1-1-2004

Three-dimensional modeling and simulation of manufacturing processes for transmuter fuel fabrication

Richard Arthur Silva
University of Nevada, Las Vegas

THREE DIMENSIONAL MODELING AND SIMULATION OF MANUFACTURING PROCESSES FOR TRANSMUTER FUEL FABRICATION

by

Richard Arthur Silva

Bachelor of Science in Mechanical Engineering
West Virginia University
1994

A thesis submitted in partial fulfillment
of the requirements for the

Master of Science in Engineering
Department of Mechanical Engineering
Howard R. Hughes College of Engineering

Graduate College
University of Nevada, Las Vegas
May 2004

UNLV

Thesis Approval
The Graduate College
University of Nevada, Las Vegas

April 2, 20 04

The Thesis prepared by

Richard Arthur Silva

Entitled

Three Dimensional Modeling and Simulation
of Manufacturing Processes for Transmuter Fuel Fabrication

is approved in partial fulfillment of the requirements for the degree of

Master of Science in Engineering

Examination Committee Chair

Dean of the Graduate College

Examination Committee Member

Examination Committee Member

Graduate College Faculty Representative

1017-53

ABSTRACT

Three Dimensional Modeling and Simulation of Manufacturing Processes for Transmuter Fuel Fabrication

by

Richard Arthur Silva

Dr. Georg F. Mauer, Examination Committee Chair
Professor of Mechanical Engineering
University of Nevada, Las Vegas

A proposed method of converting long-lived radionuclides contained within spent nuclear fuel into shorter-lived radionuclides is known as transmutation. The feasibility assessment of fuel manufacturing for transmutation is necessary for the overall success of the transmutation project. As part of that assessment, this paper reviews current methodologies for handling spent nuclear fuel and other highly radioactive material. While not exhaustive, the review encompassed facilities within the U.S. Department of Energy, international operations, as well as, work performed collegiately.

The manufacturing of fuel for transmutation will certainly utilize automation and robotics. A simplified 3-D model of a fuel handling operation was produced. Forward and inverse kinematic routines were implemented for a commercially available robotic manipulator, as well as path planning routines. Multiple software packages were integrated and used to create 3-D robotic manipulator models and simulate the model.

A timing study was performed which validated the need for process optimization throughout the design process.

TABLE OF CONTENTS

LIST OF FIGURES

ACRONYMS AND ABBREVIATIONS

Acronyms

3-D	Three Dimensional
AECL	Atomic Energy of Canada Limited
ALARA	As Low as is Reasonably Achievable
ATW	Accelerator Transmutation of Waste
CAD	Computer Aided Design
CNSF	Commercial Spent Nuclear Fuel
COGEMA	Compagnie General des Matieres Nucleaires
COM	Component Object Model
DEC	Spent Fuel Disassembly and Consolidation System
DOE	U.S. Department of Energy
FMEF	Fuels and Materials Examination Facility
IGRIP	interactive graphical robot instruction program
MATLAB	MATrix LABoratory (software)
MOX	Mixed Oxide (fuel)
MPC	Multipurpose Canister
MSM	Master Slave Manipulator
OLE	Object Linking and Embedding
PUREX	Plutonium Extraction
PWR	Pressurized Water Reactor
SAF	Secure Automated Fabrication
SSC	Systems, structures, and components
UREX	Uranium Extraction
vN4D	MSC.visualNastran 4D (software)
WALS	Weight and Leak Check System

Abbreviations

Am	Americium
Ba	Barium
C	Celsius
Cm	Curium
Cs	Cesium
He	Helium
hr	Hour
I	Iodine
min	Minutes
MT	Metric Ton
Na	Sodium
Pu	Plutonium
rem	Rotogen equivalent man
sec	Seconds
Tc	Technetium
Te	Tellurium
UF_6	Uranium Hexafloride
UO_2	Uranium Dioxide
Xe	Xenon

ACKNOWLEDGMENTS

First and foremost, I would like to thank my wife, Susan Silva, for her loving support, encouragement, and enduring patience throughout my many years at the University of Nevada, Las Vegas. I dedicate this thesis to her.

I have been very fortunate to be part of the Transmutation Research Program. As the director, Dr. T. Hechenova has been an inspiration and his commitment to the program is a substantial asset for the University.

I would like to thank Dr. Mauer for not only guiding me in the development of this work, but also for treating me as a colleague and friend. Jamil Renno is an individual with drive, ambition and a perspicacious intellect, and as such, has provided me with guidance and drive to complete this work.

I would like to thank my employer Bechtel SAIC Co., LLC for providing both financial assistance and the flexibility so that I could complete such an endeavor.

Finally, I would like to thank my parents who have been supportive and understanding in my efforts to reach this goal.

CHAPTER 1

INTRODUCTION

The elimination of certain radionuclides from commercial spent nuclear fuel (CSNF) intended for future disposal at the mined geologic repository for high-level nuclear waste, Yucca Mountain, would have significant advantages toward the long-term performance of the Yucca Mountain site. CNSF is intended to be stored in corrosive resistant containers known as waste packages. The outer surface, or shell, of the waste package will be manufactured out of a nickel-based alloy (Alloy-22) that is expected to resist corrosion for more than ten thousand years; thereby, isolating the waste for an equal period of time. The arid conditions of Yucca Mountain, Nevada also provide natural barriers for storing the waste within the host rock of Yucca Mountain. The relatively dry conditions of the proposed repository provide only trace amounts of water to come in contact with the waste packages, corrode the outer layer of the waste package, and transport radionuclides to the ground water. Although significant scientific effort has been underway to ensure that radionuclides are not capable of reaching the ground water for 10,000 years, there are some inherent uncertainties with predicting the performance of any system, structure, or component for this extended timeframe.

Approximately one percent of the CNSF contains long-lived fission products. These long-lived fission products such as Technetium (Tc), related halogens such as Iodine (I), and related actinides such as Plutonium (Pu), Americium (Am), and Curium (Cm) are all

byproducts of the nuclear fission process [1, p. 57]. These radionuclides remain radioactive for an extended period of time. For example, the isotope ^{239}Pu has a half-life of approximately 20,000 years. It is these long-lived fusion products that effect the uncertainty in predicting Yucca Mountain's performance over the extended periods.

Therefore, a need exists to convert these long-lived radionuclides contained within the CNSF into a more benign form. This can be accomplished by converting or reducing the actinide, halogen, and other fission product half-lives. Theoretically, a conversion process has the potential to reduce the waste isolation requirements of Yucca Mountain from approximately 10,000 years to a few thousand years, in general 1,000 – 3,000 years.

1.1 Purpose of the Study

A proposed method of converting these long-lived radionuclides into shorter-lived radionuclides is known as transmutation. These hazardous elements can be separated and thereby transmuted in power-producing reactors as well as accelerator-driven systems. Transmutation cannot replace the current need for a national repository, such as Yucca Mountain, but a successful transmutation program will significantly reduce the long-term requirements for nuclear waste disposal. As a result, transmutation could remove the waste management issue as a barrier to expanded use of nuclear power by addressing the long-term environmental and economic issues faced by the United States, and the world [2, p. 2].

1.1.1 Overview of High Level Wastes

The following sub-sections provide a basic background of the sources of high-level waste and how the storage of such waste is required. Spent fuel processing is introduced

as a means of reducing the overall volume of required high-level waste storage by extracting uranium for low-level disposal.

The resulting material will then be discussed in terms of its longevity, half-life, and long-term storage requirements on a geologic repository.

1.1.2 Overview of Transmutation

Often debated in the scientific and public communities is the method for dealing with spent nuclear fuel, i.e., the back-end of the nuclear fuel cycle. One topic is the partitioning and transmutation of the spent fuel. Partitioning and transmutation involves processing the spent nuclear fuel to extract the long-lived radionuclides, which are then irradiated in either a nuclear reactor or in an accelerator driven system to produce products with shorter half-lives, thereby reducing the time required for their isolation from the environment [3, p. 18].

For example, the fission-product decay for the neutron absorbing isotope Xenon (X^{135}) is shown as the tellurium (Te^{135}) decay chain shown below [4]:

$$\text{Fission} \Rightarrow {}_{52}Te^{135} \Rightarrow {}_{53}I^{135} \Rightarrow {}_{54}Xe^{135} \Rightarrow {}_{55}Cs^{135} \Rightarrow {}_{56}Ba^{135}$$

Where Table 1 shows the related half-life for the tellurium (Te^{135}) decay chain:

Table 1. Tellurium Decay Chain

Element	Nuclide	Half-Life
Tellurium	${}_{52}Te^{135}$	< 2 min
Iodine	${}_{53}I^{135}$	6.68 hr
Xenon	${}_{54}Xe^{135}$	9.13 hr
Cesium	${}_{55}Cs^{135}$	2.3×10^{6} year
Barium	${}_{56}Ba^{135}$	Stable

Cesium is contained within spent fuel, and with a half-life of 2.3×10^{6} years, will remain radiotoxic for extended periods of time. It would be advantageous to transform the cesium ($_{55}Cs^{135}$) nuclide into the stable barium ($_{56}Ba^{135}$) nuclide.

A proposed method to transform the cesium nuclide is through transmutation by neutron bombardment. The current philosophy for the source of the neutron flux is from a non-critical, accelerator-driven reactor. When a cesium ($_{55}Cs^{135}$) nuclide is bombarded by a neutron (n), it transmutes into cesium ($_{55}Cs^{136}$) while emmitting a gamma ray (γ). The newly created cesium nuclide, with a half-life of approximately 13 days decays naturally to stable barium ($_{56}Ba^{136}$), while emitting an easily shielded beta particle (β). This process is generally shown in Figure 1.

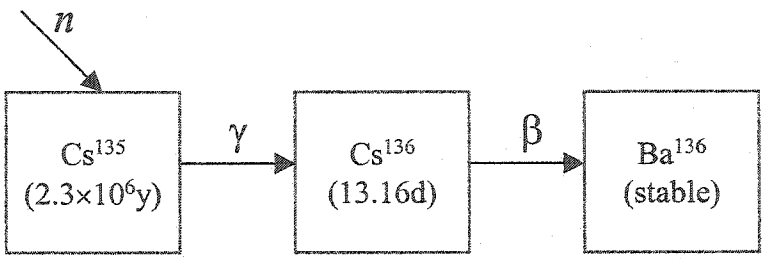


Figure 1. Neutron Absorption and Natural Decay of Cesium

While not all nuclides behave like cesium ($_{55}Cs^{135}$) and transmute to a nuclide that naturally decays to a stable state, transmutation is currently being investigated as a means of reducing the long-lived actinides to more benign actinides. Transmutation has the potential for decreasing the performance uncertainty of a deep geologic repository to contain these radionuclides over extended periods of time [3, p. 107 - 108].

1.1.3 Overview of the Fuel and Conversion to Transmuter Fuel

Various fuel types are being considered for use in an accelerator driven reactor, where the fuel contains the long-lived actinides for transmutation. Because the fuel will contain these actinides, dose rates will be significantly higher than normal fuel manufacturing. Thus, the fuel manufacturing process must take place in a shielded hot cell environment where the standard alpha/beta shielded glove box technology used in standard fuel manufacturing will not suffice. The scale of the required fabrication line is very large, and overall throughput rates are high for a remote operation. Equipment will be required to perform over a time frame of forty to sixty years, and the processes must allow the use of robust equipment that is readily maintainable. Attention must also be given to decommissioning during the design phase.

Current fuel fabrication processes for accelerator transmutation of waste (ATW) are either based on metal casting (metallic fuels) or power processing, leading to ceramic or dispersion fuels. Figure 2 depicts the generalized process flow for a ceramic fuel; in this particular case, a mixed oxide (MOX) fuel manufacturing process.

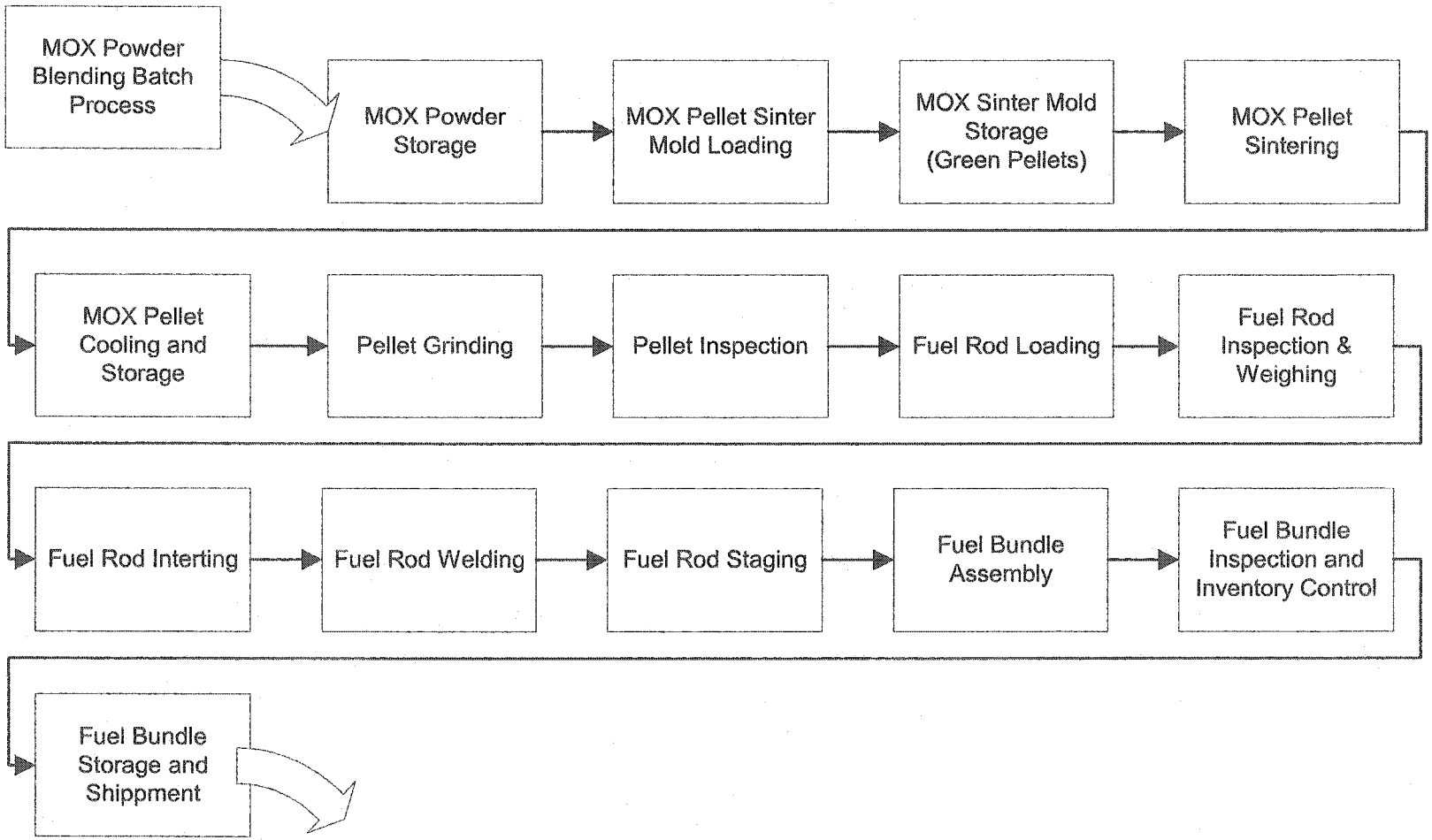


Figure 2. Simplified MOX Fuel Manufacturing Process Flow

Research and development on fuel reprocessing and manufacturing has been ongoing for years in the U.S. and other countries. With regard to fuel manufacturing, we may distinguish among three categories of fuel [5]:

- Dispersion Fuels – several subtypes exist
- Ceramic Fuels – several subtypes exist
- Metallic Fuels

For each ATW fuel type, a separate manufacturing sequence exists, as is summarized below.

Manufacturing Sequence for Dispersion Fuel [5]:

1. Manufacture spherical fuel particles by wet chemical process or direct reaction and attrition.
2. Coat the particles – process not yet determined
3. Embed fuel particles in matrix metal
4. Press fuel and matrix blend
5. Assemble billet
6. Extrude billet at approximately 800 degrees Celsius into rods about 2-meters long.
7. Finish fuel rods by trimming ends and performing rod inspections, which may include radiography, dimensional tolerance checks, bonding, and clad defects.

The Manufacturing Sequence for Metallic Fuel [5]:

1. Cast fuel slugs, where the slug is approximately 4 to 5 millimeters in diameter and are from 0.8 meter to 1.5 meters in length.
2. Insert fuel slugs into cladding tube
3. Add bonding agent to the fuel tube, which may be sodium (Na).
4. Seal cladding tube by welding end fitting into the tube.
5. Perform fuel slug inspections, which may include radiography, dimensional tolerance checks, bonding, and clad defects

The Manufacturing Sequence for Ceramic Fuel [5]:

1. Manufacture particles by wet chemical process or direct reaction, where the particles are approximately 1 μm to 30 μm in diameter.
2. Compact particles into pellet form.
3. Sinter the pellets at 1400-1800 °C
4. Inspect pellets
5. Assemble pellets into cladding tube
6. Add bonding material, which may be Helium (He) or Sodium (Na)

7. Seal cladding tube by welding
8. Perform assembled fuel pin inspections, which may include radiography, dimensional tolerance checks, bonding, and clad defects

The investigated manufacturing process for ATW fuels is similar to standard Pressure Water Reactor / Boiling Water Reactor type nuclear reactor fuel manufacturing processes. Gaseous enriched Uranium Hexafloride (UF_6), which comes out of an enrichment plant, is delivered in standard cylindrical gas bottles. The UF_6 is then converted to Uranium Dioxide (UO_2), the fuel for which is used. The UO_2 is in the form of a powder, which is then milled to provide suitable sintering properties. The powder is formed into pellets by cold pressing, sintered at approximately 1650°C to 1750°C [6], and ground to final dimensions (See Figure 3).

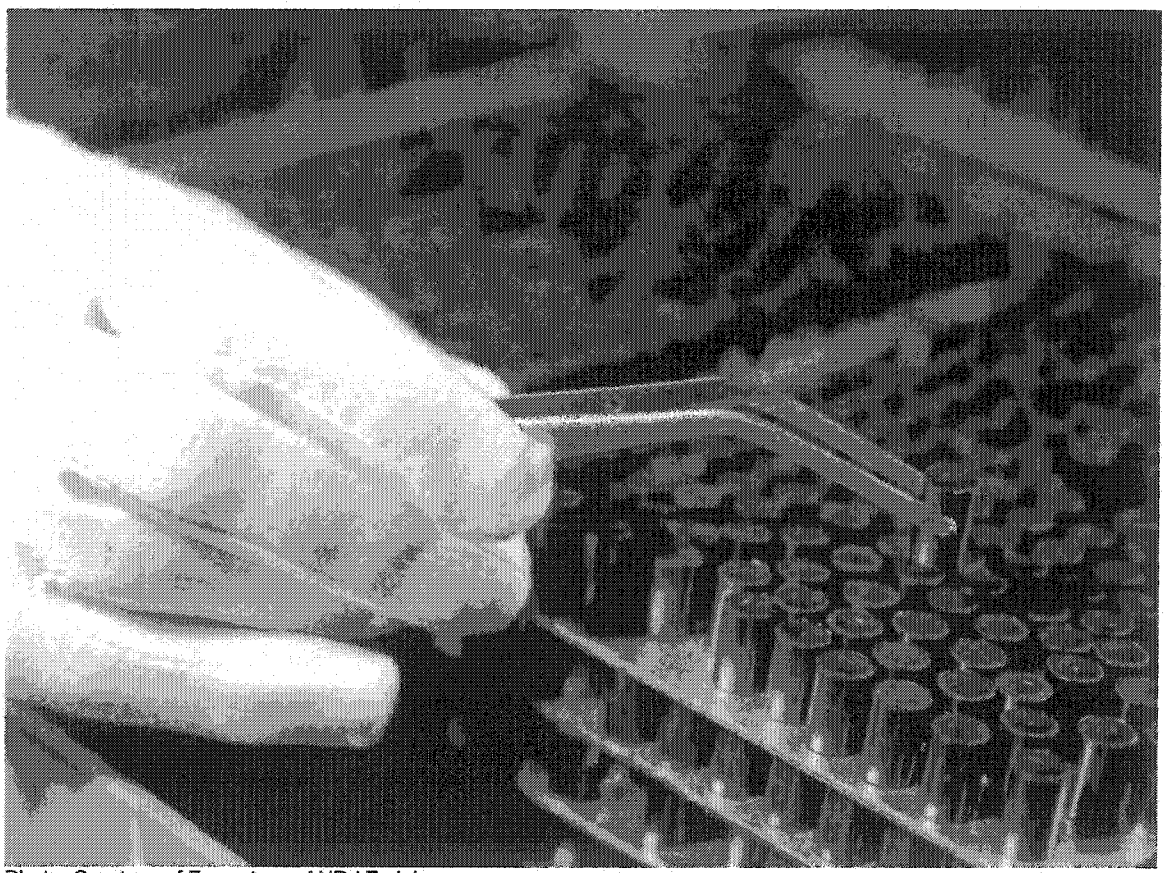


Photo: Courtesy of Framatome ANP / E. Joly

Figure 3. Uranium Oxide Pellets

The pellets are loaded into a zirconium alloy tube, also known as cladding, which is back-filled with helium (He) or other inert gases and sealed at each end with a welded-end plug. The fuel rods are assembled into complete fuel assemblies, as shown in Figure 4. The number of fuel rods in an assembly varies from 50 to 200 or more, depending on the reactor type and specific design [7, p. 9-137].

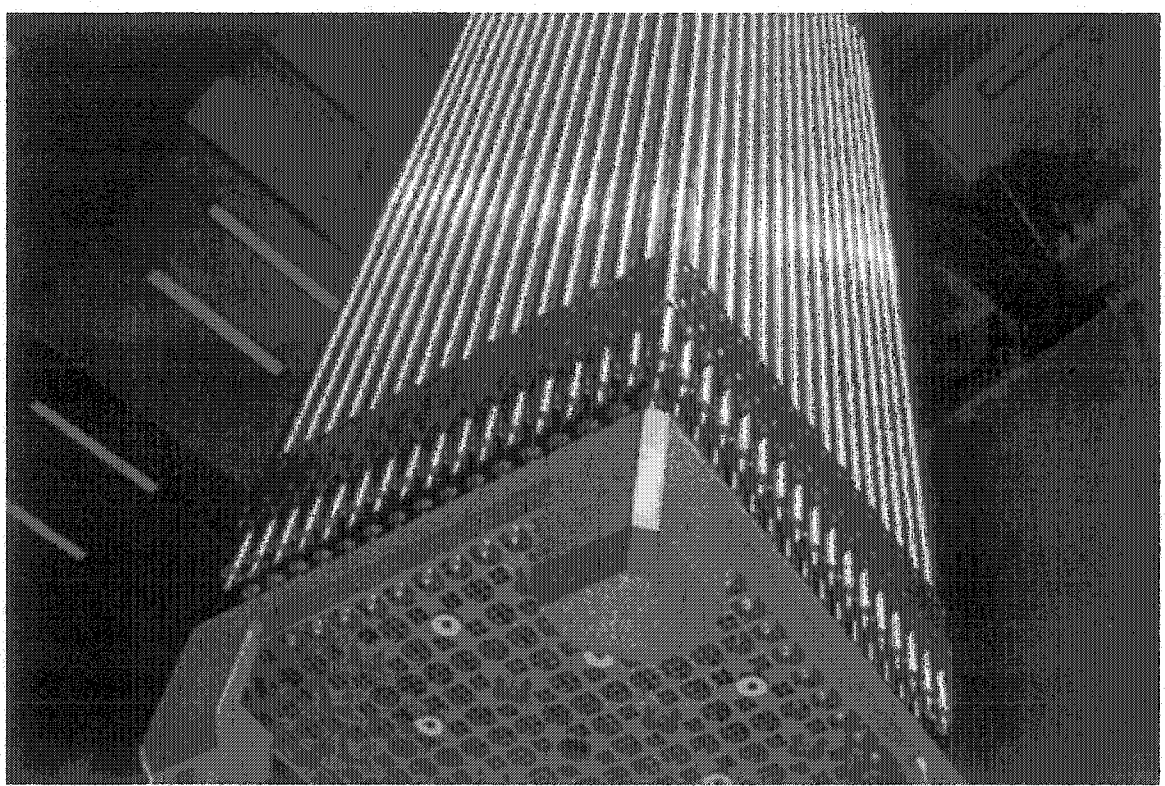


Photo Courtesy of COGEMA / P. Lesage

Figure 4. MOX Fuel in Complete Fuel Assembly at Fabrication Plant in MELOX, France

1.2 Objective of the Study

The objective of this work focuses around two primary tasks, assessing automation and robotics in the nuclear industry, and modeling the back-end of the transmuter fuel

manufacturing process. A secondary objective is to develop a preliminary cycle-time analysis to determine the applicability of process improvements and optimization

1.3 Safety Considerations in Design

Safety concerns are paramount in any nuclear facility. Current nuclear facilities implement safety programs that are based on a defense-in-depth philosophy. The defense-in-depth philosophy relies not only upon the inherent safety features of the engineered design, but also on high quality standards that provide a high degree of assurance that accidents are not likely to occur. And if they do occur, the radioactivity and any related contamination will be contained locally.

High quality begins with the selection of highly qualified personnel to design, construct and operate a nuclear facility. Each applicant for a license to build a nuclear plant must institute a quality assurance program to be applied to design, fabrication, construction, and testing of the structures, systems, and components (SSC) of the facility. An applicant for a license to operate a plant must also provide for appropriate managerial and administrative controls to be used to assure safe operation. The function of the quality assurance program is to provide a high degree of confidence that the SSC's in the plant will perform as intended during the service lift of the plant [7, p. 9-145].

