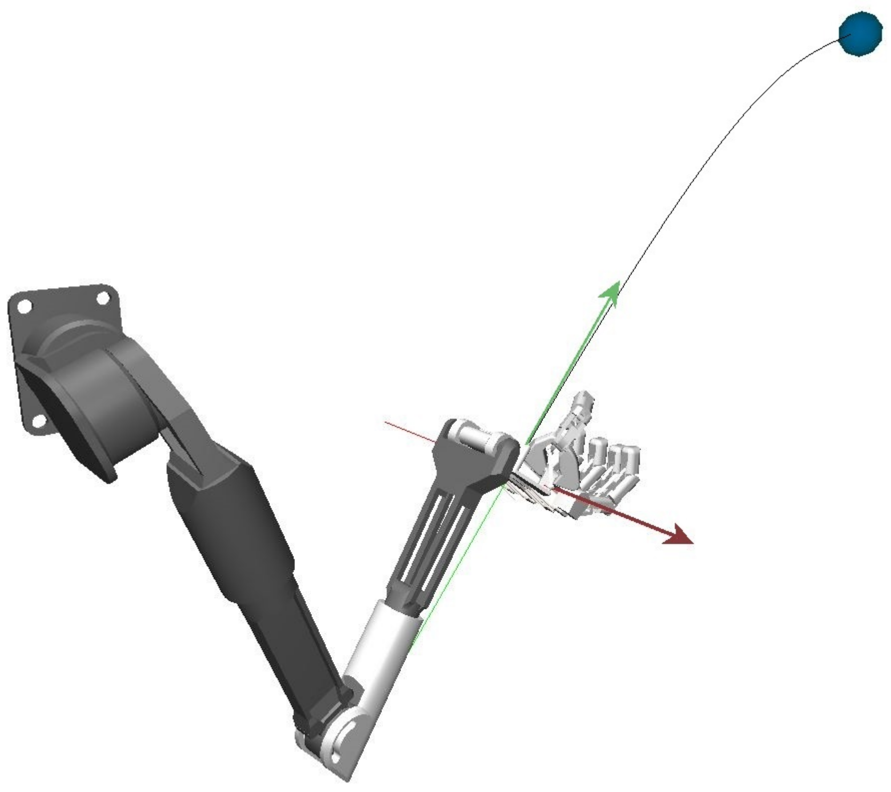


Figure 5.4. Orientation Vectors (Green vector is normal to palm, red vector is normal to palm normal)

5.2.3 Motion Control

The next step in the algorithm is to move the robotic arm from the current position to the desired catch position. Instead of using the inverse kinematics algorithm to find the desired joint angles from the initial position, a path is generated which allows the algorithm to solve small problems of incremental movement. This allows the robotic arm to be lead along the desired path to the final state. The numerical method is very compliant to this approach since the solutions are very close to the initial guesses, and only four Newton-Raphson iterations are usually needed for each solution.

Each of the six components of position and rotation are interpolated from the initial state to the desired state over a time shorter than it takes the ball to get to the catch plane. This way, the hand is in position when the ball reaches the catch location. The path is point-to-point linear.

However, this motion profile over time is not linear. Instead, a cubic polynomial is used in order to meet desired boundary conditions. The boundary conditions are the initial and final positions, as well as initial and final velocities, which are set to zero. The general position, velocity, and acceleration profiles are shown in Figures 5.5, 5.6, and 5.7.

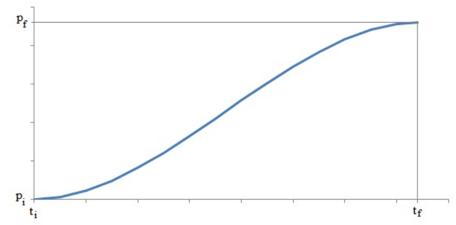


Figure 5.5. Position Profile

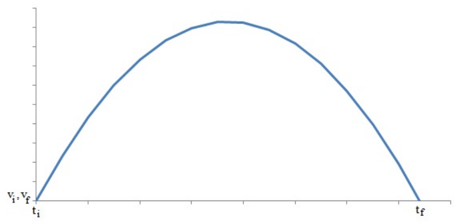


Figure 5.6. Velocity Profile

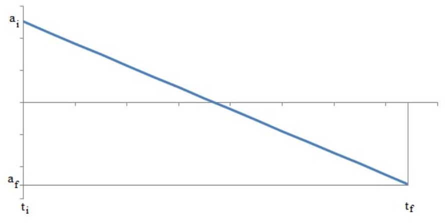


Figure 5.7. Acceleration Profile

This method of incremental, point-to-point motion is easy to use in other applications. As a part of the tests discussed later, this method can be used to move the robotic arm to any location and orientation possible.

5.2.4 Control Script

The control system is scripted within the simulation environment, and the script is run every time step. This is how signal data is communicated to and from the actuators. The joint angles are read from the actuators and the positions from the linear motors. Torques and forces are calculated in the algorithm and sent to the actuators.

6. TEST AND RESULTS

This section describes the testing performed in this study and the associated results. The tests presented are not computed real-time in order to remove the requirement for expensive computational equipment. Instead, the simulations are solved and the transient data is saved and played back at a real-time rate.

6.1 Test Setup

The first test in this study was used to fine-tune the PD parameters of the actuator controllers. This test was to run a simulation of the arm holding the initial position under the effect of gravity. The results were played back as a visualization of the system, and the parameters were reduced until the motion of the end effector was under five millimeters. Then the damping parameters were set by giving a desired angle for each actuator and watching the playback for overshoot and oscillation. The damping parameter was set so that the response was slightly overdamped (no residual oscillations while holding a position with gravity).

Then, once the system was configured and the control system was found to be working as desired, the catch tests were prepared. These tests consist of a ball with an initial velocity whose trajectory intersects the catch plane. An area is plotted with a five-by-six grid of catch positions on the catch plane. Each point is tested by configuring a ball whose trajectory passes through that point. This test grid is illustrated in Figure 6.1. The reason this area is chosen is because it is most suitable for the catch orientation described in the previous chapter. In this catch region, it is easy to constrain the local vector pointing from the side of the palm and still provide a good angle of approach for catching the ball. In order to test other regions, a different catch orientation may be required.

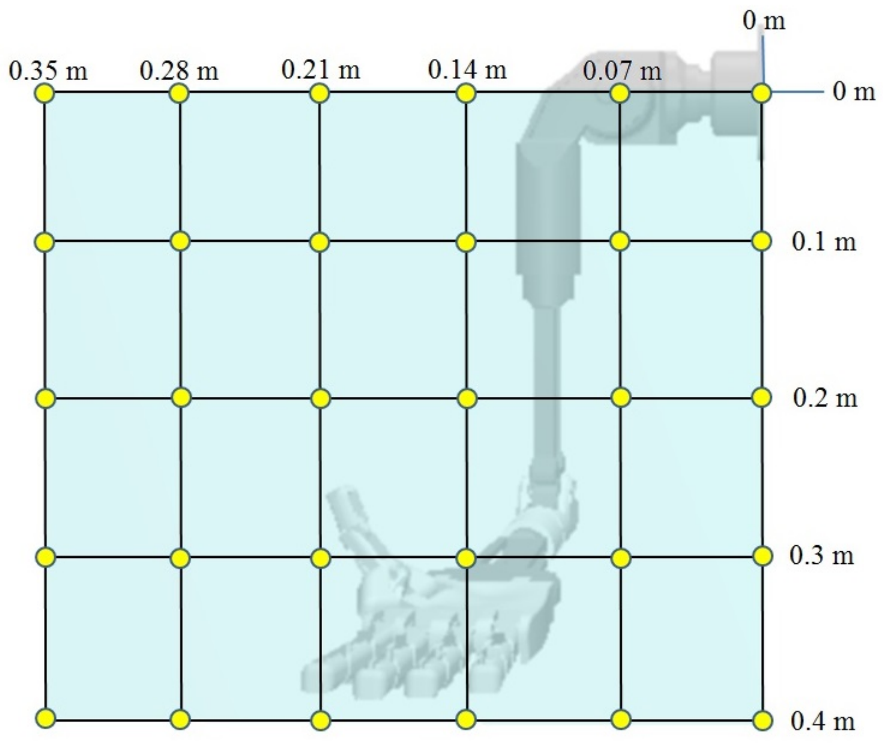


Figure 6.1. Catch Test Points

The initial conditions of the ball are the components of position and velocity. The initial position is the only parameter that changes between tests. The initial velocity is set to be 5.2 meters per second at a 45 degree angle. Also, the distance between the catch plane and the initial position of the ball is set so that the ball would intersect the plane at the same height at which it began. This makes it easy to set up each catch test. Using this setup, the ball intersects the plane just after 0.74 seconds of flight.

The ball in these tests has a mass of 0.12 kilograms, which is slightly less mass than a regulation baseball, and a diameter of 50 millimeters. The contact properties of the ball were set with a friction coefficient of 0.3, and the normal contact stiffness and damping is set experimentally until the playback looked correct for a plastic on metal contact. Since the hand completely encompasses the ball in the catch process, the exact contact stiffness is not required.

In addition to catching the ball at each of these points, the hand would drop the ball in a bin in a fixed location. A drop position and orientation is manually selected based on the position of the bin, and the same control algorithm used for moving to the catch position is used in this case. The initial setup of the robotic arm with the bin is shown in Figure 6.2.

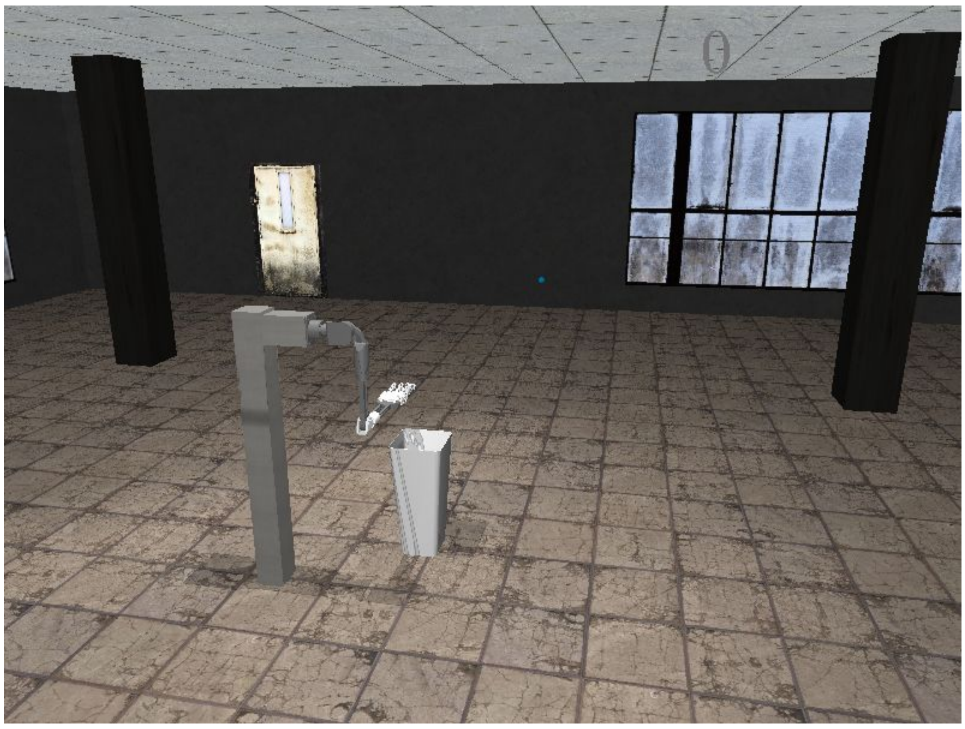


Figure 6.2. Initial setup

Once the simulation has been configured and tested, the entire catch grid is tested using a batch function in the multibody dynamics software.

6.2 Results

The results from the tests are presented in this section. After the simulations of the 30-point grid catch test are solved, the results are played back. Each 2.2 second simulation took approximately 45 minutes to solve using a machine with an Intel i7 clocked at 2.93 GHz. In each of the 30 cases, the ball is successfully caught and dropped into the bin. An example of one of the tests is shown in the frames in Figure 6.3 (figures showing catches for all tests are shown in Appendix B). The most critical aspect of these tests is the catch itself. Therefore, sequenced views of this event are shown in Figures 6.4, 6.5, and 6.6. More focused views of the hand grasping the ball after the catch are shown in Figures 6.7, 6.8, 6.9, 6.10, 6.11, and 6.12. The drop sequence is shown in Figures 6.13 and 6.14. A sample of the tracking error over the entire simulation is shown in Figures 6.15 and 6.16. Tracking error is defined as the difference between actual and desired position components or Euler angles. Another important part of the results are the actuator torques during the tests. A sample of the data is shown in Figures 6.17, 6.18, 6.19, and 6.20. It can be seen that the imact of the ball causes an oscillating signal in the actuator torques. This is because the system applies a force to return to the desired position.

