3-21-2019

Analytical Models and Control Design Approaches for a 6 DOF Motion Test Apparatus

Kyra L. Schmidt

ANALYTICAL MODELS AND CONTROL DESIGN APPROACHES FOR A 6 DOF MOTION TEST APPARATUS

THESIS

Kyra L. Schmidt, 2nd Lt, USAF

AFIT-ENY-MS-19-M-245

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENY-MS-19-M-245

ANALYTICAL MODELS AND CONTROL DESIGN APPROACHES FOR A 6 DOF MOTION TEST APPARATUS

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Aeronautical Engineering

Kyra L. Schmidt, B.S.A.E.

2nd Lt, USAF

March 21, 2019

AFIT-ENY-MS-19-M-245

ANALYTICAL MODELS AND CONTROL DESIGN APPROACHES FOR A 6 DOF MOTION TEST APPARATUS

THESIS

Kyra L. Schmidt, B.S.A.E.
2nd Lt, USAF

Committee Membership:

Dr. Richard G. Cobb
Chair

Maj Costantinos Zagaris, Ph.D.
Member

Capt Joshuah A. Hess, Ph.D.
Member

Abstract

Wind tunnels play an indispensable role in the process of aircraft design, providing a test bed to produce valuable, accurate data that can be extrapolated to actual flight conditions. Historically, time-averaged data has made up the bulk of wind tunnel research, but modern flight design necessitates the use of dynamic wind tunnel testing to provide time-accurate data for high frequency motion. This research explores the use of a 6 degree of freedom (DOF) motion test apparatus (MTA) in the form of a robotic arm to allow models inside a subsonic wind tunnel to track prescribed trajectories to obtain time-accurate force and moment coefficients. Specifically, different control laws were designed, simulated, and integrated into a 2 DOF model representative of the elbow pitch and wrist pitch joints of the MTA system to decrease positional tracking error for a desired end-effector trajectory. Stability of the closed-loop systems was proven via Lyapunov analysis for all of the control laws, and the control laws proved to decrease tracking error during the trajectory case studies. An adaptive sliding mode control scheme was chosen as most suitable to simulate on the 6 DOF model due to the small tracking error as compared to the other control schemes and the availability of parameters of the actual MTA system when subject to the time-varying aerodynamics of the wind tunnel.

Acknowledgements

I dedicate this to my family, without whom I never would have been able to finish.

I would like to extend my gratitude to all those who made this research possible, especially to Dr. Cobb for his unnecessarily outstanding patience and understanding.

Kyra L. Schmidt

Table of Contents

Page

List of Figures

List of Tables

ANALYTICAL MODELS AND CONTROL DESIGN APPROACHES FOR A 6 DOF MOTION TEST APPARATUS

I. Introduction

1.1 Motivation

Although modern computational methods and numerical models are at their most powerful since inception, they alone cannot provide accurate-enough information to sufficiently test and design the types of aerospace systems that are at the forefront of current research. For instance, at the high angles of attack that some modern aircraft can achieve, time-dependent aeroelastic effects become significantly more prominent [20]. Likewise, small unmanned aircraft systems (SUAS) and micro air vehicles (MAVs) are extremely demanding in terms of agility requirements and flying qualities [38]. Furthermore, store separation presents a unique case in that neither the aerodynamics nor the trajectory of the store after separation are steady. To improve the performance and handling of all of the aforementioned systems requires a superior understanding of basic flight mechanics and vehicle dynamics to update the models of such systems. Fortunately, ongoing progress in the field of wind tunnel research can provide the data necessary to refine the models.

Historically, wind tunnels have been used extensively to gather irreplaceable force and moment data throughout the design stages of aerospace systems. For the nondimensional parameters gathered from the tests to be accurately extrapolated to actual flight conditions, many correction factors must be considered, such as the induced vortices due to wall effects of the wind tunnel to the change in local wind velocity

due to decreased cross-sectional area encountered at the item under test (IUT) [27]. Ideally, these correction factors should be enough to provide valuable data; however, advances in modern flight technologies require improved fidelity of wind tunnel testing and the models used to update them. One way to improve wind tunnel tests is to do so dynamically. For example, a store after separation will follow a highly nonlinear trajectory. In order for a wind tunnel test of store separation to be valid, the experiment must be able to simulate the trajectory with minimal positional error and to measure the relevant data with time accuracy. Similarly, aeroelastic effects of flexible wing aircraft and other dynamic models require time-accurate simulation and measurement as well. It has become necessary to be able to characterize these potentially deleterious effects dynamically as opposed to averaging the data from a static test because of the high frequencies at which they occur.

Dynamic testing proves useful as a tool to study the time-varying force and moment coefficients on a test article as the end effector tracks a trajectory. Although a free-flying IUT would be the most representative model of an aircraft in flight, these can be difficult to control as they must provide their own propulsion system. Alternatives include using either a free-motion rig or a forced-motion rig. A free-motion rig does allow certain degrees of freedom such as free-to-roll or free-to-pitch motions while a forced-motion rig forces the model to follow a precomputed path [34]. Specific designs of these rigs differ depending on the laboratory and experiment-in-question. One example of a forced-motion rig that has flexible applications is a model attached to a sting controlled by a robotic arm to manipulate the IUT within the wind-tunnel section. The Air Force Institute of Technology (AFIT) has procured such a device referred to as the Motion Test Apparatus (MTA) for this purpose. The MTA has six degrees-of-freedom (6 DOF) pictured in Figure 1 below which are actuated by motor controllers.

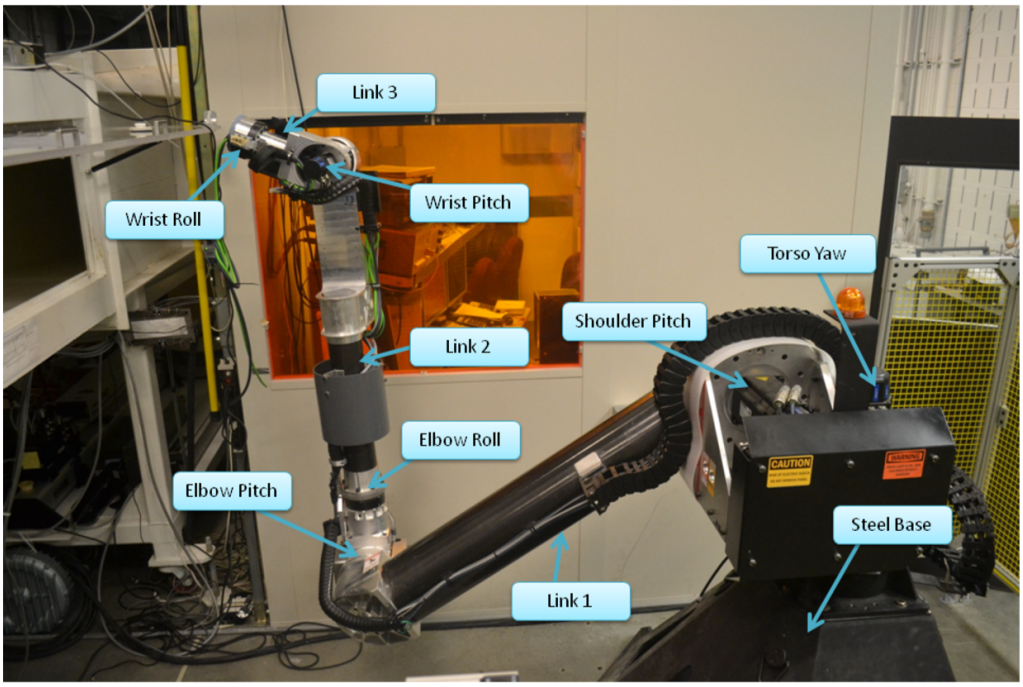


Figure 1. Air Force Institute of Technology 6 DOF Motion Test Apparatus with major components labeled. [26]

1.2 Problem Statement

The MTA, manufactured by RE2, Inc., was originally designed to test MAVs at the University of Florida-Research and Engineering Education Facility (REEF). Due to the complications with relocation of the MTA to AFIT, including the significant size difference in the wind tunnel test sections, the difference in intended tests, and the short duration in which the relocating process occurred, validation and verification of the MTA performance and functionality was not completed at the REEF. Additionally, the modes of operation potentially available to the MTA at the REEF are no longer viable for operation at AFIT. Therefore, it has become necessary both to integrate the MTA hardware and to refine the software in order to better operate in the AFIT wind tunnel. Since its integration to the AFIT subsonic wind tunnel, attempts at characterization and control of the MTA have been conducted for simple trajectories including cyclical and one-off motion [26, 41, 9]. However, only propor-

tional control was used to improve positional tracking of the model for the wrist roll degree of freedom only. The gains used in the proportional control were tuned manually to find a marginal decrease in position error. Although proportional control is one of the simplest methods for control of any system, many other control methods are available for decreasing tracking error that are more suited for the applications of the MTA. Specifically, because the end-effector of the MTA is subject to constantly changing forces and moments due to the wind tunnel's effects on the IUT, the control laws in use must be able to account for the changes in uncertainties for such properties at the end-effector. Using proportional control may allow for adequate use in small, oscillatory motions, and increasing the proportional gain can improve performance. However, as the test trajectories become more complex and nonlinear to model real-life applications, a proportional control law can become unstable as gain increases, and it may be nearly impossible to measure or even bound the uncertainties in the force- and moment-coefficients, rendering the MTA useless for the test.

As such, there is room for improvement in design of the control laws, with respect to robustness, further decreasing the tracking error, and extending the method to more degrees of freedom. For instance, adding feed-forward control to the control laws reduces lag during trajectory tracking [49]. In order to use feed-forward control, however, the control law needs to estimate physical parameters that change over time. This requires the use of adaptive control of some sort, a variety of which exist and have been proven in similar systems.

Ideally in the long term, the goal for the AFIT MTA is to have zero tracking error when following any prescribed trajectory even when subjected to unpredictable aerodynamic forces and moments from the wind tunnel. With this capability, the measurements on the test article will be solely due to aerodynamic effects as opposed to interference from the MTA dynamics. While this may not be entirely achievable,

to work towards that goal requires a working model of the MTA system so that the control laws used will have some predictive capabilities when computing control signals. To keep a reasonable scope for this study, the objectives are to derive a dynamic model for the MTA on which to simulate control laws, integrate those control laws onto the real MTA to test the same trajectories, and compare the results in order to update the models and control schemes in an iterative way.

1.3 Methodology

The MTA is situated next to the AFIT Low-Speed Wind Tunnel (subsonic). A sting attached to the end of the MTA extends into the test section of the wind tunnel through a nine-inch circular access port in the plexiglass. The IUT is attached to the end of the sting and is specifically manufactured to hold a Nano25 6-DOF force balance. This sensor produces signals in the form of analog voltages corresponding to the forces and moments experienced by the test model. The signals are sent to a primary computer via a National Instruments (NI) Data Acquisition (DAQ) system to be processed by LabVIEW software. To obtain model angle attitude data, a LORD MicroStrain inertia measurement unit (IMU) is also attached to the MTA. The signals from the IMU are collected by the DAQ via an RS-22 connection.

The MTA uses a secondary computer with a Linux operating system to perform the user-input mission trajectory files. These trajectory files are defined by the user, and can include a variety of dynamic tests. For the purposes of this research, test trajectories can include simple oscillatory pitch and pitch-plunge motions to characterize the MTA and to test the initial control laws. After the control laws are validated for simple motions, more complex trajectories can be performed to simulate store separation, for instance. Maneuvers such as store separation require all six DOFs of the MTA to work in coordination. Tests can be run at varying wind-tunnel speeds

and model configurations. Although testing to validate the control laws was initially planned, limited testing was performed on the MTA.

After the data has been collected, post-processing is performed by importing the data from NI LabVIEW into MATLAB. Ideally, the results consist of time-accurate force- and moment-coefficient data that can be compared to existing CFD results, at least for the store-separation case. Post-processing can also include comparisons of the IMU data with encoder data from the MTA.

1.4 Limitations

The ideal outcome of this research is to decrease the tracking error down to zero for the MTA with any model attachment and under any test conditions for abitrary trajectories. However, this is not a realistic goal as the design of experiments cannot be infinite, and the nonlinear nature of the system only allows for stability of the control laws for a limited range of initial and operating conditions. Additionally, there are physical and programmed constraints on the MTA to prevent damage but may lessen the effect of the control laws. The MTA can only operate in the confined space of the wind tunnel through a nine-inch access hole, limiting the available motion, but which also limits the range of motion in which the MTA must robustly operate. The primary focus of this research is to design and implement a more robust control and improved tracking accuracy for prescribed trajectories. As such, the sensors and DAQ systems will not be investigated despite their inevitable additions to the uncertainty in the measured outputs of the system.

1.5 Overview of Subsequent Chapters

The remainder of this thesis is organized as follows. Chapter II details the important background information regarding the MTA and nonlinear control in gen-

eral including relevant reference frames, coordinate transformations, and the MTA's forward and inverse kinematics. Chapter II also includes the dynamics for robotic manipulators, control methods, and experimental methods as well as an overview of relevant wind tunnel measurements that will be obtained.

Chapter III describes the experimental setup including the required equipment, operation, and specifications for the MTA and the wind tunnel. Additionally, Chapter III covers the design of the MTA including the control laws, simulation models, and the intended experimental setup. This includes a brief discussion of the wind tunnel models, the sensors and their measurements, and the data acquisition system.

Chapter IV contains the results of the simulated closed loop systems as well as the corresponding stability analysis. Additionally, Chapter IV describes the 6 DOF model as well as the intended experiments.

Chapter V discusses the conclusions and recommendations obtained from this research, including a summary of results, significance of the research, and recommendations for future MTA testing.

II. Literature Review

Chapter Overview

This chapter provides an overview of information relevant to the MTA system and current literature to solve the problem statement. It begins with definitions of the reference frames and coordinate transformations used in the MTA system. The chapter also details the forward and inverse kinematics capturing the motion of the MTA. A brief discussion of the difference between linear and nonlinear systems is followed by a derivation of the dynamic equations of motion for a robotic manipulator. A brief review of current techniques in nonlinear characterization, control methods, and experimentals methods for robotic manipulators is also presented. Additionally, the chapter covers relevant measurements and calculations for dynamic wind tunnel testing with emphasis on store separation.

2.1 Reference Frames

When discussing any dynamic problem, it is important to clearly define reference frames. For a robotic manipulator controlling a model for a dynamic wind tunnel test, there are four main references frames, depicted in Figure 2 that provide reference points for describing the kinematics of the system. Each reference frame follows the convention of the "right-hand rule" and forms an orthogonal set of x-, y-, and z-axes.

The wind tunnel reference frame, subscripted with a "w" in Figure 2, is statically centered within the wind tunnel with z_w-axis pointing down and the x_w-axis pointing in the direction of incoming wind from the wind tunnel pump. Typically, the wind tunnel reference frame is considered fixed [31].

The second reference frame in consideration, with the subscript "b", is the body-fixed reference frame. It is fixed to the IUT, usually with the origin centered at the

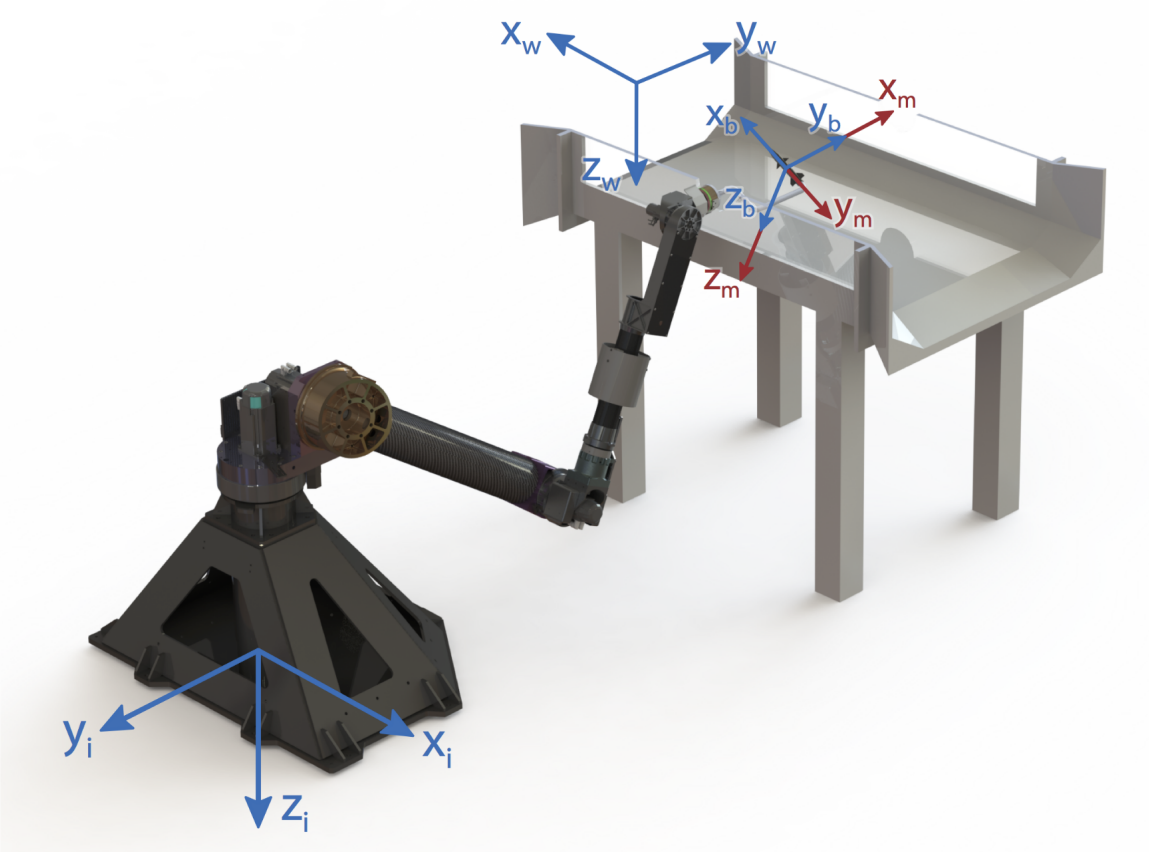


Figure 2. Subsonic wind tunnel reference frames with MTA [5]

quarter-chord along the fuselage of model. Similar to the wind tunnel reference frame, the z_b-axis also points downward out the bottom of the IUT, while the x_b-axis points out the front of the model. Accordingly, the y_b-axis points out the right wing.

The third reference frame of relevance is called the MTA-model reference frame. As shown in Figure 2, the MTA-model reference frame, subscripted with "m", is coincident with the body-fixed reference frame except with rotated axes. The z_m-axis is the same as the z_b-axis, still pointing down out the bottom of the IUT, but the x_m-axis is rotated 90°about the z_b-axis to coincide with the y_b-axis out the right side of the model. The y_m-axis then points in the opposite direction of the x_m-axis, out the back of the IUT. This reference frame exists because its origin is the model control point, the point about which trajectories are executed, for which the MTA was originally designed. The definition of the MTA-model reference frame is consistent with other research [26, 41, 9].

The final reference frame, considered the "inertial" reference frame of the MTA, is centered at the base of the MTA, directly under the torso yaw-joint on the floor. This reference frame has the subscript "i" as seen in Figure 2. The z_i-axis points into the laboratory floor, while the x_i-axis is parallel to the wind tunnel airflow, and the y_i-axis is perpendicular to the wind tunnel airflow but points away from the wind tunnel. The MTA's inertial reference frame is assumed to be fixed for all time and not accelerating with respect to the whole system setup. Desired MTA trajectory files are written with respect to the origin of the inertial reference frame [26].

Summarily, the MTA system uses both the traditional wind tunnel reference frame and the body-fixed reference frame but also needs a third inertial frame in order to define the user-input trajectory files. Due to the original design of the MTA, there is a fourth MTA-model reference frame that is coincident to but rotated from the body-fixed reference frame that will be used for deriving the kinematics of the MTA,

as described in Section 2.2.

2.2 Coordinate Transforms

In order to derive the aerodynamic forces and moments acting on the model, a coordinate transform between the body-fixed and the wind tunnel reference frames must exist. Standard practices utilize an Euler-rotation angle sequence to define the orientation and position of an IUT in a wind tunnel [31]. For the MTA, the orientation of the IUT with respect to the freestream velocity can be described by the angular rotations yaw, pitch, and roll, in that order. Yaw (ψ) is the rotation about the z_w-axis. Pitch (θ) is the rotation about the y_b-axis. Roll (ϕ) is the angular rotation about the x_b-axis. Applying the Euler angles to the components of the velocity vector yields a transformation matrix from the body-fixed frame to the wind tunnel reference frame, given below in Equation 1 where "S" signifies the sine function and "C" signifies the cosine function [31]. This rotation matrix is similar to the standard ZYX or 3-2-1 Euler transformation but differs in sign to match the reference frames described in Section 2.1 [31].

$$\begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}_w = \begin{bmatrix} C_\theta C_\psi & C_\theta S_\psi & -S_\theta \\ S_\phi S_\theta C_\psi - C_\phi S_\psi & S_\phi S_\theta S_\psi + C_\phi C_\psi & S_\phi C_\theta \\ C_\phi S_\theta C_\psi + S_\phi S_\psi & C_\phi S_\theta S_\psi - S_\phi C_\psi & C_\phi C_\theta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}_b \tag{1}$$

Equation (2) below defines the coordinate transformation from the body-fixed reference frame to the wind tunnel reference frame in terms of the angle-of-attack and sideslip. Angle-of-attack (α) is the angle between the freestream velocity (U_∞), or x_w, and the x_b-axis. Sideslip (β) is defined as the angle between x_b and the projection of U_∞ onto the x_w-y_w plane. Both of these angles can be seen below in Figure 3.

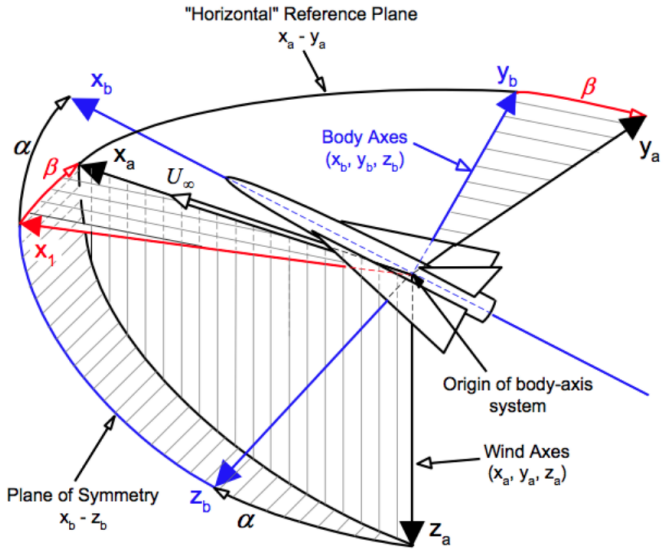


Figure 3. Model angular orientation defined within the wind tunnel [21]

Replacing the pitch angle, θ, with angle of attack, α, and the yaw angle, ψ, with the negative sideslip angle, β, in Equation (1) yields Equation (2) below where roll, ϕ, is set to zero because roll does not affect the aerodynamics of a model in a wind tunnel simulating forward flight conditions.

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_w = \begin{bmatrix} C_\alpha C_\beta & -C_\alpha S_\beta & -S_\alpha \\ S_\beta & C_\beta & 0 \\ S_\alpha C_\beta & -S_\alpha S_\beta & C_\alpha \end{bmatrix} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}_b \tag{2}$$

Equations (1) and (2) above describe the transformation matrices from the body-fixed reference frame to the wind tunnel reference frame, but all input trajectories are defined with respect to the MTA-model reference frame. As such, there must also be transformations from the MTA-model reference frame to the body-fixed frame and then another between the MTA intertial reference frame to the MTA-model reference frame. Because the body-fixed reference frame and the model reference frame are coincident but rotated, the transformation can be simply described by Equation (3)

where "b" denotes the body-fixed frame and "m" denotes the MTA-model frame. The transformation from the inertial frame to the MTA-model frame contains physical offsets but also requires calculating the kinematics of the MTA arm and will be the discussion of Section 2.3.

$$\begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix}_b = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix}_m \tag{3}$$

2.3 Motion Test Apparatus Kinematics

In this section, we develop both the forward and inverse kinematics of the MTA robotic manipulator, or the geometric relationship between the end effector[1], or IUT, and the angular values robotic joints. The *forward* or *configuration kinematics* describe the Cartesian position of the end effector given the joint angles and lengths of the MTA links [46]. In this formulation of the problem, there is only one solution for the position of the end effector given a specific set of joint values. However, when computing the inverse kinematics, the problem is done in reverse: to calculate the joint angular values given a desired Cartesian position of the end effector. Solving this problem is necessary because it is how the MTA operates: commanding joint angles defined from the desired end effector position. Calculation of the inverse kinematics is more complicated than for the forward kinematics as there is more than one combination of joint values (non-uniqueness) that yields a desired end effector

[1]In robotics, the appliance or device at the end of the robotic arm is called the end effector. Herein, the IUT, or sting location, will loosely be referred to as the end effector.

position for a 6-DOF manipulator.

Forward Kinematics.

Any robotic manipulator is comprised of a set of links held together by joints. In the case of the MTA, a simple 1-DOF motor controls each of the six joints: the torso yaw (θ_{tr}), the shoulder pitch (θ_{sp}), the elbow pitch (θ_{ep}), the elbow roll (θ_{er}), the wrist pitch (θ_{wp}), and the wrist roll (θ_{wr}). For a visual reference, refer to Figure 1.

Because each joint has only one degree of freedom and the links are assumed to be negligibly stiff, simple homogeneous transformation matrices between joints can be used to calculate the overall forward kinematics [46]. This method assigns a kinematic reference frame to the end of each link, coincident with the joints, and then describes the geometric relationship between subsequent reference frames through simple trigonometric and linear operations. There are eight kinematic reference frames overall specific to the MTA, comprised of the MTA-inertial reference frame, the six joints, and the MTA-model reference frame. The eight reference frames for the MTA can be seen in Figure 4.

There is a homogeneous transformation matrix describing the rotation and offset between each set of successive kinematic reference frames shown in Figure 4. The top-left 3x3 square of each transformation matrix contains the rotation from reference frame i to reference frame *(i+1)* with respect to the angular displacement of the joint in degrees where i denotes the kinematic reference frame and not the MTA inertial frame in this case. The fourth column of each transformation matrix corresponds to the displacement in meters between origins of each set of reference frames. The fourth row of each transformation matrix is maintained as [0 0 0 1] so that the matrices are square. For instance, the transformation matrix between the MTA-inertial reference

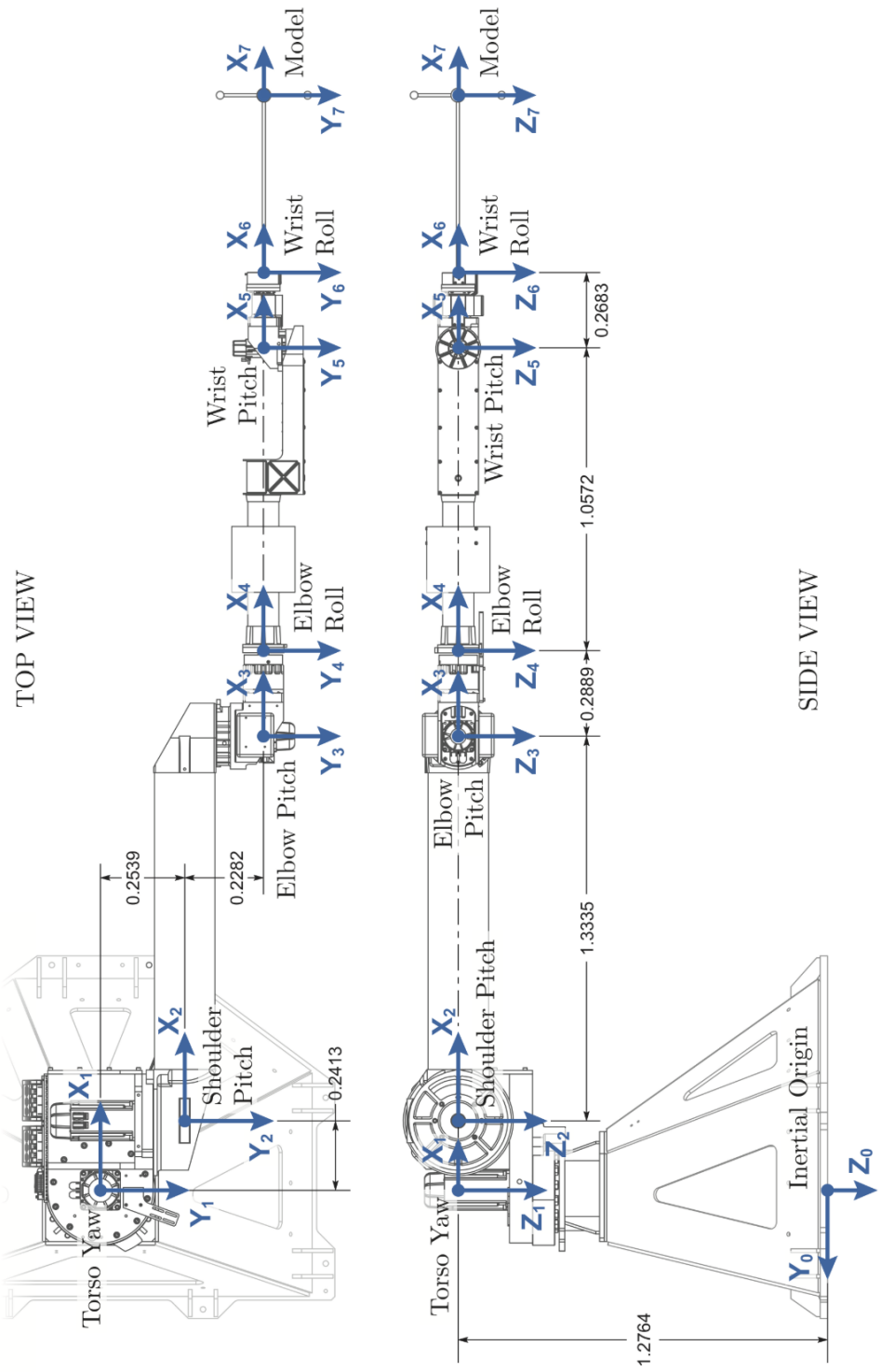


Figure 4. MTA kinematic reference frames. Lengths are in meters, and angular displacements are set to zero. [41]

frame, denoted by the subscript *0* in this section, and the torso yaw joint, denoted by the subscript *1*, is shown below in Equation (4). Premultiplying $[x, y, z]_0^T$ by the first 3x3 of ${}_1^0T$ yields $[x, y, z]_1^T$. The fourth column signifies that the origin of reference frame 1, the torso yaw joint, lies 1.2764 meters above the origin of the inertial reference frame in the z_i-direction.

$$ {}_1^0T = \begin{bmatrix} cos(\theta_{tr} - 90^\circ) & -sin(\theta_{tr} - 90^\circ) & 0 & 0 \\ sin(\theta_{tr} - 90^\circ) & -cos(\theta_{tr} - 90^\circ) & 0 & 0 \\ 0 & 0 & 1 & -1.2764 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4} $$

The homogeneous transformation matrices describing the relationships between the rest of the kinematic reference frames of the MTA are shown below in Equations (5) through (10). It is important to note that the variables in column 4 of ${}_7^6T$ in Equation (10) defined as $model_i$ depend on the dimensions of the specific sting and the test article in use, or the dimensions of the end effector.

$$ {}_2^1T = \begin{bmatrix} cos(\theta_{sp}) & 0 & sin(\theta_{sp}) & 0.2413 \\ 0 & 1 & 0 & 0.2539 \\ -sin(\theta_{sp}) & 0 & cos(\theta_{sp}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5} $$

$$ {}_3^2T = \begin{bmatrix} cos(\theta_{ep}) & 0 & sin(\theta_{ep}) & 1.3335 \\ 0 & 1 & 0 & 0.2282 \\ -sin(\theta_{ep}) & 0 & cos(\theta_{ep}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6} $$

$$
{}^{3}_{4}T = \begin{bmatrix} 1 & 0 & 0 & 0.2889 \\ 0 & cos(\theta_{er}) & -sin(\theta_{er}) & 0 \\ 0 & sin(\theta_{er}) & cos(\theta_{er}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}
$$

$$
{}^{4}_{5}T = \begin{bmatrix} cos(\theta_{wp}) & 0 & sin(\theta_{wp}) & 1.0572 \\ 0 & 1 & 0 & 0 \\ -sin(\theta_{wp}) & 0 & cos(\theta_{wp}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}
$$

$$
{}^{5}_{6}T = \begin{bmatrix} 1 & 0 & 0 & 0.2683 \\ 0 & cos(\theta_{wr}) & -sin(\theta_{wr}) & 0 \\ 0 & sin(\theta_{wr}) & cos(\theta_{wr}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}
$$

$$
{}^{6}_{7}T = \begin{bmatrix} 1 & 0 & 0 & model_x \\ 0 & 1 & 0 & model_y \\ 0 & 0 & 1 & model_x \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}
$$

All of the previously defined sub-transformation matrices can be multiplied as shown in Equation (11) to obtain a single, overall coordinate transformation from the MTA-inertial frame to the MTA-model frame.

$$
{}^{0}_{7}T = {}^{0}_{1}T *^{1}_{2}T *^{2}_{3}T *^{3}_{4}T *^{4}_{5}T *^{5}_{6}T *^{6}_{7}T \tag{11}
$$

Inverse Kinematics.

Calculating the inverse kinematics for the MTA is more complex as there is no general algorithm for doing so with systems of higher DOFs and there is no guarantee for uniqueness. A solution set, or a combination of the six joint angles associated with a desired position and orientation of the end effector, can be achieved in three ways: iterative, algebraic, or geometric methods [10]. Iterative solutions approach a single solution which is highly dependent on the starting position while the algebraic solution does not always yield a closed-form analytic solution. Geometric solutions, contrarily, can beget multiple solutions but must have a closed-form solution for the first three joints to compute a complete inverse kinematic set [46]. The MTA-specific solution method described here was defined in the RE2, Inc. user's manual and adapted by Lancaster for notation purposes [38, 26]. Per this solution method and for reasons of safety, the "elbow-down" solution is preferred.

An input pose, or a Cartesian state describing both the position and orientation of the end effector, can be converted to a homogeneous transform via Equation (12). The variables x, y, z represent the desired model position while the angles ψ (yaw, or rotation about z_0), θ (pitch, or rotation about y_0), and ϕ (roll, or rotation about x_0) represent the input model orientation with respect to the MTA-inertial frame.

$$
{}^{0}_{7}T = \begin{bmatrix}
\cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi & x \\
\sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi & y \\
-\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi & z \\
0 & 0 & 0 & 1
\end{bmatrix} \tag{12}
$$

The first step in defining the equations for the joint angles is to describe the MTA wrist pitch reference frame (ref. frame 5) with respect to the wrist roll reference frame (ref. frame 6). This is done by separating the transformation matrix (${}^{5}_{6}T$)

into a rotational component (${}^{5}_{5'}R$) and a translation component (${}^{5'}_{6}P$) where reference frame 5′ denotes a fictional, intermediate reference frame coincident with the wrist pitch (ref. frame 5) with the same orientation as the wrist roll (ref. frame 6). This separated matrix is shown in Equation (13) below.

$$ {}^{5}_{6}T = {}^{5}_{5'}R * {}^{5'}_{6}P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta_{wr}) & -sin(\theta_{wr}) & 0 \\ 0 & sin(\theta_{wr}) & cos(\theta_{wr}) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0.2683 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13} $$

Equation (13) can be substituted into Equation (11) to obtain another representation of the homogeneous transformation matrix from the inertial reference frame to the MTA model reference frame, expressed in Equation (14) below. Since both ${}^{6}_{7}T$ and ${}^{5'}_{6}P$ are translation matrices representing positive offsets fixed for all time, they are full rank and thus have constant inverses. Multiplying Equation (14) by their inverses in order results in Equation (15) which represents a homogeneous transformation from the inertial reference frame to reference frame 5', or the model's orientation located at the wrist pitch reference frame [26].

$$ {}^{0}_{7}T = {}^{0}_{1}T *^{1}_{2}T *^{2}_{3}T *^{3}_{4}T *^{4}_{5}T * ({}^{5}_{5'}R * {}^{5'}_{6}P) *^{6}_{7}T \tag{14} $$

$$ {}^{0}_{5}T' = {}^{0}_{7}T *^{6}_{7}T^{-1} * {}^{5'}_{6}P^{-1} = {}^{0}_{1}T *^{1}_{2}T *^{2}_{3}T *^{3}_{4}T *^{4}_{5}T * {}^{5}_{5'}R \tag{15} $$

The solution method utilized by RE2, Inc. applies Pieper's solution which states that a closed-form solution for the inverse kinematics of the 6 DOF manipulator exists when three consecutive joint axes intersect at one point [53]. For the MTA, the three consecutive joint angles are the elbow roll, wrist pitch, and wrist roll (ref. frames 4-6). Using Pieper's solution, solving the inverse kinematics for Equation (15) can

be split into two separate problems: computing the first three joint angles (the torso yaw, shoulder pitch, and elbow pitch) in order to achieve the desired end effector position and then computing the last three joint angles (elbow roll, wrist pitch, and wrist roll) to achieve the desired end effector orientation [38].

In order to solve for the lower three joint angles, the geometric offset between the torso yaw joint and the shoulder pitch joint must first be accounted for. The actual dimensions of the torso yaw (ref. frame 1), shoulder pitch (ref. frame 2), and wrist pitch joint (ref. frame 5) are shown in Figure 5 while Figure 6 shows a simple geometric representation of the same configuration but rotated. In this Figure, the offset in the x-direction between the shoulder pitch reference frame and the wrist pitch reference frame is neglected so that their x-axes are collinear. This assumption does not affect the inverse kinematic calculations because both reference frames are oriented similarly [26].

Length b in Figure 6 is a combination of the wrist pitch and shoulder pitch angles and lengths of the corresponding links. Length a is a fixed value of the distance between the origins of the torso yaw joint and the shoulder pitch joint and is calculated per Equation (16) below. Equations (17) and (18) below define the x- and y-locations of the wrist pitch joint with respect to the inertial frame where the value 0.2683 represents the distance in meters between the wrist pitch and wrist roll joints. Length c, the distance between the torso yaw joint and the wrist pitch joint, is then defined per Equation (19).

$$a = \sqrt{(x_2 - x_1)^2 + \left[(y_2 - y_1) + (y_5 - y_2)\right]^2} = \sqrt{0.2413^2 + (0.2539 + 0.2282)^2} = 0.5391 \text{ m} \tag{16}$$

$$x_{wp} = x_m - \big(model_x + [x_6 - x_5]\big) = x_m - (model_x + 0.2683) \tag{17}$$

$$y_{wp} = y_m - model_y \tag{18}$$

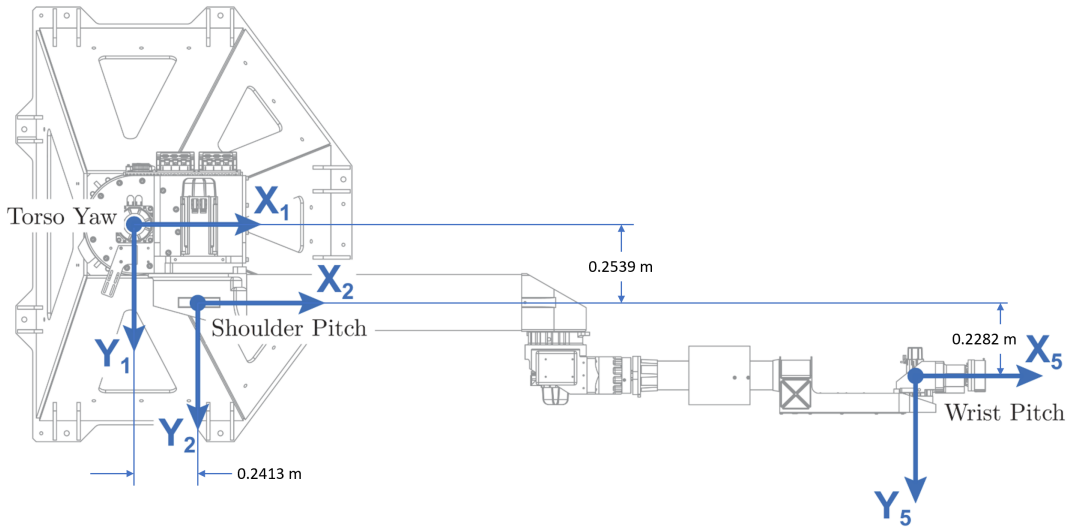


Figure 5. Geometrical definitions of angles and lengths between the torso yaw, shoulder pitch, and wrist pitch joints [41]

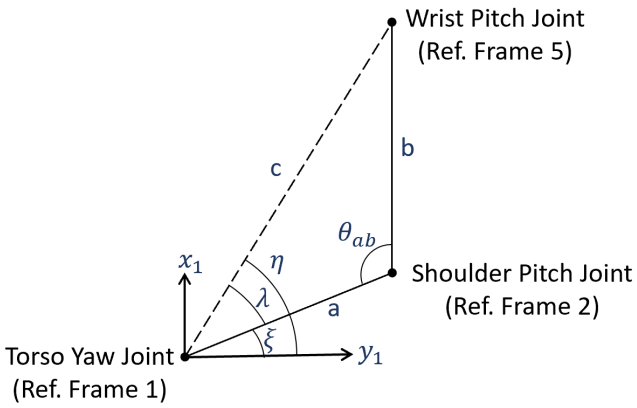


Figure 6. Geometrical definitions of angles and lengths between the torso yaw, shoulder pitch, and wrist pitch joints

$$c = \sqrt{x_{wp}^2 + y_{wp}^2} \tag{19}$$

Angle θ_{ab} in Figure 6 is then calculated using the known distances between the joints as labeled in Figure 5.

$$\theta_{ab} = 180^\circ - \tan^{-1}\left(\frac{0.2539 + 0.2282}{0.2413}\right) = 116.589^\circ \tag{20}$$

With angle θ_{ab} defined, the length c can be defined as in Equation (21) using the law of cosines. The same equation can be rearranged and set to zero as in Equation (22). Equation (22) can then be solved using the quadratic formula for length b, or the distance in the x-direction between the shoulder pitch and wrist pitch joints, as in Equation (23). Because there are two options for the solution to the quadratic equation, only the positive solution for b is chosen for the MTA, as shown in (23), to satisfy constraints due to reality.

$$c^2 = a^2 + b^2 - 2ab\cos\left(\theta_{ab}\right) \tag{21}$$

$$b^2 + \left(2a\cos\left(\theta_{ab}\right)b + (a^2 - c^2)\right) = 0 \tag{22}$$

$$b = \frac{-\left(2a\cos\left(\theta_{ab}\right)\right) + \sqrt{4a^2\cos\left(\theta_{ab}\right)} - 4(a^2 - c^2)}{2} \tag{23}$$

The angles η, λ, ξ in Figure 6 are intermediate angles used to solve for the torso yaw, shoulder pitch, and elbow pitch joints. The calculations for η, λ, ξ are shown in Equations (24), (25),and (26) below, respectively.

$$\eta = \tan^{-1}\left(\frac{y_{wp}}{x_{wp}}\right) \tag{24}$$

$$\lambda = \cos^{-1}\left(\frac{a^2 + c^2 - b^2}{2ac}\right) \tag{25}$$

$$\xi = \eta - \lambda \tag{26}$$

The x- and y-locations for the shoulder pitch joint with respect to the torso yaw joint are then calculated per Equations (27) and (28) below, respectively. With the Cartesian coordinates for the shoulder pitch and the wrist pitch joints previously defined, the torso yaw angle can be calculated per Equation (29) below as the angle of side b with respect to the x-axis.

$$x_{sp} = a\cos\left(\xi\right) \tag{27}$$

$$y_{sp} = a\sin\left(\xi\right) \tag{28}$$

$$\theta_{tr} = -\tan^{-1}\left(\frac{y_{wp} - y_{sp}}{x_{wp} - x_{sp}}\right) \tag{29}$$

In order to calculate the angles for the shoulder pitch and elbow pitch joints, the MTA is regarded as a 2 DOF system once the torso yaw joint angle is calculated. For this calculation, the offset in the z-direction between the shoulder pitch reference frame from the MTA-inertial reference frame is subtracted from the homogeneous transform of Equation (15) to calculate the angle. This is represented in Equation (30). The angle of the elbow pitch joint with respect to the MTA-inertial frame is calculated using the law of cosines as in Equation (31). The length $L_1 = 1.335$ meters is the distance between the shoulder pitch origin to the elbow pitch origin while $L_2 = (0.2889 + 1.0572) = 1.3641$ meters combines the distances between the elbow pitch origin to the elbow roll origin and then to the wrist pitch origin [38]. The inverse cosine in Equation (31) yields both a positive and a negative solution for the joint angle, corresponding to the elbow-down or the elbow-up solution, respectively. The MTA source code selects for an elbow-down solution due to safety constraints, yet both solutions are tracked in the case that the elbow-up solution is optimal for

the desired trajectory [38].

$$z_{sp} = {}^{0}_{5}T'(z) + 1.2764 \text{ meters} \tag{30}$$

$$\theta_{ep} = \cos^{-1}\left(\frac{b^2 + z_{sp}^2 + L_1^2 + L_2^2}{2L_1L_2}\right) \tag{31}$$

The angle for the shoulder pitch joint is then calculated using Equation (32) below where L_1 and L_2 are the same as in Equation (31).

$$\theta_{sp} = -\tan^{-1}\left(\frac{-z_{sp}}{b}\right) - \tan^{-1}\left(\frac{L_2\sin(\theta_{ep})}{L_1 + L_2\cos(\theta_{ep})}\right) \tag{32}$$

Having solved for the torso yaw, shoulder pitch, and elbow pitch angles, the position constraint of the end effector has been satisfied. The next three angles (elbow roll, wrist pitch, and wrist roll) can now be computed in order to achieve the desired orientation of the end effector. To do so, we can substitute the angles $\theta_t r, \theta_e p, \text{and} \theta_s p$ into the transforms of Equations (4)-(6) to obtain ${}^{0}_{3}T$, or the transform from the MTA-inertial frame to the elbow pitch reference frame. Then, ${}^{0}_{3}T$ can be inserted into ${}^{0}_{5}T'$ as in Equation (33) below where ${}^{3}_{5}T'$ represents the transformation from the elbow pitch reference frame to the intermediate reference frame with the model's orientation located at the wrist pitch origin.

$${}^{0}_{5}T' = {}^{0}_{3}T * {}^{3}_{5}T' \tag{33}$$

Equation (33) can be transformed in order to get the equation for ${}^{3}_{5}T'$ in Equation (34) which is made up solely of the variables $\theta_{er}, \theta_{wp}, \text{and} \theta_{wr}$ which correspond to the joint angles for the elbow roll, wrist pitch, and wrist roll, respectively. Equation (34) can be rewritten as in Equation (35) to show the format and individual matrix

elements for use in defining the remaining joint angles.

$$ {}^{3}_{5}T' = {}^{0}_{3}T^{-1} * {}^{0}_{5}T' \tag{34} $$

$$ {}^{3}_{5}T' = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ t_{41} & t_{42} & t_{43} & t_{44} \end{bmatrix} \tag{35} $$

Using the elements of ${}^{3}_{5}T'$, the joint angles for wrist pitch, elbow roll, and wrist roll are defined per Equations (36), (37), and (38) respectively.

$$ \theta_{wp} = \tan^{-1}\left(\frac{\sqrt{t_{21}^2 + t_{31}^2}}{t_{11}}\right) \tag{36} $$

$$ \theta_{er} = \tan^{-1}\left(\frac{t_{21}/\sin(\theta_{wp})}{-t_{31}/\sin(\theta_{wp})}\right) \tag{37} $$

$$ \theta_{wr} = \tan^{-1}\left(\frac{t_{12}/\sin(\theta_{wp})}{t_{13}/\sin(\theta_{wp})}\right) \tag{38} $$

If $\theta_{wp} = 0^{\circ}$, then $\theta_{er} = 0^{\circ}$ and θ_{wr} is calculated per Equation (39) below.

$$ \theta_{wr} = \tan^{-1}\left(\frac{t_{23}}{-t_{22}}\right) \tag{39} $$

Because θ_{wp} has two solutions, a positive and a negative one, there are actually four unique closed-form solutions for the inverse kinematics: ($2\theta_{er}$x and $2\theta_{wr}$). As such, the MTA algorithm checks all solutions to see if any of the angles exceed the range limits. For the solutions that stay within the limits, the forward kinematics are recalculated, and the calculated Cartesian position is compared against the desired Cartesian model position. If the error (Euclidean distance) is greater than the threshold of $1x10^{-6}m$, then the solution is thrown out. The MTA algorithm returns the remaining feasible

solutions for the inverse kinematics but selects the one that has the joints closest to their current position with an elbow-down configuration if possible [38].