It would be advantageous to design the transmuter fuel manufacturing hot cells so that the major system components are shielded to the maximum extent possible; whereas, if a failure within a system or component occurred, it would not preclude mitigation. For example, should a robotic manipulator fail, any high-level radioactive source could be shielded so mitigation methods that could not be performed remotely could be accomplished by human intervention. This means the radiation exposure should be kept

as low as is reasonably achievable (ALARA). Standard radiation zones are classified a low radiation zone, high radiation zone, and very high radiation zone, see Table 2 below. Manned access into low and high radiation zones is only feasible by reducing the duration the individual is within the zone or by providing localized shielding. Very high radiation zones are not accessible. It is expected that the fuel manufacturing environment will be classified as a low to high radiation zone, but not likely to be a very high radiation zone.

Table 2. Radiation Zone Classification

Zone	Minimum Exposure	Maximum Exposure
Low Radiation	n/a	100 $mrem/_{hr}$
High Radiation Zone	100 $mrem/_{hr}$	5 $rad/_{hr}$
Very High Radiation Zone	5 $rad/_{hr}$	∞

CHAPTER 2

REVIEW OF CURRENT METHODOLOGIES FOR HANDLING RADIOACTIVE MATERIAL

2.1 Department of Energy

There is a wealth of information available through the U.S. Department of Energy (DOE) relating to, not only nuclear reactor fuel manufacturing, but other automated radioactive material handling. While the U.S. Department of Defense remotely handles radiological materials, their activities are of a sensitive nature and due to security concerns, process information cannot be readily obtained. Therefore, the following subsections review a small cross-section of DOE facilities.

2.1.1 Typical Hot-Cell Applications

The design of a work cell to handle high-level nuclear wastes, activated components, and fission materials entails a significant understanding and coordination of multiple disciplines. In an effort to provide basic guidance for the design of hot-cells, the DOE published *Design Considerations* [8]. The handbook discusses general "rule-of-thumb" design considerations as well as good practices. The handbook touches on design considerations and practices for the following topics [8]:

- Architectural
  - Facility Layout
  - Equipment Arrangement
  - Piping Design and Layout
  - Special Systems

- Jumpers
- Structural Design

- Electrical Systems
- Mechanical Systems
  - Piping
  - Purge Systems
  - Pumps
  - Valves
- Instrumentation and Control
  - Control Centers/Control Room
  - Distributed Control Systems
  - Programmable Logic Controllers
  - Alarm Management
  - Electrical Noise and Wiring Practices
  - Lightning Protection for Instruments
  - Analyzers
  - Solenoid Valves
  - Instrument Installation
- Material Considerations

In developing the design for the transmutation processing facilities, analogs to other processing facilities can be made, and therefore, design considerations can be extracted. Within *Design Considerations* [8, p. I-68], the discussion of design considerations for reprocessing facilities is presented. The typical reprocessing facility takes irradiated fissile fuel material or target material and extracts uranium (UREX), plutonium (PUREX), and other selected actinides and fission products.

Contamination control within a processing facility is essential for both criticality as well as build-up of contamination, resulting in radiological "hot-spots." The establishment of confinement zones provides both physical and operational boundaries for the migration of contamination within a facility. For transmuter fuel manufacturing, it can be invisioned that entire hot-cells may be required for the manufacturing process. *Design Considerations* [8, p. I-31] recommends the use of three confinement zones,

primary, secondary, and tertiary. Within the primary confinement, the process should be totally enclosed and provided with its own ventilation system. A secondary confinement system consists of the process cells and their ventilation system. The tertiary confinement system would be the building structure and its ventilation system. The installation, maintenance, and ultimate removal of equipment into and out of the primary or secondary confinement zones should be designed to minimize the spread of contamination. Figure 5 shows the use of master slave manipulators (MSM) penetrating the primary confinement barrier. Failures of the process equipment, such as the MSM's should not cause a failure of the confinement system, and have a minimal impact of the performance of the secondary confinement system.

MSMs are remotely controlled mechanical devices designed to penetrate shield walls so that nuclear materials can be handled and manipulated. Manipulator operations are remote and are performed at a distance. In other words, the operator stands in the operating gallery behind a shielding wall, looking into the cell through a heavily shielded viewing window. A typical operator view through a shield window at the MSM end-effector can be seen in Figure 6. Control inputs by the operator on the master-side of the MSM are mechanically transferred to the slave end through cable, tapes, and electrical devices. In addition to manual movements, MSM's typically have electrical indexing motions. Manual motions allow the operator to duplicate the motions of the human hand, but are limited in the amount of movement throughout the hot cell. Electrical indexing provides the slave end effector with greater motion and reach. The slave end becomes a natural extension of the operators hands and arms into the hot-cell environment and is capable of duplicating all of the master's inputs.

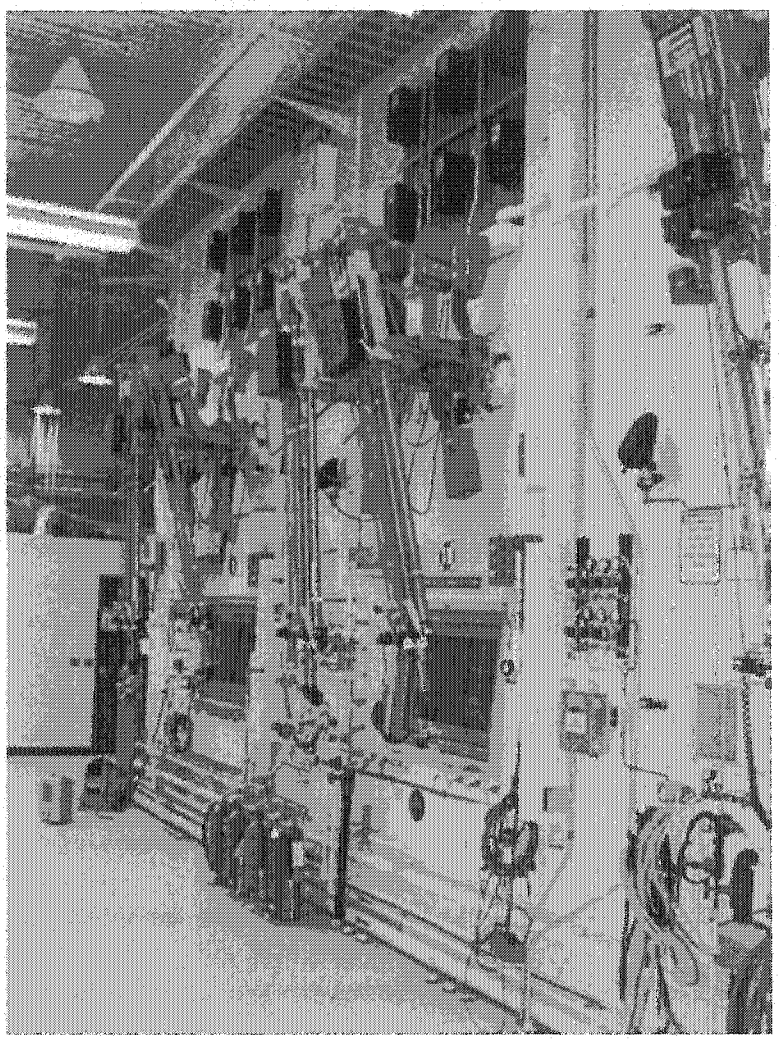


Figure 5. Typical MSM Installation

A close view of the MSM in Figure 7 reveals an actual case where a failure in the contamination barrier, known as the boot, had the possibility of compromising the primary barrier. Here an operator inadvertently moved the MSM too close to the work cell light. The heat from the lamp melted and burned the contamination barrier on the MSM. While this contamination barrier does not directly fail the primary confinement zone, it does have an impact of the secondary confinement system when the MSM is to be replaced or repaired as the internal components of the MSM are now contaminated.

Figure 6 shows the use of a MSM without the use of contamination control. In this case, the expected contamination levels were extremely low, but had very high radiation levels. Therefore, an analysis of the operating environment (i.e., contamination and radiation) would specify the use of confinement zones and other contamination control features.

Figure 6 Through Hot-Cell Window View of MSM End-Effector

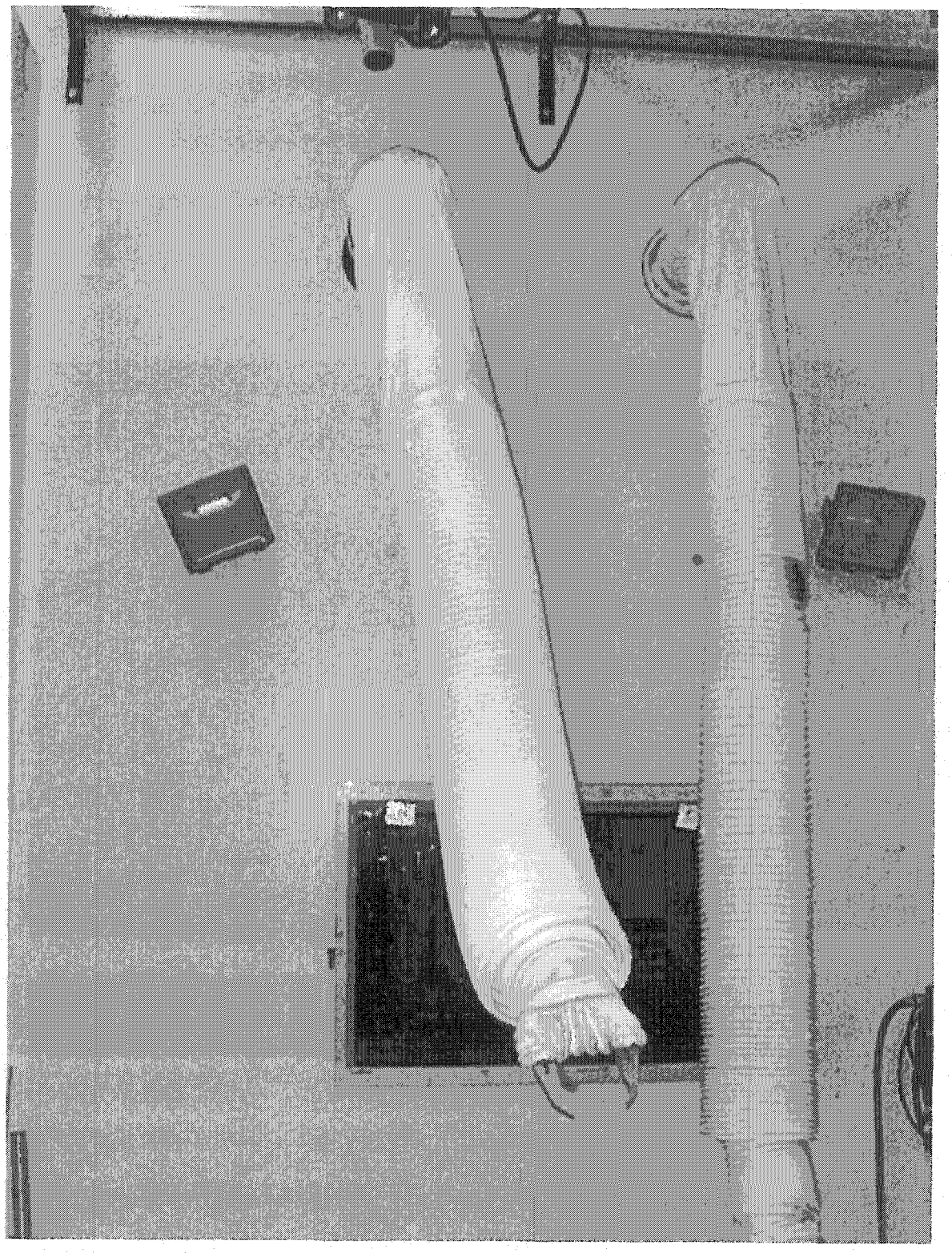


Figure 7 Contamination Control of MSM

2.1.2 Secure Automated Fabrication Program

In 1980, the DOE subcontracted the design, build, and installation of a remotely operated breeder reactor fuel pin fabrication line. The equipment was to be installed in the Fuels and Materials Examination Facility (FMEF), which at the time, was being constructed at the Hanford site near Richland, Washington [9, p. 1]. This automated fuel pin fabrication line was referred to as the Secure Automated Fabrication (SAF) program and had two primary missions: 1) to provide a supply of MOX fuel pins to the U.S. Breeder Reactor Program; and 2) to develop and demonstrate advanced plutonium fuel fabrication technology. Although construction began on the SAF program, DOE canceled the U.S. Breeder Reactor Program and thus the final construction of the FMEF and SAF program. Regardless, the SAF program provides the fuel fabrication community with a valuable resource in design effort and lessons learned.

Similar to the needs of transmutation fuel manufacturing, the original need for the SAF program arose from numerous design objectives, such as:

- Reduction in radiation exposure to workers,
- Enhanced safeguards by restricting operator access to fissile materials and near real-time accountability,
- Improved product quality, and
- Increased productivity.

Numerous articles were written to describe the various aspects of the SAF program, from the powder operations [10] and handling [11], sintering furnace [12] and chemistry analysis [9], sintering boat transport [13], pellet gauging [14], to a general overview of the remote fabrication processes [15].

In the era of the early eighties, the SAF program incorporated advanced automation techniques and processes. The program to remotely manufacture plutonium-based fuel utilized automated processes, including 24 robots, to perform complex operations. The automated equipment was to be controlled remotely from a centrally-located supervisory control and data acquisition system, and alternatively, from local control consoles dedication to individual systems [15, p. 1].

The SAF line had a designed annual throughput of 6 MT of MOX fuel containing up to 20 percent per weight of ^{240}Pu or higher [15, p. 1]. MOX fuel is made with a mixture of uranium and plutonium oxides.

2.1.3 Nuclear Weapons Complex – Pantex Plant

The Nuclear Weapons Complex at the Manson & Hanger Pantex Plant in Amarillo, Texas handles and stores thousands of radioactive nuclear materials. These materials, stored in and referred to as pits, were handled manually, which resulted in a significant accumulation of radiation dose to the workers at the plant. The handling operations included unpacking the pits from their shipping or storage containers, disassembling shipping fixtures that hold the pits, moving bare pits to a variety of measurement and inspection stations, re-assembling the shipping fixtures, and repackaging the containers for storage or shipment [16][17].

To ensure stockpile integrity, an automated weight and leak check system (WALS) was developed. By automating the process, radiation doses to the worker were significantly reduced by eliminating all direct human contact and operations with the pits [17]. The WALS system employed a six-axis robot Fanuc Model S-700 mounted on a linear track to move pits within the automation work cell [18]. By utilizing the automated process, once the operational procedure has been programmed and verified

into the robotic system, all process steps are done with high assurance of completion and completed with the use of qualified and validated procedures [17]. Procedural compliance is strictly enforced within a nuclear facility through an established quality control program, with many audits, self-assessments, and compliance checks. By automating the system, compliance with procedure is guaranteed and logged automatically for future inspections and audits.

2.1.4 PUREX Production Plant Environmental Remediation

The plutonium extraction facility, known as the PUREX Production Plant, is located at the DOE Hanford Site in southeastern Washington State. The PUREX facility became operational in the 1950's and underwent several upgrades and expansions. However, in the early 1990's the decision was made to remediate the facility. As part of the remediation process, an early assessment of the PUREX facilities found two dozen irradiated fuel elements lying at the bottom of one of the canyon hot cells [19]. These spent fuel elements were found in various locations within the hot cell, and would need to be repositioned so that they may be grabbed and retrieved from the cell floor remotely.

Due to the complexity of this operation, Jaquish [19] modeled the PUREX production plant using an interactive graphical robot instruction program (IGRIP). Through the use of the IGRIP simulation environment, the Jaquish was able to plan, review, and verify various proposed remediation activities.

Jaquish [19] collected various drawings and photographs of certain areas of the PUREX production plant so that the IGRIP model would produce a virtual walk-through inside the facility. The IGRIP software was used in developing the primary equipment to retrieve and process the irradiated fuel pellets left within the facility. The simulations showed that critical interfaces and dimensions could be assessed and implemented in the

design. Jaquish also used the IGRIP simulation to optimize the operations. Because there was flexibility in the conduct of operations for most tasks, sequence scheduling was performed to find optimum tasks that could then be implemented into a procedure.

2.1.5 Multipurpose Canister Handing

As part of the Civilian Radioactive Waste Management System, the development of a conceptual design for Multipurpose Canister (MPC) handling was performed in a simulated environment. An MPC is a sealed metallic canister designed for storage, transportation, and ultimate disposal of spent nuclear fuel assemblies. Bennet and Stringer [20] utilized IGRIP to model MPC handling at the proposed Monitored Retrievable Storage Facility, the precursor to the Yucca Mountain Project.

A complete process flow for MPC handling was simulated to visualize the sequence of preparing an MPC, e.g., fuel loading, welding, and inspecting. The simulations helped Bennet and Stringer [20] develop equipment requirements and costs. The equipment generally consisted of a single industrial robot and/or a programmable crane per work cell. In addition, a Stewart Platform was modeled and simulated for use in the process operations. The simulation validated process time estimates for each work cell at 25- and 100-percent of maximum equipment operating speeds, all under fully automated process control.

Ultimately Bennet and Stringer [20] used the IGRIP simulation of the Monitored Retrievable Storage Facility to identify operations that could be automated with robotic machinery. The results of their simulations showed, in the simulated environment, how particular operations could be executed automatically, identified equipment requirements and operational characteristics for the automation, and determined potential process times for each automated operation.

2.2 Higher Education

Mr. Tao Chen explored the use of robotics for nuclear waste handling. Although the final application of his dissertation focused mainly on the controllability of a commercially available robotic system, he faced challenges in determining the forward and reverse kinematic analysis [21].

In 1989, Kruger [22] investigated the design and use of 3-D computerized model of a robotic arm. While simplistic in both the robot model fidelity and the number of degrees of freedom, the authors work did expand the use of a modeling environment.

In 1995, O'Donnell [23] explored the use of operator assisted control of a robotic manipulator. In the thesis entitled *Automated Robotics System for Nuclear Waste Handling (Hardware and Software)*[23], the O'Donnell describes the system as consisting of a computer and accompanying software placed between an operator control station and a remote manipulator. The computer and associated software checks for collisions within the work cell and other objects defined a computer model. If no collisions are predicted, normal master/slave operation continues; otherwise, the computer takes control until the operator guides the manipulator into a safe position [23, p iii]. A majority of the author's work centered around data communications between the master and the slave, developing software code to communicate between diverse "off-the-shelf" components and their related software. While never achieving true automation, the author did show successes in real-work test cells. As the master received commands by an operator, the system would check for collisions based on a computer aided design (CAD) model of the environment. If the system predicted a collision with the environment, the system would take control of the slave by simply halting the

movement (i.e., freezing the robot). When the operator returns the master controller to a non-colliding position, control is returned to the operator [23, pp. 37-38].

2.3 International

International work in the nuclear industry is vast and extensive. The following subsections will provide a summary of the nuclear facilities that have implemented robotics and/or automation into their processes. While non-inclusive, the following subsections show a small cross section in which robotics and automation have been implemented world wide; and thereby, implies the lack of automation implementation by the United States, particularly the DOE, as discussed in Section 2.1.

2.3.1 Melox Plant

The *Compagnie General des Matieres Nucleaires* (COGEMA) MELOX plant, brought on line in 1995, is one of the most efficient and modern MOX fuel fabrication plants, where MOX fuel is a blended mixture of plutonium oxide and depleted uranium oxide. The automated fuel fabrication design increases throughput and enables the facility to produce fuel assemblies at a rate well above one 500 kg assembly per day, which is approximately 264 fuel rods and 100,000 MOX fuel pellets [24][25]. The MELOX plant estimated throughput is approximately 160 metric tones of heavy metal per year. Pending approval, design and process improvements may take the facilities capacity to over 250 metric tones of heavy metal per year [24].

2.3.2 La Hague Reprocessing Plant

The mission of the French company's COGEMA - La Hague plant, which entered service in 1966, is to reprocess spent nuclear fuel. The COGEMA La Hague industrial

complex occupies a 740-acre site 15 ½ miles west of Cherbourg on the tip of the Cotentin peninsula.

The La Hague plant receives spent nuclear fuel from reactors for processing, which consists of separating and conditioning the various spent fuel components. The separation process recycles the uranium and plutonium and disposes of the non-reusable materials, where the majority of the spent fuel's radioactivity resides.

Since La Hague entered into service, the operational philosophy and design has evolved where they have implemented numerous processes and design improvements. While exact design improvements are proprietary and could not be obtained, generalized design philosophies are available. An example of the design improvements is that the La Hague facility has been able to reduce mean occupational exposures from 500 mrem/yr to 150 mrem/yr from 1976 to 1986, even though the amount of reprocessed spent fuel has increased [26]. The philosophy behind some of their major design improvements leading to the reduced occupational exposures was obtained by utilizing a design-for-maintainability approach, where [26, p. 6]:

1. The utilization and implementation of standardized equipment specifically designed for hot-cell and other high radiation nuclear environments that is capable of remote dismantling and maintenance
2. The SSCs requiring replacement and/or refurbishment are modularized, which facilitates the removal and replacement of the SSCs.
3. Locating SSCs requiring frequent replacement in strategic locations, which also facilitates their removal and replacement.

4. The use of a "cold" test facility to prototype, test, debug, and enhance the design of SSCs prior to their use in the "hot" environment.

Using these design improvements and philosophies, COGEMA has been able to build highly automated, remotely operated facilities, which shield site personnel from radiation doses. The result of implementing these design philosophies is extremely low dose rates to COGEMA-La Hague personnel since the early 1990s.

For example, COGEMA designed a spent fuel disassembly and consolidation system (called DEC) for the COGEMA-La Hague plant. This design permitted the disassembly of spent fuel assemblies and subsequent consolidation of the spent fuel rods into densely packed canisters [26, pp. 51-52].

The DEC facility design utilized a single disassembly cell equipped with a remotely operated and maintained disassembly machine. The disassembly machine was designed to remotely cut the guide tubes from Pressurized Water Reactor (PWR) fuel assemblies and remove the top-end fitting. The disassembly machine would grip and pull all fuel rods simultaneously out of the PWR fuel assembly, and then place these rods into a canister. After which, the disassembly machine would shear and compact the fuel assembly skeleton to maximize volume reduction. The disassembly machine design allowed adequate flexibility to handle all PWR fuel assembly types, which was accomplished by changing the fuel rod pulling grapple [26, pp. 51-52].

Using the same design-for-maintainability philosophy as described above, the DEC was automated and designed in modules so that it could be easily maintained and remotely operated and replaced. The entire consolidation system, including the disassembly machine, is controlled from a local control workstation outside of the hot-

cell. Control panels and television monitoring systems provide remote control and viewing of all consolidation operations. However, the design included the capability of direct viewing through shielded windows as a backup [26, pp. 51-52].

2.3.3 Atomic Energy of Canada Limited

In 1995, the TELBOT manipulator was used as a remote handling system for the automated cleaning of the primary side of the steam generator tubes at Pont Lepreau NGS in New Brunswick, Canada. The TELBOT, shown in Figure 9, was used to handle and position cleaning equipment in the steam generator bowl. The TELBOT was simulated, tested and deployed by Atomic Energy of Canada Limited (AECL). The robot was programmed to run 24 hours a day over a period of 23 days, and successfully cleaned 8,209 tubes.

In addition in 1996, the TELBOT manipulator was used to transport and position cleaning equipment for the decontamination of a particular hot cell at AECL's Chalk River nuclear laboratory. The hot cell had relatively high dose rates and contamination levels, which prevented extended human operations within the cell; and therefore, the TELBOT was selected for use in the decontamination activities. Figure 10 shows the preparation of the TELBOT for deployment within the hot cell. Workers are "bagging" the robot by wrapping the manipulator in a plastic shroud to provide contamination control. After the TELBOT activities are completed, any contamination is affixed to the plastic and subsequently removed, leaving a relatively contamination free robot.

AECL produced computer simulations of the robot working in the environment. An iterative process of simulation and design refinement helped perfect the operations and robot movements prior to start of the decontamination process. Figure 8 shows a mock-

up of the TELBOT manipulator for real-time simulation of non-destructive testing of steam generator tubes.