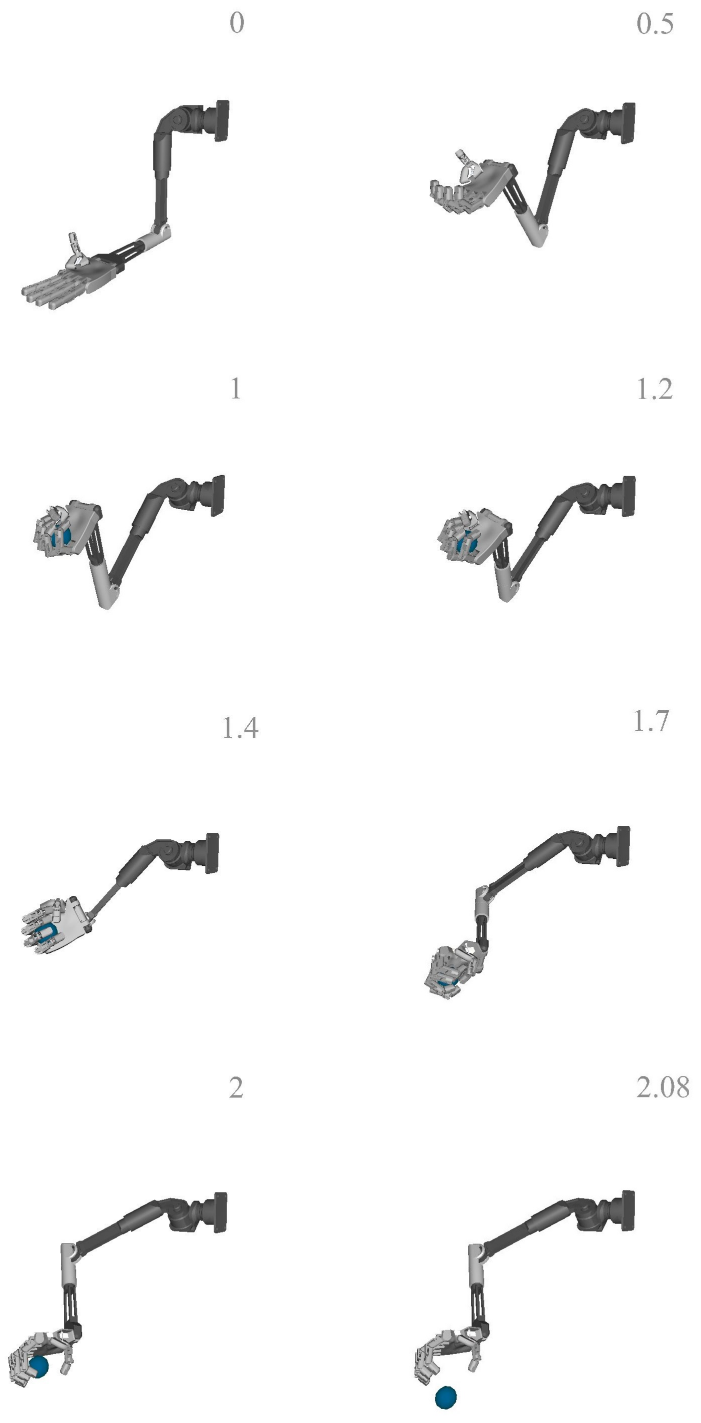


Figure 6.3. Robotic Arm Catching and Dropping the Ball

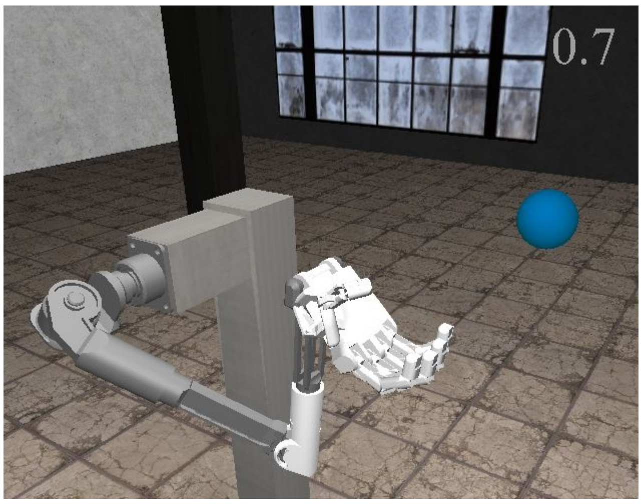


Figure 6.4. System at 0.7 seconds

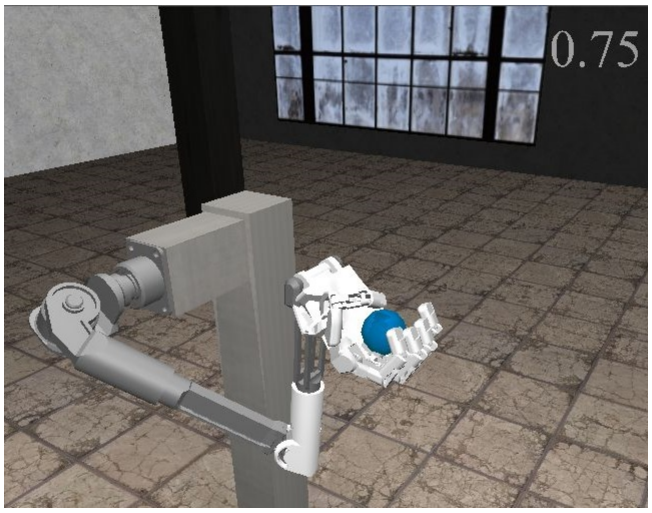


Figure 6.5. System at 0.75 seconds

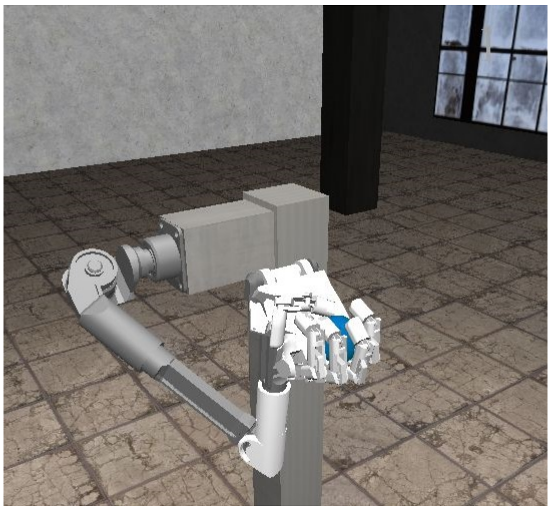


Figure 6.6. System at 1.0 seconds

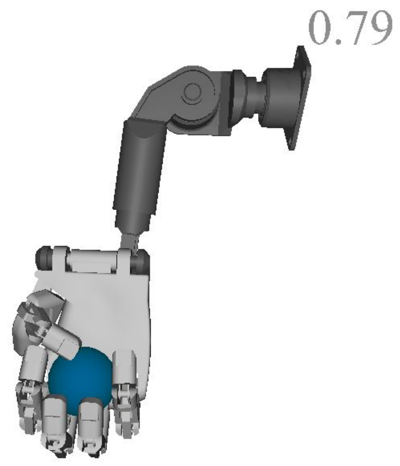


Figure 6.7. Grasp at 0.79 seconds

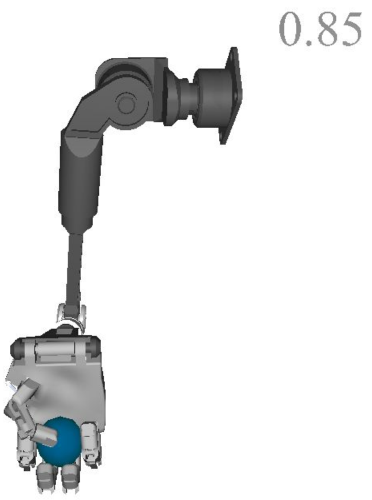


Figure 6.8. Grasp at 0.85 seconds

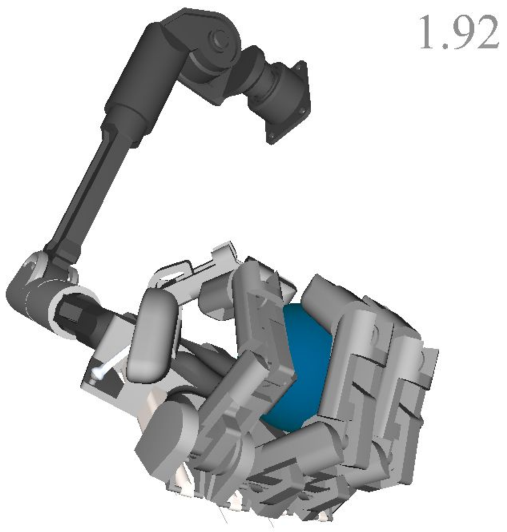


Figure 6.9. Grasp at 1.92 seconds

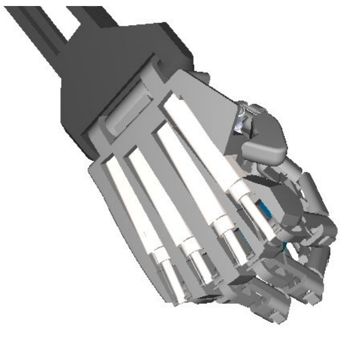


Figure 6.10. Linear Actuators of Hand During Grasp

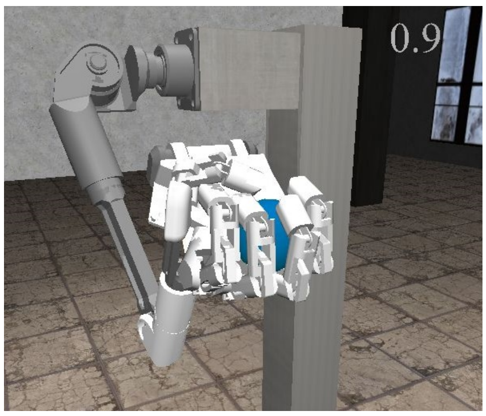


Figure 6.11. Close-up View of the System at 0.9 seconds

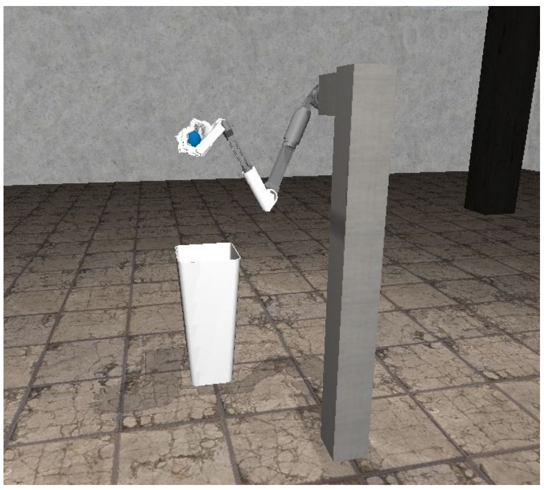


Figure 6.12. Side View of the System at 0.9 seconds

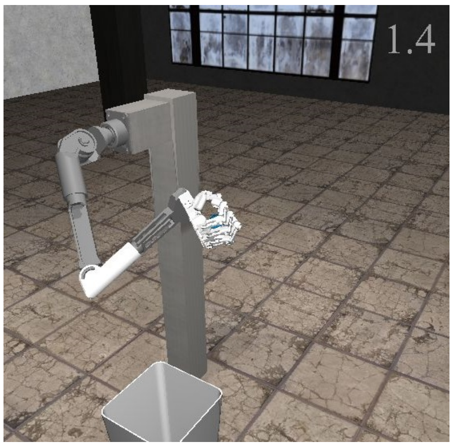


Figure 6.13. System at 1.4 seconds

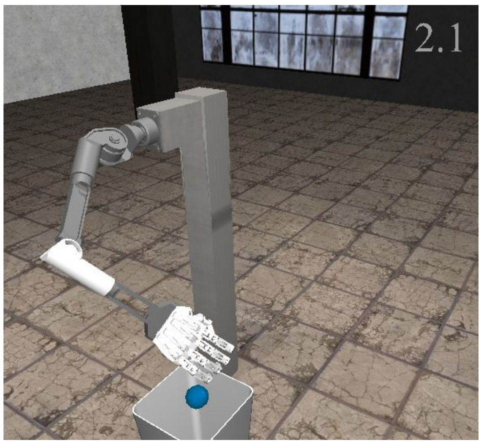


Figure 6.14. System at 2.1 seconds

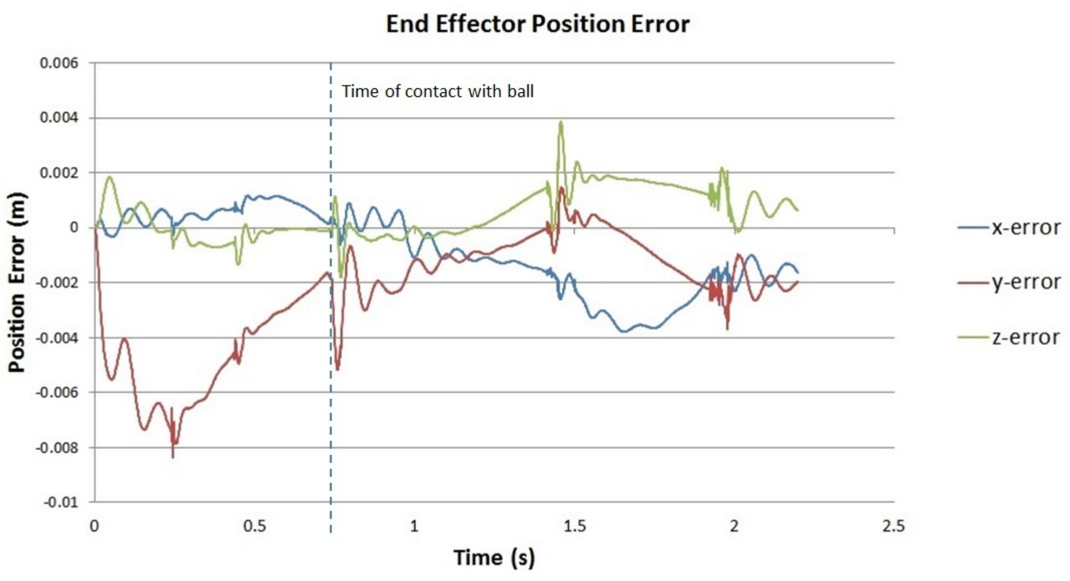


Figure 6.15. Position Error during Entire Simulation

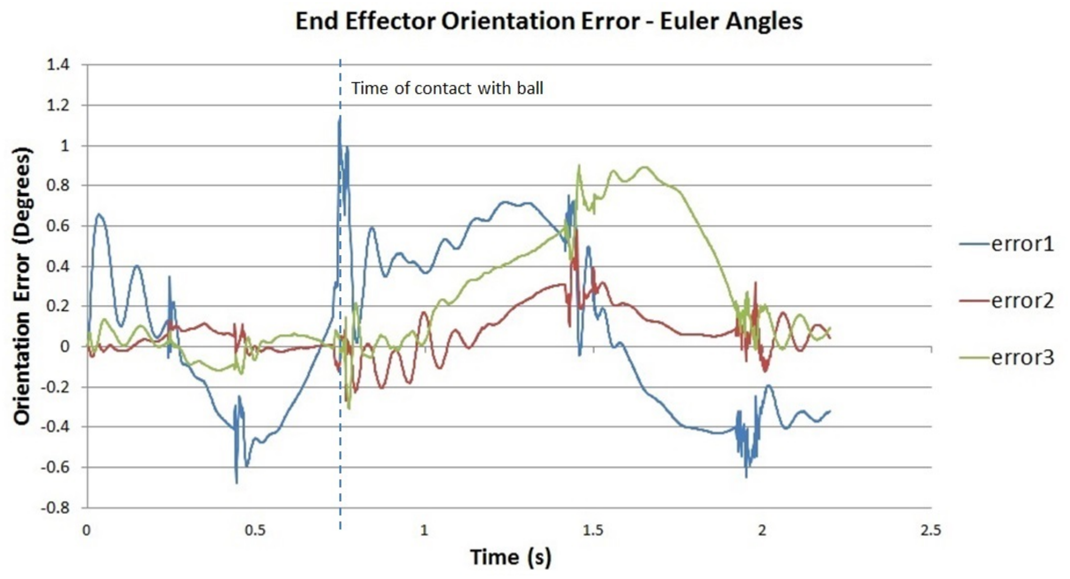


Figure 6.16. Orientation Error during Entire Simulation

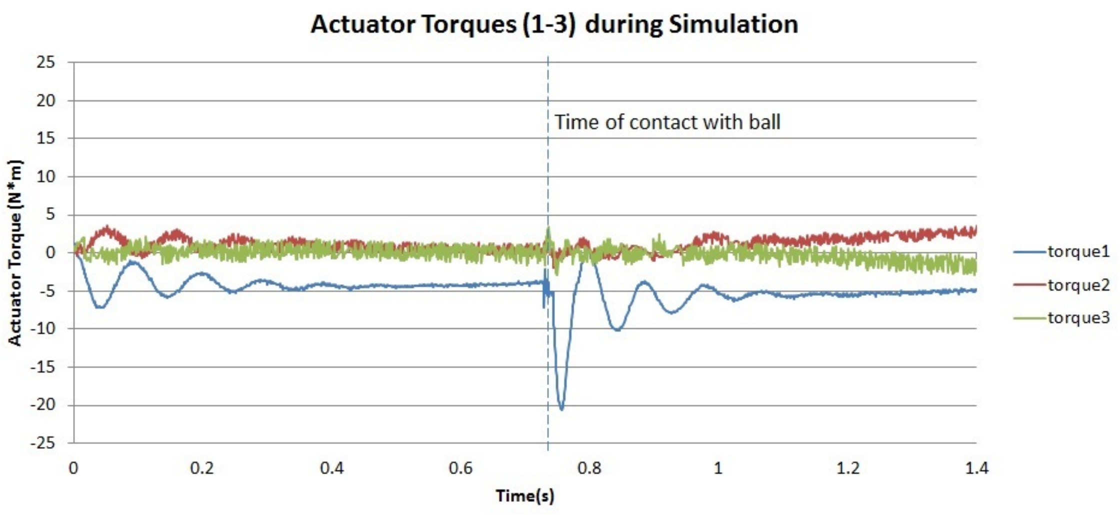


Figure 6.17. Actuator Torques 1-3

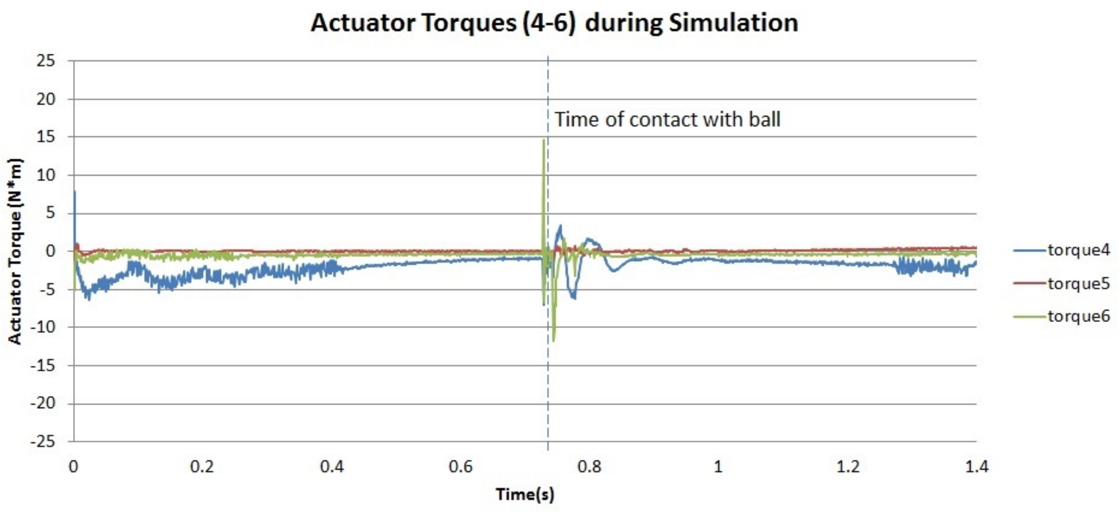


Figure 6.18. Actuator Torques 4-6

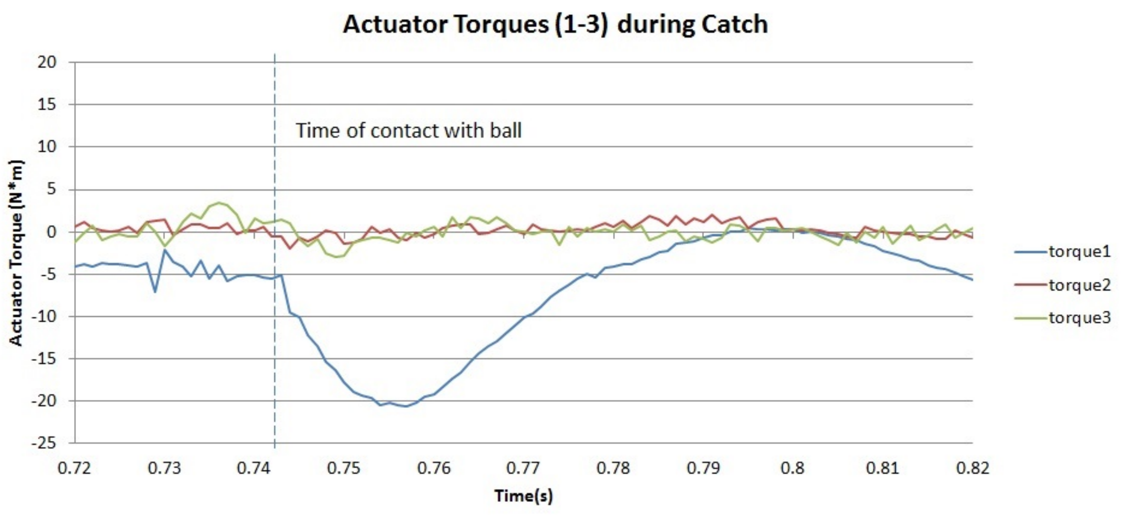


Figure 6.19. Actuator Torques 1-3 during Catch (zoomed from Figure 6.17)

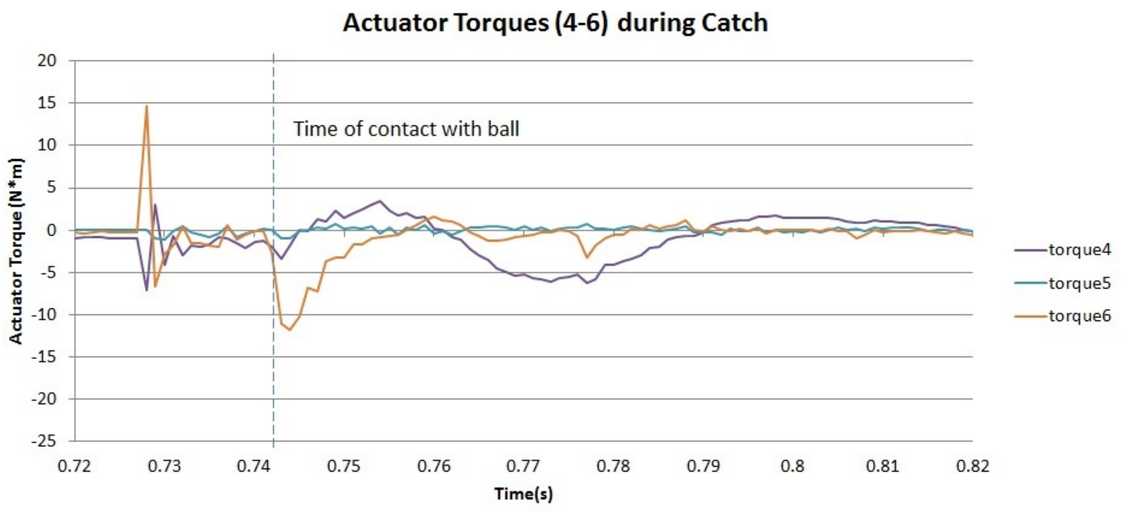


Figure 6.20. Actuator Torques 4-6 during Catch (zoomed from Figure 6.18)

6.3 Summary

The arm successfully caught all balls within the catch region. The end effector position and orientation errors were low as desired (see Figures 6.21 and 6.22). The maximum position error (three-dimensional distance) over the entire simulation for all tests is 13.6 mm, and the maximum orientation angle error is 4.3°. These values occur when the ball is caught and the impact causes a disturbance. The maximum position and orientation error for all tests before the catch is 8.4 mm and 4.3°, respectively. The averages of the absolute values of the position and orientation errors over the entire simulation time for all tests are 3.3 mm and 0.27°, respectively. Noise exists in the actuator torques, and in future work, this should be minimized. The actuator response has a diminishing oscillation which goes to zero amplitude by 0.6 seconds in the example shown (Figure 6.17). The actuator torques during the catch are high, especially the first shoulder actuator torque. In future development, a velocity-matching approach may help to reduce these torques.