All six joint angles can be calculated per the equations defined above for desired Cartesian coordinates of the end effector. Once both the forward and inverse kinematics are established, control schemes for suitable input trajectories can be developed and tested.

2.4 Linear vs. Nonlinear Systems

Extensive literature exists in the realm of analyzing and controlling linear systems, or systems that obey the superposition principle. That is, a change in the output of the system is proportional to the change in input. Due to their inherent predictability, linear systems are relatively simple to understand, solve, and control. They can be solved analytically, lending to a complete understand of their behavior, and numerous mathematical techniques such as Laplace transforms and convolution have been developed in order to modify them.

However, any real system will have counterintuitive or unpredictable nonlinear aspects to it, and superposition does not apply. The behavior of nonlinear systems in general is much more complicated and obscure. The AFIT MTA is no exception. It is clear to see from the kinematic equations of Section 2.3 that the MTA is a highly nonlinear system. The non-uniqueness also creates a significant nonlinearity. Additionally, as shall be discussed in Section 2.5, the dynamic equations of motion for a robotic manipulator of any order turn out to be nonlinear, second-order, ordinary differential equations due to inertial, centrifugal, Coriolis, friction and gravity forces acting on the robotic linkages. Although there are highly established procedures using matrix algebra or frequency domain techniques for calculating control laws for linear systems in general, the same do not exist for nonlinear systems as a whole. Nonlinear

systems also differ from linear systems in that they have the potential for limit cycles (self-excited oscillations, bifurcations (changes in stability as parameters change), and chaos (extreme sensitivity to initial conditions) [45]. As such, nonlinear systems, and specifically nonlinear control, is still a widely researched topic with new techniques currently being developed.

One common technique to solve differential nonlinear equations is to use linearization, or an approximation of the nonlinear system with a linear model [45]. However, the underlying assumption for linearization to hold is that the operating range has to be small enough for the error in the model's output and controller's accuracy to be negligible. Once the operating range is exceeded, the performance of the controller, and thus the system, becomes degraded. Because the MTA only operates within the small wind tunnel test section, the MTA model can be linearized for simple trajectories, as in Lancaster's work [26]. However, as velocities increase or the motions become more complex, the linearized controller may not be able to accurately reduce error from a prescribed trajectory.

While linear systems only have a single equilibrium point if the system is full rank, a nonlinear system can have multiple equilibrium points. Thus, there must be a differentiation made between the local stability of a nonlinear system about one of its equilibrium points or the global stability of the system as a whole. Oftentimes, this can be done using Lyapunov Stability Theory which is comprised of both an indirect and a direct part [40]. While the indirect Lyapunov method uses linearization to evaluate the local stability of an equilibrium point, the direct method uses "Lyapunov functions," a semi-representation of the energy of the system, in order to evaluate the stabilty of a nonlinear system. They are arbitrary scalar functions but usually are defined as the sum of kinetic and potential energy of the system. Lyapunov functions must meet certain conditions, such as positive definiteness, in order to

declare a system stable. Additional conditions must be met to determine different levels of stability, such as global asymptotic stability. In fact, depending on the type of stability that a researcher wants a system to exhibit, a Lyapunov function can be chosen to design a control law such that the system becomes stable after implementing the control law. Santibañez and Kelly present a procedure to obtain Lyapunov functions for the control of robotic manipulators that will be used here in to analyze the stability of the control laws used in this research [40].

2.5 Dynamic Model

Although the kinematics of the MTA have already been established, they do not take into account the forces and moments that actually cause motion of the manipulator. The relationship between such forces acting on the system and the ensuing motion caused is called the dynamic model, or the equations of motion. As aforementioned in Section 2.4, the MTA, like any robotic manipulator, has a nonlinear dynamic model in the form of ordinary differential equations. Similar to the kinematics of the system, the equations of motion can be derived in many ways. The classical approach for deriving the dynamic model for robotic manipulators is to use the Euler-Lagrange equations derived from the principle of virtual work [46]. Using Newton's second law will yield equivalent dynamics, but the Euler-Lagrange approach has specific properties that are advantageous when designing control laws [46]. Among these properties are the skew-symmetry property, the passivity property, bounds on the inertia matrix, and the linearity in the parameters property, which will be discussed later in this section. Other derivations of the dynamics include computer-aided approaches, such as the recursive Newton-Euler approach, the Composite-Rigid-Body Algorithm, and the Articulated-Body Algorithm developed at NASA's Jet Propulsion Laboratory (JPL) [16]. Closed-form solutions are not guaranteed with iterative approaches as they are

with the Lagrangian approach, but they are more appropriate when implementing real-time control.

To show the derivation of the Euler-Lagrange equations of motion for a robotic manipulator, the problem of a two-link planar robotic linkage with two revolute joints, or planar elbow manipulator will be considered. The dynamics of the full AFIT MTA could be developed here, but they will be an extension of the same form as a two-link planar example. Because the 2 DOF dynamics are a simple, similar version of the 6 DOF dynamics, if usefulness of a control law on the two-link example is proven, the control law will be more likely to work well in the 6 DOF model as opposed to lower-accuracy control laws. Additionally, many control laws in current literature have been developed for decomposed models of 6 DOF robotic manipulators as opposed to the overall dynamics due to the complexity involved with a full model [48].

As shown in Figure 7, the elbow manipulator has two joint angles, or two degrees of freedom, denoted by q_i for $i = 1, 2$. The mass of link i is denoted by m_i, the length of link i is l_i, l_{ci} is the distance from the joint to the center of mass of link i, and I_i is the moment of inertia of link i through its center of mass. There are also two control inputs or joint torques, hereafter referred to as τ_i, located at the corresponding joints.

The derivation assumes that the input torques act directly on the joints, the links are modeled as homogeneous rectangular bars where the inertia tensor is aligned with the principal axes, the centroid of the links is located on center lines through joints, the axis of rotation does not vary, and q_i and $\dot{q}_i$ are measured relative to the individual pivots. Since the derivation is based off the rigid-body assumption, the natural sources of flexibility and elastic joints inherent in a real manipulator system are not included. Additionally, acceleration due to gravity is measured along negative y-axis according to the reference frame in Figure 7 [8].

To begin the derivation of any mechanical system using the Euler-Lagrange ap-

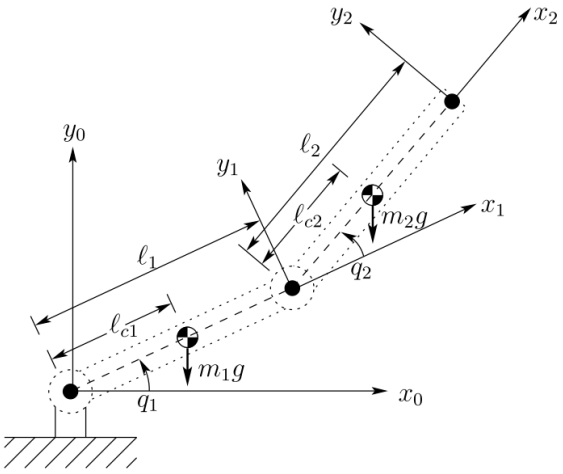


Figure 7. 2 DOF model of robotic manipulator with relevant dimensions identified [46]

proach, one must first define the Lagrangian, $\mathcal{L}$, as the difference between the total kinetic and potential energy of the system:

$$\mathcal{L}(q, \dot{q}) = T(q, \dot{q}) - V(q) \tag{40}$$

Using the Lagrangian, we can obtain the equations of motion, or the Euler-Lagrange equation, with Equation (41) below where Q_i is the generalized, non-conservative force with respect to q_i, or the torque input at each joint.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = Q_i \quad i = 1, ..., n \tag{41}$$

The total kinetic energy is shown below in Equation (42) where $\boldsymbol{v}_{ci}$ is the translational velocity of each link, and $\boldsymbol{\omega}_i$ is the angular velocity of each link, and I_i is the centroidal inertia tensor for each link [8].

$$T = \sum \left(\frac{1}{2} m_i \boldsymbol{v}_{ci}^T \boldsymbol{v}_{ci} + \frac{1}{2} \boldsymbol{\omega}_i^T I_i \boldsymbol{\omega}_i \right) \tag{42}$$

To get Equation (42) in terms of q_i and $\dot{q}_i$, we can define the translation velocity with J_{ci}, the 2x2 Jacobian matrix relating the centroid velocities to the joint velocities, as in Equation (43). Note that $\dot{\boldsymbol{q}} = [\dot{q}_1 \ \dot{q}_2]^T$.

$$\boldsymbol{v}_{ci} = J_{ci}\dot{\boldsymbol{q}} \tag{43}$$

Because the joint variables, $\boldsymbol{q} = [q_1 \ q_2]^T$, are also the generalized coordinates, the Jacobians are easily defined below in Equations (44) and (45) following the derivation in [46].

$$J_{c1} = \begin{bmatrix} -l_{c1}\sin q_1 & 0 \\ l_{c1}\cos q_1 & 0 \\ 0 & 0 \end{bmatrix} \tag{44}$$

$$J_{c2} = \begin{bmatrix} -l_1\sin q_1 - l_{c2}\sin(q_1+q_2) & -l_{c2}\sin(q_1+q_2) \\ l_1\cos q_1 + l_{c2}\cos(q_1+q_2) & l_{c2}\cos(q_1+q_2) \\ 0 & 0 \end{bmatrix} \tag{45}$$

In short form, the total translational kinetic energy of the two-link system is as follows:

$$T_{trans} = \sum\left(\frac{1}{2}m_i\boldsymbol{v}_{ci}^T\boldsymbol{v}_{ci}\right) = \frac{1}{2}\dot{\boldsymbol{q}}^T\{m_1\boldsymbol{v}_{c1}^T J_{c1}\boldsymbol{v}_{c1} + m_2\boldsymbol{v}_{c2}^T J_{c2}\boldsymbol{v}_{c2}\}\dot{\boldsymbol{q}} \tag{46}$$

To calculate the rotational portion of the kinetic energy, the angular velocities of the links are calculated as below in Equation (47) where $\boldsymbol{k}$ is the z_0-axis of the reference frame.

$$\boldsymbol{\omega}_1 = \dot{q}_1\boldsymbol{k}, \qquad \boldsymbol{\omega}_2 = (\dot{q}_1 + \dot{q}_2)\,\boldsymbol{k} \tag{47}$$

Because the angular velocities of both links are aligned with k, I_i in Equation (42) becomes $I_{zz,i}$. Then, the term $\boldsymbol{\omega}_i^T I_i\boldsymbol{\omega}_i$ in Equation (42) reduces to the form shown in

Equation (48).

$$T_{rot} = \sum \left(\frac{1}{2} \boldsymbol{\omega}_i^T I_i \boldsymbol{\omega}_i \right) = \frac{1}{2} \dot{\boldsymbol{q}}^T \left(I_1 \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} + I_2 \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \right) \dot{\boldsymbol{q}} \tag{48}$$

Thus, substituting Equations (46) and (48) into Equation (42) yields the form below, where $\boldsymbol{H}$ is known as the inertia matrix. Some trigonometric identities were used [46].

$$\begin{aligned} T(q, \dot{q}) &= \frac{1}{2} H_{11} \dot{q_1}^2 + H_{12} \dot{q_1} \dot{q_2} + \frac{1}{2} H_{22} \dot{q_2}^2 \\ &= \frac{1}{2} [\dot{q_1} \ \dot{q_2}]^T \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix} \begin{bmatrix} \dot{q_1} \\ \dot{q_2} \end{bmatrix} \end{aligned} \tag{49}$$

where the coefficients H_{ij} are shown below.

$$\begin{aligned} H_{11} &= m_1 l_{c1}^2 + I_1 + m_2 (l_1^2 + l_{c2}^2 + 2 l_1 l_{c2} \cos q_2) + I_2 \\ H_{22} &= m_2 l_{c2}^2 + I_2 \\ H_{12} &= H_{21} = m_2 (l_{c2}^2 + l_1 l_{c2} \cos q_2) + I_2 \end{aligned} \tag{50}$$

The total potential energy of both links is as follows where g is the constant of acceleration due to gravity:

$$V = V_1 + V_2 = \{ m_1 g l_{c1} \sin q_1 \} + \{ m_2 g l_1 \sin q_1 + l_{c2} \sin (q_1 + q_2) \} \tag{51}$$

We can now form and differentiate the Lagrangian to obtain the different terms

in Equation (41). For the first joint where the input is τ_1,

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial q_1} &= -[m_1 g l_{c1} \cos q_1 + m_2 g l_{c2} \cos q_1 + q_2 + l_1 \cos q_1] \\
\frac{\partial \mathcal{L}}{\partial \dot{q}_1} &= H_{11}\dot{q}_1 + H_{12}\dot{q}_2 \\
\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_1} &= H_{11}\ddot{q}_1 + H_{12}\ddot{q}_2 + \frac{\partial H_{11}}{\partial q_2}\dot{q}_1\dot{q}_2 + \frac{\partial H_{12}}{\partial q_2}\dot{q}_2^2
\end{aligned} \tag{52}$$

After some calculation, it can be shown that the Euler-Lagrange equation takes on the form below [8].

$$\boldsymbol{H}(q)\ddot{\boldsymbol{q}} + \boldsymbol{C}(q,\dot{q})\dot{\boldsymbol{q}} + \boldsymbol{G}(q) = \boldsymbol{\tau} \tag{53}$$

where $\boldsymbol{H}(q)$ is previously defined in Equation (50) and the rest of the coefficient matrices are defined below.

$$\begin{aligned}
C_{11} &= -m_2 l_1 l_{c2} \sin(q_2)\dot{q}_2 \\
C_{12} &= -m_2 l_1 l_{c2} \sin(q_2)(\dot{q}_1 + \dot{q}_2) \\
C_{21} &= m_2 l_1 l_{c2} \sin(q_2)\dot{q}_1 \\
C_{22} &= 0 \\
G_1 &= -[m_1 g l_{c1} \cos q_1 + m_2 g l_{c2} \cos q_1 + q_2 + l_1 \cos q_1] \\
G_2 &= m_2 g l_{c2} \cos q_1 + q_2 + l_1 \cos q_1 + q_2
\end{aligned} \tag{54}$$

The system shown in Equation (53) forms a Multi-Input Multi-Output (MIMO) set of coupled 2^{nd}-order, nonlinear differential equations. The terms in Equation (53) represent motion due to gravity (any term with a g), centrifugal effects ($\dot{q}_i^2$ terms), and Coriolis effects ($\dot{q}_{ij}^2$ where $i \neq j$ terms) [46]. If friction is added to the system, an additional term $\boldsymbol{F}(\dot{q})$ can be added to the left hand side of Equation (53) where the terms in $\boldsymbol{F}$ are the subject of current literature and depend on the accuracy of

the model to be obtained [12, 39]. To write the system into state-space form, one can premultiply every term in Equation (53) by $\boldsymbol{H}^{-1}$ to obtain equations for $\ddot{\boldsymbol{q}}_i$.

As previously mentioned, there are a few useful properties that become clear from using the Lagrangian formulation of the dynamics of the system. Refer to [46] for proofs of these properties. One property is that $\boldsymbol{H}(q)$ is a symmetric, positive-definite matrix, and the quantity $(\dot{\boldsymbol{H}} - 2\boldsymbol{C})$ forms a skew-symmetric matrix. This property is important in proving stability via Lyapunov analysis when designing control laws and guarantees the existence of $\boldsymbol{H}^{-1}$. Another property of the system in Equation (53) is called passivity, which means that the total energy of the system has a lower bound. When a system has all revolute joints, as in the model in Figure (7) and in the MTA, there are also lower and upper bounds on the inertia matrix. That is, there are constant bounds of the eigenvalues of the inertia matrix independent of $\boldsymbol{q}$ that can be found. The fourth property is called the linearity of parameters, which means that the dynamics can be written as a linear function of a combination of the parameters, such as masses and lengths as in [45]. That is, the parameters can be rewritten into the form below where the function $\boldsymbol{Y}(q, \dot{q}, \ddot{q})$ is called the regressor, and $\boldsymbol{\Theta}$ is the parameter vector.

$$\boldsymbol{H}(q)\ddot{\boldsymbol{q}} + \boldsymbol{C}(q, \dot{q})\dot{\boldsymbol{q}} + \boldsymbol{G}(q) = \boldsymbol{Y}(q, \dot{q}, \ddot{q})\boldsymbol{\Theta} = \boldsymbol{\tau} \tag{55}$$

Typically, finding the combinations of parameters to linearize the system in the described way is difficult, but for a two link planar manipulator, the parameters that comprise the vector Θ can easily be found in literature [46]. The parameters are shown in Equation (56) where Θ_1, Θ_2, and $\boldsymbol{\Theta}$ are the groupings of the inertia terms and Θ_4, and Θ_5 are grouped from the gravitational torques [46]. The set of parameters shown in Equation (56) is not unique, and the parameters can be grouped in other ways while still maintaining the form of Equation (55) [23, 45, 47, 51].

$$
\begin{aligned}
\Theta_1 =& m_1 l_{c1}^2 + I_1 + I_2 + m_2(l_1^2 + l_{c2}^2) \\
\Theta_2 =& m_2 l_1 l_{c2} \\
\Theta_3 =& m_2 l_1 l_{c2} \\
\Theta_4 =& m_1 l_{c1} + m_2 l_1 \\
\Theta_5 =& m_2 l_2
\end{aligned} \tag{56}
$$

With the parameters described as shown in Equation (56), the regressor matrix is calculated as follows in Equation (57). The matrix $\boldsymbol{Y}(q, \dot{q}, \ddot{q})$ is assumed to be completely known since $q, \dot{q}$, and $\ddot{q}$ are all measurable values.

$$
\boldsymbol{Y}(q, \dot{q}, \ddot{q}) = \begin{bmatrix}
\ddot{q}_1 & \cos(q_2)(2\ddot{q}_1 + \ddot{q}_2) + \sin(q_2)(\dot{q}_1^2 - 2\dot{q}_1\dot{q}_2) & \ddot{q}_2 & g\cos(q_1) & g\cos(q_1 + q_2) \\
0 & \cos(q_2)\ddot{q}_1 + \sin(q_2)\dot{q}_1^2 & \ddot{q}_2 & 0 & g\cos(q_1 + q_2)
\end{bmatrix} \tag{57}
$$

Linearizing the dynamics in terms of the parameters in this way has a few advantages. For instance, this approach can be used in system identification as when the the regressor matrix was experimentally determined at sample configurations for a PUMA 762 robotic manipulator in order to identify the physical mass, inertia, and friction properties [51]. Methods for stability analysis including limit cycles exist in the literature [40, 12].

The analysis for the 2 DOF elbow manipulator can be extended to any n degree of freedom chain manipulator as long as the kinetic energy is the same form as in Equation (42) and the potential energy of the system is independent of q. The MTA satisfies both these conditions, so the general form of the dynamics shown in Equation (53) applies to the AFIT MTA. Likewise, if the designed control laws are stable for the 2 DOF elbow manipulator (proven via Lyapunov analysis), then the AFIT MTA will also be stable for those control laws, in theory, because the same properties apply

[28].

2.6 Control Methods

There are numerous hierarchies for categorizing robot control. One source defines the control of robotic manipulator based on the goal: path planning, trajectory generation, and trajectory tracking [46]. Path planning involves computing the end effector's motion while trying to reach a specific position, or goal point, with constraints, such as avoiding physical collisions. Trajectory generation is different in that it takes into consideration the time history of the individual joints and their velocity and acceleration limits. Trajectory tracking is the problem of minimizing the error between the prescribed path and the actual path that the robot takes. The subject of this research is mainly trajectory tracking, which involves the use of feedback, or closed-loop control.

Robotic control can also be defined by the use of open-loop control or closed-loop control. An open-loop control system is a system where its output has no bearing on how the system performs. Contrarily, a closed-loop controlled system is one where the error between the output and reference input drives system behavior. A nominal system with closed-loop negative feedback is shown in Figure 8 where disturbances are also introduced to the plant, or the open-loop system.

Another more specific categorization method is to classify industrial robots based

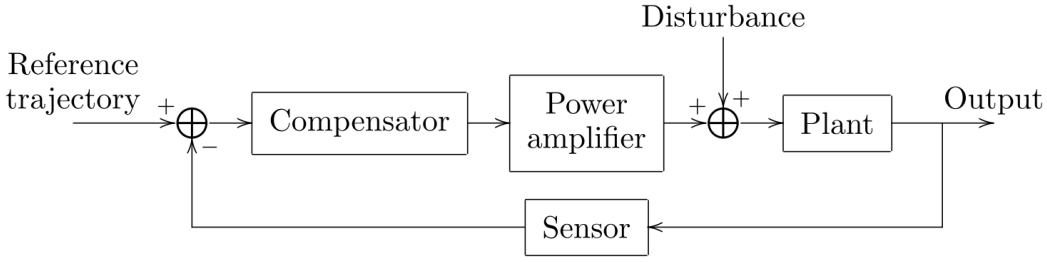


Figure 8. Basic architecture of a closed-loop system [46]

on the type of control system in use. In this categorization, there are four types: limited sequence robots, point-to-point control robots, continuous path control robots, or intelligent robots [1]. Limited sequence robots, also referred to as "stop-to-stop" or "non-servo" robots, are simple in that they rely on mechanical stops and switches to determine the stopping point for joint motion in place of complex control laws. Typically, they are controlled via open-loop pneumatic systems. Point-to-point control robots have the capability to move from one end effector point to another while storing the path memory. Continuous path control, on the other hand, controls the path from an initial to an end point, typically in a straight line. They can usually stop at any point in the path as the individual waypoints are saved in memory. Intelligent robotic systems can track more sophisticated motion including controlling velocity and acceleration using sensors as feedback.

Additionally, control problems can be classified as either by the size of the system being controlled. For instance, joints can be controlled either individually, treating the coupling as disturbances, or as a multivariate problem. The individual joint-scheme approach simplifies the control problem by treating each joint as a single-input/single-output (SISO) while the multivariate control scheme is better suited for design of more complex systems in need of robust or adaptive control, like the MTA [46]. This would be the multi-input/multi-output (MIMO) case. While the different categorizations of types of control used in robotic system could be emphasized, the rest of this section is divided based upon whether the control derives from the kinematics or the dynamics of the system.

2.6.1 Kinematic Control

One approach to controlling robotic manipulators where joint limits exist is to use kinematic control, sometimes referred to as *joint space control* [28]. This approach

is computationally light compared to other dynamic torque-based methods and thus can be useful for real-time trajectory tracking [7]. It works by creating a mapping from the robot joint space to the task space of the end effector. The joint space, or configuration space, of a robotic manipulator is the set of all potential values of the joint angles, and the task space, or workspace, is the set of all positions that an end effector can reach [28].

Once a representation for the position and orientation of the end effector in terms of the joint angles is known, one can calculate the inverse kinematics and dispense the appropriate amount of torque to get the joint motors to the calculated angles. However, the method of formulating the forward kinematics can also impact the inverse kinematic solution process. For complex geometries, such as a 6 DOF robot arm, the forward kinematics can be computed in many different ways. The method described in the RE2, Inc. Manual, or the canonical homogeneous transform method described above in Section 2.3, uses an arbitrary transformation matrices of three rotations and one translation in order to calculate the forward kinematics from reference frame i to reference frame $(i + 1)$ [38]. However, other methods to describe the forward kinematics of a robotic chain manipulator exist that are more notationally or computationally efficient. They include using the Denavit-Hartenberg transformation or using the product of exponentials method.

The Denavit-Hartenberg (D-H) method was developed by Jacques Denavit and Richard S. Hartenberg in their 1955 paper on kinematic notation for simple joint schemes [13]. This popular method improves upon previous methods by utilizing only four parameters in order to describe the position (and orientation) of each link in a robotic chain link manipulator as opposed to using six parameters: the three Cartesian coordinates and three Euler angles. The Denavit-Hartenberg parameters are link twist (α_i), link length (a_i), link offset (d_i), and joint angle (θ_i). Considering

a link with the reference frames i and $(i-1)$ attached rigidly to its joints, link twist is the angle from the z_{i-1}-axis to the z_i-axis about the x_{i-1}-axis. Link length is the offset distance between the z_{i-1}- and z_i-axes along the x_{i-1} axis. The link offset is the distance from x_{i-1} to the x_i-axis along the z_i-axis. The joint angle is defined as the angle between the x_{i-1}- and x_i-axes about the z_i-axis.

Although the D-H method is common in many textbooks, it does not work for all robotic configurations. For the convention to fully describe the forward kinematics of the manipulator, the x_i-axis must be both perpendicular to and intersect the z_{i-1}-axis. These two conditions must be met in order to guarantee the existence and uniqueness of a homogeneous transform matrix from frame $(i-1)$ to the frame i [46]. Due to the geometry of the torso yaw, shoulder pitch, and elbow pitch joints and the use of the predefined reference frames, these conditions were not met for the MTA. A simple translation matrix was multiplied to the D-H transformation matrices between the appropriate reference frames as a simple solution (see Appendix A). Considering this, the pseudo-Denavit-Hartenberg parameters for each joint of the AFIT MTA are shown in Table 1 for the joint reference frames designated in Figure 2. However, the D-H parameters could be used to create alternate intermediate reference frames for the MTA joints that would satisfy the aforementioned conditions.

Table 1. Denavit-Hartenberg parameters for the AFIT MTA

Joint (i)	**Description**	d_i **[m]**	θ_i **[deg]**	a_i **[m]**	α_i **[deg]**
1	torso yaw	0	θ_{tr-90°	0	0
2	shoulder pitch[2]	0	θ_{sp}	0.2413	0
3	elbow pitch[3]	0	θ_{ep}	1.3335	0
4	elbow roll	0	θ_{er}	0.2889	0
5	wrist pitch	0	θ_{wp}	1.0572	0
6	wrist roll	0	θ_{wr}	0.2683	0

[2]D-H does not account for offsets for adjacent joints in which x_i does not intersect z_{i-1}, such as the 0.2539 m offset between the torso yaw and shoulder pitch joints [46].

Using the D-H parameters, one can use the transformation matrix shown below in Equation (58), which shows the modified Denavit-Hartenberg convention for a revolute joint. There is also a classical version, but the difference is the order of operations. That is, the classical method assumes a rotation and translation about the z_{i-1}-axis and then a rotation and translation about x_i while the modified version describes a rotation and translation about the x_{i-1}-axis and then a rotation and translation about z_i [36]. Both methods will achieve the same kinematics provided that the parameters are chosen correctly. Additionally, both methods will describe the orientation and position of the end effector as a product of all the individual joint-to-joint transformation matrices.

$$
{}^{i-1}_{i}T = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_i \\ \sin(\theta_i)\cos(\alpha_i) & \cos(\theta_i)\cos(\alpha_i) & -\sin(\alpha_i) & -\sin(\alpha_i)d_i \\ \sin(\theta_i)\sin(\alpha_i) & \cos(\theta_i)\sin(\alpha_i) & \cos(\alpha_i) & \cos(\alpha_i)d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{58}
$$

Another way to calculate the kinematics of a robotic manipulator is to use a method called the product of exponentials, first developed by Brockett in 1984 [28]. An alternative to the Denavit-Hartenberg method, product of exponentials relies on using the twist coordinates of each joint to characterize the forward kinematics for a manipulator consisting of revolute, prismatic, or helical joints. That is, each joint is regarded as a screw motion, or a simple rotation and translation, from fixed axes. This simplifies the forward kinematics because only two reference frames are used: the base frame and the tool frame, or, in the case of the MTA, the inertial frame and the model frame. Because of this, there is no need to assign individual reference frames to each link, making the derivation more intuitive as well as increasing the

[3]The same applies to the 0.2282 m offset between the shoulder pitch and elbow pitch joints.

speed of computer processing. Using the product of exponentials formulation has the added benefit that the inverse kinematics can be solved with a canonical, geometric algorithm, known as the Paden-Kahen subproblems as described in [28].

Additionally, using the product of exponentials to derive the forward kinematics leads to a natural, simple method of obtaining the manipulator Jacobian, or the relation between the joint angular velocities and the linear and rotational velocities of the end effector [28]. Traditionally, the Jacobian is obtained by differentiating the forward kinematics with respect to time, but this calculation can be complex. Calculating the Jacobian can give insight to the structure, capabilities, and singularities of the robotic manipulator as well as possibly providing another mechanism to calculate the inverse kinematics.

Many kinematic control algorithms exist for computing the inverse kinematics of the robot in real-time, including using the pseudo-inverse of the Jacobian to obtain the angular rates [52]. The method of using a damped least-squares inverse of the Jacobian has also been studied [29, 50]. These methods work via numerical integration and so are prone to numerical drift and other errors specific to numerical techniques. To overcome this, some researchers have included feedback correction, denoted as closed-loop inverse kinematics (CLIK) algorithms. Such algorithms also include Jacobian inverses, transposes, or damped least-squares inverses even up to second-order kinematics [7].

Kinematic control is often used as a path-planning tool to calculate the trajectory online, using optimization methods to minimize the path deviation [10] or to satisfy other constraints such as torque limits [51]. Optimization solutions range from using neural networks as in [22] to genetic algorithms as in [30] and simulated annealing [18]. With some of these approaches, joint trajectories based on the planning of the Cartesian model trajectory can result in overreaching joint rate and acceleration

limits [51]. [7] circumvents this problem by introducing a time warp when joint limits are encountered to slow down the task-space trajectory.

With the MTA, the main problem is not to calculate the trajectory online, but to gain robustness with respect to all the uncertainties in pose when tracking a predefined trajectory. See Lancaster's work for trajectory design; the code includes functions to check geometric constraints that some of the above-mentioned kinematic control schemes meet [26].

2.6.2 Dynamic Model-Based Control

The main scope of this research is not to plan optimal paths or trajectories for the MTA, but to reduce tracking error of the end effector even when subjected to disturbances from the aerodynamic forces of the wind tunnel. As such, the control laws used on the two-link planar model and subsequently the 6 DOF MTA model will be closed-loop dynamic model-based controllers as opposed to kinematic controllers. Dynamic model based control, as a subclass of robotic control in general, is a diverse and ever-growing field. As such, only a few control law architectures relevant to robotic manipulators will be presented in this section.

The trajectory tracking problem is posed as follows: compute the joint torques so that the actual trajectory matches the desired trajectory in the presence of initial condition error, sensor noise, and modeling errors [28]. In other words, the control law calculates the torque applied at each joint despite disturbances. Having the dynamic model, or even just its structure, is advantageous for the design of many control laws. If given the dynamic model, one can use the inverse dynamics to calculate the torques directly from the desired trajectory, the forward dynamics to compute joint accelerations when the torques are the independent variable, and the inertia matrix to map the accelerations of the joints to the corresponding joint forces [16].

Inverse dynamics can also be used in feedforward control laws, which use the model to predict the system's behavior to increase the transient performance [16]. Feedforward loops often work in tandem with other control mechanisms, such as feedback loops or proportional-derivative (PD) or proportional-integral-derivative (PID) controllers (discussed later) to gain desired system performance quickly while decreasing error [24, 43]. Howevever, feedforward control does require a predictable model with predictable disturbances, things that are not guaranteed for a system like the MTA.

Figure 9 shows a notional example of a feedforward control scheme without any disturbances or uncertainty where $G(s)$ is the transfer function of the plant, $H(s)$ is the transfer function of the compensator, and $F(s)$ is the feedforward path. Additionally, r is the reference input while y is the output signal.

2.6.2.1 PID Control

Proportional control is one of the simplest and most common forms of feedback control. It is a linear feedback control law of the form in Equation (59) below where u_t is the input to the open-loop plant, K_p is a constant proportional gain, and $e(t) = q_{des}(t) - q_m(t)$ is the error between the desired and measured angle [32]. As is clearly evident from Equation (59), proportional control is a linear control law. As mentioned before, this could be useful for small, simple motions of the MTA. In fact, changing the proportional gains of the MTA inside the feedback loops was the subject of Lancaster's

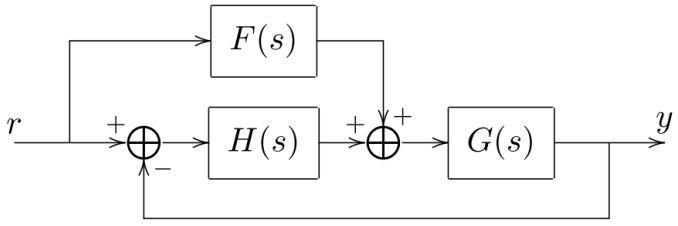


Figure 9. Block diagram of SISO Feedforward controlled system [46]

work [26].

$$u(t) = K_p e(t) \tag{59}$$

Proportional control can also be combined with derivative control and integral control. Integral control changes the controller output, $u(t)$, at a rate proportional to the integral of the error, $e(t)$, and derivative control changes the controller output proportionally to the rate of the error signal. Adding integral control eliminates the steady-state error to a step input that results from using proportional control alone. Derivative control estimates the future error based on its current rate of change. As such, it can be sensitive to sensor noise. Equation (60) below shows the control law for a proportional-plus-integral-plus-derivative (PID) controllers. The first term on the right side of the equation is the proportional control term, and the second term is the integral control term. The third term shows derivative control. T_i is the integral time, and T_d is the derivative time [32]. To apply PID control, a nonlinear system should be linearized about its equilibrium points [43]. However, linearizing the system in this way does mean that stability can only be local, provided the gains keep the linearized system stable [24].

$$\boldsymbol{\tau}(t) = K_p e(t) + \frac{K_i}{T_i} \int_0^t e(\tau) d\tau + K_p T_d \frac{d\left(e(t)\right)}{dt} \tag{60}$$

2.6.2.2 Feedback Linearization

Another common linearization method is called feedback linearization, which should provide better trajectory control than the PD controller because the control law effectively cancels out all of the nonlinear terms of the dynamics, leaving only a linear system behind for which control is highly achievable [45]. Let the control law

for an open-chain robotic manipulator be

$$\boldsymbol{\tau} = \boldsymbol{H}v + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{G} \tag{61}$$

where

$$v = \ddot{\boldsymbol{q}}_{des} - 2\lambda\dot{\tilde{\boldsymbol{q}}} - \lambda^2\tilde{\boldsymbol{q}} \tag{62}$$

In Equation (62) above, the subscript denoted by "*des*" is the desired value of that variable, $\dot{\tilde{q}}$ is defined as the error $\dot{q}_{des} - \dot{q}$, and $\tilde{q}$ is defined as $q_{des} - q$. In this formulation, the control law can be likened to a PD control system, where -2λ is the derivative gain and $-\lambda^2$ is the proportional gain. In general, λ is strictly positive, because substituting τ into the system dynamics to obtain the closed-loop dynamics yields the linear second-order equation below.

$$\ddot{\tilde{q}} + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q} = 0 \tag{63}$$

According to the Routh-Hurwitz criteria, the system will be exponentially stable if $\lambda > 0$, and the position error, $\tilde{q}$ will converge to zero [32].

Feedback linearization works in theory, but in practice it cannot be carried out exactly due to uncertainties in the system and disturbances. Additionally, the control scheme does not take into account frictional forces and torques, which play an important role in the joints of a real system. Similarly, implementing feedback linearization requires measurement of the full state, including angular positions and their derivatives. A robust feedback linearization method with uncertainty can be found in literature where [46]. Another proposed approach is to use the inverse of the analytical Jacobian derived from the kinematics to compute a different version of the variable, ν, in Equation (62) [39]. Feedback linearization can also be combined with feedforward control to create computed torque control laws, which require real-time

estimates of the torque [47]. These estimates can be done with the use of a Kalman filter in some instances [19].

2.6.2.3 Sliding Mode Control

Sliding mode control is another method of control that can be used when there is bounded uncertainty in the system parameters. As such, sliding mode control is more robust than previously described methods. It uses a nonlinear control law that switches from one mode to the next; that is, the dynamics are pushed towards a "sliding surface", or once on that virtual surface, the dynamics approach zero. For a second-order system, this surface is defined as

$$s = \dot{\tilde{q}} + \Lambda \tilde{q} \tag{64}$$

where Λ is a symmetric positive-definite matrix [45]. For a MIMO system, the sliding surface, s in Equation (64) becomes a vector, and s can be rewritten as

$$\boldsymbol{s} = \dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_r \tag{65}$$

where $\dot{\boldsymbol{q}}_r$ is the reference velocity and can be defined as

$$\dot{\boldsymbol{q}}_r = \dot{\boldsymbol{q}}_{des} - \Lambda \tilde{\boldsymbol{q}} \tag{66}$$

In order for the system to remain on the surface, s_i must satisfy the sliding condition below, which makes the surface an invariant set, as defined by Lasalle's Invariance Principle [45].

$$\frac{1}{2}\frac{d}{dt}s_i^2 \leq -\eta|s_i| \tag{67}$$

The sliding mode control law can then be defined as

$$\boldsymbol{\tau} = \hat{\boldsymbol{\tau}} - \boldsymbol{k} sgn(\boldsymbol{s}) \tag{68}$$

where $\hat{\boldsymbol{\tau}} = \hat{\boldsymbol{H}}\ddot{\boldsymbol{q}}_r + \hat{\boldsymbol{C}}\dot{\boldsymbol{q}}_r + \hat{\boldsymbol{G}}$ is the exactly known upper limit on the input torque given the upper bounds, $\hat{\boldsymbol{H}}, \hat{\boldsymbol{C}}$, and $\hat{\boldsymbol{G}}$, on the matrices, $\boldsymbol{H}, \boldsymbol{C}$, and $\boldsymbol{G}$, respectively. With $\tilde{\boldsymbol{H}} = \hat{\boldsymbol{H}} - \boldsymbol{H}$, and so on, k_i is chosen in the vector k to be as follows in order to satisfy the sliding condition in Equation (67).

$$k_i \leq |[\tilde{\boldsymbol{H}}\ddot{\boldsymbol{q}}_r + \tilde{\boldsymbol{C}}\dot{\boldsymbol{q}}_r + \tilde{\boldsymbol{G}}]_i| + \eta_i \tag{69}$$

While sliding mode control is more robust against system uncertainties, it usually introduces a lot of chatter due to the discontinuity of the signum (sgn) term in the control law of Equation (4.1.4) [15]. Chattering can be detrimental to the controller and physical parameters of the system. To combat this, a constant variable to the boundary layer thickness, ϕ, is used so that the control law can be rewritten as

$$\boldsymbol{\tau} = \hat{\boldsymbol{\tau}} - \boldsymbol{k} sat(s_i/\phi_i) \tag{70}$$

where $sat(s_i/\phi_i)$ takes on the value of the sign of s if $s > \phi$ or is s/ϕ otherwise [15]. Furthermore, the boundary layer can be computed as a pseudo-state variable where its derivative is defined as

$$\dot{\phi} = -\lambda\phi + k(q_{des}) \tag{71}$$

Then, the control law becomes

$$\boldsymbol{\tau} = \hat{\boldsymbol{\tau}} - \bar{\boldsymbol{k}} sat(s_i/\phi_i) \tag{72}$$

where

$$\bar{\boldsymbol{k}}_i = k_i - \dot{\phi}_i \tag{73}$$

Adding the time-varying boundary layer has the effect of smoothing out the sliding surface as well as lessening the control needed. Treating the boundary layer thickness as a state variable makes the boundary layer thickness proportional to the error [45].

The control laws presented for sliding mode above are only one representation of an implementation. It does not take into account gravitational or friction terms. Additionally, because there are bounds on the unknown matrices of the system dynamics, the closed-loop system may not be robust against large uncertainties, but they are of interest for use on the MTA.

2.6.2.4 Model Reference Adaptive Control

Model reference adaptive control (MRAC) is a control method that needs no *a priori* information about the model parameters by assuming a known model for the plant and picking the parameters that make the model perform as intended. Then an adaptive law changes the estimated parameters in order to regulate the error between a reference input and the plant output. The basic architecture of an MRAC scheme is shown in Figure 10 where the *adjustment mechanism* box contains the adaptive control law.

In order to use MRAC, the system dynamics must be able to be put into a form that is linear with respect to the system parameters. As discussed, the linearity of parameters property can be extended to any rigid body open-chain link manipulator under the same assumptions. The dynamics represented linearly with respect to combinations of the parameters was already shown for the elbow manipulator in Section 2.5.