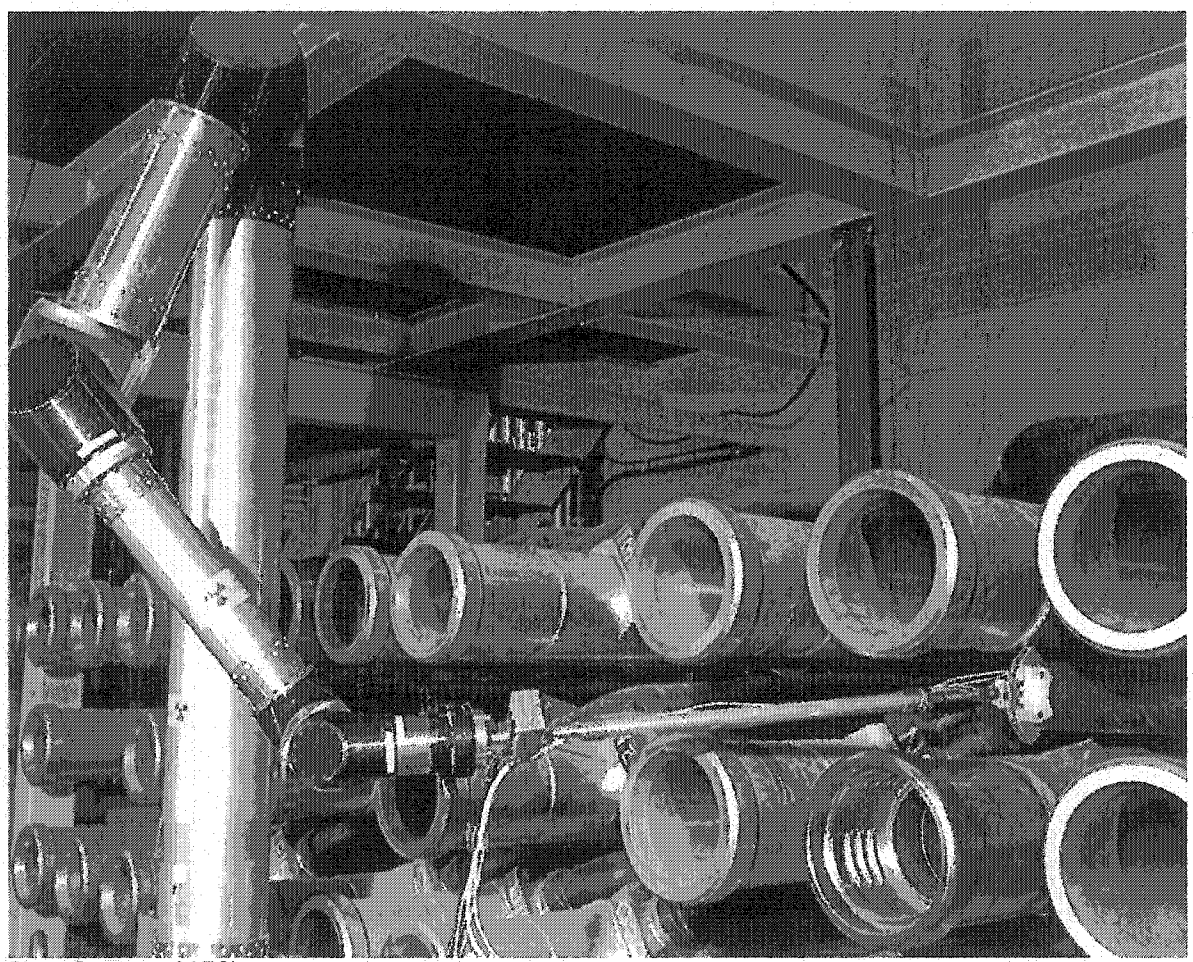


Photo Courtesy of AECL

Figure 8. TELBOT Used in Mock-up for Ultrasonic Pipe Inspection

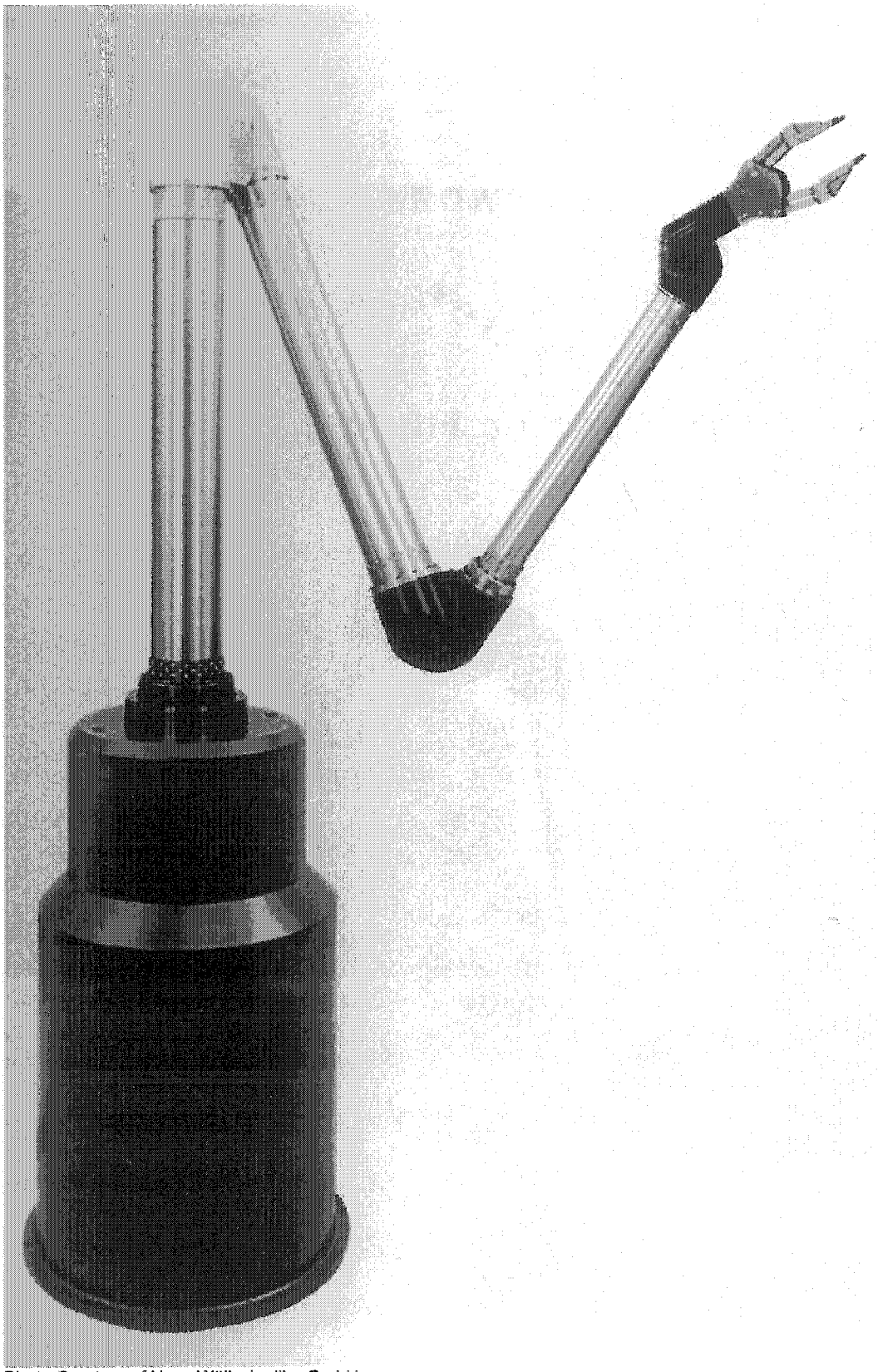


Photo Courtesy of Hans Wälischmiller GmbH

Figure 9. TELBOT TB100

Photo Courtesy of AECL

Figure 10. Hot Cell Preparation of the TELBOT TB100

CHAPTER 3

ROBOT MODELING

3.1 TELBOT Manipulator System Overview

A general-purpose industrial robot, trade named TELBOT, has been developed by Hans Wälischmiller GmbH, as shown in Figure 9. While relatively new to the nuclear industry, TELBOT has been used in high radiation environments for cleaning steam generator tubes at the primary side in Canada. Also, the robot has been used for decommissioning glove boxes in Japan [28].

The design of the TELBOT manipulator is specifically intended for hazardous and contaminated environments, while still maintaining a lightweight structure. For each of the six rotational joints and the operation of the end-effector, a respective drive motor resides in the robot base, as shown in Figure 11. Torque for each joint axis is translated from the drive motor, through transmission gears and safety clutches within the base, and through concentric torque tubes and bevel gears to each joint. By co-locating all of the motors, gears, sensors, electrical wiring, and related equipment in the robot base, the arms only contain mechanical linkages and are free to rotate a full 360 degrees, as well as, providing contamination sealing at each of the joints. This becomes important in very dirty or highly contaminated environments [28] [29].

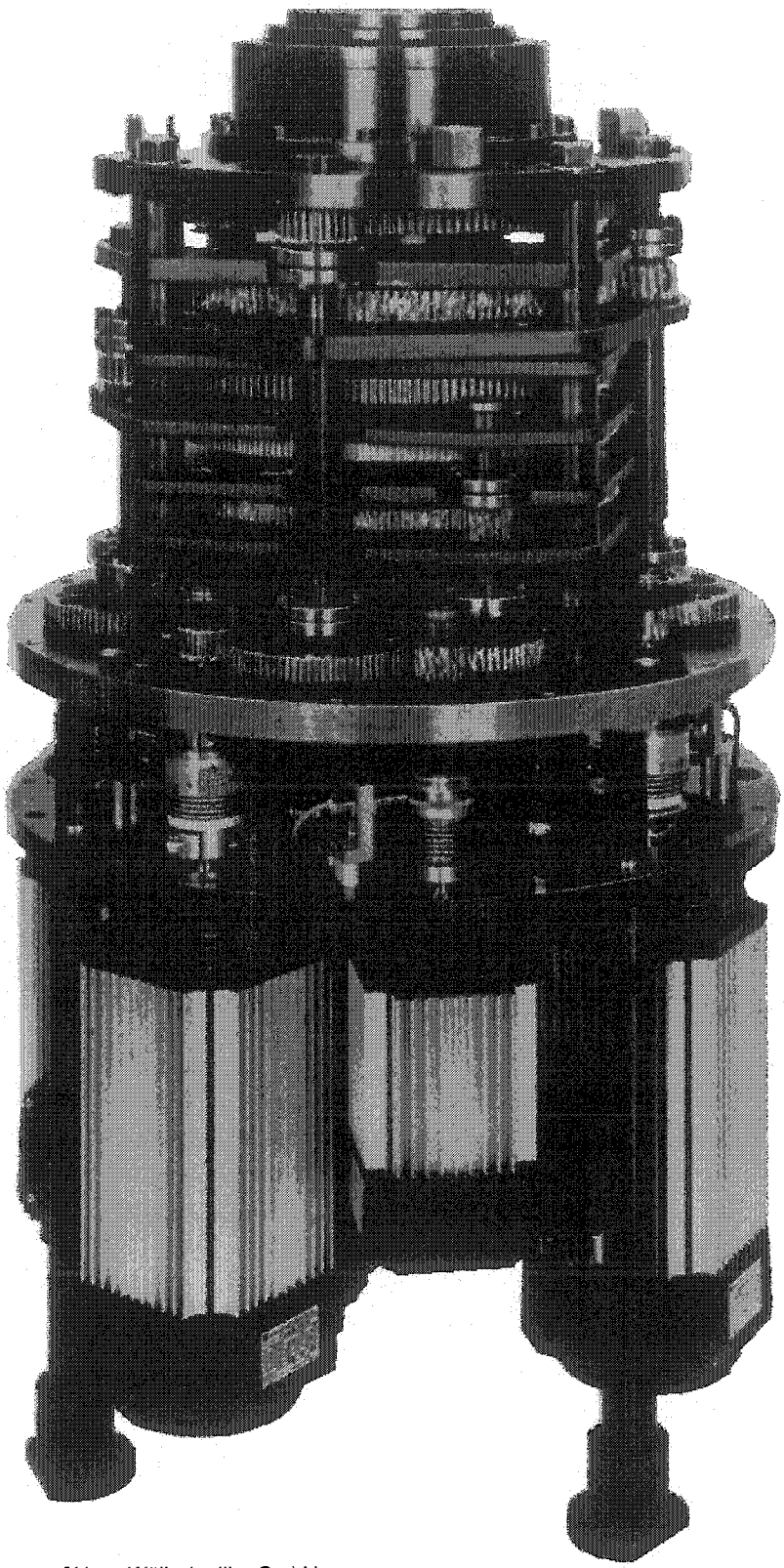


Photo Courtesy of Hans Wälischmiller GmbH

Figure 11. Drive Unit for the TELBOT TB100

The fundamental design of the TELBOT skeleton is based on a modular approach. Each joint, member, and its related bevel gear ratios can be interchanged. This allows for the manipulator to be configured for various working envelopes, payloads and transmission ratios. The modular design also allows for quick interchange of parts and members [28].

The TELBOT configuration has several advantages at the cost of some dynamic anomalies. The design of the robot base, with its corresponding drive motors, gearing, sensors, and wiring, allows the manipulator arms to pass through a shield wall port, much like a traditional MSM, while keeping the sensitive motors and sensors out of the high radiation environment.

This configuration also allows the manipulator members to be manufactured from lightweight materials and tube structures. However, due to this construction and configuration, each joint axis exhibits dynamic deflections and elasticity.

In reality, the TELBOT members act as torsional springs and need to be modeled in such a manner. This work assumes that for light loads (e.g., fuel pins), the defection of each concentric torque tube is negligible and can be considered a rigid structure, as will be further justified in the following section (Section 3.2)

3.2 Computer Aided Design of TELBOT System

Schlotter & Pfeiffer [29] explored modeling and control of the TELBOT and its above mention dynamic effects related to torsional stiffness. The authors developed an elastic multi-body system where each of the torque tubes driving the six rotational joints and the gripper were assumed as torsion springs, each with a specific stiffness and damping.

To verify the Schlotter & Pfeiffer's mathematical model of the TELBOT, a CAD model was developed by the Schlotter & Pfeiffer for the robotic system. From the CAD model, the authors obtained the robots inertial parameters. Performing a Finite Element Analysis on the CAD model, they derived the stiffness of the robot and related torque tubes. The friction values for the joints were obtained from actuarial data. Through simulation, the authors were able to determine the elastic deformation of the trunk and forearm, anatomically speaking, which would typically experience the most deflection and have the most detrimental effect on the positional accuracy of the end-effector. Based on the figures generated by Schlotter & Pfeiffer [29], their maximum deflection was approximately 0.06 mm in the trunk and 0.03 for the forearm.

It is intuitive that the deflection calculated is a function of payload, robot configuration, joint accelerations, and other dynamical effects. However, based on their relatively small deflection results, any effects due to dynamic deflection on the simulation of the TELBOT within this scope of work is assumed to be negligible.

3.3 Mathematical Modeling of TELBOT

The mathematical modeling of the TELBOT follows the methodology established by Fu et al. [30]. In the body of this work, kinematic analysis is performed where only the geometry with respect to time of robot motion is studied without divulging into the forces or torques required to solve for the equations of motion. Position, velocity, and acceleration of the TELBOT joints, as well as end-effector, are of particular concern; and therefore, only a kinematic analysis is required. Velocity and acceleration evaluations are performed in Section 3.5, which uses a pseudo-inverse Jacobian methodology to iteratively solve for the inverse kinematics.

3.4 Forward Kinematic

Forward kinematics mathematically represents the robot geometry in space. Of particular concern is the location and orientation of the robot end-effector with respect to a particular coordinate system (e.g., pose), typically at the robot base or a central work-cell coordinate system. The position of the end-effector of the TELBOT manipulator, a six degree of freedom system, is translated from the robot base by six rotational components, which allow the TELBOT manipulator to reach any arbitrary end-effector pose within the robot's workspace. A conversion from the various robot joints to the end-effector location and orientation is the root of forward kinematics, where the end-effector pose is mathematically translated from joint-variables space to Cartesian-variable space.

Nof [31, p. 83] provides some definition of the robot workspace, where the workspace of a particular manipulator is defined as the set of all end-effector locations (poses) that can be reached by arbitrary choices of joint variables. If the complete end-effector pose (i.e., both end effector position and orientation) is considered, then the workspace is classified as the *complete workspace*. In some instances, the complete end-effector pose is not required for discussion and only the position is of concern, which gives the *reachable workspace*. The subset of the reachable workspace that can be attained with arbitrary orientations of the end-effector is the *dexterous workspace*. We will be discussing the complete workspace within this work.

Within the robots complete workspace, one could image a homogenous coordinate system located at the base of the robot and a second homogenous coordinate system located at the end-effector. One could then draw a vector ($\vec{r}$) from the origin of the base coordinate frame to the origin of the end-effector coordinate frame. It would be

beneficial to be able to transform this vector, known as the position vector ($\hat{\mathbf{p}}_{xyz}$) from one coordinate system representation to another, such as the base coordinate to the end-effector coordinate system. In fact, this position vector ($\hat{\mathbf{p}}_{xyz}$) can be transformed from one coordinate system, both rotationally and by translation, to an alternate coordinate system by the basic homogenous transformation matrix (**T**), resulting an a new position vector ($\hat{\mathbf{p}}_{uvw}$). This is shown in equation 1:

$$\hat{\mathbf{p}}_{xyz} = \mathbf{T} \cdot \hat{\mathbf{p}}_{uvw} \qquad (1)$$

As Fu et al. reveals [30], Denavit and Hartenberg developed a mathematical tool in the form of a matrix, known as the D-H Coordinate Matrix, that would systematically establish a body-attached coordinate system frame to each link of an articulated chain, e.g., a series of robotic linkages [32]. These body-attached coordinates are shown in Figure 12 for the TELBOT. The matrix is a 4×4 homogenous transformation matrix representing each link's coordinate system at the joint with respect the previous link's coordinate system [30, pp. 35-36][33, p. 42].

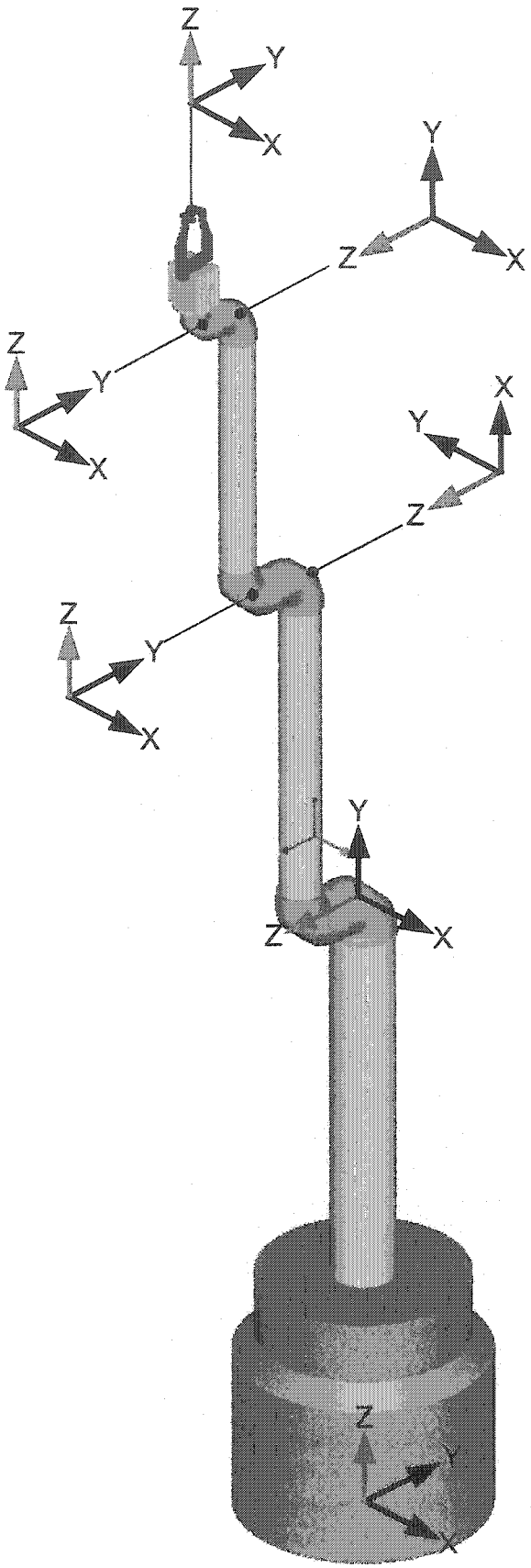


Figure 12. Coordinate Representation of the TELBOT Manipulator

The rules for determining the D-H Coordinate Matrix are summarized as [30]:

θ_i = the joint angle from the $\mathbf{x}_{i-1}$ axis to the $\mathbf{x}_i$ axis about the $\mathbf{z}_{i-1}$ axis (using the right hand rule)

d_i = the distance from the origin of the $(i\text{-}1)^{th}$ coordinate frame to the intersection of the $\mathbf{z}_{i-1}$ axis with the $\mathbf{x}_i$ axis along the $\mathbf{z}_{i-1}$ axis

a_i = the offset distance of the $\mathbf{z}_{i-1}$ axis with the $\mathbf{x}_i$ axis to the origin of the i^{th} frame along the $\mathbf{x}_i$ axis (or the shortest distance between the $\mathbf{z}_{i-1}$ and $\mathbf{z}_i$ axes)

α_i = the offset angle from the $\mathbf{z}_{i-1}$ axis to the $\mathbf{z}_i$ axis about the $\mathbf{x}_i$ axis (using the right hand rule)

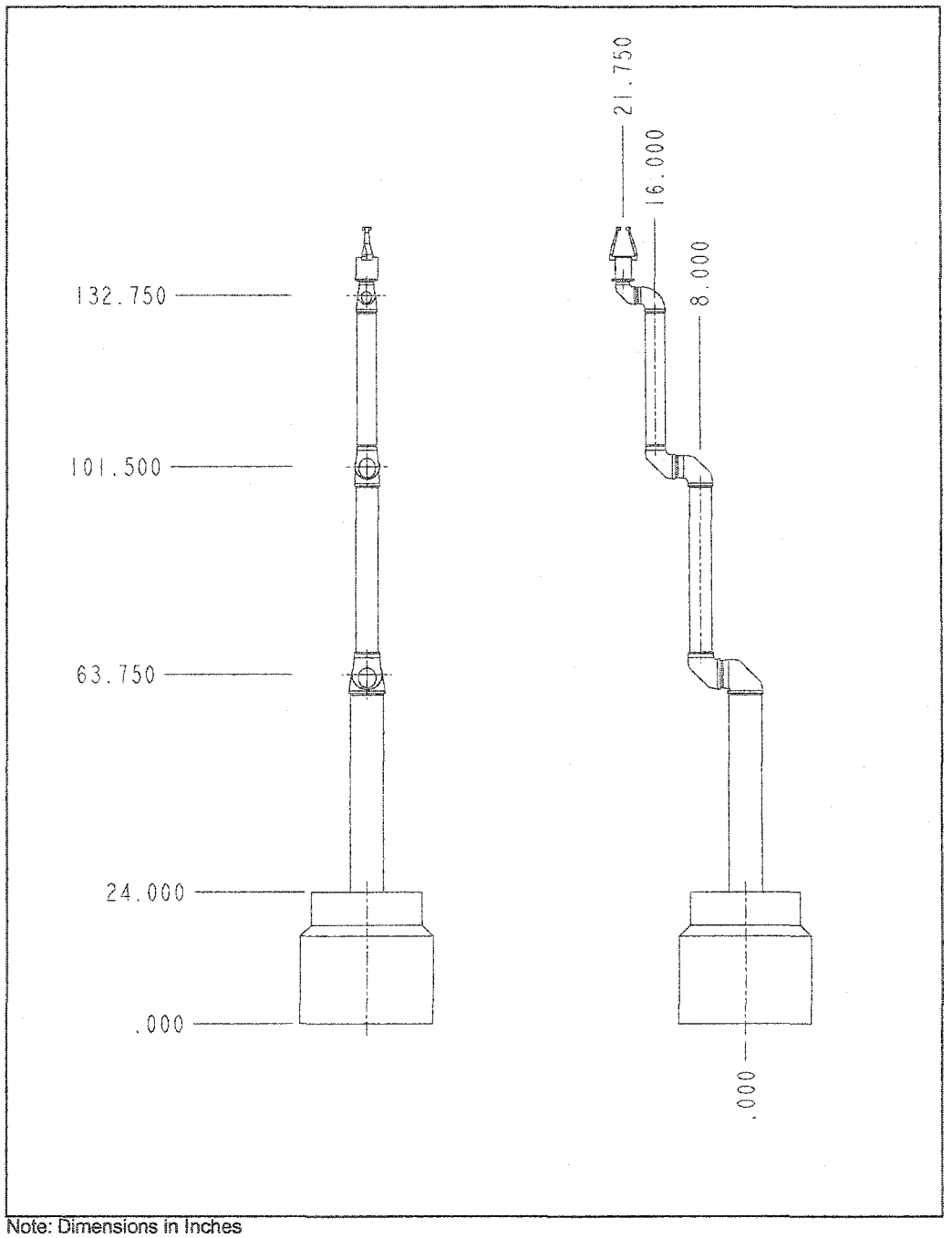
Following this methodology and mapping each link's coordinate system, the D-H Coordinate Matrix for the TELBOT is shown in Table 3 below.

Table 3. D-H Coordinate Matrix for the TELBOT

Joint *i*	θ_i	α_i	a_i	$\mathbf{d_i}$	**Joint Range**
1	$\theta_1 + \pi/2$	0	0	71.75	360 / Full Rotation
2	$\theta_2 - \pi/2$	$-\pi/2$	37.75	8	360 / Full Rotation
3	θ_3	0	0	8	360 / Full Rotation
4	θ_4	0	0	31.25	360 / Full Rotation
5	θ_5	0	0	5.75	360 / Full Rotation
6	θ_6	0	0	12	360 / Full Rotation

As discussed in Section 3.1 - *TELBOT Manipulator System Overview*, the TELBOT utilizes all rotary joints capable of full rotation. For a rotary joint, d_i, a_i, *and* α_i are the link and joint parameters, which remain constant for a robot, while θ_i is the joint variable that changes when link i rotates with respect to link i-1.

The inherent nature of the TELBOT design allows for each link to be a variable length, specified by the customer. While an "off-the-shelf" design for the TELBOT may exist, we assumed specific dimensions for our simulations.



Note: Dimensions in Inches

Figure 13. Assumed TELBOT Dimensions

Figure 13 above shows the overall assumed size of the TELBOT. Figure 10 shows an actual image of the TELBOT where workers are preparing for hot cell use, which provides a good understanding of the robot size. While the dimensions for the TELBOT used in this work are not factory-specified, Figure 10 provides a good basis that the dimensions used are consistent with actual equipment.

Once the D-H coordinate system has been established for each link, a homogeneous transformation matrix (T) is developed relating the i^{th} coordinate frame to the $(i\text{-}1)^{th}$ coordinate frame, as discussed in Equation 1.