Test	Test Number
0.0, 0.0	1
0.0, 0.07	2
0.0, 0.14	3
0.0, 0.21	4
0.0, 0.28	5
0.0, 0.35	6
0.1, 0.0	7
0.1, 0.07	8
0.1, 0.14	9
0.1, 0.21	10
0.1, 0.28	11
0.1, 0.35	12
0.2, 0.0	13
0.2, 0.07	14
0.2, 0.14	15
0.2, 0.21	16
0.2, 0.28	17
0.2, 0.35	18
0.3, 0.0	19
0.3, 0.07	20
0.3, 0.14	21
0.3, 0.21	22
0.3, 0.28	23
0.3, 0.35	24
0.4, 0.0	25
0.4, 0.07	26
0.4, 0.14	27
0.4, 0.21	28
0.4, 0.28	29
0.4, 0.35	30

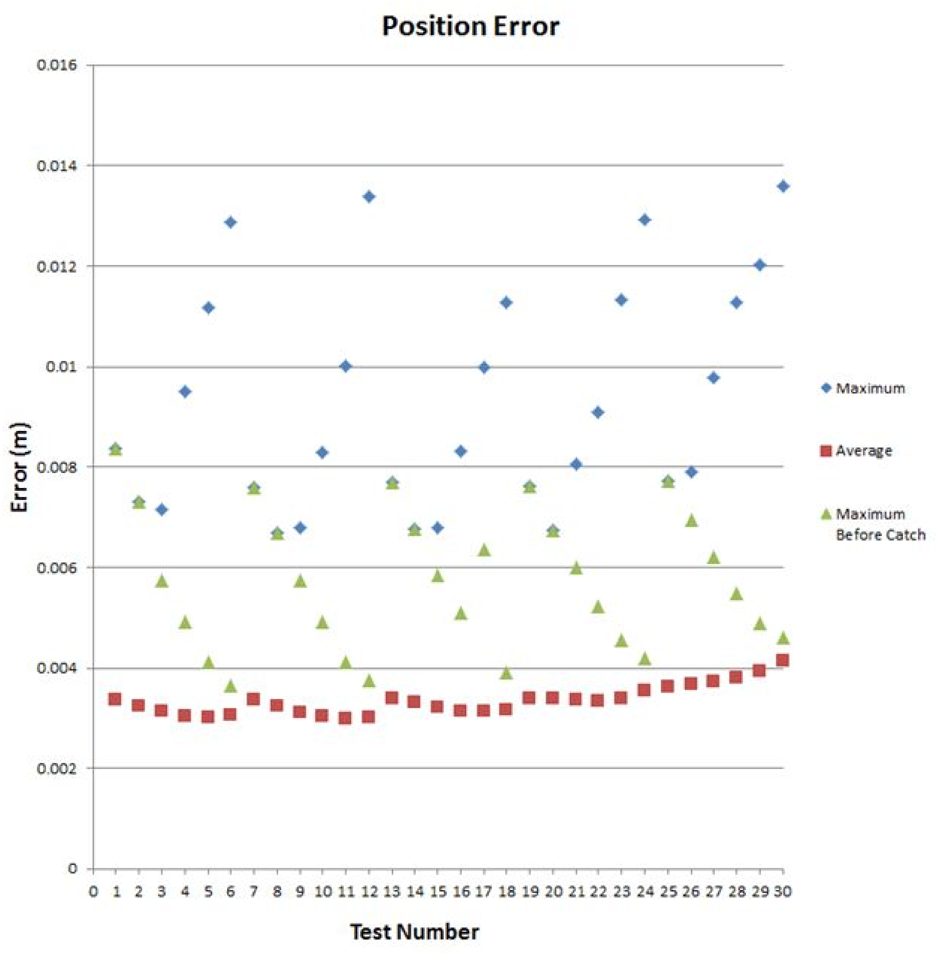


Figure 6.21. Position Error for all Tests

Test	Test Number
0.0, 0.0	1
0.0, 0.07	2
0.0, 0.14	3
0.0, 0.21	4
0.0, 0.28	5
0.0, 0.35	6
0.1, 0.0	7
0.1, 0.07	8
0.1, 0.14	9
0.1, 0.21	10
0.1, 0.28	11
0.1, 0.35	12
0.2, 0.0	13
0.2, 0.07	14
0.2, 0.14	15
0.2, 0.21	16
0.2, 0.28	17
0.2, 0.35	18
0.3, 0.0	19
0.3, 0.07	20
0.3, 0.14	21
0.3, 0.21	22
0.3, 0.28	23
0.3, 0.35	24
0.4, 0.0	25
0.4, 0.07	26
0.4, 0.14	27
0.4, 0.21	28
0.4, 0.28	29
0.4, 0.35	30

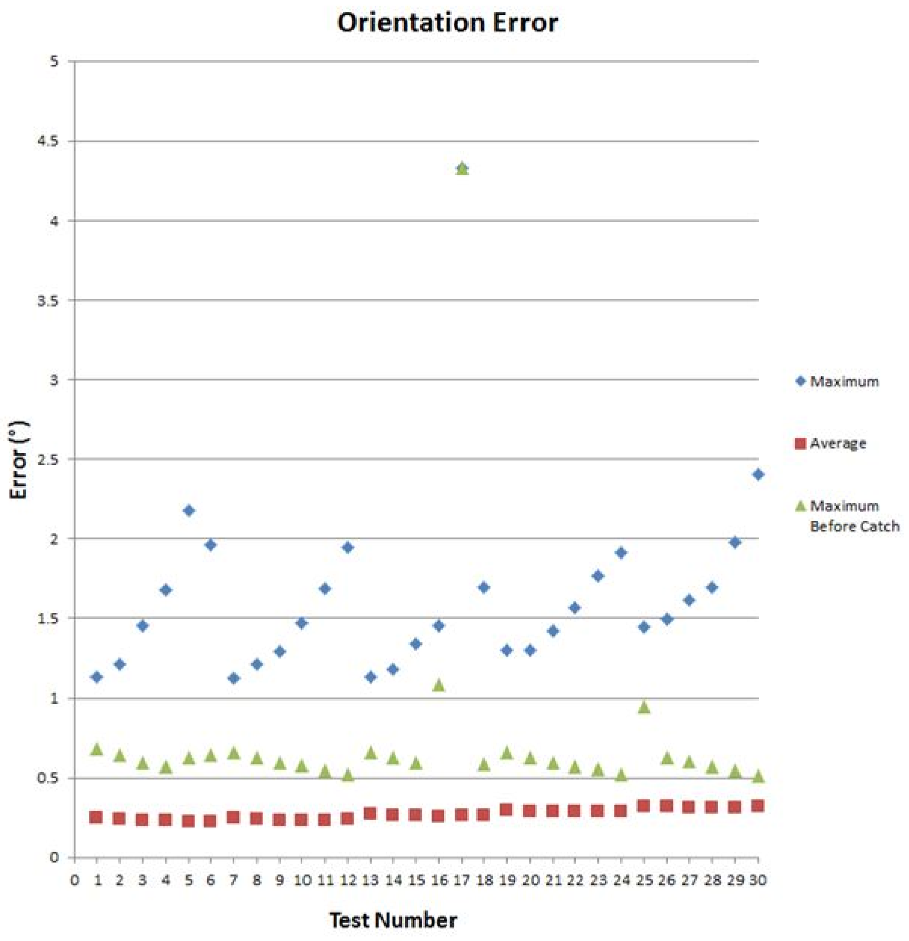


Figure 6.22. Orientation Error for all Tests

7. CONCLUSION AND FUTURE WORK

7.1 Conclusion

In this study, a robotic humanoid arm was developed. This included the design of the geometric model, development of the control system and kinematics approach, simulation setup, and virtual environment tests. The objectives of this study were the catch rate of the system for the tested region, the tracking error, and the actuator torques.

The catch rate for the tested region is 100%, based on the 30 point test which was completed. The tracking error is within 3.5 mm and 0.3° on average. The maximum tracking error is 13.6 mm (distance), and 4.3° (Euler angle). The actuator torques were stable through point-to-point motion, but in some tests, the actuators would exert large, instantaneous torques during the catch (in excess of 200 Newton-meters). The actuators which reached these values were the first shoulder actuator and the elbow actuator. A solution for reducing these torques may be a velocity matching control approach which would reduce the impact, or simply adding torque limits that would allow the arm to be displaced for a small period of time.

7.2 Future Work

In the course of this study, a robotics platform was developed which can be used for further robotics research and development. The control system and simulation developed in this study can be easily modified and enhanced for finer, more detailed and accurate studies. While a large step was accomplished in this study, many more areas could be explored with the developed model.

A focus on actuator torque would enhance the usability of this simulation platform, since all actuators have a physical torque limit. Optimizing and developing more catch orientations may allow for catches outside of the region tested in this study. Also, a method for handling singularities would make this system more robust. Initial tests in this study show that singular positions do exist in this system, although the catch tests never involved singular positions. Several methods have been developed in other studies to mitigate the effects of singularities, and integrating a technique would allow the robotic arm to freely use the entire workspace. Obstacle avoidance would become necessary if this system is integrated with a larger humanoid system. Integrating the vision system into the current study would also be a way to further the current study. Finally, developing a cooperative interaction with a second arm would greatly advance this system toward the end goal of a complete humanoid robotic system.

A related application which was briefly experimented with in this study is ball throwing. A minor change was made to the control scheme to program the robot to throw the ball back after the catch. Figure 7.1 illustrates this test. This quick experimentation shows how the platform developed in this study can readily be used to study other areas or further develop current applications.

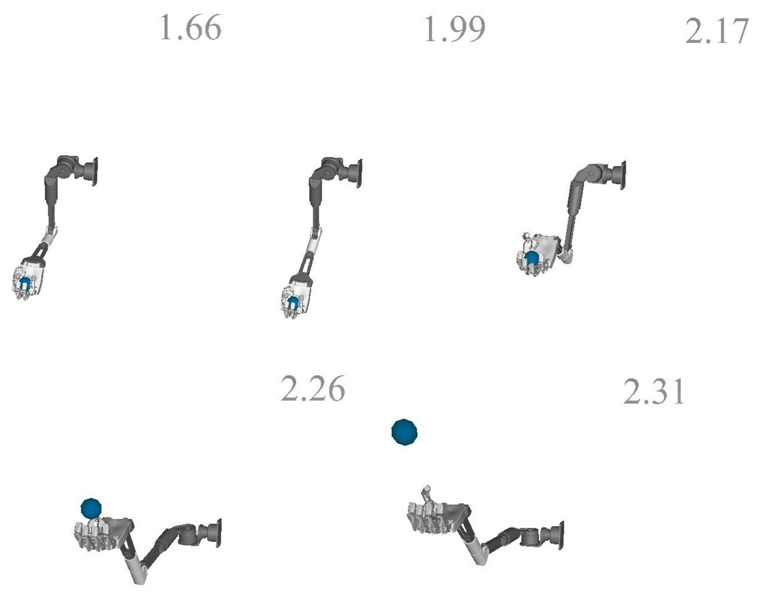


Figure 7.1. Robotic Arm Throwing the Ball

The system developed in this study could be applied to other applications, such as grasping stationary and moving objects. These studies could benefit many fields of study, such as humanoid robots in rescue and medical applications, as well as prosthetics development.

The control system developed for this robotic arm is the main focus of this study, and this approach could also be applied to walking and other limb functions. While some adaptation of the script would be necessary, the same general approach could be used. Future work with this algorithm could eventually include whole body control.

The ultimate purpose of this work is to advance the study of humanoid robotics. This study developed a simulation of a humanoid robotic arm, its control system, and a set of tests to determine the performance of the arm's catching abilities. The presented robotic arm and control system design produced desired results in the simulated tests.

The work and tools presented in this study provide the opportunity for further research on humanoid robotics and other applications.

LIST OF REFERENCES

LIST OF REFERENCES

[1] ASIMO - The Worldwide ASIMO Site. http://world.honda.com/ASIMO/, (Last Accessed 2014).

[2] Robonaut: Home. http://robonaut.jsc.nasa.gov/, (Last Accessed 2014).

[3] DRC Trials. http://www.theroboticschallenge.org/, (Last Accessed 2014).

[4] S. Behnke. Humanoid robots - from fiction to reality? *Knstliche Intelligenz*, 2008.

[5] B. Qu and B. Xu. A control algorithm of general 6R mechanic arm based on inverse kinematics. In *2011 IEEE International Conference on Computer Science and Automation Engineering (CSAE)*, 2011.

[6] A.A. Goldenberg, B. Benhabib, and R.G. Fenton. A complete generalized solution to the inverse kinematics of robots. *Robotics and Automation, IEEE Journal of*, 1985.

[7] K. Gupta and K. Kazerounian. Improved numerical solutions of inverse kinematics of robots. In *Proceedings, 1985 IEEE International Conference on Robotics and Automation.*, 1985.

