The Pennsylvania State University

The Graduate School

Department of Mechanical and Nuclear Engineering

MODELING, DESIGN, AND EXPERIMENTAL VALIDATION OF A TAILBOOM VIBRATION ABSORBER USING FLUIDIC FLEXIBLE MATRIX COMPOSITE TUBES

A Dissertation in

Mechanical Engineering

by

Kentaro Miura

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2016

The dissertation of Kentaro Miura was reviewed and approved* by the following:

Christopher D. Rahn
Professor of Mechanical Engineering
Dissertation Co-Advisor, Co-Chair of Committee

Edward C. Smith
Professor of Aerospace Engineering
Dissertation Co-Advisor, Co-Chair of Committee

Mary I. Frecker
Associate Department Head for Graduate Studies

Charles E. Bakis
Distinguished Professor of Engineering Science and Mechanics

*Signatures are on file in the Graduate School

ABSTRACT

Fluidic Flexible Matrix Composite (F^2MC) tubes are a promising new class of high-authority, lightweight fluidic devices that can passively provide damping and vibration absorption. The aim of this research is to model, design, and experimentally demonstrate passive vibration control of rotorcraft tailbooms using F^2MC-based damped absorbers.

Rotorcraft tailbooms are subject to periodic excitation from the main rotor and broadband excitation from aerodynamic forces. These excitations cause excessive driveline component wear, structural fatigue, and passenger discomfort. Lightweight and compact passive tailboom vibration treatments are needed to replace the heavy and bulky vibration treatments widely used in production helicopters today. A lightweight vibration absorber can be produced by fluidically coupling two pairs of F^2MC tubes, mounted on the top and bottom of a tailboom, via an inertia track. Partially closing an orifice in the inertia track results in a damped absorber.

To demonstrate the performance of F^2MC-based damped absorbers, analytical models are developed for a lab-scale tailboom structure and a full-scale tailboom on a Bell Helicopter OH-58C. These models are used to design F^2MC-based damped absorbers for the lab-scale and full-scale tailbooms. Simulation results show that F^2MC tubes reduce vibration at the first vertical displacement bending mode of the lab-scale tailboom by nearly 78% for the open orifice configuration, and increase damping by nearly 8% at the first mode. For a weight penalty of under 4.5 kg (10 lb), F^2MC tubes are predicted to reduce vibration at the first vertical bending mode of the OH-58C tailboom by over 96%, as well as add over 8% damping to the first mode.

A prototype F^2MC-based damped absorber is installed onto the lab-scale tailboom structure and tested to validate the analytical model. Experimental frequency responses show good agreement with model predictions, with the experimental absorber reducing response amplitude at the first vertical bending mode by over 70%, and a partially closed orifice adding

nearly 8% damping to the first mode. The effect of fluid pre-pressure, orifice size, and tailboom forcing amplitude are studied.

Design rules for F^2MC-based damped absorbers are developed to enable sizing of the F^2MC tubes, as well as proper tuning of the inertia track and orifice. Guidelines are provided for selection of F^2MC tube mounting locations, bladder materials and thickness, and fluid. A fluidic circuit with two parallel inertia tracks is proposed and analyzed for applications requiring adjustable tuning for multiple operating conditions.

Performance-per-weight comparisons are made between F^2MC-based and conventional piston-based absorbers, as well as an active control solution. This research shows that, per added weight, the F^2MC tube outperforms conventional piston-type pumpers by almost three times. Furthermore, on a per-added weight basis, the F^2MC-absorber may outperform an active solution using piezoelectric actuators, depending on the power limits.

Finally, ongoing and future work for a prototype absorber on a full-scale OH-58C tailboom is discussed. In particular, static test results of a prototype absorber are presented to demonstrate authority.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my advisors, Dr. Christopher Rahn and Dr. Edward Smith, for their patience, invaluable guidance research-related and not, and for showing me how to calmly approach problems. I could not have asked for better advisors. My gratitude extends to my committee members, Dr. Mary Frecker, Dr. Charles Bakis, and Dr. Eric Mockensturm, whose insightful comments and questions have helped me in my research and dissertation writing.

My research would not have been possible without the financial support provided by Bell Helicopter, the Bell Graduate Fellowship, the College of Engineering Fellowship, and the U.S. Army Research Laboratory via the American Society of Engineering Education and the Oak Ridge Institute for Science and Education/Oak Ridge Associated Universities. The technical guidance I received from Peter Romano, Michael Seifert, and Dr. David Heverly from Bell Helicopter, as well as Dr. Hao Kang from the Army Research Laboratory, was critical to the success of this Ph.D. project. I am deeply indebted to Dr. Bin Zhu for his friendship and for being my F^2MC mentor, as well as to Matt Krott and Steven LaBarge for their significant contributions to the PSU tailboom experiment. I also could not have asked for better friends and labmates: Dan Aglione, Alexandre Bondoux, Chris Ferone, Mayank Garg, Nicolas Kurczewski, Jun Ma, Xiaokun Ma, Kiron Mateti, Chris Melville, Githin Prasad, Chinmay Rao, Mike Robinson, Tahzib Safwat, Lloyd Scarborough, Zheng Shen, Ying Shi, Tanvir Tanim, Shawn Treacy, and Andrew Wilson.

I thank Francesco Acciai for being an endless source of humor and a good sport, and Jeremy Horwitz for his willingness to listen and help me organize my thoughts. Last but not least, to my family: thank you for your unconditional support and encouragement.

Chapter 1

Introduction

Rotorcraft tailbooms are subject to periodic and broadband excitation. As illustrated in Figure 1-1, harmonic aerodynamic forcing can result from trailing vortices from the main rotor interacting with the tailboom and structures such as the horizontal stabilizer [1]. Broadband excitation is caused by maneuvers, wind gusts, and separated flow behind the rotor hub [2]. Due to the low inherent structural damping in the tailboom, these excitations cause high vibration and slowly decaying transients that decrease fatigue life and damage electronics. Vibration-induced structural fatigue increases maintenance costs and reduces vehicle availability [3]. Despite previous research on tailboom vibration reduction techniques [2, 4-6], heavy and bulky treatments remain the typical standard used in many production helicopters.

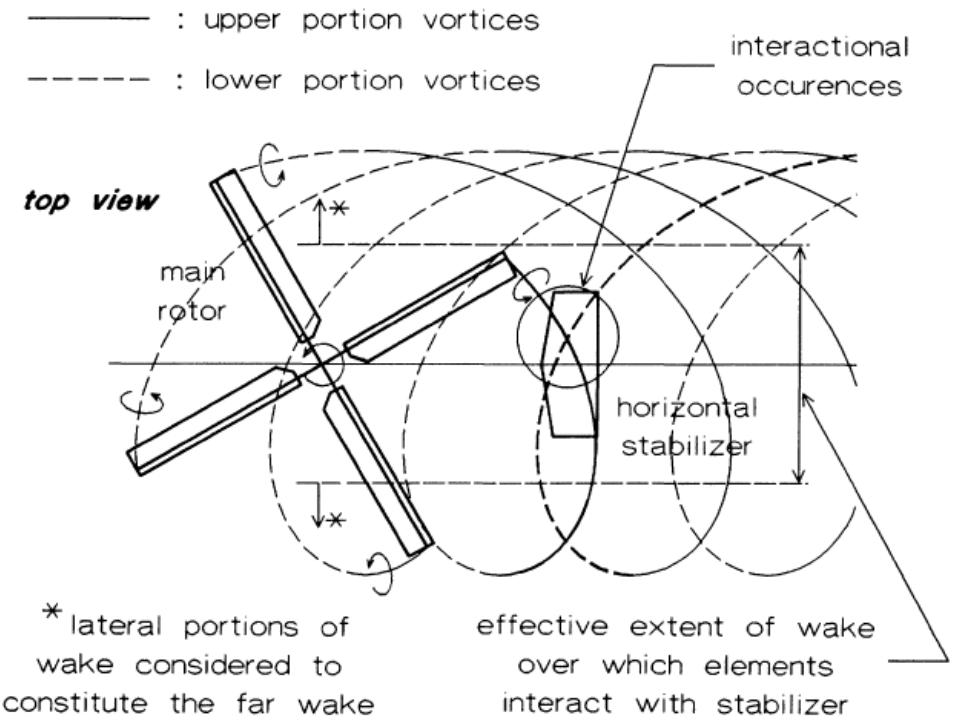

Figure 1-1. Diagram of main rotor wake-tailboom structure interaction [1].

1.1 Research Objectives

The aim of this research is to model, design, and demonstrate the performance of a lightweight passive fluidic vibration treatment for structures using Fluidic Flexible Matrix Composite (F^2MC) tubes. To that end, a mathematical model of a tailboom with F^2MC tubes is developed and implemented in MATLAB. This MATLAB model is used to propose F^2MC-based vibration absorber designs for two test bed rotorcraft structures: a simulated lab-scale tailboom and a full-scale OH-58C tailboom. Prototype absorbers are fabricated and installed onto the test bed structures, and experiments are conducted to demonstrate performance. To validate the model's predictive capability, numerous system parameters including fluid pre-pressure, orifice size, and tailboom forcing amplitude are varied and compared against theoretical predictions.

1.2 Vibration Control Techniques

The three main types of vibration control are isolation, damping, and absorption. Vibration isolators reduce load transmission to a system, and are typically placed between the system and the excitation source [7]. Comprising a load-supporting component and an energy-dissipating component [8], the most basic isolator uses a spring and dashpot placed in between the system and the ground. More sophisticated isolators like the Dynamic Antiresonant Vibration Isolator (DAVI) [9] use a tuned mass to provide an inertial force that counteracts vibration at a desired frequency. As in the Liquid Inertia Vibration Eliminator (LIVE®) device [10] and Fluidlastic technology developed by LORD Corporation [11], the inertia may also be provided by inertance resulting from fluid accelerated through an inertia track.

Dampers convert or dissipate energy to reduce vibration. The typical mechanism of mechanical damping is friction, for example between solids or fluids [12]. Damping can be

achieved through viscoelastic layers [13], piezoelectric shunt circuits [14], fluid dampers, eddy currents, hysteretic elements, and by dissipating energy through dynamic vibration absorbers [15], among other techniques.

Absorbers, which use the inertial effects of a mass added to the vibrating system to counteract vibratory excitation, have been developed by numerous researchers [16-19]. A tuned absorber produces an anti-resonance, flanked by two resonant peaks, in the structure's frequency response [20]. As illustrated in Figure. 1-2, the absorber adds an anti-resonance flanked by two resonance peaks to the system's frequency response. The heavier the added mass relative to the structural mass, the wider the resulting anti-resonance [2] and the broader the frequency range over which response amplitude is attenuated. Adding damping to the absorber results in a damped absorber or tuned mass damper, which reduces the amplitudes of the resonant peaks at the expense of increasing the anti-resonance amplitude [21, 22].

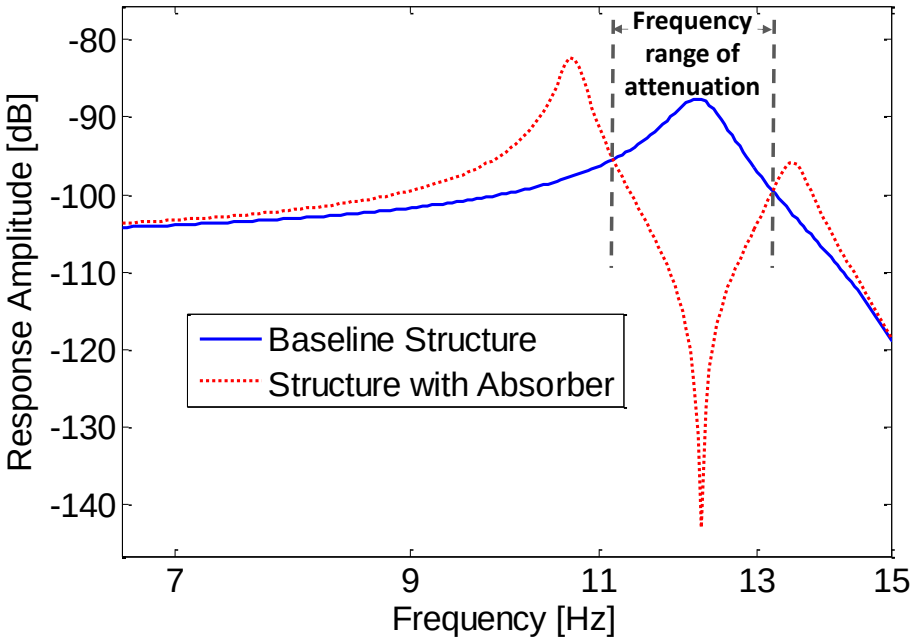

Figure 1-2. Frequency response of baseline structure with and without an added inertia.

Absorbers, which are the focus of this research, can be implemented passively, actively, or semi-actively [23]. Passive absorbers are tuned to reduce response over a specific range of operating conditions. Requiring no external power [24] or sensing, passive vibration absorbers are typically more easily retrofitted to existing structures compared to active or semi-active devices. Active absorbers use feedback from sensors to actuate the structure to cancel out the effects of vibratory excitation [6]. Active approaches outperform passive techniques but are more complex and costly, and therefore less reliable [25]. Semi-active control uses feedback to modify properties of the absorber itself to adapt to different operating conditions [26, 27]. While earlier passive, active, and semi-active vibration absorbers used mechanical masses and springs [16-19], more recently researchers have developed designs using other means, such as smart materials, magnetorheological fluids, and resonant circuit shunting [6, 28-33]. The inertance resulting from fluid motion may also be harnessed for vibration control [10, 11, 34, 35].

1.3 Vibration Treatments for Helicopter Tailbooms

Researchers have previously studied various approaches to reducing vibration in rotorcraft tailbooms. Heverly et al. demonstrated active control of a lab-scale tailboom structure by replacing a segment of the stringers with piezoelectric actuators [36]. Similar approaches using actuators and sensors distributed throughout the airframe have been proposed [37, 38]. Bansemir developed a passive tailboom vibration treatment using weights attached to elastically deformable walls within the tailboom [4]. Gaffey et al. proposed a nonlinear pendulum-like absorber with a mechanism to improve the frequency range of the anti-resonance [39]. Passive piezoelectric absorbers have also been put forth [37]. Vuillet and Zoppitelli designed an absorber that uses fluid sloshing inside a chamber in the tailboom to reduce vibration [40].

1.4 Fluidic Flexible Matrix Composite Tubes for Tailboom Vibration Control

Fluidic Flexible Matrix Composite (F^2MC) tubes are a new class of high-authority and low-weight fluidic devices that can passively provide vibration damping [41], absorption [42, 43], and isolation [44]. The F^2MC tube, shown in Figure 1-3, is constructed from a highly anisotropic Flexible Matrix Composite (FMC) laminate [45]. Two families of interlocking fibers in the laminate form winding angles $\pm\alpha$ with respect to the longitudinal axis. For winding angles less than 55°, tensile axial stress causes volume decrease, and compressive axial stress causes volume increase, as illustrated in Figure 1-3. When filled internally with fluid, F^2MC tubes under loading act as fluid pumps. The constraining effect of the fibers allows the F^2MC tube to pump up to two orders of magnitude more fluid per unit axial strain than can a piston of the same diameter [46]. The F^2MC tubes pump fluid into an external fluidic circuit consisting of an inertia track and an orifice. With a small-diameter inertia track and through the pumping amplification of the tube fibers, a small mass of vibrating fluid can provide the same effect as a large inertia. Flow restriction due to the orifice, as well as losses in the inertia track wall, contributes to damping.

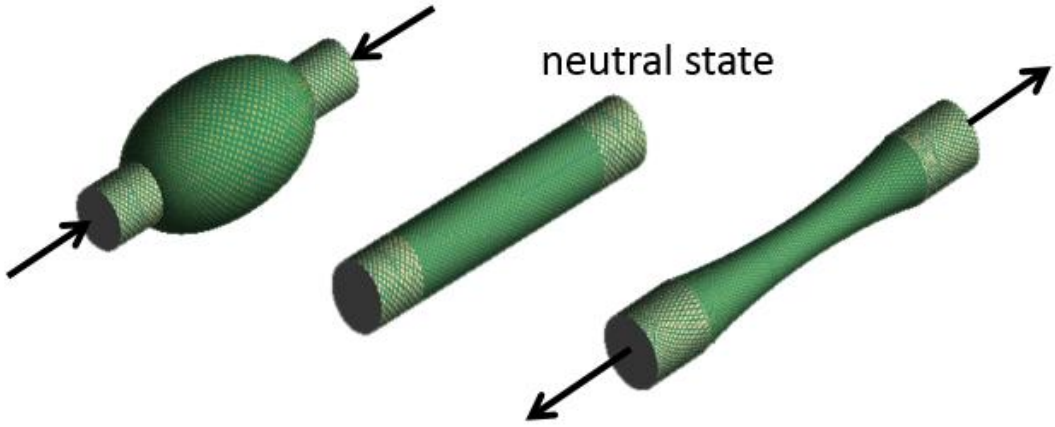

Figure 1-3. Volume change in a F^2MC tube ($\alpha < 55°$).

F^2MC tubes can be constructed by filament-winding carbon fibers in a soft matrix material such as polyurethane [45] (filament-wound tube) or by placing a rubber bladder inside a fiber-wound tube [44] (braid-sheathed tube). Filament-wound tubes are desirable as various tube properties such as fiber winding angle can be precisely controlled and custom-made. However, braid-sheathed F^2MC tubes are used in this work for their ease of construction and their analogous behavior to filament-wound tubes. Each braid-sheathed F^2MC tube is constructed of a stainless steel braided mesh and a rubber bladder. They are similar in construction to McKibben actuators except that, as illustrated in Figure 1-4, the mesh and bladder are independently fastened to end fittings that mechanically connect and fluidically seal the F^2MC tube (pictured in Figure 1-5), respectively. Unlike the typically pneumatically actuated McKibben muscles, these devices are used passively and use a liquid as the working fluid instead.