The homogeneous rotation and translation matrices are multiplied together to obtain a composite homogeneous transformation matrix (T). However, matrix multiplication is not commutative, and therefore, one must ensure that the order of multiplication is correct, otherwise resulting in an erroneous transformation matrix.

$$\mathbf{T}_{rotation} = \begin{bmatrix} [\text{Rotation Matrix}]_{3\times3} & [\text{Position Vector}]_{3\times1} \\ [\text{Perspective Transformation}]_{1\times3} & [\text{Scale Factor}]_{1\times1} \end{bmatrix} \quad (2)$$

$$\mathbf{T}_{transformation} = \begin{bmatrix} [\text{Localized Scale Matrix}]_{3\times3} & [\text{Position Vector}]_{3\times1} \\ [\text{Perspective Transformation}]_{1\times3} & [\text{Scale Factor}]_{1\times1} \end{bmatrix} \quad (3)$$

$$\mathbf{T}_{x,\alpha} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

$$\mathbf{T}_{y,\phi} = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$$\mathbf{T}_{z,\theta} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

$$\mathbf{T}_{trans} = \begin{bmatrix} n_x & s_x & a_x & dx \\ n_y & s_y & a_y & dx \\ n_z & s_z & a_z & dx \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{n} & \mathbf{s} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

$$\mathbf{T}_{trans}^{-1} = \begin{bmatrix} n_x & s_x & a_x & -\mathbf{n}^T\mathbf{p} \\ n_y & s_y & a_y & -\mathbf{s}^T\mathbf{p} \\ n_z & s_z & a_z & -\mathbf{a}^T\mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} & \mathbf{R}_{3\times3}^T & & -\mathbf{n}^T\mathbf{p} \\ & & & -\mathbf{s}^T\mathbf{p} \\ & & & -\mathbf{a}^T\mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

Where, for the case of an end-effector:

- **n** = Normal vector, assuming a parallel-jaw end-effector, it is orthogonal to the fingers of the robot arm, which is the case modeled for the TELBOT manipulator.
- **s** = Sliding vector – it is pointed in the direction of the finger motion as the gripper opens and closes.
- **a** = Approach vector – it is pointing the direction normal to the palm of the hand
- **p** = Position vector – it points from the origin of the base coordinate system to the origin of the hand coordinate system, which is usually located at the center point of the fully closed fingers.

For adjacent coordinate frames (i and i-1), a composite homogenous transformation matrix ($^{i-1}\mathbf{A}_i$) can be determined and known as the D-H transformation matrix. Thus,

$$^{i-1}\mathbf{A}_i = \mathbf{T}_{z,d}\mathbf{T}_{z,\theta}\mathbf{T}_{x,a}\mathbf{T}_{x,\alpha}\dots\mathbf{T}_i \quad (9)$$

Therefore, using the composite homogenous transformation matrix, kinematic equations for manipulators are formulated, where

$$ {}^{0}\mathbf{T}_i = {}^{0}\mathbf{A}_1 \, {}^{1}\mathbf{A}_2 \ldots {}^{i-1}\mathbf{A}_i = \prod_{j=1}^{i} {}^{j-1}\mathbf{A}_j \qquad (10) $$

The direct kinematics solution for a manipulator is, therefore, simple a matter of calculating the ${}^{0}\mathbf{T}_i$ by chain-multiplying the individual ${}^{i-1}\mathbf{A}_i$ matrices and evaluating each element in the **T** matrix. As Fu et al. [30] further explains, the direct kinematics approach produces a unique solution (**T** matrix) for a given input matrix **q**, where:

$$ \mathbf{q} = (\theta_1, \theta_2, \ldots, \theta_6)^T \qquad (11) $$

3.5 Inverse Kinematics

Inverse kinematics is used to find the joint angles in joint-variable space that will place the end-effector at a desired location in Cartesian-variable space. The forward kinematic analysis of the TELBOT manipulator system is straightforward. However, the inverse kinematic solution proved to be considerably more difficult. The six-revolute (6R) robot utilizes unique joints that are able to rotate over 360 degrees allowing the end effectors to attain its work position at multiple orientations. Because of the configuration and 6R nature of the TELBOT manipulator, there is no closed-form solution of the inverse kinematics. This is because the wrist-joint axes of the TELBOT violate one of the fundamental rules of the Denavit-Hartenberg method, where the wrist-joint axes do not intersect at one point [32][30], as is the case with a spherical end-effector.

Most industrial manipulators utilize a spherical wrist, where the wrist-joint axes intersect at one point. For a 6R robot, the wrist is positioned using the first three joints (dexterous workspace), and the reachable workspace is visualized as a sphere at the end of the wrist joint [31, p. 83].

Lee and Wälischmiller [28], Nof [31, pp. 84-85], Paul [34], and Goldenberg et al. [35] all showed that for a 6R system that met the Denavit-Hartenberg method (i.e., as is the case with most industrial manipulators), a closed-form solution can be obtained. There are up to 16 solutions of the inverse kinematics for robots with six revolute joints.

However, for the TELBOT, a non-zero offset at joint-5 is used; and therefore the closed form solution does not exist. Lee and Wälischmiller [28] showed that for the TELBOT, the inverse kinematics can be reduced to a 16^{th} order polynomial with one unknown and five linear equations in the other five unknown joint angles. For these cases, the method in solving the inverse kinematics problem is to disconnect the manipulator into two sub-chains at two joints and then formulate scalar products of the vectors representing the joints and links for both sub-chains. Once the single unknown is solved numerically, the other five unknowns of the linear equations are solved simultaneously [28].

3.5.1 Iterative Inverse Kinematics Solution

For this work a solution was found, albeit less robust, through an iterative approach using the Differential Translation and Rotation Method [34][30, pp. 547-553], also known as Jacobian Control Method. A solution was found for many poses of the TELBOT manipulator. The inverse kinematics routine, derived from the *Robotics Toolbox* [36], was used to perform the iterative solution for the TELBOT system. While fairly robust in solving the inverse kinematics for most poses, particular robot configurations resulted in the kinematic equations to be ill conditioned and the routine would not converge on a solution. Thus, the determination of the robot configuration for any arbitrary end-effector location and orientation may result in a kinematic singularity, as further discussed in Section 3.5.2.

Manocha and Canny [37] explored the inverse kinematics problem for general serial manipulators with six joints. They also found difficulties solving the inverse kinematics routines employing numerical methodologies. Manocha and Canny [37], Park et al. [38], Goldenberg et al. [35], and Grudic and Lawrence [39] all discussed that classical Newton-based numerical routines are typically too slow for practical applications and are unable to find all solutions. Manocha and Canny [37] were able to develop an inverse kinematic method for a general 6R manipulator by configuring the inverse kinematics method as an eigenvalue problem, which was solved numerically. Grudic and Lawrence [39] developed a method to solve the inverse kinematics by use of an offset modification method. They systematically modified the manipulator offset (e.g., position in Cartesian space) until it was possible to derive closed-form inverse kinematic equations for the modified manipulator. These closed-form inverse kinematics equations were then used to obtain a system of three nonlinear equations in three unknowns that could be solved using established numerical techniques. Park et al. [38] developed a method of solution by arbitrarily adding enough internal or external constraints to make the originally ill-posed manipulator a well-posed system

Park et al. [38] method of arbitrarily adding constraints to a manipulator is a tool often employed in solving the inverse kinematics for redundant manipulators. A manipulator is considered redundant when the robot has more degrees of freedom than the task space. In other words, a robot with more than six degrees of freedom is redundant in Cartesian Space, which is specified by six degrees of freedom – three for position and three for rotation. For simulation of a redundant manipulator, this technique of adding constrains has been used by Sciavicco and Siciliano [40] to solve for the

inverse kinematics. As Ahuactzin and Gupta [41] states, kinematically redundant manipulators are gaining increased appeal in the robotics arena in that the additional degrees of freedom can be used to avoid singularities and collisions, and to optimize performance criteria.

As stated in Section 3.5, inverse kinematics requires the transformation from Cartesian space to joint variable space. From Equation 1, we have the forward kinematic mapping from the joint-variable space to the Cartesian space. It would be intuitive to assume that the inverse kinematics solution would simply be:

$$\hat{\mathbf{p}}_{uvw} = \mathbf{T}^{-1} \cdot \hat{\mathbf{p}}_{xyz} \tag{12}$$

Utilizing Equation 12 is a common approach for solving this problem where a closed-form solution to the inverse transformation ($\mathbf{T}^{-1}$) exists. As Corke [36], Manocha and Canny [37], and Goldenberg et al. [35] point out, this solution is non-unique, and for some classes of manipulators, no closed-form solution exists, as is the case with the TELBOT manipulator.

Similar to the transformation matrix (T), the Jacobian matrix (J) maps the respective joint velocities ($\dot{q}(\theta)$) in joint-variable space to end-effector velocities ($\dot{x}(\vec{r})$) in Cartesian-variable space.

$$^{0}\dot{x}_{n} = {}^{0}\mathbf{J}_{\theta} \cdot \dot{q} \tag{13}$$

For the TELBOT, or any other 6R manipulator, the Jacobian is a 6×6 matrix. Therefore, in solving for the joint-variables (q), the inverse of the Jacobian, a square matrix, is taken.

$$\dot{q} = {}^{0}\mathbf{J}_{\theta}^{-1} \cdot {}^{0}\dot{x}_{n} \tag{14}$$

The solution of the inverse of the Jacobian is shown in invkin.m, the inverse kinematics routine:

```
    dq = inv(tb_jacobian(robot, q)) * e;
    q = q + dq;
    nm = norm(dq);
    count = count+1;
    if count > ilimit,
        errordlg('Solution wouldn''t converge','Inverse
Kinematics');
        error('Solution wouldn''t converge')
    end
end
```

invkin.m

Kreutz-Delgado and Agahi [42] implemented a similar inverse kinematics routine using a recursive algorithm. However, their algorithm required a knowledge of the mass and inertial properties of the manipulator links, which was not available for this body of work. Kline et al. [43] investigated the rate of convergence of iterative inverse kinematics methods, including Jacobian control.

3.5.2 Singularity of Inverse Kinematics Routine

Wang and Chen [44] stated that the major difficulty with the iterative method discussed above is that they do not converge on a solution when the Jacobian matrix is singular. The above iterative inverse kinematics solution utilized in this work would not converge on a solution at or near a kinematic singularity, as predicted by Wang and Chen.

Again, if no solution can be determined for a particular manipulator pose, that configuration is said to be singular. At a singularity, the Jacobian matrix (**J**) becomes rank deficient (e.g., loses rank), and the matrix-algebra inversion of the Jacobian matrix

to find the associated velocity required to follow a particular trajectory path may result in arbitrarily large joint velocities ($\dot{q}$) and accelerations ($\ddot{q}$) [45].

If the rows of the square Jacobian matrix of order n are linearly independent, then the determinant of the Jacobian is nonzero, and the matrix is said to be non-singular. If the determinate of the square Jacobian matrix of order n is zero, then the Jacobian is singular and the rows of the matrix are not linearly independent [30, p. 541]. Hence, when the robot Jacobian is singular, the robot pose is said to be kinematically singular.

3.6 Trajectory Planning

Several works have been dedicated to trajectory planning of general robotic manipulators as well as kinematically redundant manipulators. For example, Janabi-Sharifi and Wilson [46] explored methods for end-effector trajectory planning that not only incorporated singularity avoidance but also established collision avoidance.

For the simulation of the fuel manufacturing process, a simplified methodology for generating joint angles for the TELBOT was required. For our application, the robot operating environment is well established and modeled in CAD, and therefore, the trajectory planning did not have to incorporate any obstacle avoidance. Any necessary obstacle avoidance was established by the selection knot points in Cartesian space that did not allow the end-effector to come in contact, or the manipulator to collide, with unintended objects. Therefore the following discussion deals with the formulization of the trajectory-planning scheme used in the obstacle-free motion.

As the Cartesian space knot points are established in the CAD model of the simulated process, a method is required for generating smooth joint-angle trajectories for all six robot joints as the end effector passes through these CAD generated points in space.

Typically, trajectory-planning schemes approximate or interpolate the desired end effector path. Path endpoints and intermediate points are specified in Cartesian space, as it is intuitive with the orientation and attitude of the end-effector with respect to the modeled environment.

For each of the knot points, the end-effector position and attitude are established and the corresponding six robot joint positions are derived using the inverse kinematics routine, as discussed in Section 3.5. This process shifts the trajectory planning from the Cartesian variable space to the joint-variable space. As Fu et al. [30, p. 152] explains, planning in the joint-variable space has three advantages; 1) the trajectory is planned directly in terms of the controlled variables during the motion, i.e., the joints themselves; 2) the trajectory planning can be done in near real time, and 3) the joint trajectories are easier to plan. Fu et al. [30, p. 150] suggests a "systematic approach to the trajectory planning problem is to view the trajectory planner as a black box" where the various process inputs are provided and the trajectory planner creates a sequence of time-based intermediate configurations for each of the robot joints. A software script was written in MATLAB to perform this "black-box" operation of joint trajectory, as is shown in Appendix A.8. Implementation of this script is discussed further in Section 4.1.2.4.

While there exists numerous trajectory-planning routines, a simple but smooth method is preferred for our application. Known knot points provide explicit constraints on the path, where a parameterized trajectory can be created by interpolating between knot points. In order to ensure smoothness of movement for the end-effector, constrains must be placed on each joint's velocity and acceleration for each knot point. As discussed above, the inverse kinematics routine determines the desired angle for each

robot joint at each of the knot points. The inverse kinematics is essentially used to map from the Cartesian-space knot points to the joint-variable space, based on the geometry and configuration of the robot.

The trajectory generator function, file name traj434.m, accepts process variables and outputs time-based joint configurations. The function accepts two variables, **Q** and **T**.

```
function [ts,h]=traj434(Q,T)
```

traj8.m

The variable **Q** is a 4×6 matrix containing four knot points for each of the six robot joints generated through the inverse kinematics routine. A 1×4 array of process time stamps for each of the four knot points is passed to the function with the **T** variable. Returned from the function call is a $n\times1$ array of time stamps (ts) and an $n\times6$ matrix of joint angles (h) for each of the time stamps, where n is the number of time steps based on the input time stamps **T** so that:

$$ts = \frac{\mathbf{T}}{ss} \tag{15}$$

Global variables ss, **v**, and **a** are used to establish the process time step interval, maximum joint velocities, and maximum joint accelerations, respectively. The step interval (ss) is the discretized time interval used throughout the trajectory generation program. The step interval is established in traj8.m as 0.02 seconds, which results in large data sets for joint angles, but produces smooth simulations within MSC.visualNastran 4D (vN4D), see Chapter 4. The joint velocities (**v**) and accelerations (**a**) are established in the data file velacceldata.m and are 4×6 matrices for the four knot points for each of the six joints.

Following the methodology from Fu et al. [30, pp. 154 – 165], the joint-interpolated trajectory generator is written in MATLAB. The methodology is a 4-3-4 Trajectory, where each joint has three trajectory segments. The first trajectory is a 4^{th}-order polynomial specifying the discrete joint angles from the initial position to the lift-off position. The second trajectory is a 3^{rd}-order polynomial generating the discrete joint angles from the lift-off position to the set-down position. The third trajectory, following suite, is a 4^{th}-order polynomial generating the discrete joint angles from the set-down position to the final position. The equations for the trajectory generation $h_i(t)$ for each of the three polynomial equations, expressed in normalized time (t), are:

$$h_1(t) = a_{14}t^4 + a_{13}t^3 + a_{12}t^2 + a_{11}t + a_{10} \tag{16}$$

$$h_2(t) = a_{23}t^3 + a_{22}t^2 + a_{21}t + a_{20} \tag{17}$$

$$h_3(t) = a_{34}t^4 + a_{33}t^3 + a_{32}t^2 + a_{31}t + a_{30} \tag{18}$$

As discussed in Equation 15 above, the number of discrete steps (n) for each joint with a 4-3-4 trajectory is dependent on the passed process time-stamp array (T), and therefore, the time required for a joint movement is variable.

In equations 16 through 18 above, the unknown coefficient a_{ij} represents the i^{th} trajectory segment for the j^{th} coefficient of a particular joint. The coefficient a_{ij} is solved for all six joints of the robot. Fu et al. [30] suggests using a normalized time variable, where $t \in [0,1]$, which allows us to utilize the same trajectory generation equations regardless of the start-time or the finish-time for each movement.

For the desired smooth motion of each robot joint, the polynomial segments for the trajectory must be continuous in position (θ), velocity (v), and acceleration (a) at the midpoints (θ_1, v_1, a_1), e.g., lift-off position and set-down position. It is intuitive that the

velocity and acceleration at the initial position (v_0, a_0) and the final position (v_f, a_f) must be zero. Thus, taking the first and second derivatives with respect to normalized time (t) of Equations 16 through 18 above, one achieves the continuity equations at the two midpoints [30]:

$$\frac{\dot{h}_2(0)}{t_2} = \frac{\dot{h}_1(1)}{t_1} \tag{19}$$

$$\frac{\ddot{h}_2(0)}{t_2^2} = \frac{\ddot{h}_1(1)}{t_1^2} \tag{20}$$

The determination of each of the polynomial variables for Equations 16 through 18 above involves solving for some of the coefficients explicitly by using the established boundary conditions, while solving for the remaining coefficients simultaneously.

The explicit solution of the coefficients based on the boundary conditions are [30]:

$$a_{10} = \theta_0 \tag{21}$$

$$a_{11} = v_0 t_1 \tag{22}$$

$$a_{12} = \frac{a_0 t_1^2}{2} \tag{23}$$

$$a_{21} = v_1 t_2 \tag{24}$$

$$a_{22} = \frac{a_1 t_2^2}{2} \tag{25}$$

$$a_{n1} = v_f t_n \tag{26}$$

$$a_{n2} = \frac{a_f t_n^2}{2} \tag{27}$$

The solution in the MATLAB program utilized Equations 21 through 27 to solve for the polynomial coefficients. For example, the solution for a_{i2} shown in equations 23, 25,

and 27 is computed for each of the joints passed in the matrix **Q**, as shown in the following excerpt from the traj434.m MATLAB file.

```
[sr,sc]=size(Q); %Size of rows and size of columns
for i=1:sc
    a12(i)=(a(1,i)*t1^2)/2;
    a22(i)=(a(2,i)*t2^2)/2;
    a32(i)=(a(4,i)*t3^2)/2;
end
```

traj434.m

Following Fu et al. [30], the simultaneous solution for the remanding coefficients is as follows. :

$$\mathbf{y} = \mathbf{C}\mathbf{x} \tag{28}$$

The difference of joint angles (δ_n) between successive trajectory segments is simplified as:

$$\delta_1 = \theta_2 - \theta_1 \tag{29}$$

$$\delta_2 = \theta_3 - \theta_2 \tag{30}$$

$$\delta_3 = \theta_4 - \theta_3 \tag{31}$$

So the unknown coefficients of the trajectory polynomial can be determined as:

$$\mathbf{y} = \begin{bmatrix} \delta_1 - \frac{a_0 t_1^2}{2} - v_0 t_1 \\ -a_0 t_1 - v_0 \\ -a_0 \\ \delta_2 \\ -a_f t_n + v_f \\ a_f \\ \delta_n + \frac{a_f t_n^2}{2} - v_f t_n \end{bmatrix} \tag{32}$$

$$\mathbf{C}=\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 3/t_1 & 4/t_1 & -1/t_2 & 0 & 0 & 0 & 0 \\ 6/t_1^2 & 12/t_1^2 & 0 & -2/t_2^2 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1/t_2 & 2/t_2 & 3/t_2 & -3/t_n & 4/t_n \\ 0 & 0 & 0 & 2/t_2^2 & 6/t_2^2 & 6/t_n^2 & -12/t_n^2 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \tag{33}$$

$$\mathbf{x}=\begin{bmatrix} a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{n3} \\ a_{n4} \end{bmatrix} \tag{34}$$

Solving for **x** results in the simultaneous solution of the remanding coefficients.

$$x = C^{-1}y \tag{35}$$

The solution in the trajectory generation function (traj434.m) utilized equations 28 through 35 to solve for the remaining polynomial coefficients. An example of the solution for each of the six robot joints is as follows:

```
for i=1:sc,
    y(1,i)=del1(i)-a12(i)-a11(i);
    y(2,i)=-1*a(1,i)*t1-v(1,i);
    y(3,i)=-1*a(1,i);
    y(4,i)=del2(i);
    y(5,i)=-1*a(4,i)*t3+v(4,i);
    y(6,i)=a(4,i);
    y(7,i)=del3(i)+a32(i)-a31(i);
end
C=[ 1 1 0 0 0 0 0
    3/t1 4/t1 -1/t2 0 0 0 0
    6/t1^2 12/t1^2 0 -2/t2^2 0 0 0
    0 0 1 1 1 0 0
    0 0 1/t2 2/t2 3/t2 -3/t3 4/t3
    0 0 0 2/t2^2 6/t2^2 6/t3^2 -12/t3^2
    0 0 0 0 0 1 -1];
for i=1:sc,
    yy=y(:,i);
    x(:,i)=inv(C)*yy;
end
```

traj434.m

The inverse of C always exits if the time intervals are positive values. Since the time for each of the trajectory segments was normalized to run from [0, 1], it is assured that the inverse of C always meets the positive time interval requirement.

Now that all of the coefficients of the trajectory polynomial equations are solved, the coefficients can be substituted back into Equations 16 through 18, above.

An example of the smooth trajectories generated by the MATLAB code can be seen in Figure 14 for each of the six joints over the course of four 4-3-4 trajectories. Each of the six graphs show the scaled joint angles verses time. Particular attention must be made to the scale of the individual graph, as Joint 4 has relatively small motions throughout the trajectory path.

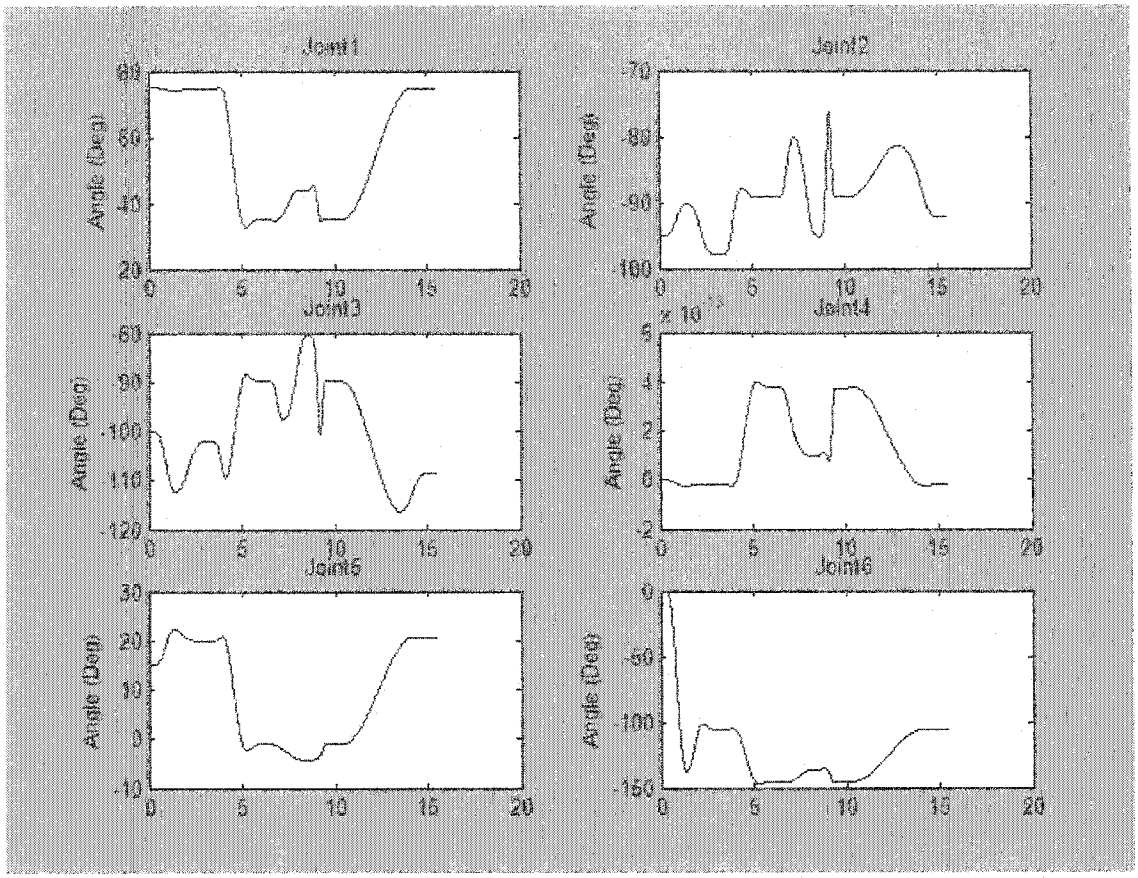


Figure 14. Sample Joint Trajectory Generation

Following the Fu et al. [30] methodology, the constraint to achieve absolute position congruency, i.e., the interpolating polynomial must pass thought the position exactly, is relaxed so that velocity and acceleration can be held constant.

CHAPTER 4

SOFTWARE DESIGN AND INTEGRATION

Nof [31, p. 20] provided a general overview of the historical progress of the robotics industry, progressing from first to second and then to third generation robots. As Nof explains, first generation robots utilize little, if any, computer power to adaptively control the robot. First generation robot intelligence resides in their ability to reproduce "taught" movements or sequences. Several different sequences may be taught, stored, and called up as required by some external sensory input or as a result of a conditional test. The programming of these sequences has traditionally been implemented by using a "teach-box" for first- and most second-generation robot systems. This method of on-line programming is very attractive, being readily learned by factory personnel who are not trained software specialists. [31]

Nof explains [31] that second-generation robotic systems utilize computer control more effectively. Real time control for the robot is possible, where the calculations required to control the motions of each degree of freedom in a cooperative manner to effect smooth motions of the end-effector along predetermined paths — for example, along a straight line in space. Operations by these robots on work pieces in motion along an assembly line could be accommodated. Some simple sensors, such as force, torque, and proximity, could be integrated into the robot system, providing some degree of adaptability to the robot's environment. Major applications of second-generation robots

include spot welding, paint spraying, arc welding and some assembly. Which are all operations that are part of an automated manufacturing process.