[8] C.M. Gosselin, J. Cote, and D. Laurendeau. Inverse kinematic functions for approach and catching operations. *IEEE Transactions on Systems, Man and Cybernetics*, 1993.

[9] T. Asfour and R. Dillmann. Human-like motion of a humanoid robot arm based on a closed-form solution of the inverse kinematics problem. In *Proceedings, 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003).*, volume 2, pages 1407–1412 vol.2, Oct 2003.

[10] G. Yang, W. Lin, M.S. Kurbanhusen, C.B. Pham, and S.H. Yeo. Kinematic design of a 7-DOF cable-driven humanoid arm: a solution-in-nature approach. In *Proceedings, 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics.*, pages 444–449, 2005.

[11] W. Chen, Q. Chen, J. Zhang, and S. Yu. Forward kinematics and workspace analysis for a 7-DOF cable-driven humanoid arm. In *IEEE International Conference on Robotics and Biomimetics, 2006. ROBIO '06.*, pages 1175–1180, Dec 2006.

[12] M. Folgheraiter and G. Gini. A bio-inspired control system and a VRML simulator for an autonomous humanoid arm. *IEEE Humanoids*, 2003.

[13] Y. Ogura, H. Aikawa, K. Shimomura, A. Morishima, H. Lim, and A. Takanishi. Development of a new humanoid robot WABIAN-2. In *Proceedings, 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 76–81, May 2006.

[14] C. Smith and H.I. Christensen. Using COTS to construct a high performance robot arm. In *2007 IEEE International Conference on Robotics and Automation*, 2007.

[15] Z. Lin, V. Zeman, and R.V. Patel. On-line robot trajectory planning for catching a moving object. In *Proceedings, 1989 IEEE International Conference on Robotics and Automation, 1989.*, 1989.

[16] D. Chwa, J. Kang, and J. Choi. Online trajectory planning of robot arms for interception of fast maneuvering object under torque and velocity constraints. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 2005.

[17] B. Hove and J. Slotine. Experiments in robotic catching. In *American Control Conference, 1991*, 1991.

[18] B. Bauml, T. Wimbock, and G. Hirzinger. Kinematically optimal catching a flying ball with a hand-arm-system. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[19] B. Bauml, O. Birbach, T. Wimbock, U. Frese, A. Dietrich, and G. Hirzinger. Catching flying balls with a mobile humanoid: System overview and design considerations. In *2011 11th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, 2011.

[20] R. Wagner, U. Frese, and B. Bauml. Real-time dense multi-scale workspace modeling on a humanoid robot. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5164–5171, Nov 2013.

[21] Department of Defense. Anthropometry of U.S. military personnel, February 1991.

[22] B. Cesqui. Catching a ball at the right time and place: Individual factors matter. *PLoS ONE*, 2012.

[23] ASA :: Products :: DIS :: Features. http://ascience.com/DIS.htm, (Last Accessed 2014).

[24] T. Wasfy. Modeling spatial rigid multibody systems using an explicit-time integration finite element solver and a penalty formulation. In *ASME 2004 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 901–908. American Society of Mechanical Engineers, 2004.

[25] T. Wasfy. Asperity spring friction model with application to belt-drives. In *Paper No. DETC2003-48343, Proceeding of the DETC: 19th Biennial Conference on Mechanical Vibration and Noise*, 2003.

[26] T. Wasfy and M. Leamy. Modeling the dynamic frictional contact of tires using an explicit finite element code. In *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 2381–2388. American Society of Mechanical Engineers, 2005.

[27] T. Wasfy and J. O'Kins. Finite element modeling of the dynamic response of tracked vehicles. In *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 1999–2009. American Society of Mechanical Engineers, 2009.

[28] C.S. Lovchik and M.A. Diftler. The robonaut hand: a dexterous robot hand for space. In *Proceedings. IEEE International Conference on Robotics and Automation*, 1999.