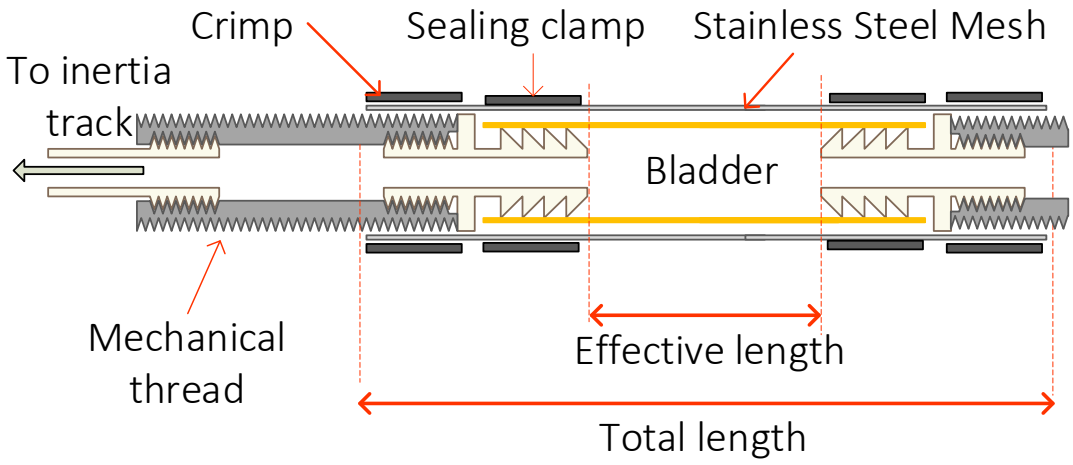

Figure 1-4. Schematic diagram of a braid-sheathed F^2MC tube.

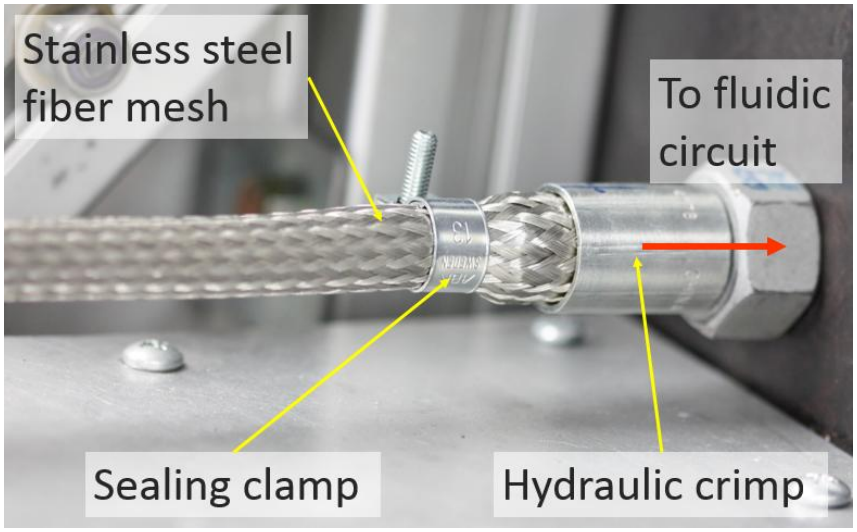

Figure 1-5. A braid-sheathed F^2MC tube attached to the lab-scale tailboom structure.

Nonlinear models of F^2MC tube behavior have been developed by several researchers [45, 47]. Zhu et al. studied F^2MC tubes embedded in a relatively rigid matrix and properties such as fluid pumping and actuation ability [46, 49]. Lotfi-Gaskarimahalle *et al.* showed that the fluid port connecting to an F^2MC tube can be designed to reduce vibration at a desired frequency [43]. Additional studies on characterizing F^2MC tubes and applying them to structures have been carried out [50, 51]. There have also been numerous studies specifically on passive vibration control using F^2MC tubes. Zhu et al. demonstrated analytically that F^2MC tubes attached to a cantilever beam can absorb beam vibration [42], and confirmed experimentally that a tunable orifice in the flow port can be used to provide damping [41].

1.5 Contributions

This research is the first instance in which a F^2MC-based absorber has been designed for a lab-scale aerospace structure. A new configuration of F^2MC tubes is explored in which F^2MC tubes on opposing sides of a bending structure are coupled fluidically. In contrast to the device

demonstrated by Zhu et al. [41, 42], the coupled F^2MC tubes do not require a separate fluid accumulator. Miura et al. demonstrate the feasibility of passively reducing vibration in helicopter tailbooms using such a configuration [52], with simulation results predicting over 10% damping added to the first bending mode. A model of braid-sheathed F^2MC tubes developed by Scarborough et al. [44] is modified, in the same fashion as Shan et al. for filament-wound tubes [48], to account for compliance in the inner bladder wall. The actuation and pumping ability of the F^2MC tubes are demonstrated through static tests on a lab-scale tailboom structure [53]. Static component-level tests further characterize the F^2MC tubes' pumping ability and compliance [54]. Dynamic tests on the lab-scale tailboom structure show that a F^2MC-based absorber weighing only 2.7 kg can reduce response amplitude by over 70% at the first bending mode, and that a partially closed orifice augments structural damping by nearly 8% [55]. Furthermore, the model predictions are shown to correlate well with experimental results. A novel absorber configuration using F^2MC tubes is presented, and a model of a tailboom treated with such an absorber is derived in Chapter 2. Chapter 3 introduces the lab-scale tailboom structure used to test the absorber performance. Experimental results validating the model developed in Chapter 2 is presented. Design considerations and process for F^2MC-based damped absorbers are detailed in Chapter 4. The design rules presented provide an excellent estimate of the optimal absorber parameters on simple SDOF structures. For more complex structures, the estimates are reasonably accurate and significantly reduce the optimal parameter search space. A tunable absorber using two inertia tracks in parallel is proposed and shown to allow absorption at one of two distinct frequencies, or a combination of partial absorption at both. The performance per weight of the F^2MC-absorber is shown to be three times that of a piston-based absorber, and, depending on power requirement, exceed a piezoelectric active controller's. Finally, work in progress for a test on the OH-58C tailboom is summarized in Chapter 5.

Chapter 2

Model Development for F^2MC-Based Passive Tailboom Vibration Control

The coupled F^2MC tube vibration absorber configuration is proposed. Analytical models for the two test bed structures, the lab-scale PSU tailboom and the OH-58C tailboom, are then developed. A modified model of braid-sheathed F^2MC tubes connected in the proposed absorber configuration is presented. Finally, the fluidic model is integrated into a generic structural model.

2.1 F^2MC-Absorber Configuration Design

A F^2MC-based vibration absorber for a beam-like structure like a helicopter tailboom can be implemented as the coupled F^2MC configuration shown in Figure 2-1. In this configuration, pre-tensioned tubes are attached on opposite sides of the tailboom's neutral axis and connected by an inertia track. The tubes and inertia track are filled with a working fluid. As the tailboom bends and vibrates, tubes on one side contract while the tubes on the opposite side elongate. This axial displacement results in volume change in the F^2MC tubes, accelerating the fluid back and forth between the tubes as the structure vibrates. Unlike previously tested designs, which relied on a separate fluid accumulator to provide a restoring force to the fluid, this configuration utilizes two pairs of "push-pull" F^2MC tubes.

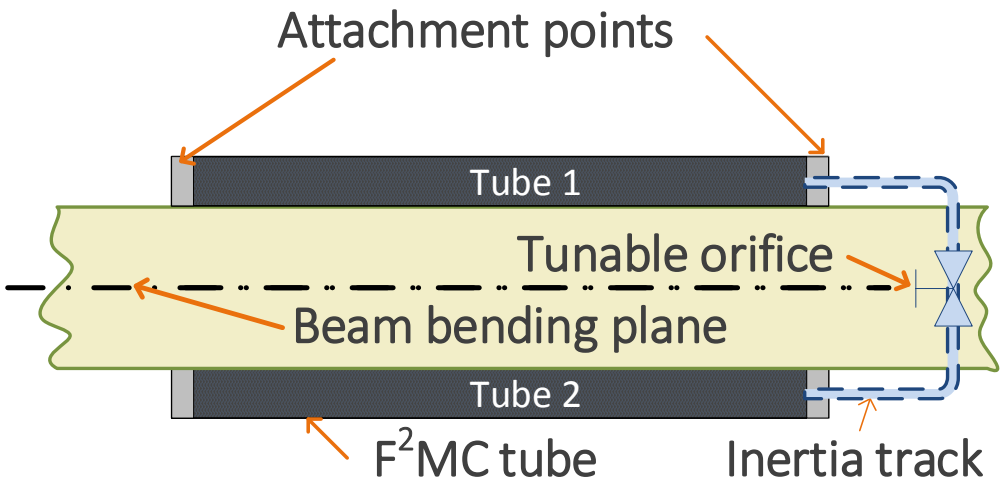

Figure 2-1. Coupled F^2MC tubes on opposing sides of a bending structure.

2.2 Tailboom Structure Model Development

A beam model of a helicopter tailboom is developed to determine the feasibility of F^2MC-based vibration treatments for rotorcraft structures. The tailboom structures are approximated as beams attached at the root to a wall by hinge-springs, as shown in the generic tailboom model diagram in Figure 2-2. The hinge and torsional spring at the root model compliance at the attachment, and the spring constant is determined using static stiffness measurements of the tailboom. Discrete masses are used to represent tailboom components and other airframe structures.

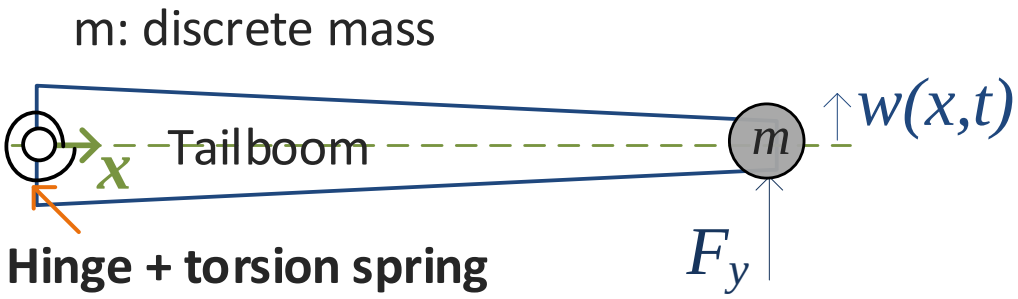

Figure 2-2. A generic hinge-springed beam model of a tailboom.

2.2.1 Lab-Scale Tailboom Structure Model

A lab-scale 1/3 scale model, pictured in Figure 2-3, of an Apache AH-64 is used as a test bed for the F^2MC-based absorbers. The tailboom is an aluminum, 72 in long, semi-monocoque structure cantilevered to a rigid frame. The tailboom cross-section is linearly tapered, and aluminum sheet metal skin is attached to eight stringers with a total cross-sectional area of 766 mm^2, as shown in Figure 2-4. The four corner stringers are aluminum angles that are 20.6 mm (0.8125 in) by 20.6 mm and 3.18 mm (0.125 in) thick. The side and top stringers are 28.6 mm (1.125 in) by 3.18 mm (0.125 in). Masses and hollow aluminum tubing attached to the tip of the tailboom simulate vertical and horizontal tail surfaces. Detailed drawings with dimensions can be found in Heverly's dissertation [6].

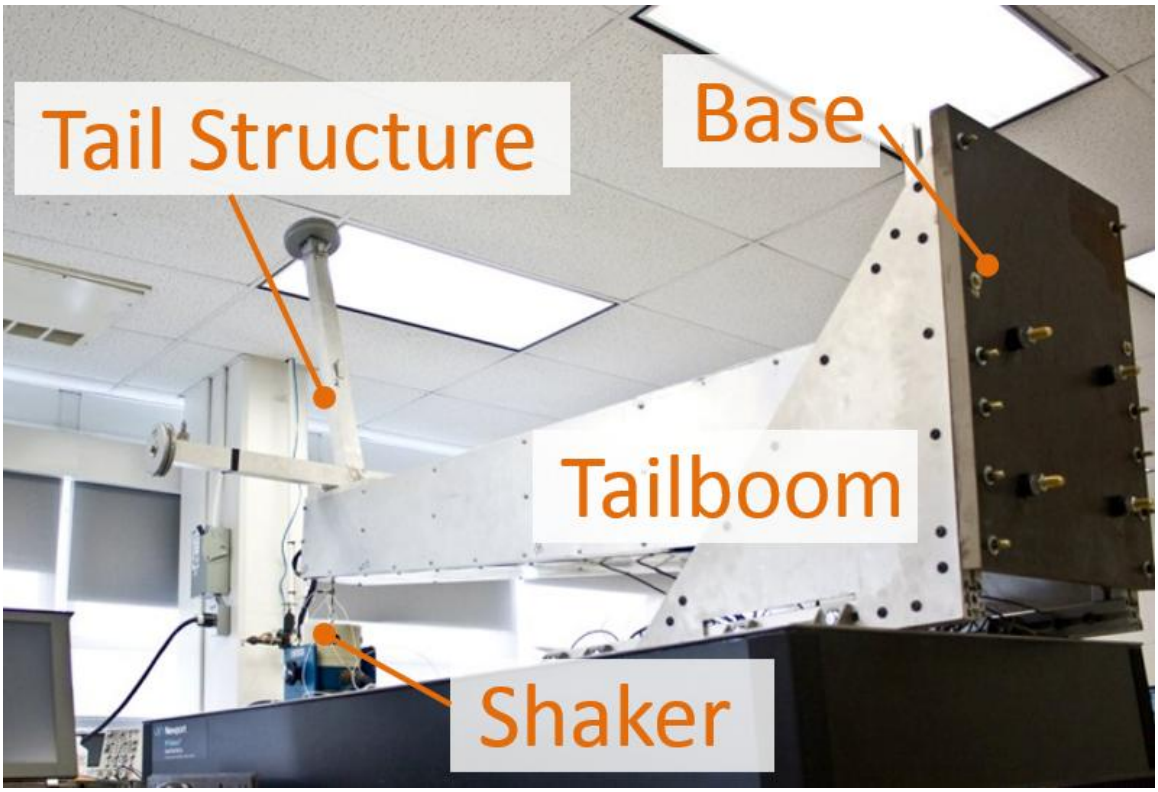

Figure 2-3. Lab-scale tailboom structure.

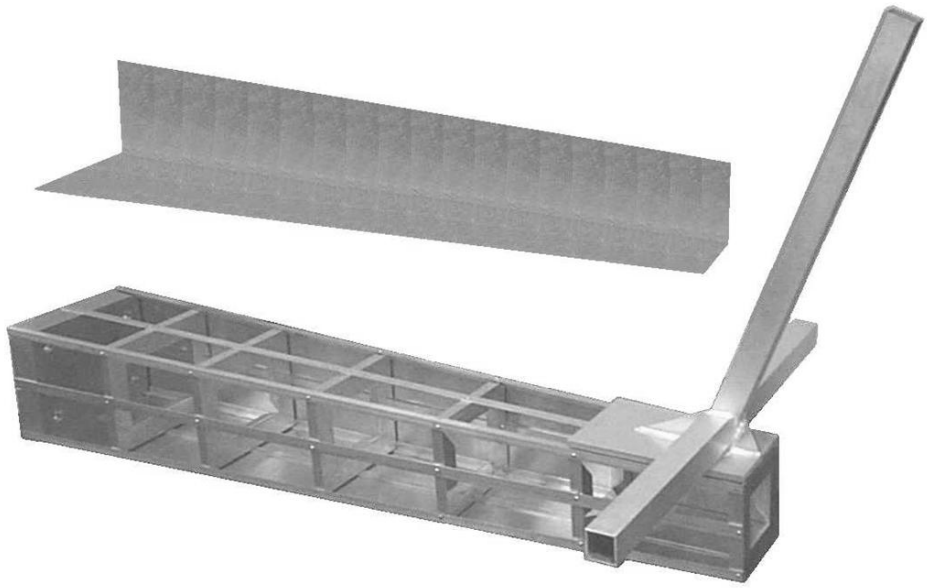

Figure 2-4. Internal structure of the lab-scale tailboom model [6].

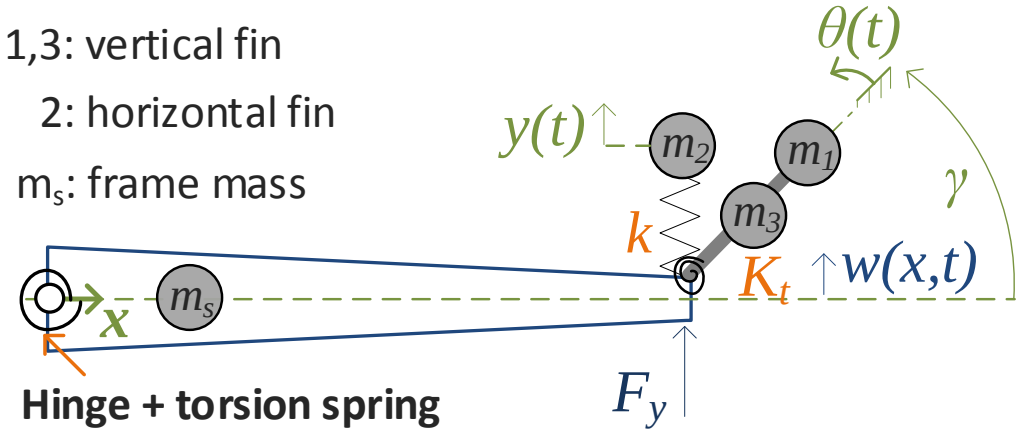

Figure 2-5. Diagram of the lab-scale tailboom model.

The lab-scale tailboom is modeled as a spring-hinged beam with discrete masses and springs to represent various tailboom components and structures, as shown in Figure 2-5. In response to vertical tip load F_y, the tailboom vibrates with vertical displacement $w(x,t)$, the

horizontal fin with vertical displacement $y(t)$, and the vertical fin with angle $\theta(t)$. m_1 is the vertical fin tip mass, and m_3 is the mass of the vertical hollow tube. m_2 is the effective mass of the horizontal fin. m_s represents the mass of the frame structure inside the tailboom. k is the effective linear spring constant of the horizontal fin, and K_t is the effective torsional spring constant for the vertical fin. l_v is the length of the vertical fin.

The Rayleigh-Ritz method is used to approximate the tailboom displacement

$$w(x,t) = \sum_{i=1}^{N} \psi_i(x) q_i(t) = \{\psi(x)\}^T \{q\}, \tag{2.1}$$

where $\psi_i(x)$ are the Ritz basis functions and $q_i(t)$ are the generalized coordinates. Basis functions for a spring-hinged beam, derived in Appendix A, are used, and the spring stiffness is determined based on static deflection measurements. Power P_{in} into the tailboom due to tip load F_y is given by

$$P_{in} = F_y \dot{w}(L), \tag{2.2}$$

where L is the length of the tailboom and $\dot{(\ )}$ denotes a derivative with respect to time. Defining the state vector

$$\{\eta\} = \begin{Bmatrix} \{q\} \\ y \\ \theta \end{Bmatrix}, \tag{2.3}$$

and substituting in Eq. (2.1), Eq. (2.2) becomes

$$P_{in} = \{\dot{\eta}\}^T \begin{bmatrix} \{\psi_L\} \\ 0 \\ 0 \end{bmatrix} F_y, \tag{2.4}$$

where $\psi_L = \psi(L)$.