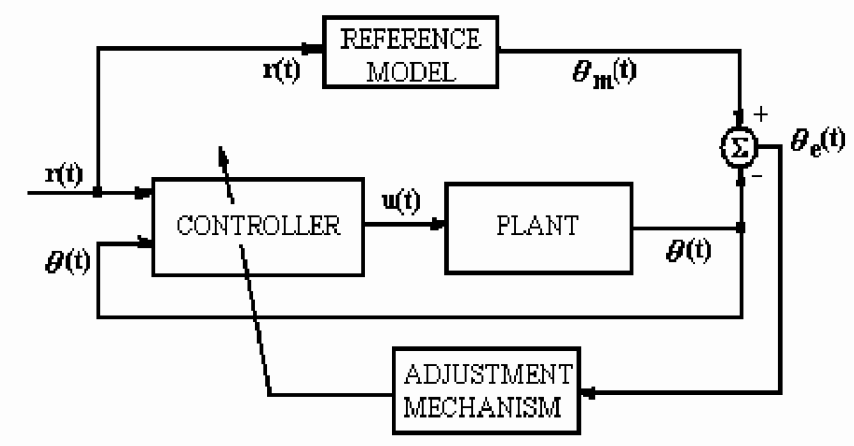


Figure 10. Block diagram of MRAC system [33]

A representative example of MRAC has the form of the control law below

$$\tau = Y\hat{a} - K_D s = \hat{H}\ddot{q}_r + \hat{C}\dot{q}_r - K_D s \tag{74}$$

where the adaptive law is

$$\dot{\hat{a}} = -\Gamma Y^T s \tag{75}$$

where Γ is a user-defined symmetric positive definite matrix, a property that will be necessary in determining stability. Stability analysis of the specific control laws used in this research will be discussed further in Chapter III.

However, for many large-scale systems, MRAC schemes can pose convergence problems and be inefficient [6]. To simplify the MRAC method for robotic control, many researchers decompose the overall robotic system into smaller subsystems with coupling terms. Then, the adaptive law applies to each subsystem with their own respective models. This can be done in many ways, using two subsystems of 3 DOF models to simulate a full 6 DOF robot [48] or using linearized model of each degree of freedom [6]. The subsystems could also be modeled with respect to the control input to take into account the interacting forces and torques [33].

Although MRAC schemes are more robust to uncertainties in the system parameters than regular PID or sliding mode controllers generally, they are only appropriate when the structure of the dynamic system is known and when the parameters are varying with respect to time more slowly than the dynamics of the system can respond [33]. Therefore, MRAC schemes can be used to validate a dynamic model of the MTA, or in system identification of the parameters [51].

2.6.2.5 Model Predictive Control

If the system parameters are changing more quickly than the system or model dynamics, then an MRAC method may not be the most suitable control scheme. As an example, one could institute Model Predictive Control (MPC) which continually solves an open-loop optimal control (OLOC) problem [14]. As such, MPC is similar to feedback control but instead of measuring the error between reference input and output, it measures the error between the actual states and the previously-assumed optimal states through repeating the OLOC problem. There are many versions of MPC: with or without constraints, with varying durations of the prediction horizon, and with different ways to solve the optimization problems. However, the general MPC algorithm is shown in Figure 11.

Model predictive control is most useful for linear systems, but has been implemented on many nonlinear mobile robotic systems such as for autonomous underwater vehicles (AUVs) [44]. Most of the current literature of robotic systems with model predictive control laws are for mobile robots following a trajectory as opposed to a chain manipulator manipulating an object. However, MPC can be used in control of a chain manipulator after the dynamics are initially linearized using feedback linearization or a Taylor series approximation [11]. Additionally, as with MRAC, MPC can be computationally expensive for large-scale systems. Therefore, many control

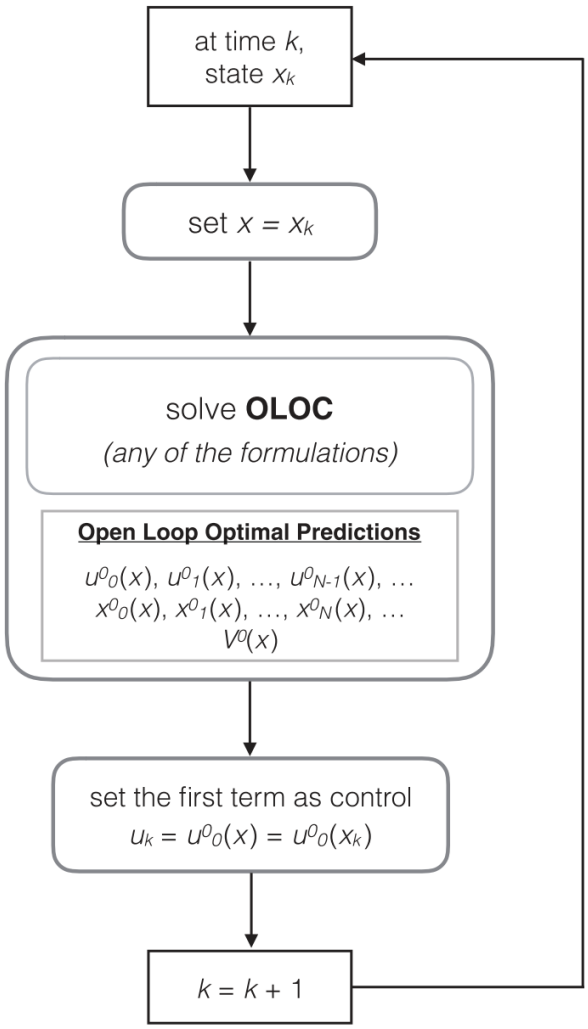


Figure 11. Block diagram of MP system [14]

schemes using MPC are decentralized [14].

2.6.2.6 Other Control Methods

One common way to combat the large computational costs is to combine different control schemes into one closed-loop system. For instance, one control scheme for a three-axis Selective Compliance Assembly Robot Arm (SCARA) manipulator uses adaptive fuzzy sliding mode control to control and stabilize the system [25]. The fuzzy part of the control law uses fuzzy, user-defined logic in order to alleviate the chattering from the sliding mode while the adaptive rule switches the input gains online. Another example of a combined control mechanism uses an inverse dynamic fuzzy sliding mode controller to account for the uncertainties in the nonlinear parameters of a robotic manipulator [35]. The law gives an improvement in performance as opposed to classical inverse dynamics by using the adaptive fuzzy rules to estimate uncertainties in the unknown parameters.

Many factors that go into choosing a control law, such as the models and parameters available, the processing speed of the equipment, and more. The specific control laws used to simulate the two-link elbow manipulator and recommended for use on the 6 DOF MTA model are discussed in Section 3.4 in Chapter III.

2.7 Experimental Methods

As mentioned in Chapter I, wind tunnel testing still plays a vital role in aerospace research today. Theoretically, it can be used to gain performance data or diagnostic data by integrating measured pressure over the appropriate surface area of the model, or it can be used to visualize flow fields [17]. Additionally, wind tunnel testing can be used to directly measure forces and moments acting on a model when using a force balance [17]. Sophisticated force balances usually use electronic strain gauges

in order to calculate the force acting on the object. As such, the balance must be calibrated before and after testing in order to ensure that the correct forces are being measured. Calibration becomes complicated for dynamic tests, as nonlinear effects due to unsteady aerodynamics compound and may not be measurable if the test rig does not accurately track the desired trajectory. Additionally, the data must be normalized for Reynolds number and Mach number effects for wider applicability.

For dynamic wind tunnel testing, it is also important to select a test rig that does not interfere with the measurements or flow field. Most test rigs are single-DOF mechanisms, providing one change in orientation of the model during testing. However, as advances in aerospace research continue to progress, the number of degrees of freedom as well as the number of designs of test rigs is increasing. For instance, some test rigs in current literature are using gimbals sometimes coupled with a compensator acting as a weathervane to simulate free-flight in a wind tunnel [20, 34]. For the MTA, which is an example of a forced-motion rig, the control acts on the robotic manipulator rather than the model itself which creates a challenged in allocating the appropriate control signals.

2.7.1 Experimental Measurements

As mentioned, wind tunnel testing is used to obtain force, moment, and pressure data as well as flow visualization data [17]. Measurements must be time-accurate in order to synchronize the sensor outputs with the motion of the model especially for dynamic tests. Therefore, the MTA must have rapid processing, and the sensors themselves must be able to sample data at a high enough rate to capture the phenomena of interest. Previous testing with the AFIT MTA has used an ATI Nano25 force/torque transducer to directly measure forces and moments as well as a LORD MicroStrain 3DM Inertial Measurement Unit (IMU) to measure the orientation over

time of the model [26, 41, 9]. More information on these specific sensors can be found in Sections 3.6.2.

The forces and moments measured by the Nano25 or any 6 DOF force balance are normal force, side force, axial force, yaw moment, pitch moment, and roll moment. Equation (76) shows the normal force coefficient where F is the force in pounds measured by the Nano25. Equation (77) shows the side force coefficient. Equation (78) shows the axial force coefficient.

$$C_N = \frac{F_x}{\frac{1}{2}\rho V_\infty^2(\frac{\pi}{4}D^2)} \tag{76}$$

$$C_Y = \frac{F_y}{\frac{1}{2}\rho V_\infty^2(\frac{\pi}{4}D^2)} \tag{77}$$

$$C_X = \frac{F_z}{\frac{1}{2}\rho V_\infty^2(\frac{\pi}{4}D^2)} \tag{78}$$

Equation (79) shows the yaw moment coefficient where T is the torque in foot-pounds measured by the Nano25. Equation (80) shows the pitch moment coefficient. Equation (81) shows the roll moment coefficient.

$$C_n = \frac{T_x}{\frac{1}{2}\rho V_\infty^2(\frac{\pi}{4}D^2)D} \tag{79}$$

$$C_m = \frac{T_y}{\frac{1}{2}\rho V_\infty^2(\frac{\pi}{4}D^2)D} \tag{80}$$

$$C_l = \frac{T_z}{\frac{1}{2}\rho V_\infty^2(\frac{\pi}{4}D^2)D} \tag{81}$$

Aerodynamic coefficients are important as a standard for which to compare wind tunnel data across different platforms and laboratories. There are numerous coefficients that can be calculated from dynamic testing; however, this research focuses on quantifying the error due to different control mechanisms. As such, only a few major

aerodynamic coefficients pertinent to dynamic store separation will be discussed in this section including pressure coefficient and the force- and moment-coefficients for the six spatial degrees-of-freedom.

In order to calculate such aerodynamic coefficients, certain dimensional references, or measurements specific to the wind tunnel and model in use, must be ascertained. These dimensional references include cross-sectional area of the wind tunnel, weight of the model, lengths, etc. See Section 3.6 and Appendix B for specific information about the test fixtures and model used in this research.

Pressure measurements are especially important for experiments involving store separation because the bay creates a low dynamic pressure region inside the bay while high dynamic pressure flows past the bay [9]. Equation (82) shows the pressure coefficient.

$$C_p = \frac{P - P_\infty}{\frac{1}{2}\rho V_\infty^2} \tag{82}$$

These measurements and calculations will be necessary to experimentally validate the closed-loop dynamic model of the MTA.

2.8 Chapter II Summary

Chapter II covered data and theory pertinent to designing the control system of the AFIT MTA. Relevant reference frames and coordinate transformations were defined, and the forward and inverse kinematics were shown. Additionally, the chapter covered important details about the differences between linear and nonlinear systems. The dynamic model of a two-link elbow manipulator and its relevance to the MTA were also presented. An overview of potential control methods for the MTA based on robotic manipulation and current topical literature was also covered along with a brief review of experimental data and methods relevant to wind tunnel testing. The next chapter describes the specific methodology used in this research.

III. AFIT MTA System Description

3.1 Motion Test Apparatus Design

3.1.1 MTA Arm Manipulator

The Motion Test Apparatus was initially designed to act as a 6-DOF robotic manipulator for use in an open-section wind tunnel forced-motion rig testing. It is primarily made of aluminum and steel, weighing approximately 1500 lbs and with a steel base with dimensions 46x60x33 inches [9]. Due to the MTA's limited operating space at AFIT and large capability, there is a safety fence surrounding it. The fence must be closed in order for the MTA to operate, and there are emergency shutdown switches on the fence as well as next to the MTA controller computer as in Figure 12. Additional safety measures include mechanical stops to limit the over-rotation of the joints as well as angle and rate limits built into the software to prevent wear of the hardware and contact with the safety fence.

Primarily, the MTA operates by moving a test article along a prescribed trajectory described by user input waypoints using a synchronized combination of the six joint motors. Each motor is also integrated with a digital encoder to keep track of angular orientation of the joint. The motors and their respective controllers are detailed in Table 2. The ELMO® controllers receive velocity commands from the MTA computer which uses a cubic spline to calculate the velocities from the Cartesian waypoints (refer to Section 3.4 for more information) [38]. The MTA software has been updated by Neya Systems, LLC, since the work of Lancaster [26] to include angular feedback control from the motor encoders using position-time scripts. Work by Sellers [41] and Bower [9] reflect the use of the newer software but only for the wrist pitch and wrist roll motors. Further information about the MTA components can be found in the user's manual [38] or detailed in Lancaster's thesis [26].

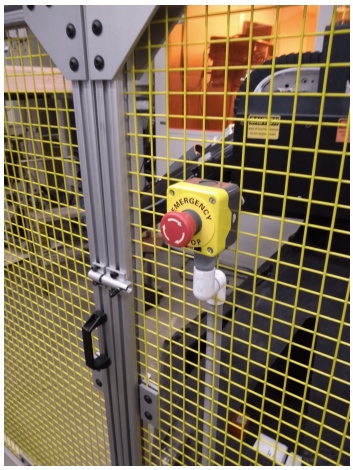


Figure 12. Emergency button on MTA safety fence

Table 2. MTA joint hardware components [41]

Joint	Motor	Controller	Gearbox
Manufacturer:	Kollmorgen[1]	Elmo	Onvio[2]
Torso Yaw	AKM65N	Drum HV (G-DRU-A35)	DM10090
Shoulder Pitch	AKM65N	Drum HV (G-DRU-A35)	DM08055
Elbow Pitch	AKM52K	Trombone (G-TRO6.1)	DN05078
Elbow Roll	AKM22E	Trombone (G-TRO6.1)	DN03055
Wrist Pitch	AKM22E	Trombone (G-TRO6.1)	DN03055
Wrist Roll	AKM22E	Trombone (G-TRO6.1)	DN02015

3.1.2 MTA Control System

The MTA system also includes a Linux-powered controller computer, hereafter referred to as the MTA computer [9]. The MTA computer stands adjacent to the wind tunnel as shown in Figure (13) and is directly connected to the MTA manipulator and performs all the necessary calculations to transcribe coordinates into motor motion. Specifically, since the software update from Neya Systems, LLC, the angular velocity commands, calculated with input from the motor encoders, are sent to the MTA joint controllers. The encoders, in turn, output angular orientation data that is used to

[1]EnDat Absolute Encoder with each BLDC motor
[2]Zero Backlash

Figure 13. MTA computer in subsonic wind tunnel laboratory at AFIT

calculate the next set of angular velocity commands. There are no velocity sensor on the joints. The degree to which the MTA actually follows the prescribed trajectory then depends on many factors including calculation speed, the speed and degree at which the forces and moments on the IUT change, and the accuracy of the angular position encoders. The control laws in use will be discussed more in Section 3.4.

Previously, trajectory command files were input to the MTA by uploading them as *.txt* files to a Linux-powered laptop. Since Fall of 2018, the MTA computer can be accessed via an ethernet cable connected to a desktop in the AFIT Low-Speed Wind-Tunnel Laboratory. The desktop uses PuTTY, an open-source terminal emulator, to access the Linux server of the MTA via a secure shell (SSH) connection. The *.txt* trajectory files can be uploaded to the Linux server through PuTTY. The format of the trajectory files depends on the control systems in use, either from the Neya Systems, LLC, software update or from the adaptive control laws developed from this work. Refer to Section 3.3 for further details on the formatting and content of such trajectory files.

3.2 MTA Operation

The procedures for operation of the MTA outlined in this section are detailed in the RE2, Inc. User's Manual for the MTA [38]. Further description of the operating procedures can also be found in the theses of Lancaster, Sellers, and Bower [26, 41, 9].

To begin operation, the MTA computer must be connected to the host-computer via an ethernet cable. To power it on, a key is used to unlock the controls on the MTA computer. Then, the power switch can be turned to the "on" position, indicated by a green light, which routes power through the MTA computer to the MTA. After a short time, the operator can log onto the host-computer and connect to the MTA computer through PuTTY. To do so, one must enter the commands outlined in the left hand column of Table 3.

Table 3. Linux login commands

Command	Description
`ssh root@10.10.10.10`	Executes SSH protocol for MTA computer to login
`ls`	Lists all files in current directory
`cd re2mtav3`	Changes current directory to version 3 of MTA source code

The MTA computer has the static IP address of 10.10.10.10, so the first command of Table 3 should not change. The third command in row 3 of Table 3 changes the linux directory to whatever is listed after `cd`. The table lists only one possible directory; however, past directories that may not be compatible with the input trajectory files may still be listed. Once the correct directory is selected, the researcher can use the command `ls` to view all the files and functions in that directory. This is where all the kinematics and control laws are stored as functions, along with the desired trajectory files (refer to Section 3.3).

The directory also includes four basic commands that are used in any version

of the code which execute the motion of the MTA, as listed in Table 4. The first command of Table 4 brings the MTA to the first position listed in the trajectory file specified. The Linux input would be typed as "`./mtaHome trajFile.txt`," where `trajFile.txt` is the desired trajectory file. The second command, `mta` would then execute the whole trajectory file, starting from the first position listed, typed as "`./mta trajFile.txt`."

Table 4. MTA motion commands

Command	Description
`mtaHome`	Moves MTA to first point of specified trajectory file
mta	Executes specified trajectory
`mtaAngles`	Gives joint angles at current position of MTA
`mtaMoveTo`	Moves joints to specified angular positions

Since the software update by Neya Systems, LLC, the version of the directory used in Bower's work, `re2mta_rollpitch`, is slightly different [9]. In the directory, there are two forms that desired trajectory files can take: *home trajectory* and *dynamic trajectory*. A home trajectory file brings the MTA to the starting position of a particular dynamic test by specifying the position and orientation of the IUT at each time step. A dynamic trajectory file is then executed after the home trajectory file to run particular tests. All trajectory files have a fixed time interval of 0.008 seconds between way points per manufacturer design [38]. As an example, the homing file would be labeled "`startPos.txt`" and the trajectory file would be named "`movePos.txt`." The corresponding commands would then be input as "`./mtaHome startPos.txt`," which homes the MTA to the starting position of the predefined trajectory file, and "`./mta startPos.txt movePos.txt`," which executes the trajectory file. The two types of trajectory files which are described in more detail in Section 3.3 below.

The last two commands in Table 4 are used as safety checks and for programming

trajectory files. For instance, the command `mtaMoveTo` moves the MTA from its current position to a specified set of angular positions. One example is to move from a starting position of a trajectory file to its final position. Executing a single motion such as this allows the researcher to become aware of any motion that is outside the MTA's reachable space or could possibly cause damage to the equipment, such as exceeding velocity limits of the motors. In essence, the researcher is testing any potentially-damaging movements that the MTA might incur so that the trajectory file can be redesigned to avoid such damage.

Likewise, the command `mtaAngles` outputs the angular positions in degrees and radians for the current position of the MTA. This information can be used to design new trajectory files by allowing the operator to manually move the MTA to and from desired positions and just reading the joint angles from the command.

During execution of a trajectory file, the MTA records angular position at each time step. The velocity at each time step is then computed by the MTA by dividing the difference in position between two points in space by the corresponding time interval. The position and velocity data can then be saved and copied to the host-computer after the MTA concludes its motion, at which time the joint motors are disengaged and the joint brakes are engaged.

To save data after a test run, input the following commands: "`cd Desktop`", and "`scp root@10.10.10.10:/root/re2mta_rollpitch/*.csv`." Again, the password is *mtare2*. After executing the above commands, the files *MtaCartPos.csv* and *MtaRawAngles.csv* will appear on the desktop and should be saved or renamed before executing another test run.

At all times during operation of the MTA, the red emergency stop button should be located next to the Linux host-computer. Pressing any of the emergency stops during operation of the MTA will disable its motion. Likewise, typing `Cntrl-C` will

also exit the program.

To shutdown the system normally, type `shutdown now` before turning the switch on the MTA computer to the OFF position. Refer to the RE2, Inc. User's Manual for further information [38].

3.3 Desired Trajectory Files

3.3.1 Format

A home trajectory file has seven columns in order to manipulate all of the joints into the correct orientation. Column 1 indicates time in seconds with the 0.008 time interval. Columns 2-5 indicate the body-fixed Cartesian coordinates of the model, x_b, y_b, and z_b, with respect to the MTA-inertial reference frame origin expressed in meters. Columns 5-7 indicate the roll, pitch, and yaw orientation of the model with respect to the MTA-inertial reference frame expressed in radians. A sample home trajectory file can be shown below in Table 5 where the column headers are only shown for reader reference.

Table 5. Sample Home Trajectory File [38]

Time	X	Y	Z	Phi	Theta	Psi
0.0000	-0.0640	-2.5120	-1.7930	0.0000	0.0000	-1.6718
0.0080	-0.0640	-2.5110	-1.7930	0.0000	0.0000	-1.6718
0.0160	-0.0640	-2.5100	-1.7930	0.0000	0.0000	-1.6718
⋮	⋮	⋮	⋮	⋮	⋮	
0.0640	-0.0643	-2.5030	-1.7930	0.0000	0.0000	-1.6718
0.0720	-0.0644	-2.5029	-1.7930	0.0000	0.0000	-1.6718
0.0800	-0.0645	-2.5019	-1.7930	0.0000	0.0000	-1.6718

A dynamic trajectory file has only three columns indicating (1) time, (2) the pitch of the wrist joint in radians (WP), and (3) roll of the wrist joint in radians (WR). This kind of trajectory file contains coordinates for only wrist pitch and wrist roll

because the software update by Neya Systems, LLC, only included feedback control from the two joints. Only two DOFs were included due to contract support hours for development and also to keep the system simple when the software was first being updated. A sample of the dynamic trajectory file using only the 2 DOF control system is shown below in Table 6.

Table 6. Sample Dynamic Trajectory File [9]

Time	WP	WR
0.0000	-0.28200	0.06300
0.0080	-0.27973	0.06300
0.0160	-0.27746	0.06300
⋮	⋮	⋮
0.09844	-0.00254	0.06300
0.99200	-0.00027	0.06300
1.0000	0.00002	0.06300

Although the trajectory files contain only position values, the velocity commanded is not explicitly stated in the files. Rather, it is specified by the difference in angular position between time steps.

Because the focus of this research is to create a control system with less tracking error for all six degrees of freedom of the system, the dynamic trajectory file for experiments testing the new control system will have seven columns as well, including the desired angles for all of the joints, not solely for the wrist pitch and wrist roll motors. The three-column dynamic trajectory files will be used as a basis of comparison for the experiments of this report.

3.3.2 Design

Similar to the path-planning problem, designing a new trajectory file for the MTA to track is not a simple task due to physical constraints of the MTA joint motion

and limited operating space. The original trajectory files for pitch-plunge and pitch oscillation motions were created by Patrick Rowe of RE2, Inc. To simplify the design and creation of new trajectory files, Lancaster created a simple Matlab® program, the code for which can be found in Appendix B of his report [26].

Following Lancaster's program, a user can write a trajectory file with respect to the wind tunnel reference frame that is easily convertable to the body-fixed reference frame via software of the user's choice (refer to Section 2.2). Then, the file, in terms of the body-fixed reference frame, can be loaded into Lancaster's Matlab® script named `Trajectory_Coordinate_Transformation.m` to convert the Cartesian coordinates to the MTA-inertial frame with the appropriate format and time intervals. The same program has a graphical user interface (GUI) interface that allows the user to change the coordinates back to the body-fixed reference frame for post-experiment comparison. The user must define the output file with the extension ".txt."

The GUI also has a tab entitled "Operating Conditions" which allows the user to input the location of the MTA-model reference frame's origin with respect to the MTA-inertial reference frame as the starting location of the model for a specific trajectory. The same tab has inputs for the user to specify the maximum allowable direction in the x-, y-, and z-directions in meters. Because these are user-defined inputs, they are not necessarily consistent with the hardware or software limits for the MTA. Therefore, one must take caution when defining the inputs and when testing the MTA to avoid collision with the wind tunnel walls or window.

The specifics of Lancaster's program are that it ensures that the trajectory file has the correct time step of 0.008 seconds, corresponding to a sampling frequency of 125 Hertz, with a cubic spline interpolation before overlaying the interpolated trajectory with the original trajectory for user verification. After this, the program performs the coordinate transformation before adding the offsets to the MTA-model reference

frame origin. The units of the original trajectory file are also user-defined in the GUI. The program, as mentioned, does not take into account limits imposed by the MTA hardware or software [26].

3.4 MTA Control System

The original control system for the MTA has four steps to obtain the output velocity commands to each motor from the initial given desired trajectory. The first step is to convert the Cartesian coordinates of the trajectory to joint angles using the inverse kinematics. Due to Pieper's solution method and the mechanical configuration of the arm, the inverse kinematics give a closed-form solution for the joint angles if the trajectory is achievable (refer to Section 2.3). If the solution cannot be found, the algorithm reports an error.

The second step of the control algorithm is to calculate the desired angular velocities for each joint at each time step. This simple calculation is performed by first computing the slope between two waypoints, as shown in Equation (83) below where the subscripts, A and B, identify two consecutive waypoints.

$$s_{AB} = \frac{\theta_B - \theta_A}{0.008} \tag{83}$$

Then the slope s_{BC} is calculated between waypoints B and C, which is the next consecutive waypoint. If both s_{AB} and s_{BC} have the same sign, the velocity of the middle waypoint, B, is calculated by averaging the slopes. If two consecutive slopes have opposite signs, then the velocity of B is set to zero according to the algorithm. Similarly, the first and last velocities are set to zero [38].

Once the position and velocity for each waypoint is calculated, the MTA algorithm fits a cubic polynomial between each pair of adjacent waypoints to find a smooth curve

for position. The cubic polynomial takes the form in Equation (84) where t is the interval of time between the start and end waypoint, between zero and Δt.

$$\theta(t) = at^3 + bt^2 + ct + d \tag{84}$$

The coefficients of the cubic polynomial are defined in Equation (85) where the subscript i denotes a starting position or velocity, the subscript $i+1$ denotes the end position or velocity, and Δt is the time step. These coefficients are computed for each neighboring pair of waypoints.

$$\begin{aligned} a &= \frac{-2}{\Delta t^3}(\theta_{i+1} - \theta_i) + \frac{1}{\Delta t^2}(\dot{\theta}_{i+1} + \dot{\theta}_i) \\ b &= \frac{3}{\Delta t^2}(\theta_{i+1} - \theta_i) - \frac{2}{\Delta t}\dot{\theta}_i - \frac{1}{\Delta t}\dot{\theta}_{i+1} \\ c &= \dot{\theta}_i \\ d &= \theta_i \end{aligned} \tag{85}$$

The fourth and final step of the MTA computer's original algorithm is to compute the output velocity. The velocity is computed by the time derivative of Equation (84) to obtain the quadratic polynomial with the same coefficients, as shown in Equation (86) where the time, t, is the interval of time between a pair of waypoints. The output velocity commands can be calculated for any frequency because the parabolic spline is a smooth function.

$$\dot{\theta}(t) = 3at^2 + 2bt + c \tag{86}$$

Each ELMO® controller would then receive the set of velocity commands calculated by the MTA computer and output both the angular position and velocity data from the digital motor encoders for use in feedback control. The measured velocity was calculated by dividing the difference between angular position measurements by the time interval between those measurements. Thus, each joint was controlled as a

single-input multi-output system. Figure 14 shows the control block diagram with position and velocity feedback for each joint system.

As shown in Figure 14, each joint used PD, or proportional-derivative, control with two tunable gains on the position and velocity: K_p and K_v, respectively. According to the MTA RE2, Inc. User's Manual, only the first four joints (torso yaw, shoulder pitch, elbow pitch, elbow roll) were tuned during development to output the lowest velocity error while remaining stable [38]. This was due in part to the inability to test the full velocity ranges of the joints in their test environment, hoping to be able to test and tune all of the joints once installed in the proper laboratory. This is a necessary step due to the nonlinearity of the joint motors, but it did not occur at installation. To remedy this, Lancaster changed the gains for the wrist pitch and wrist roll joints so that the control law would work well as a linear approximation for the trajectories he was testing [26].

Because the system uses independent joint-space control without any coupling between joints, there is no global control system. Due to the local feedback laws, if there is position or velocity error between the desired and measured values in a joint, it will likely compound the error of the other joints to create a larger error in the Cartesian position of the end effector. Furthermore, as the maneuvers and

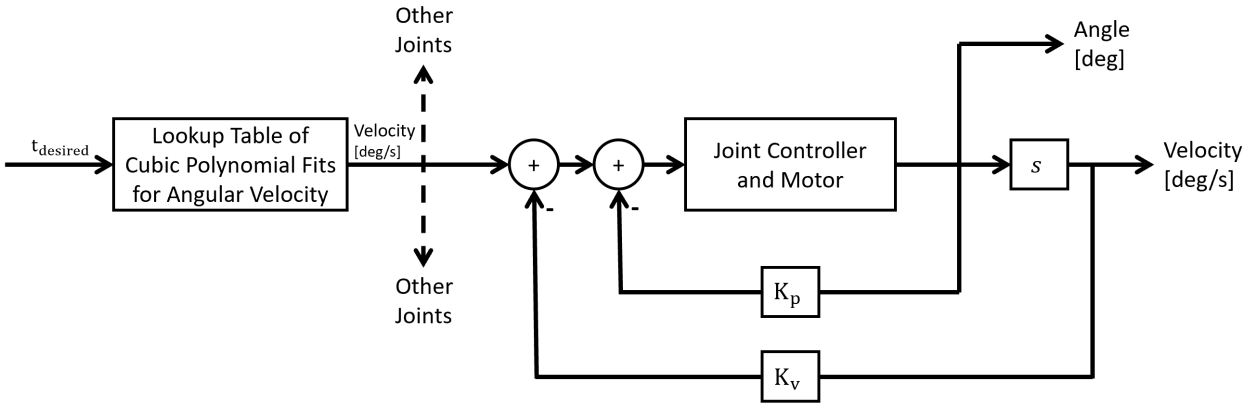


Figure 14. Block diagram of PD control system for each joint

trajectories of the end effector become more complex, the error will compound more because the joint angles will fall outside the linearly-approximated region of the PD controllers. Additionally, Lancaster found that there was hysteresis in the system even during simple maneuvers, such as pitch oscillation using only the wrist roll joint [26].

The MTA source code with the Neya Systems, LLC update was downloaded from the MTA computer, but it was not able to be transcribed into Matlab in order to give a full analysis of the change in control laws. However, it is known that the new software uses motor encoder position feedback for only the wrist pitch and wrist roll motor [9]. Thus, there is still no global control law in terms of the MTA's workspace.

3.5 Simulations

As discussed in Section 2.5, control laws can be tested on the dynamic model of a lower-degree-of-freedom system and still be applicable to the MTA if the derivation includes the rigid body assumption. Therefore, designing and testing a variety of control laws on the elbow-manipulator in simulation first is more efficient than trying to design for and control the global system first. Realistically, however, the MTA links are not infinitely stiff, so there will be differences in the model and simulation results. Thus, the first step was to explore the 2 DOF control methodologies.

3.5.1 2 DOF System

The control laws to be tested on the 2 DOF system in Matlab® are PD with feedforward control, feedback linearization (computed torque), sliding mode control, and model reference adaptive control. The parameters needed for the 2 DOF model are shown in Figure 15.

The mass properties are based off the last two links of the MTA so that q_1 simulates

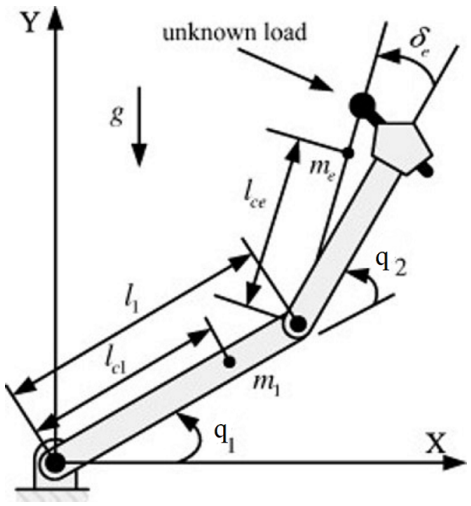


Figure 15. 2 DOF model of robotic manipulator with end-effector uncertainties

the elbow pitch degree of freedom and q_2 simulates the wrist pitch degree-of-freedom. More specifically, they were obtained from the CAD model of the MTA developed by RE2, Inc. and compared to estimated values [5]. The values with the subscript "e" in Figure 15 are a combination of the properties of the second link combined with the properties of the end effector. Because the manipulator is in constant motion when executing a trajectory, the properties denoted by "e" will be time-varying. Additionally, because the end effector is subject to the aerodynamic forces imposed by the wind tunnel, there will be more uncertainty for end effector properties than for the properties of the links. For simulation, they will be modeled as sinusoidal inputs bounded by the maximum measurements for the pitch-plunge movement in Sellers's work [41]. The specific values used are noted in Chapter IV because different control laws have different assumptions about the availability and uncertainty of the system parameters. Chapter IV will also discuss the stability of the different control laws via Lyapunov analysis.

The state vector used in the 2 DOF simulations will be of the form in Equation (87) where q_1 and q_2 are the joint angles, θ_{ep} and θ_{wp}, or the generalized coordinates. The positional derivative states are the angular rates of the joints: $\dot{q}_1$ and $\dot{q}_2$. The

positional states can be converted from Cartesian coordinates via inverse kinematics, but for simplicity and to be able to use the control laws are kept as angles at first. The initial conditions for the simulation are the same as the desired trajectory's starting position to mimic the MTA's operational procedures, unless otherwise stated in Chapter IV. The initial velocities are set to zero.

$$\boldsymbol{x} = [q_1 \ \dot{q}_1 \ q_2 \ \dot{q}_2]^T \tag{87}$$

Once the 2 DOF control system is complete and the model is verified, the same control laws can be extended to the 6 DOF case.

3.5.2 6 DOF System

A Matlab® program was modified to create a simulation of the AFIT MTA on which to test control laws after the 2 DOF system was simulated and verified. The existing code, from MathWorks® File Exchange, was written by Don Riley, a Professor at Walla Wall University, simulates the 6 DOF Programmable Universal Machine for Assembly (PUMA) 762 robot [37]. More specifically, the code animates a 3 dimensional CAD model of the PUMA 762 robot with the forward kinematics calculated from the Denavit-Hartenberg parameters. The animation is depicted in a GUI window with slider controls for the user to input the joint angles, as shown in Figure 16. It also has a "Demo" button that moves the robot randomly via the inverse kinematics, but the function was not animated.

The original PUMA demo code was updated for the MTA graphics and kinematics. Additionally, the model was updated to follow the original control algorithm of the AFIT MTA as described in Section 3.4 to act as a model. During the research herein a number of improvements were identified that could and should be made to the model before using it to simulate control laws. For instance, the model should

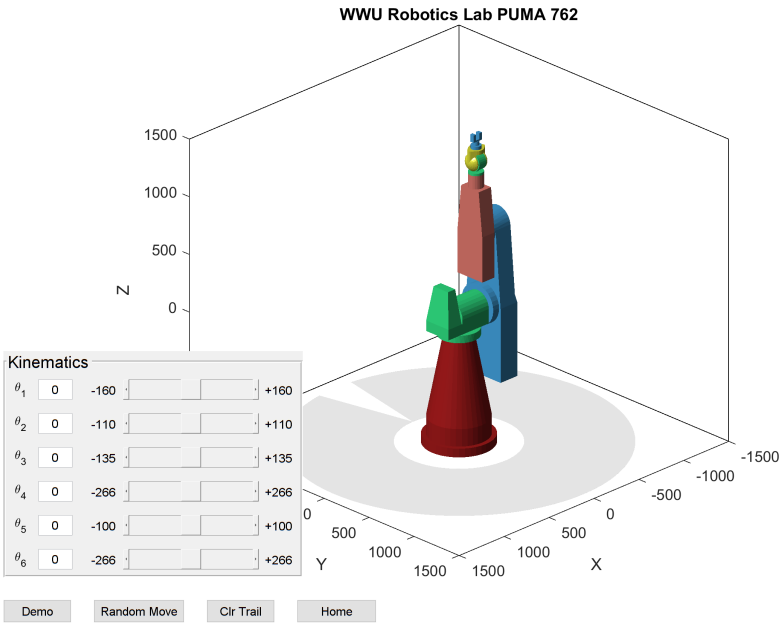


Figure 16. Original Matlab® 3D robotic model of PUMA 762 [37]

include friction, gravity, joint angles, angular velocity limits (software- and hardware-induced), uncertainties, and most importantly the dynamics. Some of these were partly developed, as will be discussed in Chapter IV.

3.6 Experimental Setup

After designing the control laws for the 2 DOF and 6 DOF and running them in simulations, it is important to test them on the actual MTA for model validation as well as verification of the control schemes' usefulness in decreasing error and ensuring robustness.

3.6.1 Wind Tunnel Models and Model Support Sting

The model support sting was designed by Lancaster to attach to the MTA wrist and support the experimental model in the wind tunnel. It was designed so that the quarter-chord, or aerodynamic center, of the model it holds would be in line with the

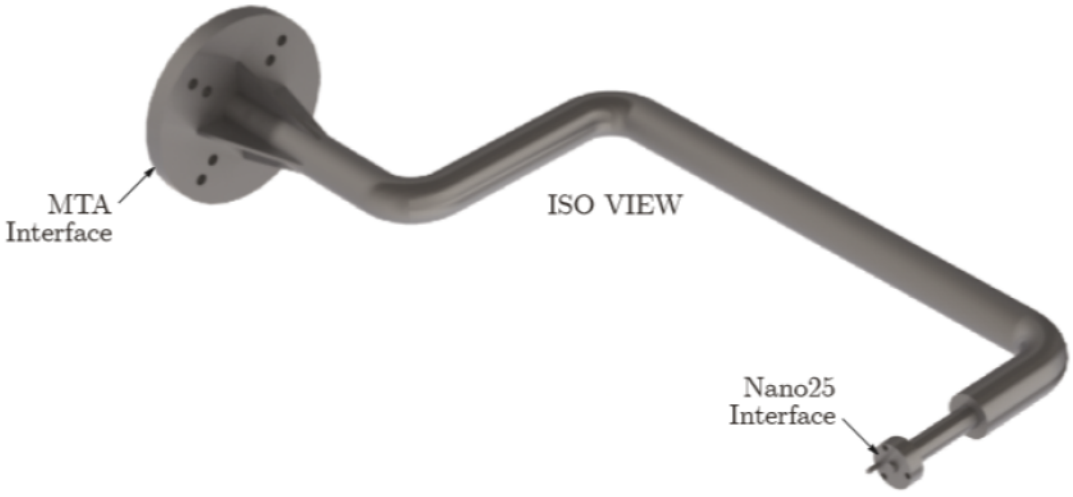


Figure 17. AFIT MTA model support sting [41]

wrist roll axis of rotation. Therefore, the pitch oscillation motion could be performed using only the wrist roll degree-of-freedom. The sting is comprised of a base plate that bolts into the MTA wrist that is welded to a 1-inch diameter piece of stock aluminum with three bends in it, as shown in Figure 17. The sting extends into the wind-tunnel test section through the 9-inch diameter circular hole in the plexiglass and interfaces with the Nano25 sensor as shown in Figure 18. There is an extruded pin on the model interface end that inserts into the Nano25 to ensure the correct alignment. Refer to Appendix B for a more detailed drawing of the model support sting.

Previous research with the AFIT MTA used various wind tunnel test articles (IUT). Lancaster and Sellers both used symmetric NACA 0012 models with an 8-inch span and a 4-inch chord [42]. The difference between the two IUTs were how they interfaced with the sensors; the IUT used in Lancaster's work was designed to attach directly to the sting while the IUTs used in Sellers's work were designed to have space in their fuselages to fit around sensors. One was made to attach onto the Nano25 transducer, and another was made to attach to the AFIT 6 DOF force

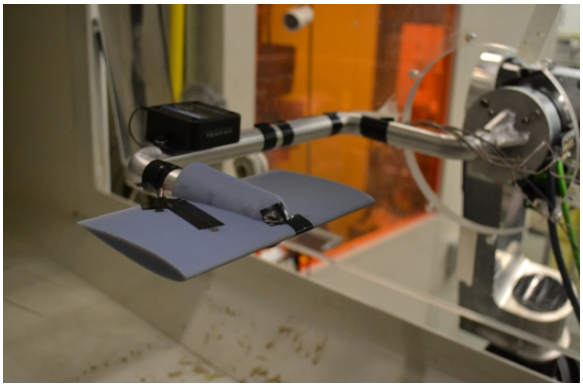


Figure 18. AFIT MTA model support sting in 9 inch diameter cut-out in plexiglass window [41]

balance. All three IUT models were designed so that the quarter-chord is in line with the wrist roll axis of rotation even with the respective sensors inside. To see more detailed drawings of the NACA 0012 IUTs with sensor interfaces, refer to Appendix B. IUTs were also created to mimic store separation models for use in Sellers's and Bower's work but will not be discussed in this research [9, 41].

3.6.2 Sensors and Measurements

As outlined in Chapter II, there are two main sensors in use during testing of the MTA: the ATI Nano25 force/torque transducer and the modified MicroStrain 3DM GX1 IMU. To reiterate, the Nano25 measures the aerodynamic moments and forces on the IUT during testing. The IMU measures the orientation of the model. In the works of Lancaster and Sellers, the IMU had a digital output, but since the work of Bower, both the Nano25 transducer and the IMU transmit analog outputs via voltage signals to the PXIe-6123 DAQ (data acquisition) card. This allows for the sensor measurements to be synchronized.

3.6.2.1 Nano25 F/T Transducer

Built specifically for robotic applications, the ATI Nano25 Force/Torque Transducer offers many benefits as a sensor for the AFIT MTA. It is currently one of the smallest 6-axis transducers worldwide, yet it still maintains its high strength due to being manufactured from high yield-strength stainless steel via electrical discharge machining (EDM) wire-cutting [3]. Because of its high strength, the maximum allowable single-axis overload values range from 7.1 to 15.1 times the rated capacities, detailed in Table 7 below.

Table 7. ATI Nano25 F/T transducer technical specifications [3]

Calibration Specifications		
	Sensing Ranges	Resolution
F_x, F_y	25 lbf	1/224 lbf
F_z	100 lbf	3/224 lbf
T_x, T_y	25 lbf-in	1/160 lbf-in
T_z	25 lbf-in	1/320 lbf-in
Single-Axis Overload		
F_{xy}	±520 lbf	
F_z	±1600 lbf	
T_{xy}	±280 lbf	
T_z	25 ±560 lbf	
Physical Specifications		
Weight	0.14 lb	
Diameter	0.984 in	
Height	0.85 in	

In Table 7 above, the variables $F_x, F_y,$ and F_z represent the measured forces in the x-, y-, and z-directions with respect to the Nano25's orientation, respectively. The variables $T_x, T_y,$ and T_z represent the measured torques about the x-, y-, and z-axes of the Nano25 reference frame, respectively. The variable F_{xy} represents any combination of forces in the x-, and y-directions while T_{xy} represents any combination

of torques about the x-, and y-axes.

The coordinate system about which the forces and torques are referenced in Table 7 is a standard right-hand rule Cartesian coordinate system with the origin located on and at the center of the Nano25's front face where it interfaces with a model. The positive z-axis extends out normal to the face while the x-axis extends in the direction of the signal output cable, as shown in Figure (19). A more detailed drawing by ATI of the Nano25 can be seen in Appendix B.

With respect to the MTA setup, the Nano25 is attached to the MTA via the sting which connects to the rear of the Nano25 on the side opposite the location of the sensing reference frame, or to the mounting side of the Nano25. The sting was designed to place the quarter-chord of a rectangular planform model (4 in chord by a 8 in wing span with symmetric NACA 0012 airfoil) coincident to the wrist roll axis of rotation while the Nano25 is mounted between the model and the sting [26]. Originally, this design was chosen to enable pitch oscillation of the model about the quarter-chord utilizing only 1 DOF of the MTA, to limit interference of the sting with the airflow affecting the model, and to limit collision of MTA links with the wind tunnel components [26]. A detailed drawing of the sting by Lancaster can be seen in Appendix B.

With the Nano25 attached to the sting as described above, the z-axis of Nano25 sensor reference frame aligns with the negative y_m-axis of the MTA-model reference frame, the y-axis of Nano25 sensor reference frame aligns with the negative z_m-axis of the MTA-model reference frame, and the Nano25 x-axis aligns with the positive x_m-axis of the MTA-model reference frame so that the wire extends out the right wind of the model. Applying this transform to the body-fixed reference frame shows that the x-, y-, and z-axes of the Nano25 correspond to the y_b-, z_b-, and x_b-axes of the body fixed frame, respectively.