Third generation robot systems [31] typically incorporate multiple computer processors, each operating asynchronously to perform specific functions. A typical third-generation robot system includes a separate low-lever processor for each degree of freedom, and a master computer supervising and coordinating these processors and providing a higher-level function. Often referred to a Digital Control and Management Information System, the master computer is able to control the process as well as collect plant data for purchasing, quality control, inventory control, just-in-time manufacturing, among others.

4.1 Model Development

The following section discusses the application of the first and second generation robot philosophies to the simulation process. Initially, a first generation robot was modeled, taught, and simulated as a method to learn the related simulation software packages. Later, a second generation robot was simulated utilizing more sophisticated software interfaces and robot generation schemes, as is discussed in Chapter 3, *Robot Modeling*, and Section 4.1.2, *MATLAB Joint Trajectory Generation.*

The simulation of the robotic environment utilized four different software packages, which include [1]Pro/ENGINEER®, [2]MATLAB®, [3]Excel®, and [4]MSC.visualNastran 4D® (vN4D). A simplified process flow of the robot path planning software integration is shown in Figure 15.

[1] Pro/ENGINEER is a registered trademark of the Parametric Technology Corporation.
[2] MATLAB is a registered trademark of The MathWorks, Inc.
[3] Excel is a registered trademark of the Microsoft Corporation.
[4] MSC.visualNastran 4D (vN4D) is a registered trademark of the MSC.Software Corporation.

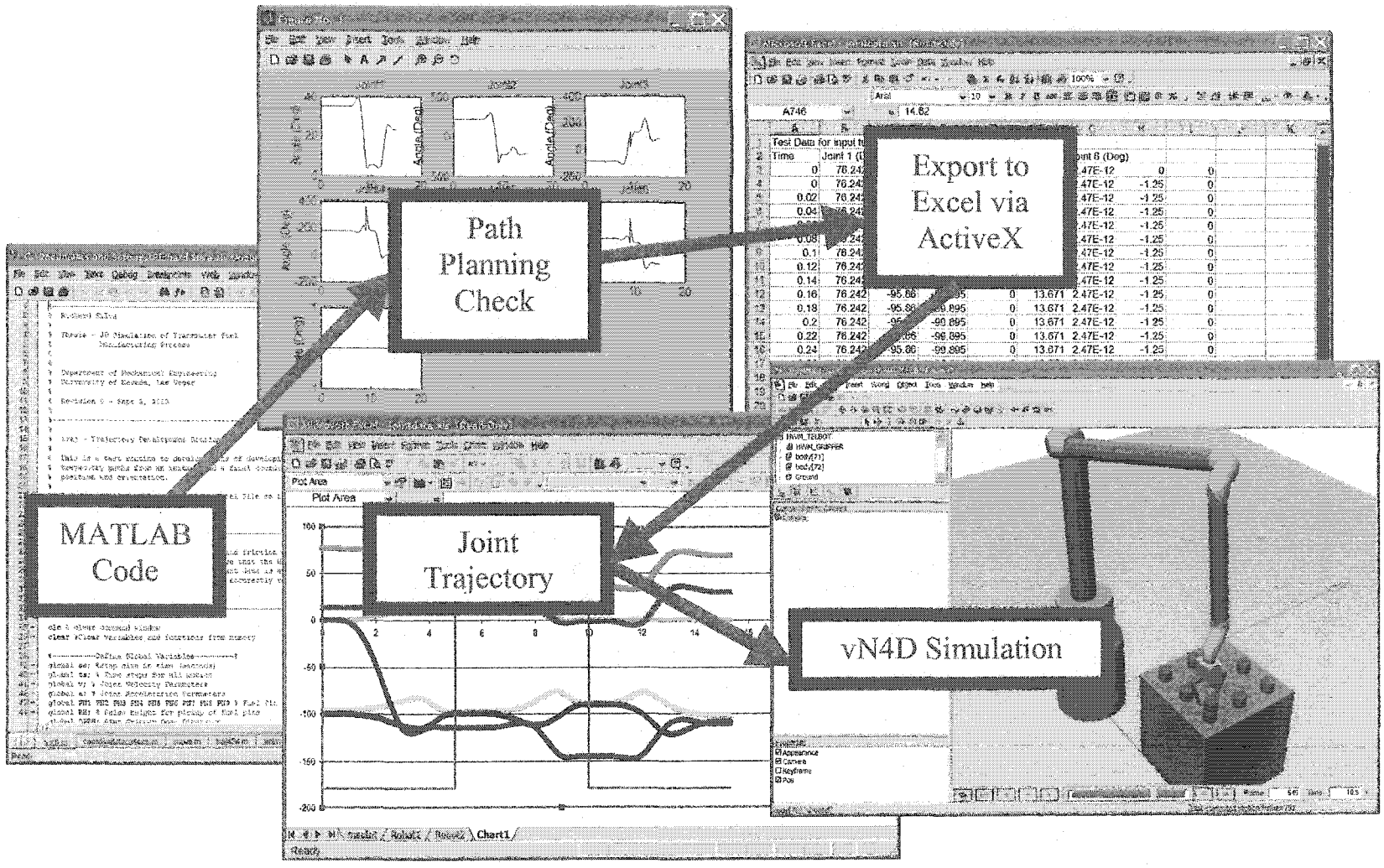


Figure 15. Robot Path Planning Software Integration Process Flow

The student version of Pro/ENGINEER was utilized to develop the base three-dimensional (3-D) models of the workspace and robot manipulators. Pro/ENGINEER models were imported into vN4D environment for simulation. While Pro/Engineer renders the robot image with excellent quality, vN4D does provide renderings of the simulation environment, as shown in Figure 16.

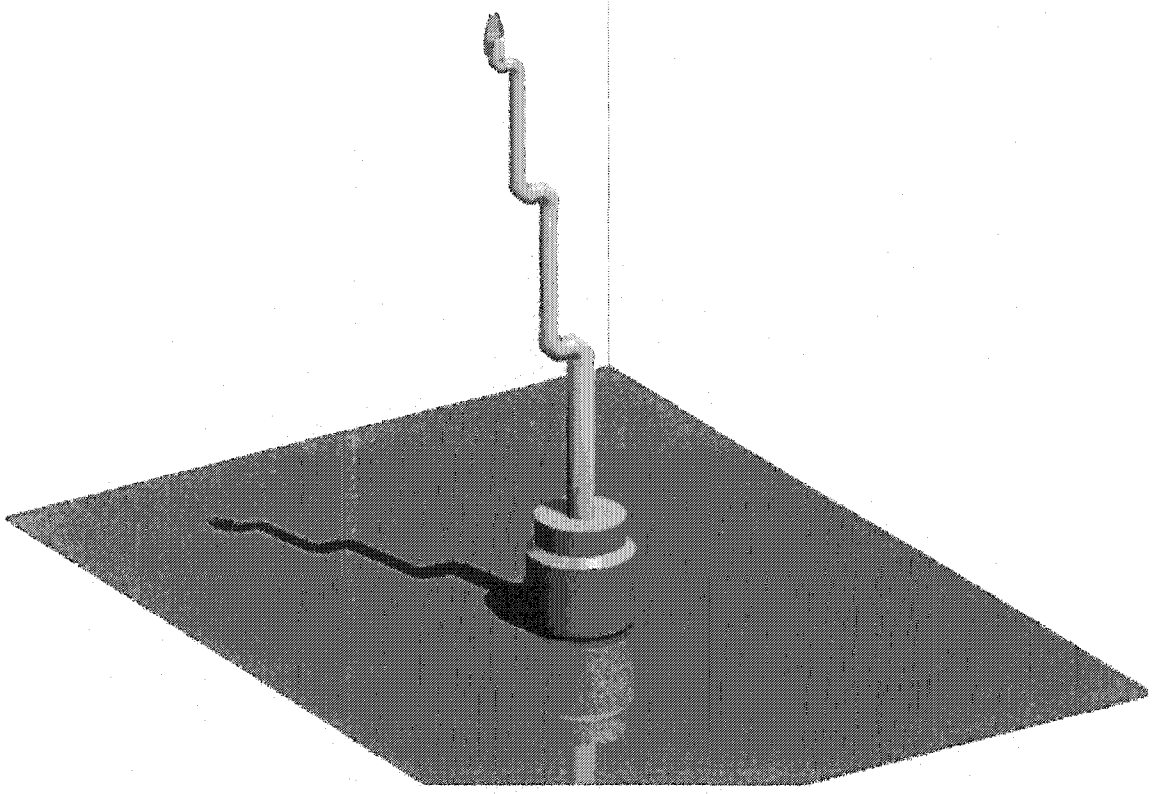


Figure 16. Generalized TELBOT 3-D Model

Among other functions, vN4D is a time-based simulation package that is able to import the native Pro/Engineer files. Once established in vN4D, the 3-D models can be moved and animated. Special attention must be made in the development of the robot model within Pro/ENGINEER so that the model behaves in an expected and predictable

manner within vN4D. Once imported, all robot joints including the grippers must be verified in vN4D to validate that the joint location and orientation is logical, the joint moves in the expected direction (i.e., right-hand-rule), and that the limits to the movement are established. This task, at face value, seems straight forward in theory; however, application was difficult and non-intuitive as each software package contained nuances and idiosyncrasies that were not readily apparent nor described within either software package documentation. In addition, a documented or established procedure for importing 3-D models does not exist.

Early implementation of the vN4D software proved difficult, where numerous iterations were made between Pro/ENGINEER and vN4D to ensure successful transfer of the 3-D model. The first fully successful transfer of the 3-D model is shown in Figure 17, where the robot model is set so that all joints are zeroed, i.e., the robot stands vertically. Then in simulation, the robot was allowed to free-fall under the force of gravity without any joint torque being applied. The resulting simulation of a collapsing robot was a major milestone in this work, where the following occurred:

1) Successful import of the 3-D robot model from Pro/ENGINEER to vN4D.
2) All robot joints were accurate and modeled correctly.
3) Robot members would interfere with each other and would impact the floor, showing that all components were correctly identified within the vN4D software.
4) Camera movements for recording and visualizing the simulation were correctly controlled.

5) Video of the simulation (*.avi) could be produced for export into presentations, etc.

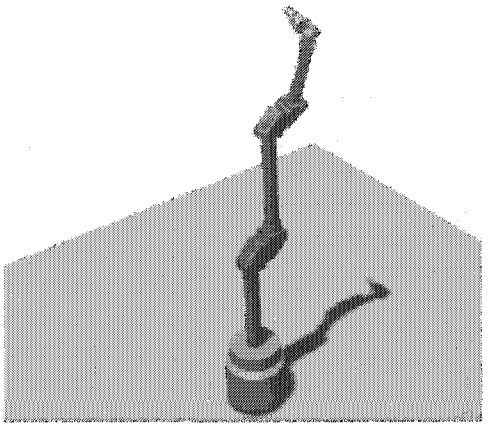


Figure 17. Screen Capture of Animation Showing Initial Robot Import Success

4.1.1 Robot Teaching

Once a model of the robot was imported and simulated correctly in the vN4D software, a simple simulation of the robot was needed. This was accomplished by manually moving the robot to desired positions within the vN4D software, i.e., "teaching". The robot was moved to the desired pose within vN4D and the joint angles were extracted and placed in an Excel spreadsheet. This was done for several manipulator poses and gripper configurations. Once all joint angles for each robot pose were extracted to Excel, a linear interpolation was performed between each pose.

A simple algorithm was utilized in Excel to perform the linear interpolation of each joint angle with respect to time. The algorithm performs the linear interpolation between the initial joint angle ($\theta_{i,j}$) at time (t_i) and the final joint angle ($\theta_{f,j}$) at time (t_f). This is performed for each joint (θ_j) at the discrete time step (t). Therefore, the resulting Excel

spreadsheet is quite large, with over 500 rows of joint data for a simple move. Symbolically, the generalized linear interpolation is shown in Equation 36

$$\theta_{t,j} = \frac{\theta_{f,j} - \theta_{i,j}}{t_f - t_i} \cdot (t - t_i) + \theta_{i,j} \tag{36}$$

While this methodology provided a means for simple movement of the robot within the simulation environment, the inherent nature of the linear interpolation resulted in jerky movements.

Also, all work was performed in joint-variable space. The end effector location was never precisely known in Cartesian-variable space, which is more intuitive for movement of objects within a work cell.

Finally, this methodology lacked the flexibility for changes in the work cell, to the trajectory, or to the knot points. For each change, the Excel spreadsheet had to be reconfigured, mathematical equations placed in the respective cells, and rerun. These manual operations often resulted in errors. Such an error can be seen in the latter movements of the manipulator shown in Figure 18, below, where a spike in the joint angles occurred.

Figure 18 shows the joint trajectories for each of the six joints created using the robot teaching methods. The straight-line segments show the linear interpolation between knot points. The spikes in the joint trajectories occurring near time step 450 where intentionally left in the figure to show the susceptibility of error when utilizing the robot teaching methodology. This probability of error provides a justification for implementing a more automated method for generating joint trajectories.

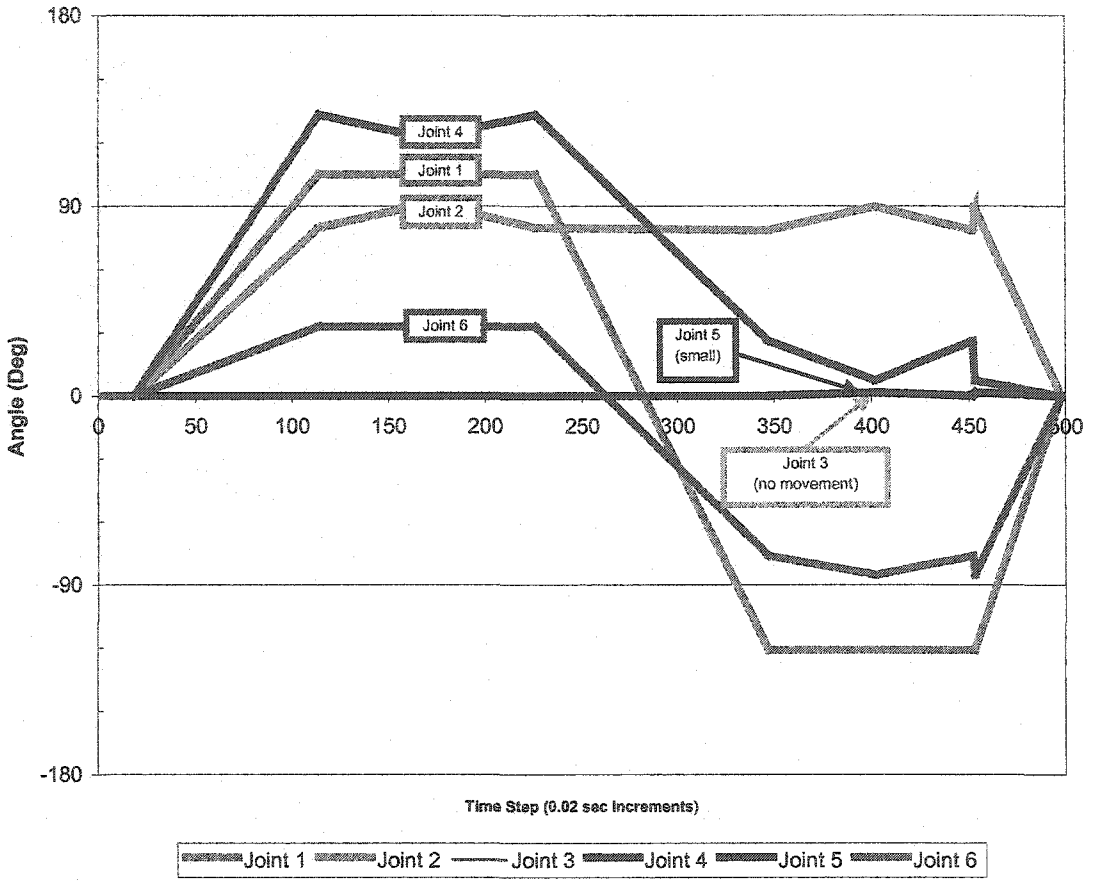


Figure 18 Linear Interpolated Joint Trajectory (with Errors)

4.1.2 MATLAB Joint Trajectory Generation

It would be advantageous to have a method for generating joint trajectories that were in a more intuitive Cartesian variable space, that were easy to change and modify, would produce smooth joint-variable trajectories, and be less susceptible to error in the joint trajectory generation methodology. MATLAB was utilized by writing a primary script with several sub-functions/scripts that would produce a trajectory. The trajectory would be created ultimately utilizing high-level commands, such as "move", "wait", and "gripper=open" commands.

4.1.2.1 Background of MATLAB

MATLAB is the software-programming environment in which the mathematical version of the robot model was created, its joint angles calculated, and the resultant time-based joint data passed to Excel. As discussed above, Excel is the tool that is interfaced directly by vN4D. vN4D opens an Excel spreadsheet linked to the model for all of the movements of the robot. A primary focus of the following text is how to get the following software codes to communicate for use in the simulation.

4.1.2.2 Object Oriented Programming in MATLAB

The MATLAB functionality includes object oriented programming, which is a method of software programming that utilizes "objects" and "classes". While this text is not intended to be a tutorial of object oriented programming, a brief description of the methodology is provided for the reader new to this subject.

A "class" is a data type that, encapsulated within, contains various other data types, functions, and other behaviors. For example, a class known as a "robot class" could 1) be established to encapsulate the various parameters of a robot; 2), redefine the addition (+) function with one that can add serial links of a robot; and 3) redefine MATLAB's plot functions to show a three-dimensional representation of the particular robot instance. This very robot class was developed by Corke [36] to facilitate the use of functions that are useful in robotics including kinematics, dynamics, and trajectory generation. Corke's methods for robot class and inverse kinematics were used as guidance in developing the forward and inverse kinematics routines discussed in Chapter 3-*Robot Modeling*.

Each time a class is invoked, a new "instance" of the class is created, known as an object. One creates an instance of a particular object by calling the embedded class constructor and passing the appropriate variables. The class constructor creates the

object by initializing the data structure and instantiating an object of the class [47, Chapter 21].

4.1.2.3 Robot Class

To develop a function within MATLAB to automatically generate joint angles and trajectories for a particular manipulator, a mathematical representation of the particular robot must exist. For this work, a mathematical robot class was created using object oriented programming techniques, so that each time a robot was needed, an instance of the robot class could be generated.

Within the base trajectory generation file traj8.m, a call to TELBOT function (telbot.m) is made to create an instance of the robot manipulator.

```
tb=telbot;    % Load Robot Class Code
```

traj8.m

Once the instanced of the robot is created, a child class is created to allow parameters within the robot instance to be changed so that the now augmented robot class can handle gripper configurations and a fuel pin numbering sequences.

```
    %-- Set Child Class Properties --%
    tb.gripper='open';
    tb.pin=0;
```

telbot.m

As the simulation within vN4D does not handle part collisions well, i.e., a gripper engaging a fuel pin, a separate methodology had to be implemented to adequately show within the simulation environment the TELBOT manipulator gripping a fuel pin, lifting and moving the pin to a new location, and subsequently un-engaging the pin. The method to accomplish pin-pickup was to set an arbitrary time-based constraint between one of the gripper fingers and the fuel pin.

Within the MATLAB code, a pin-variable sets a Boolean identifier in vN4D so that the MATLAB code is able to turn on-and-off the pin constraint within the simulation. To accomplish the time-based constraint, the robot class was augmented with a "child class" to set two variables, *gripper* and *pin*. The gripper variable would generate time-based gripper movements for the open and closed positions. The pin variable would set the active pin constraint; where *pin*=0 turns off all pin constrains and a non-zero setting would activate the respective pin constraint, and therefore, simulate the gripper engaging the pin.

4.1.2.4 Trajectory Generator Script

As mentioned in Section 4.1.2, it would be advantageous to have a method for generating joint trajectories where the implementation is intuitive, modifiable, scalable and effective. Within MATLAB, a trajectory generator script was created (traj8.m) to provide a user with a high-level interface to create robot trajectories. While not a sophisticated as graphical user interface, the purpose of the script was to ensure that all of the low-level functions were called from the main script and were robust enough to handle all reasonable trajectories. A generalized software flow diagram for the *traj8.m* script is shown in Figure 19, which depicts the major process steps and calls to other functions used to generate trajectories for the simulation.

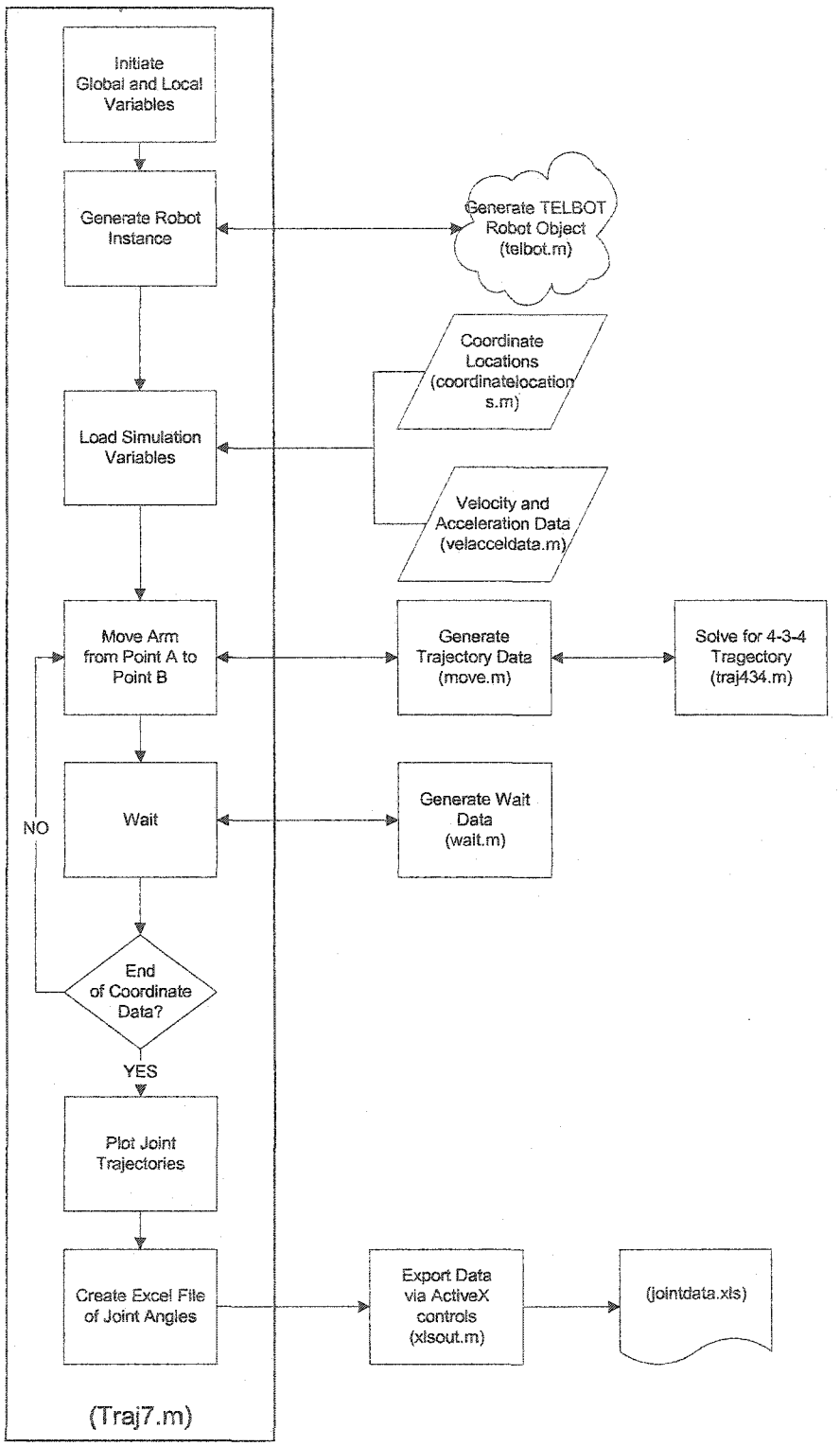


Figure 19. Generalized Software Flow Diagram

The script first initializes all local and global variables, where the global variables are used throughout all scripts. The use of global variables simplifies the passing of parameters from one function or script to another. Examples of global variables include discrete timing variables, joint velocity and accelerations, fuel pin locations and gripper positions. Local variables are established within each function and are unique to the particular function. This allows the same variable name to be used across various functions, while remaining independent of each other.

Once the local and global variables are established, then the robot instance is created, as discussed in Section 4.1.2.3. Variables specific to the simulation environment are then loaded from the velacceldata.m and coordinatelocations.m files. The velacceldata.m file contains the specific velocities and accelerations for each of the six manipulator joints used in the 4-3-4 trajectory generation function, traj434.m. The coordinatelocations.m file contains all key global coordinates, and will be discussed further below.