Assuming no losses, power into the tailboom is equal to the time derivative of the total energy:

$$P_{in} = \frac{\partial}{\partial t}\left[\frac{1}{2}\int_0^L \dot{w}^2 \rho A dx + \frac{1}{2}m_s \dot{w}_s^2 + \frac{1}{2}m_1 v_1^2 + \frac{1}{2}m_2 \dot{y}^2 + \frac{1}{2}m_3 v_3^2 + \frac{1}{2}\int_0^L EI(w'')^2 dx + \frac{1}{2}k(y - w_L)^2 + \frac{1}{2}K_t(\theta - w_L')^2\right], \tag{2.5}$$

where ρ is the tailboom density, A is the tailboom cross-sectional area, E is the tailboom elastic modulus, I is the tailboom area moment, $w_L = w(L)$, $v_{1,2,and\ 3}$ are the velocities of $m_{1,2,and\ 3}$, respectively, and $(\quad)'$ indicate derivatives with respect to position x. The velocity vector of m_1 is

$$\overrightarrow{v_1} = \left(-l_v \dot{\theta} \sin(\theta + \gamma)\right)\hat{\imath} + \left(\dot{w}_L + l_v \dot{\theta} \cos(\theta + \gamma)\right)\hat{\jmath}, \tag{2.6}$$

where γ is the nominal angle formed between the vertical stabilizer and the x-axis. Similarly, the velocity vector of m_3 is

$$\overrightarrow{v_3} = \left(-\frac{l_v}{2}\dot{\theta} \sin(\theta + \gamma)\right)\hat{\imath} + \left(\dot{w}_L + \frac{l_v}{2}\dot{\theta} \cos(\theta + \gamma)\right)\hat{\jmath}. \tag{2.7}$$

The angular deflections of the tailboom tip and vertical stabilizer are relatively small. Since under the small-angle approximation $\gamma \gg \theta$ and $\gamma \gg w_L'$, $\cos(\theta + \gamma) \approx \cos\gamma$. Thus, Eqs. (2.6) and (2.7) are rewritten as:

$$v_1^2 = l_v^2 \dot{\theta}^2 + \dot{w_L}^2 + 2 l_v \cos\gamma\, \dot{w}_L \dot{\theta} \tag{2.8}$$

and

$$v_3^2 = \frac{l_v^2 \dot{\theta}^2}{4} + \dot{w_L}^2 + l_v \cos\gamma\, \dot{w}_L \dot{\theta}. \tag{2.9}$$

The time derivatives of the kinetic energies of masses m_1 and m_3 are therefore:

$$\frac{\partial}{\partial t}\left(\frac{1}{2}m_1 v_1^2\right) = m_1\left(l_v^2 \ddot{\theta}\dot{\theta} + \ddot{w}_L \dot{w}_L + l_v \cos\gamma\, \ddot{w}_L \dot{\theta} + l_v \cos\gamma \dot{w}_L\, \ddot{\theta}\right) \tag{2.10}$$

and

$$\frac{\partial}{\partial t}\left(\frac{1}{2}m_3 v_3^2\right) = m_3\left(\frac{l_v^2}{4}\ddot{\theta}\dot{\theta} + \ddot{w}_L\dot{w}_L + \frac{l_v}{2}\cos\gamma\,\ddot{w}_L\dot{\theta} + \frac{l_v}{2}\cos\gamma\dot{w}_L\,\ddot{\theta}\right). \tag{2.11}$$

Substituting Eqs. (2.10) and (2.11) into Eq. (2.5), we obtain:

$$\begin{aligned} P_{in} = \int_0^L \dot{w}\rho A\ddot{w}dx + m_s\dot{w}_s\ddot{w}_s + m_1\left(l_v^2\ddot{\theta}\dot{\theta} + \ddot{w}_L\dot{w}_L + l_v\cos\gamma\,\ddot{w}_L\dot{\theta} + l_v\cos\gamma\dot{w}_L\,\ddot{\theta}\right) \\ + m_2\dot{y}\ddot{y} + m_3\left(\frac{l_v^2}{4}\ddot{\theta}\dot{\theta} + \ddot{w}_L\dot{w}_L + \frac{l_v}{2}\cos\gamma\,\ddot{w}_L\dot{\theta} + \frac{l_v}{2}\cos\gamma\dot{w}_L\,\ddot{\theta}\right) \\ + \int_0^L EI\dot{w}"w"dx + k(\dot{y}y - \dot{y}w_L - \dot{w}_L y + \dot{w}_L w_L) \\ + K_t\left(\dot{\theta}\theta - \dot{\theta}w_L' - \dot{w}_L'\theta + \dot{w}_L'w'_L\right). \end{aligned} \tag{2.12}$$

Recalling Eq. (2.1), the vertical displacements w can be written in matrix form. The first term of Eq. (2.12) becomes

$$\int_0^L \dot{w}\rho A\ddot{w}dx = \{\dot{q}\}^T\left(\int_0^L \rho A\{\psi\}\{\psi\}^T dx\right)\{\ddot{q}\}, \tag{2.13}$$

and the second term becomes

$$m_s\dot{w}_s\ddot{w}_s = \{\dot{q}\}^T m_s\{\psi_s\}\{\psi_s\}^T\{\ddot{q}\}. \tag{2.14}$$

In a similar fashion, the other terms of Eq. (2.12) can be rewritten in matrix form and in terms of the state vector from Eq. (2.3) as follows:

$$P_{in} = \frac{1}{2}\frac{\partial}{\partial t}[\{\dot{\eta}\}^T(A_0 + m_s A_s + A_1 + m_2 A_2)\{\dot{\eta}\} + \{\eta\}^T(B_0 + K_t B_1 + kB_2)\{\eta\}], \tag{2.15}$$

where

$$A_0 = \int_0^L \rho A\begin{bmatrix} \{\psi\}\{\psi\}^T & \{0\} & \{0\} \\ \{0\}^T & 0 & 0 \\ \{0\}^T & 0 & 0 \end{bmatrix} dx, \tag{2.16}$$

$$A_s = \begin{bmatrix} \{\psi_s\}\{\psi_s\}^T & \{0\} & \{0\} \\ \{0\}^T & 0 & 0 \\ \{0\}^T & 0 & 0 \end{bmatrix}, \tag{2.17}$$

$$A_1 = \begin{bmatrix} (m_1 + m_3)\{\psi_L\}\{\psi_L\}^T & \{0\} & m_{13} l_v \cos\gamma \{\psi_L\} \\ \{0\}^T & 0 & 0 \\ m_{13} l_v \cos\gamma \{\psi_L\}^T & 0 & l_v^2 \left(m_1 + \frac{m_3}{4}\right) \end{bmatrix}, \tag{2.18}$$

$$A_2 = \begin{bmatrix} [0] & \{0\} & \{0\} \\ \{0\}^T & 1 & 0 \\ \{0\}^T & 0 & 0 \end{bmatrix}, \tag{2.19}$$

$$B_0 = \int_0^L EI \begin{bmatrix} \{\psi''\}\{\psi''\}^T & \{0\} & \{0\} \\ \{0\}^T & 0 & 0 \\ \{0\}^T & 0 & 0 \end{bmatrix} dx, \tag{2.20}$$

$$B_1 = \begin{bmatrix} \{\psi_L'\}\{\psi_L'\}^T & \{0\} & -\{\psi_L'\} \\ \{0\}^T & 0 & 0 \\ -\{\psi_L'\}^T & 0 & 1 \end{bmatrix}, \tag{2.21}$$

and

$$B_2 = \begin{bmatrix} \{\psi_L\}\{\psi_L\}^T & -\{\psi_L\} & \{0\} \\ -\{\psi_L\}^T & 1 & 0 \\ \{0\}^T & 0 & 0 \end{bmatrix}, \tag{2.22}$$

where $m_{13} = m_1 + m_3/2$. Evaluating the time derivative in Eq. (2.15) and setting the result equal to Eq. (2.4) yields:

$$\{\dot{\eta}\}^T \begin{bmatrix} \{\psi_L\} \\ 0 \\ 0 \end{bmatrix} F_y = \{\dot{\eta}\}^T (A_0 + m_s A_s + A_1 + m_2 A_2)\{\ddot{\eta}\} + \{\dot{\eta}\}^T (B_0 + K_t B_1 + k B_2)\{\eta\}. \tag{2.23}$$

Eliminating [56] the common term $\{\dot{\eta}\}^T$ from Eq. (2.15) gives

$$\begin{bmatrix} \{\psi_L\} \\ 0 \\ 0 \end{bmatrix} F_y = (A_0 + m_s A_s + A_1 + m_2 A_2)\{\ddot{\eta}\} + (B_0 + K_t B_1 + k B_2)\{\eta\}. \tag{2.24}$$

More compactly, Eq. (2.24) may be expressed as

$$[M_{eff}]\{\ddot{\eta}\} + [C_{eff}]\{\dot{\eta}\} + [K_{eff}]\{\eta\} = \{F_{eff}\}, \tag{2.25}$$

where $[M_{eff}]$, $[C_{eff}]$, and $[K_{eff}]$ are the mass, damping, and stiffness matrices, respectively, and $\{F_{eff}\}$ is the generalized forcing vector. The matrices in Eq. (2.25) are

$$\left[M_{eff}\right] = \begin{bmatrix} [M] & \{0\} & m_{13} l_v \cos\gamma \{\psi_L\} \\ \{0\}^T & m_2 & 0 \\ m_{13} l_v \cos\gamma \{\psi_L\}^T & 0 & \left(m_1 + \frac{m_3}{4}\right) l_v^2 \end{bmatrix}, \tag{2.26}$$

where

$$M_{jn} = \int_0^L \rho A \psi_j \psi_n dx + (m_1 + m_3)\psi_j(L)\psi_n(L) + m_s \psi_j(x_s)\psi_n(x_s) \tag{2.27}$$

and x_s is the longitudinal position of mass m_s,

$$\left[K_{eff}\right] = \begin{bmatrix} [K] & -k\{\psi_L\} & -K_t\{\psi_L'\} \\ -k\{\psi_L\}^T & k & 0 \\ -K_t\{\psi_L'\}^T & 0 & K_t \end{bmatrix}, \tag{2.28}$$

where

$$K_{jn} = \int_0^L EI\psi"_j \psi"_n dx + K_t \psi_j'(L)\psi_n'(L) + k\psi_j(L)\psi_n(L), \tag{2.29}$$

and

$$\left\{F_{eff}\right\} = \begin{bmatrix} \{\psi_L\} \\ 0 \\ 0 \end{bmatrix} F_y. \tag{2.30}$$

$$\left[C_{eff}\right] = [\Phi] \begin{bmatrix} 2\zeta\omega_1 & & \\ & \ddots & \\ & & 2\zeta\omega_N \end{bmatrix} [\Phi]^T, \tag{2.31}$$

where $[\Phi]$ is the modal transformation matrix, ω_n are the natural frequencies, and ζ is the modal damping ratio, which is obtained by applying the half-power method to the experimental frequency response plot of the tailboom structure. In state-variable form, Eq. (2.25) can be written as

$$\begin{aligned} \{\dot{z}_1\} &= \{z_2\} \\ \{\dot{z}_2\} &= -\left[M_{eff}\right]^{-1}\left[K_{eff}\right]\{z_1\} - \left[M_{eff}\right]^{-1}\left[C_{eff}\right]\{z_2\} + \left[M_{eff}\right]^{-1}\left\{F_{eff}\right\}, \end{aligned} \tag{2.31}$$

where

$$\{z_1\} = \begin{bmatrix} \{q\} \\ y \\ \theta \end{bmatrix}, \tag{2.32}$$

and

$$\{z_2\} = \begin{bmatrix} \{\dot{q}\} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}. \tag{2.33}$$

The parameter values of the lab-scale tailboom test stand are summarized in Table 2-1.

Table 2-1. Lab-Scale Tailboom Test Stand Parameters.

Characteristic	Imperial	Metric
Length	$72\ in$	$1.83\ m$
Base Width	$14.25\ in$	$0.362\ m$
Base Height	$11.26\ in$	$0.286\ m$
Tip Width	$7.4\ in$	$0.188\ m$
Tip Height	$7.4\ in$	$0.188\ m$
Skin Thickness	$0.028\ in$	$0.7\ mm$
Vertical Fin Length	$33\ in$	$0.837\ m$
Horizontal Fin Length	$38.9\ in$	$0.99\ m$
m_1	$12.5\ lb$	$5.68\ kg$
m_2	$15\ lb$	$6.81\ kg$
m_3	$9.85\ lb$	$4.47\ kg$
m_s	$0.44\ lb$	$0.2\ kg$
k	$360\ lb/in$	$63000\ N/m$
K_t	$4 \times 10^3\ \frac{kips}{rad}$	$4.5 \times 10^5\ \frac{N \cdot m}{rad}$
Root Spring Constant	$2.7 \times 10^4\ \frac{kips}{rad}$	$3 \times 10^6\ \frac{N \cdot m}{rad}$
Tailboom Weight	$60\ lb$	$27.2\ kg$
Natural Frequencies	$12.3\ Hz, 21.8\ Hz$	
Damping Ratio	3%	

2.2.2 Full-Scale OH-58C Tailboom Model

The full-scale tailboom model is derived in a similar fashion to the lab-scale tailboom structure model. The beam model is illustrated in the diagram in Figure 2-6.

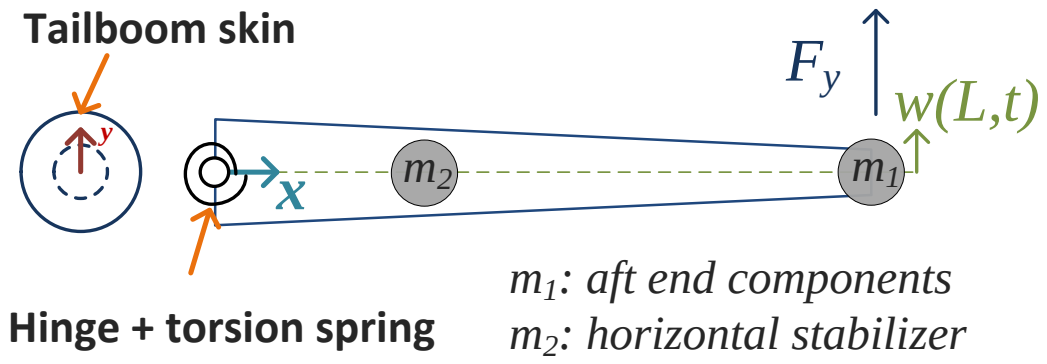

Figure 2-6. Diagram of the full-scale OH-58C tailboom model.

The beam model is developed using representative mass, damping, and flexural rigidity data provided by the manufacturer, Bell Helicopter. The tailboom data is omitted due to its proprietary nature. Mass m_1 represents the gearbox at the tip of the tailboom, and mass m_2 simulates the effect of various structural components in the tailboom, including the horizontal stabilizer. Lacking the large inertias seen in the vertical and horizontal stabilizers of the lab-scale tailboom structure, the OH-58C tailboom model has two fewer degrees of freedom. Thus, the governing equations are of the form

$$[M]\{\ddot{q}\} + [C]\{\dot{q}\} + [K]\{q\} = \{F\}, \tag{2.34}$$

where

$$M_{jn} = \int_0^L \rho A \psi_j \psi_n dx + m_1 \psi_j(L)\psi_n(L) + m_2 \psi_j(x_2)\psi_n(x_2), \tag{2.35}$$

$$K_{jn} = \int_0^L EI \psi"_j \psi"_n dx\,, \tag{2.36}$$

$$[\mathrm{C}] = [\Phi]\begin{bmatrix} 2\zeta\omega_1 & & \\ & \ddots & \\ & & 2\zeta\omega_N \end{bmatrix}[\Phi]^T, \tag{2.37}$$

and

$$\{F\} = \begin{bmatrix} \{\psi_L\} \\ 0 \\ 0 \end{bmatrix} F_y. \tag{2.38}$$

As with the lab-scale tailboom, basis functions $\psi_j(x)$ for a hinge-springed-free beam are used, with the root spring constant determined empirically. In state space form, Eq. (2.34) becomes

$$\begin{aligned} \{\dot{z}_1\} &= \{z_2\} \\ \{\dot{z}_2\} &= -[M]^{-1}[K]\{z_1\} - [M]^{-1}[C]\{z_2\} + [M]^{-1}\{F\}, \end{aligned} \tag{2.39}$$

where $\{z_1\} = \{q\}$ and $\{z_2\} = \{\dot{q}\}$.

2.3 Analytical Model of Fluidic Vibration Treatment

Firstly, the fluidic circuit model is introduced for the coupled F^{2}MC tube configuration. Secondly, a modified version of the braid-sheathed F^{2}MC tube model by Scarborough et al. [44] is presented and combined with the fluidic circuit model to produce the F^{2}MC absorber model. Finally, the absorber model is incorporated into a generic structural model.

2.3.1 Dynamic Model of Fluidic Circuit

Figure 2-7 shows four F^{2}MC tubes, of the type shown schematically Figure 2-8, arranged in the coupled configuration.