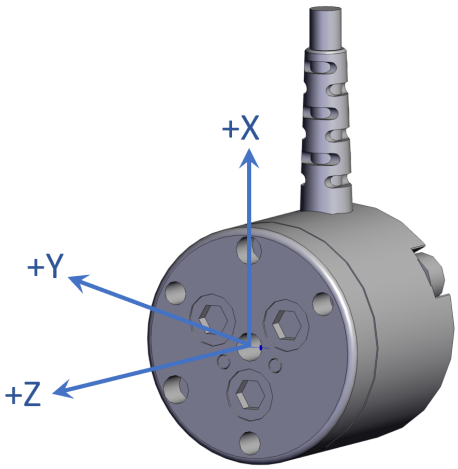


Figure 19. ATI Nano25 sensing reference frame [3]

With this orientation, the positive x-axis aligns with positive side forces, or yaw, applied to the model/sensor, the positive y-axis aligns with negative normal forces, or negative lift, and the positive z-axis aligns with negative axial forces, or drag. Then, $T_x, T_y,$ and T_z represent positive yaw moment, negative pitch moment, and positive roll moment, respectively. Unless otherwise noted in the setup for each experiment of this research, the reference frame for the Nano25 is consistent with the description mentioned above.

The strain gauges of the Nano25 are made of silicon rather than conventionally-used foil; this provides a signal 75 times higher than would be possible solely for foil strain gauges. Because the signal is at a much higher level, the Signal-to-Noise Ratio increases significantly, resulting in a "ear-zero" noise distortion [3]. The Nano25 outputs analog voltages from the aforementioned semiconductor strain gauges through a wheatstone bridge that calculate strain in three internal beams in order to compute the forces and torques applied to the beams. Further details of the Nano25 specifications dependent on the data acquisition system in use are detailed in Section 3.6.2.3.

AFIT originally acquired a Nano25 force/torque transducer to obtain time-accurate

force and moment measurements for dynamic applications as described in Sellers' work for pitch oscillation [41]. AFIT procured a second Nano25 sensor with a signal output wire extending from the back (negative z-axis), diminishing the interference of the airflow from the wire [9].

3.6.2.2 Inertial Measurement Unit

AFIT also acquired a MicroStrain® 3DM-GX15 inertial measurement unit (IMU) for use with the MTA in the subsonic wind tunnel. Its primary use is to measure the Euler angles of the MTA as a verification of the MTA output. Similarly, because it can measure accelerations and angular rates, it can be used for feedback. In Lancaster's work, the IMU was also used to measure the rigidity characteristics of the sting [26]. There are also a plethora of other direct and computed measurements that the 3DM-GX15 is capable of [2].

The 3DM-GX15 IMU operates with the use of Micro-Electro-Mechanical System (MEMS) technology so that it is a lightweight system [2]. More specifically, it contains a triaxial accelerometer and gyroscope paired with on-board processors and an Adaptive Kalman Filter (AKF) to reduce noise and provide accurate measurements [2]. Lancaster's work used a 3DM-GX1, with the major differences being a slower data output rate and larger dimensions [26]. The technical specifications including data output rate for the IMU are listed in Table 8.

Bower also used a 3DM-GX1 IMU in his store separation experiments, but the sensor was manufacturer-modified to give an analog output (at 100 Hz) that can be synchronized with the Nano25 analog outputs [9]. The work of Lancaster and Sellers both used IMUs with digital outputs and had to subtract an initial time offset from the data. Because the sensors were not synchronized, there was more uncertainty introduced in their data.

Table 8. MicroStrain® 3DM-GX1 IMU technical specifications [2]

	Accelerometer	Gyroscope
Range	±5 g	300°/sec
Resolution	< 0.1 mg	< 0.008°/sec
Initial Bias Error	±0.002 g	±0.05°/sec
Sampling Rate	4 kHz	4 kHz
Data Output Rate	1 Hz to 1000 Hz	
Dimensions	36.0 x 24.4 x 11.1 mm	
Weight	16.5 grams	

To measure the orientation of the model, the IMU can be attached to the wrist roll joint of the MTA. Disregarding the flexibility in the MTA links and model sting, the wrist roll reference frame and the MTA model have the same orientation, but attaching the IMU to the MTA joint reduces airflow interference by the sensor. Like the Nano25, the 3DM-GX15 has its own sensor reference frame, with the orienation as defined in Figure 20. A more detailed drawing including the exact location of the sensor origin can be seen in Appendix B.

Data from the IMU can be saved as .lvm files via labVIEW, as described in the next section. Lancaster and Cobb also created a Matlab® GUI to display the Euler angles from the IMU data in real-time adapted from routines provided by the IMU

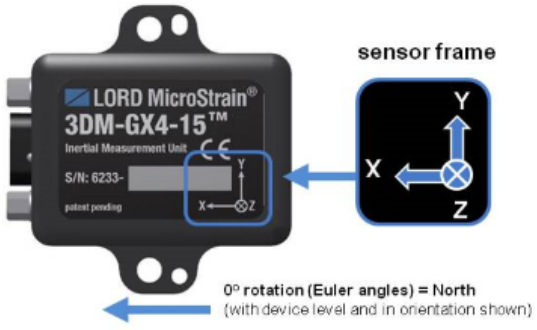


Figure 20. MicroStrain® 3DM-GX1 sensing reference frame [2]

manufacturer [26].

3.6.2.3 Data Acquisition

Past AFIT Thesis efforts researched the implementation of time-accurate force and moment measurements for the AFIT subsonic wind tunnel [26, 41, 9]. As such, only a brief overview of the National Instruments® (NI) data acquisition (DAQ) software package, known as LabVIEW, will be described here. LabVIEW is a system design and measurement platform commonly used for DAQ set up in the form of GUIs or visual block diagrams.

All sensor signals were obtained via a National Instruments® Data Acquisition (DAQ) system located in a PXIe-1078 chassis. The PXIe-8133, an Intel Core i7 embedded controller, is embedded in the PXIe-1078 chassis. Also within the chassis are two PXI-6123 DAQ cards, which allow for simultaneous-sampling of the analog inputs from the sensors.

The Nano25 sensor connects to its own unique signal converter box that also acts as an off-board power supply. Six channels for the six measured forces and moments route from the signal converter box into a TB-2709 terminal block which provides server message block (SMB) connectivity to the PXI-6123 DAQ cards.

The 3DM-GX1 IMU has its own power plug and an analog signal output wire that plugs directly into the TB-2709 terminal block. Only one attitude angle can be recorded for a given trajectory because LabVIEW only records voltage data for the IMU's attitude change about a single axis [9].

LabVIEW records the voltage data from the Nano25 and the IMU as *.lvm* files. The LabVIEW program used in prior MTA experiments can be found in the Appendices of Sellers's work [41]. However, Bower set the sampling rate to 100 Hertz to accommodate the use of the 3DM-GX1 IMU and merged the "while-loops" for the

Nano25 and IMU signals in order to synchronize the data.

3.6.3 Test Plan

At the conclusion of the thesis, no hardware experiments were completed. The control laws were not written into the MTA source code (C++ language from Matlab®), nor was the 6 DOF model complete with its modifications.

Ideally, the plan was to use the same trajectories as used in the 2 DOF simulations for the elbow pitch and wrist pitch motions to validate that model and quantify the actual error. The test model would be the NACA 0012 wing model (IUT) adapted for the Nano25 interface with the 3DM-GX1 IMU attached to the wrist joint as another source of comparison. Ideally, test runs would occur at 30, 60, 90, and 120 miles per hour (mph) for 15 seconds at a time with continuous data collection by the Nano25 and the IMU, based on prior research [41].

Once the error was quantified for the 2 DOF motions, more complex arcing trajectories would be tested on the 6 DOF simulation and on the MTA for a similar range of speeds to validate the Matlab® MTA model and quantify the error based upon different control laws in use. The arcing trajectories would be an imitation of store separation, which entails all 6 joints to work synchronously with one another. Store separation, however, is difficult to test due to the relatively small motions constricted by the test space as compared to the large size of the links [41]. As such, it is likely that there will need to be much more work done on the 6 DOF Matlab® model before being able to track such a complex motion and a modification to the subsonic wind tunnel test section to expand the current 9 inch opening to accomodate larger motion tests.

3.7 Chapter III Summary

Chapter III described the methodology and setup for analysis and experiments relevant to the AFIT MTA control system. The MTA design and operation were detailed, along with a discussion of the format and design of trajectory files used for MTA motion. Chapter III also covered the MTA's kinematic joint-space control algorithms in depth as well as some specific details to include in the 2 DOF and 6 DOF simulations. Finally, Chapter III provided an overview of the experimental setup that can be used for validating the control laws of the aforementioned simulation models. Results from these simulations and planned trajectories for testing the MTA in 2 DOF and 6 DOF motion are presented in the next chapter.

IV. Analysis and Results

This chapter presents the analysis and results from the simulations of the 2 DOF elbow manipulator model and the 6 DOF model in Matlab®. First, the MTA workspace was explored with varying end effector trajectories and the non-unique solutions to the inverse kinematics of those trajectories. Different control laws were tested on the simulations, varying the set of parameters necessary to run the closed-loop system and comparisons between the models were made. The stability analysis of each control method is also presented. Different trajectories were also run in order to observe the difference in system behavior as a function of the input trajectory. This section also describes the steps in improving the 6 DOF model so as to make it more representative of the real MTA system. Experimental results were not completed, but a brief discussion of expected results is examined. Intermediate steps and conclusions drawn from the simulation results are also presented.

4.1 2 DOF Simulation

To reiterate, the 2 DOF elbow manipulator system represents the last two links of the MTA where q_1 represents of the elbow pitch angle of the MTA, and q_2 represents the wrist pitch angle of the MTA. The two links are joined in a chain configuration with the origin coincident with the first joint angle. The elbow manipulator can be oriented in several ways depending on the orientation of the plane formed by the links with respect to the wind tunnel reference frame. Two orientations used in the simulations are when the manipulator is in the y_i-z_i plane and in the x_i-y_i plane, which correspond to when the elbow roll angle is at 0° or 90°, respectively. In the 90° elbow roll configuration, the elbow manipulator model does not include effects due to gravity because the gravitational force is parallel to the axes of rotation for q_1 and

q_2.

The lengths of the links, as with the rest of the mass properties mentioned in Section 3.5, were measured from the CAD model provided by RE2, Inc. [5]. Table 9 shows the values of physical properties including the sting as part of link 2. Again, the parameters denoted by the subscript "e" include mass, length, and inertia of the model in the wind tunnel. Thus, they are time-varying and subject to higher uncertainty. Additionally, using the time-varying end effector mass properties changes the 2 DOF system from autonomous to non-autonomous due to the explicit time-dependence. As such, there are different conditions to declare stability for a non-autonomous system which was not taken into account at the time of this research. The stability analysis in the following sections is valid for the autonomous system only, but the simulations show the non-autonomous system being controlled unless where otherwise stated.

Table 9. Mass properties of 2 DOF system including sting

Parameter	Value
m_1	10.395 kg
m_2	2.355 kg
m_e	$2.355 + 2.268\sin(t)$ kg
l_1	1.354 m
l_2	0.808 m
l_{c1}	0.845 m
l_{c2}	0.161 m
l_{ce}	$0.186 + 0.019\sin(t)$ m
I_1	1.598 kg·m^2
I_2	0.095 kg·m^2
I_e	$0.1295 + 0.1247\sin(t)$ kg·m^2
δ_e	$-10 + 5\sin(t)^\circ$
g	9.81 m/s^2

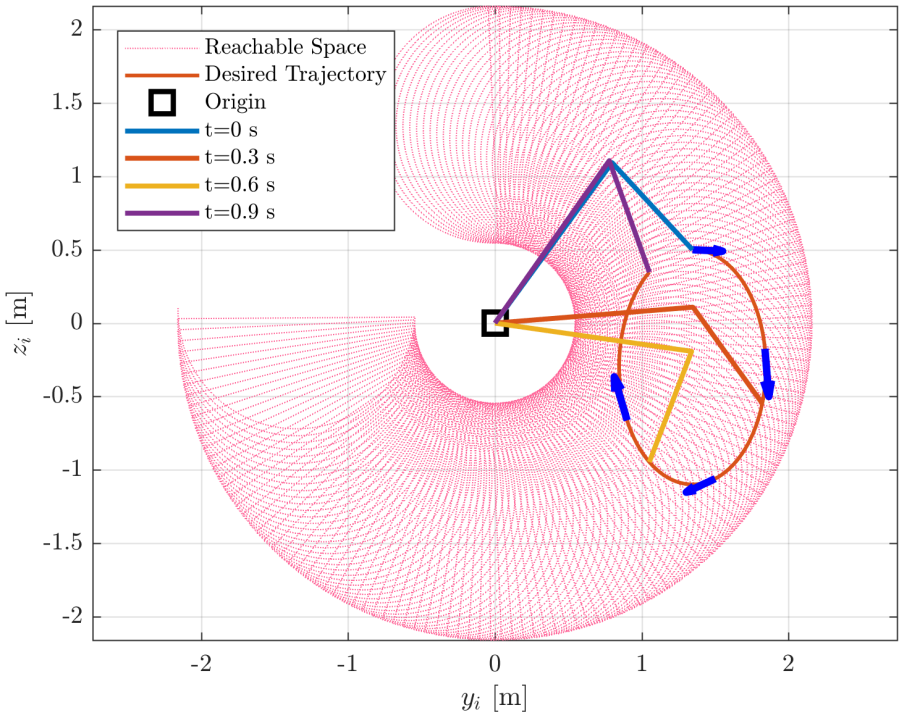


Figure 21. Elbow manipulator reachable workspace

4.1.1 Manipulator Workspace

A manipulator's workspace is the total volume swept out by the end-effector as all possible manipulator DOFs are traversed through their respective limits and is further broken down into a reachable workspace and the dextrous workspace [46]. The reachable workspace is the total set of points that the end effector can reach while the dextrous workspace is the subset of the reachable space that the manipulator can reach with any arbitrary orientation of the end effector. Clearly, the workspace is defined by the physical geometry of the links, but also by the joint variable ranges, which are shown in Table 10. As shown in the table, the joint angle ranges for the elbow pitch joint and wrist pitch joints are -90.3° to 184.5° and -180.4° to -0.6°, respectively. Then, calculating the geometric forward kinematics described in Equation (88) for the grid of reachable joint angles gives the reachable workspace of the elbow manipulator

model as shown below in Figure 21 [28].

$$\begin{aligned} y =& l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2) \\ z =& l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \end{aligned} \tag{88}$$

Table 10. MTA joint ranges defined per the RE2, Inc. User Manual [38]

Name of Joint	Min Angle (°)	Max Angle (°)	Max Velocity ($^\circ/s$)	Max Acceleration ($^\circ/s^2$)	Angular Resolution (°)
Torso Yaw	-93.0	96.2	200.0	359.5	0.0008
Shoulder Pitch	-55.5	89.4	120.0	279.6	0.0008
Elbow Pitch	-90.3	184.5	197.0	253.5	0.00056
Elbow Roll	-249.4	69.3	320.0	799.0	0.0008
Wrist Pitch	-180.4	-0.6	280.0	479.4	0.0008
Wrist Roll	-354.4	370.2	499.0	2929.6	0.0029

Figure 21 also shows an arbitrarily chosen example end effector trajectory defined by Equation (89) that lies within the elbow manipulator's workspace where t is some arbitrary time. This is included to show the position of the manipulator when executing a trajectory. The blue arrows in Figure 21 show the directional velocity of the end effector at those points in time.

$$\begin{aligned} y_{des} =& 1.346 + 0.5 \sin(2\pi t) \\ z_{des} =& -0.3 + 0.8 \sin(2\pi t) \end{aligned} \tag{89}$$

The algebraic inverse kinematics used to calculate the joint angles from the end effector position for an elbow manipulator are presented below. For clarity, the notation shown in Figure 22 will be used. The standard solution for a 2 DOF planar manipulator uses polar coordinates (r, ϕ), as shown in Figure 22 [28]. Using $r = \sqrt{x^2 + y^2}$

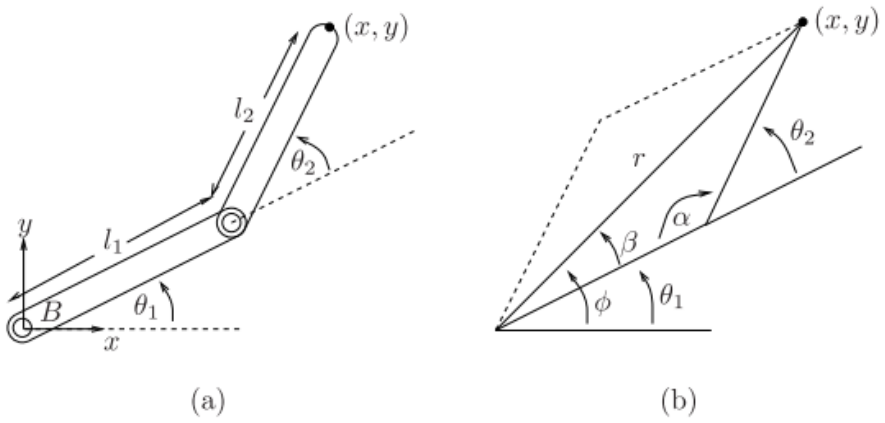


Figure 22. Inverse kinematics for the planar 2 DOF elbow-manipulator [28]

and the law of cosines gives the second joint angle as in Equation (90).

$$\theta_2 = \pi \pm \alpha, \quad \text{where} \quad \alpha = \cos^{-1}\left(\frac{l_1^2 + l_2^2 - r^2}{2l_1 l_2}\right) \tag{90}$$

When α is nonzero, there are two solutions for θ_2, represented by the solid and dashed lines in Figure 22. Then, θ_1 is solved for using both solutions of θ_2 as in Equation (91).

$$\theta_1 = \text{atan2}(y, x) \pm \beta, \quad \text{where} \quad \beta = \cos^{-1}\left(\frac{r^2 - l_1^2 - l_2^2}{2l_1 r}\right) \tag{91}$$

where the sign of β is the same as the sign of α used. Depending on the given variables for the end effector (choosing from x, y, r, ϕ), there can be anywhere from zero to multiple solutions, if the position is in the dextrous workspace of the manipulator. If there are multiple solutions, then all subproblems (i.e. solving for θ_2 first) must be carried through.

For instance, the notional end effector trajectory described in Equation (89) has two solutions for the joint angles. Although trajectory does lie in the reachable space as shown in Figure 21, one of the solutions does not satisfy the joint range constraints. That is, the inverse kinematics of the elbow manipulator does not take

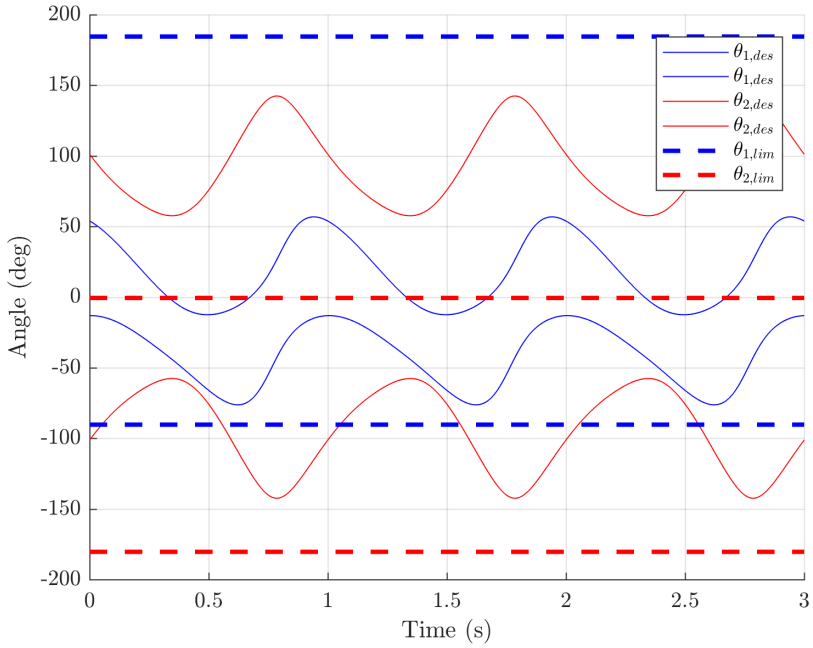


Figure 23. Computed joint angles from trajectory described by $y_{des} = 1.346 + 0.5\sin(2\pi t)$, $z_{des} = -0.3 + 0.8\sin(2\pi t)$

the joint angle limits into account. Figure 23 shows the calculated angles as compared to the MTA joint ranges for elbow pitch and wrist pitch. Thus, the solutions to the inverse kinematics must take the joint limits into account in order to obtain a feasible solution. This was done with logic statements in Matlab® to pick the appropriate set of joint angles (see Appendix A). If a trajectory is input that does not have a closed-form solution for the joint angles, then the script returns an error.

As shown in Figure 23, the end effector trajectory described by Equation (89) does have one set of angles that stays within the limits of the MTA-specified joint ranges. The trajectory both falls within the reachable space and stays within the joint limits of the MTA, so it is feasible. Thus, the trajectory in Equation (89) will be used in simulation. So far, the inverse kinematics script does not take into account the joint velocity and acceleration limits. For simulation, the derivatives of the desired joint angles were determined analytically from the inverse kinematics of the desired

trajectory at first because the points are calculated one at a time through `ode45` in Matlab®. However, because analytically solving the inverse kinematics for each time step is computationally expensive, a look-up table for the specified desired trajectory was used to interpolate the desired joint positions, velocities, and accelerations at each time step of the solver.

However, there is another constraint on the end effector's trajectory, which is the location of the wind tunnel with respect to the manipulator. Because the location of the elbow manipulator origin depends on the first two joints of the MTA, the box representing the confines of the wind tunnel in either axis can be placed at a semi-arbitrary distance to the elbow manipulator origin as long as the arm does not collide with the walls of the wind tunnel or the edges of the plexiglass window. As such, the center of the wind tunnel in both the y_i-z_i and x_i-y_i planes can be specified with respect to the manipulator origin to maximize the reachable workspace inside the wind tunnel while still subject to the physical constraints of all the MTA joints. This is a potential optimization problem to orient all of the MTA joints to maximize the reachable and dextrous space of the last two links. However, it will not be attempted in these simulations.

4.1.2 PD Control

For proportional-derivative control, both the friction and gravitational terms are neglected, for ease and in conjunction with the robot operating in the horizontal x_i-y_i plane. The PD control law shown in Section 2.6.2.1 can be modified to include gravity as in Equation (92) where $\tilde{q}_i = q_i - q_{i,des}$:

$$\tau = -K_P\tilde{q} - K_D\dot{q} + G(q) \tag{92}$$

This control law essentially turns the dynamic system into a mass-spring damper

as in Equation (93) below. When the gains K_p and K_D are positive definite, then the system should behave with damped oscillations.

$$H\ddot{q} + (C + K_D)\dot{q} + K_P\tilde{q} = 0 \tag{93}$$

To assess the stability of the system with the PD controller, the chosen Lyapunov function is below in Equation (94) where the two terms correspond to the kinetic energy and the virtual potential energy associated with the "virtual spring constant", K_P in the control law [40]. V is positive definite because H is always positive definite due to the dynamic model structure, and K_p is defined as a positive definite matrix.

$$V = \frac{1}{2}\left(\dot{q}^T H\dot{q} + \tilde{q}^T K_P\tilde{q}\right) \tag{94}$$

If gravity is neglected in both the dynamics and the control law, the time derivative of the Lyapunov function can be written as shown in Equation (95).

$$\begin{aligned} \dot{V} &= \dot{q}^T(\tau + K_P\tilde{q}) \\ &= -\dot{q}^T K_D\dot{q} \end{aligned} \tag{95}$$

where $\dot{V} \leq 0$ when K_D is independent of q and defined as positive definite. At this point, Lyapunov analysis points to the control law having local Lyapunov stability; however, applying the invariant set theorem (IVT) shows that if $\dot{V} = 0$, then $\dot{q} = 0$. This implies that $\ddot{q} = H^{-1}K_P\tilde{q}$, so for $\dot{V}$ to be zero, $\tilde{q}$ must also be zero [45]. Thus, the system is locally asymptotically stable, and the tracking error should converge.

If gravity is included in the system dynamics and the PD control law as shown in Equation (92), then the Lyapunov function is shown in Equation (92) where $P(q)$ is

the total potential energy of the planar manipulator with a root at $q = 0$.

$$V = \frac{1}{2}\left(\dot{q}^T H q + \tilde{q}^T K_P \tilde{q}\right) + P(q) \tag{96}$$

The time derivative ($\dot{V}$) is shown in Equation (97).

$$\begin{aligned} \dot{V} &= \dot{q}^T(\tau + K_P\tilde{q}) + G(q)\dot{q} \\ &= -\dot{q}^T K_D \dot{q} \end{aligned} \tag{97}$$

where the same logic applies as for the system without gravity. Therefore, the origin is locally asymptotically stable with a PD controller with and without gravity effects.

For the simulation, $K_D = 100I$ and $K_P = 20K_D$. Both are 4x4 positive definite matrices. The simulations were run for three seconds for both a sinusoidal input and step input for the model with gravity included. The sinusoidal inputs are defined by the notional trajectories in Equation (89) above while the desired trajectories step inputs are to 50° for q_1 and $-60°$ for q_2. Figure 24 and Figure 25 show the desired joint angles and the simulated joint angles over time for the sinusoidal inputs and the step inputs, respectively. Figure 26 and Figure 27 show the control inputs and the state errors over time for periodic joint input (1 Hz) and the step inputs, respectively.

As shown in Figure 24, the error does not converge, even when the simulation is run for longer times. This is due to the ineffectiveness of the PD controller to track a high-frequency, high-angle trajectory which probably lies outside the linear range of the controller. Similarly, this corresponds to the notional damped mass-spring-damper system that the closed-loop system approximates; the system dynamics are too slow to react to the input.

In Figure 25, the error over time for step inputs to the trajectories does converge to

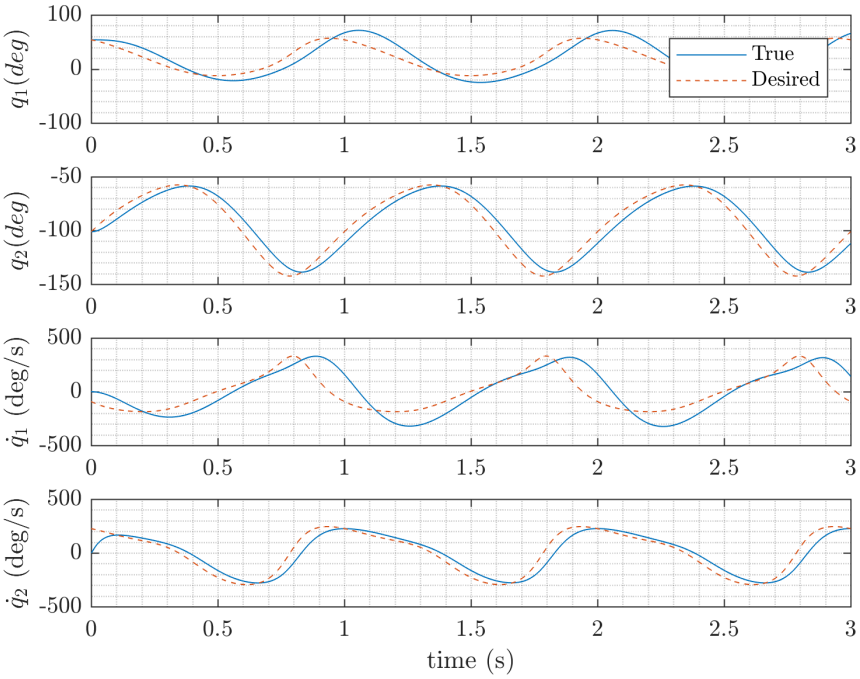


Figure 24. Desired and simulated joint angles over time for periodic joint input (1 Hz) using PD control

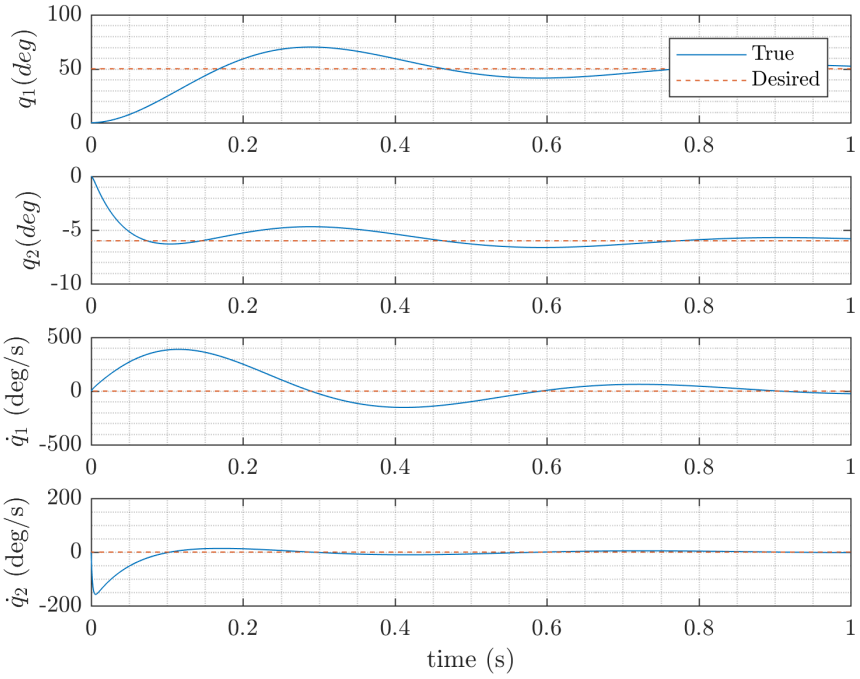


Figure 25. Desired and simulated joint angles over time for step input using PD control

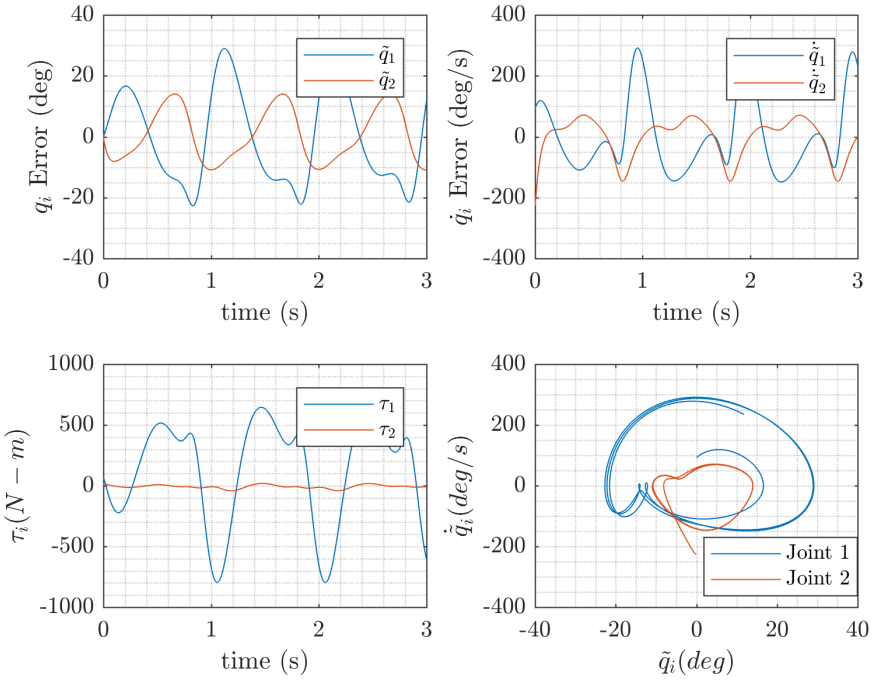


Figure 26. Control input and error in state variables over time with periodic joint inputs (1 Hz) using PD control

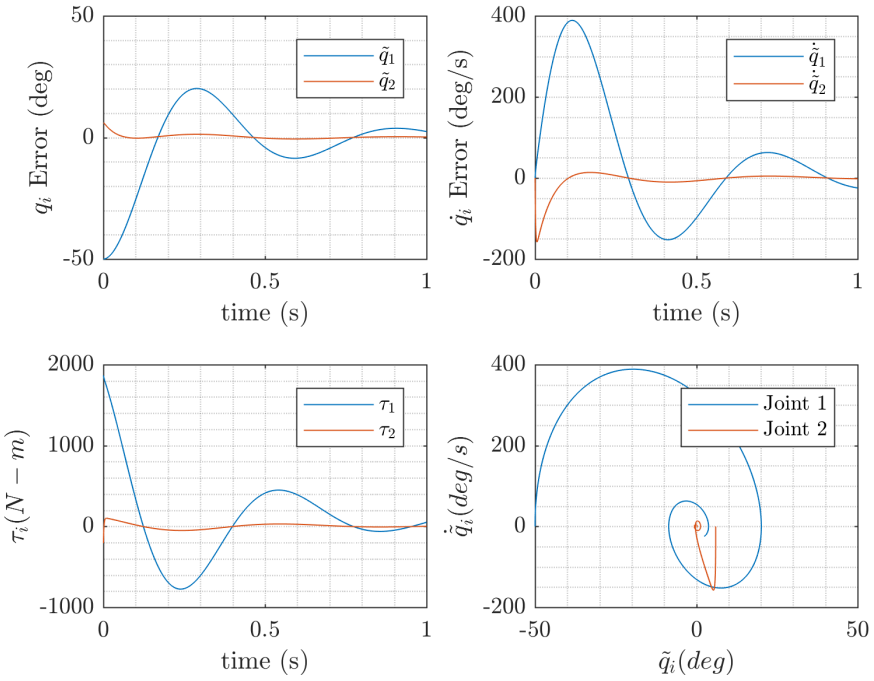


Figure 27. Control input and error in state variables over time with step input using PD control

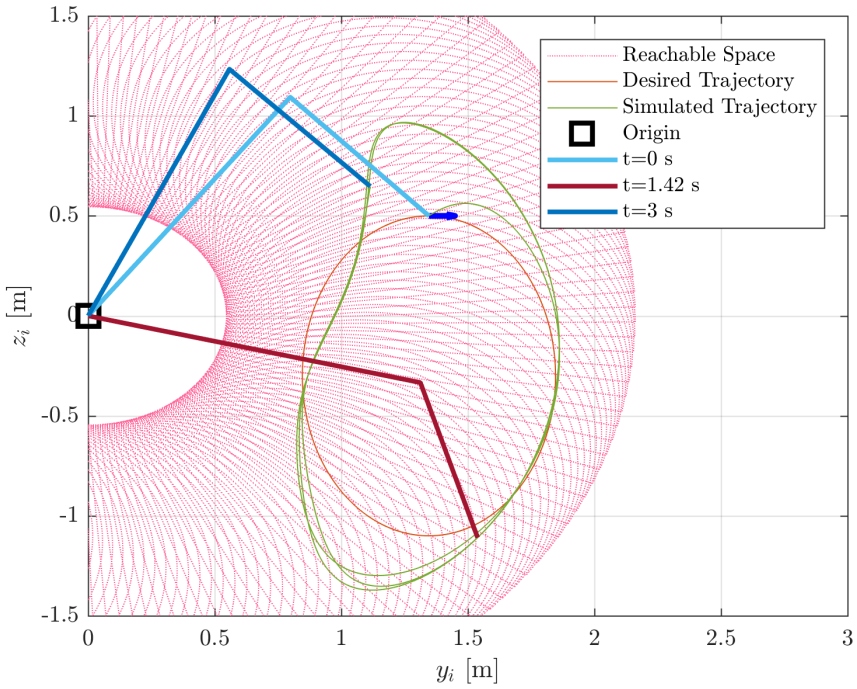


Figure 28. Comparison of desired end effector trajectory and simulated trajectory using PD control with a 1 Hz input

a steady-state value if given enough time. It does not converge to zero due to the lack of integral control in the system and because of the nature of proportional trying to track a step input. However, it does react like an underdamped mass-spring-damper system with respect to the oscillatory value of the torque input and the states' error over time.

The sinusoid trajectory drawn out by the end effector is shown in Figure 28. Clearly, the PD controller is better for point-to-point robots in need of position control rather than trajectory control, even without disturbances and uncertainties in the system. Additionally, the torque used to execute the motion is very large. Computed torque is not currently an output of the MTA system, but 500 Nm is just under 4500 in-lbs, about halfway in the range of the rated torques for the brand of gearbox used in the MTA joints according to Lancaster's work [4, 26]. Thus, the simulated torques seem reasonable for a simulation of the MTA system if not for the actual MTA.

However, acceleration and velocity limits of the joint motors might be strained.

A study of the maximum error as the end-effector input trajectory varied with respect to frequency was planned, but the computation time limited the utility of the simulations even with interpolation of the inverse kinematic solution. Investigating other numerical techniques to solve the simulations was outside the scope of this thesis.

4.1.2.1 Feedforward Control

Adding feedforward control to the PD controller, the control law becomes

$$\tau = -K_P\tilde{q} - K_v\dot{\tilde{q}} + H(q_d)\ddot{q}_d + C(q_d, \dot{q}_d)\dot{q}_d + G(q_d) \tag{98}$$

where the subscript "*d*" denotes the desired joint angle values [24]. Inputting this control law into the dynamics yields the closed-loop system defined in terms of the state vector by Equation (99) below. The gains K_p and K_v are defined as positive definite, and K_v is analogous to K_D in the PD controller.

$$\frac{d}{dt}\begin{bmatrix}\tilde{q}\\ \dot{\tilde{q}}\end{bmatrix} = \begin{bmatrix}\dot{\tilde{q}}\\ -H^{-1}\left(-K_p\tilde{q} - K_v\dot{\tilde{q}} + [H_d - H]\ddot{q} + C_d\dot{q}_d - C\dot{q} + G_d - G\right)\end{bmatrix} \tag{99}$$

where $H = H(q)$, $C = C(q, \dot{q})$ and $G = G(q)$. Likewise, $H_d = H(q_d)$, $C_d = C(q_d, \dot{q}_d)$ and $G_d = G(q_d)$.

To assess the stability of the closed loop system, the candidate Lyapunov function is chosen in Equation (100). Assuming the eigenvalues of K_v are less than the

maximum eigenvalues of H, V is positive definite [24].

$$V = \frac{1}{2} \begin{bmatrix} \tilde{q} \\ \dot{\tilde{q}} \end{bmatrix}^T P(q) \begin{bmatrix} \tilde{q} \\ \dot{\tilde{q}} \end{bmatrix} \tag{100}$$

where

$$P(q) = \begin{bmatrix} K_p + K_v & H(q) \\ H(q) & H(q) \end{bmatrix} \tag{101}$$

The expanded time derivative is then shown in Equation (102) below.

$$\begin{aligned} \dot{V} = & \dot{\tilde{q}}^T H \dot{\tilde{q}} - \tilde{q}^T K_p \tilde{q} - \dot{\tilde{q}}^T K_v \dot{\tilde{q}} \\ & + [\tilde{q} + \dot{\tilde{q}}]^T (H - H_d) \ddot{q}_d \\ & + [\tilde{q} + \dot{\tilde{q}}](C - C_d)\dot{q}_d + \dot{\tilde{q}}^T C \tilde{q} \\ & + [\tilde{q} + \dot{\tilde{q}}](G - G_d) \end{aligned} \tag{102}$$

The derivative of the Lyapunov function can be bounded as in Equation (103).

$$\begin{aligned} \dot{V} \leq & \dot{\tilde{q}}^T H \dot{\tilde{q}} - \tilde{q}^T K_p \tilde{q} - \dot{\tilde{q}}^T K_v \dot{\tilde{q}} \\ & + c_1 \|\tilde{q}\|^2 + c_2 \|\dot{\tilde{q}}\|^2 + (c_1 + 2c_2)\|\tilde{q}\|\|\dot{\tilde{q}}\| \\ & + c_3 \|\tilde{q}\|\|\dot{\tilde{q}}\|^2 \end{aligned} \tag{103}$$

where

$$\begin{aligned} c_1 = & k_H \|\ddot{q}_d\|_H + k_{c_2} \|\dot{q}_d\|_H^2 + k_g \\ c_2 = & k_{c_1} \|\dot{q}_d\|_H \\ c_3 = & k_{c_1} \end{aligned} \tag{104}$$

and where the constants k_H, k_{c_1}, k_{c_2}, and k_g are the positive upper bounds on the inertia matrix, Christoffel symbols, and gravity matrix [24]. The subscript "H" denote the upper bounds of the norms of the vector.

Transforming Equation (103) into a quadratic form yields

$$\dot{V} \leq -\begin{bmatrix} \|\dot{\tilde{q}}\| \\ \|\dot{\tilde{q}}\| \end{bmatrix}^T Q \begin{bmatrix} \|\dot{\tilde{q}}\| \\ \|\dot{\tilde{q}}\| \end{bmatrix} + c_3\|\tilde{q}\|\|\dot{\tilde{q}}\|^2 \tag{105}$$

where

$$Q = \begin{bmatrix} \lambda_m\{K_p\} - c_1 & -\frac{c_1+2c_2}{2} \\ -\frac{c_1+2c_2}{2} & \lambda_m\{K_v\} - \lambda_m\{M\} - c_2 \end{bmatrix} \tag{106}$$

The symbol λ represents the upper bounds on the eigenvalues of the matrix in the corresponding curly brackets. If the norms of the error vectors ($\|\tilde{q}\|$ and $\|\dot{\tilde{q}}\|$) are small, the first term in Equation (105) dominates $\dot{V}$. If Q is positive definite, then $\dot{V}$ will be locally negative definite. As such, the closed-loop system is locally exponentially stable. For more information on the conditions for Q to be positive definite, refer to the code used in Appendix A [24].

For the simulation, the same gains were used as with the PD controller, where K_v is analogous to K_D. Figure 29 and Figure 30 show the desired joint angles and the simulated joint angles over time for the sinusoidal inputs and the step inputs, respectively.

Figure 31 and Figure 32 show the control inputs and the state errors over time for the sinusoidal inputs and the step inputs, respectively. While the system responds much faster than with PD control alone because of its predictive nature, the necessary torque to compute such a response for either trajectory approaches the maximum rated torques for the type of gearboxes used on the MTA [4].