All parameters, variables, and robot instances are created at this point. The trajectory generator is now able to accept move (move.m) and wait (wait.m) commands. If one wanted to move the manipulator by creating a trajectory from one point to another, then the move command could be called. By passing command variables consistent with the 4-3-4 trajectory methodology discussed in Section 3.6, the move.m function calls the necessary inverse kinematics function on the passed robot instance for each of the passed knot points. Once the joint angles are solved for each knot point, the function then calls the traj434.m function, which calculates the subsequent discrete joint angles for the desired movement.

```
T=[6 8 9 10];
nd = move(tb,nd,T,PN1+RH,PN9+PH+RH,PN9+PH);
```
traj8.m

In the above code, the move.m script is called. The robot instance (**tb**) is passed with the data array of all previous joint angles, gripper positions, and pin sets (**nd**). A 1×4 time vector (**T**) is established, where each of the time variables coincide with the 4-3-4 trajectory knot points. Finally, the desired knot points are passed. The initial knot point ($\mathbf{Q}_1$) is taken from the last position in the data array (**nd**) of all the previous joint angles.

```
[sr,sc]=size(nd);
```
move.m

The MATLAB function *size* evaluates the *M*-by-*N* matrix **nd**, returns a two-element row vector [sr, sc] containing the number of rows and columns in the matrix. The last position of the **nd** matrix used for the initial knot point $\mathbf{Q}_1$ is found by:

```
Q(1,:)=[ nd(sr,2) nd(sr,3) nd(sr,4) nd(sr,5) nd(sr,6)
nd(sr,7)];
```
move.m

The second through knot points ($\mathbf{Q}_2$,$\mathbf{Q}_3$,$\mathbf{Q}_4$) of the 4-3-4 trajectory are pin locations stored in the world coordinate data file (coordinatelocations.m) and used simply by the passing the pin location variable name, e.g., the pin locations (**PNx**). For mid-movement knot points as used in the move command above, one can add to the knot points the fuel pin height (**PH**), and the specified raise height (**RH**).

The raise height is simply a z-axis translation of the pin location, allowing the manipulator end effector approach a fuel pin from a height just above the pin. This ensures that the approach and departure created by the 4-3-4 trajectory function (traj434.m) is relatively vertical, i.e., along the z-axis of the base coordinate system.

Each of the key coordinate location variables is actually 4×4 homogeneous transformation matrix (${}^{base}T_{pin}$) transforming the orthonormal coordinate frame at the base of the robot to the pin location. The transformation matrix is further defined in Section 3.4, specifically Equation 3.

For example, the first pin location is the homogeneous transformation matrix:

```
PN1=    [1  0 0 28.126
         0 -1 0 19.874
         0  0 1 25.204
         0  0 0 1];
```

coordinatelocations.m

and the pin height and raise height transformation matrices are:

```
raiseheight= 8; %in the z direction
pinheight = (1.5*2.6);
RH  =       [0 0 0 0
             0 0 0 0
             0 0 0 raiseheight
             0 0 0 0];

PH  =       [0 0 0 0
             0 0 0 0
             0 0 0 pinheight
             0 0 0 0];
```

coordinatelocations.m

The use of the transformation matrix allows for the transparent implementation of the forward and inverse kinematic routines, summarized in Sections 3.4 and 3.5, respectively. In addition, the pin height and raise height can modify knot points for various movements used in the *move* function call.

```
nd = move(tb,nd,T,PN1+RH,PN9+PH+RH,PN9+PH);
```

traj8.m

The wait function (wait.m) is a high level function that holds the manipulator in a particular pose for a specific duration. The wait command is useful once the manipulator

end-effector has reached a desired Cartesian coordinate, and the gripper open/close or pin active commands can be called.

For example:

```
%Activate the Gripper on Pin1
T=[5.75 6 0 0];
tb.pin=1; % Pin 1 is now active
nd=wait(tb,nd,T);
```

traj8.m

At time 5.75 seconds, the pin constraint is applied for fuel pin 1, and the 3-D simulation is given until the time of 6 seconds to implement the constraint. It should be noted, that setting the active pin (`tb.pin=1`) actually changes the internal variables for the child-class of the TELBOT instance (**tb**), an object-oriented programming method. Therefore, when the wait function is called and is passed the TELBOT instance, the newly changed internal variables are inherently passed as part of the "package".

Once all move and wait commands are called, then the traj8.m script plots the trajectory for each of the six manipulator joints, as well as, the gripper. As can be seen in Figure 14, the resulting joint trajectory is smooth over the range of motion. The plots show all move and wait commands with respect to time. The intent of plotting the joint trajectories is to serve one primary purpose; that is, to allow the user a preview of the robot motion prior to writing the data to the Excel file. As the manipulator trajectories are solved, there exists the possibility that the manipulator may reach a location at or near a singularity. These singularities will become inherently clear as they are readily observable on the plot.

4.1.2.5 MATLAB to Microsoft Excel

Within the software development community, the difficulty of using diverse and dissimilar software packages together for transferring data and information has lead to the

development of Component Object Model (COM) objects. These object-oriented programming COM objects allow software programs to utilize other software packages and related data, regardless of who manufactured the software or even what compilers they used. Embedded within Object Linking and Embedding (OLE) compliant software is a suite of COM objects that allow software developers to integrate application-specific components from different vendors into their own applications.

The COM objects are used within the MATLAB code to open the Excel spreadsheet that is interfaced with by the simulation software (vN4D), flip to the correct tab within the spreadsheet, and place all of the time steps and joint trajectories within the spreadsheet.

The COM objects are implemented using ActiveX controls. ActiveX is not a programming language, but rather a set of rules for how applications should share information. Programmers can develop ActiveX controls in a variety of languages, including, but not limited to, C, C++, Visual Basic, and Java. ActiveX controls have access to the Microsoft Windows operating system, unlike its Java cousin. MATLAB supports the use of ActiveX controls, albeit less powerful than Java, C++ or Visual Basic. Of lesser importance, ActiveX is an offspring-development of two other Microsoft technologies; OLE and COM.

Once all moves within the trajectory generation have been solved and the resultant joint trajectories are plotted, the traj8.m script asks the user whether the data should be exported to Excel. Assuming the user is satisfied with the trajectory results shown on the plot, a series of commands are called with xlsout.m to activate the ActiveX controls and transfer the data to Excel.

```
%--------------------------------------%
%--- Create Excel File information ----%
%--------------------------------------%
Choice=menu('Do you want to create the Excel file of the
Data?','Yes','No');
if Choice == 1;
    Header='Test Data for input to Excel';
    Col={'Time','Joint 1 (Deg)','Joint 2 (Deg)','Joint 3
(Deg)','Joint 4 (Deg)','Joint 5 (Deg)','Joint 6 (Deg)'};
    Filename='C:\Documents and Settings\Richard Silva\My
Documents\thesis\aaa\waelishmiller2\jointdata.xls';
    tab=3; %Place the data on the Tab 3
%--- Write Data to Excel File ---%
xlsout(nd_deg,Header,Col,Filename,tab);
else
    disp('**********************************')
    disp('* DATA NOT WRITTEN TO EXCEL FILE *')
    disp('**********************************')
end
```

traj8.m

Passed to the xlsout.m function is the joint data array (**nd**), spreadsheet header information, an array of column names, the filename of the Excel file, and finally the spreadsheet tab for which the data will be placed.

Shown in Figure 20 is the generalized software flow diagram showing the major process steps and function calls to export the trajectory data to Excel. First, an instance of Excel must be initiated, which is performed using an ActiveX control, as shown:

```
Excel = actxserver('Excel.Application');
```

xlsout.m

A successful Excel instance will return a handle to that instance. A "handle" is a software term that gives a link to the instance so that the particular instance can be manipulated. Technically, a handle is simply a pointer to a pointer, where a pointer is the address of another piece of data. Using handles and pointers help protect the actual memory location from being inadvertently altered, e.g., corrupted.

The handle of the Excel instance is stored in the *ExcelWorkBook* variable and simply obtained by looking within the Excel instance:

```
ExcelWorkBook=Excel.Workbooks;
```
xlsout.m

Once the handle is obtained, then the specific passed file name is opened using

```
Workbook=invoke(ExcelWorkBook,'Open',filename);
```
xlsout.m

Now that the Excel file is open, the active sheet of the spreadsheet is selected and subsequently, the handle for the particular sheet obtained. If no sheet is passed when the xlsout.m function is called, the system will open sheet one by default.

```
Tabs=Excel.ActiveWorkBook.Sheets;
if nargin<5
    sheet=get(Tabs, 'Item', 1);
else
    sheet=get(Tabs, 'Item', tab);
end
invoke(sheet,'Activate');
```
xlsout.m

Obtaining the sheet handle is performed in similar manner to obtaining the handle to Excel instance described above

```
Activesheet = Excel.Activesheet;
```
xlsout.m

Now that the active sheet is selected, the process of writing the data to the sheet is performed. While more intense, the passed joint variable data (**nd**) is stored in the local variable (**m**) within the xlsout.m function. A significant portion of the code determines the number of rows and columns to use in the Excel file, and labeling the columns of data. The primary line in which the joint variable data is placed in the Excel file is:

```
set(ActivesheetRange, 'Value', m);
```
xlsout.m

The remainder of the xlsout.m function is to save the Excel file, close the file, and close the ActiveX controls. Saving of the file and closing Excel is performed by:

```
    invoke(Workbook, 'Save');
    invoke(Excel, 'Quit');
```

xlsout.m

And closing the ActiveX controls is done with

```
%Delete the ActiveX object
%disp('Delete the ActiveX object')
delete(Excel);
```

xlsout.m

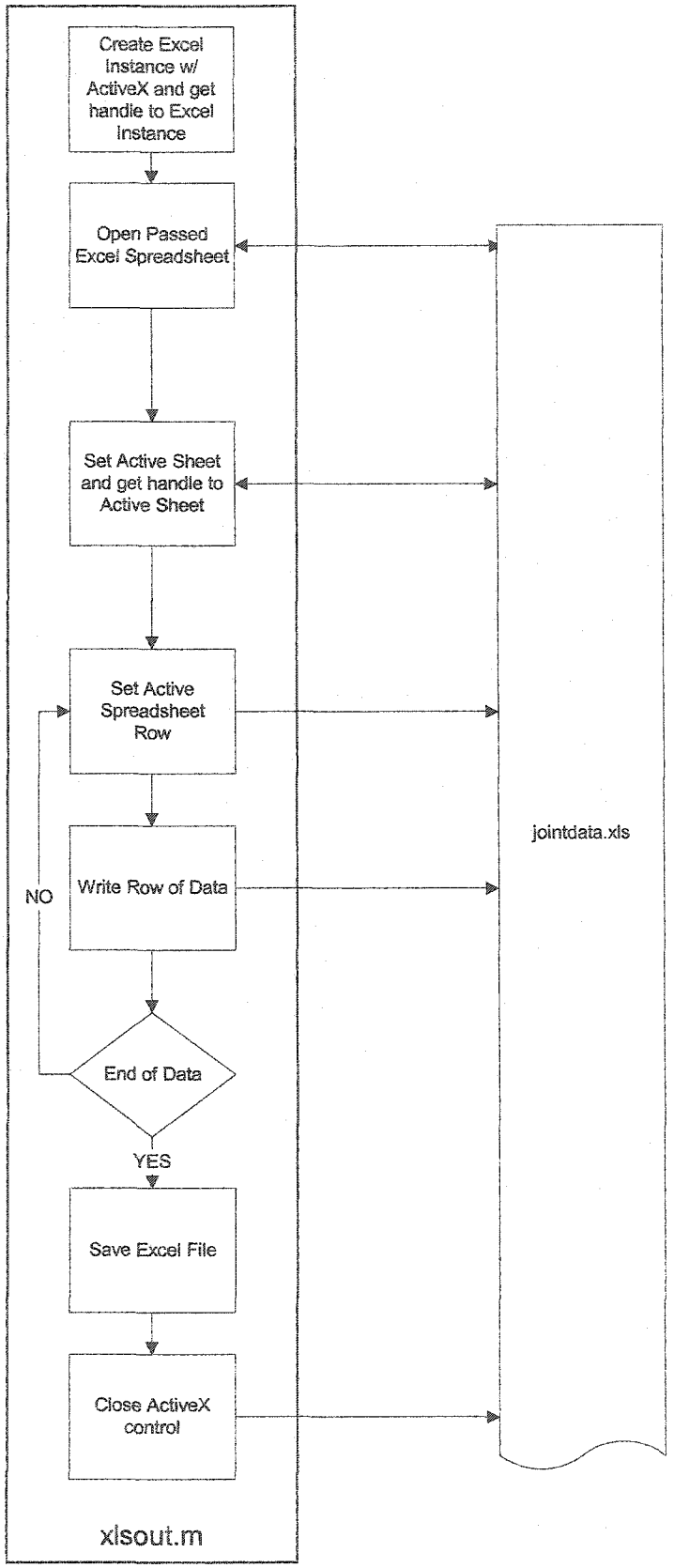


Figure 20. ActiveX MATLAB Function

4.1.3 Implementation of Simulation

The development of the MATLAB code provided a simple means of creating joint trajectories that could be implemented into vN4D quickly. The trajectory path generated in MATLAB was arbitrary, and therefore, verification that vN4D was implementing the desired path is required. To verify that the vN4D software was simulating the joints accurately, comparisons of MATLAB generated joint trajectories to the actual simulated joint trajectories were performed for each joint and for various trajectory paths.

Figure 21 shows a comparison of the trajectory for the first joint in the TELBOT robot. There is no discernable difference between the two trajectory paths, which shows that the vN4D is moving as directed by the MATLAB joint trajectory data.

The full simulation of moving simulated fuel pins can now be generated, repeated, changed, and verified within the simulation environment. Figure 22 shows one frame of the simulation environment for a multiple move operation. This simulation picks fuel pin 1 (Pin 1) from its initial position and places it on top of Pin 9. The manipulator then moves to a stand-by location before moving back to Pin 1. The gripper then engages Pin 1 for the second time and moves it back to its original position. The trajectory generation script, traj8.m, is shown in its entirety within Appendix A.7.

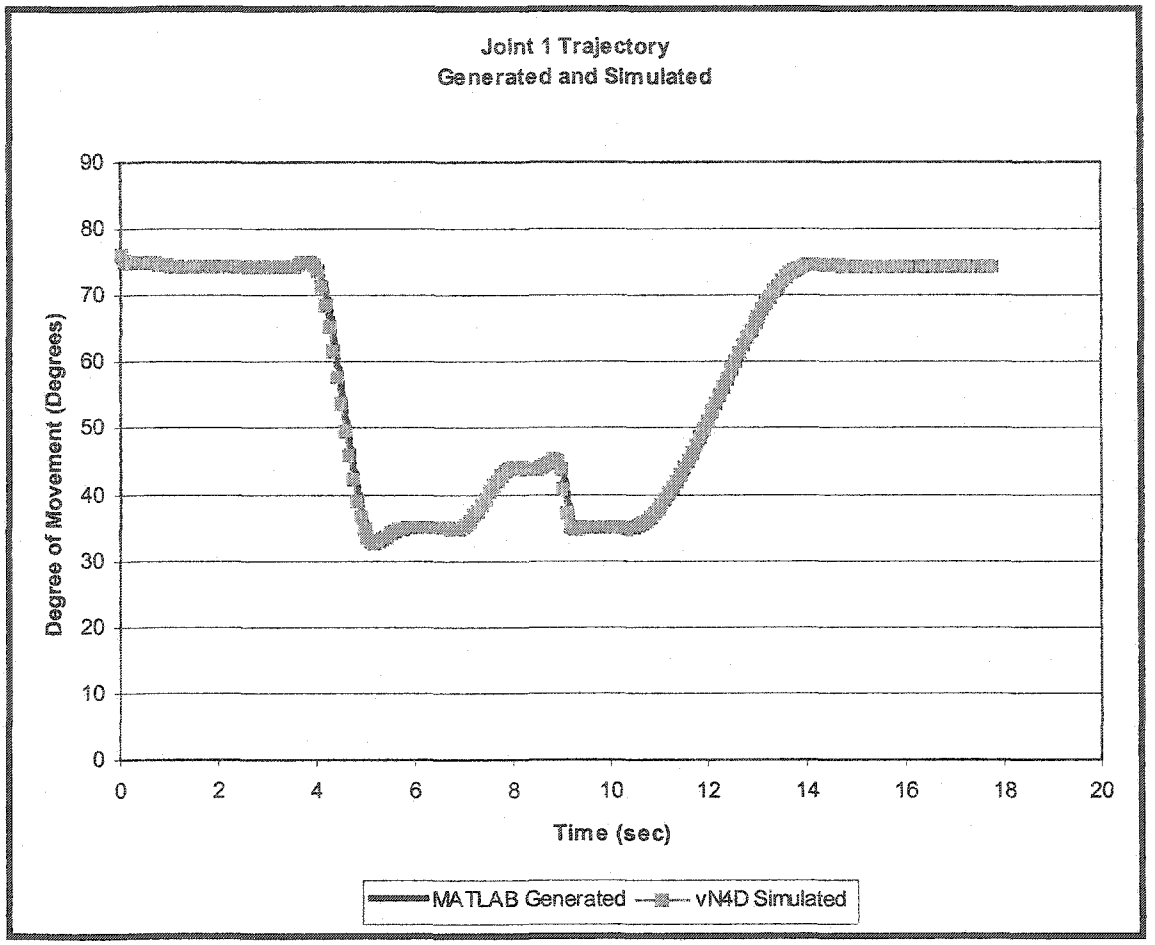


Figure 21. Joint Trajectory Data

4.1.3.1 Limitations of the Simulation

While the trajectory generation script worked for many of the desired robot motions, the script was not entirely successful. As described in Section 3.5.2, the inverse kinematics routine would fail to find a solution at or near robot singularities. As the inverse kinematics routine is only called for each knot point, correct selection of these points within the traj8.m script is required. If a knot point is selected near a robot singularity, then the iterative inverse kinematics routine would fail to converge on a solution.

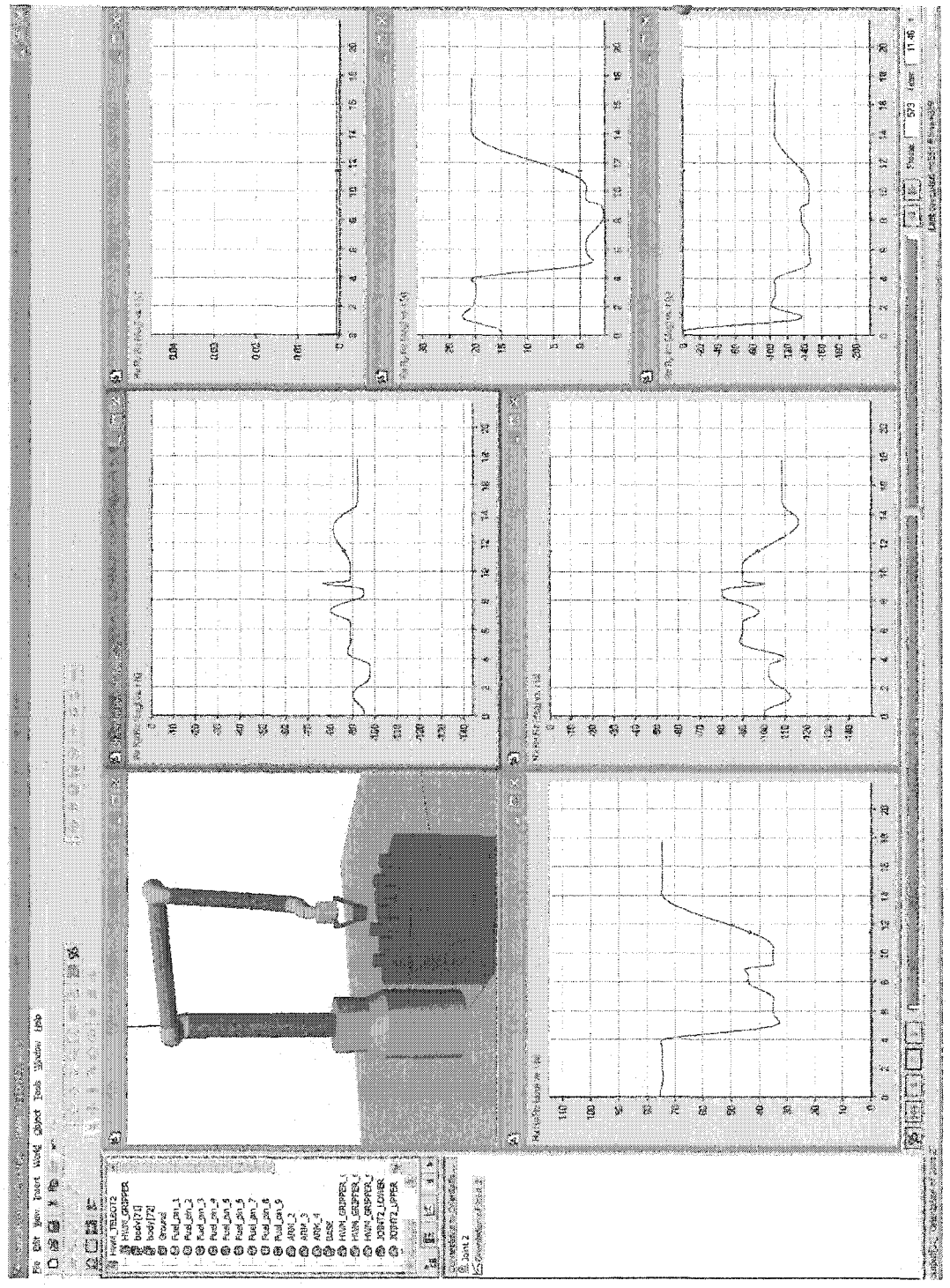


Figure 22. Simulation Environment

Without adequate joint-space variables for each of the robot movements, a trajectory path could not be created, and subsequently, a simulation could not be performed.

Therefore, an iterative process of trail-and-error was used to ensure that the selected knot points did not approach a robot singularity or extend beyond the reach of the robot.

4.2 Cell Layout Considerations and Time-Motion Studies

The development of a nuclear materials production facility is no small endeavor involving a vast array of functional, operational and design requirements. In order to simplify design, a facility is typically broken down into discrete systems, structures or components. While the design of the entire transmutation fuel manufacturing facility is beyond the scope of the work, a generalized effort is made to address design concerns at the sub-system level, specifically within the realm of the robotics work cells used in the back-end of the transmutation fuel manufacturing.

Process throughput measurements of fuel-pins per minute, fuel rods per hour, or fuel assemblies per day are of great importance for the owners and operators of the fuel manufacturing facility. To determine overall process throughput, individual cycle times are required. Not only are cycle time evaluations used to ensure efficient production, but also have the benefit of providing and optimizing material accountability, process functions, manufacturing cell layout, worker exposure, and Capital and Operating Costs, to name a select few.

Within the systems engineering realm, process function requirements are measured by performance requirements. Typically process throughput is a performance

requirement that must be met, typically due to cost drivers or other downstream processes that require a cycle time so that they too do not go idle.

4.2.1 Computer Aided Process Planning

The design of a process often entails some sort of design optimization. Product throughput and system availability is often of primary concern, which has lead to the development of numerous software packages to perform various levels of process simulation. Simulation can range from a simple tally of process steps, implementation of a discrete process simulation tool, or to a statistical based simulation that utilizes Monte Carlo or similar methodologies.

4.2.2 Work Cell Cycle Time Study

When machines and processes are placed in a work cell, the overall process becomes dependent on each of the in-cell machines and their direct interaction. In our arbitrary case, the primary interaction revolves around the utilization of a robotic manipulator to move individual fuel pins. Similarly, Kats and Levner [48] studied periodic scheduling of parts in a robotic production system functioning under a given repetitive robot's route. Kats and Levner [48] objective was to determine the starting times and durations of processing operations so as to minimize the cycle time. They were able reduce the problem to finding parametric critical paths, also known a process "bottlenecks". The study of cycle time and generally identifying critical paths is discussed in this work.

As shown in Figure 23, a general representation is made for a hypothetical work cell that would process fuel pins from the sintering and boat unloading machine, to the grinder, and subsequently to the fuel rod loading platform. The process involves mainly pick-and-place operations. The process begins with the robotic manipulator picking an individual fuel pin out of the gravity shoot of one of the two sintering ovens. It is

assumed that the sintering oven processes thirty fuel pins in its sintering boat. It is also assumed that the sintering oven is able to sinter one boat of fuel pins per hour. The hot boats are placed in an internal buffer. A cold sintering boat is subsequently unloaded, and the fifty fuel pins enter the sintering oven's gravity shoot where they are arranged and presented for engagement.

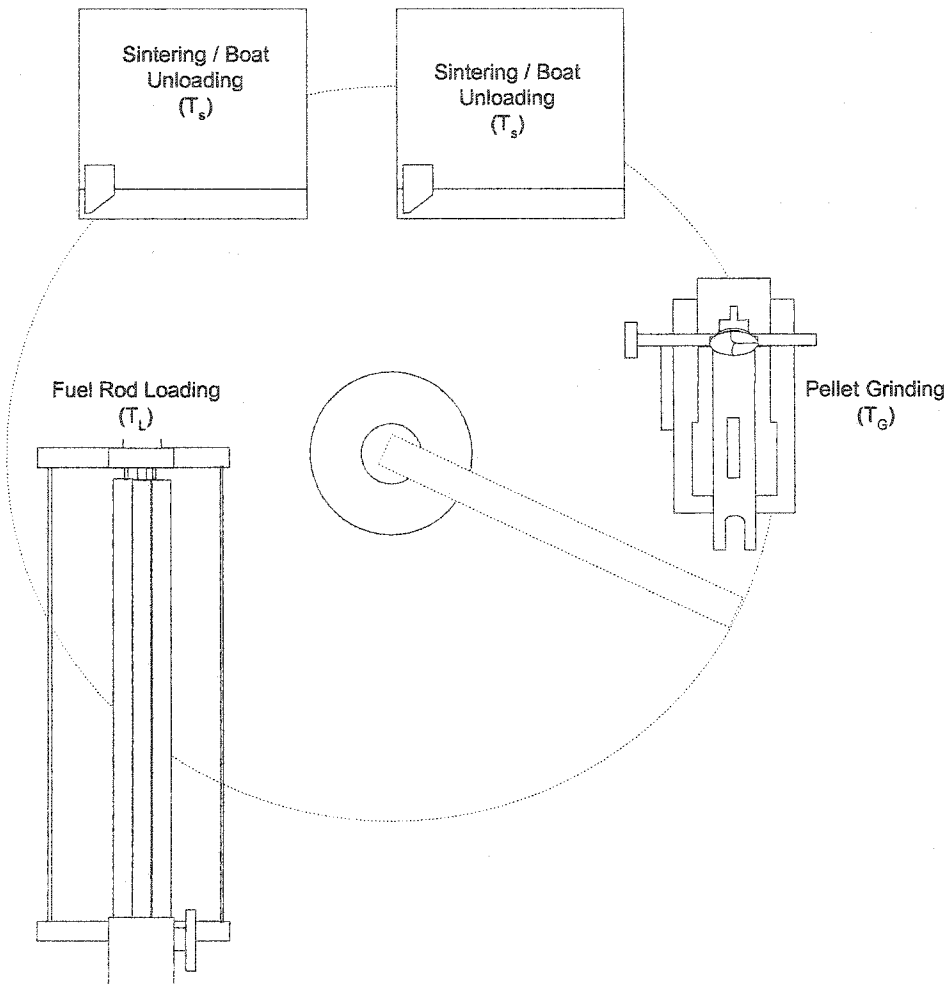


Figure 23. Simplified Work Cell

After gripping an individual fuel pin at the sintering oven, the manipulator then rotates to the grinding machine and subsequently drops off the individual pin. The robot

then moves to the grinder gravity shoot to pick up a subsequent ground fuel pin. The grinding machine is assumed to utilize a first-in first-out sequence, where pins are ground and inspected automatically within the machine. While only one pin can be assumed to be processed at once, it is assumed that there is an input and output buffer. It is reasonable to assume that the grinding machine can process one fuel pin per minute.