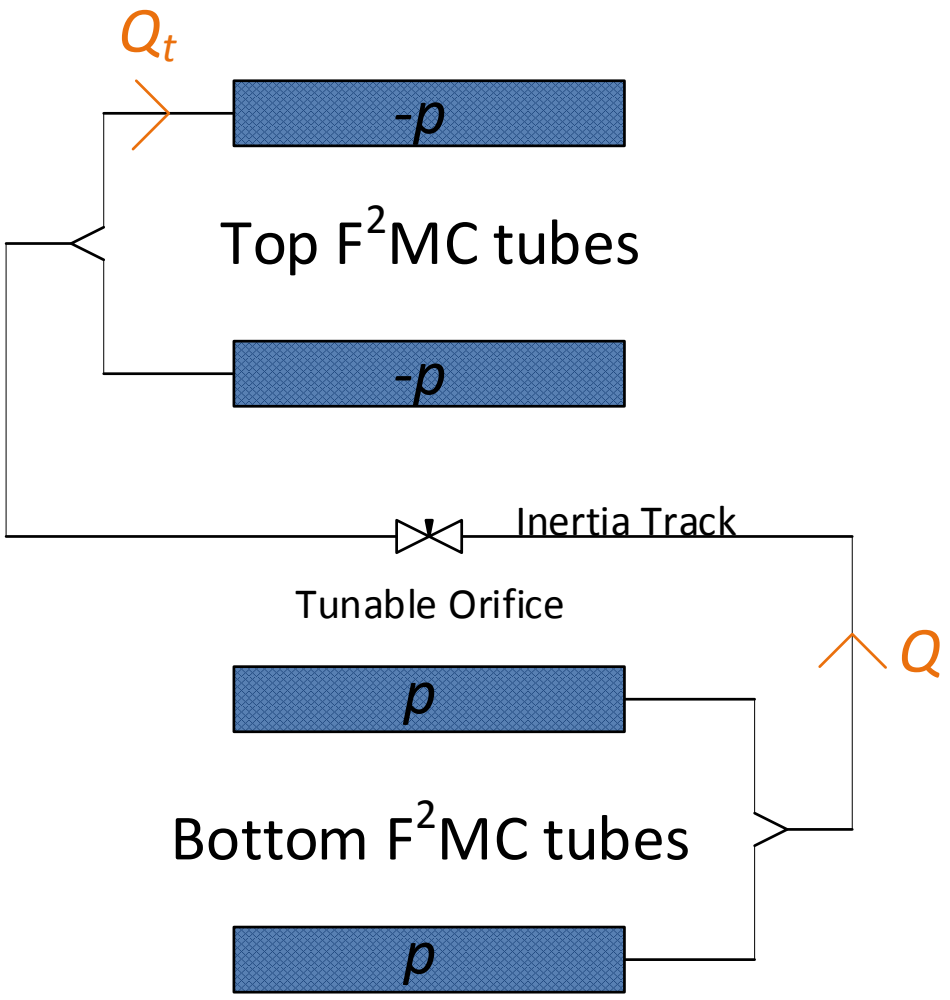

Figure 2-7. Diagram of F^{2}MC tubes arranged in the coupled configuration.

In the coupled configuration, pairs of F^{2}MC tubes are attached to the top and bottom surfaces of the tailboom. The top and bottom tubes are interconnected using an inertia track with a tunable orifice. As the tailboom vibrates, the top and bottom pairs of F^{2}MC tubes generate equal and

opposite gauge pressures p that drive flow through the interconnecting inertia track. The volumetric flow rate Q through the inertia track is governed by

$$I_f \dot{Q} + RQ = p - (-p) = 2p, \tag{2.40}$$

where inertance I_f is given by

$$I_f = \frac{\rho_f l_p}{\pi r_p^2} \alpha_I \tag{2.41}$$

and resistance R by

$$R = \frac{8\mu l_p}{\pi r_p^4} \alpha_R + R_{orf}, \tag{2.42}$$

where $r_p$is inertia track inner radius, l_p is inertia track length, ρ_f is fluid density, μ is fluid viscosity, and R_{orf} is orifice resistance. Since flow inertance and resistance in the inertia track vary with oscillation frequency, they must each be multiplied by correction factors α_I and α_R, respectively. $\alpha_{I,R}$ vary with frequency and are determined using plots generated by Donavan et al. relating the correction factors to non-dimensional frequency-dependent parameters. Without the correction factors, inertance would be under-predicted by about 30% and resistance by an order of magnitude for the frequency regime of interest. While this frequency-dependence presents an inconvenience (the system must be simulated at each frequency point to obtain a frequency response plot), the variation in resistance and inertance across a narrow frequency range is typically small. For instance, for the tailboom discussed in Chapter 3, the change in inertance over 6 Hz is negligible, and the increase in resistance is 20%. So, in practice one can use α values in the middle of the frequency range or take an average. The validity of this approximation can be seen in Figure 2-8, in which a plot using this approximation is compared to a plot resulting from calculating α at each frequency. By inspection, the differences between the frequency responses are negligible.

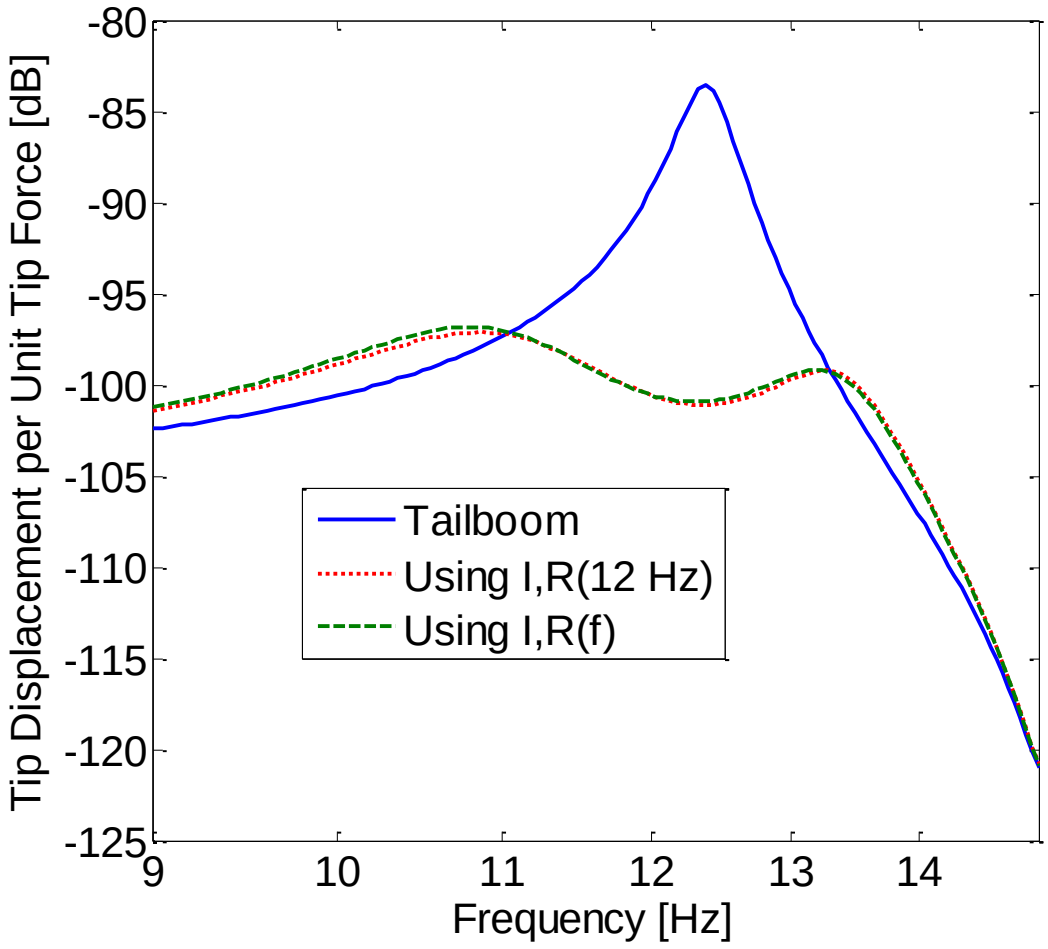

Figure 2-8. Comparison of frequency response plots using averaged and non-averaged fluid correction factors.

2.3.2 Model of Braid-Sheathed F^{2}MC Tube

The tensile axial load F_t acting on the F^{2}MC tube shown schematically in Figure 2-9 is

$$c_1 x_t + c_2 p = F_t, \tag{2.43}$$

and the volumetric fluid flow rate contributed by the F^{2}MC tube is

$$-c_3 \dot{x}_t - c_4 \dot{p} = \frac{Q}{n_p} = Q_t, \tag{2.44}$$

where positive x_t indicates tension of the bottom F^{2}MC tube, p is the internal fluid pressure, and positive Q describes fluid flow from the bottom to the top tubes in response to positive $w(x,t)$, or upward tailboom bending. $c_{1\sim4}$ are F^{2}MC parameters determined by the tube's dimensions, material, and fiber winding angle [44, 55]. c_{1-2} are computed numerically. c_3 may be calculated or measured. The empirically determined parameter c_4 represents compliance in the F^{2}MC tube walls. All four parameters except c_3 are positive. Although the prior braid-sheathed model neglected compliance effects [44], experiments described in Chapter 3 reveal that compliance has a significant effect on the absorber behavior. Thus, Eq. (2.44) contains the term $c_4\dot{p}$ to capture the tube compliance effect. (Such a correction was made by Shan et al. to the filament-wound model previously [48].)

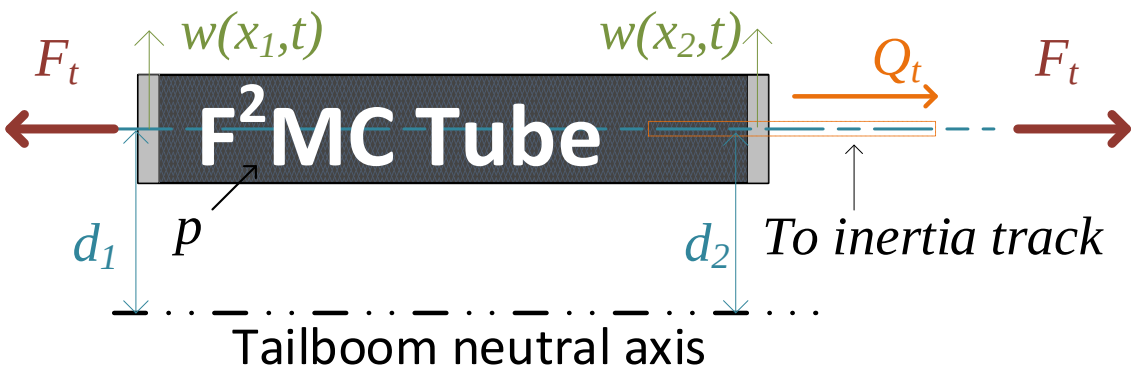

Figure 2-9. Diagram of a F^{2}MC tube.

Taking the Laplace transform of and combining Eqs. (2.40), (2.43), and (2.44) gives the transfer function from axial displacement to axial load

$$\frac{F_t(s)}{X_t(s)} = \frac{N_2 s^2 + N_1 s + N_0}{I_f s^2 + Rs + D_0}, \tag{2.45}$$

where

$$N_0 = \frac{2c_1}{n_p c_4}, \tag{2.46}$$

$$N_1 = \left(c_1 - \frac{c_2 c_3}{c_4}\right) R, \tag{2.47}$$

$$N_2 = \left(c_1 - \frac{c_2 c_3}{c_4}\right) I_f, \tag{2.48}$$

$$D_0 = \frac{2}{n_p c_4}, \tag{2.49}$$

n_p is the number of tubes per side, $F_t(s) = \mathcal{L}[F_t(t)]$, and $X_t(s) = \mathcal{L}[x_t(t)]$.

In state space form, Eq. (2.45) becomes

$$\begin{aligned} \{\dot{z}_3\} &= \begin{bmatrix} -R/I_f & -D_0/I_f \\ 1 & 0 \end{bmatrix} \{z_3\} + \begin{bmatrix} 1/I_f \\ 0 \end{bmatrix} x_t \\ &= [A_t]\{z_3\} + [B_t]x_t, \end{aligned} \tag{2.50}$$

and

$$\begin{aligned} F_t &= \begin{bmatrix} N_1 - \frac{N_2 R}{I_f} & N_0 - \frac{N_2 D_0}{I_f} \end{bmatrix} \{z_3\} + \frac{N_2}{I_f} x_t \\ &= [C_t]\{z_3\} + [D_t]x_t, \end{aligned} \tag{2.51}$$

where $\{z_3\}$ is the state vector of the fluidic system.

2.4 Analytical Model of a Tailboom with F^{2}MC Tubes

As fluid pressure develops inside the F^{2}MC tube, the tube exerts a force onto the tailboom surface to which it is attached. As the F^{2}MC tube is offset from the tailboom neutral axis, the tube effectively applies a moment onto the tailboom. Thus, to couple the fluidic and structural models together, Eq. (2.30) is modified to account for the influence of the moments exerted by the F^{2}MC tubes onto the tailboom structure. When the tailboom bends, a tube below the tailboom centroid has the strain

$$\varepsilon_t = d(x)\frac{\partial^2 w}{\partial x^2} = d(x)\{\psi"\}^T\{q\}. \tag{2.52}$$

where $d(x)$ is the vertical distance between the tube centroid and the tailboom neutral axis. Thus, for an F^{2}MC tube attached at longitudinal positions x_1 and $x_2 > x_1$,

$$x_t = \left(\int_{x_1}^{x_2} d(x)\,\{\psi"\}^T dx\right)\{q\} = \{d_t\}^T\{z_1\}, \tag{2.53}$$

where

$$\{d_t\} = \begin{bmatrix} \int_{x_1}^{x_2} d(x)\,\{\psi"\}dx \\ 0 \\ 0 \end{bmatrix}. \tag{2.54}$$

Substitute Eq. (2.53) into Eqs. (2.50) and (2.51):

$$\{\dot{z}_3\} = [B_t]\{d_t\}^T\{z_1\} + [A_t]\{z_3\} \tag{2.55}$$

and

$$F_t = [D_t]\{d_t\}^T\{z_1\} + [C_t]\{z_3\}. \tag{2.56}$$

The generalized forcing vector from Eq. (2.30) becomes

$$\{F_{eff}\} = \begin{bmatrix} \{\psi_L\} \\ 0 \\ 0 \end{bmatrix} F_y - 2n_p F_t\{d_{21}\psi'_{21}\}, \tag{2.57}$$

where

$$\{d_{21}\psi'_{21}\} = \begin{bmatrix} d(x_2)\{\psi'(x_2)\} - d(x_1)\{\psi'(x_1)\} \\ 0 \\ 0 \end{bmatrix}. \tag{2.58}$$

Substituting Eq. (2.56) into (2.57) gives

$$\{F_{eff}\} = \begin{bmatrix} \{\psi_L\} \\ 0 \\ 0 \end{bmatrix} F_y - 2n_p\{d_{21}\psi'_{21}\}([D_t]\{d_t\}^T\{z_1\} + [C_t]\{z_3\}). \tag{2.59}$$

Substitute Eq. (2.59) into (2.31) to obtain

$$\{\dot{z}_2\} = -\left[M_{eff}\right]^{-1}\left[2n_p[D_t]\{d_{21}\psi'_{21}\}\{d_t\}^T + \left[K_{eff}\right]\right]\{z_1\} - \left[M_{eff}\right]^{-1}\left[C_{eff}\right]\{z_2\} - 2n_p\left[M_{eff}\right]^{-1}\{d_{21}\psi'_{21}\}[C_t]\{z_3\} + \left[M_{eff}\right]^{-1}\{F_{eff}\}. \tag{2.60}$$

Let

$$\beta_a = -\left[M_{eff}\right]^{-1}\left[2n_p[D_t]\{d_{21}\psi'_{21}\}\{d_t\}^T + \left[K_{eff}\right]\right], \tag{2.61}$$

$$\beta_b = -\left[M_{eff}\right]^{-1}\left[C_{eff}\right], \tag{2.62}$$

and

$$\beta_1 = -2n_p\left[M_{eff}\right]^{-1}\{d_{21}\psi'_{21}\}[C_t]. \tag{2.63}$$

The coupled fluidic and structural equations are

$$\begin{bmatrix}\{\dot{z}_1\}\\ \{\dot{z}_2\}\\ \{\dot{z}_3\}\end{bmatrix} = \begin{bmatrix}[0] & [1] & [0]\\ [\beta_a] & [\beta_b] & [\beta_1]\\ [B_t]\{d_t\}^T & [0] & [A_t]\end{bmatrix}\begin{bmatrix}\{z_1\}\\ \{z_2\}\\ \{z_3\}\end{bmatrix} + \begin{bmatrix}\{0\}\\ \{U\}\\ \{0\}\end{bmatrix}F_y. \tag{2.64}$$

The tailboom displacement output

$$w(x,t) = [\{\psi(x)\}^T \quad \{0\}^T]\begin{bmatrix}\{z_1\}\\ \{z_2\}\\ \{z_3\}\end{bmatrix}. \tag{2.65}$$

The state space model of a tailboom with F^{2}MC tubes comprises Eqs. (2.64) and (2.65). This model is implemented in MATLAB for both the lab-scale tailboom structure and for the OH-58C structural model (see Appendix B).

Chapter 3

Experimental Validation of the F^{2}MC-Absorber on a Simulated Tailboom

The model developed in Chapter 2 is validated experimentally on the lab-scale tailboom structure. The experimental test stand is first introduced. Then, results of static actuation and pumping tests are presented, followed by benchtop component-level testing results. Finally, the dynamic response of the treated and untreated tailboom is measured and showed to correlate with theoretical predictions.

3.1 The Lab-Scale Tailboom Structure

The lab-scale tailboom structure described in Section 2.2.1 is fitted with four F^{2}MC tubes (two on top, two on bottom) and an accompanying fluidic circuit. The test stand is pictured in Figure 3-1.

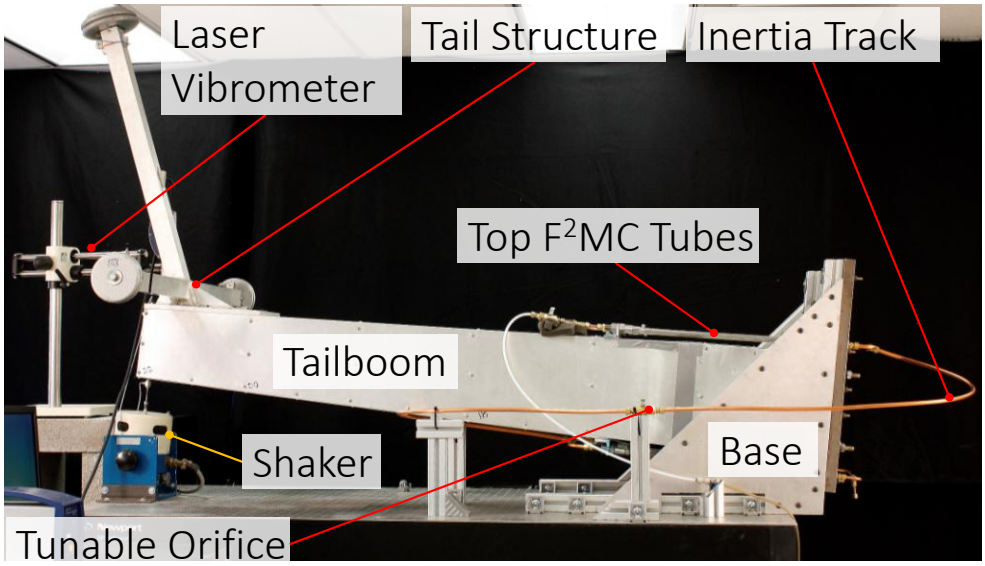

Figure 3-1. F^{2}MC tubes installed onto a lab-scale tailboom structure.