Figure 33 shows the trajectory for the PD-feedforward controller. The elbow manipulator at time $t = 3$ seconds overlaps the manipulator at $t = 0$ seconds. Clearly, it tracks the trajectory much more closely than with PD control alone. However, the parameters of the system must be known for this to work, so the simulation is not

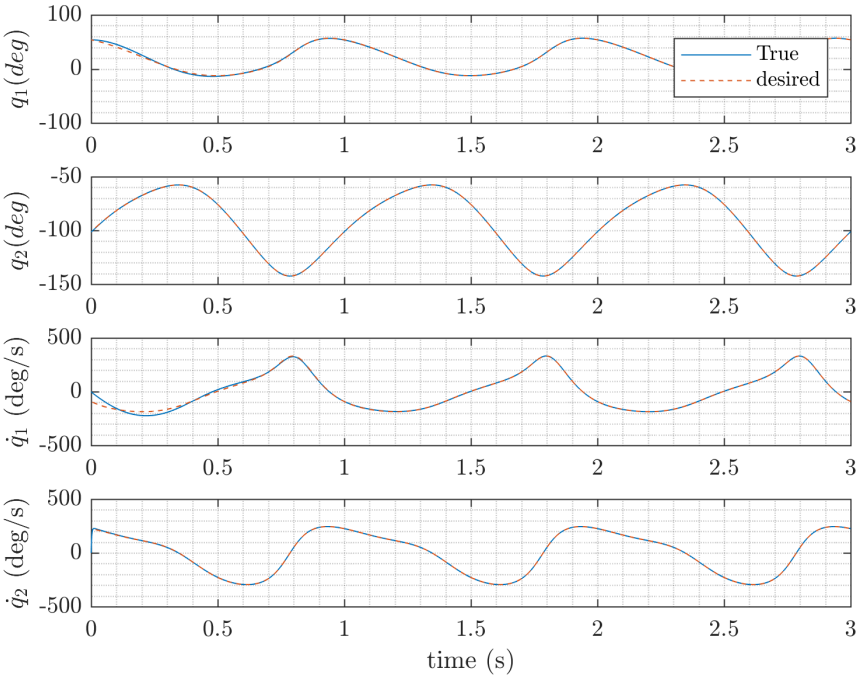


Figure 29. Desired and simulated joint angles over time for periodic joint input (1 Hz) using PD-Feedforward control

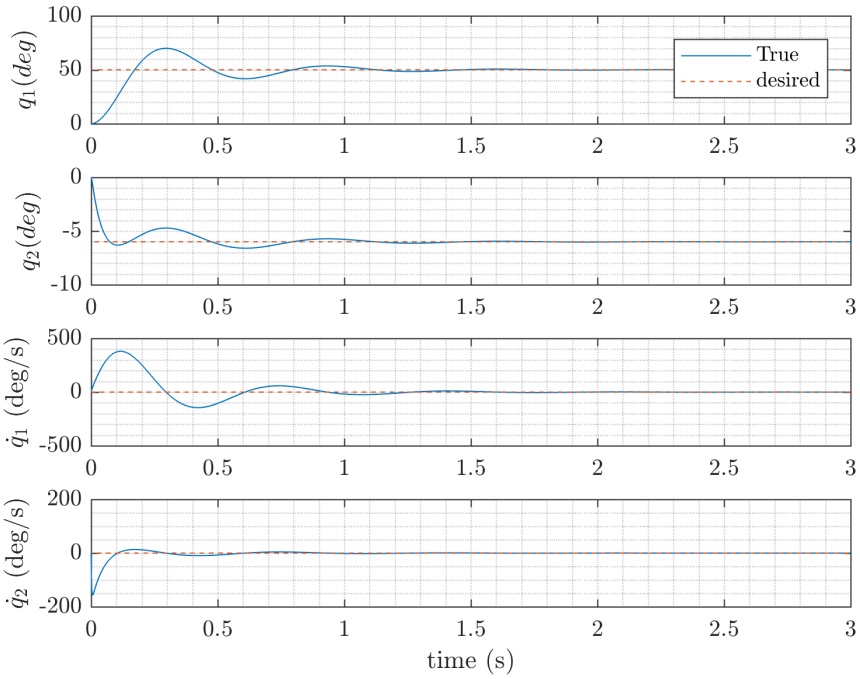


Figure 30. Desired and simulated joint angles over time for step input using PD-Feedforward control

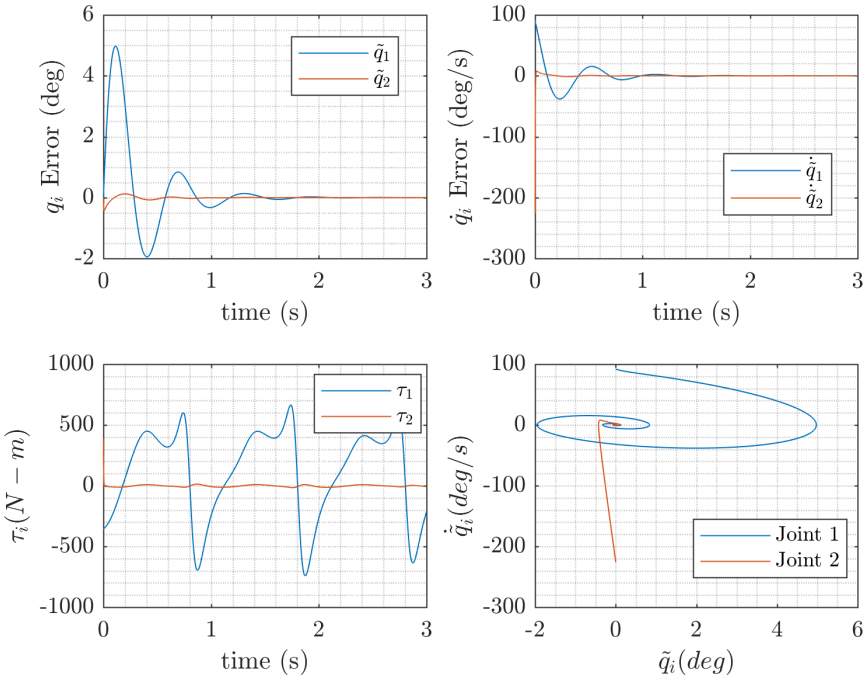


Figure 31. Control input and error in state variables over time with periodic joint input (1 Hz) using PD-Feedforward control

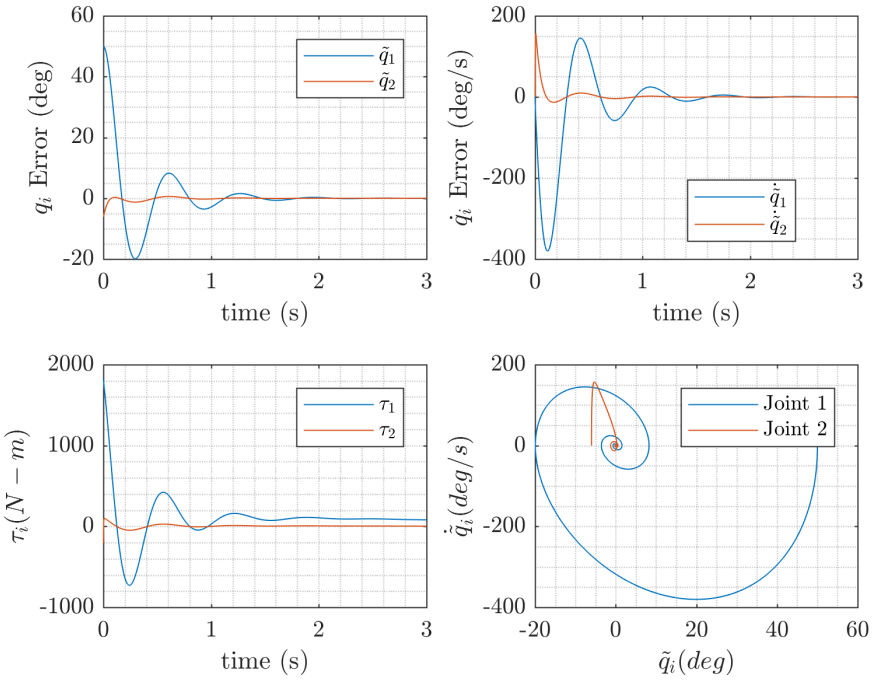


Figure 32. Control input and error in state variables over time with step input using PD-Feedforward control

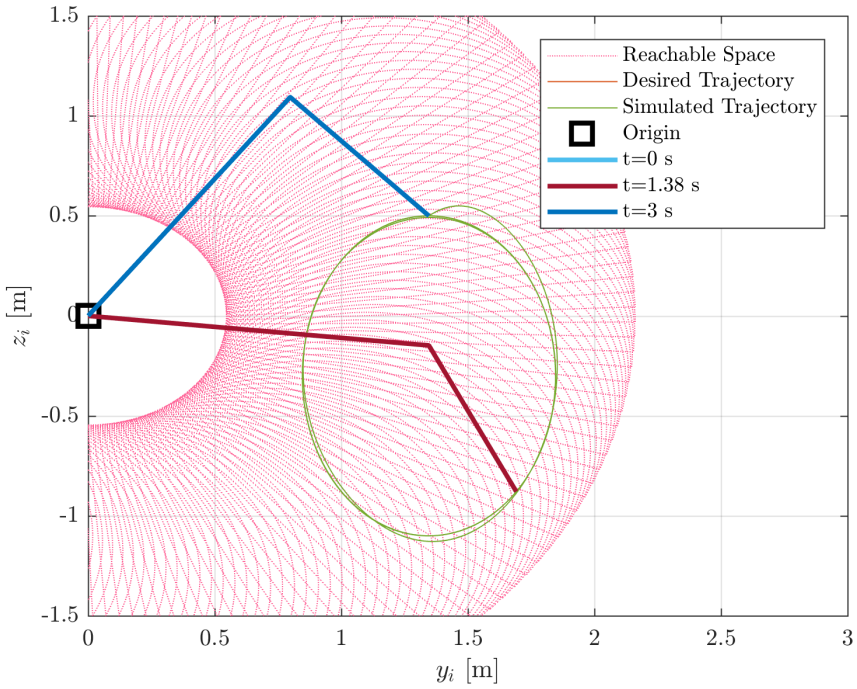


Figure 33. Comparison of desired, periodic end effector trajectory (1 Hz) and simulated trajectory using PD-Feedforward control

indicative of the real system especially when unknown nonlinearities are added such as friction and disturbances.

4.1.3 Feedback Linearization

Next, the elbow manipulator dynamics are simulated with a feedback linearization control law, which should provide better trajectory control than the PD controller because nonlinear terms of the dynamics are subtracted, leaving only a linear system behind that the tools of linear control theory apply to. The feedback linearization control law for a robotic manipulator is discussed in Section 2.6.2.2 but is shown again here for purposes of consolidation:

$$\boldsymbol{\tau} = Hv + C\dot{q} + G, \text{ where } v = \ddot{q}_{des} - 2\lambda\dot{\tilde{q}} - \lambda^2\tilde{q}$$

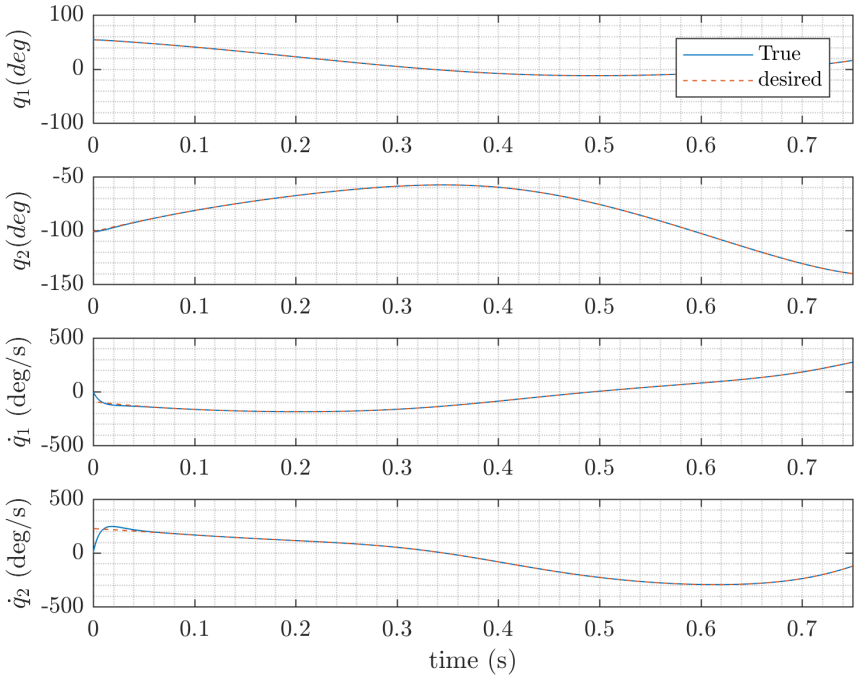


Figure 34. Desired and simulated joint angles over time for periodic input using Feedback Linearization with a 1 Hz input

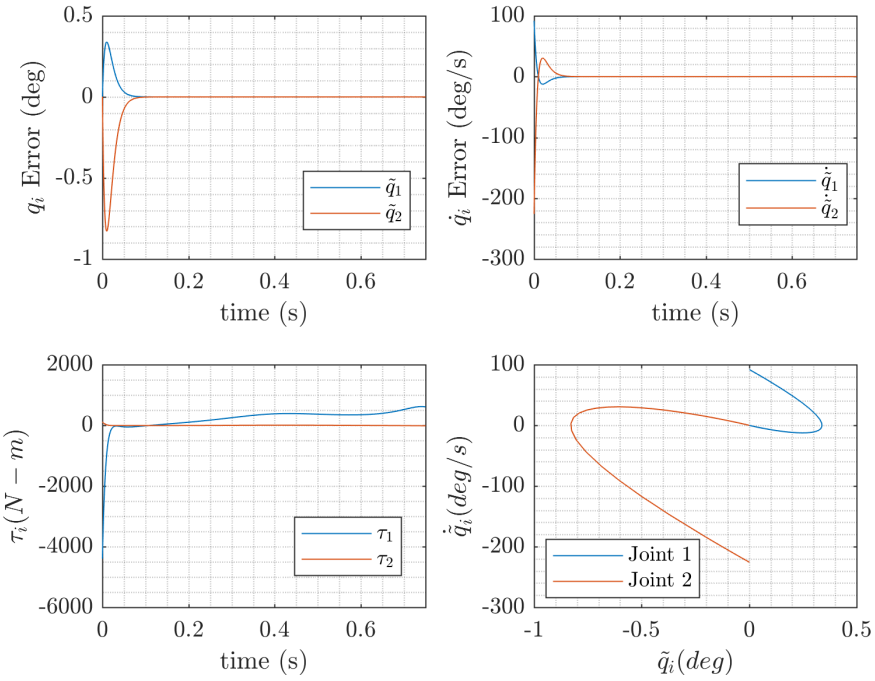


Figure 35. Control input and error in state variables over time with periodic inputs (1 Hz) using Feedback Linearization

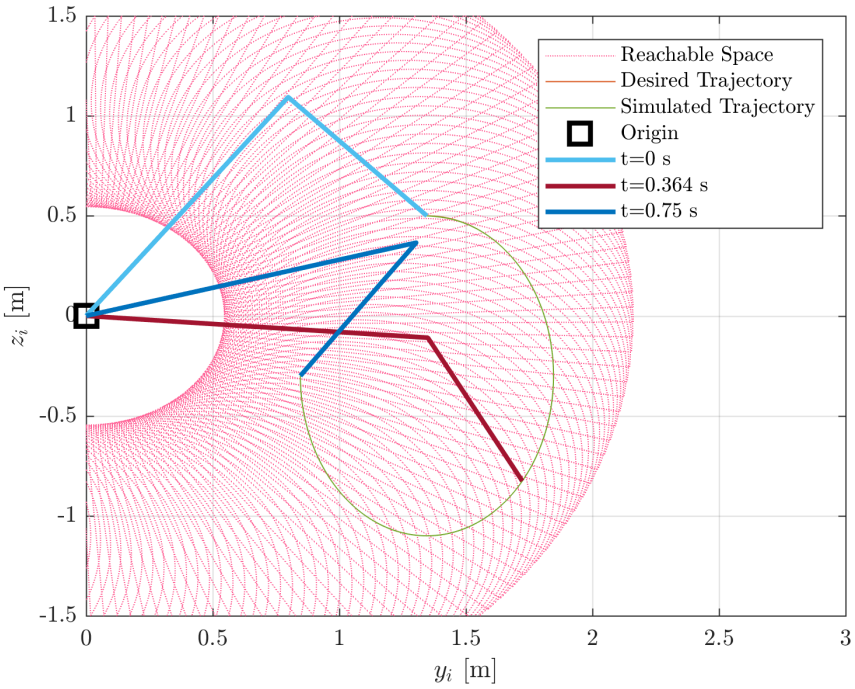


Figure 36. Comparison of desired end effector trajectory and simulated trajectory using feedback linearization with a 1 Hz input

For the simulation, $\lambda = 100$, but in general, it is strictly positive. This is because substituting τ into the dynamics gives the linear second-order equation below, which was previously shown in Section 2.6.2.2.

$$\ddot{\tilde{q}} + 2\lambda\dot{\tilde{q}} + \lambda^2\tilde{q} = 0$$

Again, the system is exponentially stable according to the Routh-Hurwitz criteria, so $\tilde{q}$ should converge to zero. The states over time for the desired trajectories specified in Equation (89) are shown in Figure 34, and the error, $\tilde{q}$ and $\dot{\tilde{q}}$, and control inputs are shown in Figure 35 in Chapter II. The simulation was only run for 0.75 seconds because it tracked to near-zero error within a tenth of a second. Showing the simulation run for longer yields no further insight. The Matlab® code for the feedback linearization simulation can be seen in Appendix A.

As expected, the error is small, on the order of 10^{-4} degrees for angular position error. Additionally, the system converges, within a tenth of a second. The step input results are not shown because if the system can track a sinusoidal input that closely, the step input will be implemented even more accurately.

Clearly, this control law would be advantageous to implement, but in order to do so, we must know the structure of the model and its parameters, values that are not known for the MTA. Additionally, the exact nonlinearities must be known, which is not possible when friction and disturbances add unmodeled uncertainty to the system.

To update the model, the control law could be adapted to add robustness into the closed-loop system [46]. Additionally, friction could be added into the model used [15].

4.1.4 Sliding Mode Control

The control laws for sliding mode control are shown in Section 2.6.2.3 but are rewritten here. To reiterate, the sliding surface, where the dynamics approach zero, for a MIMO system is defined as

$$\boldsymbol{s} = \dot{\boldsymbol{q}} - \dot{\boldsymbol{q}}_r$$

where $\dot{\boldsymbol{q}}_r$, the reference velocity, is defined as

$$\dot{\boldsymbol{q}}_r = \dot{\boldsymbol{q}}_{des} - \Lambda\tilde{\boldsymbol{q}}$$

where Λ is strictly positive definite. The sliding condition which makes the surface an invariant set, is then defined as

$$\frac{1}{2}\frac{d}{dt}s_i^2 \leq -\eta|s_i|$$

Finally, the sliding mode control law is defined as

$$\boldsymbol{\tau} = \hat{\boldsymbol{\tau}} - \boldsymbol{k} sgn(\boldsymbol{s})$$

where $\hat{\boldsymbol{\tau}} = \hat{\boldsymbol{H}}\ddot{\boldsymbol{q}}_r + \hat{\boldsymbol{C}}\dot{\boldsymbol{q}}_r + \hat{\boldsymbol{G}}$ is the exactly known upper limit on the input torque given the known upper bounds, $\hat{\boldsymbol{H}}, \hat{\boldsymbol{C}}$, and $\hat{\boldsymbol{G}}$, on the matrices, $\boldsymbol{H}, \boldsymbol{C}$, and $\boldsymbol{G}$, respectively. With $\tilde{\boldsymbol{H}} = \hat{\boldsymbol{H}} - \boldsymbol{H}$, and so on, k_i is chosen in the vector k to satisfy the sliding condition as follows.

$$k_i \leq |[\tilde{\boldsymbol{H}}\ddot{\boldsymbol{q}}_r + \tilde{\boldsymbol{C}}\dot{\boldsymbol{q}}_r + \tilde{\boldsymbol{G}}]_i| + \eta_i$$

To reduce chattering, the boundary layer thickness, ϕ, is introduced as a constant so that the control law can be rewritten as

$$\boldsymbol{\tau} = \hat{\boldsymbol{\tau}} - \boldsymbol{k} sat(s_i/\phi_i)$$

where $sat(s_i/\phi_i)$ takes on the value of the sign of s if $s > \phi$ or is s/ϕ otherwise. The boundary layer can also be computed as a pseudo-state variable where its derivative is defined as

$$\dot{\phi} = -\lambda\phi + k(q_{des})$$

Finally, the control law becomes

$$\boldsymbol{\tau} = \hat{\boldsymbol{\tau}} - \bar{\boldsymbol{k}} sat(s_i/\phi_i)$$

where

$$\bar{\boldsymbol{k}}_i = k_i - \dot{\phi}_i$$

To prove the stability of the sliding mode controller, consider the candidate Lya-

punov function in Equation (107), which is positive definite and radially unbounded,

$$V = \frac{1}{2} s^T H s \tag{107}$$

Taking the time derivative of V and substituting in the dynamics $H\ddot{q} = \tau - C\dot{q} - G$ where $\dot{q} = s + \dot{q}_r$ yields

$$\begin{aligned} \dot{V} &= s^T(\tau - H\ddot{q}_r - C\dot{q}_r - G) \\ &= s^T(\hat{\tau} - ksgn(s) - H\ddot{q}_r - C\dot{q}_r - G) \\ &= s^T(\tilde{\boldsymbol{H}}\ddot{\boldsymbol{q}}_r + \tilde{\boldsymbol{C}}\dot{\boldsymbol{q}}_r + \tilde{\boldsymbol{G}}) - \Sigma_{i=1}^{n} k_i |s_i| \\ &\leq -\Sigma_{i=1}^{n} \eta_i |s_i| \end{aligned} \tag{108}$$

Therefore, choosing a positive η and conforming to the sliding condition proves that the error, $\tilde{q}$, approaches zero once the trajectories are on the sliding surface.

Figure 37 shows the desired joint angles and the simulated joint angles over time with sliding mode control with a constant boundary layer. Figure 38 shows the desired and simulated states with sliding mode control with a time-varying boundary layer. Although the simulations do seem to initially track the calculated joint angles, the error between the desired and actual states follows a sinusoidal pattern if the simulation is run for long enough. This might be a coding error, but it might also be a symptom of the fact that the uncertainty bounds of the physical properties are too high for what the control law is designed for. Unfortunately, even with a simulation time of three seconds, the simulation took upwards of twenty minutes to complete. Thus, the best results reached are presented here, even with the obviously large tracking error for very low inaccuracy in the parameters (1%). To see the Matlab®, refer to Appendix A.

Figure 39 shows state error and control input over time with sliding mode control

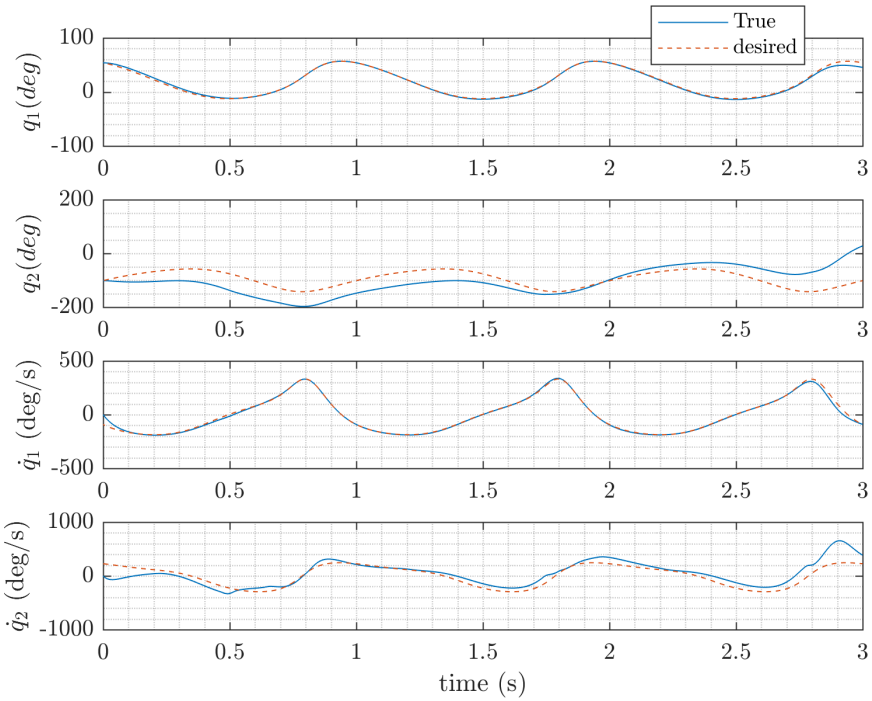


Figure 37. Desired and simulated joint angles over time for periodic input (1 Hz) using sliding mode control, constant boundary layer

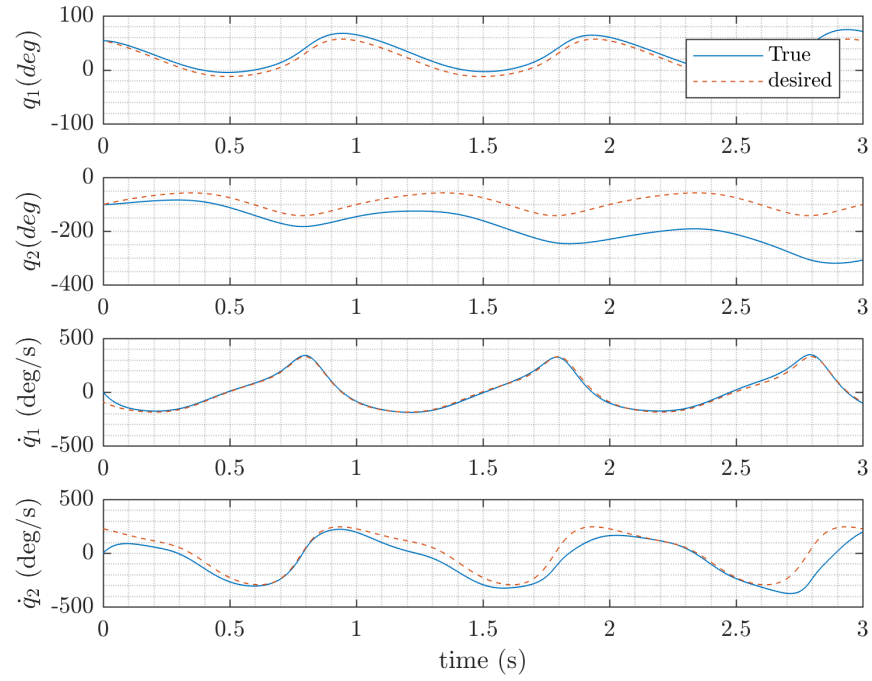


Figure 38. Desired and simulated joint angles over time for periodic input (1 Hz) using sliding mode control, time-varying boundary layer

with a constant boundary layer. Figure 40 shows the state error and control input over time with sliding mode control with a boundary layer. The former has a constant boundary layer, and the latter has a time-varying boundary layer. For the three seconds of simulation, it is clear that the torque input followed a cycle corresponding to the cyclic nature of the desired trajectory.

Figure 41 shows the actual and calculated parameters (when the system is linearized with respect to the parameters) over time with a constant boundary layer. Figure 42 shows actual and calculated parameters over time with a time-varying boundary layer. Neither calculate the parameters correctly.

Figure 43 shows the sliding surface and the boundary layer over time with a constant boundary layer. The boundary layer chart is used for comparison. Figure 44 sliding surface and the boundary layer over time with a time-varying boundary layer. Adding the time-varying boundary layer has the effect of smoothing out the sliding surface as well as lessening the control needed, so it was used in both simulations of the sliding mode controller. It seems that only one of the sliding surfaces is reached, forcing an error on the other one. This could be an explanation for the sinusoidal error over long simulation times.

Figure 45 shows the desired and simulated trajectory over time by the system with sliding mode control with a constant boundary layer. Figure 46 shows the desired and simulated trajectory over time by the system with sliding mode control with a time-varying boundary layer. Clearly, neither of the control laws tracked the trajectory. To see the effect of the inaccuracy in the parameters and the time-changing quality of the end-effector mass properties, the sliding mode controller was run with 0% inaccuracy and static values for the end-effector properties as shown in Figure 47 and Figure 48. Clearly, as shown, even making the system autonomous does not make the error converge to zero for all states in the time simulated, and the trajectory is

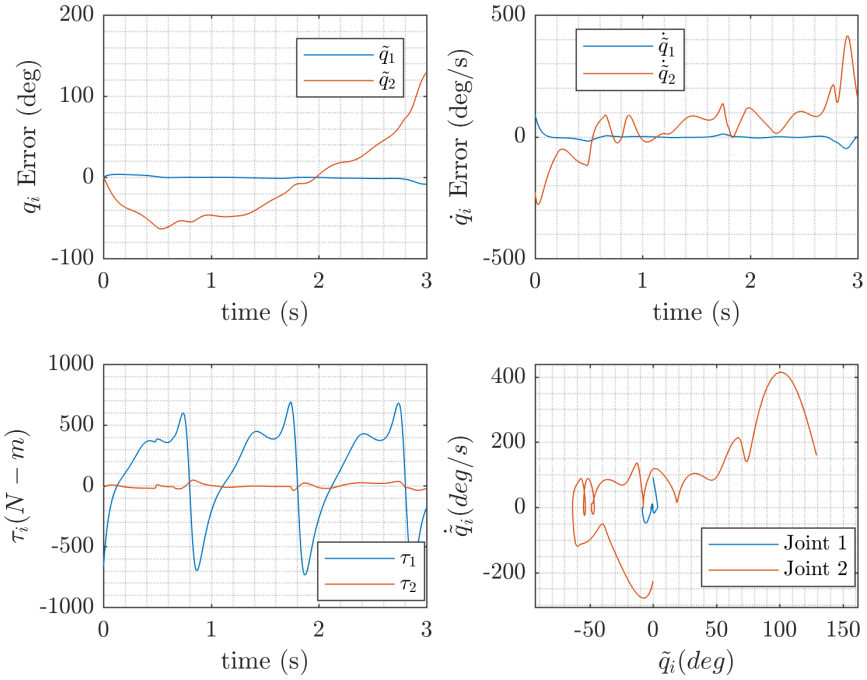


Figure 39. Control input and error in state variables over time for periodic input (1 Hz) using sliding mode control, constant boundary layer

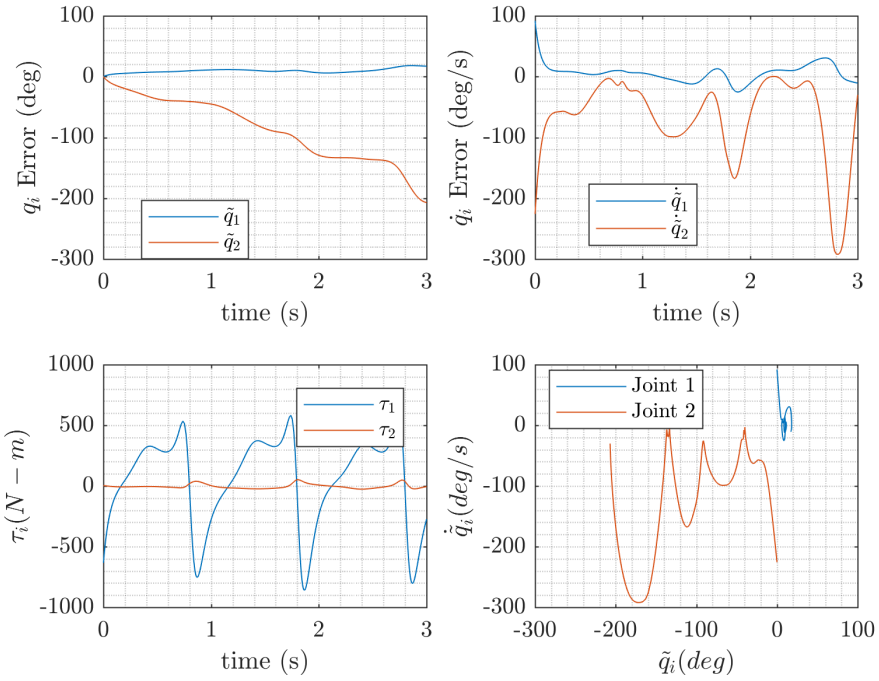


Figure 40. Control input and error in state variables over time for periodic input (1 Hz) using sliding mode control, time-varying boundary layer

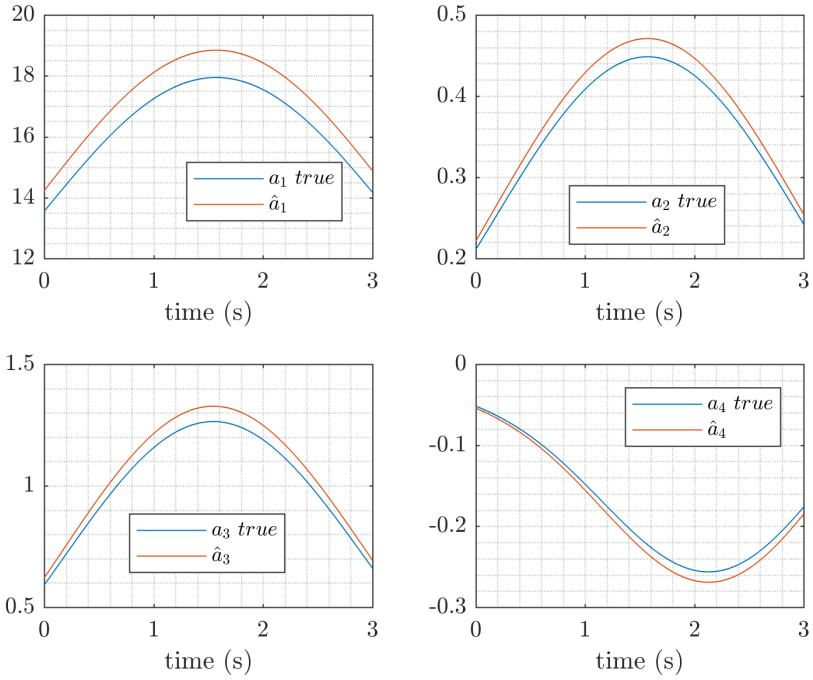


Figure 41. Actual and estimated parameters for periodic input (1 Hz) using sliding mode control, constant boundary layer

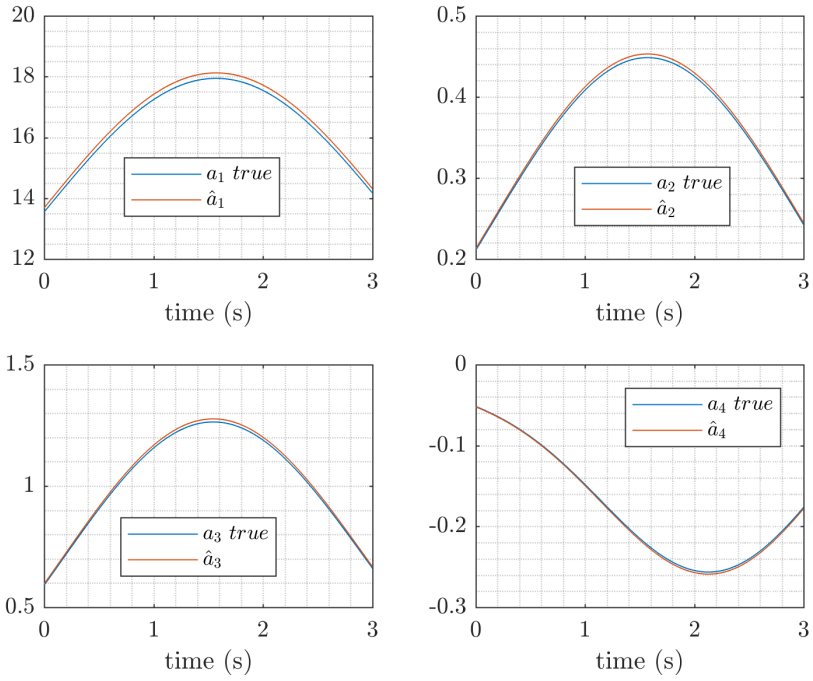


Figure 42. Actual and estimated parameters for periodic input (1 Hz) using sliding mode control, time-varying boundary layer

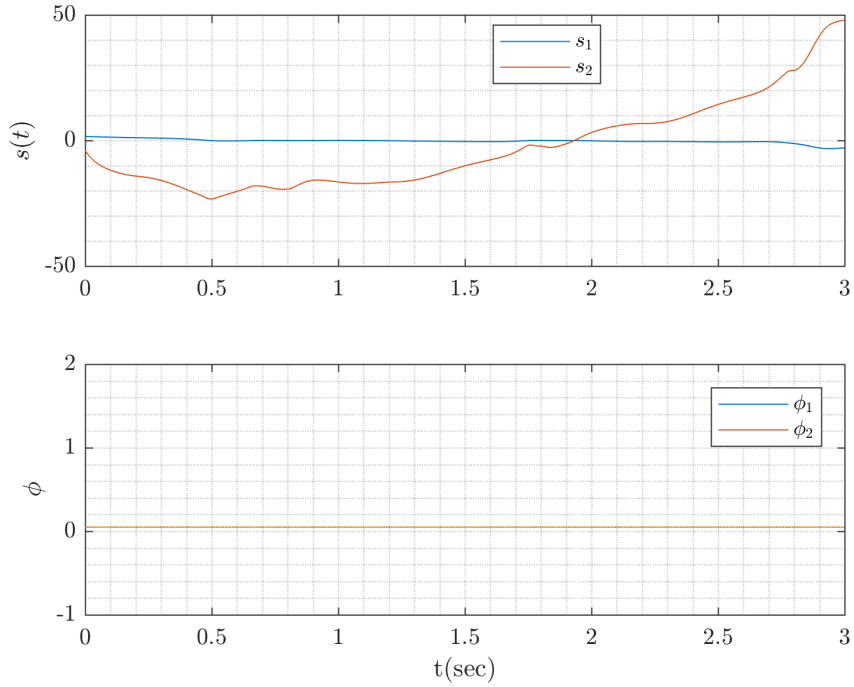


Figure 43. Sliding surface and boundary layer using sliding mode control, constant boundary layer with a 1 Hz input

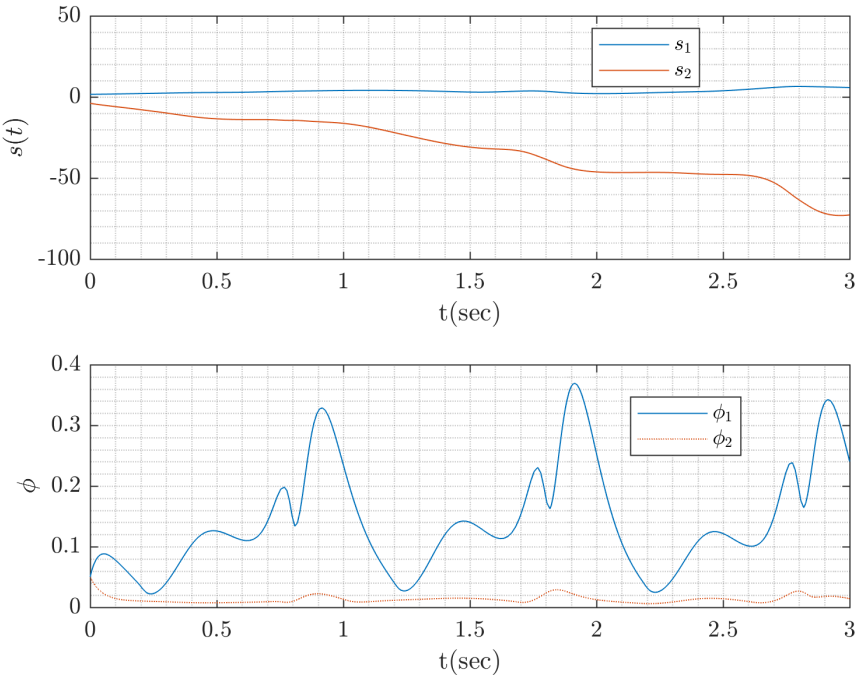


Figure 44. Sliding surface and boundary layer using sliding mode control, time-varying boundary layer with a 1 Hz input

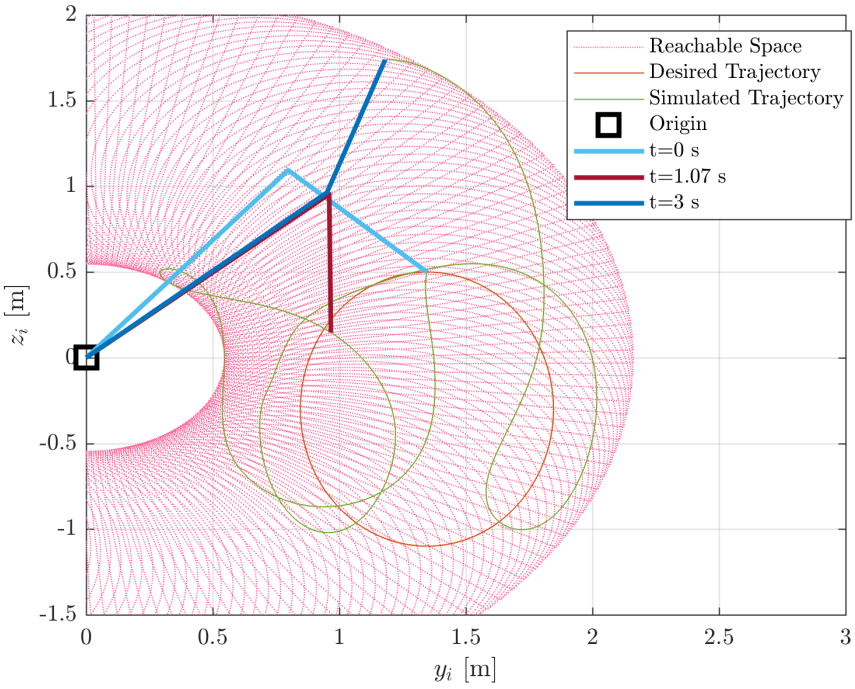


Figure 45. Comparison of desired end effector trajectory and simulated trajectory using sliding mode control with a constant boundary layer with a 1 Hz input

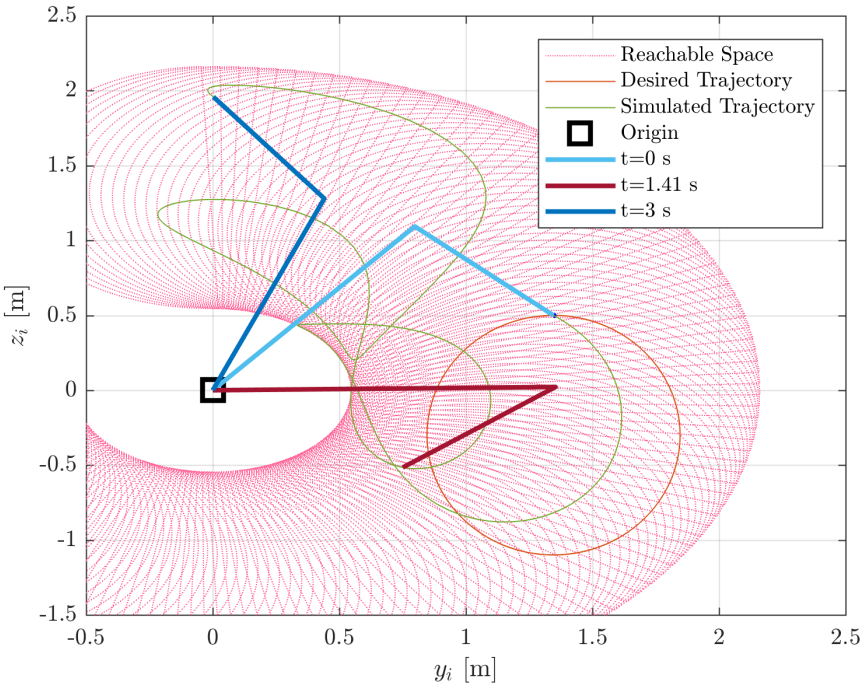


Figure 46. Comparison of desired end effector trajectory and simulated trajectory using sliding mode control with a time-varying boundary layer with a 1 Hz input

not accurately tracked. As such, sliding mode control was not pursued further.

4.1.5 Adaptive Control

Model reference adaptive control needs information about the exact structure of the dynamics to be useful. Additionally, one must be able to write the dynamics linearly with respect to the parameters. For the 2 DOF elbow manipulator, this takes on the form $Y(q, \dot{q}, \dot{q}_r, \ddot{q}_r)a = \tau$ where Y is a 2x4 matrix containing the part of the dynamics that has only sinusoidal combinations of the states, and $\boldsymbol{a}$ has only the parameters. Section 2.6.2.4 outlined one notation for writing the dynamics in this way, but the representation used in simulations is slightly different. The contents of the parameter matrix is shown below in Equation (109). This configuration does not take gravity into account.

$$\begin{aligned} a_1 =& m_1 l_{c1}^2 + I_1 + I_e + m_e(l_1^2 + l_{ce}^2) \\ a_2 =& I_e + m_e l_{ce}^2 \\ a_3 =& m_e l_{c1} l_1 \cos\delta_e \\ a_4 =& m_e l_{c1} l_1 \sin\delta_e \end{aligned} \tag{109}$$

To reiterate, the MRAC control law is

$$\tau = Y\hat{a} - K_D s = \hat{H}\ddot{q}_r + \hat{C}\dot{q}_r - K_D s$$

where the adaptive law is

$$\dot{\hat{a}} = -\Gamma Y^T s$$

where Γ is a symmetric positive definite matrix. Clearly, since s shows up again, this control law is a combination of a sliding mode and an adaptive controller. The parameters are added to the state vector as pseudo-states in the simulation.

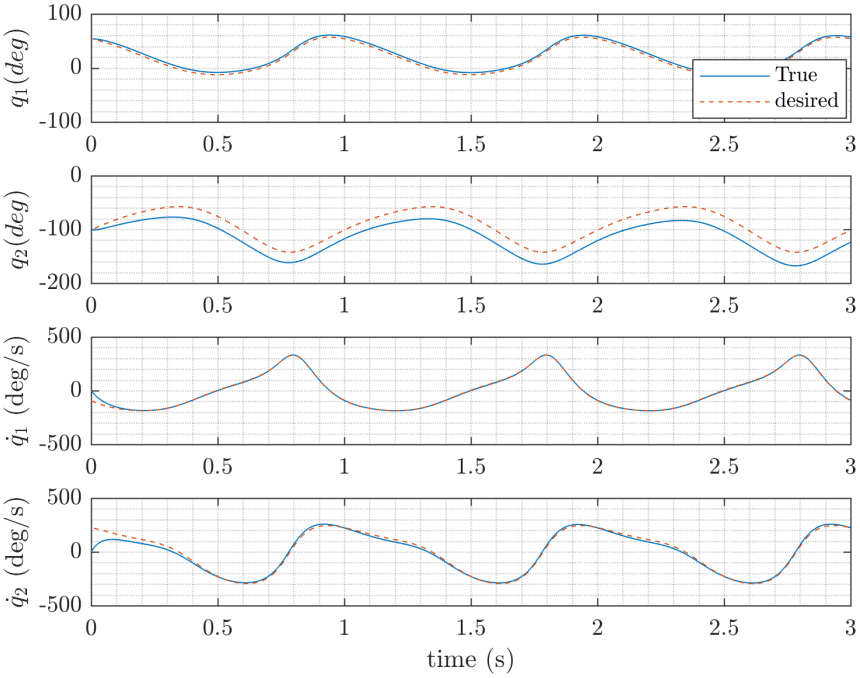


Figure 47. Desired and simulated states using sliding mode control with 0% inaccuracy in parameters and static end-effector values with a 1 Hz input

As such, the candidate Lyapunov function is positive definite and radially unbounded if defined as in Equation (110) below.

$$V = \frac{1}{2} s^T H s + \tilde{a}^T \Gamma^{-1} \tilde{a} \tag{110}$$

Differentiating the Lyapunov function and making the appropriate substitutions yields

$$\dot{V} = -s^T K_D s \tag{111}$$

which is negative semi-definite when K_D is positive definite. Then, by the invariant set theorem, if $\dot{V} = 0$, then $s = 0$, implying that q and $\dot{q} \to 0$ as $t \to 0$. Therefore, the system is globally asymptotically stable, and the tracking error should converge to zero.