After the manipulator grips a fuel pin out of the grinder's gravity shoot, the pin is moved to and placed on a fuel rod loading tray and subsequently pushed into the fuel rod cladding tube. The manipulator then moves to the process start point, at the sintering oven.

The above process description reveals some assumed times for processing, e.g., sintering and grinding. In investigating the cycle time for the process, the discrete processes for the sintering and grinding can not be changed. However, the pick-in-place operations can theoretically be evaluated. The sintering time (T_S) and grinding time (T_G) was assumed above to be:

$$T_S = 30\,{}^{\text{Pins}}\!/_{\text{Boat}} \times 1\,{}^{\text{Boat}}\!/_{\text{hr}} = 30\,{}^{\text{Pin}}\!/_{\text{hr}} \tag{37}$$

$$T_G = 60\,{}^{\text{Pin}}\!/_{\text{hr}} \tag{38}$$

As Figure 23 depicts, there are two sintering ovens, which allows for the nominal production rate of $60\,{}^{\text{Pin}}\!/_{\text{hr}}$. It is now up to the manipulator to move the $60\,{}^{\text{Pin}}\!/_{\text{hr}}$ or $1\,{}^{\text{Pin}}\!/_{\text{min}}$ from the two sintering ovens to the grinder, and from the grinder to the fuel rod loading and back to the sintering oven.

Using a preliminary timing study shown in Figure 24, the robot movement nominally takes 96 seconds to complete the entire cycle, which is consistent with the 3-D

simulation. This is 36 seconds longer than the cycle time for either the grinding machine or the two sintering ovens.

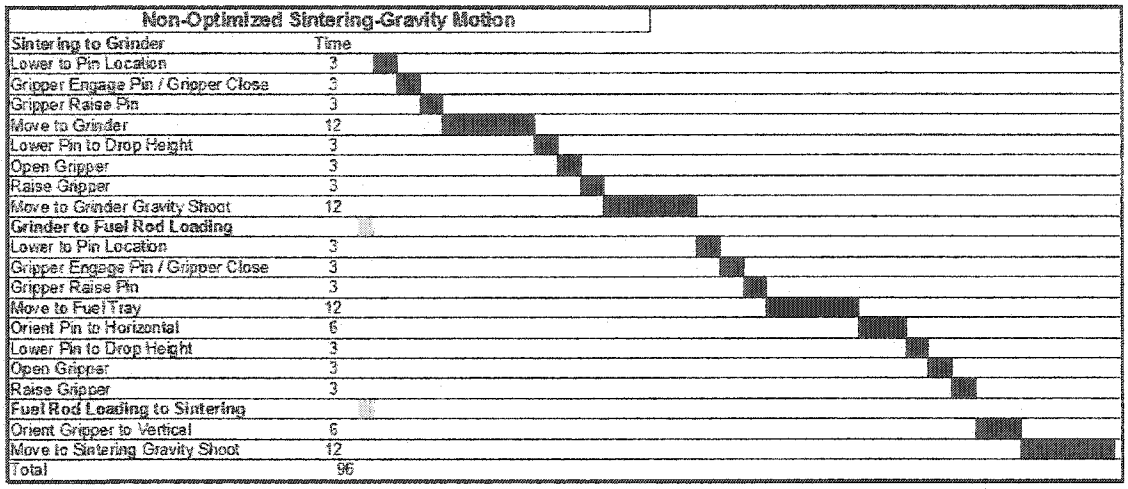


Figure 24. Time Study for Robot Move

In order to optimize these times, an analysis of the movement is required. Primarily, the large movements are performed at the robot base (Joint 1), where the entire robot rotates from station to station.

Using in Figure 23, it can be assumed that the three process stations are nominally 120 degrees apart. The movement from one cell is assumed to take 12 seconds, which is consistent with the 3-D simulation. Therefore, the rotational speed of the robot base is:

$$\dot{\theta}_{base} = \frac{120\,\text{deg}}{12\,\text{sec}} = 10\,{}^{\text{deg}}/_{\text{sec}} \tag{39}$$

This rotational speed for the base joint is relatively slow for a commercial manipulator, where one could expect speeds of $90\,{}^{\text{deg}}/_{\text{sec}}$. Because of the nature of the material being handled by the manipulator, it may not be desirable to utilize the full speed capabilities of the robot. Angular joint accelerations required to reach these speeds would need to be analyzed for adverse effects to the fuel pins, as the accelerations may be

amplified at the end-effector. Therefore, for our optimization study, we will utilize speeds of one-third of the maximum 90 $^{deg}/_{sec}$.

Because of the large movements by the base joint, the overall cycle time is very sensitive to improvements in this joint. As expected, when the speeds for the robot were increased three-fold, i.e., base joint speed increase from 10 $^{deg}/_{sec}$ to 30 $^{deg}/_{sec}$, significant reductions cycle time where achieved, as shown in Figure 25.

Now, the process is throughput is restricted by the capabilities of the sintering oven and grinder. With these robot movement rates, two more sintering ovens and one more grinder could be added to the process, resulting a approximately 2 $^{Pin}/_{min}$ or 120 $^{Pin}/_{hr}$.

Optimized Sintering-Gravity Motion	
Sintering to Grinder	Time 3
Lower to Pin Location	1
Gripper Engage Pin / Gripper Close	1
Gripper Raise Pin	1
Move to Grinder	4
Lower Pin to Drop Height	1
Open Gripper	1
Raise Gripper	1
Move to Grinder Gravity Shoot	4
Grinder to Fuel Rod Loading	
Lower to Pin Location	1
Gripper Engage Pin / Gripper Close	1
Gripper Raise Pin	1
Move to Fuel Tray	4
Orient Pin to Horizontal	2
Lower Pin to Drop Height	1
Open Gripper	1
Raise Gripper	1
Fuel Rod Loading to Sintering	
Orient Gripper to Vertical	2
Move to Sintering Gravity Shoot	4
Total	32

Figure 25. Optimized Time Study for Robot Move

CHAPTER 5

SUMMARY AND RECOMMENDATIONS

5.1 Summary

The development of this work was focused around two primary objectives, assessing automation and robotics in the nuclear industry, and modeling the back-end of the transmuter fuel manufacturing process.

Good assessment of the use of automation and robotics in the nuclear industry was made. Actual use of automation and robotics in the nuclear industry is quite limited. This is especially true in the United States, where a majority of the hot-cell activities are performed utilizing master-slave-manipulators, as shown in Figure 5. Most fuel manufacturing is performed with "glove-box" technologies due to the low doses from uranium. Automation, if used, is employed in areas requiring enhanced throughput and/or where tasks are highly repetitive and structured. Fuel reprocessing shows the highest utilization of automation where dose rates are relatively high.

The 3-D simulation of the back-end of the transmuter fuel manufacturing process was initially successful, as the models were of low fidelity. After extensive effort, integration of diverse software programs provided sufficient methods of producing 3-D models, robot trajectories, and subsequent simulations.

As a side effort, a preliminary timing study proved that significant throughput improvements could be obtained. While specific process times were assumed, the

exercise showed the importance of using as discrete simulator or other Computer Aided Process Planning software for process optimization.

An unexpected result of this effort was the ability to provide lessons-learned to other graduate students in their implementation and application of the various software packages, especially vN4D. Jamil Renno, a fellow graduate student, has been able to utilize this work to continue and expand the simulation development.

These early results continue to confirm the viability of fuel manufacturing processes for transmutation. As the manufacturing process moves from a conceptual nature to preliminary and final design stages, the methodology used in this work will be applicable, albeit at a much larger scale and greater fidelity.

5.2 Recommendations for Further Study

Utilization of vN4D for the simulation proved to be successful for small-scale process simulation. Most simulation was performed on a Laptop Computer with a Pentium 4 processor running at 1.2 GHz with 512 Meg of RAM. The video card used is an ATI Mobility Radion 9000 with 64 Meg of on-board RAM. While a relatively powerful platform, simulation times were less than optimal. The MATLAB generation of the trajectories was relatively fast, on the order of 15 to 70 seconds, depending on the number of iterations required by the numerical inverse kinematics solver. However, simulations in vN4D for a 15-second series of movements would take approximately 15 minutes to complete. Jamil Renno was able to simulate the process faster on a workstation employing quad-processors.

During the assessment of robotics and automation within the nuclear facilities, several instances of simulation tools were used. These tools offer both computer-aided design

and multi-body simulation features. Various robot simulation packages throughout the robotics industries, of which IGRIP (Delima / DENEB Robotics) and ROBCAD (Technomatix) have found wide acceptance [31, p. 50].

As the design matures and the process can be modeled with higher fidelity, it is recommended that simulation package that is design specifically for the automated manufacturing and robotics field is used rather than a using vN4D - a dynamic simulation and Finite Element Analysis package. This would be an improvement over the integration of tasks across multiple software platforms, Pro/ENGINEER for 3-D modeling, MATLAB for trajectory generation, Excel for data storage, and vN4D for simulation.

APPENDIX – MATLAB CODE

A.1 coordinatelocations.m

```
%-------------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Sept. 2, 2003
%
%-------------------------------------------------------------------%
%
%  coordinatelocations - Data file of initial fuel pin locations
%                        and other key coordinate locations
%
%
%---------------------ASSUMPTIONS---------------------------------%
%
%  All units are in inches at the WCS
%
%-------------------------------------------------------------------%

global PN1 PN2 PN3 PN4 PN5 PN6 PN7 PN8 PN9 RH PH% Pin Location Varibles

% All fuel pin initial locations
PN1=    [     1 0 0 28.126
              0 -1 0 19.874
              0 0 -1 25.225
              0 0 0 1];


PN2=    [     1 0 0 28.126
              0 -1 0 12.000
              0 0 -1 25.225
              0 0 0 1];

PN3=    [1 0 0 28.126
              0 -1 0 4.126
              0 0 -1 25.225
              0 0 0 1];

PN4=    [1 0 0 36.000
              0 1 0 19.874
              0 0 1 25.225
              0 0 0 1];

PN5=    [1 0 0 36.000
              0 1 0 12.000
              0 0 1 25.225
              0 0 0 1];

PN6=    [1 0 0 36
              0 1 0 4.126
              0 0 1 25.225
```

```
            0 0 0 1];


PN7=    [1 0 0 43.874
             0 1 0 19.874
             0 0 1 25.225
             0 0 0 1];

PN8=    [1 0 0 43.874
             0 -1 0 12.000
             0 0 -1 25.225
             0 0 0 1];

PN9=        [1 0 0 43.874
             0 -1 0 4.126
             0 0 -1 25.225
             0 0 0 1];

% The following set the pick-up and Set-Down locations
% for the fuel pins
raiseheight= 4; %in the z direction
pinheight = (1.5*2.6);
RH  =       [0 0 0 0
            0 0 0 0
            0 0 0 raiseheight
            0 0 0 0];

PH  =       [0 0 0 0
            0 0 0 0
            0 0 0 pinheight
            0 0 0 0];

dropoff1=[1 0 0  0
          0 1 0 40
          0 1 -1 48
          0 0 0 1];

      dropoff2=[1 0 0 28
          0 -1 0 -10
          0 1 -1 0
          0 0 0 1];

%--------------------------%
%Test Coordinate Locations %
%--------------------------%


% Hand Picked robot location near pin 2

home1=[  -0.9994   -0.0349   -0.0051  -17.4754
   -0.0349    0.9992    0.0185  -18.8249
    0.0044    0.0187   -0.9998   68.6626
         0         0         0    1.0000];

home2=[ 0.0037    0.3301   -0.9439   10.3804
    0.5890   -0.7635   -0.2647   25.5025
   -0.8081   -0.5550   -0.1973   77.5399
         0         0         0    1.0000];

test1=[1 0 0 10.3804;
       0 1 0 25.5025;
       0 0 1 77.5399;
       0 0 0 1];
```

A.2 hold.m

```
function [ts,h]=hold(manip,q,T,ss)
timestep=0;
q2 = [-65.7 85.486 121.83 0 -26.873 -76];
q=ikine(manip,q,q2);
for Time = T(1):ss:T(2),
    timestep=timestep+1;
    ts(timestep)=Time;
    h(timestep,:)=q;
end
```

A.3 invkin.m

```
%-----------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Sept 2, 2003
%
%-----------------------------------------------------------------%
%
%  invkin - Inverse Kinematic Solver
%
%  This is a test routine to develop means of developing
%  inverse kinematic routines for use in solving at singularities
%
%
%---------------------ASSUMPTIONS---------------------------------%
%
% Solution is computed iteratively using the pseudo-inverse of the
% manipulator Jacobian.
%
% Such a solution is completely general, though much less efficient
% than specific inverse kinematic solutions derived symbolically.
%
% This approach allows a solution to obtained at a singularity, but
% the joint angles within the null space are arbitrarily assigned.
%
%---------------------CREDITS/References--------------------------%
%  The following routine is an adaptation of the files associated
%  with the robotics toolbox, which was developed by Peter Corke.
%  The code was modified to deal only with the TELBOT model case.
%  The original code is more robust and handles cases where there
%  are other than 6DOF robot objects.
%
%  P.I. Corke, "A Robotics Toolbox for MATLAB"
%  IEEE Robotics and Automation Magazine, Volume 3(1),
%  March 1996, pp. 24-32.
%  http://www.cat.csiro.au/cmst/staff/pic/robot/
%
%
%  The Differential Translation and Rotation Method is described
%  in Fu et al., pp. 547-553.
%-----------------------------------------------------------------%

function qt = invkin(robot, tr, q, m)

ilimit = 5000; %this is  the limit of iterations (i) for the process
stol = 1e-12;  %This is the accuracy of the solution (Solution Tolerance)
q = q(:); %Otherwise sets the guess q-matrix to the passed matrix - ras
nm = 1;
count = 0;
while nm > stol,
```

```
    %Find the error (e) using differential
    %translation and rotation method
    e = tr2diff(fkine(robot, q'), tr);  %Call to the robotics toolbox
    %Determine the inverse Jacobian
    dq = inv(tb_jacobian(robot, q)) * e;
    q = q + dq;
    nm = norm(dq);
    count = count+1;
    if count > ilimit
        errordlg('Solution wouldn''t converge','Inverse Kinematics');
        error('Solution wouldn''t converge')
    end
end

%while nm > stol,
%                       e = tr2diff(fkine(robot, q'), tr);
%                       %dq = pinv( jacob0(robot, q) ) * e;
%            dq = inv( jacob0(robot, q) ) * e;
%                       q = q + dq;
%                       nm = norm(dq);
%                       count = count+1;
%                        if count > ilimit
%                   errordlg('Solution wouldn''t converge','Inverse Kinematics');
%                   error('Solution wouldn''t converge')
%              end
%         end


if count > 1000
    h=warndlg('Large number of Iterations required to reach solution','Solution
Warning');
    pause(2);
    close(h);
end
qt = q'; % Pass the inverse kinematics q-matrix back
```

A.4 move.m

```
%--------------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Sept 2, 2003
%
%--------------------------------------------------------------------%
%
%  move - generate the move command
%
%
%---------------------ASSUMPTIONS----------------------------------%
%


function trajdata = move(tb,nd,T,P2,P3,P4)

h=waitbar(0,'Solving Inverse Kinematics');
[sr,sc]=size(nd);
disp('Solving inverse kinemaics')
Q(1,:)=[ nd(sr,2) nd(sr,3) nd(sr,4) nd(sr,5) nd(sr,6) nd(sr,7)];
waitbar(.25,h);
Q(2,:)=invkin(tb,P2,Q(1,:)); % This is the iterative solution routine
%Q(2,:)=ikine(tb,P2,Q(1,:));
```

```
waitbar(.5,h);
Q(3,:)=invkin(tb,P3,Q(2,:)); % This is the iterative solution routine
waitbar(.75,h);
Q(4,:)=invkin(tb,P4,Q(3,:)); % This is the iterative solution routine
waitbar(1,h);
disp('----  The Joint Coordinates for the Motion are    ---')
Q
close(h)

%Perform the 4-3-4 Joint Interpolation
clear gripper pin1;
[ts,h]=traj434(Q,T);
[nr,nc]=size(h);
for i=1:nr
    gripper(i,:)=tb.gripper;
    pin(i,:)=tb.pin;
end
trajdata=[nd; ts' h gripper pin]; %New Data (nd) concatinates the time and joint data
```

A.5 tb_jacobian.m

```
%-----------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Sept 2, 2003
%
%-----------------------------------------------------------------%
%
%  tb_jacobian(robot, q)
%
%  Robot Jacobian
%
%---------------------ASSUMPTIONS---------------------------------%
%
% Solution of the TELBOT Specific Jacobian is based on a 6X6 matrix
%
%---------------------CREDITS/References--------------------------%
%  The following routine is an adaptation of the files associated
%  with the robotics toolbox, which was developed by Peter Corke.
%  The code was modified to deal only with the TELBOT model case.
%  The original code is more robust and handles cases where there
%  are other than 6DOF robot objects.
%
%  P.I. Corke, "A Robotics Toolbox for MATLAB"
%  IEEE Robotics and Automation Magazine, Volume 3(1),
%  March 1996, pp. 24-32.
%  http://www.cat.csiro.au/cmst/staff/pic/robot/
%-----------------------------------------------------------------%

function J0 = tb_jacobian(robot, q)
        %
        %   dX_tn = Jn dq
        %
        n = robot.n;
        L = robot.link;                 % get the links

        J = [];
        U = robot.tool;
        for j=n:-1:1,

                        U = L{j}( q(j) ) * U;
                        d = [   -U(1,1)*U(2,4)+U(2,1)*U(1,4)
```

```
                                -U(1,2)*U(2,4)+U(2,2)*U(1,4)
                                -U(1,3)*U(2,4)+U(2,3)*U(1,4)];
                        delta = U(3,1:3)';      % nz oz az
                        J = [[d; delta] J];
    end
    %  convert to Jacobian in base coordinates
       Tn = fkine(robot, q); % end-effector transformation
    R = Tn(1:3,1:3); %Extract Rotation matrix
%    R = tr2rot(Tn);
       J0 = [R zeros(3,3); zeros(3,3) R] * J;
```

A.6 telbot.m

```
%-----------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Sept 2, 2003
%
%-----------------------------------------------------------------%
%
%   This routine creates the fundamental DH-Matrix of the
%   TELBOT robotic system.  These parameters can be modified
%   to suite the exact configuration of the specific robot
%   For Revision 0, the DH-parameters from W.Waelishmiller
%   and Lee are used to develop the fundamental description
%   of the robot.
%
%--------------------ASSUMPTIONS--------------------------------%
%
%   The use of the Puma560 mass, moment, and friction were assumed
%   This is assumed to be valid in the case that the Waelishmiller
%   arm is similar in function.  No relevant data is available for
%   the arm.  Modeling in Pro/E does not accuractly reveal
%   the arm properties sufficiently.
%
%-----------------------------------------------------------------%

function tb=telbot(varargin);

if nargin == 0, %Empty Input Arguments

%-- Set Global Variables --%

%-- Set the Manipulator lengths from the base to the end effector --%
    s1=71.75;
    s2=8;
    s3=8;
    s4=31.25;
    s5=5.75;
    s6=12;
    a23=37.75;

    % Set the Link Matrix for the Robot
    % in the form of:
    % L = LINK([alpha A theta D])
    L{1}=link([ pi/2    0     0      s1  0   0],'standard');
    L{2}=link([ 0       a23   pi/2   s2  0   pi/2],'standard');
    L{3}=link([ -pi/2   0    -pi/2   s3  0   -pi/2],'standard');
    L{4}=link([ pi/2    0     0      s4  0   0],'standard');
    L{5}=link([ -pi/2   0     0      s5  0   0],'standard');
    L{6}=link([ 0       0     0      s6  0   0],'standard');
```

```
    tb=robot(L); %call to robotics toolbox

    tb.name='TELBOT Manipulator';
    tb.manuf='Waelishmiller';
    tb.gravity = [ 0 0 386.09];

    %-- Set Child Class Properties --%
    tb.gripper='open';
    tb.pin=0;
else
    disp('ERROR - Telbot Class does not accept input arguments')
end
```

A.7 traj8.m

```
%-------------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - March 10, 2004
%
%-------------------------------------------------------------------%
%
%  traj - Trajectory Development Routine
%
%  This is a test routine to develop means of developing
%  trejecotry paths from an initial and a final coordinate
%  position and orientation.
%
%  This data will then be stored in an Excel File so that it may
%  be simulated in MSC.VisualNastran.
%
%
%---------------------ASSUMPTIONS-----------------------------------%
%
%   The use of the Puma560 mass, moment, and friction were assumed
%   This is assumed to be valid in the case that the Waelishmiller
%   arm is similar in function.  No relavant data is available for
%   the arm.  Modelling in Pro/E does not accuractly reveal
%   the arm properties suffeciently.
%
%-------------------------------------------------------------------%

clc % clear command window
clear %Clear variables and functions from memory

%----------Define Global Variables---------%
global ss; %step size in time (seconds)
global ts; % Time steps for all motion
global v; % Joint Velocity Parameters
global a; % Joint Acceleration Parameters
global PN1 PN2 PN3 PN4 PN5 PN6 PN7 PN8 PN9 % Fuel Pin Location Varibles
global RH; % Raise height for pickup of fuel pins
global OPEN; %Set Gripper Open Dimension
global CLOSED; %Set Gripper Closed Dimension
%-----------Set Model Parameters------------%
ss=0.02;
ts=0;
T = [0 0 0 0]; % Time values for solving Trajectories
OPEN=-1.25; %Set Gripper Open Dimension
CLOSED=-0.7; %Set Gripper Closed Dimension
```

```
true=1; % Set variable for true and false
false=0;
%--- Create an instance of the Telbot Robot---%
tb=telbot

%
% some useful poses of the Telbot Robot
%

qz = [0 0 0 0 0 0]; % zero angles, L shaped pose
qr = [0 pi/2 -pi/2 0 0 0]; % ready pose, arm up
q4=[ 76.242 -95.86 -99.895 0 13.671 2.4748e-12];
q4=[ 75 -95 -100 0 15 0];
q4rad=q4*pi/180;

%Load Key Coordinate Locations Fuel Pin Locations
coordinatelocations;

%Load Velocity and Acceleration Data
velacceldata;

%-----------------------------------------------------------------%
%   The following sets the manipulator positions in time          %
%-----------------------------------------------------------------%

%-- Initial Position Hold for MSC Software Initialization
T=[0 0.25 0 0];
tb.gripper=OPEN;
tb.pin=0;
nd=wait(tb,[0 q4rad 0 0],T);

%---- Move from First Point to Second
T=[0.25 1 2 3];
nd = move(tb,nd,T,PN1+RH,PN1+RH,PN1);

%Hold at this position
T=[3 3.25 0 0];
tb.gripper=CLOSED;
nd=wait(tb,nd,T);

T=[3.25 3.5 0 0];
tb.pin=1;
nd=wait(tb,nd,T);

%Move to next position
T=[3.5 4 5 6];
nd = move(tb,nd,T,PN1+RH,PN9+RH,PN9+PH);

tb.pin=0;
tb.gripper=OPEN;
T=[6 6.5 0 0];
nd=wait(tb,nd,T);

%Move to next position
T=[6.5 7 8 8.5];
nd = move(tb,nd,T,PN9+RH+PH,PN8+RH,PN8);

T=[8.5 9 9.25 9.5];
nd = move(tb,nd,T,PN8+RH,PN9+PH+PH+RH,PN9+PH);

T=[9.5 9.75 0 0];
tb.gripper=CLOSED;
nd=wait(tb,nd,T);

T=[9.75 10 0 0];
tb.pin=1;
nd=wait(tb,nd,T);

T=[10 10.5 14 15];
nd = move(tb,nd,T,PN9+PH,PN1+RH+PH,PN1+PH);
```

```
tb.pin=0;
tb.gripper=OPEN;
T=[15 15.5 0 0];
nd=wait(tb,nd,T);
%----------Convert Trajectory Data from Radian to Degrees---------%
for m=2:7,
        nd_deg(:,m)=nd(:,m)*180/pi;
end
nd_deg(:,1)=nd(:,1);% Transferrers time vector
nd_deg(:,8:9)=nd(:,8:9);

%-------------------------------------------------------------------%
% Plot the joint trajectory data for all point-to-point motions   %
%-------------------------------------------------------------------%
clf;
for i=2:7,
        subplot(3,2,i-1);
        plot(nd_deg(:,1),nd_deg(:,i));
        title(strcat('Joint',int2str(i-1)));
        ylabel('Angle (Deg)')
    end

%--------------------------------------%
%--- Create Excel File information ----%
%--------------------------------------%
Choice=menu('Do you want to create the Excel file of the Data?','Yes','No');
if Choice == 1;
    Header='Test Data for input to Excel';
    Col={'Time','Joint 1 (Deg)','Joint 2 (Deg)','Joint 3 (Deg)','Joint 4 (Deg)','Joint 5
(Deg)','Joint 6 (Deg)'};
    Filename='C:\Documents and Settings\Richard Silva\My
Documents\thesis\aaa\waelishmiller2\jointdata.xls';
    %Filename='C:\My Documents\jointdata-trajdev.xls';
    tab=3; %Place the data on the Tab 3

    %--- Write Data to Excel File ---%

    xlsout(nd_deg,Header,Col,Filename,tab);
else
    disp('**********************************')
    disp('* DATA NOT WRITTEN TO EXCEL FILE *')
    disp('**********************************')
end
```