The tailboom structure is cantilevered to a heavy mount on a vibration table. A shaker is used to vertically excite the tailboom at the tip. The vertical tailboom tip response is measured using a laser vibrometer. As illustrated in Figure 3-2, pairs of prototype F^2MC tubes are attached to the top and bottom surfaces of the tailboom. One end of each F^2MC tube is anchored to the base of the cantilever, and the other is attached to aluminum L-brackets bolted to the tailboom 0.56 m (22 in) from the base. For ease of installation and observation, the F^2MC tubes are offset 25.4 mm (1 in) from the tailboom skin. The distance between the end fittings (i.e. the effective length of the F^2MC tubes) is 0.38 m (15 in), as illustrated in Figure 1-4.

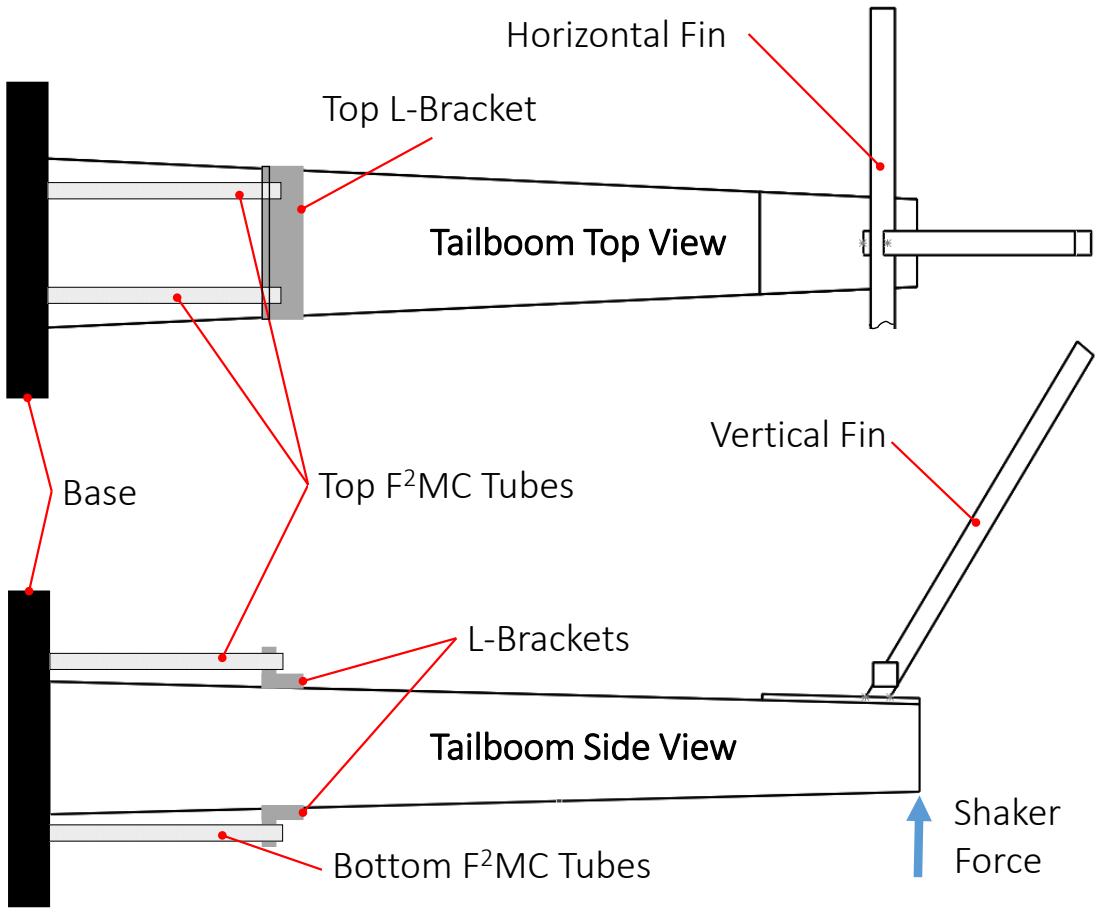

Figure 3-2. Schematic diagram of F^2MC tubes attached to the lab-scale tailboom structure.

The F^2MC tubes are connected to the fluidic circuit, shown schematically in Figure 3-3, which includes a fill valve, bleed valve, tunable orifice, and an inertia track. The circuit is filled by pumping fluid through the fill valve, the lowest point of the circuit, and bleeds air out of the bleed valve, the highest point. As evident in Figure 3-1, the inertia track monotonically increases in height from the fill valve to the bleed valve. This is to ensure that air bubbles, which are shown in Section 3.3 to affect absorber performance negatively, do not become entrapped in the fluidic circuit and to facilitate removal of bubbles from the fluid.

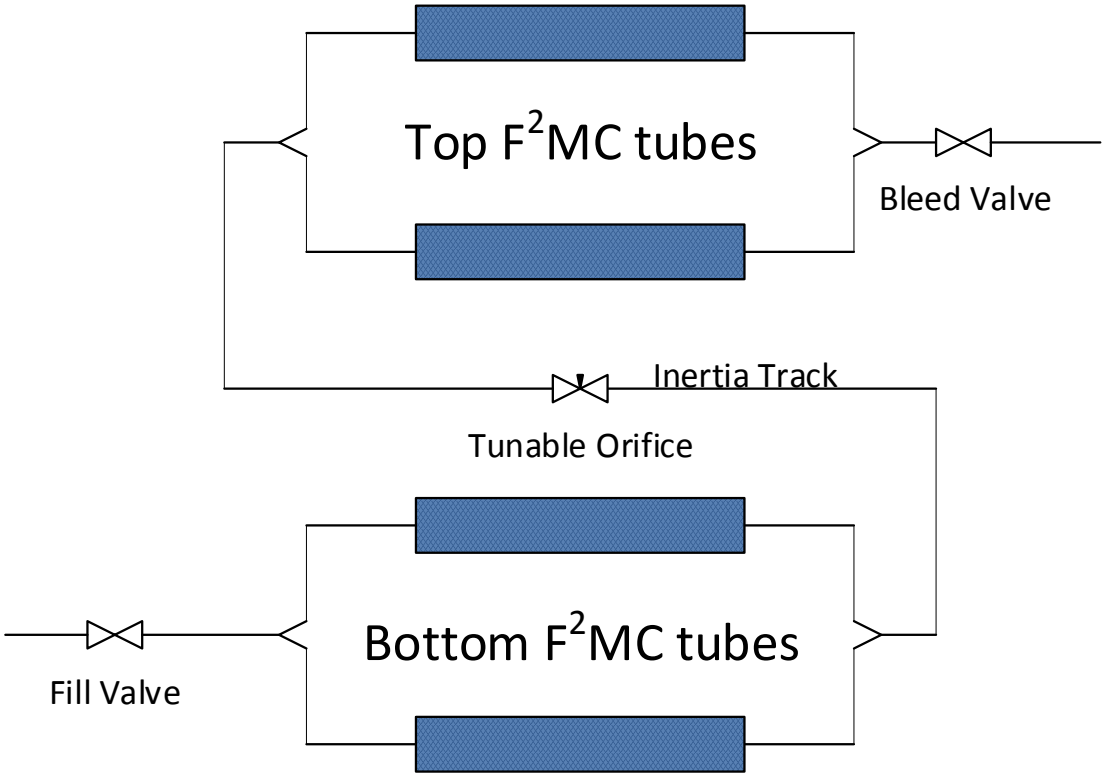

Figure 3-3. Fluidic circuit diagram of the tailboom test stand.

3.2 Static Tailboom Tests

Two series of static tests are performed to characterize the F^{2}MC tube pumping and actuation capability. In order to damp vibration, the F^{2}MC tubes must be able to pump fluid when the tailboom vibrates and actuate tailboom displacement when pressurized. Thus, a solid understanding of the static behavior is needed before damping performance can be demonstrated experimentally. The static pumping and actuation experiments are described and their respective equations derived. Experimental measurements of the tube volume change due to imposed tailboom displacement, as well as tailboom tip displacement in response to tube pressurization, are presented.

3.2.1 Static Test Stand

The static tests require a modified version of the fluidic circuit, shown in Figure 3-4. The prototype F^{2}MC tubes used are pictured in Figure 3-5.

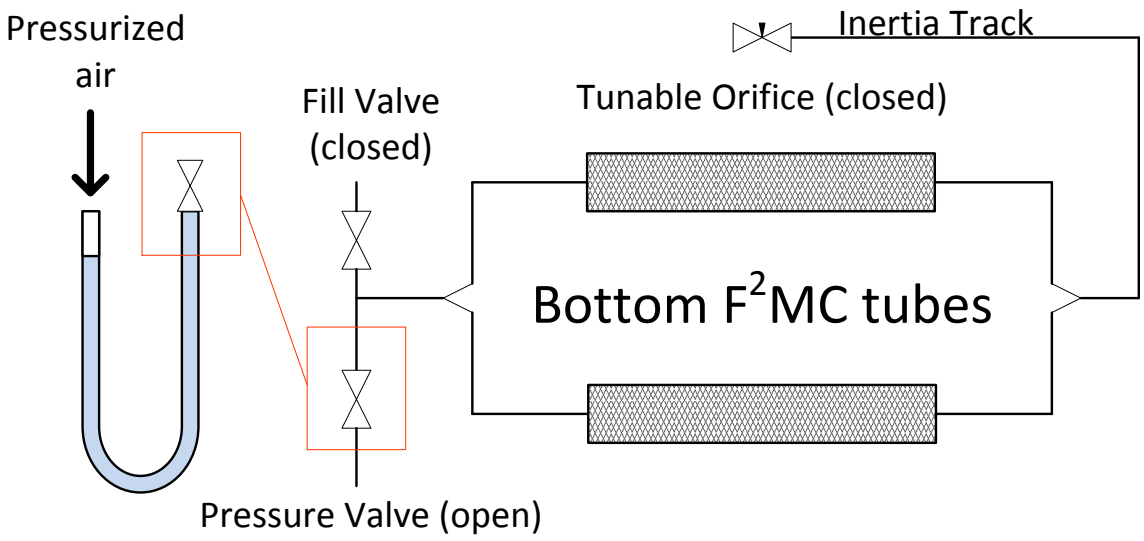

Figure 3-4. Fluidic circuit diagram for the static tailboom test.

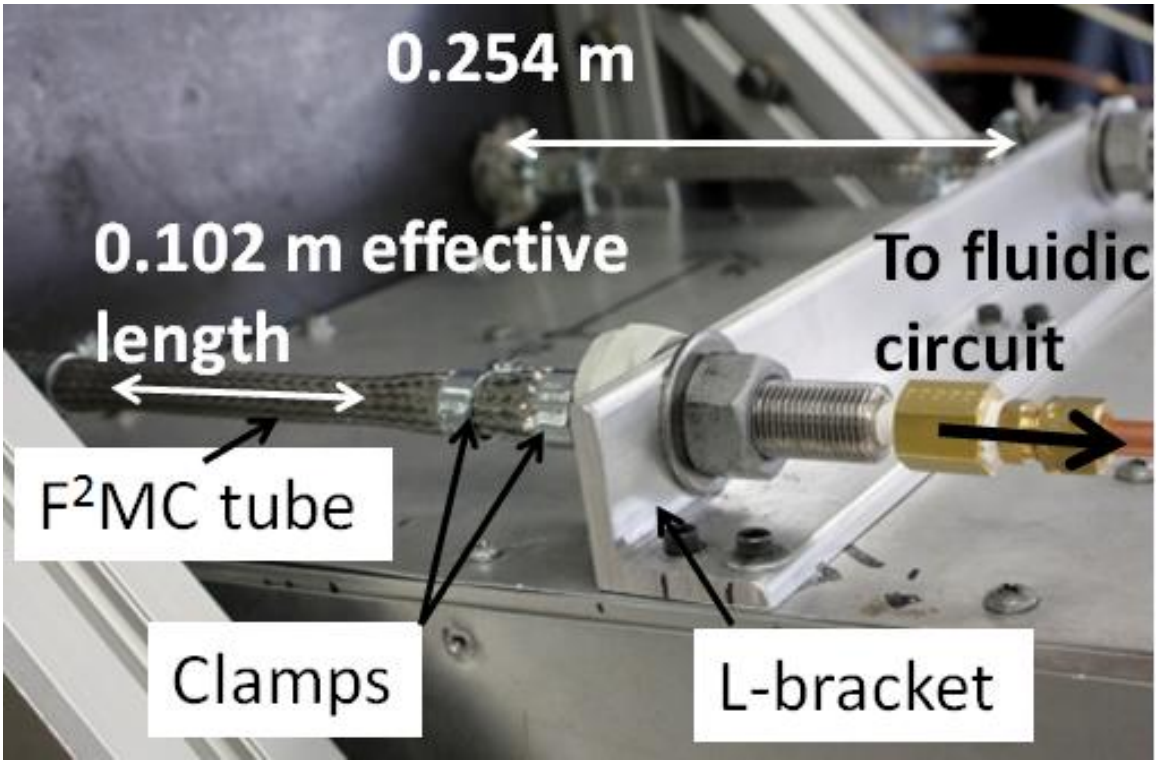

Figure 3-5. F^2MC tubes attached to the top of the tailboom structure for static testing.

One end of each F^2MC tube is anchored to the base of the cantilever. To anchor the tubes firmly, the threaded rod end fittings are placed through holes drilled into the cantilever wall/steel plate. By tightening nuts on either side of the plate, the F^2MC tube is secured. The other is attached to aluminum L-brackets bolted to the tailboom 0.254 m from the base. The F^2MC tubes are secured to the vertical section of the L-brackets in the same fashion as with the cantilever base. For ease of installation and observation, the F^2MC tubes are offset 25.4 mm from the tailboom skin (a production solution would likely be implemented inside of the tailboom). The distance between the end fittings (the effective length of the F^2MC tubes) is 0.102 m. Pressure can be applied to the fluid via the pressure valve. Upstream of the pressure valve is a length of clear, rigid plastic tubing through which a column of fluid can be observed. The test parameters are summarized in Table 3-1. For this test, 91% isopropyl alcohol is used because it is a safe, readily available fluid

with lower surface tension than water, making it less prone to forming air bubbles. As discussed in Section 3.3, air bubbles are highly undesirable in this application. A fiber winding angle of 20 degrees is chosen because, as discussed in Chapter 4, smaller fiber angles typically result in higher pumping, and 20 degrees was the smallest angle achievable with the readily available fiber meshes at the time. Supplied by McMaster-Carr, corrosion-resistant 25/32" ID Super-Abrasion-Resistant Expandable Sleeving was used for the fiber sleeving.

Table 3-1. Static Tailboom Test Stand Parameters.

Characteristic	Imperial	Metric
Fluid	91% Isopropyl Alcohol	
F^2MC tube		
- length	$3.98\ in$	$101\ mm$
- inner radius	$0.156\ in$	$3.97\ mm$
- fiber angle	$20°$	

3.2.2 Actuation Test

The authority and pumping amplification mechanics of the F^2MC tube are evaluated by actuating the tailboom. Internally pressurizing the F^2MC tubes causes the tailboom to bend. To demonstrate this, the fluidic circuit is configured as shown in Figure 3-4. The tunable orifice in the inertia track is closed, preventing fluid flow. The fluid in the bottom F^2MC tubes is pressurized using compressed air, while the fluid in the top half of the circuit is left open to the atmosphere. As fluid pressure in the bottom tubes increases, the bottom F^2MC tubes expand radially and contract axially. The ends of the F^2MC tubes are attached to the tailboom, so the axial contraction exerts a moment on the tailboom, causing it to bend downward. This downward deflection is measured using a laser vibrometer. A plot of tailboom vertical deflection vs. applied fluid pressure is obtained to evaluate the actuation capability of the F^2MC tubes.

3.2.3 Actuation Model

To predict actuation, the model developed in Chapter 2 is adapted for the static case. Figure 3-6 shows the cantilevered beam model with F^{2}MC tubes attached to the top and bottom.

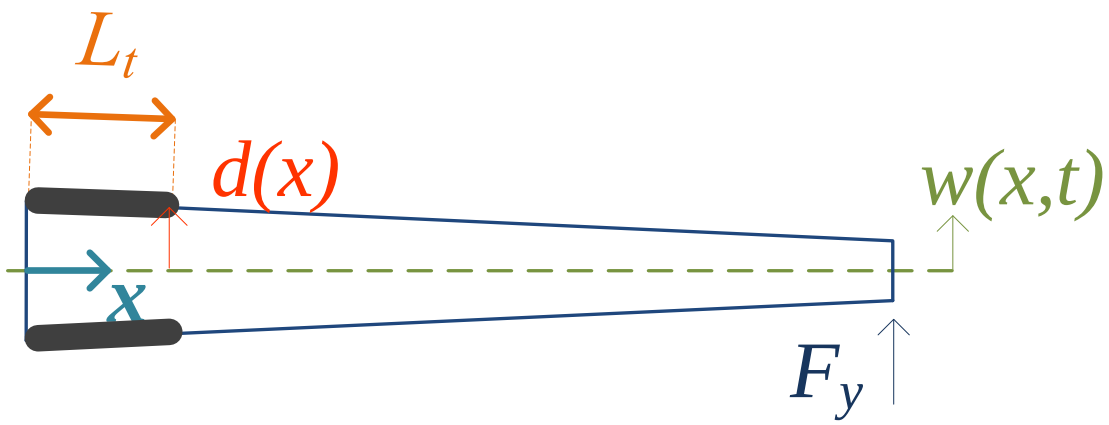

Figure 3-6. Schematic of the cantilevered beam model with F^{2}MC tubes.