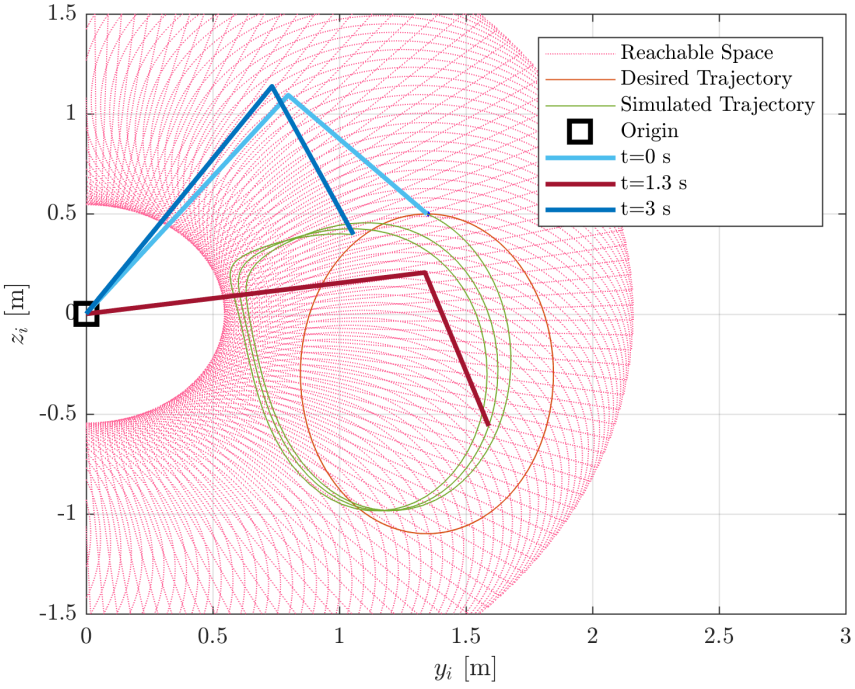


Figure 48. Comparison of desired end effector trajectory and simulated trajectory using sliding mode control with 0% inaccuracy in parameters and static end-effector values with a 1 Hz input

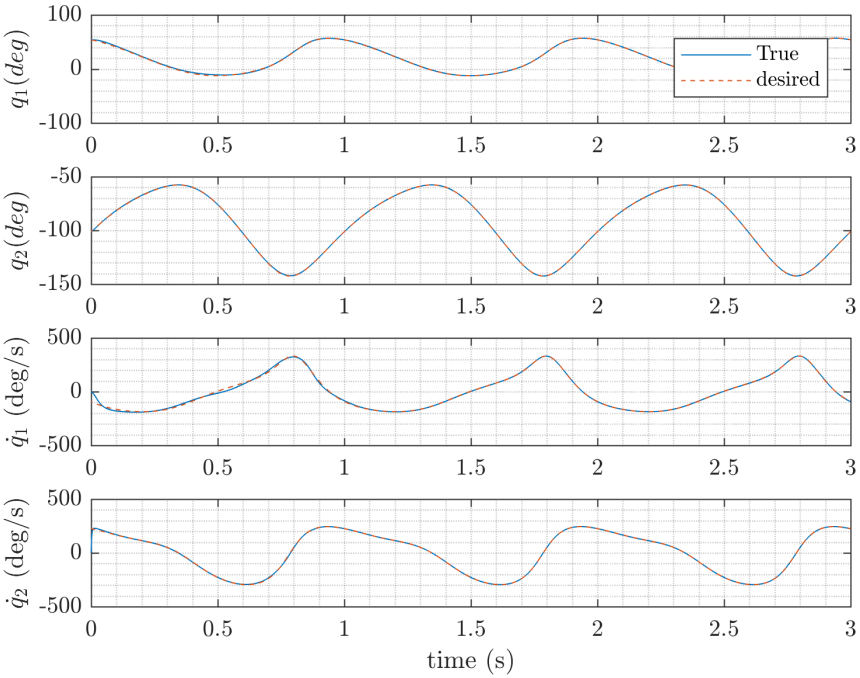


Figure 49. Desired and simulated joint angles over time for periodic input (1 Hz) using model reference adaptive control

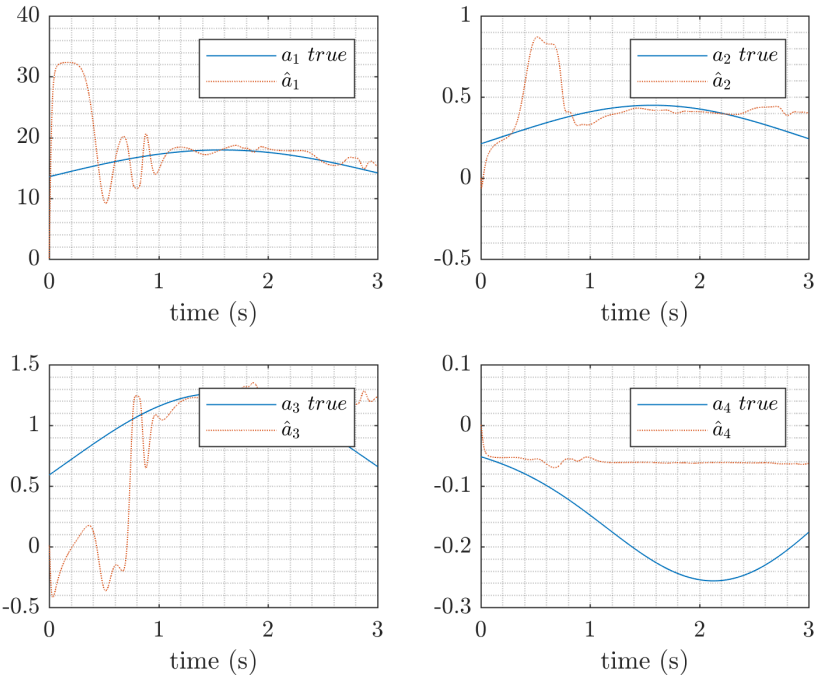


Figure 50. Actual and calculated parameters over time for periodic input (1 Hz) using model reference adaptive control

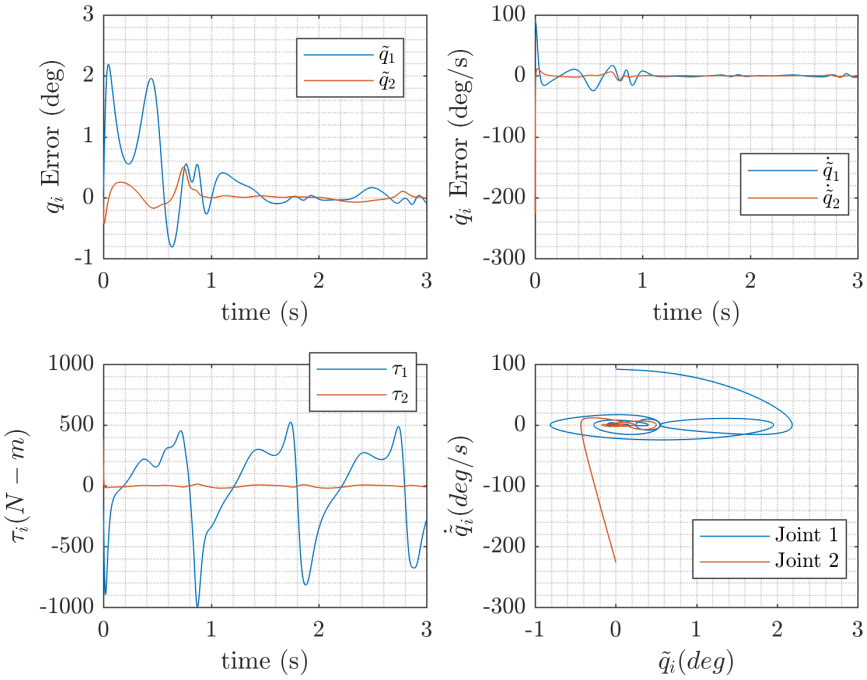


Figure 51. Control input and error in state variables over time using model reference adaptive control with a 1 Hz input

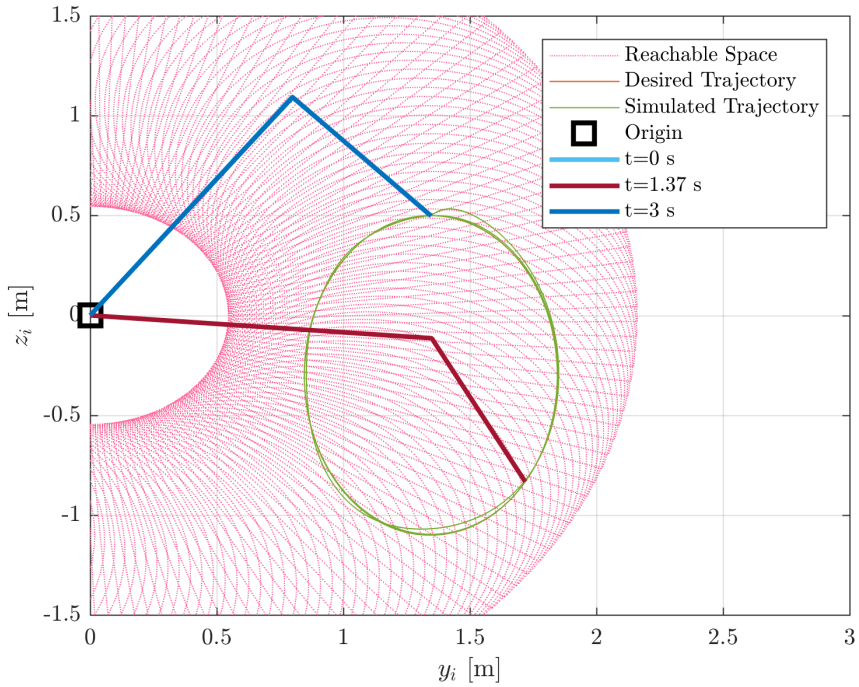


Figure 52. Desired and simulated end effector trajectories using model reference adaptive control with a 1 Hz input

A simulation of the control law for the reference trajectory described in Equation (89) can be shown in Figure 49 where $K_D = 100I$ and $\Gamma = diag(13.5,\ 0.2,\ 0.6,\ 0.01)$, which contains the rounded off initial values of $\boldsymbol{a}$. In order to simulate the system, the state had to be appended with $\hat{a}_i$, and their estimated values can be seen in Figure 50. The code can be seen in Appendix A.

Clearly, the estimated values of the parameters defined in Equation (109) differ from the actual values especially near the beginning of the simulation. Using MRAC, then, would be a better way to estimate the system parameters than using sliding mode control alone. Figure 51 shows the state error and control input over time with using model reference adaptive control. Although the control input does get relatively large in the negative peaks, it does seem to decrease over time. It follows then, that the overall error in parameter estimation and states would decrease, needing less torque input, as the simulation time increases, but computation time was approximately twenty minutes when analytically solving the inverse kinematics in the solver. Regardless, the trajectory is mostly tracked, as shown by Figure 52.

4.2 Summary of 2 DOF results

Of note, all of the Lyapunov stability analysis presented here and in Table 11 holds for the autonomous system. However, when the end effector has non-static values for the mass properties due to aerodynamic effects, the system becomes non-autonomous due to the explicit time-dependence. Because the simulated system is non-autonomous, there are different conditions for showing stability as well as different types of stability that the system can exhibit. For instance, the invariant set theorems are not applicable to non-autonomous systems; therefore, the stability analysis described in the previous sections is not completely valid for the non-autonomous system. The valid stability analysis can be performed using alternative, potentially

time-dependent Lyapunov functions, but creating such functions to prove stability is the subject of many papers and is thus outside the scope of this research with the limited time available.

However, based upon the results of the simulation presented here, the MRAC scheme is the most suitable control method for the MTA. Not only does it track relatively well to the desired trajectory, but it assumes the least information available as compared to the other methods. A qualitative comparison of the various control schemes used in simulation is shown in Table 11. MRAC is useful in system identification, and can also be used in a decentralized scheme so that global position control can be achieved through interactions of its subsystems. Further research on the MTA should use and improve upon the MRAC control scheme in order to characterize the MTA system and create an adaptive closed-loop system.

4.3 6 DOF Model

The original plan was to implement one of the control strategies tested on the 2 DOF system onto the actual MTA. However, simulating the controllers on the 2 DOF model alone proved to be a larger endeavor than at first thought. The computation time of the 2 DOF system ranged from approximately one to twenty minutes using simulation times up to three seconds maximum and thus were not ready for hardware implementation.

Time was spent updating a preexisting 6 DOF robot (PUMA 762) model to be more akin to the MTA, as described in Chapter III. Both the forward kinematics and the inverse kinematics were changed to reflect the geometry of the MTA. Additionally, the graphics were updated with the CAD files provided by RE2, Inc, as shown in Figure 53.

The MTA source code was downloaded from the MTA computer with the intention

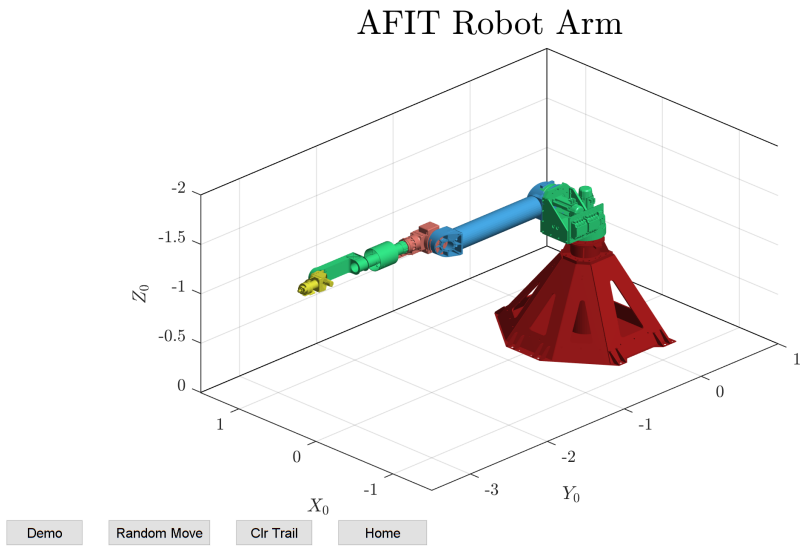


Figure 53. PUMA 762 model updated with MTA graphics and kinematics in zero angle configuration in MTA inertial reference frame

of including the control algorithm in the MTA model, but the update by Neya Systems proved to be elusive. Necessary improvements include integrating the MTA dynamics to include gravitational effects, time delays, and physical constraints based upon the size of the test section in the wind tunnel. Once the dynamics are integrated and new trajectories are defined, an adaptive control law (decentralized or global) can be tested on the full 6 DOF MTA model. More specific potential improvements to the model will be discussed in Chapter V.

4.4 Experiment

Although, as mentioned, the hardware experiments with the MTA did not come to fruition, a brief overview of the description of the tests and measurements which needs to be included in future research will be discussed here.

To begin, the MTA source code should be updated to output computed torque, as a basis of comparison for the simulation data. Then, a new directory needs to be created for the MTA including the control laws for the last two joints of the MTA,

including a PD-feedforward and adaptive sliding mode control. This will include translating or wrapping the code from Matlab® into C++.

Using the encoder data and the IMU data, the positional error between the desired, simulated, and experimental trajectories should be calculated. If possible, a system identification of the MTA, similar to what Lancaster did for one specific motion, should be completed for the whole system, particularly the last two joints to verify the dynamic model parameters [26]. Another method is to rely upon the linearity of parameters and individually excite each joint so that the Y matrix can be computed at several test points via the method of least squares as described for smaller robotic manipulators [51]. Once the planar 2 DOF elbow manipulator model is updated, it can be used as a test bed for future trajectory planning.

Ideally, once the 2 DOF model has been developed and tested, the control laws can be directly adapted to all six degrees of freedom of the MTA. However, because computation time in simulating the 2 DOF model has been so high, there will likely need to be linearizations and numerical analysis done in place of the analytical solutions that this research preferred in order to keep the control laws acting in real-time. Further recommendations for research will be discussed in Chapter V.

4.5 Chapter IV Summary

Chapter IV presented the simulations of the 2 DOF elbow manipulator model representing the elbow pitch and wrist pitch degrees of freedom of the MTA. Specifically, the physical properties were discussed, as well as the constraints put on creating a feasible trajectory for the robot arm to follow. With specific assumptions, the 2 DOF manipulator system was then transformed into a closed-loop system with the use of PD control, PD with feed-forward control, feedback linearization, sliding mode control with constant and time-varying boundary layers, and with an adaptive controller.

Results of the simulation were presented in figures and discussed. Chapter IV also briefly covered the 6 DOF MTA model as well as the experimental plan that should come from the MTA 2 DOF model results. Some of the Matlab scripts discussed in this chapter can be seen in Appendix A.

Table 11. Qualitative comparison of 2 DOF control schemes

	PD	**PDFF**	**FBL**	**SMC**	**MRAC**
Includes Gravity?	Yes	Yes	Yes	Yes	No
Stability	Local asymptotic stability	Local asymptotic stability (conditional)	Exponential stability	Local exponential stability	Global asymptotic stability
Information Needed	Structure and parameters	Structure and parameters	Structure and parameters	Structure and bounds	Structure
Robust?	No	No	No	No	Yes
Error Converges?	No	Yes	Yes	No	Yes
Quick Transient Response?	No	Yes	Yes	No	Yes

V. Conclusions and Recommendations

While previous efforts by Lancaster, Sellers, and Bower have focused on establishing and improving upon the reliability of the wind tunnel test platform that is the MTA, this research focused on building the MTA's dynamic model from the ground up. With a high-enough fidelity model and robust-enough control laws, the error in the MTA's trajectory tracking can be decreased instead of only measured. Chapter V details the work that has gone into the creation of the model so far as well as the control simulations that have been tested to begin the choosing of a more suitable control law for the MTA.

5.1 Summary of Results

A dynamic model for a 2 DOF planar elbow manipulator with two revolute joints was derived via the Lagrangian method to represent the last two links of the MTA. The physical properties of the model were used in the simulation with time-varying properties of the end-effector to simulate the wind-tunnel environment. A feasible end-effector trajectory was calculated based upon the joint angle ranges of the MTA and the reachable space of the manipulator. The closed-loop system consisting of the 2 DOF model and a control law was simulated to track the feasible desired trajectory. The control laws used include proportional derivative control, proportional derivative with feedforward control, feedback linearization (computed torque), sliding mode control with different boundary conditions, and an adaptive sliding mode control. All of the control laws were stable at least locally via Lyapunov analysis, but the simulations did not all converge to zero tracking error due to the nature of the time-varying parameters and the inherent system lag. Additionally, most of the simulations included assumptions that are not representative of the real MTA

control system problem such as lack of availability of exact physical parameters and unmodeled disturbances including friction.

5.2 Significance of Research

Although this thesis started out with an emphasis on experimental control of the MTA, the completed simulations of the 2 DOF model yield some insight. First, it provides a start of a model of the MTA based upon first principles, including work-energy. Although it might be computationally costly to go the analytical route, doing so can provide insight to the true dynamics of the MTA. For instance, one is more likely to conclude that tracking error is based upon the model's lack of a friction term, or the wrong bounds on the inertial matrix, as opposed to having to attribute error to finite wordlength.

As the model fidelity increases, it can be used to design trajectories for the actual MTA as well as pick the control law that will be most suitable for the wind tunnel experiment planned. Integrated with the research done on the time-accuracy of the wind-tunnel sensors and DAQ system performed by Sellers and Bower, having a trajectory tracking system will increase the reliability and repeatibility of dynamic tests executed in the AFIT subsonic wind tunnel. Not only will this increase the capability and quality of research that AFIT outputs with the MTA, but will help the Air Force at large with the data that accurate dynamic wind tunnel testing will provide for design and validation of future aerospace technologies.

5.3 Recommendations for Future MTA Testing

Although the significance of providing a suitable nonlinear control method for the MTA is unquestionable, there is still much that needs to be done before the MTA can be utilized to its full potential.

Specific short-term research goals include finding a way to calculate the inverse kinematics including the joint velocities and accelerations more quickly when called during a simulation. As the code stands, the functions are derived symbolically at each time step in the *ode45* solver which drives up the computation time significantly. To do this, the author recommends either solving the inverse kinematics with an iterative scheme, or to have the joint angles, velocities, and positions fully calculated and input to a look-up table. The look-up table method requires an interpolation of the desired joint kinematic values at each time step of the solver. This will be quicker than solving the inverse kinematics each time, but requires a different look-up table for each desired trajectory. Before such a table should be created, the desired trajectory should also be limited to the angular velocity and acceleration software limits already discussed in Chapter IV. This way, the computed torque will be more comparable to what the MTA will calculate and can actually control. Adding the limits will be simple if the trajectories have a low enough frequency. Another way to increase the computation speed might be to calculate the product of exponentials formulation for the MTA and using those to calculate the pseudo- or inverse manipulator Jacobian, disregarding singularities [28].

Depending on the trajectory being tested, using PD control might still be a feasible alternative to an adaptive control law due to its simplicity. However, the MTA must be near its linear region. Thus, when the 6 DOF MTA model is fully updated with its dynamics, it would be prudent to test the PD-feedforward control law on the full system for simple trajectories.

In the near future, as discussed in Chapter IV, the control laws used on the 2 DOF model should be executed on the AFIT MTA as a basis of validation of the dynamic model. From there, the experimentally-obtained data can be used to update the physical properties of the model so that simulation of the control laws better

represent actual MTA performance. Similarly, system identification of the AFIT MTA's physical parameters using a least squares method can help with the model updates.

At the same time, developing more robust control laws (with respect to disturbances to the end-effector) will be of primary importance once aerodynamic forces and moments are applied during actual testing. When these are introduced, the computation time will greatly increase, so the next researcher might have to revert to iterative approaches to calculating the dynamics and kinematics as opposed to using the analytical equations. Doing so will entail quantifying the error introduced using a numerical method and ensuring it is acceptable for the trajectory-tracking task.

Once an acceptable amount of tracking error has been reached for the elbow pitch and wrist pitch joints of the MTA in trajectory tracking, the control laws can be implemented on the 6 DOF model and then the MTA itself in the same approach as for the 2 DOF model.

In the long-term, the goal is to provide a simple way to design a trajectory that easily fits within the constraints imposed by the MTA software and hardware and also by the location and size of the wind-tunnel test section. This will most likely entail path-planning optimization, many examples of which exist in literature. Constraints would include the angular position, velocity, and acceleration motor limits, physical space of the wind tunnel test-section, and the bounds on the tuning parameters if using certain control laws. Creating a simple method to create new prescribed trajectories will ensure that the MTA's utility is fully realized.

Appendix A. Matlab®, C++, and LabVIEW code

This appendix includes some of the Matlab® code written for simulations of the closed-loop systems. They include a function for the inverse kinematics of the planar 2 DOF elbow manipulator and the mass properties of the MTA's last two links. Additionally, the routines for the PD controller, the PD-Feedforward controller, the sliding mode controller both with a constant and a time-varying boundary layer, and an adaptive controller are listed. Also listed is the code used to graph the end effector trajectories for comparison once the control schemes are run.

Inverse kinematics for planar elbow manipulator with joint limits

```
function [theta1,theta2]= invkin2(x,y,param,t);
%Returns the angles of the first two links in the robotic arm as a list.
%    returns -> (th1, th2) (RAD), th1 is angle of link 1 wrt ground, th2
%    is angle of link 2 wrt link 1
%    input: x,y coordinates of end effector

% Not good with symbolic functions!

L1 = param.L1; L2 = param.L2;
r = sqrt(x.^2 + y.^2);

% Check if the region is feasible if coordinates are values/nonsymbolic:
if isa(x,'double') && isa(y,'double')
    if (r>(L1 + L2))
    fprintf('Cannot reach coordinates with current geometry \n')
    end
end

% Do the inverse kinematics calcs:
a = acos((L1.^2+L2.^2-r.^2)./(2.*L1.*L2));
th2_1 = pi + a;
th2_2 = pi - a; % two possible solutions
b = acos((r.^2+L1.^2-L2.^2)./(2.*L1.*r));
th1_1 = atan2(y,x) + b; % sign of b corresponds to sign of a
th1_2 = atan2(y,x) - b;



% Wrap all the angles:
th2_1 = wrapToPi(th2_1);
th2_2 = wrapToPi(th2_2);
```

```
th1_1 = wrapToPi(th1_1);
th1_2 = wrapToPi(th1_2);

% Check to see if both solutions are the same:
if (th2_1==th2_2)  % pretty much if theta2 is pi
    theta1 = th1_1; % size is 1xlength(t)
    theta2 = th2_1;
    fprintf('There is one solution to IK.\n')
elseif (th2_1≠th2_2)
    theta1 = [th1_1;th1_2]; % row vectors (2xlength(t))
    theta2 = [th2_1;th2_2];
    fprintf('There are two solutions to IK.\n')
end

% Check if angles are within joint ranges (for MTA EP and WP):
for i=1:size(theta1,1) % counts to # of rows in theta1
    if (max(theta1(i,:))>deg2rad(184.5)) | ...
        (min(theta1(i,:))<deg2rad(-90.3)) % OR statement
        fprintf('Solution %1.0f falls outside theta 1 range. \n',i)
        theta1(i,:)=zeros(1,size(theta1,2)); % zero out that row in both ...
            angles
        theta2(i,:)=zeros(1,size(theta2,2));
    end
    if (max(theta2(i,:))>deg2rad(-0.6)) | ...
        (min(theta2(i,:))<deg2rad(-180.4)) % OR statement
        fprintf('Solution %1.0f falls outside theta 2 range. \n',i)
        theta2(i,:)=zeros(1,size(theta2,2)); % delete that row in both angles
        theta1(i,:)=zeros(1,size(theta1,2));
    end
end

% Delete zeroed out rows
theta1 = theta1(any(theta1,2),:);
theta2 = theta2(any(theta2,2),:);
% If both are zeroed out, returns No solution
if (any(theta1)==0 | any(theta2)==0)
    fprintf('No solution within angle ranges. \n')
end

end
```

2 DOF mass properties/simulation setup

```
%% Define the mass properties (metric units) and desired trajectories ...
    (angles) for 2 DOF planar robot
% 20 Sep 2018
% A) mass properties for notional example
```

```
% B) mass properties for MTA from CAD with sting
% C) desired trajectories (see twoDof_plotInvKin.m)
% D) inverse kinematics example

% Works with twoDof_PD, PDFF, FBL
% not with  SM,
% Will work with twoDof_adaptive control, SMTVBL, MRAC

%% (A) Example with notional values (non MTA values at all)
%{
g = 9.81;   % gravitational constant
m1 = 1;     % mass of link 1 (shoulder to elbow) kg
me = 2;     % mass of link 2 +b object
L1 = 1;     % length of link 1
r1 = 0.5;  % length from actuator 1 to centroid of link 1  (r1=l_c1)
re = 0.6;  % length from actuator 2 to centroid of link 2 +object (r2=l_c2)
I1 = 0.12;
Ie = 0.25;
delE = 30*pi/180; % angle between link two and link two+objects effective ...
    centroid line
% define parameters for ode45 and animating
param.g = g;
param.m1 = m1; param.me = me; param.m2 = me;
param.L1 = L1; param.L2 = L1;
param.r1 = r1; param.re = re; param.r2 = re;
param.I1 = I1; param.Ie = Ie; param.I2 = Ie;
param.delE = delE;
%}
%% (B) Dimensions for MTA from RE2 SolidWorks CAD file with sting
% see MTA dimensions.xls [metric units, rad]
syms t
g = 9.81;   % gravitational constant
m1 = 10.395;    % mass of link 1,kg (elbow to wrist)
m2 = 2.355;    % mass of link 2, kg (wrist + sting)
me = 2.355+2.268*sin(t);    % mass of link 2 +sting +object, kg
L1 = 1.354;    % length of link 1, m(elbow to wrist)
L2 = 0.808;    % length of link 2, m (wrist + sting)
r1 = 0.845;  % length from actuator 1 to centroid of link 1  (r1=l_c1)
r2 = 0.161;  % length from actuator 2 to centroid of link 2+sting (r2=l_c2)
re = 0.186+0.019*sin(t);   % length from actuator 2 to centroid of link 2 ...
    +sting +object(re=l_ce)
I1 = 1.598; % moment of inertia of link 1, kg-m^2
I2 = 0.095;  % moment of inertia of link 2+sting, kg-m^2
Ie = 0.1295+0.1247*sin(t);   % moment of inertia of link 2+sting+object, ...
    kg-m^2
delE = (-10+5*cos(t))*pi/180; % angle(rad) between link 2 and link ...
    2+sting+object effective centroid line
```

```
47
48 % define parameters for ode45 and animations
49 param.g = g;
50 param.m1 = m1; param.me = me; param.m2 = m2;
51 param.L1 = L1; param.L2 = L2;
52 param.r1 = r1; param.re = re; param.r2 = r2;
53 param.I1 = I1; param.Ie = Ie; param.I2 = I2;
54 param.delE = delE;
55
56 %% (C) Desired Trajectories
57 %param.inputMethod = 2;
58 % Trajectories are defined in function called below:
59 %[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
60
61 %% (D) Inverse Kinematics Example
62 % Check to make sure joint angle ranges for MTA are satisfied
63 % See twoDof_plotInvKin for more details/plots
64 % invkin2.m is the function for determing the inverse kinematics
65 % syms t
66 % t=linspace(0,3,500);
67 % y_d = 1.346+0.5*sin(2*pi*t);
68 % z_d = -0.3+0.8*cos(2*pi*t);
69 %
70 % [theta1,theta2]= invkin2(y_d,z_d,param); % should give angles symbolically
71 % hold on
72 % plot(t,rad2deg(theta1),'DisplayName',...
73 '$\theta_{1,des}$','Color','b') % plot theta1
74 % plot(t,rad2deg(theta2),'DisplayName',...
75 '$\theta_{2,des}$','Color','r') % plot theta2
76 % plot(t,-90.3*ones(size(t)),'--','color','b',...
77 'DisplayName','$\theta_{1,lim}$','linewidth',2)
78 % plot joint angle limits
79 % plot(t,184.5*ones(size(t)),'--','color','b',...
80 'HandleVisibility','off','linewidth',2)
81 % plot(t,-180.4*ones(size(t)),'--','color','r',...
82 'DisplayName','$\theta_{2,lim}$','linewidth',2)
83 % plot(t,-0.6*ones(size(t)),'--','color','r',...
84 'HandleVisibility','off','linewidth',2)
85 % legend('show')
86 % xlabel('Time (s)'); ylabel('Angle (deg)')
87 % hold off; grid on
88 % %title('Check to make sure these are within the limits')
```

PD Controller

```
1 %% PD Control Example
2 % 20 Sep 2018
```

```
clc; clear all; close all

%% INPUTS
% Set mass properties
twoDof_A_massProperties % run the file

% Set simulation time span
t0 = 0; tf = 3; tspan = [t0 tf];

% Set input method (see twoDof_DesiredTrajectories.m)
param.inputMethod =3; % 2 is step, 3 is sinusoid

% Define initial conditions for states (starts at q1_d)
if param.inputMethod ==3
    q10 = 0.9405;  dq10 = 0;   q20 = -1.7660;   dq20 = 0; % for sinusoid
elseif param.inputMethod ==2
    q10 = 0; dq10 = 0; q20 = 0; dq20 = 0; % for step
end
X0 = [q10; dq10; q20; dq20];

% Set properties specific to the control law
K_d = 100*eye(2); param.K_d = K_d;
K_p = 20*K_d; param.K_p = K_p;

% Do the simulation
options = []; [t,X] = ode45(@PDcontroller, tspan, X0, options, param);
q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
% calculate control
[q1_d ,q2_d,dq1_d,dq2_d,q1tilde, q2tilde, tau] = PDcontrol(t,X,param);

%% Plot Figures
figure () % STATES
subplot(411)
plot(t,rad2deg(q1),t,rad2deg(q1_d),'--');legend('True','Desired')
ylabel('$q_1 (deg)$'); grid minor
%title(['$PD Control: K_D=$',num2str(K_d(1)),'$*I, ...
    K_p=$',num2str(K_p(1)),'*I'])
subplot(412)
plot(t,rad2deg(q2),t,rad2deg(q2_d),'--')%;legend('True','Desired')
ylabel('$q_2 (deg)$'); grid minor
subplot(413)
plot(t,rad2deg(dq1),t,rad2deg(dq1_d),'--')%;legend('True','Desired')
ylabel('$\dot{q}_1$ (deg/s)'); grid minor
subplot(414)
plot(t,rad2deg(dq2),t,rad2deg(dq2_d),'--')%;legend('True','Desired')
xlabel('time (s)');
ylabel('$\dot{q}_2$ (deg/s)'); grid minor
```

```
%saveas(gcf,'PDstates.png')

figure () % ERROR AND CONTROL
subplot(2,2,1)
plot(t,rad2deg(q1tilde),t,rad2deg(q2tilde))
legend('$\tilde{q}_1$','$\tilde{q}_2$')
ylabel('$q_i$ Error (deg)');xlabel('time (s)');grid minor
%title(['$PD Control: K_D=$',num2str(K_d(1)),'$*I, ...
    K_p=$',num2str(K_p(1)),'*I'])
subplot(2,2,2)
plot(t,rad2deg(dq1-dq1_d),t,rad2deg(dq2-dq2_d))
legend('$\dot{\tilde{q}}_1$','$\dot{\tilde{q}}_2$')
ylabel('$\dot{q}_i$ Error (deg/s)');xlabel('time (s)');grid minor
subplot(2,2,3)
plot(t,squeeze(tau(1,:,:)),t,squeeze(tau(2,:,:)))
legend('$\tau_1$','$\tau_2$')
xlabel('time (s)'); ylabel('$\tau_i (N-m)$');grid minor
subplot(2,2,4)
plot(rad2deg(q1tilde),rad2deg(dq1-dq1_d),rad2deg(q2tilde),rad2deg(dq2-dq2_d));
%title(['\lambda=', num2str(lam),', \eta_1=', num2str(eta1)])
xlabel('$\tilde{q}_i (deg)$');
ylabel('$\dot{\tilde{q}}_i (deg/s)$')
legend('Joint 1','Joint 2');grid minor
%saveas(gcf,'PDerror.png')

%% Animate the simulation
% addpath('Draw Robot')
% z = [q1'; dq1'; q2'; dq2']; % actual states
% A.plotFunc = @(t,z)( drawRobot(t,z,param) );
% A.speed = 0.5;
% A.figNum = 101;
% %animate(t,z,A)

%% The control Law
function xdot= PDcontroller(t,X,param)
% unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
q1 = X(1); dq1 = X(2); q2 = X(3); dq2 = X(4);

% define parameters
g = param.g;
m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
L1 = param.L1; delE = param.delE;
r1 = param.r1; re = param.re;
I1 = param.I1; Ie = param.Ie;
K_d = param.K_d; K_p = param.K_p;
me = eval(me); % evaluate the time-changing parameters
re = eval(re);
```

```
Ie = eval(Ie);
delE = eval(delE);

% define true parameters
a1 = I1+m1*r1^2+Ie+me*re^2+me*L1^2;
a2 = Ie+me*re^2;
a3 = me*L1*re*cos(delE);
a4 = me*L1*re*sin(delE);

% define coefficient matrices
H11 = a1+2*a3*cos(q2)+2*a4*sin(q2);
H22 = a2;
H12 = a2+a3*cos(q2)+a4*sin(q2);
H21 = H12;
h = a3*sin(q2)-a4*cos(q2);
G1 = m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1));
G2 = me*g*re*cos(q1+q2);
H_mat = [H11 H12; H21 H22];
c_mat = [-h*dq2 -h*(dq1+dq2); h*dq1 0];

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
qtilde = [q1-q1_d; q2-q2_d];

% Calculate PD control
tau = -K_p*qtilde-K_d*[dq1;dq2]+[G1;G2];

% Calculate True Response
ddQ=inv(H_mat)*(tau-c_mat*[dq1; dq2]-[G1;G2]);
ddq1 = ddQ(1);
ddq2 = ddQ(2);

xdot = [dq1; ddq1; dq2; ddq2];
end

%% Get other Outputs
function [q1_d ,q2_d ,dq1_d,dq2_d,q1tilde, q2tilde, tau] = ...
    PDcontrol(t,X,param)
% unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
% define parameters
g = param.g;
m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
L1 = param.L1; delE = param.delE;
r1 = param.r1; re = param.re;
I1 = param.I1; Ie = param.Ie;
K_d = param.K_d; K_p = param.K_p;
```

```
me = eval(me); % evaluate the time-changing parameters
re = eval(re);
Ie = eval(Ie);
delE = eval(delE);

G1= m1*r1*g*cos(q1) + me.*g.*(re.*cos(q1+q2) + L1*cos(q1));
G2 = me.*g.*re.*cos(q1+q2);

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
q1tilde = q1-q1_d;
q2tilde = q2-q2_d;

% Calculate PD control
for i = 1:length(q1tilde)
tau(:,:,i) = -K_p*[q1tilde(i);q2tilde(i)]-K_d*[dq1(i);dq2(i)]+[G1(i);G2(i)];
end

end
```

PD-FF Controller

```
%% PD with Feedback Control
% 20 Sep 2018
% From "PD Control with Computed Feedforward of Robot Manipulators: A design
% Procedure" 1994 IEEE paper, Kelly & Salgado
% Control law assumes exact knowledge of M,C,G matrices
clc; clear all; %close all

%% INPUTS
% Set mass properties
twoDof_A_massProperties % run the file

% Set simulation time span
t0 = 0; tf = 3; tspan = [t0 tf];

% Set input method (see twoDof_DesiredTrajectories.m)
param.inputMethod =3; % 2 is step, 3 is sinusoid

% Define initial conditions for states (starts at q1_d)
if param.inputMethod ==3
    q10 = 0.9405;  dq10 = 0;   q20 = -1.7660;   dq20 = 0; % for sinusoid
elseif param.inputMethod ==2
    q10 = 0; dq10 = 0; q20 = 0; dq20 = 0; % for step
end
X0 = [q10; dq10; q20; dq20];

```

```
% Set properties specific to the control law (must be pos. def.)
K_v = 100*eye(2); param.K_v = K_v;
K_p = 20*K_v; param.K_p = K_p;

% Do the simulation
options = []; [t,X] = ode45(@PDcontroller, tspan, X0, options, param);
q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);

% Calculate control
[q1_d ,q2_d,dq1_d,dq2_d,qtilde, qtilde_dot, tau] = PDcontrol(t,X,param);

%% Plot Figures
figure (1) % STATES
subplot(411)
plot(t,rad2deg(q1),t,rad2deg(q1_d),'--');legend('True','desired')
ylabel('$q_1 (deg)$'); grid minor
%title(['PD FF: $K_v$=',num2str(K_v(1)),'I, $K_p$=',num2str(K_p(1)),'I'])
subplot(412)
plot(t,rad2deg(q2),t,rad2deg(q2_d),'--')%;legend('True','desired')
ylabel('$q_2 (deg)$'); grid minor
subplot(413)
plot(t,rad2deg(dq1),t,rad2deg(dq1_d),'--')%;legend('True','desired')
ylabel('$\dot{q}_1$ (deg/s)'); grid minor
subplot(414)
plot(t,rad2deg(dq2),t,rad2deg(dq2_d),'--')%;legend('True','desired')
xlabel('time (s)');
ylabel('$\dot{q}_2$ (deg/s)'); grid minor
%saveas(gcf,'PDFFstates.png')

figure (2) % ERROR AND CONTROL
subplot(2,2,1)
plot(t,rad2deg(qtilde(1,:)),t,rad2deg(qtilde(2,:)))
legend('$\tilde{q}_1$','$\tilde{q}_2$')
ylabel('$q_i$ Error (deg)');xlabel('time (s)');grid minor
%title(['PD FF: $K_v$=',num2str(K_v(1)),'I, $K_p$=',num2str(K_p(1)),'I'])
subplot(2,2,2)
plot(t,rad2deg(qtilde_dot(1,:)),t,rad2deg(qtilde_dot(2,:)))
legend('$\dot{\tilde{q}}_1$','$\dot{\tilde{q}}_2$')
ylabel('$\dot{q}_i$ Error (deg/s)');xlabel('time (s)');grid minor
subplot(2,2,3)
plot(t,squeeze(tau(1,:,:)),t,squeeze(tau(2,:,:)))
legend('$\tau_1$','$\tau_2$')
xlabel('time (s)'); ylabel('$\tau_i (N-m)$');grid minor
subplot(2,2,4)
plot(rad2deg(qtilde(1,:)),rad2deg(qtilde_dot(1,:)),rad2deg(qtilde(2,:)),...
rad2deg(qtilde_dot(2,:))); %title(['\lambda=', num2str(lam),', \eta_1=', ...
    num2str(eta1)])
```

```
xlabel('$\tilde{q}_i (deg)$');
ylabel('$\dot{\tilde{q}}_i (deg/s)$')
legend('Joint 1','Joint 2');grid minor
%saveas(gcf,'PDFFerror.png')

%% Animate the simulation
% addpath('Draw Robot')
% z = [q1'; dq1'; q2'; dq2']; % actual states
% A.plotFunc = @(t,z)( drawRobot(t,z,param) );
% A.speed = 0.5;
% A.figNum = 101;
% %animate(t,z,A)

%% The control Law
function xdot= PDcontroller(t,X,param)
% unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
q1 = X(1); dq1 = X(2); q2 = X(3); dq2 = X(4);

% define parameters
g = param.g;
m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
L1 = param.L1; delE = param.delE;
r1 = param.r1; re = param.re;
I1 = param.I1; Ie = param.Ie;
K_v = param.K_v; K_p = param.K_p;
me = eval(me); % evaluate the time-changing parameters
re = eval(re);
Ie = eval(Ie);
delE = eval(delE);

% define true parameters
a1 = I1+m1*r1^2+Ie+me*re^2+me*L1^2;
a2 = Ie+me*re^2;
a3 = me*L1*re*cos(delE);
a4 = me*L1*re*sin(delE);

% define coefficient matrices
H11 = a1+2*a3*cos(q2)+2*a4*sin(q2);
H22 = a2;
H12 = a2+a3*cos(q2)+a4*sin(q2);
H21 = H12;
h = a3*sin(q2)-a4*cos(q2);
G1 = m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1));
G2 = me*g*re*cos(q1+q2);
H = [H11 H12; H21 H22];
C = [-h*dq2 -h*(dq1+dq2); h*dq1 0];