APPENDICES

A. CONTROL SYSTEM SCRIPT

This is the script used within the multibody dynamics simulation software to control the robotic arm system.

```
javascript:

var time = parseFloat(IvWorld.DISrun.solutionTime);
var tolerance = 0.00025;
var dt = parseFloat(IvWorld.DISrun.timeStep);

var count = 0;

var f1=[0, 0, 0, 0, 0, 0];
var posRot=[0, 0, 0, 0, 0, 0];
var blank=[0, 0, 0, 0, 0, 0];

var x1=[parseFloat(IvWorld.DISrunactShoulder1.angle), parseFloat(IvWorld.
    DISrunactShoulder2.angle), parseFloat(IvWorld.DISrunactShoulder3.angle),
    parseFloat(IvWorld.DISrunactUpperArm.angle), parseFloat(IvWorld.DISrunactElbow.
    angle), parseFloat(IvWorld.DISrunactForearm.angle)];

if (time==0){
    var pos1 = parseFloat(IvWorld.DISrunProjectile.x1);
    var pos2 = parseFloat(IvWorld.DISrunProjectile.x2);
    var pos3 = parseFloat(IvWorld.DISrunProjectile.x3);

    var vel1 = parseFloat(IvWorld.DISrunProjectile.v1);
    var vel2 = parseFloat(IvWorld.DISrunProjectile.v2);
    var vel3 = parseFloat(IvWorld.DISrunProjectile.v3);
    var speed = Math.sqrt(vel1*vel1+vel2*vel2+vel3*vel3);

    var int_plane = -0.35;
    var t = absV((int_plane-pos3)/vel3);

    //Position at intersection
    pos1 = pos1 + vel1*t;
    pos2 = pos2 + vel2*t - 0.5*9.81*t*t;
    pos3 = pos3 + vel3*t;

    //Velocity at intersection
    vel2 = vel2 - 9.81*t;

    var ar = f(x1[0], x1[1], x1[2], x1[3], x1[4], x1[5], blank);

    IvWorld.pr1.value = ar[0];
    IvWorld.pr2.value = ar[1];
    IvWorld.pr3.value = ar[2];
    IvWorld.pr4.value = ar[3];
    IvWorld.pr5.value = ar[4];
    IvWorld.pr6.value = ar[5];

    var orient = [0,0,0,0,0,0,0,0,0];

    orient[pos(0,1,3)] = -1;
    orient[pos(1,1,3)] = 0;
```

```
        orient[pos(2,1,3)] = 0;

        orient[pos(0,2,3)] = -vel1/speed;
        orient[pos(1,2,3)] = -vel2/speed;
        orient[pos(2,2,3)] = -vel3/speed;

        orient[pos(0,0,3)] = orient[pos(1,1,3)]*orient[pos(2,2,3)] - orient[pos(2,1,3)]*
            orient[pos(1,2,3)];
        orient[pos(1,0,3)] = orient[pos(2,1,3)]*orient[pos(0,2,3)] - orient[pos(0,1,3)]*
            orient[pos(2,2,3)];
        orient[pos(2,0,3)] = orient[pos(0,1,3)]*orient[pos(1,2,3)] - orient[pos(1,1,3)]*
            orient[pos(0,2,3)];

    var theta = Math.acos(orient[pos(2,2,3)]);

    if (absV(Math.cos(theta)) >= 0){
        var phi = Math.atan2(orient[pos(2,0,3)],orient[pos(2,1,3)]);
        var gamma = Math.atan2(orient[pos(0,2,3)],orient[pos(1,2,3)]);
    }
    else {
        var phi = -3.14159/2;
        theta = 3.14159/2;
        var gamma = phi + Math.atan2(orient[pos(0,1,3)],orient[pos(0,2,3)]);
    }

        IvWorld.prf1.value = ab(theta);
        IvWorld.prf2.value = ab(phi);
        IvWorld.prf3.value = ab(gamma);
        IvWorld.prf4.value = pos1;
        IvWorld.prf5.value = pos2;
        IvWorld.prf6.value = pos3;

        IvWorld.cTime.value = t;
    }

    var desiredAngle8 = -0.35; //Calculate
    var desiredPosition9 = 0.027; //Calculate
    var desiredPosition10 = 0.027; //Calculate
    var desiredPosition11 = 0.027; //Calculate
    var desiredPosition12 = 0.027; //Calculate

    if (time >= (parseFloat(IvWorld.cTime.value)-0.02)){

        desiredAngle8 = -0.75; //Calculate
        desiredPosition9 = 0.01; //Calculate
        desiredPosition10 = 0.01; //Calculate
        desiredPosition11 = 0.01; //Calculate
        desiredPosition12 = 0.01; //Calculate

    }

    if (time>(parseFloat(IvWorld.cTime.value)+1.2)){
    var desiredAngle8 = -0.35; //Calculate
    var desiredPosition9 = 0.027; //Calculate
    var desiredPosition10 = 0.027; //Calculate
    var desiredPosition11 = 0.027; //Calculate
    var desiredPosition12 = 0.027; //Calculate
    }

        var posRotStart=[parseFloat(IvWorld.pr1.value), parseFloat(IvWorld.pr2.value),
            parseFloat(IvWorld.pr3.value), parseFloat(IvWorld.pr4.value), parseFloat(
            IvWorld.pr5.value), parseFloat(IvWorld.pr6.value)]; //PERSISTANT
        var posRotEnd = [parseFloat(IvWorld.prf1.value), parseFloat(IvWorld.prf2.value),
            parseFloat(IvWorld.prf3.value), parseFloat(IvWorld.prf4.value), parseFloat(
            IvWorld.prf5.value), parseFloat(IvWorld.prf6.value)];
        var posRotDrop = [parseFloat(IvWorld.pr1.value)+0.3, parseFloat(IvWorld.pr2.value
            )+.2, parseFloat(IvWorld.pr3.value)-2.5, 0.3, -0.3, -0.35];

        if (time < parseFloat(IvWorld.cTime.value)){
                for (var i = 0; i < 6; i++){
                        var q0 = posRotStart[i];
                        var qf = posRotEnd[i];
                        var tf = parseFloat(IvWorld.cTime.value);
                        posRot[i] = -2*(qf-q0)/(tf*tf*tf)*(time*time*time) + 3*(qf-q0)/(
                            tf*tf)*(time*time) + q0;

                }
        }
        else if (time<(parseFloat(IvWorld.cTime.value)+0.2)){
                for (var i = 0; i < 6; i++){
                    posRot[i] = posRotEnd[i];
                }
        }
        else if (time<(parseFloat(IvWorld.cTime.value)+1.2)){

                    for (var i = 0; i < 6; i++){
                        var q0 = posRotEnd[i];
                        var qf = posRotDrop[i];
                        var tf = (parseFloat(IvWorld.cTime.value)+1.2);
                        var t0 = (parseFloat(IvWorld.cTime.value)+0.2);

                        var a0 = (tf*tf*(3*t0 - tf))/((t0 - tf)*(t0 - tf)*(t0 - tf))*q0 +
                             (t0*t0*(t0 - 3*tf))/((t0 - tf)*(t0 - tf)*(t0 - tf))*qf;
                        var a1 = -(6*t0*tf)/((t0 - tf)*(t0 - tf)*(t0 - tf))*q0 + (6*t0*tf
                            )/((t0 - tf)*(t0 - tf)*(t0 - tf))*qf;
                        var a2 = (3*t0 + 3*tf)/((t0 - tf)*(t0 - tf)*(t0 - tf))*q0 + -(3*
                            t0 + 3*tf)/((t0 - tf)*(t0 - tf)*(t0 - tf))*qf;
                        var a3 = -2/((t0 - tf)*(t0 - tf)*(t0 - tf))*q0 + 2/((t0 - tf)*(t0
                             - tf)*(t0 - tf))*qf;

                        posRot[i] = a3*(time*time*time) + a2*(time*time) + a1*(time) + a0
                            ;
                }
        }
        else {
        for (var i = 0; i < 6; i++){
                    posRot[i] = posRotDrop[i];
                }

        }

    IvWorld.theta.value = posRot[0];
    IvWorld.phi.value = posRot[1];
    IvWorld.gamma.value = posRot[2];

    f1=f(x1[0], x1[1], x1[2], x1[3], x1[4], x1[5], posRot);

    IvWorld.e1.value = f1[0];
    IvWorld.e2.value = f1[1];
    IvWorld.e3.value = f1[2];
    IvWorld.e4.value = f1[3];
    IvWorld.e5.value = f1[4];
    IvWorld.e6.value = f1[5];

    if (absV(tcheck(f1))>tolerance){

        // Newton-Raphson Secant Method

        var x2 = [];

        count = 0;
        var J=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
        var Ji=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
        var dx=[0, 0, 0, 0, 0, 0];

        while (absV(tcheck(f1))>tolerance){

            count++;
            for (var i = 0; i < 6; i++){
            x2[i] = x1[i] + 0.12;
            }

            J = Jacobian(x1[0], x1[1], x1[2], x1[3], x1[4], x1[5], x2[0], x2[1], x2[2],
                x2[3], x2[4], x2[5], posRot);

            Ji=inverse(J);

            dx = mMultiply(Ji,f1,6,6,1);

            for (var i=0; i<6; i++) {
                x1[i]=x1[i]-dx[i];
            }

            f1=f(x1[0], x1[1], x1[2], x1[3], x1[4], x1[5], posRot);
        }
    }

    desiredAngle1 = ab(x1[0]); //Calculate
    desiredAngle2 = ab(x1[1]); //Calculate
    desiredAngle3 = ab(x1[2]); //Calculate
    desiredAngle4 = ab(x1[3]); //Calculate
    desiredAngle5 = ab(x1[4]); //Calculate
    desiredAngle6 = ab(x1[5]); //Calculate

    // Shoulder 1
    p1 = 1200; d1 = 5;
    curAngle1 = ab(parseFloat(IvWorld.DISrunactShoulder1.angle));
    curAngleV1 = parseFloat(IvWorld.DISrunactShoulder1.angularVelocity);
    desiredAngle01 = ab(parseFloat(IvWorld.Shoulder1_Desired_Angle.value));
    //desiredAngle1 = desiredAngle01; //Calculate
    desiredAngleV1 = ab(desiredAngle1-desiredAngle01)/dt;
    IvWorld.Shoulder1_Desired_Angle.value = desiredAngle1;
    IvWorld.DISrunactShoulder1.torque = -p1 * ab(curAngle1 - desiredAngle1) - d1*(
        curAngleV1 - desiredAngleV1);

    // Shoulder 2
    p2 = 1000; d2 = 3;
    curAngle2 = ab(parseFloat(IvWorld.DISrunactShoulder2.angle));
    curAngleV2 = parseFloat(IvWorld.DISrunactShoulder2.angularVelocity);
    desiredAngle02 = ab(parseFloat(IvWorld.Shoulder2_Desired_Angle.value));
    //desiredAngle2 = desiredAngle02; //Calculate
    desiredAngleV2 = ab(desiredAngle2-desiredAngle02)/dt;
    IvWorld.Shoulder2_Desired_Angle.value = desiredAngle2;
    IvWorld.DISrunactShoulder2.torque = -p2 * ab(curAngle2 - desiredAngle2) - d2*(
        curAngleV2 - desiredAngleV2);

    // Shoulder 3
    p3 = 1500; d3 = 5;
    curAngle3 = ab(parseFloat(IvWorld.DISrunactShoulder3.angle));
    curAngleV3 = parseFloat(IvWorld.DISrunactShoulder3.angularVelocity);
    desiredAngle03 = ab(parseFloat(IvWorld.Shoulder3_Desired_Angle.value));
    //desiredAngle3 = desiredAngle03; //Calculate
    desiredAngleV3 = ab(desiredAngle3-desiredAngle03)/dt;
    IvWorld.Shoulder3_Desired_Angle.value = desiredAngle3;
    IvWorld.DISrunactShoulder3.torque = -p3 * ab(curAngle3 - desiredAngle3) - d3*(
        curAngleV3 - desiredAngleV3);

    // Upper Arm
    p4 = 1000; d4 = 3;
    curAngle4 = ab(parseFloat(IvWorld.DISrunactUpperArm.angle));
    curAngleV4 = parseFloat(IvWorld.DISrunactUpperArm.angularVelocity);
    desiredAngle04 = ab(parseFloat(IvWorld.UpperArm_Desired_Angle.value));
    //desiredAngle4 = desiredAngle04; //Calculate
    desiredAngleV4 = ab(desiredAngle4-desiredAngle04)/dt;
    IvWorld.UpperArm_Desired_Angle.value = desiredAngle4;
    IvWorld.DISrunactUpperArm.torque = -p4 * ab(curAngle4 - desiredAngle4) - d4*(
        curAngleV4 - desiredAngleV4);

    // Elbow
    p5 = 600; d5 = 2;
    curAngle5 = ab(parseFloat(IvWorld.DISrunactElbow.angle));
    curAngleV5 = parseFloat(IvWorld.DISrunactElbow.angularVelocity);
    desiredAngle05 = ab(parseFloat(IvWorld.Elbow_Desired_Angle.value));
    //desiredAngle5 = desiredAngle05; //Calculate
    desiredAngleV5 = ab(desiredAngle5-desiredAngle05)/dt;
    IvWorld.Elbow_Desired_Angle.value = desiredAngle5;
    IvWorld.DISrunactElbow.torque = -p5 * ab(curAngle5 - desiredAngle5) - d5*(curAngleV5
        - desiredAngleV5);

    // Forearm
    p6 = 500; d6 = 2;
    curAngle6 = ab(parseFloat(IvWorld.DISrunactForearm.angle));
    curAngleV6 = parseFloat(IvWorld.DISrunactForearm.angularVelocity);
    desiredAngle06 = ab(parseFloat(IvWorld.Forearm_Desired_Angle.value));
    //desiredAngle6 = desiredAngle06; //Calculate
    desiredAngleV6 = ab(desiredAngle6-desiredAngle06)/dt;
    IvWorld.Forearm_Desired_Angle.value = desiredAngle6;
    IvWorld.DISrunactForearm.torque = -p6 * ab(curAngle6 - desiredAngle6) - d6*(
        curAngleV6 - desiredAngleV6);

    // Thumb
    p8 = 5; d8 = 0.03;
    curAngle8 = ab(parseFloat(IvWorld.DISrunactThumb.angle));
    curAngleV8 = parseFloat(IvWorld.DISrunactThumb.angularVelocity);
    desiredAngle08 = ab(parseFloat(IvWorld.Thumb_Desired_Angle.value));
    desiredAngleV8 = ab(desiredAngle8-desiredAngle08)/dt;
    IvWorld.Thumb_Desired_Angle.value = desiredAngle8;
    IvWorld.DISrunactThumb.torque = (-p8* ab(curAngle8 - desiredAngle8) - d8*(curAngleV8
        - desiredAngleV8));

    // IndexFinger
    p9 = 1600; d9 = 50;
    curPosition9 = parseFloat(IvWorld.DISrunactIndexFinger.displacement);
    curPositionV9 = parseFloat(IvWorld.DISrunactIndexFinger.speed);
    desiredPosition09 = parseFloat(IvWorld.IndexFinger_Desired_Position.value);
    desiredPositionV9 = ab(desiredPosition9-desiredPosition09)/dt;
    IvWorld.IndexFinger_Desired_Position.value = desiredPosition9;
    IvWorld.DISrunactIndexFinger.force = (p9* (curPosition9 - desiredPosition9) + d9*(