For the purely static case, Eq. (2.25) can be simplified significantly; all derivatives of $\{q\}$ are zero, and y and θ may be neglected since those motions are zero in the absence of dynamic forcing. Thus, the static response satisfies

$$[K]\{q\} = \{F\}, \tag{3.1}$$

where

$$K_{jn} = \int_0^L EI \frac{d^2\psi_j}{dx^2}\frac{d^2\psi_n}{dx^2}\,dx, \tag{3.2}$$

is the stiffness matrix of the tailboom structure, and

$$\{F\} = F_y\{\psi(L)\} + 2n_p(F_t)\{d_{21}\psi'_{21}\}, \tag{3.3}$$

where L_t is the tube length. Substituting Eq. (2.53) into Eq. (2.43) gives

$$F_t = c_1\{d_t\}^T\{q\} + c_2 p \tag{3.4}$$

Substituting Eq. (3.4) into Eq. (3.3), we get

$$[K]\{q\} = F_y\{\psi_L\} - 2n_p(c_1\{d_t\}^T\{q\} + c_2 p)\{d_{21}\psi'_{21}\}. \tag{3.5}$$

Since the actuation test involves no external excitation, $F_y = 0$. Solving Eq. (3.5) for $\{q\}$ yields:

$$\{q\} = -2n_p c_2[J]\{d_{21}\psi'_{21}\}p, \tag{3.6}$$

where

$$[J] = \left[[K] + 2n_p c_1(\{d_t\}\{d_{21}\psi'_{21}\}^T)^T\right]^{-1}. \tag{3.7}$$

Observing that $w_L = \{\psi_L\}^T\{q\}$, we premultiply Eq. (3.6) by $\{\psi_L\}^T$ to obtain

$$w_L = -2n_p c_2\{\psi_L\}^T[J]\{d_{21}\psi'_{21}\}p. \tag{3.8}$$

Eq. (3.8) relates tailboom tip displacement to the fluid pressure inside the fluidic circuit and depends on the geometric and material properties of the F^{2}MC tubes and cantilever beam.

3.2.4 Experimental Actuation Results

Figure 3-7 shows the experimental results of tailboom tip displacement as a function of tube pressure. The F^2MC tubes have sufficient authority to cause the tailboom to deflect almost 1 mm at 414 kPa. During depressurization, the tailboom displacement is larger than during pressurization. This slight hysteresis may be due to friction in the tailboom structure, F^2MC tube attachment hardware, and between the stainless steel mesh and rubber bladder. The results agree with the model predictions, proving its ability to predict actuation up to 414 kPa. Higher pressures were not tested due to limitations in the available air pressure source.

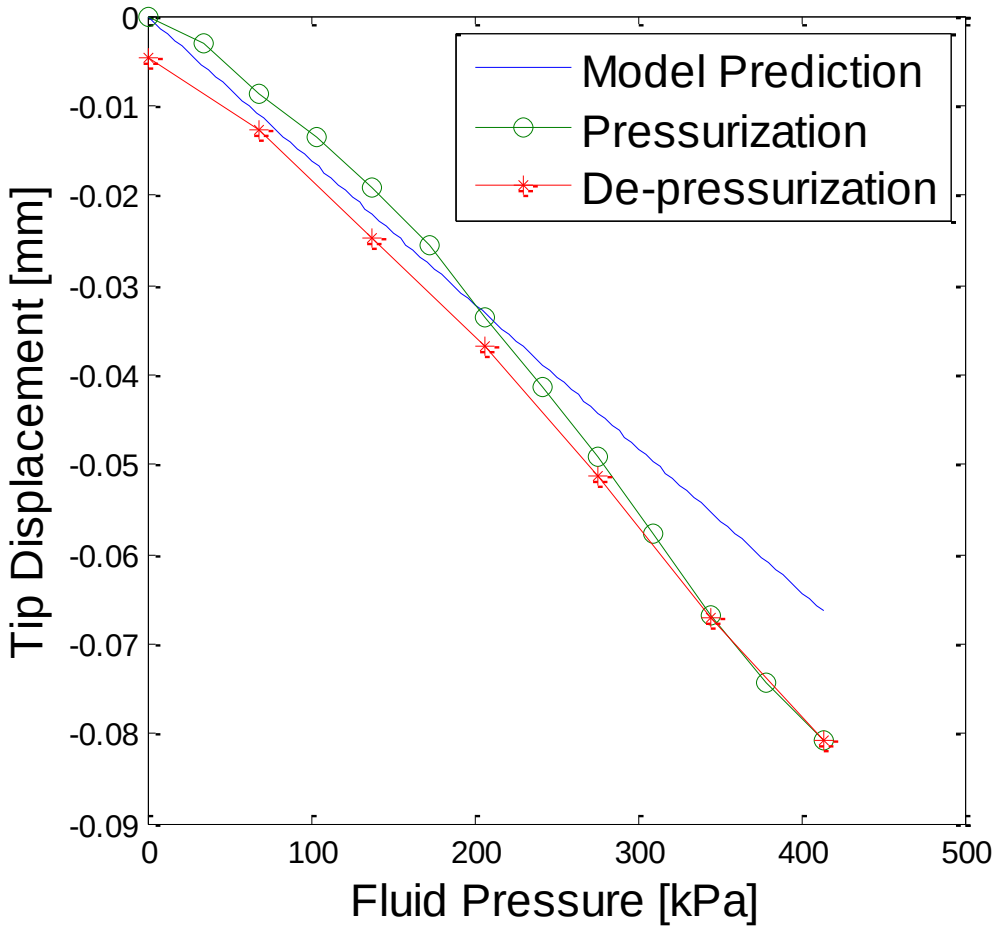

Figure 3-7. Theoretical and experimental tailboom tip deflections versus F^2MC tube pressure.

3.2.4 Pumping Test

As the tailboom deflects, F^{2}MC tubes on one side contract while the others extend, resulting in fluid pumping between the top and bottom tubes. To quantify the fluid pumping ability of the system, the tailboom is statically loaded near the tip and the fluid volume pumped out of the tubes is measured.

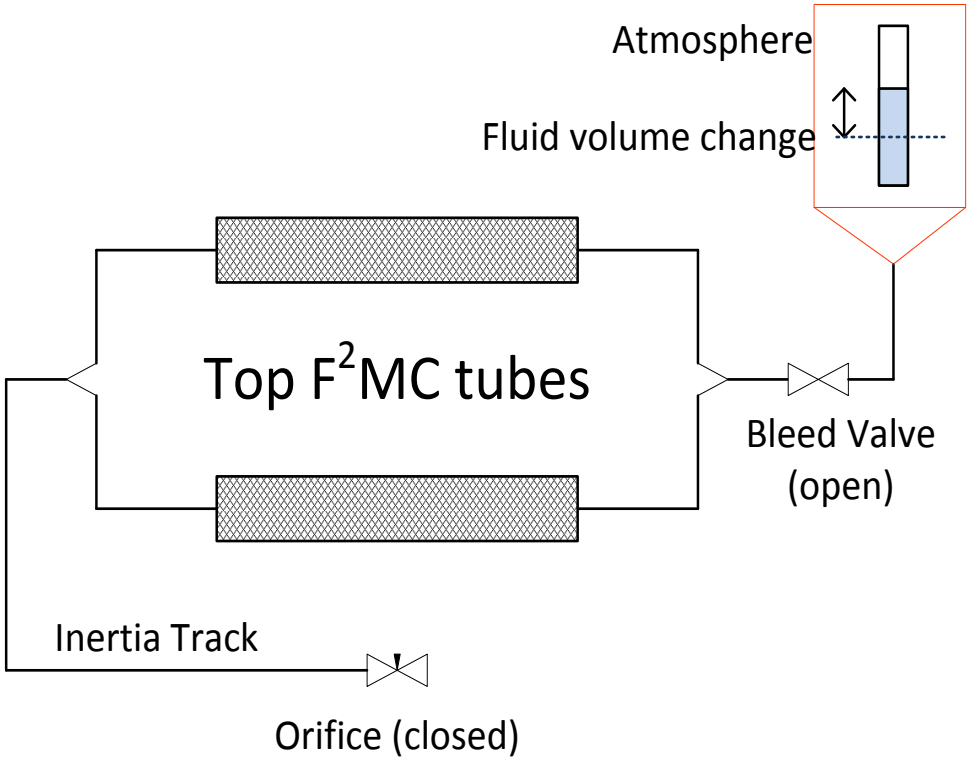

Figure 3-8. Schematic of the fluid pumping experiment fluidic circuit.

To enable this measurement, the fluidic circuit is configured as illustrated in Figure 3-8. A short length of clear 1.59 mm diameter hard plastic tubing is connected to the bleed valve, so the height of the fluid column fluid can be measured. The tunable orifice is closed and the bleed valve is opened, allowing the top F^{2}MC tubes to pump fluid into the atmosphere. Static loads are

applied to the tailboom, causing it to deflect downward. As in the actuation experiment, tailboom displacement is measured with the laser vibrometer.

3.2.5 Pumping Model

The volume of fluid pumped by the tube can be found by integrating Eq. (2.44) with respect to time:

$$V = -c_3 x_t - c_4 p. \tag{3.9}$$

Setting $p = 0$ and substituting Eq. (2.53), Eq. (3.9) becomes

$$V = -c_3 \{d_t\}^T \{q\}. \tag{3.10}$$

Note that c_3 is computed numerically for this study. Similarly, Eq. (3.5) becomes

$$[K]\{q\} = F_y\{\psi_L\} - 2n_p(c_1\{d_t\}^T\{q\})\{d_{21}\psi'_{21}\}. \tag{3.11}$$

Combining like terms and solving for $\{q\}$,

$$\{q\} = F_y[J]\{\psi_L\}. \tag{3.12}$$

The tip displacement

$$w_L = \{\psi_L\}^T\{q\} = \{\psi_L\}^T[J]\{\psi_L\}F_y. \tag{3.13}$$

Substituting Eq. (3.12) into Eq. (3.10) gives

$$V = -c_3\{d_t\}^T[J]\{\psi_L\}F_y. \tag{3.14}$$

Finally, Eqs. (3.13) and (3.14) are combined to obtain

$$V = -\frac{c_3\{d_t\}^T[J]\{\psi_L\}}{\{\psi_L\}^T[J]\{\psi_L\}}w_L, \tag{3.15}$$

the static relationship between fluid pumping and tip displacement.

3.2.6 Experimental Pumping Results

Figure 3-9 compares the theoretically predicted and experimentally measured volume versus displacement. The experimental results confirm the predicted linear relationship between fluid volume change and tailboom tip displacement. During the loading portion of the experiment, weights are added at the tip of the tailboom. The slope of the least squares fit of the loading data is 10% lower than the theoretically predicted slope. Compliance in the L-bracket attaching the F^2MC tubes is one possible explanation; if the attachment is not perfectly rigid, then the tubes will contract less, reducing fluid volume change. The weights are then removed to produce the unloading curve. Again, there is hysteresis, so after fully unloading the tailboom tip was 0.512 mm lower than where it originated.

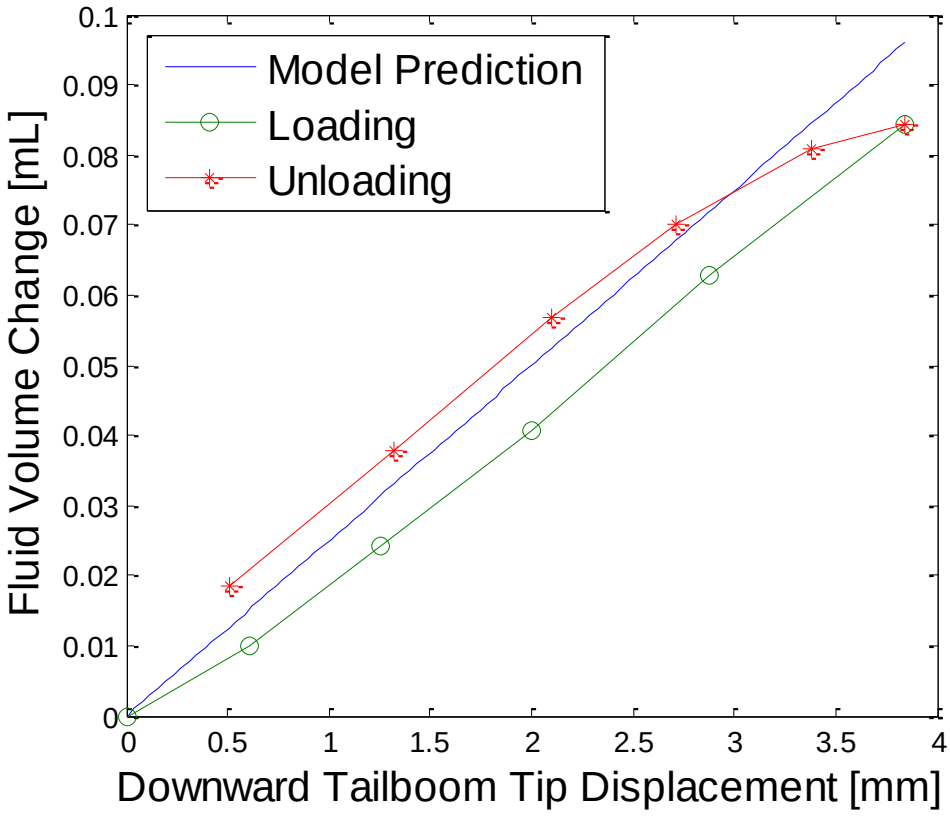

Figure 3-9. Theoretical and experimental fluid volume change versus tailboom tip deflection.

3.3 Benchtop F^{2}MC Tube Testing

The governing equations of the F^{2}MC tube have four parameters: c_{1-4}. c_1 and c_2 are computed numerically by solving the nonlinear equations derived by Scarborough et al. [44]. The solution process can be seen in the MATLAB code in Appendix B, under the mcKibbenTubeConstants(P) subroutine. The MATLAB function fsolve.m is used for the numerical solutions, the settings are specified in the variable "options," and Scarborough's initial guesses are specified as the second argument of fsolve.m in Scarborough's code. c_3 and c_4 are determined empirically. This section describes the work conducted to measure these two parameters and better understand braid-sheathed F^{2}MC tube behavior at the component level.

c_3 describes the F^{2}MC tube pumping ability, or the volume of fluid pumped per unit axial tube displacement. Larger pumping coefficients are desirable since they result in more inertance and therefore more weight-efficient absorbers. c_4 is the tube compliance. Braid-sheathed F^{2}MC tubes encounter three main sources of compliance. As illustrated in Figure 3-10, the first is air bubbles in the fluid, which reduce the utilization of the working fluid. This is solved by pressurizing the fluid such that the air bubbles collapse to a negligible volume. As shown in Figure 3-11, the second is poor engagement between the bladder and fiber mesh, which results in under-utilization of axial displacement. This is solved by pre-tensioning the F^{2}MC tube, which causes the fiber mesh tube to constrict and better engage with the inner bladder. The third source of compliance is the internal bladder. Shan et al. showed that the inner liner, which has a bulk modulus less than two orders of magnitude of that of the fluid, is a significant source of compliance [48]. A thinner liner material produces a tube with a higher modulus ratio between the open and closed-valve case, since a smaller volume of material is compressed. Thus, both the bladder bulk modulus and thickness are important determinants of F^{2}MC tube performance. The

work presented here not only characterizes F^{2}MC tube pumping and compliance, but also seeks to provide a solution to the bladder compliance issue.

In terms of bladder design, the parameters c_3 and c_4 present a practical engineering tradeoff. Lower compliance c_4 is desired and requires a stiff bladder material. However, stiffer bladder materials tend to have worse pumping performance. In the most extreme case, a rigid metal bladder would allow virtually no compliance but provide zero pumping benefit. To find a suitable bladder material and thickness, a number of F^{2}MC tubes were built using candidate bladders, and their respective pumping (c_3) and compliance (c_4) values are measured. The measured parameters are then inputted to the model to simulate the dynamic performance for each type of F^{2}MC tube.

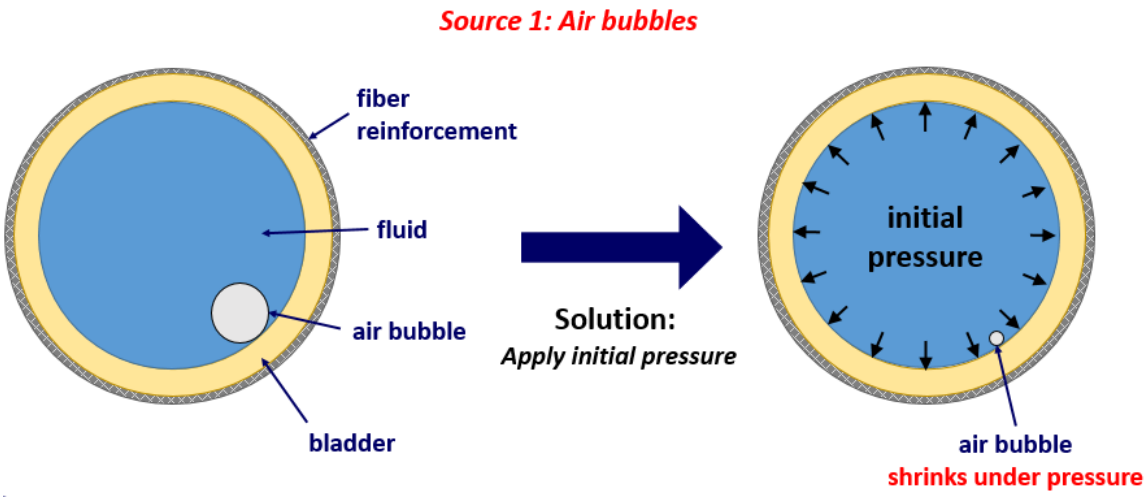

Figure 3-10. Illustration of the effect of air bubbles in the fluid.

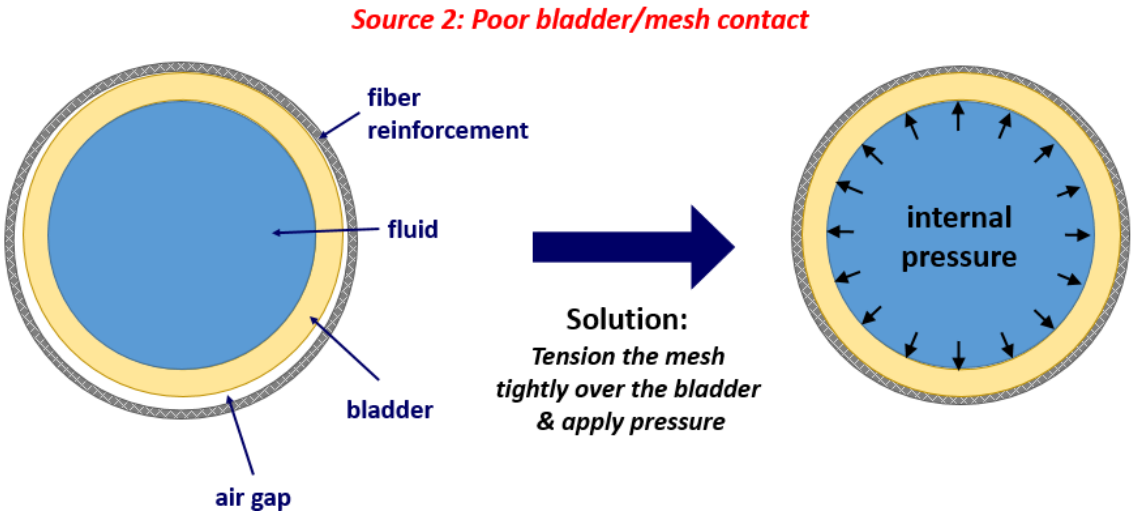

Figure 3-11. Illustration of the effect of poor engagement between the bladder and fiber mesh.