```

```
119 % desired trajectories
120 [q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
121
122 qtilde = [q1-q1_d; q2-q2_d];
123 qtilde_dot = [dq1-dq1_d; dq2-dq2_d];
124
125 % Calculate Feedforward Terms (assume model parameters are known)
126 H11_d = a1+2*a3*cos(q2_d)+2*a4*sin(q2_d);
127 H22_d = a2;
128 H12_d = a2+a3*cos(q2_d)+a4*sin(q2_d);
129 H21_d = H12_d;
130 h_d = a3*sin(q2_d)-a4*cos(q2_d);
131 G1_d = m1*r1*g*cos(q1_d) + me*g*(re*cos(q1_d+q2_d) + L1*cos(q1_d));
132 G2_d = me*g*re*cos(q1_d+q2_d);
133 H_d = [H11_d H12_d; H21_d H22_d];
134 C_d = [-h_d*dq2_d -h_d*(dq1_d+dq2_d); h_d*dq1_d 0];
135 G_d = [G1_d;G2_d];
136
137 % The Full Control Law: PD control with Feedforward
138 % tau=Kp*qtilde+Kv*qtilde_dot+M(q_d)*ddq_d+C(q_d,dq_d)*dq_d+G(q_d);
139 % feedforward part starts here^
140 tau = -K_p*qtilde-K_v*qtilde_dot+ H_d*[ddq1_d; ddq2_d]+C_d*[dq1_d; ...
        dq2_d]+G_d;
141
142 % Calculate True Response
143 ddQ=inv(H)*(tau-C*[dq1; dq2]-[G1;G2]);
144 ddq1 = ddQ(1);
145 ddq2 = ddQ(2);
146
147 xdot = [dq1; ddq1; dq2; ddq2];
148 end
149
150 %% Get other Outputs
151 function [q1_d ,q2_d ,dq1_d,dq2_d,qtilde, qtilde_dot, tau] = ...
        PDcontrol(t,X,param)
152 % unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
153
154 q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
155 % define parameters
156 g = param.g;
157 m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
158 L1 = param.L1; delE = param.delE;
159 r1 = param.r1; re = param.re;
160 I1 = param.I1; Ie = param.Ie;
161 K_v = param.K_v; K_p = param.K_p;
162 me = eval(me); % evaluate the time-changing parameters
163 re = eval(re);
```

```
Ie = eval(Ie);
delE = eval(delE);

% define true parameters
a1 = I1+m1*r1^2+Ie+me.*re.^2+me.*L1^2;
a2 = Ie+me.*re.^2;
a3 = me.*L1.*re.*cos(delE);
a4 = me.*L1.*re.*sin(delE);
G1 = m1*r1*g.*cos(q1) + me.*g.*(re.*cos(q1+q2) + L1.*cos(q1));
G2 = me.*g.*re.*cos(q1+q2);

% desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);

qtilde = [q1-q1_d, q2-q2_d]';
qtilde_dot = [dq1-dq1_d, dq2-dq2_d]';

% Calculate Feedforward Terms (assume model parameters are known)
    H11_d = a1+2*a3.*cos(q2_d)+2*a4.*sin(q2_d);
    H22_d = a2;
    H12_d = a2+a3.*cos(q2_d)+a4.*sin(q2_d);
    H21_d = H12_d;
    h_d = a3.*sin(q2_d)-a4.*cos(q2_d);
    G1_d = m1*r1*g.*cos(q1_d) + me.*g.*(re.*cos(q1_d+q2_d) + L1.*cos(q1_d));
    G2_d = me.*g.*re.*cos(q1_d+q2_d);
for i=1:length(q1)
    H_d(:,:,i) = [H11_d(i) H12_d(i); H21_d(i) H22_d(i)];
    C_d(:,:,i) = [-h_d(i)*dq2_d(i) -h_d(i)*(dq1_d(i)+dq2_d(i)); ...
        h_d(i)*dq1_d(i) 0];
end
    G_d = [G1_d';G2_d'];

% Calculate PD control
for i = 1:length(q1)
% The Full Control Law: PD control with Feedforward
tau(:,i) = -K_p*qtilde(:,i)-K_v*qtilde_dot(:,i)+ H_d(:,:,i)*[ddq1_d(i); ...
    ddq2_d(i)]+C_d(:,:,i)*[dq1_d(i); dq2_d(i)]+G_d(:,i);
end

end
```

Feedback Linearization

```
%% Feedback Linearization Control
% Aug 2018
% Control law assumes exact knowledge of M,C,G matrices
clc; clear all; close all
```

```
5
6  %% INPUTS
7  % Set mass properties
8  twoDof_A_massProperties % run the file
9
10 % Set simulation time span
11 t0 = 0; tf = 0.75; tspan = [t0 tf];
12
13 % Set input method (see twoDof_DesiredTrajectories.m)
14 param.inputMethod =3; % 2 is step, 3 is sinusoid
15
16 % Define initial conditions for states (starts at q1_d)
17 if param.inputMethod ==3
18     q10 = 0.9405;  dq10 = 0;   q20 = -1.7660;   dq20 = 0; % for sinusoid
19 elseif param.inputMethod ==2
20     q10 = 0; dq10 = 0; q20 = 0; dq20 = 0; % for step
21 end
22 X0 = [q10; dq10; q20; dq20];
23
24 % Set properties specific to the control law
25 lam = 100; param.lam = lam;
26
27 % Do the simulation
28 options = []; [t,X] = ode45(@feedbackLinController, tspan, X0, options, ...
       param);
29 q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
30 % calculate control
31 [q1_d ,q2_d,dq1_d,dq2_d,qtilde, qtilde_dot, tau] = ...
       feedbackLinControl(t,X,param);
32
33 %% Plots
34 figure (1) % STATES
35 subplot(411)
36 plot(t,rad2deg(q1),t,rad2deg(q1_d),'--');legend('True','desired')
37 ylabel('$q_1 (deg)$'); grid minor;xlim([0 0.75])
38 %title(['FBL Control: $\lambda$=',num2str(lam)])
39 subplot(412)
40 plot(t,rad2deg(q2),t,rad2deg(q2_d),'--');%legend('True','desired')
41 ylabel('$q_2 (deg)$'); grid minor;xlim([0 0.75])
42 subplot(413)
43 plot(t,rad2deg(dq1),t,rad2deg(dq1_d),'--');%legend('True','desired')
44 ylabel('$\dot{q}_1$ (deg/s)'); grid minor;xlim([0 0.75])
45 subplot(414)
46 plot(t,rad2deg(dq2),t,rad2deg(dq2_d),'--');%legend('True','desired')
47 xlabel('time (s)');
48 ylabel('$\dot{q}_2$ (deg/s)'); grid minor;xlim([0 0.75])
49 %saveas(gcf,'FBLstates.png')
```

```
50
51 figure () % ERROR AND CONTROL
52 subplot(2,2,1)
53 plot(t,rad2deg(qtilde(1,:)),t,rad2deg(qtilde(2,:)))
54 legend('$\tilde{q}_1$','$\tilde{q}_2$');xlim([0 0.75])
55 ylabel('$q_i$ Error (deg)');xlabel('time (s)');grid minor
56 %title(['FBL Control: $$\lambda$$=',num2str(lam)])
57 subplot(2,2,2)
58 plot(t,rad2deg(qtilde_dot(1,:)),t,rad2deg(qtilde_dot(2,:)))
59 legend('$\dot{\tilde{q}}_1$','$\dot{\tilde{q}}_2$');xlim([0 0.75])
60 ylabel('$\dot{q}_i$ Error (deg/s)');xlabel('time (s)');grid minor
61 subplot(2,2,3)
62 plot(t,squeeze(tau(1,:,:)),t,squeeze(tau(2,:,:)))
63 legend('$\tau_1$','$\tau_2$');xlim([0 0.75])
64 xlabel('time (s)'); ylabel('$\tau_i (N-m)$');grid minor
65 subplot(2,2,4)
66 plot(rad2deg(qtilde(1,:)),rad2deg(qtilde_dot(1,:)),...
67 rad2deg(qtilde(2,:)),rad2deg(qtilde_dot(2,:)));
68 %title(['\lambda=', num2str(lam),', \eta_1=', num2str(eta1)])
69 xlabel('$\tilde{q}_i (deg)$')
70 ylabel('$\dot{\tilde{q}}_i (deg/s)$')
71 legend('Joint 1','Joint 2');grid minor
72 %saveas(gcf,'FBLerror.png')
73 twoDof_CompareTraj
74 %% animate
75 % z = [q1'; dq1'; q2'; dq2']; % actual states
76 % A.plotFunc = @(t,z)( drawRobot(t,z,param) );
77 % A.speed = 0.25;
78 % A.figNum = 101;
79 % animate(t,z,A)
80 %% The Function
81 function xdot= feedbackLinController(t,X,param)
82 % unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
83 q1 = X(1); dq1 = X(2); q2 = X(3); dq2 = X(4);
84
85 % define parameters
86 g = param.g;
87 m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
88 L1 = param.L1; delE = param.delE;
89 r1 = param.r1; re = param.re;
90 I1 = param.I1; Ie = param.Ie;
91 lam = param.lam;
92 me = eval(me); % evaluate the time-changing parameters
93 re = eval(re);
94 Ie = eval(Ie);
95 delE = eval(delE);
96
```

```
% define true parameters
a1 = I1+m1*r1^2+Ie+me*re^2+me*L1^2;
a2 = Ie+me*re^2;
a3 = me*L1*re*cos(delE);
a4 = me*L1*re*sin(delE);

% define coefficient matrices
H11 = a1+2*a3*cos(q2)+2*a4*sin(q2);
H22 = a2;
H12 = a2+a3*cos(q2)+a4*sin(q2);
H21 = H12;
h = a3*sin(q2)-a4*cos(q2);
G1 = m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1)); % gravity terms
G2 = me*g*re*cos(q1+q2);
H_mat = [H11 H12; H21 H22];
c_mat = [-h*dq2 -h*(dq1+dq2); h*dq1 0];

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);

qtilde = [q1-q1_d; q2-q2_d];
qtilde_dot = [dq1-dq1_d; dq2-dq2_d];

v = [ddq1_d;ddq2_d]-2*lam*qtilde_dot-lam^2*qtilde;

% Calculate control law (torque)
tau =H_mat*v+c_mat*[dq1; dq2]+[G1;G2];

% Calculate True Response
ddQ=inv(H_mat)*(tau-c_mat*[dq1; dq2]-[G1;G2]);
ddq1 = ddQ(1);
ddq2 = ddQ(2);


xdot = [dq1; ddq1; dq2; ddq2];
end

%% want to get control as output
function [q1_d ,q2_d,dq1_d,dq2_d,qtilde, qtilde_dot, tau] = ...
    feedbackLinControl(t,X,param)
% unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);

% define parameters
g = param.g;
m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
L1 = param.L1; delE = param.delE;
```

```
r1 = param.r1; re = param.re;
I1 = param.I1; Ie = param.Ie;
lam = param.lam;
me = eval(me); % evaluate the time-changing parameters
re = eval(re);
Ie = eval(Ie);
delE = eval(delE);

% define true parameters
a1 = I1+m1*r1^2+Ie+me.*re.^2+me.*L1^2;
a2 = Ie+me.*re.^2;
a3 = me.*L1.*re.*cos(delE);
a4 = me.*L1.*re.*sin(delE);

% define coefficient matrices
H11 = a1+2*a3.*cos(q2)+2*a4.*sin(q2);
H22 = a2;
H12 = a2+a3.*cos(q2)+a4.*sin(q2);
H21 = H12;
h = a3.*sin(q2)-a4.*cos(q2);
G1 = m1*r1*g*cos(q1) + me*g.*(re.*cos(q1+q2) + L1.*cos(q1)); % gravity
G2 = me.*g.*re.*cos(q1+q2);
for i=1:length(H11)
H_mat(:,:,i) = [H11(i) H12(i); H21(i) H22(i)];
c_mat(:,:,i) = [-h(i)*dq2(i) -h(i)*(dq1(i)+dq2(i)); h(i)*dq1(i) 0];
end

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
qtilde = [q1-q1_d, q2-q2_d]';
qtilde_dot = [dq1-dq1_d, dq2-dq2_d]';

v = [ddq1_d,ddq2_d]'-2*lam*qtilde_dot-lam^2*qtilde;

% Calculate control law (torque)

for i = 1:length(dq1)
tau(:,:,i) =H_mat(:,:,i)*v(:,i)+c_mat(:,:,i)*[dq1(i); dq2(i)]+[G1(i);G2(i)];
end

end
```

Sliding Mode Controller

```
%% Sliding Mode Control
% Sep 2018
% Control law assumes exact knowledge of dynamics structure
```

```
clc; clear all; %close all

%% INPUTS
% Set mass properties
twoDof_A_massProperties % run the file

% Set simulation time span
t0 = 0; tf = 3; tspan = [t0 tf];

% Set input method (see twoDof_DesiredTrajectories.m)
param.inputMethod =3; % 2 is step, 3 is sinusoid

% Define initial conditions for states (starts at q1_d)
if param.inputMethod ==3
    q10 = 0.9405;  dq10 = 0;   q20 = -1.7660;   dq20 = 0; % for sinusoid
elseif param.inputMethod ==2
    q10 = 0; dq10 = 0; q20 = 0; dq20 = 0; % for step
end
X0 = [q10; dq10; q20; dq20];

% Set properties specific to the control law
lam = 20; param.lam = lam;
eta1 = 0.1; param.eta1 = eta1;
eta2 = 0.1; param.eta2 = eta2;
mInacc = 1.05; param.mInacc = mInacc; % percent inaccuracy on mass ...
    parameters (estimated are 120%)
phi1 = 0.05; param.phi1 = phi1;  % boundary layer thickness
phi2 = 0.05; param.phi2 = phi2;

options = []; [t,X] = ode45(@slideModeController, tspan, X0, options, param);
q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
% % Calculate control
[q1_d ,q2_d,dq1_d,dq2_d,qtilde, qtilde_dot, tau,s,a_rlz,ahat] = ...
    slideModeControl(t,X,param);
q1tilde = qtilde(1,:); q2tilde = qtilde(2,:);
a1=a_rlz(:,1); a2=a_rlz(:,2); a3=a_rlz(:,3); a4=a_rlz(:,4);
a1hat=ahat(:,1); a2hat=ahat(:,2); a3hat=ahat(:,3); a4hat=ahat(:,4);

%% Plots

figure (1) % STATES
subplot(411)
plot(t,rad2deg(q1),t,rad2deg(q1_d),'--');legend('True','desired')
ylabel('$q_1 (deg)$'); grid minor
%title(['SMC: $\lambda$=',num2str(lam),', $\eta_1$=',num2str(eta1),', ...
    $\eta_2$=',num2str(eta2),', constant $\phi$'])
subplot(412)
```

```
plot(t,rad2deg(q2),t,rad2deg(q2_d),'--');legend('True','desired')
ylabel('$q_2 (deg)$'); grid minor
subplot(413)
plot(t,rad2deg(dq1),t,rad2deg(dq1_d),'--');legend('True','desired')
ylabel('$\dot{q}_1$ (deg/s)'); grid minor
subplot(414)
plot(t,rad2deg(dq2),t,rad2deg(dq2_d),'--');legend('True','desired')
xlabel('time (s)');
ylabel('$\dot{q}_2$ (deg/s)'); grid minor
%saveas(gcf,'SMCstates.png')

figure (2) % ERROR
subplot(2,2,1)
plot(t,rad2deg(qtilde(1,:)),t,rad2deg(qtilde(2,:)))
legend('$\tilde{q}_1$','$\tilde{q}_2$')
ylabel('$q_i$ Error (deg)');xlabel('time (s)');grid minor
%title(['SMC: $\lambda$=',num2str(lam),'$, \eta_1$=',num2str(eta1),'$, ...
    \eta_2$=',num2str(eta2)])
subplot(2,2,2)
plot(t,rad2deg(qtilde_dot(1,:)),t,rad2deg(qtilde_dot(2,:)))
legend('$\dot{\tilde{q}}_1$','$\dot{\tilde{q}}_2$')
ylabel('$\dot{q}_i$ Error (deg/s)');xlabel('time (s)');grid minor
subplot(2,2,3)
plot(t,squeeze(tau(1,:,:)),t,squeeze(tau(2,:,:)))
legend('$\tau_1$','$\tau_2$')
xlabel('time (s)'); ylabel('$\tau_i (N-m)$');grid minor
subplot(2,2,4)
plot(rad2deg(q1tilde),rad2deg(dq1-dq1_d),...
rad2deg(q2tilde),rad2deg(dq2-dq2_d));
%title(['\lambda=', num2str(lam),', \eta_1=', num2str(eta1)])
xlabel('$\tilde{q}_i (deg)$');
ylabel('$\dot{\tilde{q}}_i (deg/s)$')
legend('Joint 1','Joint 2');grid minor
%saveas(gcf,'SMCerror.png')

figure (3)  % SLIDING SURFACE & BOUNDARY LAYER
subplot(2,1,1)
plot(t,s);
ylabel('$s(t)$'); legend('$$s_1$$','$$s_2$$');grid minor
%title('Sliding Surface')
subplot(2,1,2)
plot(t,phi1*ones(length(t)),t,phi2*ones(length(t)),':')
xlabel('t(sec)'); ylabel('$\phi$');
legend('$$\phi_1$$','$$\phi_2$$');grid minor
%title('Constant Boundary Layer')
%saveas(gcf,'SMCsurface.png')

```

```
figure (4)  % PARAMETERS
subplot(221)
plot(t,a1,t,a1hat)
legend('$$a_1 \ true$$','$$\hat{a}_1$$')
xlabel('time (s)'); grid minor;
%title('Inaccuracy in Parameters')
subplot(222)
plot(t,a2,t,a2hat)
legend('$$a_2 \ true$$','$$\hat{a}_2$$')
xlabel('time (s)'); grid minor
subplot(223)
plot(t,a3,t,a3hat)
legend('$$a_3 \ true$$','$$\hat{a}_3$$')
xlabel('time (s)'); grid minor
subplot(224)
plot(t,a4,t,a4hat)
legend('$$a_4 \ true$$','$$\hat{a}_4$$')
xlabel('time (s)'); grid minor
%saveas(gcf,'SMCwTVBLparam.png')

 %% animate
% z = [q1'; dq1'; q2'; dq2']; % actual states
% A.plotFunc = @(t,z)( drawRobot(t,z,param) );
% A.speed = 0.25;
% A.figNum = 101;
% %animate(t,z,A)

%%
function xdot= slideModeController(t,X,param)
% unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
q1 = X(1); dq1 = X(2); q2 = X(3); dq2 = X(4);

% define parameters
g = param.g;
m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
L1 = param.L1; delE = param.delE;
r1 = param.r1; re = param.re;
I1 = param.I1; Ie = param.Ie;
me = eval(me); % evaluate the time-changing parameters
re = eval(re);
Ie = eval(Ie);
delE = eval(delE);

lam = param.lam;
eta1 = param.eta1;
eta2 = param.eta2;
mInacc = param.mInacc;
```

```
141 phi1 = param.phi1;
142 phi2 = param.phi2;
143
144 % define true parameters/coefficients
145 a1 = I1+m1*r1^2+Ie+me*re^2+me*L1^2;
146 a2 = Ie+me*re^2;
147 a3 = me*L1*re*cos(delE);
148 a4 = me*L1*re*sin(delE);
149 H11 = a1+2*a3*cos(q2)+2*a4*sin(q2);
150 H22 = a2;
151 H12 = a2+a3*cos(q2)+a4*sin(q2);
152 H21 = H12;
153 h = a3*sin(q2)-a4*cos(q2);
154 G1 = m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1));
155 G2 = me*g*re*cos(q1+q2);
156 H_mat = [H11 H12; H21 H22];
157 c_mat = [-h*dq2 -h*(dq1+dq2); h*dq1 0];
158 g_mat = [G1;G2];
159
160 % Estimated coefficient matrices
161 a1hat = mInacc*a1;
162 a2hat = mInacc*a2;
163 a3hat = mInacc*a3;
164 a4hat = mInacc*a4;
165 H11hat = a1hat+2*a3hat*cos(q2)+2*a4hat*sin(q2);
166 H22hat = a2hat;
167 H12hat = a2hat+a3hat*cos(q2)+a4hat*sin(q2);
168 H21hat = H12hat;
169 hhat = a3hat*sin(q2)-a4hat*cos(q2);
170 G1_hat = mInacc*(m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1)));
171 G2_hat = mInacc*(me*g*re*cos(q1+q2));
172 H_mathat = [H11hat H12hat; H21hat H22hat];
173 c_mathat = [-hhat*dq2 -hhat*(dq1+dq2); hhat*dq1 0];
174 g_mathat = [G1;G2];
175 % Coefficient Error Matrices
176 Htilde = H_mathat-H_mat;
177 ctilde = c_mathat-c_mat;
178 gtilde = g_mathat-g_mat;
179
180 % Desired trajectories
181 [q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
182 qtilde = [q1-q1_d; q2-q2_d];   % position error
183 qtilde_dot = [dq1-dq1_d; dq2-dq2_d]; % velocity error
184
185 % Calculate s (sliding surface)
186 s = qtilde_dot +lam*qtilde;
187 s1 = s(1);  s2 = s(2);
```

```
188
189 % define reference velocities
190 dq_r = [dq1_d;dq2_d]-lam*qtilde;
191 dq1_r = dq_r(1); dq2_r = dq_r(2);
192 ddq_r = [ddq1_d;ddq2_d]-lam*qtilde_dot;
193 ddq1_r = ddq_r(1); ddq2_r = ddq_r(2);
194
195 % Calculate control law (tau = tau_hat-k*sign(s))
196 tau_hat = H_mathat*ddq_r+c_mathat*dq_r+g_mathat;
197 k = (abs(Htilde*ddq_r+ctilde*dq_r+gtilde)+[eta1;eta2])';
198 tau = tau_hat-k*[sat(s1,phi1);sat(s2,phi2)];
199
200 % Calculate True Response
201 ddQ=inv(H_mat)*(tau-c_mat*[dq1; dq2]-g_mat);
202 ddq1 = ddQ(1);
203 ddq2 = ddQ(2);
204
205
206 xdot = [dq1; ddq1; dq2; ddq2];
207 end
208
209 %% want to get control as output
210 function [q1_d ,q2_d,dq1_d,dq2_d,qtilde,qtilde_dot,tau,s,a_rlz,ahat] ...
       =slideModeControl(t,X,param)
211 % unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
212 q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
213
214 % define parameters
215 g = param.g;
216 m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
217 L1 = param.L1; delE = param.delE;
218 r1 = param.r1; re = param.re;
219 I1 = param.I1; Ie = param.Ie;
220 me = eval(me); % evaluate the time-changing parameters
221 re = eval(re);
222 Ie = eval(Ie);
223 delE = eval(delE);
224
225 lam = param.lam;
226 eta1 = param.eta1;
227 eta2 = param.eta2;
228 mInacc = param.mInacc;
229 phi1 = param.phi1;
230 phi2 = param.phi2;
231
232 % define true parameters/coefficients
233 a1 = I1+m1*r1^2+Ie+me.*re.^2+me.*L1^2;
```

```
a2 = Ie+me.*re.^2;
a3 = me.*L1.*re.*cos(delE);
a4 = me.*L1.*re.*sin(delE);
H11 = a1+2*a3.*cos(q2)+2*a4.*sin(q2);
H22 = a2;
H12 = a2+a3.*cos(q2)+a4.*sin(q2);
H21 = H12;
h = a3.*sin(q2)-a4.*cos(q2);
G1 = m1*r1*g.*cos(q1) + me.*g.*(re.*cos(q1+q2) + L1.*cos(q1));
G2 = me.*g.*re.*cos(q1+q2);
for i=1:length(H11)
H_mat(:,:,i) = [H11(i) H12(i); H21(i) H22(i)];
c_mat(:,:,i) = [-h(i)*dq2(i) -h(i)*(dq1(i)+dq2(i)); h(i)*dq1(i) 0];
g_mat(:,:,i) = [G1(i);G2(i)];
end

% Estimated coefficient matrices
a1hat = mInacc*a1;
a2hat = mInacc*a2;
a3hat = mInacc*a3;
a4hat = mInacc*a4;
H11hat = a1hat+2*a3hat.*cos(q2)+2*a4hat.*sin(q2);
H22hat = a2hat;
H12hat = a2hat+a3hat.*cos(q2)+a4hat.*sin(q2);
H21hat = H12;
hhat = a3hat.*sin(q2)-a4hat.*cos(q2);
G1_hat = mInacc*(m1*r1*g.*cos(q1) + me.*g.*(re.*cos(q1+q2) + L1*cos(q1)));
G2_hat = mInacc*(me.*g.*re.*cos(q1+q2));
for i=1:length(H11hat)
H_mathat(:,:,i) = [H11hat(i)  H12hat(i); H21hat(i)  H22hat(i)];
c_mathat(:,:,i) = [-hhat(i)*dq2(i) -hhat(i)*(dq1(i)+dq2(i));...
hhat(i)*dq1(i) 0];
g_mathat(:,:,i) = [G1_hat(i);G2_hat(i)];
end
% Coefficient errors
Htilde = H_mathat-H_mat;
ctilde = c_mathat-c_mat;
gtilde = g_mathat-g_mat;

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
qtilde = [q1-q1_d, q2-q2_d]';
qtilde_dot = [dq1-dq1_d, dq2-dq2_d]';

% Calculate s (sliding surface)
for i=1:length(qtilde)
s(:,i)= qtilde_dot(:,i)+lam.*qtilde(:,i);
```

```
281 end
282 s1 = squeeze(s(1,:)); s2 = squeeze(s(2,:));
283
284 % Define dynamics matrix (Y)
285 dq_r = [dq1_d,dq2_d]'-lam*qtilde; % reference velocity
286 dq1_r = dq_r(1,:)'; dq2_r = dq_r(2,:)';
287 ddq_r = [ddq1_d,ddq2_d]'-lam*qtilde_dot;
288 ddq1_r = ddq_r(1,:)'; ddq2_r = ddq_r(2,:)';
289
290
291 % Calculate control law (tau = tau_hat-k*sign(s))
292 for i = 1:length(H_mathat)
293 tau_hat(:,:,i) = H_mathat(:,:,i)*[ddq1_r(i); ...
        ddq2_r(i)]+c_mathat(:,:,i)*[dq1_r(i);dq2_r(i)]+g_mathat(:,:,i);
294 k(:,:,i) = (abs(Htilde(:,:,i)*[ddq1_r(i); ...
        ddq2_r(i)]+ctilde(:,:,i)*[dq1_r(i);dq2_r(i)]+gtilde(:,:,i))+[eta1;eta2])';
295 tau(:,:,i) = tau_hat(:,:,i)-k(:,:,i)*[sat(s1(i),phi1);sat(s2(i),phi2)];
296 end
297
298 % For output to compare
299 a_rlz=[a1, a2, a3, a4];  % real
300 ahat=[a1hat, a2hat, a3hat, a4hat]; % with inaccuracy
301 end
302
303 function y=sat(s,phi) % function from TimeVaringBLCode.m (Capt Hess)
304      % sat is the saturation function with unit limits and unit slope.
305      if abs(s)>phi
306          % elseif x<-Δ
307          y=sign(s);
308      else y=s./phi;
309      end
310  end
```

Sliding Mode Controller, time-varying Boundary Layer

```
1  %% Sliding Mode WITH time-varying boundary layer
2  % Sep 2018
3  % Control law assumes exact knowledge of dynamics structure
4  clc; clear all; close all
5
6  %% INPUTS
7  % Set mass properties
8  twoDof_A_massProperties % run the file
9
10 % Set simulation time span
11 t0 = 0; tf = 3; tspan = [t0 tf];
12
```

```
% Set input method (see twoDof_DesiredTrajectories.m)
param.inputMethod =3; % 2 is step, 3 is sinusoid

% Define initial conditions for states (starts at q1_d)
if param.inputMethod ==3
    q10 = 0.9405;  dq10 = 0;   q20 = -1.7660;   dq20 = 0; % for sinusoid
elseif param.inputMethod ==2
    q10 = 0; dq10 = 0; q20 = 0; dq20 = 0; % for step
end
X0 = [q10; dq10; q20; dq20; 0.05; 0.05];  % phi1,phi2

% Set properties specific to the control law
lam = 20; param.lam = lam;
eta1 = 0.1; param.eta1 = eta1;
eta2 = 0.1; param.eta2 = eta2;
mInacc = 1.01; param.mInacc = mInacc; % percent inaccuracy on mass ...
    parameters (estimated are 120%)

options = []; [t,X] = ode45(@slideModeBVController, tspan, X0, options, ...
    param);
q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
phi1 = X(:,5); phi2 = X(:,6);

[q1_d ,q2_d,dq1_d,dq2_d,qtilde, qtilde_dot, tau,s,a_rlz,ahat] = ...
    slideModeBVControl(t,X,param);
q1tilde = qtilde(1,:); q2tilde = qtilde(2,:); s=squeeze(s);
a1=a_rlz(:,1); a2=a_rlz(:,2); a3=a_rlz(:,3); a4=a_rlz(:,4);
a1hat=ahat(:,1); a2hat=ahat(:,2); a3hat=ahat(:,3); a4hat=ahat(:,4);

%% Plots

figure (1) % STATES
subplot(411)
plot(t,rad2deg(q1),t,rad2deg(q1_d),'--');legend('True','desired')
ylabel('$q_1 (deg)$'); grid minor
%title(['SMC: $\lambda$=',num2str(lam),', $\eta_1$=',num2str(eta1),', ...
    $\eta_2$=',num2str(eta2),', Varying $\phi$'])
subplot(412)
plot(t,rad2deg(q2),t,rad2deg(q2_d),'--')%;legend('True','desired')
ylabel('$q_2 (deg)$'); grid minor
subplot(413)
plot(t,rad2deg(dq1),t,rad2deg(dq1_d),'--')%;legend('True','desired')
ylabel('$\dot{q}_1$ (deg/s)'); grid minor
subplot(414)
plot(t,rad2deg(dq2),t,rad2deg(dq2_d),'--')%;legend('True','desired')
xlabel('time (s)');
ylabel('$\dot{q}_2$ (deg/s)'); grid minor
```

```
%saveas(gcf,'SMCwTVBLstates.png')

figure (2) % ERROR
subplot(2,2,1)
plot(t,rad2deg(qtilde(1,:)),t,rad2deg(qtilde(2,:)))
legend('$\tilde{q}_1$','$\tilde{q}_2$')
ylabel('$q_i$ Error (deg)');xlabel('time (s)');grid minor
%title(['SMC: $\lambda$=',num2str(lam),'$, \eta_1$=',num2str(eta1),'$, ...
    \eta_2$=',num2str(eta2)])
subplot(2,2,2)
plot(t,rad2deg(qtilde_dot(1,:)),t,rad2deg(qtilde_dot(2,:)))
legend('$\dot{\tilde{q}}_1$','$\dot{\tilde{q}}_2$')
ylabel('$\dot{q}_i$ Error (deg/s)');xlabel('time (s)');grid minor
subplot(2,2,3)
plot(t,squeeze(tau(1,:,:)),t,squeeze(tau(2,:,:)))
legend('$\tau_1$','$\tau_2$')
xlabel('time (s)'); ylabel('$\tau_i (N-m)$');grid minor
subplot(2,2,4)
plot(rad2deg(q1tilde),rad2deg(dq1-dq1_d),...
rad2deg(q2tilde),rad2deg(dq2-dq2_d));
%title(['\lambda=', num2str(lam),', \eta_1=', num2str(eta1)])
xlabel('$\tilde{q}_i (deg)$');
ylabel('$\dot{\tilde{q}}_i (deg/s)$')
legend('Joint 1','Joint 2');grid minor
%saveas(gcf,'SMCwTVBLerror.png')

figure (3)  % SLIDING SURFACE & BOUNDARY LAYER
subplot(2,1,1)
plot(t,s);
xlabel('t(sec)'); ylabel('$s(t)$');
legend('$$s_1$$','$$s_2$$');grid minor
%title('Sliding Surface')
subplot(2,1,2)
plot(t,phi1,t,phi2,':')
xlabel('t(sec)'); ylabel('$\phi$');
legend('$$\phi_1$$','$$\phi_2$$');grid minor
%title('Time-Varying Boundary Layer')
%saveas(gcf,'SMCwTVBLsurface.png')

figure (4)  % PARAMETERS
subplot(221)
plot(t,a1,t,a1hat)
legend('$$a_1 \ true$$','$$\hat{a}_1$$')
xlabel('time (s)'); grid minor;
%title('Inaccuracy in Parameters')
subplot(222)
plot(t,a2,t,a2hat)
```

```
legend('$$a_2 \ true$$','$$\hat{a}_2$$')
xlabel('time (s)'); grid minor
subplot(223)
plot(t,a3,t,a3hat)
legend('$$a_3 \ true$$','$$\hat{a}_3$$')
xlabel('time (s)'); grid minor
subplot(224)
plot(t,a4,t,a4hat)
legend('$$a_4 \ true$$','$$\hat{a}_4$$')
xlabel('time (s)'); grid minor
%saveas(gcf,'SMCwTVBLparam.png')

%% animate
% z = [q1'; dq1'; q2'; dq2']; % actual states
% A.plotFunc = @(t,z)( drawRobot(t,z,param) );
% A.speed = 0.25;
% A.figNum = 101;
% %animate(t,z,A)
%%
function xdot= slideModeBVController(t,X,param)
% unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
q1 = X(1); dq1 = X(2); q2 = X(3); dq2 = X(4);
phi1 = X(5); phi2 = X(6);

% define parameters
g = param.g;
m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
L1 = param.L1; delE = param.delE;
r1 = param.r1; re = param.re;
I1 = param.I1; Ie = param.Ie;
me = eval(me); % evaluate the time-changing parameters
re = eval(re);
Ie = eval(Ie);
delE = eval(delE);

lam = param.lam;
eta1 = param.eta1;
eta2 = param.eta2;
mInacc = param.mInacc;


% define true parameters
a1 = I1+m1*r1^2+Ie+me*re^2+me*L1^2;
a2 = Ie+me*re^2;
a3 = me*L1*re*cos(delE);
a4 = me*L1*re*sin(delE);

```

```
% True coefficient matrices
H11 = a1+2*a3*cos(q2)+2*a4*sin(q2);
H22 = a2;
H12 = a2+a3*cos(q2)+a4*sin(q2);
H21 = H12;
h = a3*sin(q2)-a4*cos(q2);
G1 = m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1));
G2 = me*g*re*cos(q1+q2);
H_mat = [H11 H12; H21 H22];
c_mat = [-h*dq2 -h*(dq1+dq2); h*dq1 0];
g_mat = [G1;G2];

% Estimated coefficient matrices
a1hat = mInacc*a1;
a2hat = mInacc*a2;
a3hat = mInacc*a3;
a4hat = mInacc*a4;
H11hat = a1hat+2*a3hat*cos(q2)+2*a4hat*sin(q2);
H22hat = a2hat;
H12hat = a2hat+a3hat*cos(q2)+a4hat*sin(q2);
H21hat = H12hat;
hhat = a3hat*sin(q2)-a4hat*cos(q2);
G1_hat = 1.2^(2)*m1*r1*g*cos(q1) + 1.2*me*g*(1.2*re*cos(q1+q2) + ...
    1.2*L1*cos(q1));
G2_hat = 1.2^(2)*me*g*re*cos(q1+q2);
H_mathat = [H11hat H12hat; H21hat H22hat];
c_mathat = [-hhat*dq2 -hhat*(dq1+dq2); hhat*dq1 0];
g_mathat = [G1_hat;G2_hat];
% Coefficient errors
Htilde = H_mathat-H_mat;
ctilde = c_mathat-c_mat;
gtilde = g_mathat-g_mat;

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
qtilde = [q1-q1_d; q2-q2_d];   % positions
qtilde_dot = [dq1-dq1_d; dq2-dq2_d]; % derivatives

% Calculate s (sliding surface)
s = qtilde_dot +lam*qtilde;
s1 = s(1);  s2 = s(2);

% define reference velocities
dq_r = [dq1_d;dq2_d]-lam*qtilde;
dq1_r = dq_r(1); dq2_r = dq_r(2);
ddq_r = [ddq1_d;ddq2_d]-lam*qtilde_dot;
ddq1_r = ddq_r(1); ddq2_r = ddq_r(2);
```

```
195
196 % Calculate control law (tau = tau_hat-k*sign(s))
197 tau_hat = H_mathat*[ddq1_r; ddq2_r]+c_mathat*[dq1_r;dq2_r]+g_mathat;
198 k = (abs(Htilde*[ddq1_r; ddq2_r]+ctilde*[dq1_r;dq2_r])+[eta1;eta2])';
199 k_xd=abs(Htilde*[ddq1_d; ddq2_d]+ctilde*[dq1_d;dq2_d])+[eta1;eta2];
200 % k(xd)
201 phidot = -lam.*[phi1;phi2]+ k_xd; % time varying boundary layer
202 kbar= k - phidot;
203 tau = tau_hat-kbar*[sat(s1,phi1);sat(s2,phi2)];
204
205 % Calculate True Response
206 ddQ=inv(H_mat)*(tau-c_mat*[dq1; dq2]-g_mat);
207 ddq1 = ddQ(1);
208 ddq2 = ddQ(2);
209
210 xdot = [dq1; ddq1; dq2; ddq2; phidot];
211 end
212
213 %% want to get control as output
214 function [q1_d ,q2_d,dq1_d,dq2_d,qtilde,qtilde_dot,tau,s,a_rlz,ahat] ...
       =slideModeBVControl(t,X,param)
215 % unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
216 q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
217 phi1 = X(5); phi2 = X(6);
218
219 % define parameters
220 g = param.g;
221 m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
222 L1 = param.L1; delE = param.delE;
223 r1 = param.r1; re = param.re;
224 I1 = param.I1; Ie = param.Ie;
225 me = eval(me); % evaluate the time-changing parameters
226 re = eval(re);
227 Ie = eval(Ie);
228 delE = eval(delE);
229
230 lam = param.lam;
231 eta1 = param.eta1;
232 eta2 = param.eta2;
233 mInacc = param.mInacc;
234
235 % define true parameters
236 a1 = I1+m1*r1^2+Ie+me.*re.^2+me.*L1^2;
237 a2 = Ie+me.*re.^2;
238 a3 = me.*L1.*re.*cos(delE);
239 a4 = me.*L1.*re.*sin(delE);
240 % define coefficient matrices
```

```
H11 = a1+2*a3.*cos(q2)+2*a4.*sin(q2);
H22 = a2;
H12 = a2+a3.*cos(q2)+a4.*sin(q2);
H21 = H12;
h = a3.*sin(q2)-a4.*cos(q2);
G1 = m1*r1*g.*cos(q1) + me.*g.*(re.*cos(q1+q2) + L1.*cos(q1));
G2 = me.*g.*re.*cos(q1+q2);
for i=1:length(H11)
H_mat(:,:,i) = [H11(i) H12(i); H21(i) H22(i)];
c_mat(:,:,i) = [-h(i)*dq2(i) -h(i)*(dq1(i)+dq2(i)); h(i)*dq1(i) 0];
g_mat(:,:,i) = [G1(i);G2(i)];
end

% Estimated coefficient matrices
a1hat = mInacc*a1;
a2hat = mInacc*a2;
a3hat = mInacc*a3;
a4hat = mInacc*a4;
H11hat = a1hat+2*a3hat.*cos(q2)+2*a4hat.*sin(q2);
H22hat = a2hat;
H12hat = a2hat+a3hat.*cos(q2)+a4hat.*sin(q2);
H21hat = H12;
hhat = a3hat.*sin(q2)-a4hat.*cos(q2);
G1_hat = mInacc*(m1*r1*g.*cos(q1) + me.*g.*(re.*cos(q1+q2) + L1*cos(q1)));
G2_hat = mInacc*(me.*g.*re.*cos(q1+q2));
for i=1:length(H11hat)
H_mathat(:,:,i) = [H11hat(i)  H12hat(i); H21hat(i)  H22hat(i)];
c_mathat(:,:,i) = [-hhat(i)*dq2(i) -hhat(i)*(dq1(i)+dq2(i));...
hhat(i)*dq1(i) 0];
g_mathat(:,:,i) = [G1_hat(i);G2_hat(i)];
end
% Coefficient errors
Htilde = H_mathat-H_mat;
ctilde = c_mathat-c_mat;
gtilde = g_mathat-g_mat;

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
qtilde = [q1-q1_d, q2-q2_d]';
qtilde_dot = [dq1-dq1_d, dq2-dq2_d]';

% Calculate s (sliding surface)
for i=1:length(qtilde)
s(:,:,i)= qtilde_dot(:,i)+lam.*qtilde(:,i);
end
s1 = s(1,1,:); s2 = s(2,1,:);

```

```
% Define dynamics matrix (Y)
dq_r = [dq1_d,dq2_d]'-lam*qtilde; % reference velocity
dq1_r = dq_r(1,:)'; dq2_r = dq_r(2,:)';
ddq_r = [ddq1_d,ddq2_d]'-lam*qtilde_dot;
ddq1_r = ddq_r(1,:)'; ddq2_r = ddq_r(2,:)';


% Calculate control law (tau = tau_hat-k*sign(s))
for i = 1:length(H_mathat)
tau_hat(:,:,i) = H_mathat(:,:,i)*[ddq1_r(i); ...
    ddq2_r(i)]+c_mathat(:,:,i)*[dq1_r(i);dq2_r(i)]+g_mathat(:,:,i);
k(:,:,i) = (abs(Htilde(:,:,i)*[ddq1_r(i); ...
    ddq2_r(i)]+ctilde(:,:,i)*[dq1_r(i);dq2_r(i)])+[eta1;eta2])';
k_xd(:,:,i)=abs(Htilde(:,:,i)*[ddq1_d(i); ...
    ddq2_d(i)]+ctilde(:,:,i)*[dq1_d(i);dq2_d(i)])+[eta1;eta2];   % k(xd)
phidot(:,:,i) = -lam.*[phi1;phi2]+ k_xd(:,:,i); % time varying boundary layer
kbar(:,:,i)= k(:,:,i) - phidot(:,:,i);
tau(:,:,i) = tau_hat(:,:,i)-kbar(:,:,i)*[sat(s1(:,:,i),phi1);...
sat(s2(:,:,i),phi2)];
end

% For output to compare
a_rlz=[a1, a2, a3, a4];  % real
ahat=[a1hat, a2hat, a3hat, a4hat];% with inaccuracy

end

function y=sat(s,phi) % function from TimeVaringBLCode.m (Capt Hess)
     % sat is the saturation function with unit limits and unit slope.
     if abs(s)>phi
         % elseif x<-Δ
         y=sign(s);
     else y=s./phi;
     end
 end
```