        curPositionV9 - desiredPositionV9));

    // MiddleFinger
    p10 = 1600; d10 = 50;
    curPosition10 = parseFloat(IvWorld.DISrunactMiddleFinger.displacement);
    curPositionV10 = parseFloat(IvWorld.DISrunactMiddleFinger.speed);
    desiredPosition010 = parseFloat(IvWorld.MiddleFinger_Desired_Position.value);
    desiredPositionV10 = ab(desiredPosition10-desiredPosition010)/dt;
    IvWorld.MiddleFinger_Desired_Position.value = desiredPosition10;
    IvWorld.DISrunactMiddleFinger.force = (p10* (curPosition10 - desiredPosition10) + d10
        *(curPositionV10 - desiredPositionV10));
```

```
// RingFinger
p11 = 1600; d11 = 50;
curPosition11 = parseFloat(IvWorld.DISrunactRingFinger.displacement);
curPositionV11 = parseFloat(IvWorld.DISrunactRingFinger.speed);
desiredPosition011 = parseFloat(IvWorld.RingFinger_Desired_Position.value);
desiredPositionV11 = ab(desiredPosition11-desiredPosition011)/dt;
IvWorld.RingFinger_Desired_Position.value = desiredPosition11;
IvWorld.DISrunactRingFinger.force = (p11* (curPosition11 - desiredPosition11) + d11*(
    curPositionV11 - desiredPositionV11));

// SmallFinger
p12 = 1600; d12 = 50;
curPosition12 = parseFloat(IvWorld.DISrunactSmallFinger.displacement);
curPositionV12 = parseFloat(IvWorld.DISrunactSmallFinger.speed);
desiredPosition012 = parseFloat(IvWorld.SmallFinger_Desired_Position.value);
desiredPositionV12 = ab(desiredPosition12-desiredPosition012)/dt;
IvWorld.SmallFinger_Desired_Position.value = desiredPosition12;
IvWorld.DISrunactSmallFinger.force = (p12* (curPosition12 - desiredPosition12) + d12
    *(curPositionV12 - desiredPositionV12));

// Functions

// Allows interpretation of 1D arrays as 2D arrays
function pos(r, c, d)
{
    return (r*d+c);
}

function absV(num)
{
    if (num<0){
        num = num*-1;
    }
    return num;
}

function mMultiply(A, B, m, n, p){

    var C = [];
    for (var i = 0; i<m; i++){
        for (var k = 0; k<p; k++){
            var sum1 = 0;
            for (var j = 0; j<n; j++){
                sum1 = sum1 + A[pos(i,j,n)]*B[pos(j,k,p)];
            }
            C[pos(i,k,p)]=sum1;
        }
    }
    return C;
}

// Gauss elimination with Partial Pivoting
function inverse(J){
    var C=[0,0,0,0,0,0];
    var X=[0,0,0,0,0,0];
    var n = 6;
    var Jcopy=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
    var Jinv=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

    for (var r = 0; r<n*n; r++) {
        Jcopy[r]=J[r];
    }

    for (var i = 0; i<n; i++) {
        for (var l = 0; l<n*n; l++) {
            J[l]=Jcopy[l];
        }
        for (var h = 0; h<n; h++) {
            C[h]=0;
        }
        C[i]=1.0;

        var w=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
        var Tolerance = 0.00001;
        var MaxCoefficient=0;
        var MaxI=0;
        var temp=0;
        for(var k=0;k<n;k++){
            if(absV(J[pos(k,k,n)])<Tolerance){
                //Loop to find the maximum coefficient
                MaxCoefficient=absV(J[pos(k,k,n)]);
                MaxI=k;
                for(var p=k+1;p<n;p++){
                    if(absV(J[pos(p,k,n)])>MaxCoefficient){
                        MaxCoefficient=absV(J[pos(p,k,n)]);
                        MaxI=p;
                    }
                }

                if(MaxCoefficient===0){
                    return 0;
                }

                temp=C[k];
                C[k]=C[MaxI];
                C[MaxI]=temp;

                for(var j=k;j<n;j++){
                    temp=J[pos(k,j,n)];
                    J[pos(k,j,n)]=J[pos(MaxI,j,n)];
                    J[pos(MaxI,j,n)]=temp;
                }
            }
            for(var g=k+1;g<n;g++){
                w[pos(g,k,n)]=J[pos(g,k,n)]/J[pos(k,k,n)];
                C[g]= C[g]-w[pos(g,k,n)]*C[k];

                for(var jj=k;jj<n;jj++){
                J[pos(g,jj,n)]=J[pos(g,jj,n)]-w[pos(g,k,n)]*J[pos(k,jj,n)];
                }
            }
        }

        //Back substitution:
        for (var kk = n-1; kk>=0; kk--){
            X[kk] = 0;
            for (var c = kk; c<n; c++){
                                X[kk]+=J[pos(kk,c,n)]*X[c];
            }
            X[kk] = (C[kk]-X[kk])/J[pos(kk,kk,n)];
        }

        //Load into big inverse matrix.
        for (var v = 0; v<n; v++){
            Jinv[pos(v,i,n)]=X[v];
        }
    }
    return Jinv;
}
```

```
function f(th1, th2, th3, th4, th5, th6, posRot){
        //Matrix operations:
        T1=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T2=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T25=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T3=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T4=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T45=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T5=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T55=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T6=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        T7=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];

    //Parameter creation
    a1=0;
    alpha1=90*3.14159265/180;
    d1=0;
    t1=90*3.14159265/180; // t0

    a2=0;
    alpha22=90*3.14159265/180;
    d2 =0.12;
    //t2=0; //THETA 1
        t2=th1;

    a25=0.09;
    alpha25=0*3.14159265/180; //Linked to next joint
    d25 = 0.0075;
    //t25=180*3.14159265/180;  //THETA 2
        t25 = th2;

    a3=0.04743165;
    alpha3=-90*3.14159265/180; //ALPHA FOR JOINT 3
    d3 = 0;
    t3=-90*3.14159265/180;//STATIC

    a4=0;
    alpha4=-90*3.14159265/180; //ALPHA FOR JOINT 4 V
    d4 = 0.231;
    //t4=-90*3.14159265/180; //THETA 3 V
        t4=th3;

    a45=0.012;
    alpha45=0*3.14159265/180;
    d45 = 0;
    //t45=-90*3.14159265/180; //THETA 4 V
        t45 = th4;

    a5=0.061;
    alpha5=0*3.14159265/180;
    d5 = 0;
    t5=90*3.14159265/180;  //STATIC

    a55=0;
    alpha55=-90*3.14159265/180; //ALPHA FOR JOINT 5
    d55 = 0;
    t55=-90*3.14159265/180; // STATIC

    a6=0;
    alpha6=90*3.14159265/180;
    d6 = 0.19;
    //t6=0*3.14159265/180; //THETA 5
        t6=th5;

    a7=0.095;//0.0501397;
```

```
alpha7=-90*3.14159265/180;
d7 = 0;
//t7=90.*3.14159265/180; //THETA 6
    t7=th6;

//Matrix creation
M1=[Math.cos(t1),-Math.sin(t1)*Math.cos(alpha1),Math.sin(t1)*Math.sin(alpha1),a1*
    Math.cos(t1),Math.sin(t1),Math.cos(t1)*Math.cos(alpha1),-Math.cos(t1)*Math.
    sin(alpha1),a1*Math.sin(t1),0,Math.sin(alpha1),Math.cos(alpha1),d1,0,0,0,1];
M25=[Math.cos(t25),-Math.sin(t25)*Math.cos(alpha25),Math.sin(t25)*Math.sin(
    alpha25),a25*Math.cos(t25),Math.sin(t25),Math.cos(t25)*Math.cos(alpha25),-
    Math.cos(t25)*Math.sin(alpha25),a25*Math.sin(t25),0,Math.sin(alpha25),Math.
    cos(alpha25),d25,0,0,0,1];
M2=[Math.cos(t2),-Math.sin(t2)*Math.cos(alpha22),Math.sin(t2)*Math.sin(alpha22),
    a2*Math.cos(t2),Math.sin(t2),Math.cos(t2)*Math.cos(alpha22),-Math.cos(t2)*
    Math.sin(alpha22),a2*Math.sin(t2),0,Math.sin(alpha22),Math.cos(alpha22),d2
    ,0,0,0,1];
M3=[Math.cos(t3),-Math.sin(t3)*Math.cos(alpha3),Math.sin(t3)*Math.sin(alpha3),a3*
    Math.cos(t3),Math.sin(t3),Math.cos(t3)*Math.cos(alpha3),-Math.cos(t3)*Math.
    sin(alpha3),a3*Math.sin(t3),0,Math.sin(alpha3),Math.cos(alpha3),d3,0,0,0,1];
M4=[Math.cos(t4),-Math.sin(t4)*Math.cos(alpha4),Math.sin(t4)*Math.sin(alpha4),a4*
    Math.cos(t4),Math.sin(t4),Math.cos(t4)*Math.cos(alpha4),-Math.cos(t4)*Math.
    sin(alpha4),a4*Math.sin(t4),0,Math.sin(alpha4),Math.cos(alpha4),d4,0,0,0,1];
M45=[Math.cos(t45),-Math.sin(t45)*Math.cos(alpha45),Math.sin(t45)*Math.sin(
    alpha45),a45*Math.cos(t45),Math.sin(t45),Math.cos(t45)*Math.cos(alpha45),Math
    .cos(t45)*Math.sin(alpha45),a45*Math.sin(t45),0,Math.sin(alpha45),Math.cos(
    alpha45),d45,0,0,0,1];
M5=[Math.cos(t5),-Math.sin(t5)*Math.cos(alpha5),Math.sin(t5)*Math.sin(alpha5),a5*
    Math.cos(t5),Math.sin(t5),Math.cos(t5)*Math.cos(alpha5),-Math.cos(t5)*Math.
    sin(alpha5),a5*Math.sin(t5),0,Math.sin(alpha5),Math.cos(alpha5),d5,0,0,0,1];
M55=[Math.cos(t55),-Math.sin(t55)*Math.cos(alpha55),Math.sin(t55)*Math.sin(
    alpha55),a55*Math.cos(t55),Math.sin(t55),Math.cos(t55)*Math.cos(alpha55),-
    Math.cos(t55)*Math.sin(alpha55),a55*Math.sin(t55),0,Math.sin(alpha55),Math.
    cos(alpha55),d55,0,0,0,1];
M6=[Math.cos(t6),-Math.sin(t6)*Math.cos(alpha6),Math.sin(t6)*Math.sin(alpha6),a6*
    Math.cos(t6),Math.sin(t6),Math.cos(t6)*Math.cos(alpha6),-Math.cos(t6)*Math.
    sin(alpha6),a6*Math.sin(t6),0,Math.sin(alpha6),Math.cos(alpha6),d6,0,0,0,1];
M7=[Math.cos(t7),-Math.sin(t7)*Math.cos(alpha7),Math.sin(t7)*Math.sin(alpha7),a7*
    Math.cos(t7),Math.sin(t7),Math.cos(t7)*Math.cos(alpha7),-Math.cos(t7)*Math.
    sin(alpha7),a7*Math.sin(t7),0,Math.sin(alpha7),Math.cos(alpha7),d7,0,0,0,1];

//Multiply matrices
for (var i = 0; i<16; i++) {
    T1[i]=M1[i];
}

T2 = mMultiply(M1,M2,4,4,4);
T25 = mMultiply(T2,M25,4,4,4);
T3 = mMultiply(T25,M3,4,4,4);
T4 = mMultiply(T3,M4,4,4,4);
T45 = mMultiply(T4,M45,4,4,4);
T5 = mMultiply(T45,M5,4,4,4);
T55 = mMultiply(T5,M55,4,4,4);
T6 = mMultiply(T55,M6,4,4,4);
T7 = mMultiply(T6,M7,4,4,4);

var theta = Math.acos(T7[pos(2,2,4)]);

if (absV(Math.cos(theta)) >= 0){
    var phi = Math.atan2(T7[pos(2,0,4)],T7[pos(2,1,4)]);
    var gamma = Math.atan2(T7[pos(0,2,4)],T7[pos(1,2,4)]);
}
else {
    var phi = -3.14159/2;
    theta = 3.14159/2;
    var gamma = phi + Math.atan2(T7[pos(0,1,4)],T7[pos(0,2,4)]);
    }

        var retArVal = [(theta-posRot[0]),(phi-posRot[1]),(gamma-posRot[2]),(T7[pos
            (0,3,4)]-posRot[3]),(T7[pos(1,3,4)]-posRot[4]),(T7[pos(2,3,4)]-posRot[5])
            ];

        return retArVal;
}

function Jacobian(t1,t2,t3,t4,t5,t6,t12,t22,t32,t42,t52,t62,posRot)
{
        var t=[t1,t2,t3,t4,t5,t6];
        var tt2=[t12,t22,t32,t42,t52,t62];
        var Jac=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];
        var reg=[0,0,0,0,0,0];
        var f12=[0,0,0,0,0,0];
        var f22=[0,0,0,0,0,0];
        var f32=[0,0,0,0,0,0];
        var f42=[0,0,0,0,0,0];
        var f52=[0,0,0,0,0,0];
        var f62=[0,0,0,0,0,0];
        var reg2=[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
            0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0];

        reg = f(t1, t2, t3, t4, t5, t6, posRot);
        f12 = f(t12,t2,t3,t4,t5,t6,posRot);
        f22 = f(t1,t22,t3,t4,t5,t6,posRot);
        f32 = f(t1,t2,t32,t4,t5,t6,posRot);
        f42 = f(t1,t2,t3,t42,t5,t6,posRot);
        f52 = f(t1,t2,t3,t4,t52,t6,posRot);
        f62 = f(t1,t2,t3,t4,t5,t62,posRot);

        for (var i = 0; i<6; i++){
                reg2[pos(0,i,6)]=f12[i];
                reg2[pos(1,i,6)]=f22[i];
                reg2[pos(2,i,6)]=f32[i];
                reg2[pos(3,i,6)]=f42[i];
                reg2[pos(4,i,6)]=f52[i];
                reg2[pos(5,i,6)]=f62[i];
        }

        for (var w=0; w<6; w++){
                for (var j=0; j<6; j++){
                        Jac[pos(w,j,6)] = (reg2[pos(j,w,6)]-reg[w])/(tt2[j]-t[j]);

                }
        }
        return Jac;
}

function tcheck(f1){
    var temp = 0;

    temp=f1[0];

    for (var t=1; t<6; t++){
        if (f1[t]>temp){
            temp=f1[t];
        }
     }
     return temp;
}

function ab(par){
    var temporary=par;
    while (temporary>2*3.14159265358979323846264338327950){
        temporary = temporary - 2*3.14159265358979323846264338327950;
    }

    while (temporary<(-2*3.14159265358979323846264338327950)){
        temporary = temporary + 2*3.14159265358979323846264338327950;
    }

    if (absV(temporary)>3.14159265358979323846264338327950){
        if(temporary<0){
            temporary=temporary+2*3.14159265358979323846264338327950;
        }
        else{
            temporary=temporary-2*3.14159265358979323846264338327950;
        }
    }
    return temporary;
}
```