A.8 traj434.m

```
%-------------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Aug 17, 2003
%  Revision 1 - Nov 24, 2003
%-------------------------------------------------------------------%
%
%  traj434 - Trajectory Development Routine
%            Using 4-3-4 Joint Trajectory
%
%  This routine develops the joint trajectory for a
```

```
%  manipulator based on four knot points.
%       1) Initial position
%       2) Lift-Off position
%       3) Set Down position
%       4) Final position
%  Each joint has the following three trajectory segments:
%       1) The fist segment is a fourth-degree polynomial
%          specifying the trajectory from the initial position
%          to the lift-off position.
%       2) The second trajectory segment (or mid-trajectory
%          segment) is a third-degree polynomial specifying the
%          trajectory from the lift-off position to the set-down
%          position.
%       3) The last trajectory segment is a fourth-degree polynomial
%          specifying the trajectory from the set-down position to the
%          final position.
%
% [ts,H] = function traj434(Q,T,v,a,ss,plot)
%
%  The function receives the following variables
%       Q = 4x6 matrix containing the four knot points for each
%           of the six manipulator joints
%       T = 1x4 array of the time stamps for each of the four
%           knot points
%       v = 4x6 matrix containing the four knot point velocities
%           for each of the six manipulator joints
%       a = 4x6 matrix containing the four knot point accelerations
%           for each of the six manipulator joints
%       ss = the time step for each interpolated position (e.g.
%           0.02 seconds)
%
%   The function returns an Nx6 matrix (H) containing the joint positions
%   for the time steps.  The time steps (t) are also returned
%
%
%---------------------ASSUMPTIONS---------------------------------%
%
%
%---------------------References----------------------------------%
%
% Fu, et. al. Chapter 4.3.1
%-----------------------------------------------------------------%

%-----------------------------------------------------------------%
%  Rev 01
%  Updated code to handle multiple sized Q input matrix
%

function [ts,h]=traj434(Q,T)

%--- The polynomial coefficient are solved for below ---%
%     See Fu, et al. Chapter 4.3.1 for an explanation
%     of these polynomials as well as a derivation of the
%     solution scheme used for the simultaneous solution
%     of many equations with many unknowns (See C matrix)

% Establish Global Variables

global ss v a;

% Establish real time variables

t1=T(2)-T(1);
t2=T(3)-T(2);
t3=T(4)-T(3);

[sr,sc]=size(Q); %Size of rows and size of columns

for i=1:sc
    a10(i)=Q(1,i); %Eq. 4.3-10
    a20(i)=Q(2,i); %Eq. 4.3-17
```

```
    a30(i)=Q(4,i); %Eq.4.3-32
end

for i=1:sc
    a11(i)=(v(1,i)*t1); %Eq. 4.3-11
    a21(i)=(v(2,i)*t2); %Eq. 4.3-18
    a31(i)=(v(4,i)*t3); %Eq. 4.3-33
end

for i=1:sc
    a12(i)=(a(1,i)*t1^2)/2; %Eq. 4.3-12
    a22(i)=(a(2,i)*t2^2)/2; %Eq. 4.3-19
    a32(i)=(a(4,i)*t3^2)/2; %Eq. 4.3-34
end

for i=1:sc,
    del1(i)=Q(2,i)-Q(1,i); %Eq. 4.3-41
    del2(i)=Q(3,i)-Q(2,i); %Eq. 4.3-42
    del3(i)=Q(4,i)-Q(3,i); %Eq. 4.3-43
end

for i=1:sc, %Eq. 4.3-45
    y(1,i)=del1(i)-a12(i)-a11(i);
    y(2,i)=-1*a(1,i)*t1-v(1,i);
    y(3,i)=-1*a(i,i);
    y(4,i)=del2(i);
    y(5,i)=-1*a(4,i)*t3+v(4,i);
    y(6,i)=a(4,i);
    y(7,i)=del3(i)+a32(i)-a31(i);
end

C=[ 1 1 0 0 0 0 0
    3/t1 4/t1 -1/t2 0 0 0 0
    6/t1^2 12/t1^2 0 -2/t2^2 0 0 0
    0 0 1 1 1 0 0
    0 0 1/t2 2/t2 3/t2 -3/t3 4/t3
    0 0 0 2/t2^2 6/t2^2 6/t3^2 -12/t3^2
    0 0 0 0 0 1 -1]; %Eq. 4.3-46
for i=1:sc,
    yy=y(:,i);
    x(:,i)=inv(C)*yy; %Eq. 4.3-49
end

a13=x(1,:); %Eq. 4.3-47
a14=x(2,:);
a21=x(3,:);
a22=x(4,:);
a23=x(5,:);
a33=x(6,:);
a34=x(7,:);

timestep=0; %Initialize and set to Zero
Time=0; %Initialize and set to Zero

% Determine Joint positions for the First Segment

for Time = T(1) : ss : T(2),
    timestep=timestep+1;
    t=(Time-T(1))/(T(2)-T(1)); %normalized Time Varible
    ts(timestep)=Time;
    h(timestep,:)=a14*t^4 + a13*t^3 + a12*t^2 + a11*t + a10; %Equation (4.3-3)
end

% Determine Joint positions for the Second Segment

for Time = T(2) : ss : T(3),
    timestep=timestep+1;
    t=(Time-T(2))/(T(3)-T(2));
    ts(timestep)=Time;
    h(timestep,:)=a23*t^3 + a22*t^2 + a21*t + a20; %Equation (4.3-3)
end
```

```
% Determine Joint positions for the Third and Final Segment

for Time = T(3) : ss : T(4),
    timestep=timestep+1;
    t=(Time-T(3))/(T(4)-T(3));
    ts(timestep)=Time;
    h(timestep,:)= a34*t^4 + (-4*a34 + a33)*t^3 +(6*a34-3*a33 + a32)*t^2 + (-4*a34 +
3*a33 -2*a32 + a31)*t + (a34 - a33 + a32 -a31 + a30); %Equation (4.3-50)
%    h(timestep,:)=a34*t^4+a33*t^3 + a32*t^2 + a31*t + a30; %Equation (4.3-4)
end
```

A.9 velacceldata.m

```
%-------------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Aug 17, 2003
%
%-------------------------------------------------------------------%
%
%  velacceldata -   data file containing the joint velocity
%                   and acceleration data used in the trajectory
%                   generation routine.


%--- Set the velocity matrix ---%
global v,a;

v(1,:)=[0 0 0 0 0 0 0];
v(2,:)=[2 2 2 2 2 2 2];
v(3,:)=[2 2 2 2 2 2 2];
v(4,:)=[0 0 0 0 0 0 0];

%--- Set the Acceleration Matrix --%

a(1,:)=[0 0 0 0 0 0 0];
a(2,:)=[5 5 5 5 5 5 5];
a(3,:)=[5 5 5 5 5 5 5];
a(4,:)=[0 0 0 0 0 0 0];
```

A.10 wait.m

```
function nd=wait(tb,nd,T)

global ss

timestep=0;
[sr,sc]=size(nd);
%q=[ nd(sr,2) nd(sr,3) nd(sr,4) nd(sr,5) nd(sr,6) nd(sr,7) nd(sr,8)];
q=[ nd(sr,2) nd(sr,3) nd(sr,4) nd(sr,5) nd(sr,6) nd(sr,7)];

for Time = T(1):ss:T(2),
    timestep=timestep+1;
    ts(timestep)=Time;
    h(timestep,:)=q;
        gripper(timestep,:)=tb.gripper;
    pin1(timestep,:)=tb.pin;
end
```

```
nd=[nd; ts' h gripper pin1];
```

A.11 xlsout.m

```
%-------------------------------------------------------------------%
%  Richard Silva
%
%  Thesis - 3D Simulation of Transmuter Fuel
%           Manufacturing Process
%
%
%  Department of Mechanical Engineering
%  University of Nevada, Las Vegas
%
%  Revision 0 - Aug 24, 2003
%
%-------------------------------------------------------------------%
%
%  xlsout(m,header,colnames,filename) opens a Microsoft Excel spreadsheet using
%  the MATLAB ActiveX interface.
%
% Inputs:
%    m           Matrix to write to file
% (Optional):
%    header      String of header information.  Use cell array for multiple lines
%                   DO NOT USE multiple row character arrays!!
%    colnames    (Cell array of strings) Column headers.  One cell element per column
%    filename    (string) Name of Excel file.  If not specified, contents will
%                   be opened in Excel.
%    tab         The tab (aka sheet) of the Excel file to place the data
%
%---------------------ASSUMPTIONS---------------------------------%
%
%

function xlsout(m,header,colnames,filename,tab);

% Determine the size of the passed matrix to be placed in Excel
[nr,nc] = size(m);
if nc>256
    error('Matrix is too large.  Excel only supports 256 columns');
end;

% Open Excel, open workbook, change active worksheet,
% get/put array, save.
% First, open an Excel Server.

% Create the Excel AxtiveX client-sever
% Interface for data-passing

%Create an instance of the Status Bar

h=waitbar(0,'Creating Excel Instance');
%disp('Creating the Excel Instance')
Excel = actxserver('Excel.Application');


%If the user does not specify a filename, we'll make Excel visible
%If they do, we'll just save the file and quit Excel without ever making
% it visible
if nargin<4
    set(Excel, 'Visible', 1);      %You might want to hide this if you autosave the file
end;

% Get a handle to the Excel.Workbook
% Open a file into the Excel Instance
waitbar(.1,h,'Set instance of Excel Workbook')
```

```
%disp('Set instance of the Excel.Workbook')
ExcelWorkBook=Excel.Workbooks;
waitbar(.2,h,'Invoke Passed Filename');
%disp('Invoke the passed filename')
Workbook=invoke(ExcelWorkBook,'Open',filename);

% Set the active Excel Sheet to the
% Passed sheet number (aka tab)
% If no sheet is passed, then set the
% first sheet as the active sheet
Tabs=Excel.ActiveWorkBook.Sheets;
if nargin<5
    sheet=get(Tabs, 'Item', 1);
else
    sheet=get(Tabs, 'Item', tab);
end
invoke(sheet,'Activate');


% Get a handle to the active sheet.
waitbar(.3,h,'Get a Handle to the Active Sheet');
%disp('Get a handle to the active sheet')
Activesheet = Excel.Activesheet;


%Write header
waitbar(.4,h,'Writing Passed Header');
%disp('Writing Passed Header')
% Check for empty header information
if nargin<2 | isempty(header)
    nhr=0;
% Since header is non-empty, put into Excel
elseif iscell(header)
    nhr = length(header);       %Number header rows
    for ii=1:nhr
        ActivesheetRange = get(Activesheet,'Range',['A' num2str(ii)],['A' num2str(ii)]);
        set(ActivesheetRange, 'Value', header{ii});
    end;
else
    nhr = 1;                    %Number header rows
    ActivesheetRange = get(Activesheet,'Range','A1','A1');
    set(ActivesheetRange, 'Value', header);
end;

waitbar(.5,h,'Writing Column Names');
%disp('Writing Column Names')
%Add column names
if nargin>2 & ~isempty(colnames)
    nhr = nhr + 1;      %One extra column name
    ncolnames = length(colnames);
    for ii=1:ncolnames
        colname = localComputLastCol('A',ii);
        %    cellname = [char(double('A')+ii-1) num2str(nhr+1)];
        cellname = [colname num2str(nhr)];
        ActivesheetRange = get(Activesheet,'Range',cellname,cellname);
        set(ActivesheetRange, 'Value', colnames{ii});
    end;
end;
waitbar(.6,h,'Putting Matlab Array into Excel');
%disp('Putting Matlab Array into Excel')
% Put a MATLAB array into Excel.
FirstRow = nhr+1;           %You can change the first data row here.  I start right after
the headers
LastRow = FirstRow+nr-1;
FirstCol = 'A';         %You can change the first column here
LastCol = localComputLastCol(FirstCol,nc);
ActivesheetRange = get(Activesheet,'Range',[FirstCol num2str(FirstRow)],[LastCol
num2str(LastRow)]);
set(ActivesheetRange, 'Value', m);

waitbar(.8,h,'Saving the Excel File');
```

```
%disp('Saving the file')
% If user specified a filename, save the file and quit Excel
%=if nargin==4
%=    invoke(Workbook, 'SaveAs', [pwd filesep filename]);
if nargin>4
    invoke(Workbook, 'Save');
    invoke(Excel, 'Quit');

    [pathstr,name,ext] = fileparts(filename);
%=    disp(['Excel file ' name '.xls has been created.']);
    disp(['Excel file ' name '.xls has been saved.']);
end;
waitbar(1,h,'Deleting the ActiveX Object');
%Delete the ActiveX object
%disp('Delete the ActiveX object')
delete(Excel);
waitbar(1,h,'File Saved Successfully')
pause(1);
close(h);

function LastCol = localComputLastCol(FirstCol,nc);
% Comput the name of the last column where we will place data
%Input
%  FirstCol  (string) name of first column
%  nc        total number of columns to write

%Excel's columns are named:
% A B C ... A AA AB AC AD .... BA BB BC ...
FirstColOffset = double(FirstCol) - double('A');    %Offset from column A
if nc<=26-FirstColOffset       %Easy if single letter
    %Just convert to ASCII code, add the number of needed columns, and convert back
    %to a string
    LastCol = char(double(FirstCol)+nc-1);
else
    ng = ceil(nc/26);       %Number of groups (of 26)
    rm = rem(nc,26)+FirstColOffset;         %How many extra in this group beyond A
    LastColFirstLetter = char(double('A') + ng-2);
    LastColSecondLetter = char(double('A') + rm-1);
    LastCol = [LastColFirstLetter LastColSecondLetter];
end;
```

BIBLIOGRAPHY

[1] Hechanova, Anthony E. "Quarterly Report Fourth Quarter (December 2001 to February 2002) University of Nevada, Las Vegas, Advanced Accelerator Applications, University Participation Program" Harry Reid Center for Environmental Studies. University of Nevada, Las Vegas. March 26, 2002.

[2] Los Alamos National Laboratory (LANL). "Addressing the Nuclear Waste Issue" Brochure. LALP-01-227. October 2001.

[3] Nuclear Energy Agency "Trends in the Nuclear Fuel Cycle: Economic, Environmental and Social Aspects" http://www1.oecd.org/publications/e-book/6602011e.pdf

[4] Benedict, M. and Pigford. T.H. *Nuclear Chemical Engineering*. New York: Mc Graw-Hill. 1957. Library of Congress Catalog Card Number 56-12523

[5] Haas, D., J. Somers, A. Renard, A. La Fuente. *Feasibility of the Fabrication of Americium Targets*. Online Article: Nuclear Energy Association http://www.nea.fr/html/trw/docs/mol98/session3/SIIIpaper1.pdf

[6] National Institute of Standards and Technology. Web Site: http://www.ceramics.nist.gov/srd/summary/ftguo2.htm.

[7] Avallone, Eugene A. and Theodore Baumeister III. *Mark's Standard Handbook for Mechanical Engineers*. Tenth Edition. New York: McGraw-Hill. 1987. ISBN 0-07-022625-3

[8] U.S. Department of Energy. *DOE Handbook – Design Considerations*. DOE-HDBK-1132-99. National Technical Information Service, Springfield Virginia. 1999.

[9] Bowen, W. W., D. L. Sherrell, M. J. Wiemers. "Automated Handling for SAF Batch Furnace and Chemistry Analysis Operations" Hanford Engineering Development Laboratory 1981. HEDL-SA—2414-FP

[10] Frederickson, J. R. and R. M. Horgos. "SAF Line Powder Operations" Hanford Engineering Devlopment Laboratory. 1983. HEDL-SA—2894-FP.

[11] Frederickson, J. R., R. C. Eschenbaum, and L. H. Goldmann. "Powder Handling for Automated Fuel Processing" Hanford Engineering Devlopment Laboratory. 1987. HEDL-SA—3690-FP

[12] Geber, E. W., W. W. Bowen, and J. A. Williams. "SAF Line Furnace Operations" Hanford Engineering Development Laboratory. 1983. HEDL-SA—2891-FP.

[13] Egli, W. and R. L. Bogart. "Sintering Boat Transport System for the SAF Line" Hanford Engineering Development Laboratory. 1983. HEDL-SA—2906-FP

[14] Jedlovec, D. R., W.W. Bowen, and R. L. Brown. "SAF Line Pellet Gaging" Hanford Engineering Devlopment Laboratory. 1983. HEDL-SA—2903-FP

[15] Gerber, E. W.; N.C. Hoitink; and R. A. Graham. "Remote Fabrication of Breeder Reactor Fuel" Hanford Engineering Development Laboratory. 1984. HEDL-SA—3157-FP.

[16] Drotning, W.; Kimberly, H.; Wapman, W.; Darras, D.[and others] "A sensor-based automation system for handling nuclear materials" 1997 international conference on robotics and automation, Albuquerque, NM (United States), 20-25 Apr 1997 ; PBD: 1997 (SAND--96-2303C; CONF-970469—5)

[17] U.S. Department of Energy "A DOE SUCCESS: Automated Nuclear Weapon Component Handling" http://www.rim.doe.gov/WALS.pdf

[18] Robinson, D.G.; Atcitty, C.B. "Safety assessment of high consequence robotics system" National System Safety Conference, Albuquerque, NM (United States), 12-17 Aug 1996 ; PBD:[1996] (SAND—96-2014C; CONF-960869—15)

[19] Jaquish, W.R. "PUREX irradiated fuel recovery simulation" Deneb user group meeting; 10-14 Oct 1994; Detroit, MI (United States); DOE Project (WHC-SA-2664;CONF-9410231—1)

[20] Bennett, P.C. and J.B. Stringer. "Monitored Retrievable Storage/Multi-Purpose Canister Analysis: Simulation and Economics of Automation" *Proceedings from the Fifth Annual International Conference of High Level Radioactive Waste Management.* Volume 2. Las Vegas Nevada, May, 1994

[21] Chen, Tao. "Robotics for Nuclear Waste Handling" Thesis. Department of Electrical and Computer Engineering. University of Nevada, Las Vegas. May 1998

[22] Krueger, Allison Jeanne. "Three Dimensional Computerized Model of an Elastic Robot Arm" Thesis. Department of Civil and Mechanical Engineering. University of Nevada, Las Vegas. August, 1989

[23] O'Donnell, William M. "Automated Robotics System for Nuclear Waste Handling (Hardware and Software)" Department of Electrical and Computer Engineering, University of Nevada, Las Vegas. December, 1995.

[24] Bousquet Guy & G Brähler "A MOX Fuel Plant in Russia: The Engineering Work Has Started" *The Uranium Institute, Twenty-Second Annual Symposium*. 3-5 September 1997: London

[25] Jacquet Roland, "MOX fuel fabrication and its external communication in France" Presentation. COGEMA. http://www.jnc.go.jp/news/topics/PT021203/pdf/engilsih_original/06_r_jacquet.pdf

[26] Blomeke, J. O. *A Review and Analysis of European Industrial Experience in Handling LWR Spent Fuel and Vitrified High-Level Waste*. June 1998. Oak Ridge National Laboratory. Office of Civilian Radioactive Waste Management. ORNL/TM-10696

[27] The Math Works, Inc. *MATLAB External Interfaces*. Version 6. 2002. Natick, MA: The Math Works, Inc.

[28] Lee, H.Y. and W. Wälischmiller. "Development and Application of the TELBOT System – A new TELE Robot System" *11th CISM-IFToMM Symposium on Theory and Practice of Robots and Manipulators*. Udine, Italy. July 1996.

[29] Schlotter, A.; Pfeiffer, F.; "Modeling, Control and Optimization of a New Telerobot Robotics and Automation", 2000. *Proceedings. International Conference on Robotics & Automation*. Volume: 3 , 2000 San Francisco, CA. April 2000. Page(s): 2659 -2664 vol.3

[30] Fu K.S., R.C. Gonzales, C.S.G. Lee. *Robotics: Control, Sensing, Vision, and Intelligence*. New York: McGraw-Hill. 1996. ISBN 0-07-004997-1

[31] Nof, Shimon Y. ed. *Handbook of Industrial Robots* Second Edition. New York, New York. John Wiley & Sons, Inc. 1999. ISBN: 0-471-17783-0

[32] Denavit, J. and R.S. Hartenberg. "A kinematic notation for lower-pair mechanisms based upon matricies". *Journal of Applied Mechanics*. Vol. 77 pp.215-221, 1955

[33] Koivo, Antti J. *Fundamentals for Control of Robotic Manipulators*. John Wiley & Sons, Inc: 1989. ISBN 0-471-85714-9

[34] Paul, Richard P *Robot manipulators : mathematics, programming, and control: the computer control of robot manipulators* Cambridge, Massachusetts: MIT Press, 1981

[35] Goldenberg, A.; Benhabib, B.; Fenton, R.; "A complete generalized solution to the inverse kinematics of robots", *IEEE Journal of Robotics and Automation*, Volume: 1 Issue: 1 , Mar 1985 Page(s): 14 –20

[36] Corke, P.I. "A Robotics Toolbox for MATLAB", *IEEE Robotics and Automation Magazine*, Volume 3(1), March 1996, pp. 24-32. http://www.cat.csiro.au/cmst/staff/pic/robot/

[37] Manocha, D.; Canny, J.F., "Efficient inverse kinematics for general 6R manipulators" *IEEE Transactions on Robotics and Automation*, Volume: 10, Issue: 5 , Oct. 1994 Pages:648 – 657

[38] Jonghoon Park; Wan-Kyun Chung; Youngil Youm; "Characteristics of optimal solutions in kinematic resolutions of redundancy", *IEEE Transactions on Robotics and Automation*, Volume: 12 , Issue: 3 , June 1996 Pages:471 – 478

[39] Grudic, G.Z.; Lawrence, P.D. "Iterative inverse kinematics with manipulator configuration control"; Robotics and Automation, IEEE Transactions on , Volume: 9 Issue: 4 , Aug 1993 Page(s): 476 -483

[40] Sciavicco, L.; Siciliano, B.; "A solution algorithm to the inverse kinematic problem for redundant manipulators" IEEE Journal of Robotics and Automation, Volume: 4 , Issue: 4 , Aug. 1988 Pages:403 – 410

[41] Ahuactzin, J.M.; Gupta, K.K.; "The kinematic roadmap: a motion planning based global approach for inverse kinematics of redundant robots", IEEE Transactions on Robotics and Automation, Volume: 15 , Issue: 4 , Aug. 1999 Pages:653 – 669

[42] Kreutz-Delgado, K.; Agahi, D.; "A recursive singularity-robust Jacobian generalized inverse", *IEEE Transactions on Robotics and Automation*, Volume: 11 , Issue: 6 , Dec. 1995 Pages:887 – 892

[43] Klein, C.A.; Chu-Jeng, C.; Kittivatcharapong, S.; "Testing iterative robotic algorithms by their rate of convergence", *IEEE Transactions on Robotics and Automation*, Volume: 7, Issue: 5 , Oct. 1991 Pages:686 – 687

[44] Wang, L.-C.T.; Chen, C.C.; "A combined optimization method for solving the inverse kinematics problems of mechanical manipulators", *IEEE Transactions on Robotics and Automation*, Volume: 7 , Issue: 4 , Aug. 1991 Pages:489 – 499

[45] Lloyd, John E. and Vincent Hayward; "A Discrete Algorithm for Fixed-path Trajectory Generation at Kinematic Singularities" *IEEE International Conference on Robotics and Automation*, Minneapolis, Minnesota, April 22-28, 1996.

[46] Janabi-Sharifi, F.and W.J. Wilson; "A Fast Approach for Robot Motion Planning" *Journal of Intelligent and Robotic Systems*, Volume 25, Issue 3, July 1999 Pages: 187-212

[47] The Math Works, Inc. *Using MATLAB*. Version 6. 2002. Natick, MA: The Math Works, Inc.

[48] Kats, Vladimir and Eugene Levner; "Cyclic Scheduling of Operations for a Part Type in an FMS Handled by a Single Robot: A Parametric Critical-Path Approach" *International Journal of Flexible Manufacturing Systems* Volume 10, Issue 2, Apr 1998 Pages: 129-138

VITA

Graduate College
University of Nevada, Las Vegas

Address:

2534 Cortina Avenue
Henderson, Nevada 89074

Degree:

Bachelor of Science, Mechanical Engineering, 1994
West Virginia University

Special Honors and Awards:

Tau Beta Pi – Engineering Honor Society

Thesis Title:

Three Dimensional Modeling and Simulation of Manufacturing Processes for Transmuter Fuel Fabrication

Thesis Examination Committee:

Chairperson, Dr. Georg Mauer, Ph. D.
Committee Member, Dr. William Culbeth, Ph. D.
Committee Member, Dr. Mohamed Trabia, Ph. D.
Graduate Faculty Representative, Dr. Robert Schill, Ph. D.