Adaptive Controller

```
%% Adaptive Control
% Oct 2018
% Control law assumes exact knowledge of dynamics structure
% IDK how to account for gravity - unknown parameters are not linear wrt
% dynamics when gravity is included
clc; clear all; %close all

%% INPUTS
% Set mass properties
```

```
10 twoDof_A_massProperties % run the file
11
12 % Set simulation time span
13 t0 = 0; tf = 3; tspan = [t0 tf];
14
15 % Set input method (see twoDof_DesiredTrajectories.m)
16 param.inputMethod =3; % 2 is step, 3 is sinusoid
17
18 % Define initial conditions for states (starts at q1_d)
19 if param.inputMethod ==3
20     q10 = 0.9405;  dq10 = 0;   q20 = -1.7660;   dq20 = 0; % for sinusoid
21 elseif param.inputMethod ==2
22     q10 = 0; dq10 = 0; q20 = 0; dq20 = 0; % for step
23 end
24 X0 = [q10; dq10; q20; dq20; 0;0;0;0]; % last 4 are for ahat
25
26 % Set properties specific to the control law
27 K_d = 100*eye(2); param.K_d = K_d;
28 lam = 20; param.lam = lam;
29 Gamma = diag([13.5 0.2 0.6 0.01]); param.Gamma = Gamma;
30
31 options = [];
32 [t,X] = ode45(@AdaptiveController, tspan, X0, options, param);
33 q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
34
35 [q1_d ,q2_d,dq1_d,dq2_d,qtilde, qtilde_dot, tau,s,a_rlz,ahat] = ...
       AdaptiveControl(t,X,param);
36 q1tilde = qtilde(1,:); q2tilde = qtilde(2,:); s=squeeze(s);
37 a1=a_rlz(:,1); a2=a_rlz(:,2); a3=a_rlz(:,3); a4=a_rlz(:,4);
38 a1hat=ahat(:,1); a2hat=ahat(:,2); a3hat=ahat(:,3); a4hat=ahat(:,4);
39
40 %% Plots
41 figure (1) % STATES
42 subplot(411)
43 plot(t,rad2deg(q1),t,rad2deg(q1_d),'--');legend('True','desired')
44 ylabel('$q_1 (deg)$'); grid minor
45 %title(['MRAC: $\lambda$=',num2str(lam),'$\ K_D$=',num2str(K_d(1))])
46 subplot(412)
47 plot(t,rad2deg(q2),t,rad2deg(q2_d),'--')%;legend('True','desired')
48 ylabel('$q_2 (deg)$'); grid minor
49 subplot(413)
50 plot(t,rad2deg(dq1),t,rad2deg(dq1_d),'--')%;legend('True','desired')
51 ylabel('$\dot{q}_1$ (deg/s)'); grid minor
52 subplot(414)
53 plot(t,rad2deg(dq2),t,rad2deg(dq2_d),'--')%;legend('True','desired')
54 xlabel('time (s)');
55 ylabel('$\dot{q}_2$ (deg/s)'); grid minor
```

```
%saveas(gcf,'ADAPTstates.png')

figure (2) % ERROR
subplot(2,2,1)
plot(t,rad2deg(qtilde(1,:)),t,rad2deg(qtilde(2,:)))
legend('$\tilde{q}_1$','$\tilde{q}_2$')
ylabel('$q_i$ Error (deg)');xlabel('time (s)');grid minor
%title(['MRAC: $\lambda$=',num2str(lam),'$\ K_D$=',num2str(K_d(1))])
subplot(2,2,2)
plot(t,rad2deg(qtilde_dot(1,:)),t,rad2deg(qtilde_dot(2,:)))
legend('$\dot{\tilde{q}}_1$','$\dot{\tilde{q}}_2$')
ylabel('$\dot{q}_i$ Error (deg/s)');xlabel('time (s)');grid minor
subplot(2,2,3)
plot(t,squeeze(tau(1,:,:)),t,squeeze(tau(2,:,:)))
legend('$\tau_1$','$\tau_2$')
xlabel('time (s)'); ylabel('$\tau_i (N-m)$');grid minor
subplot(2,2,4)
plot(rad2deg(q1tilde),rad2deg(dq1-dq1_d),...
rad2deg(q2tilde),rad2deg(dq2-dq2_d));
%title(['\lambda=', num2str(lam),', \eta_1=', num2str(eta1)])
xlabel('$\tilde{q}_i (deg)$');
ylabel('$\dot{\tilde{q}}_i (deg/s)$')
legend('Joint 1','Joint 2');grid minor
%saveas(gcf,'ADAPTerror.png')

figure (3)  % SLIDING SURFACE
subplot(2,1,[1 2])
plot(t,s);
xlabel('t(sec)'); ylabel('$s(t)$');
legend('$$s_1$$','$$s_2$$');grid minor
title('Sliding Surface')
%subplot(2,1,2)
%plot(t,phi1,t,phi2,':')
%xlabel('t(sec)'); ylabel('$\phi$');
legend('$$\phi_1$$','$$\phi_2$$');grid minor
%title('Time-Varying Boundary Layer')
%saveas(gcf,'ADAPTsurface.png')

figure (4)  % PARAMETERS
subplot(221)
plot(t,a1,t,a1hat,':')
legend('$$a_1 \ true$$','$$\hat{a}_1$$')
xlabel('time (s)'); grid minor;
%title('Parameter Estimation')
subplot(222)
plot(t,a2,t,a2hat,':')
legend('$$a_2 \ true$$','$$\hat{a}_2$$')
```

```
xlabel('time (s)'); grid minor
subplot(223)
plot(t,a3,t,a3hat,':')
legend('$$a_3 \ true$$','$$\hat{a}_3$$')
xlabel('time (s)'); grid minor
subplot(224)
plot(t,a4,t,a4hat,':')
legend('$$a_4 \ true$$','$$\hat{a}_4$$')
xlabel('time (s)'); grid minor
%saveas(gcf,'ADAPTparam.png')

%% animate
% z = [q1'; dq1'; q2'; dq2']; % actual states
% A.plotFunc = @(t,z)( drawRobot(t,z,param) );
% A.speed = 0.25;
% A.figNum = 101;
% %animate(t,z,A)

%%
function xdot= AdaptiveController(t,X,param)
% unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
q1 = X(1); dq1 = X(2); q2 = X(3); dq2 = X(4);
a1hat = X(5); a2hat = X(6); a3hat = X(7); a4hat = X(8);
ahat=[a1hat, a2hat, a3hat, a4hat];% estimated

% define parameters
g = param.g;
m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
L1 = param.L1; delE = param.delE;
r1 = param.r1; re = param.re;
I1 = param.I1; Ie = param.Ie;
me = eval(me); % evaluate the time-changing parameters
re = eval(re);
Ie = eval(Ie);
delE = eval(delE);
%
K_d = param.K_d; lam = param.lam;
Gamma = param.Gamma;

% define true parameters/coefficients
a1 = I1+m1*r1^2+Ie+me*re^2+me*L1^2;
a2 = Ie+me*re^2;
a3 = me*L1*re*cos(delE);
a4 = me*L1*re*sin(delE);
H11 = a1+2*a3*cos(q2)+2*a4*sin(q2);
H22 = a2;
H12 = a2+a3*cos(q2)+a4*sin(q2);
```

```
150 H21 = H12;
151 h = a3*sin(q2)-a4*cos(q2);
152 % G1 = m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1));
153 % G2 = me*g*re*cos(q1+q2);
154 H_mat = [H11 H12; H21 H22];
155 c_mat = [-h*dq2 -h*(dq1+dq2); h*dq1 0];
156 % g_mat = [G1; G2];
157
158 % Estimated coefficient matrices
159 H11hat = a1hat+2*a3hat*cos(q2)+2*a4hat*sin(q2);
160 H22hat = a2hat;
161 H12hat = a2hat+a3hat*cos(q2)+a4hat*sin(q2);
162 H21hat = H12;
163 hhat = a3hat*sin(q2)-a4hat*cos(q2);
164 % G1_hat = (m1*r1*g*cos(q1) + me*g*(re*cos(q1+q2) + L1*cos(q1)));
165 % G2_hat = (me*g*re*cos(q1+q2));
166 H_mathat = [H11hat H12hat; H21hat H22hat];
167 c_mathat = [-hhat*dq2 -hhat*(dq1+dq2); hhat*dq1 0];
168 % g_mathat = [G1;G2];
169
170 % Desired trajectories
171 [q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
172 qtilde = [q1-q1_d; q2-q2_d];   % positions
173 qtilde_dot = [dq1-dq1_d; dq2-dq2_d]; % derivatives
174
175 % Calculate s (sliding surface)
176 s = qtilde_dot +lam*qtilde;
177
178 % Define reference velocity
179 dq_r = [dq1_d;dq2_d]-lam*qtilde;
180 dq1_r = dq_r(1); dq2_r = dq_r(2);
181 ddq_r = [ddq1_d;ddq2_d]-lam*qtilde_dot;
182 ddq1_r = ddq_r(1); ddq2_r = ddq_r(2);
183
184 % Define dynamics matrix (Y) such that H*qdd_r+C*qd_r+g=Y*a
185 Y11 = ddq1_r;
186 Y12 = ddq2_r;
187 Y21 = 0;
188 Y22 = ddq1_r +ddq2_r;
189 Y13 = (2*ddq1_r +ddq2_r)*cos(q2)-(dq2*dq1_r+dq1*dq2_r+dq2*dq2_r)*sin(q2);
190 Y14 = (2*ddq1_r +ddq2_r)*sin(q2)+(dq2*dq1_r+dq1*dq2_r+dq2*dq2_r)*cos(q2);
191 Y23 = ddq1_r*cos(q2)+dq1*dq1_r*sin(q2);
192 Y24 = ddq1_r*sin(q2)-dq1*dq1_r*cos(q2);
193 Y=[Y11 Y12 Y13 Y14; Y21 Y22 Y23 Y24]; % linear wrt parameters a,ahat
194
195 % Calculate control law (same as Y*ahat-Kd*s)
196 tau = H_mathat*[ddq1_r; ddq2_r]+c_mathat*[dq1_r;dq2_r]-100*eye(2)*s; ...
```

```
        %+g_mathat
197
198 % Adapative Law
199 ahatdot = -Gamma*transpose(Y)*s;
200
201 % Calculate True Response
202 ddQ=inv(H_mat)*(tau-c_mat*[dq1; dq2]); %-g_mat
203 ddq1 = ddQ(1);
204 ddq2 = ddQ(2);
205
206 xdot = [dq1; ddq1; dq2; ddq2; ahatdot];
207 end
208
209 %% Other outputs
210 function [q1_d ,q2_d,dq1_d,dq2_d,qtilde, qtilde_dot, tau,s,a_rlz,ahat] = ...
        AdaptiveControl(t,X,param)
211 % unpack X = [q1;dq1;q2;dq2]=[q1; dq1; q2; dq2]
212 q1 = X(:,1); dq1 = X(:,2); q2 = X(:,3); dq2 = X(:,4);
213 a1hat = X(:,5); a2hat = X(:,6); a3hat = X(:,7); a4hat = X(:,8);
214
215 % define parameters
216 g = param.g;
217 m1 = param.m1; me = param.me; % e means link 2 + whatever it's holding
218 L1 = param.L1; delE = param.delE;
219 r1 = param.r1; re = param.re;
220 I1 = param.I1; Ie = param.Ie;
221 me = eval(me); % evaluate the time-changing parameters
222 re = eval(re);
223 Ie = eval(Ie);
224 delE = eval(delE);
225
226 K_d = param.K_d; lam = param.lam;
227 Gamma = param.Gamma;
228
229 % define true parameters
230 a1 = I1+m1*r1^2+Ie+me.*re.^2+me.*L1^2;
231 a2 = Ie+me.*re.^2;
232 a3 = me.*L1.*re.*cos(delE);
233 a4 = me.*L1.*re.*sin(delE);
234 % define coefficient matrices
235 H11 = a1+2*a3.*cos(q2)+2*a4.*sin(q2);
236 H22 = a2;
237 H12 = a2+a3.*cos(q2)+a4.*sin(q2);
238 H21 = H12;
239 h = a3.*sin(q2)-a4.*cos(q2);
240 % G1 = m1*r1*g*c1 + m2*g*(r2*c12 + L1*c1); % neglect gravity for now
241 % G2 = m2*g*r2*c12;
```

```
for i=1:length(H11)
H_mat(:,:,i) = [H11(i) H12(i); H21(i) H22(i)];
c_mat(:,:,i) = [-h(i)*dq2(i) -h(i)*(dq1(i)+dq2(i)); h(i)*dq1(i) 0];
end

% For output to compare
a_rlz=[a1, a2, a3, a4];  % real
ahat=[a1hat, a2hat, a3hat, a4hat];% estimated

% Estimated coefficient matrices
H11hat = a1hat+2*a3hat.*cos(q2)+2*a4hat.*sin(q2);
H22hat = a2hat;
H12hat = a2hat+a3hat.*cos(q2)+a4hat.*sin(q2);
H21hat = H12;
hhat = a3hat.*sin(q2)-a4hat.*cos(q2);
% G1 = m1*r1*g*c1 + m2*g*(r2*c12 + L1*c1); % neglect gravity for now
% G2 = m2*g*r2*c12;
for i=1:length(H11hat)
H_mathat(:,:,i) = [H11hat(i)  H12hat(i); H21hat(i)  H22hat(i)];
c_mathat(:,:,i) = [-hhat(i)*dq2(i) -hhat(i)*(dq1(i)+dq2(i)); ...
    hhat(i)*dq1(i) 0];
end

% Desired trajectories
[q1_d,dq1_d,ddq1_d,q2_d,dq2_d,ddq2_d]=twoDof_DesiredTrajectories(t,param);
qtilde = [q1-q1_d, q2-q2_d]';   % positions
qtilde_dot = [dq1-dq1_d, dq2-dq2_d]'; % derivatives

% Calculate s (sliding surface)
for i=1:length(qtilde)
s(:,:,i)= qtilde_dot(:,i)+lam.*qtilde(:,i);
end

% Define dynamics matrix (Y)
dq_r = [dq1_d,dq2_d]'-lam*qtilde; % reference velocity
dq1_r = dq_r(1,:)'; dq2_r = dq_r(2,:)';
ddq_r = [ddq1_d,ddq2_d]'-lam*qtilde_dot;
ddq1_r = ddq_r(1,:)'; ddq2_r = ddq_r(2,:)';

Y11 = ddq1_r;
Y12 = ddq2_r;
Y21 = zeros(size(ddq1_r));
Y22 = ddq1_r +ddq2_r;
Y13 = (2.*ddq1_r +ddq2_r).*cos(q2)-...
(dq2.*dq1_r+dq1.*dq2_r+dq2.*dq2_r).*sin(q2);
Y14 = (2.*ddq1_r +ddq2_r).*sin(q2)+...
(dq2.*dq1_r+dq1.*dq2_r+dq2.*dq2_r).*cos(q2);
```

```
288 Y23 = ddq1_r.*cos(q2)+dq1.*dq1_r.*sin(q2);
289 Y24 = ddq1_r.*sin(q2)-dq1.*dq1_r.*cos(q2);
290
291 for i=1:length(ddq2_r)
292 Y(:,:,i)=[Y11(i) Y12(i) Y13(i) Y14(i); Y21(i) Y22(i) Y23(i) Y24(i)]; % ...
        linear wrt parameters a,ahat
293 end
294
295
296 % Calculate control law (same as Y*ahat-Kd*s)
297 for i = 1:length(H_mathat)
298 tau(:,:,i) = H_mathat(:,:,i)*[ddq1_r(i); ...
        ddq2_r(i)]+c_mathat(:,:,i)*[dq1_r(i);dq2_r(i)]-100.*s(:,:,i);%+g_mathat
299 end
300
301
302 end
```

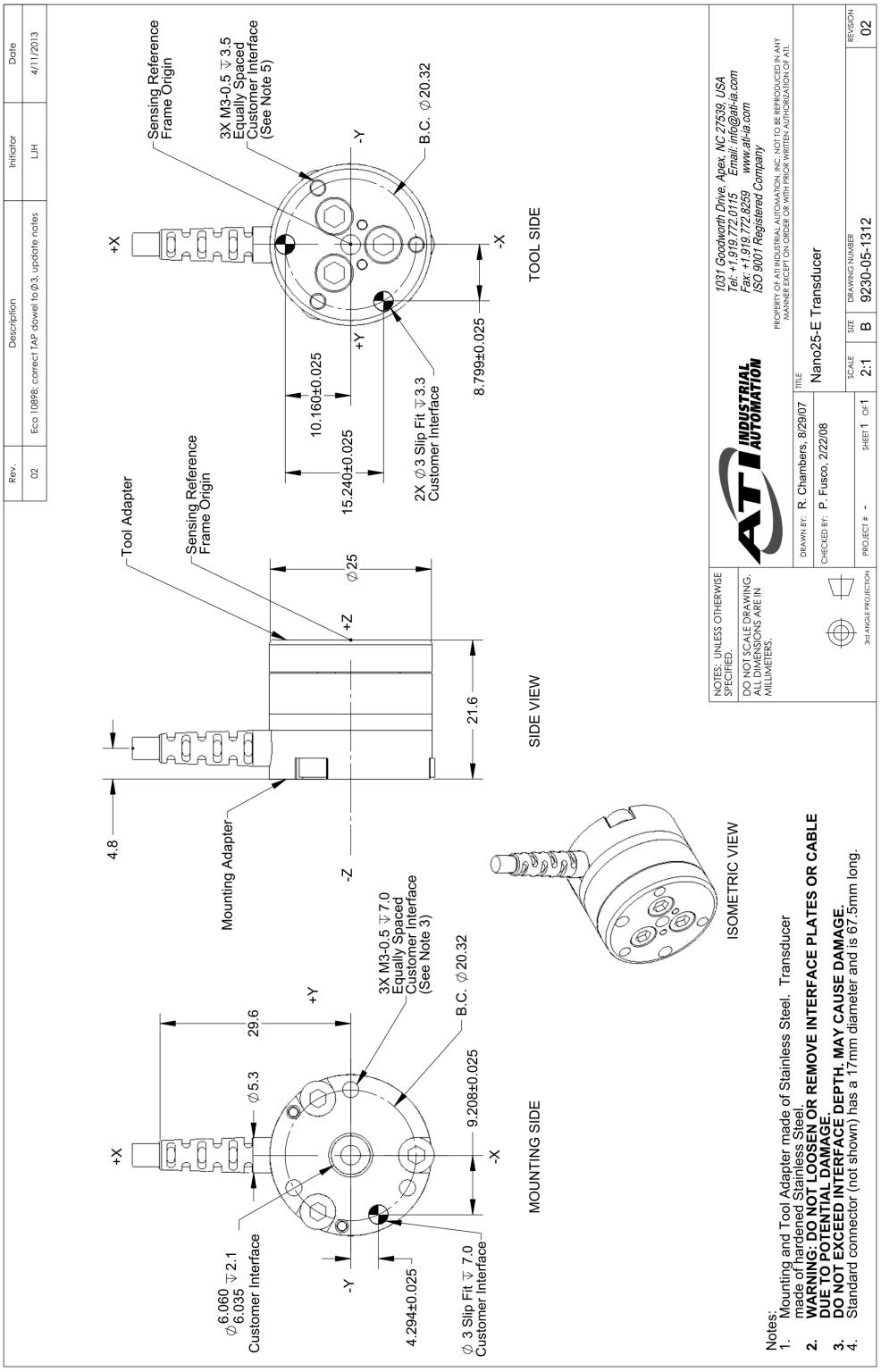
Graph comparison of trajectories

```
1  %% Plots of Desired trajectory vs Simulated Traj (using FK)
2  % 2d Lt Kyra Schmidt, Feb '19
3  % assumes that one of control laws was run, needs this info already saved:
4  % t, q1, q2, param.inputMethod ==3
5  % A) Plot Reachable Space
6  % B) plot desired trajectory
7  % C) Plot actual/simulated trajectory
8  % D) Plot links for simulation
9
10 %% A) Plot Reachable Space based on Angular Limits of Links
11 % Code mostly from mathworks, path needs to be eval wrt t (t=='double')
12 % ****************************** INPUTS **********************************
13 theta1_range = -deg2rad(-90.3:2:184.5); % all possible theta1 values
14 theta2_range = -deg2rad(-180.4:2:-0.6); % all possible theta2 values
15 L1 = 1.354;    % length of link 1, m(elbow to wrist)
16 L2 = 0.808;    % length of link 2, m (wrist + sting)
17 % ************************************************************************
18 [THETA1,THETA2] = meshgrid(theta1_range,theta2_range); % make grid of all ...
       theta values
19 [~,~,X,Y]=fwdkin2(THETA1,THETA2,param); % forward kinematics
20
21 figure (100)
22 plot(X(:),Y(:),':','color',[1 0.4 0.6],'DisplayName','Reachable Space');
23 xlabel('$y_i$ [m]');ylabel('$z_i$ [m]');xlim([0 3]);ylim([-1.5 1.5])
24 legend('show','location','ne');grid on; hold on
25
26 %% B) Plot Desired Trajectory
```

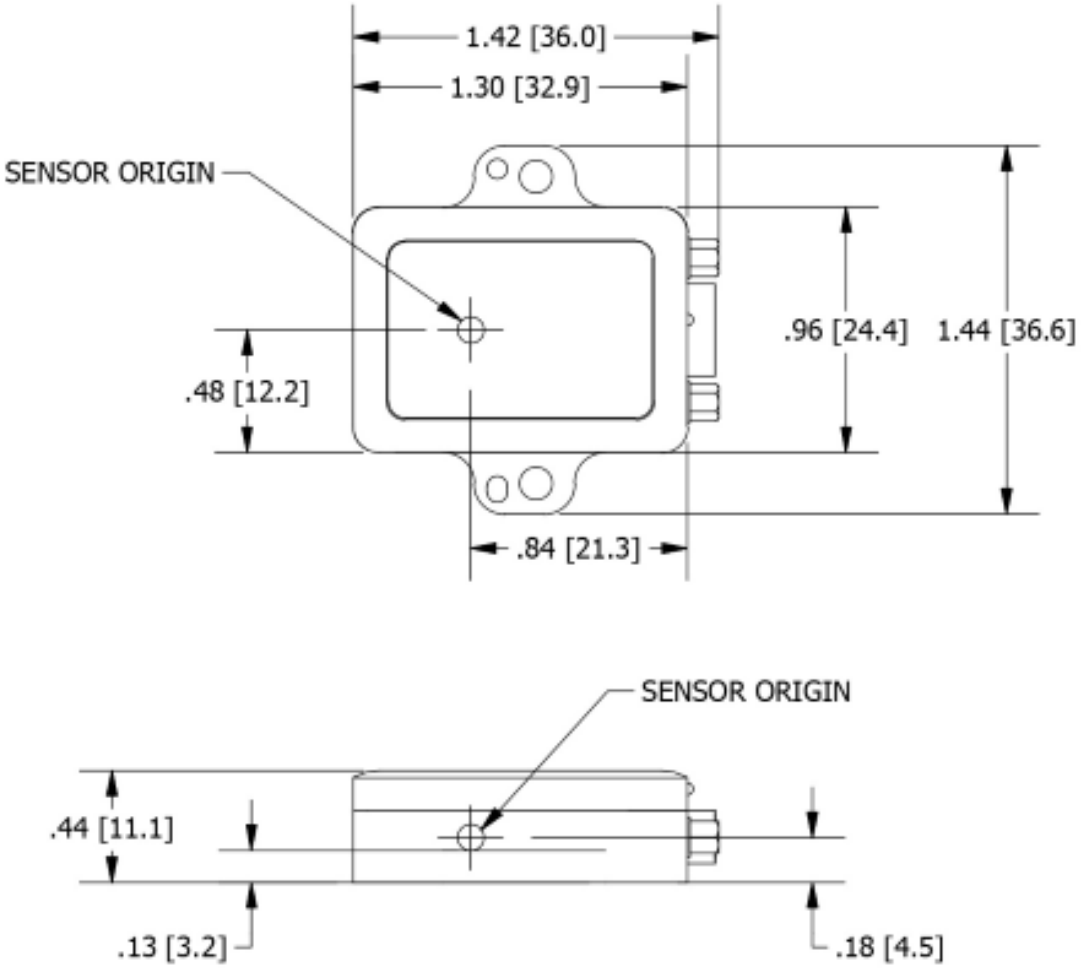
```
    y_d = 1.346+0.5*sin(2*pi*t);
    z_d = -0.3+0.8*cos(2*pi*t);
plot(y_d,z_d,'DisplayName','Desired Trajectory', 'LineWidth',0.5)
addarrows(t,y_d,z_d,2,170,2.5) % 2 arrows w/ length scaling x2, width 0.3

%% C) Plot Simulated Trajectory
[x_2,y_2,x_e,y_e]=fwdkin2(q1,q2,param);
hold on
plot(x_e,y_e,'DisplayName','Simulated Trajectory', 'LineWidth',0.5)

%% D) optional: if i want to plot the links too
p1=[x_2';y_2'];    % position of joint 2 (x;y)
p2=[x_e';y_e'];    % position of end (x;y)
pos=zeros(3,2,length(x_2)); % initialize matrix
for i=1:length(x_2)
    pos(:,:,i)=[0,0;     % origin/joint 1 (x,y)
    p1(1,i),p1(2,i);     % joint 2 (x,y)
    p2(1,i),p2(2,i)];    % end point (x,y)
end

numFrames=3; % length of pos must be divisible by numFrames
plot(0,0,'ks','MarkerSize',12,'LineWidth',3,'DisplayName','Origin')
% plot origin
for i=linspace(1,length(pos),numFrames)
plot(pos(:,1,i),pos(:,2,i),'LineWidth',2,'DisplayName',...
['t=',num2str(t(i),3),' s']) % plot the links
end
```

Appendix B. Drawings of Test Fixtures Models

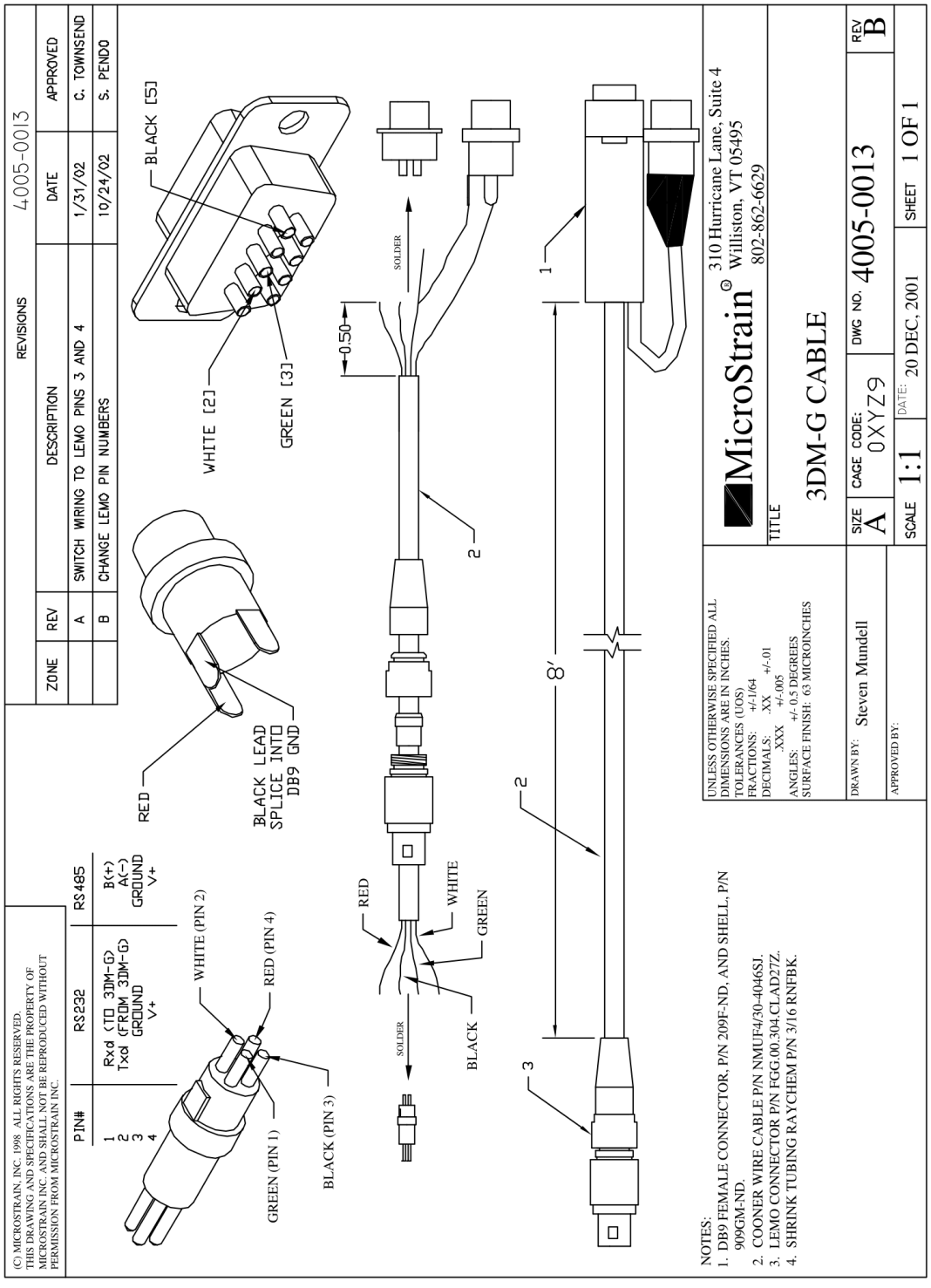
ATI Nano25-E F/T Transducer Drawing



IMU Drawing [2]



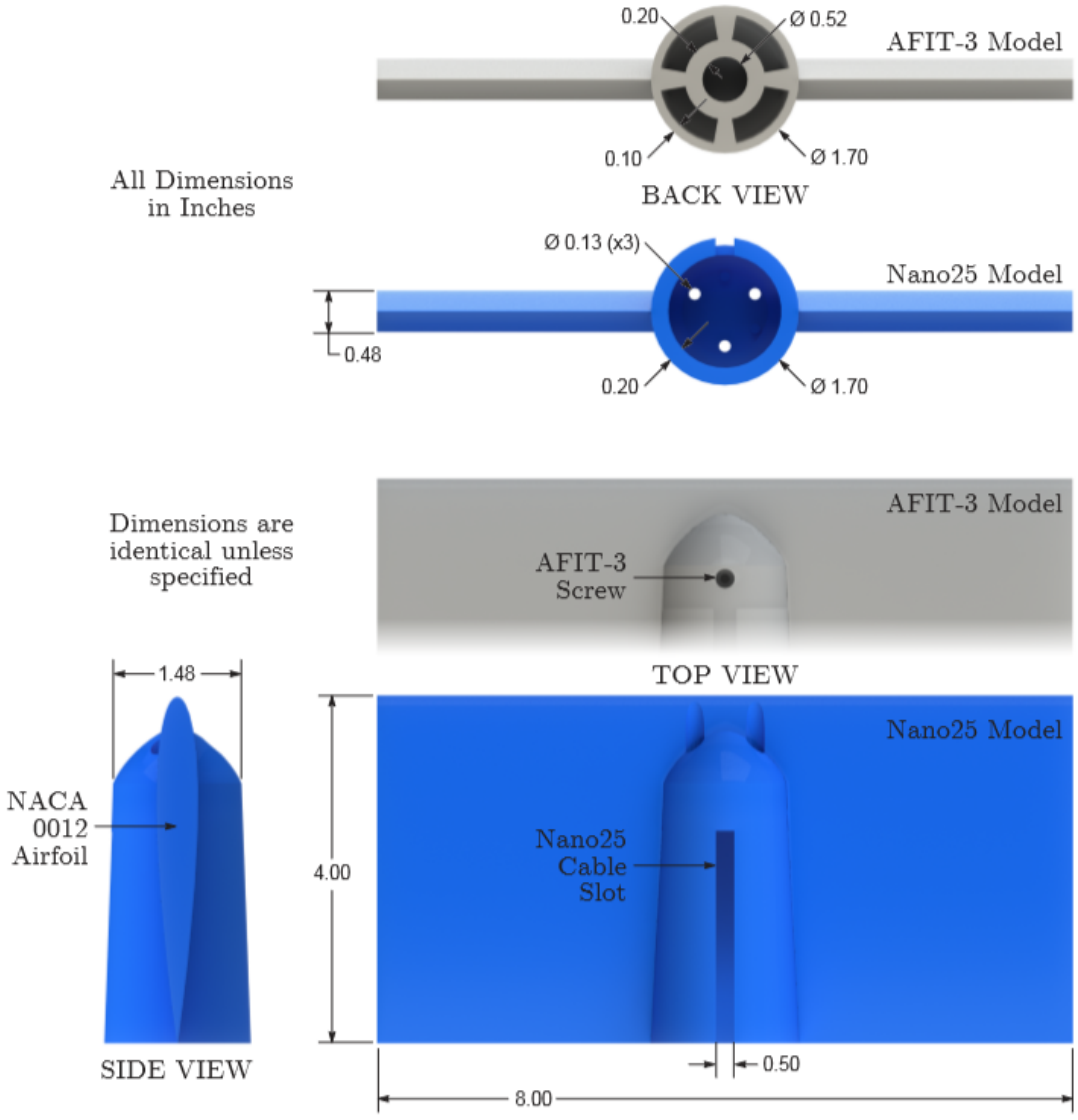
3DM-G Cable Schematic



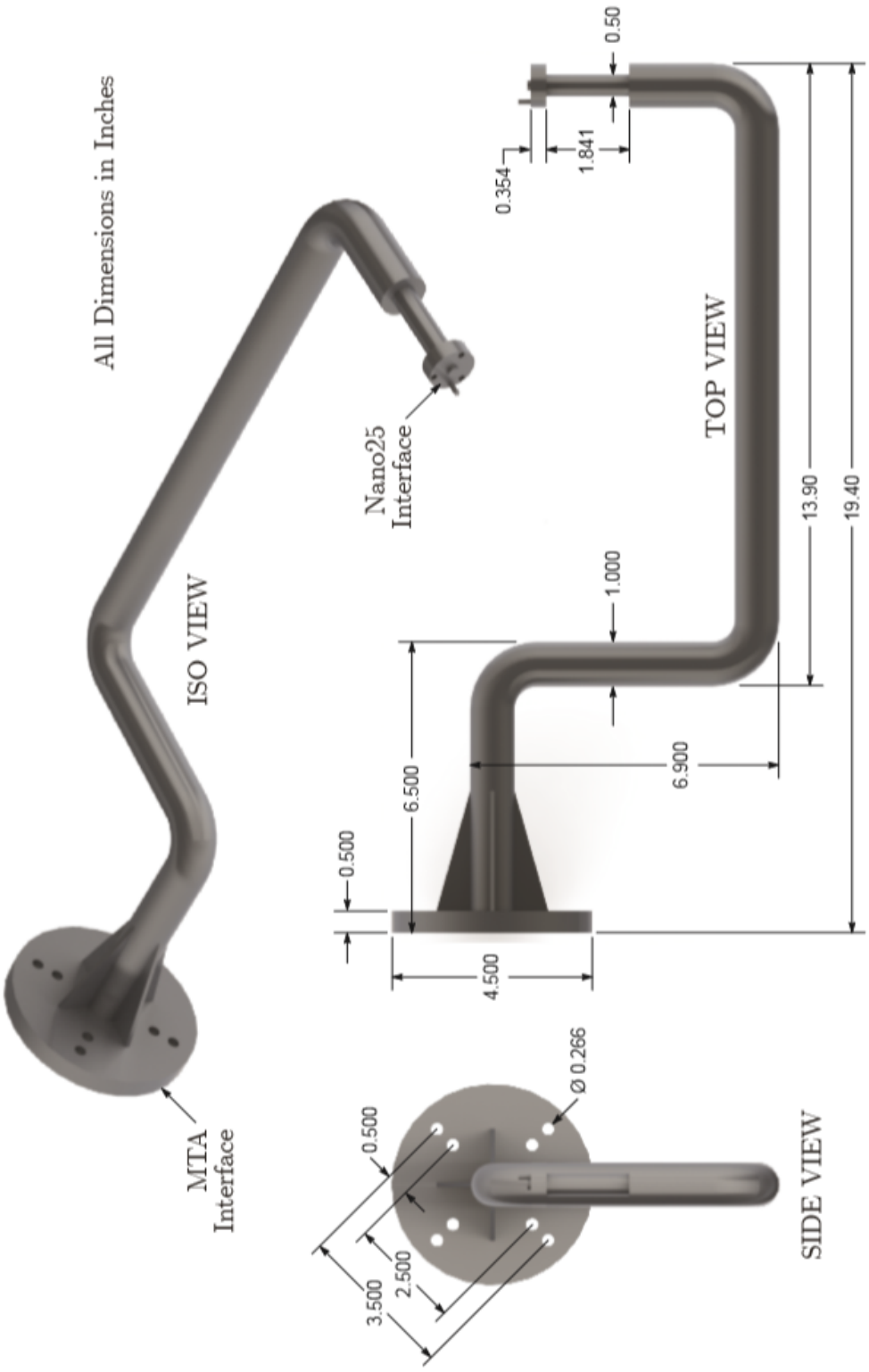
Wing Model Drawing [41]

NACA 0012 Wing Model

for AFIT-3 Balance and Nano25 F/T Transducer



MTA Model Support Sting [41]



Bibliography

1. Industrial Robot Control System, 2011.

2. 3DM-GX4-15 Inertial Measurement and Vertical Reference Unit (IMU/VRU), 2015.

3. F/T Sensor: Nano25, 2018.

4. Introducing Onvios Dojen Zero Backlash, Cycloidal Gearbox / Speed Reducer, 2018.

5. Carmelo Allegro. MTA CAD Drawings, 2012.

6. Rabeb Ben Amor and Salwa Elloumi. Decentralized Robust Model Reference Adaptive Control for Interconnected Time-delay Systems. (3):4285–4289, 2004.

7. Gianluca Antonelli, Stefano Chiaverini, and Giuseppe Fusco. A new on-line algorithm for inverse kinematics of robot manipulators ensuring path tracking capability under joint limits. *IEEE Transactions on Robotics and Automation*, 19(1):162–167, 2003.

8. H. Harry Asada. Chapter 7: Dynamics. *Introduction to Robotics*, pages 1–16, 2005.

9. Andrew D. Bower. *Investigation of Dynamic Store Separation Out of a Weapons Bay Cavity Utilizing a Low Speed Wind Tunnel*. Master's thesis, Air Force Institute of Technology, 2017.

10. Himanshu Chaudhary, Rajendra Prasad, and N. Sukavanum. Trajectory tracking control of scorbot-er v plus robot manipulator based on kinematical approach. *International Journal of Engineering Science and Technology*, 4(03):1174–1182, 2012.

11. Vincius Menezes De Oliveira and Walter Fetter Lages. Linear predictive control of a brachiation robot. In *Canadian Conference on Electrical and Computer Engineering*, pages 1518–1521, Ottawa, 2006. IEEE.

12. C. Canudas De Wit, B. Brogliato, P. Noel, A. Aubin, and P. Drevet. Compensation in Robot Manipulators : Low Velocities. *International Journal of Robotics Research*, pages 189–199, 1989.

13. Jacques Denavit and Richard S. Hartenberg. A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices, 1955.

14. Utku Eren, Anna Prach, Baaran Bahadr Koçer, Saa V. Raković, Erdal Kayacan, and Behet Açkmeşe. Model Predictive Control in Aerospace Systems: Current State and Opportunities. *Journal of Guidance, Control, and Dynamics*, 40(7):1541–1566, 2017.

15. Charles J. Fallaha, Maarouf Saad, Hadi Youssef Kanaan, and Kamal Al-Haddad. Sliding-mode robot control with exponential reaching law. *IEEE Transactions on Industrial Electronics*, 58(2):600–610, 2011.

16. Roy Featherstone and David Orin. Robot dynamics: Equations and algorithms. *Proceedings-IEEE International Conference on Robotics and Automation*, 1:826–834, 2000.

17. Nancy Hall. Wind Tunnel Testing, 2015.

18. Sikandar Hayat and Zareena Kausar. Mobile robot path planning for circular shaped obstacles using simulated annealing. *Proceedings - 2015 International Conference on Control, Automation and Robotics, ICCAR 2015*, pages 69–73, 2015.

19. Renato V. B. Henriques and Jos Jaime da Cruz. Robust Position Control of Mechanical Manipulators. *IFAC Proceedings Volumes*, 33(27):93–97, 2000.

20. Dmitry I. Ignatyev, Maria E. Sidoryuk, Konstantin A. Kolinko, and Alexander N. Khrabrov. Wind Tunnel Three-Degree-of-Freedom Dynamic Rig for Control Validation. *30th Congress of the International Council of the Aeronautical Sciences*, pages 1–11, 2016.

21. Jerry E. Jenkins. *The Care, Feeding, and Measurement of Dynamic Stability Derivatives*. 2007.

22. G. Josin, D. Charney, and D. White. Robot Control Using Neural Networks. In *International Conference on Neural Networks*, pages 625–631, San Diego, 1988. IEEE.

23. Norihiko Kato, Kenji Matsuda, and Tatsuya Nakamura. Adaptive control for a throwing motion of a 2 DOF robot. *Proceedings of 4th IEEE International Workshop on Advanced Motion Control - AMC '96 - MIE*, 1:203–207, 1996.

24. Rafael Kelly and Ricardo Salgado. PD Control with Computed Feedforward of Robot Manipulators: A Design Procedure. *IEEE Transactions on Robotics and Automation*, 10(4):566–571, 1994.

25. T. C. Kuo, B. W. Hong, Y. J. Huang, and C. Y. Chen. Adaptive Fuzzy Controller Design for Robotic Manipulators with Sliding Mode Control. In *IEEE International Conference On Fuzzy Systems*, pages 581–586, Hong Kong, 2008.

26. James C. Lancaster. *Characterization of a Robotic Manipulator for Dynamic Wind Tunnel Applications*. Master's thesis, Air Force Institute of Technology, 2015.

27. Henry Mcdonald, James Ross, David Driver, and Stephen Smith. Wind Tunnels and Flight. page 19, 2000.

28. Richard M. Murray, Li Zexiang, and Shankar S. Sastry. *A Mathematical Introduction to Robotic Manipulation*, volume 4. CRC Press LLC, 1994.

29. Yoshihiko Nakamura and Hideo Hanafusa. Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control. *Journal of Dynamic Systems, Measurement, and Control*, 108(3):163, 1986.

30. Milad Nazarahari, Esmaeel Khanmirza, and Samira Doostie. Multi-objective multi-robot path planning in continuous environment using an enhanced genetic algorithm. *Expert Systems with Applications*, 115:106–120, 2019.

31. Robert C. Nelson. *Flight Stability and Automatic Control*. McGraw-Hill, Singapore, 2 edition, 1998.

32. Katsuhiko Ogata. *Modern Control Engineering*. Prentice-Hall, Upper Saddle River, NJ, 5th edition, 2010.

33. Stelian Emilian Olean and Alexandru Morar. Simulation of the Local Model Reference Adaptive Control of the Robotic Arm with D.C. Motor Drive. *Mediamira*, pages 114–118, 2010.

34. John Pattinson, Mark Lowenberg, and Mikhail Goman. A Multi-Degree-of-Freedom Rig for the Wind Tunnel Determination of Dynamic Data. *AIAA Atmospheric Flight Mechanics Conference*, (August), 2009.

35. Farzin Piltan, Nasri Sulaiman, Hagar Nasiri, Sadeq Allahdadi, and Mohammad A. Bairami. Novel Robot Manipulator Adaptive Artificial Control : Design a Novel SISO Adaptive Fuzzy Sliding Algorithm Inverse Dynamic Like Method. *International Journal of Engineering (IJE), Volume (5) : Issue (5) : 2011*, (5):399–418, 2011.

36. A. Chennakesava Reddy. Difference Between Denavit - Hartenberg (D-H) Classical and Modified Conventions for Forward Kinematics of Robots With Case Study. *International Conference on Advanced Materials and manufacturing Technologies*, (figure 1):267–286, 2014.

37. Don Riley. 3D Puma Robot Demo, 2007.

38. Patrick Rowe. Motion Test Apparatus (MTA) Manipulator User Manual. Technical report, RE2, Inc., 2014.

39. Viboon Sangveraphunsiri and Kummun Chooprasird. Dynamics and control of a 5-DOF manipulator based on an H-4 parallel mechanism. *International Journal of Advanced Manufacturing Technology*, 52(1-4):343–364, 2011.

40. Victor Santibáñez and Rafael Kelly. Strict Lyapunov functions for control of robot manipulators. *Automatica*, 33(4):675–682, 1997.

41. James B. A. Sellers. *Force and Moment Measurements Applicable to a Flexible Weapons System*. Master's thesis, Air Force Institute of Technology, 2016.

42. James B. A. Sellers, Andrew D. Bower, Ian Maatz, and Mark F. Reeder. Dynamic Measurement of Forces and Moments with the Motion Test Apparatus. In *55th AIAA Aerospace Sciences Meeting*, pages 1–10, Grapevine, 2017. AIAA.

43. Ali Dokht Shakibjoo and Mohammad Dokht Shakibjoo. 2-DOF PID with reset controller for 4-DOF robot arm manipulator. *2015 International Conference on Advanced Robotics and Intelligent Systems, ARIS 2015*, 2015.

44. Chao Shen, Yang Shi, and Brad Buckham. Trajectory Tracking Control of an Autonomous Underwater Vehicle Using Lyapunov-Based Model Predictive Control. *IEEE Transactions on Industrial Electronics*, 65(7):5796–5805, 2018.

45. Jean-Jacques E. Slotine and Weiping Li. *Applied Nonlinear Control*. Prentice-Hall, Upper Saddle River, 1991.

46. Mark W. Spong, Seth Hutchinson, and M. Vidyasagar. *Robot dynamics and control*, volume 2. 2004.

47. Tung-Ching Tsao and Michael G. Safonov. Unfalsified Direct Adaptive Control of a Two-Link Robot Arm. In *Proceedingsof the 1999 IEEE InternationalConferenceon ControlApplications*, volume 1, pages 680–686, Kohala Coast-Island of Hawaii, 1999. IEEE.

48. Spyros G. Tzafestas, G. Stavrakakis, and A. Zagorianos. Robot model reference adaptive control through lower/upper part dynamic decoupling. *Journal of Intelligent & Robotic Systems*, 1(2):163–184, 1988.

49. S. N. Van Den Brink. *Modelling and control of a robotic arm actuated by nonlinear artificial muscles*. PhD thesis, Technische Universiteit Eindhoven, 2007.

50. C. W. Wampler II. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man and Cybernetics*, 16(1):93–101, 1986.

51. Chia-Yu E. Wang, Wojciech K. Timoszyk, and James E. Bobrow. Weightlifting Motion Planning For A Puma 762 Robot. In *International Conference on Robotics & Automation*, number May, pages 480–485. IEEE, 1999.

52. D. E. Whitney. Resolved motion rate control of manipulators and human prostheses. *IEEE Transactions on Man-Machine Systems*, 10(2):47–52, 1969.

53. Robert L. Williams. NotesBook Supplement for EE/ME 4290/5290 Mechanics and Control of Robotic Manipulators, 2019.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
02–11–2019	Master's Thesis	July 2018 — Mar 2019

4. TITLE AND SUBTITLE

ANALYTICAL MODELS AND CONTROL DESIGN APPROACHES FOR A 6 DOF MOTION TEST APPARATUS

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Kyra L. Schmidt

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENY-MS-19-M-245

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Department of Aeronautical Engineering
2950 Hobson Way
WPAFB OH 45433-7765
DSN 785-6565x4559, COMM 937-255-3636
Email: richard.cobb@afit.edu

10. SPONSOR/MONITOR'S ACRONYM(S)

AFRL/RW

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A:
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Wind tunnels play an indispensable role in the process of aircraft design, providing a test bed to produce valuable, accurate data that can be extrapolated to actual flight conditions. Historically, time-averaged data has made up the bulk of wind tunnel research, but modern flight design necessitates the use of dynamic wind tunnel testing to provide time-accurate data for high frequency motion. This research explores the use of a 6 degree of freedom (DOF) motion test apparatus (MTA) in the form of a robotic arm to allow models inside a subsonic wind tunnel to track prescribed trajectories to obtain time-accurate force and moment coefficients. Specifically, different control laws were designed, simulated, and integrated into a 2 DOF model representative of the elbow pitch and wrist pitch joints of the MTA system to decrease positional tracking error for a desired end-effector trajectory. Stability of the closed-loop systems was proven via Lyapunov analysis for all of the control laws, and the control laws proved to decrease tracking error during the trajectory case studies. An adaptive sliding mode control scheme was chosen as most suitable to simulate on the 6 DOF model due to the small tracking error as compared to the other control schemes and the availability of parameters of the actual MTA system when subject to the time-varying aerodynamics of the wind tunnel.

15. SUBJECT TERMS

control, robotic manipulator, simulation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Richard Cobb, AFIT/ENY
U	U	U	U	187	19b. TELEPHONE NUMBER *(include area code)* (312) 785-3636, x4559; richard.cobb@afit.edu