B. CATCH TEST RESULTS

The catch pictures from the simulations for each test are presented here.

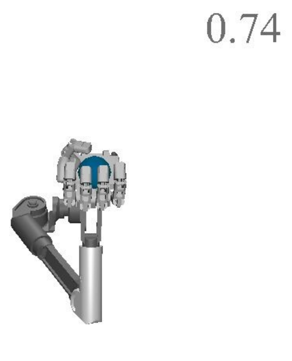


Figure B.1. Catch at (0, 0)

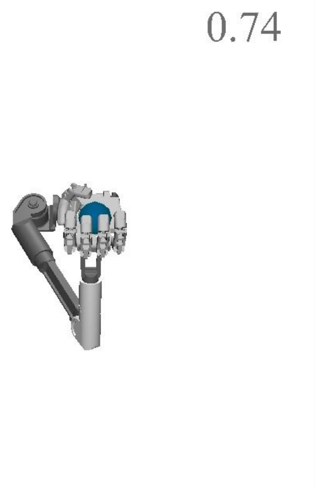


Figure B.2. Catch at (0, 0.07)

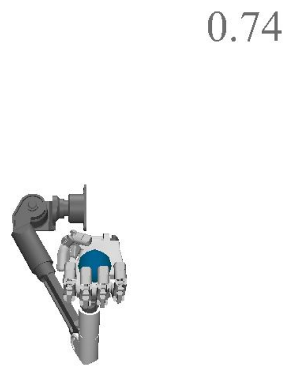


Figure B.3. Catch at (0, 0.14)

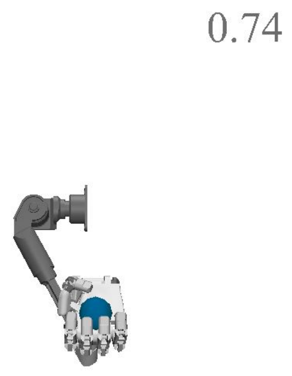


Figure B.4. Catch at (0, 0.21)

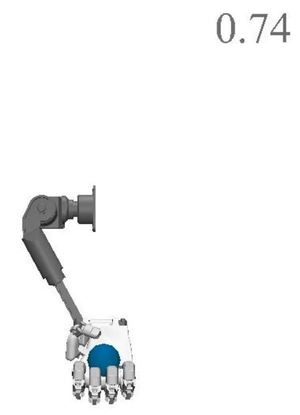


Figure B.5. Catch at (0, 0.28)

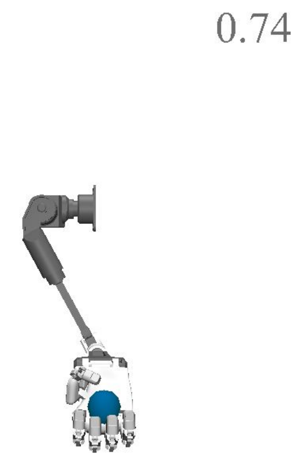


Figure B.6. Catch at (0, 0.35)

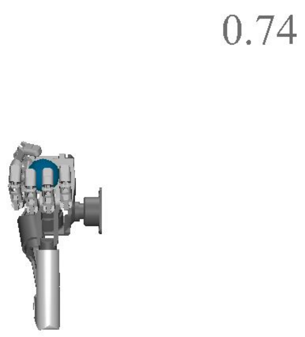


Figure B.7. Catch at (0.1, 0)

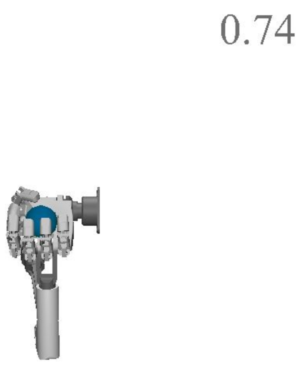


Figure B.8. Catch at (0.1, 0.07)