3.3.1 Benchtop Test Apparatus

F^2MC tubes are constructed and secured to a table, as pictured in Figure 3-12. Each tube is attached to steel L-brackets, and nuts are used to pre-tension the F^2MC tube until the diameter of the tube midpoint reaches a pre-determined nominal value of 9.86 mm (0.388 in). This pre-tensioning is needed to ensure sufficient engagement between the internal bladder and the reinforcing fibers.

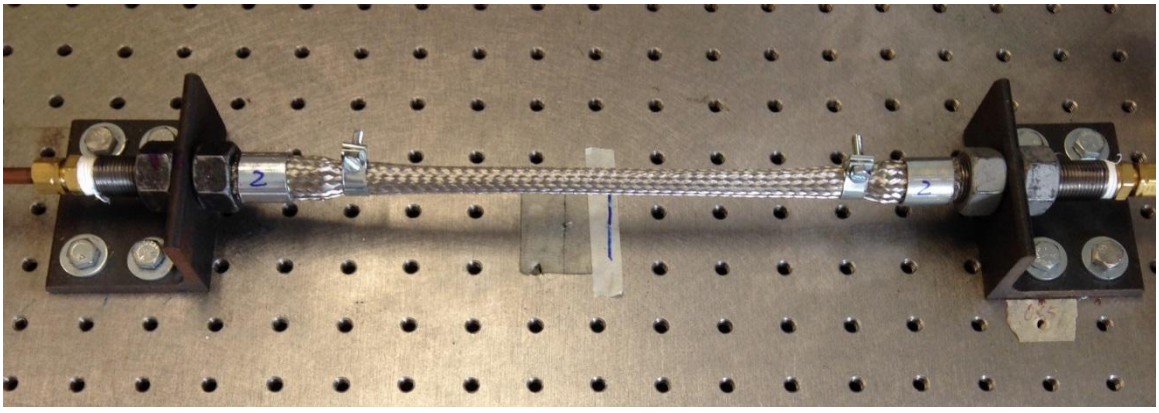

Figure 3-12. Benchtop F^2MC test stand.

The bladder materials tested are summarized in Table 3-2 and pictured in Figure 3-13. As the suppliers (Qcare for the 1/32" Latex Rubber, McMaster Carr for all other tubing) do not provide elastic modulus data for these materials, they are estimated using Shore Durometer correlations. The F^2MC tubes have an inner diameter of 9.53 mm (3/8 in) and a nominal effective length of 152 mm (6 in). Bladder thicknesses of 3.18 mm (1/8 in) and 1.59 mm (1/16 in) are tested for each material. 0.79 mm (1/32 in) thick latex rubber is also tested.

Table 3-2. List of bladder materials tested.

Bladder Material	Estimated Elastic Modulus [MPa]
Latex Rubber	1
Santoprene	4
Masterkleer PVC	4.7
Crack-resistant polyethylene	45

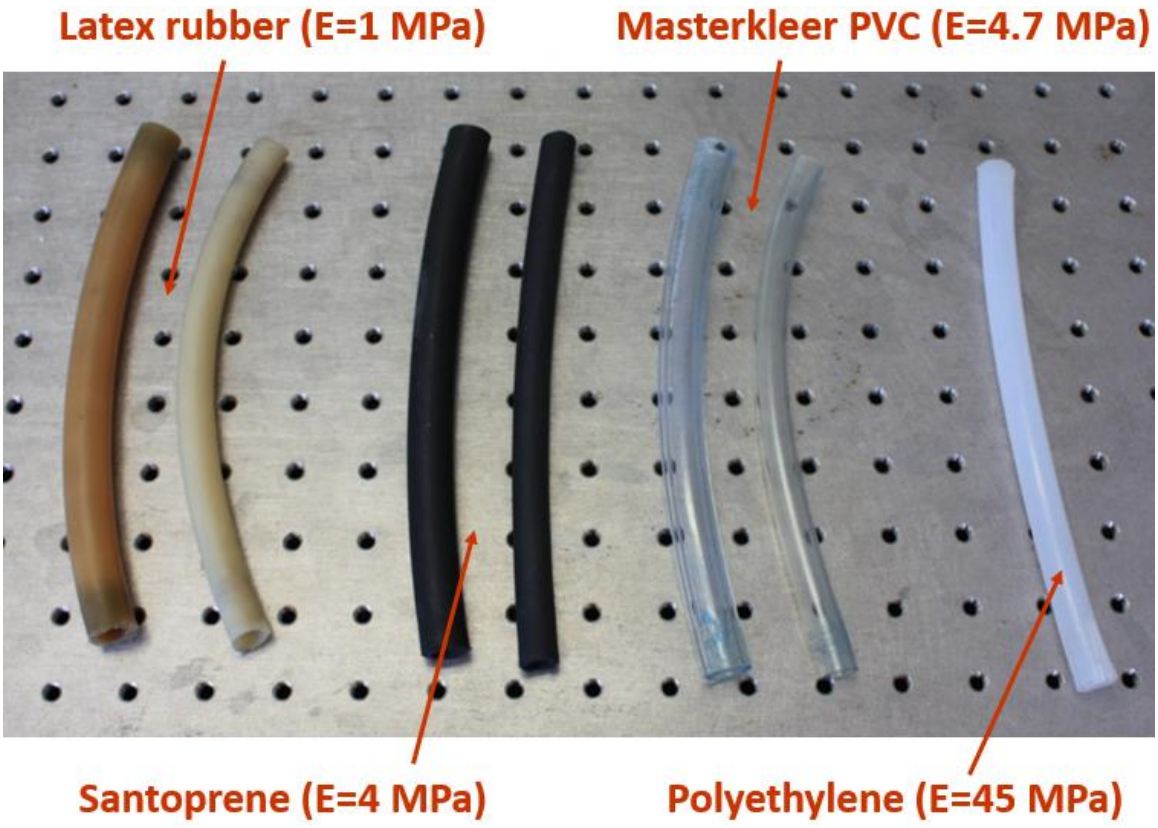

Figure 3-13. Photograph of the different bladders tested.

3.3.2 Pumping Test

The fluidic circuit diagram for the pumping test is shown schematically in Figure 3-14. The F^{2}MC tube is connected to clear, rigid plastic tubing, through which a column of fluid (91% isopropyl alcohol) can be observed. The clear tubing is made of 3.18 mm (1/8 in) inner diameter hard propylene. The circuit is filled through the fill valve and all air bubbles are removed from the fluid. After closing the fill valve, the nuts securing the end fittings to the L-brackets are turned to adjust the length of the F^{2}MC tube. This axial displacement causes a volume change inside the F^{2}MC tube, which is observed as a change in fluid column height. Volume pumped is plotted against the axial displacement of the F^{2}MC tube, and the slope of the resulting curve yields the pumping coefficient c_3.

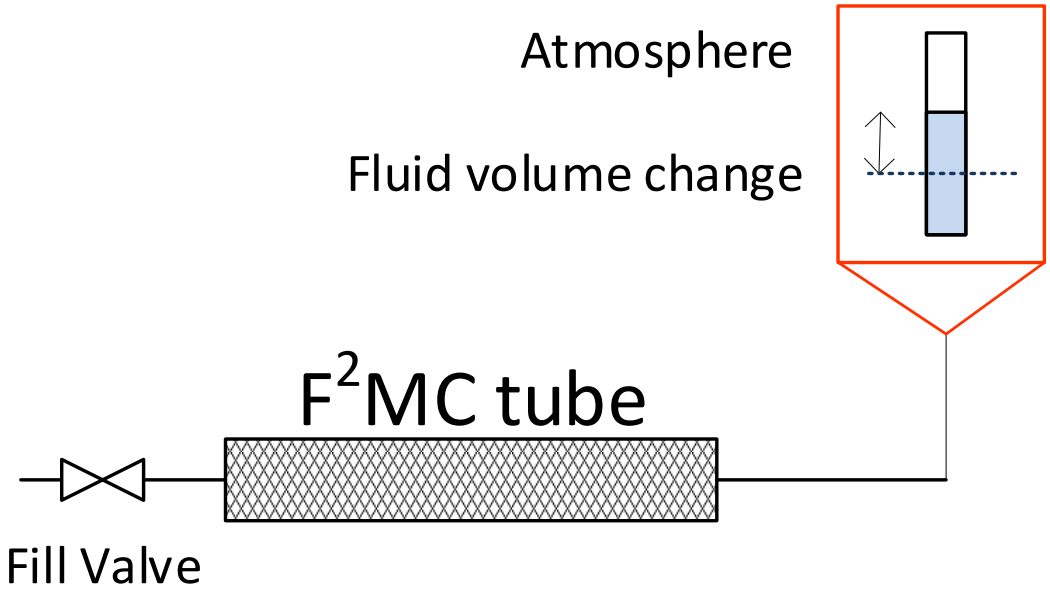

Figure 3-14. Fluidic circuit diagram of the benchtop pumping test.

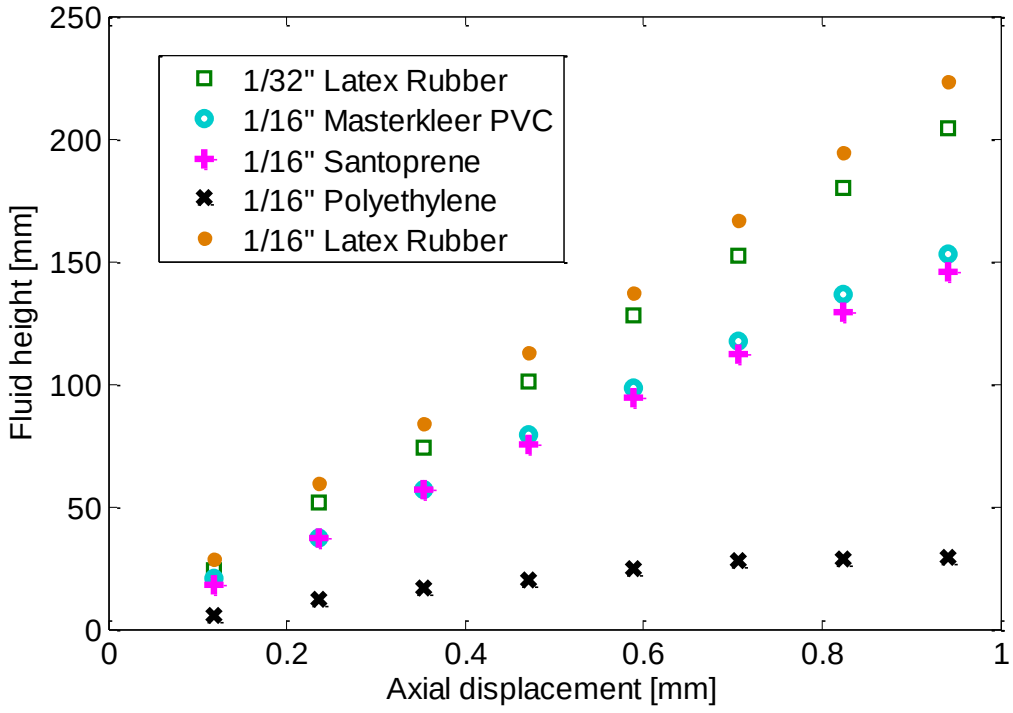

Figure 3-15. Experimental measurements of fluid column height vs. axial displacement for different bladders.

Figure 3-15 shows the experimental pumping results obtained for 1.59 mm (1/16 in) thick bladders for all materials, as well as for 0.79 mm (1/32 in) thick latex rubber. Bladders with better pumping ability exhibit steeper slopes. 1.59 mm (1/16 in) polyethylene, the stiffest of the tested bladders, pumps the least fluid, whereas the latex rubber bladders pump the most. The results summarized in Figure 3-16 show clearly that stiffer materials perform worse on the pumping test than those with lower moduli. A four-fold increase in elastic modulus between Latex Rubber and Santoprene resulted in an approximately 21% decrease in pumping performance. A 45-fold increase in modulus between Latex Rubber and Polyethylene resulted in over 86% reduction in pumping.

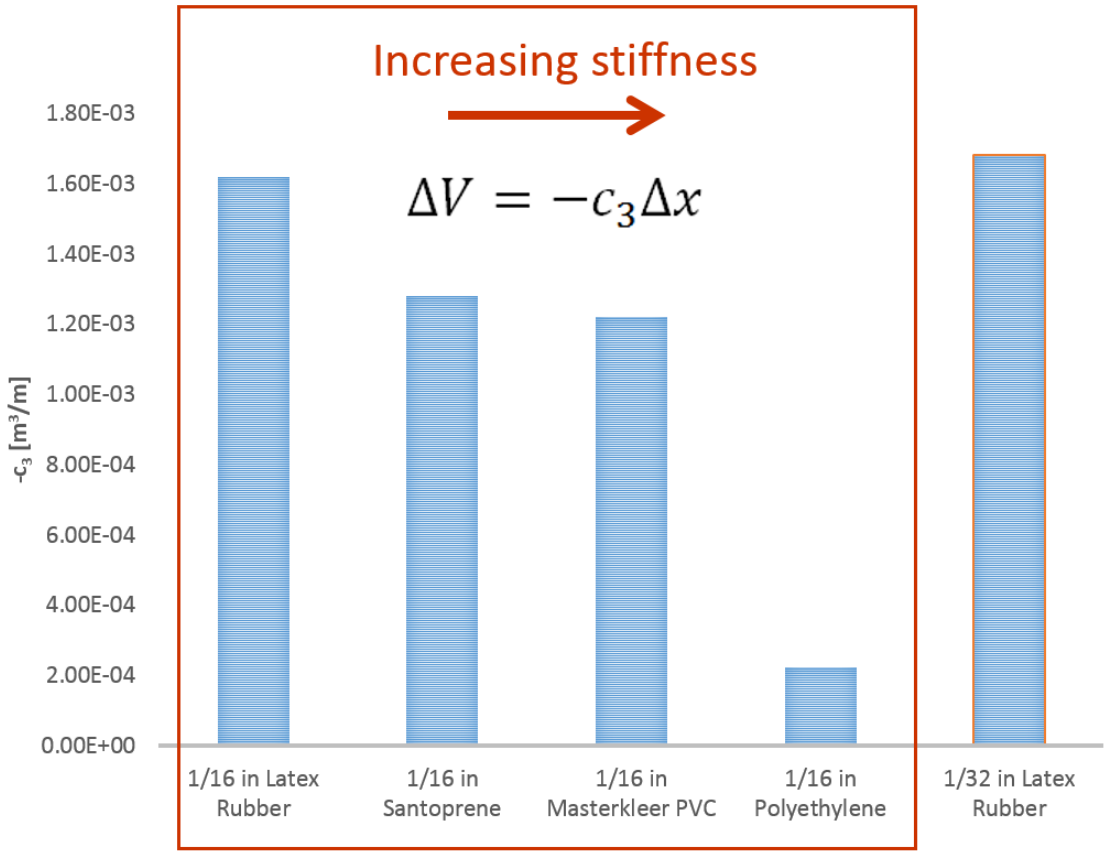

Figure 3-16. Comparison of pumping coefficient for different bladders.

3.3.3 Compliance Test

The fluidic circuit diagram for the compliance test is shown schematically in Figure 3-17. A F^{2}MC tube is mounted to the table in the same fashion as in the pumping test. The tube is pre-tensioned to the nominal setting used in the pumping test and its ends are fixed such that the tube does not extend or contract. Upon filling the system and removing air bubbles, air in a clear plastic column is pressurized. As the system is pressurized, the F^{2}MC tube expands, causing a drop in the fluid column level. Fluid volume measurements are taken at increments of 5 psi of pressure.

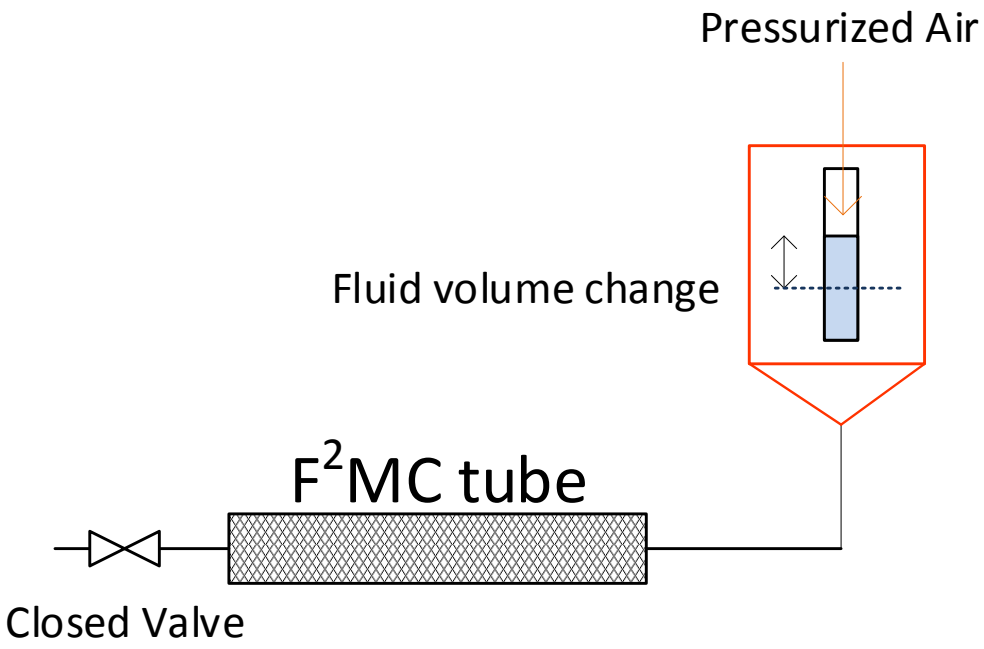

Figure 3-17. Fluidic circuit diagram for the benchtop compliance test.