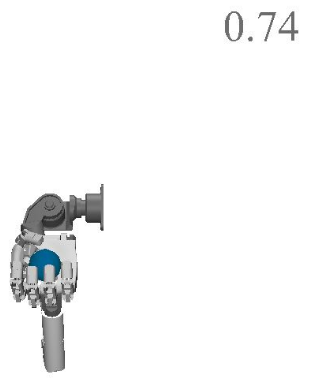


Figure B.9. Catch at (0.1, 0.14)

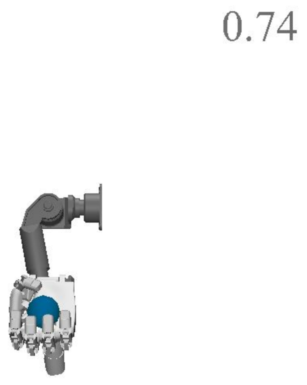


Figure B.10. Catch at (0.1, 0.21)

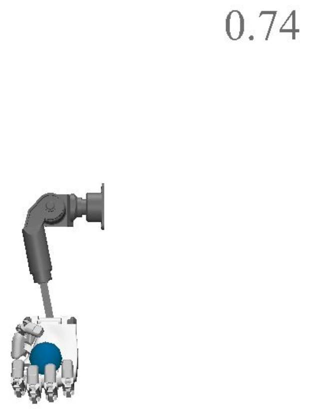


Figure B.11. Catch at (0.1, 0.28)

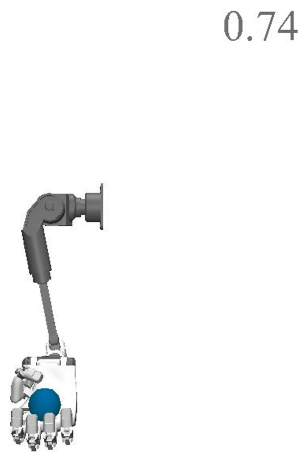


Figure B.12. Catch at (0.1, 0.35)

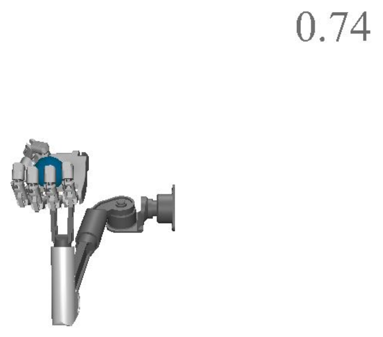


Figure B.13. Catch at (0.2, 0)

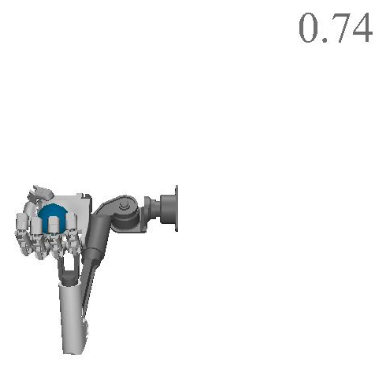


Figure B.14. Catch at (0.2, 0.07)

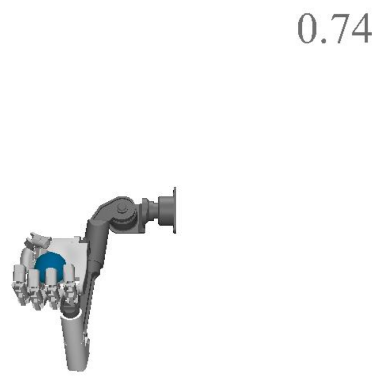


Figure B.15. Catch at (0.2, 0.14)

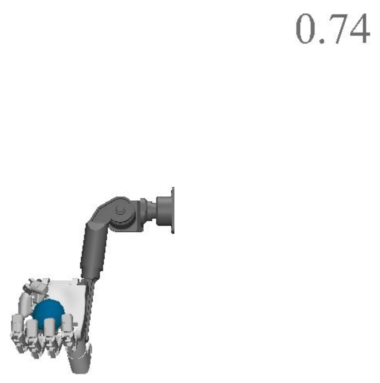


Figure B.16. Catch at (0.2, 0.21)

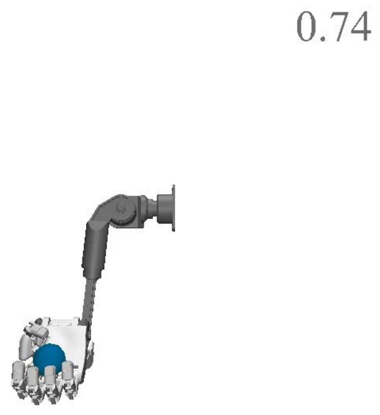


Figure B.17. Catch at (0.2, 0.28)

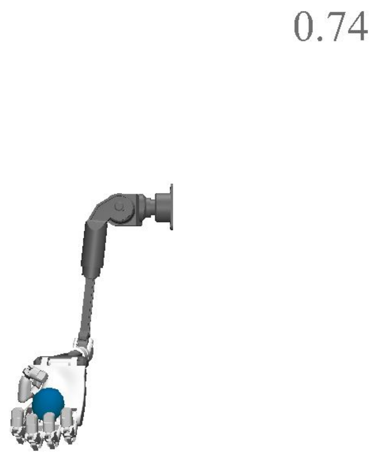


Figure B.18. Catch at (0.2, 0.35)

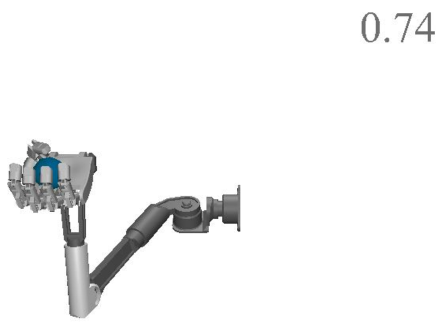


Figure B.19. Catch at (0.3, 0)

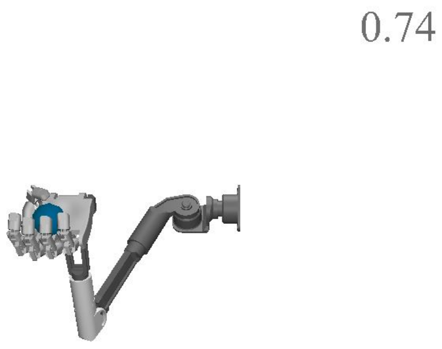


Figure B.20. Catch at (0.3, 0.07)

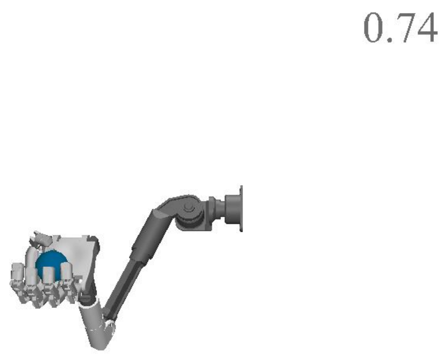


Figure B.21. Catch at (0.3, 0.14)

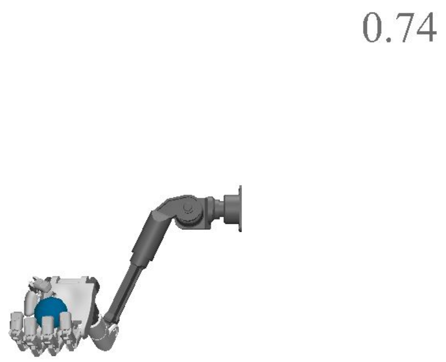


Figure B.22. Catch at (0.3, 0.21)

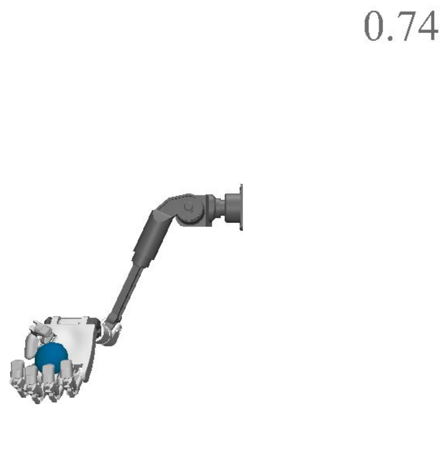


Figure B.23. Catch at (0.3, 0.28)

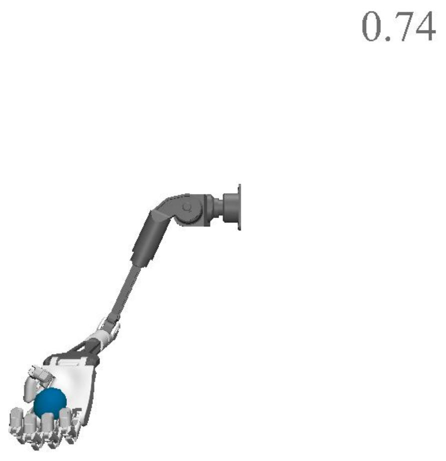


Figure B.24. Catch at (0.3, 0.35)

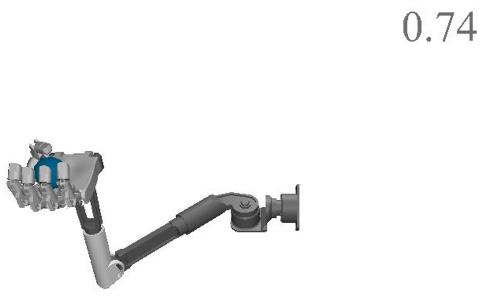


Figure B.25. Catch at (0.4, 0)

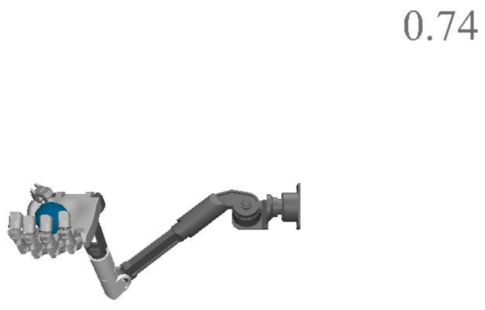


Figure B.26. Catch at (0.4, 0.07)

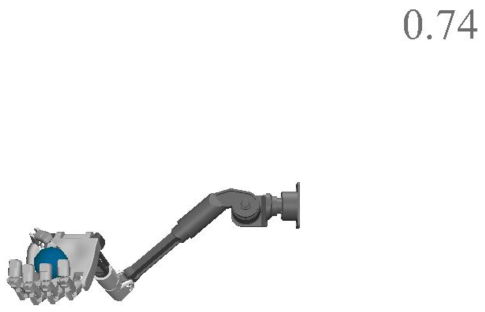


Figure B.27. Catch at (0.4, 0.14)

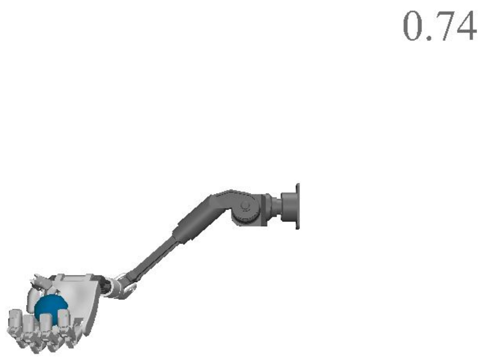


Figure B.28. Catch at (0.4, 0.21)

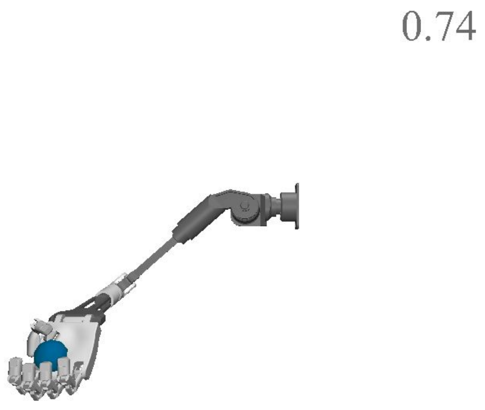


Figure B.29. Catch at (0.4, 0.28)

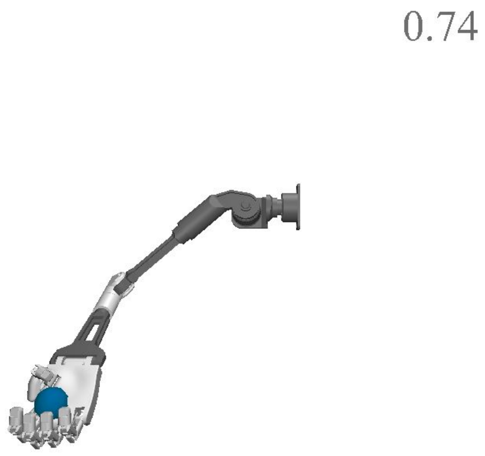


Figure B.30. Catch at (0.4, 0.35)