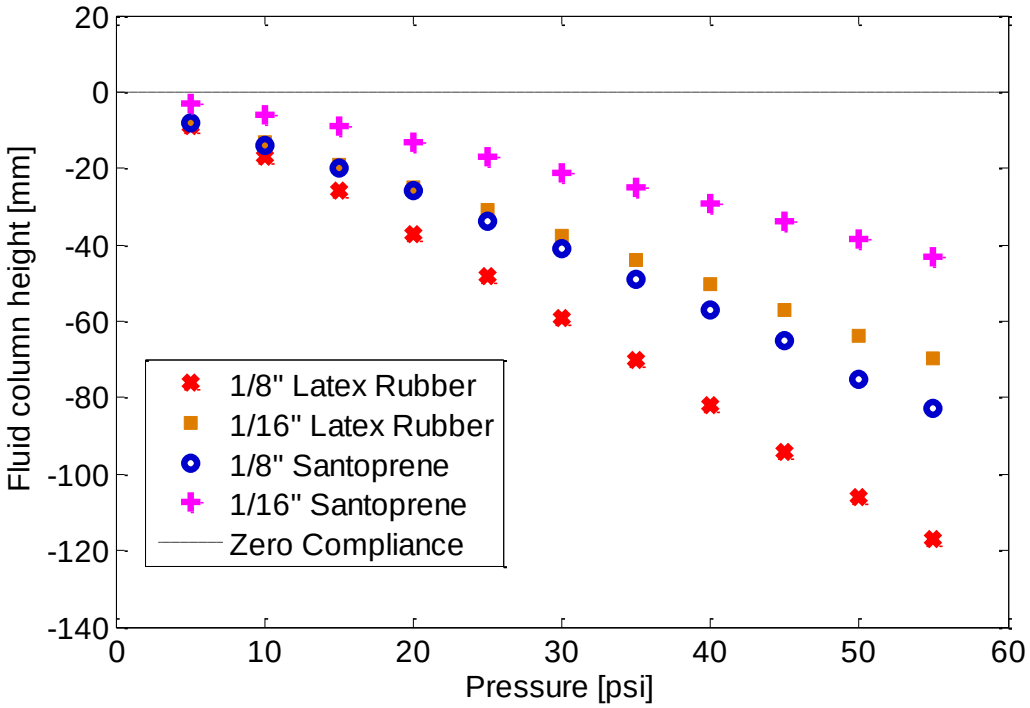

Figure 3-18. Fluid column height vs. pressure for various bladders.

Figure 3-18 shows a plot of fluid column height versus fluid pressure. A steeper slope indicates higher compliance. For both the Latex Rubber and Santoprene, the thinner bladder exhibits less compliance than the thicker versions. Furthermore, the stiffer material (Santoprene) shows lower compliance than the softer material.

Figure 3-19 shows a compliance comparison between bladders of the same thickness but different materials, as well as for the 0.79 mm (1/32 in) thick Latex Rubber bladder. The plot shows the same trend as in Figure 3-18: stiffer materials produce less compliant F^2MC tubes. This trend is summarized in Figure 3-20 and in Table 3-3.

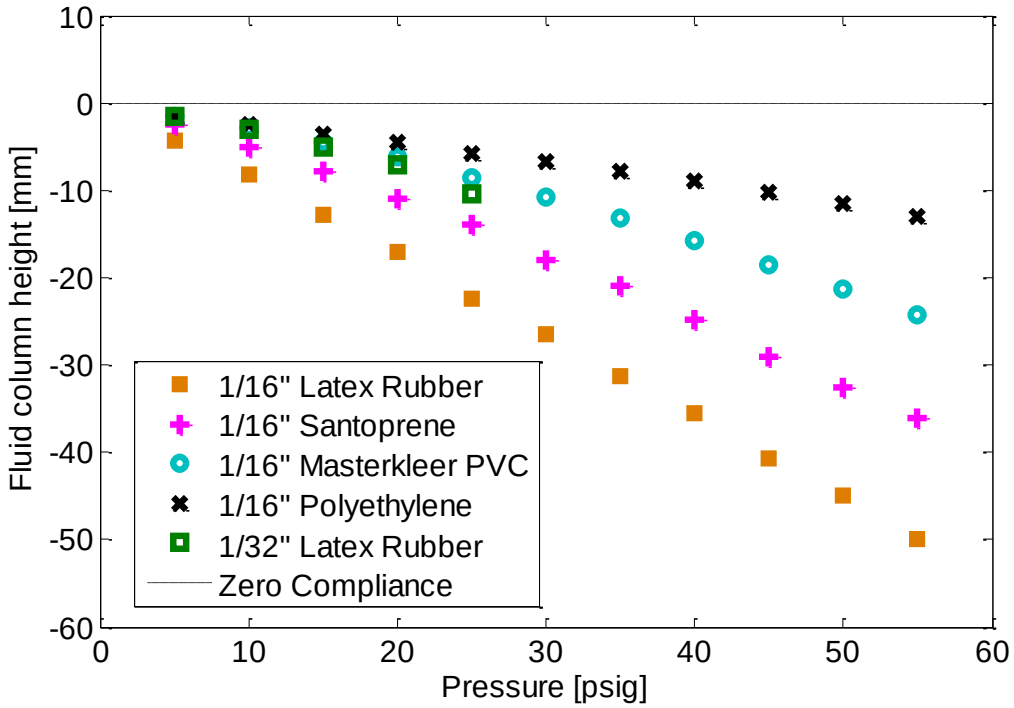

Figure 3-19. Experimental measurements of fluid column height vs. pressure for different bladder materials.

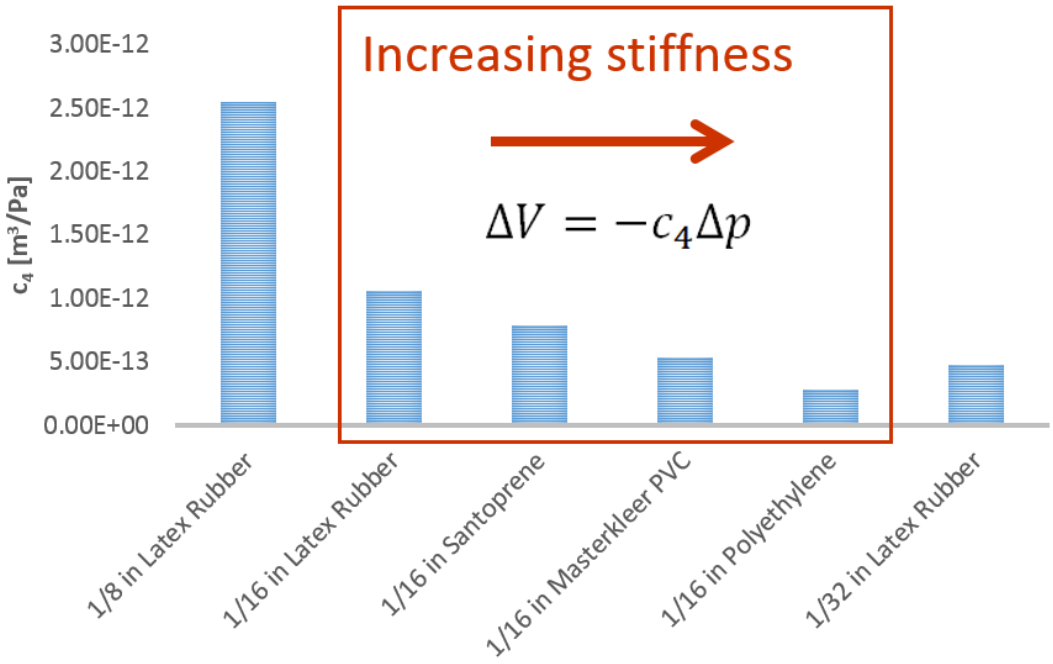

Figure 3-20. Comparison of compliance for different bladders.

Table 3-3. Summary of benchtop pumping and compliance test results.

Bladder	Pumping $[m^3/m]$	Compliance $[m^3/Pa]$
1.59 mm (1/16 in) Latex Rubber	1.62×10^{-3}	10.55×10^{-13}
1.59 mm (1/16 in) Santoprene	1.28×10^{-3}	7.89×10^{-13}
1.59 mm (1/16 in) Masterkleer PVC	1.22×10^{-3}	5.31×10^{-13}
1.59 mm (1/16 in) Polyethylene	0.23×10^{-3}	2.81×10^{-13}
0.79 mm (1/32 in) Latex Rubber	1.68×10^{-3}	4.75×10^{-13}

Figure 3-18 shows that thinner bladders cause less tube compliance than do thicker bladders. The results from Figure 3-16 show that bladders with lower elastic modulus have a more favorable pumping coefficient, while Table 3-3 shows that lower modulus also causes higher compliance.

3.3.4 Simulations Using Benchtop Test Results

To investigate the tradeoff between pumping and compliance discussed in Section 3.3.3, c_3 and c_4 measurements for each F^{2}MC tube are inputted to the model and the simulation results are compared. All frequency-domain simulations presented in this dissertation are performed using the MATLAB command bode.m. An inertia track inner diameter of 3.86 mm (0.152 in) is used in all simulations, but the inertia track length is adjusted for each material to ensure that the absorber is properly tuned. Frequency response plots for 1.59 mm (1/16 in) Polyethylene, 1/16 in Masterkleer PVC, 1/16 in Santoprene, 1.59 mm (1/16 in) Latex Rubber, and 0.79 mm (1/32 in) Latex Rubber are shown in Figures 3-21, 3-22, 3-23, 3-24, and 3-25, respectively.

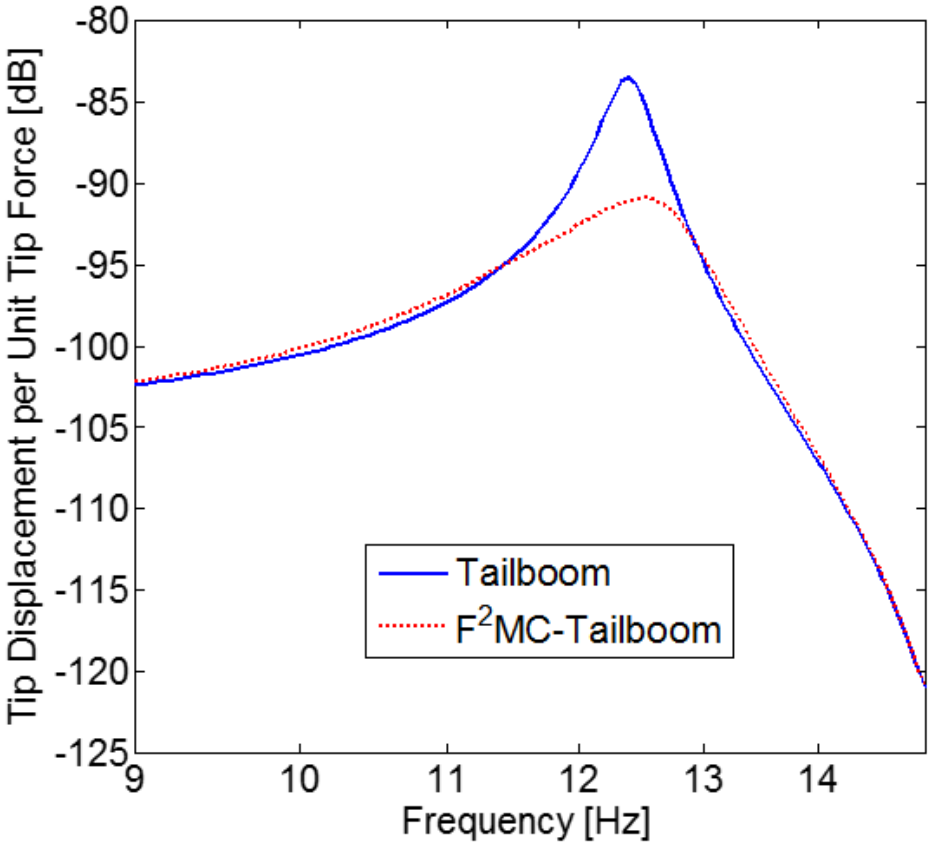

Figure 3-21. Simulated frequency response of a tailboom with F^{2}MC tubes with 1.59 mm (1/16 in) Polyethylene bladder.

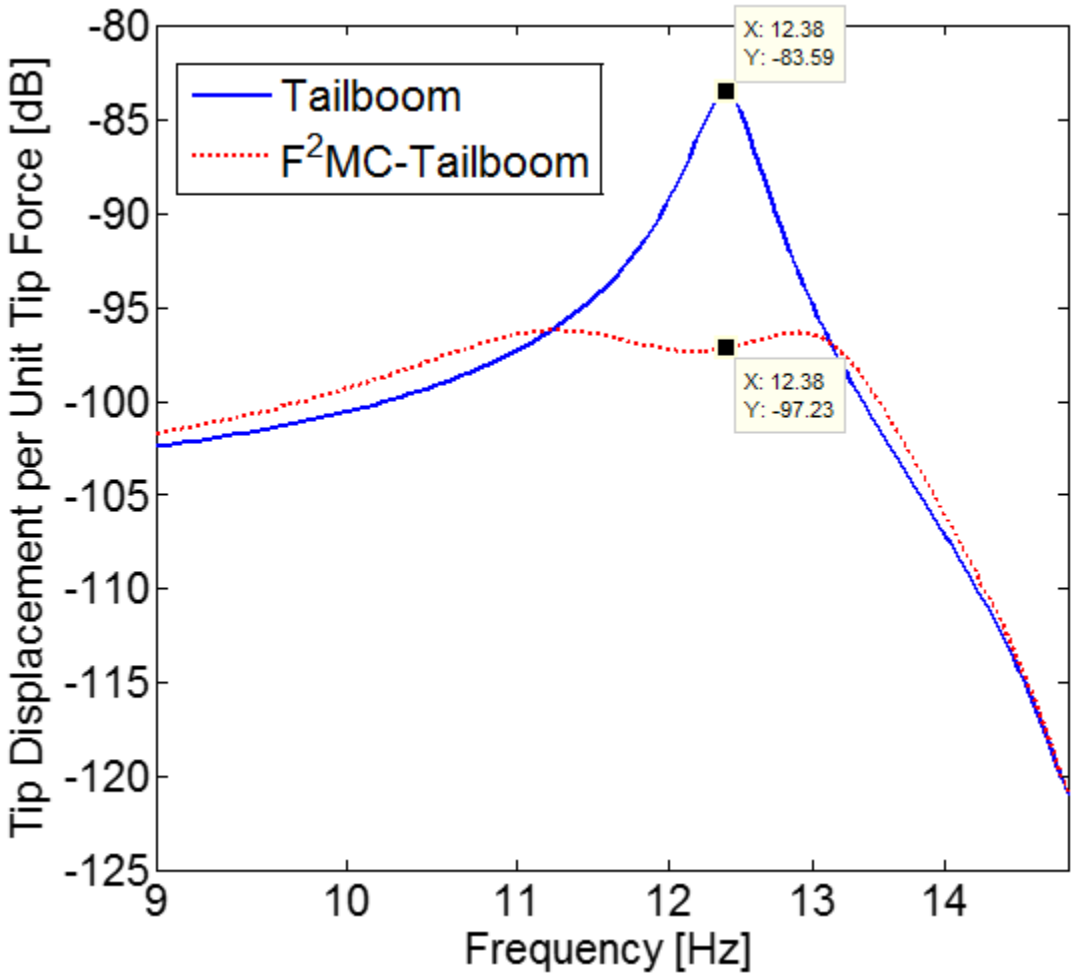

Figure 3-22. Simulated frequency response of a tailboom with F^2MC tubes with 1.59 mm (1/16 in) Masterkleer PVC bladder.

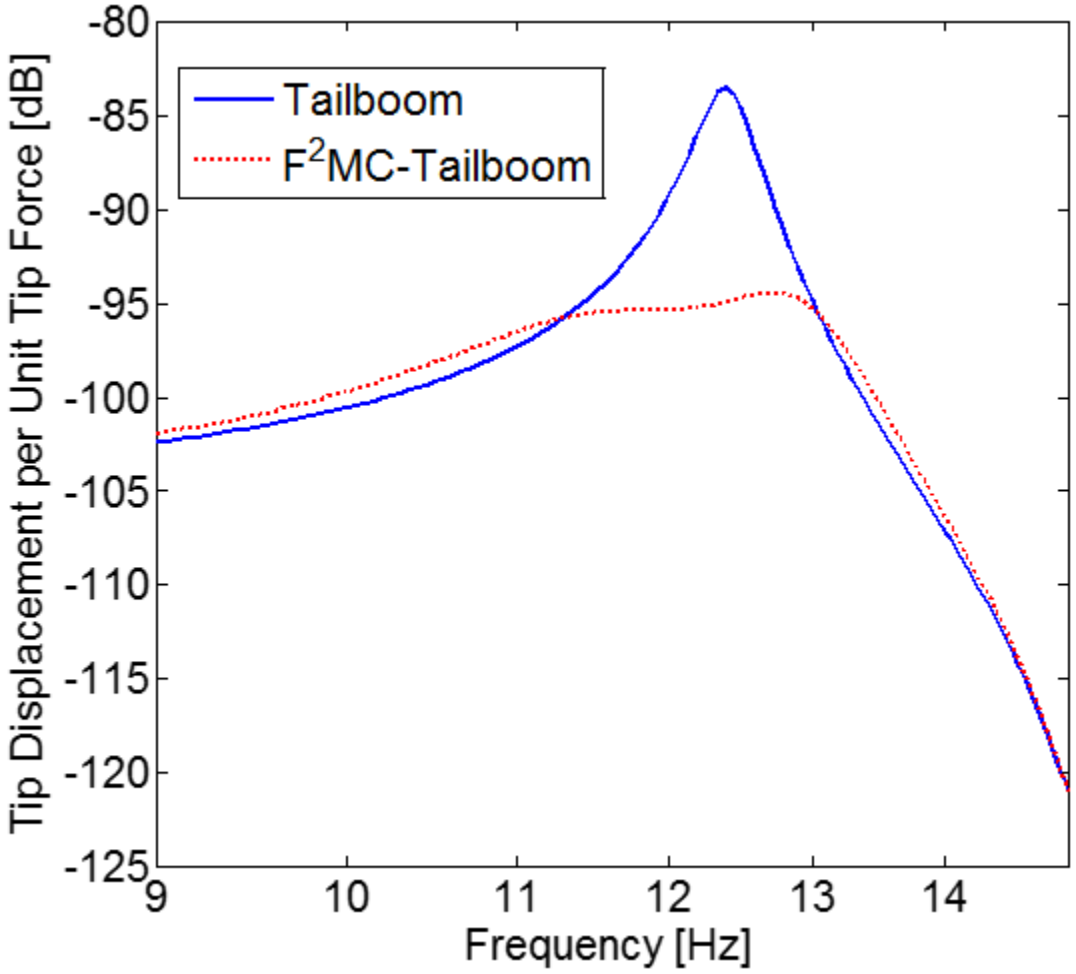

Figure 3-23. Simulated frequency response of a tailboom with F^2MC tubes with 1.59 mm (1/16 in) Santoprene bladder.

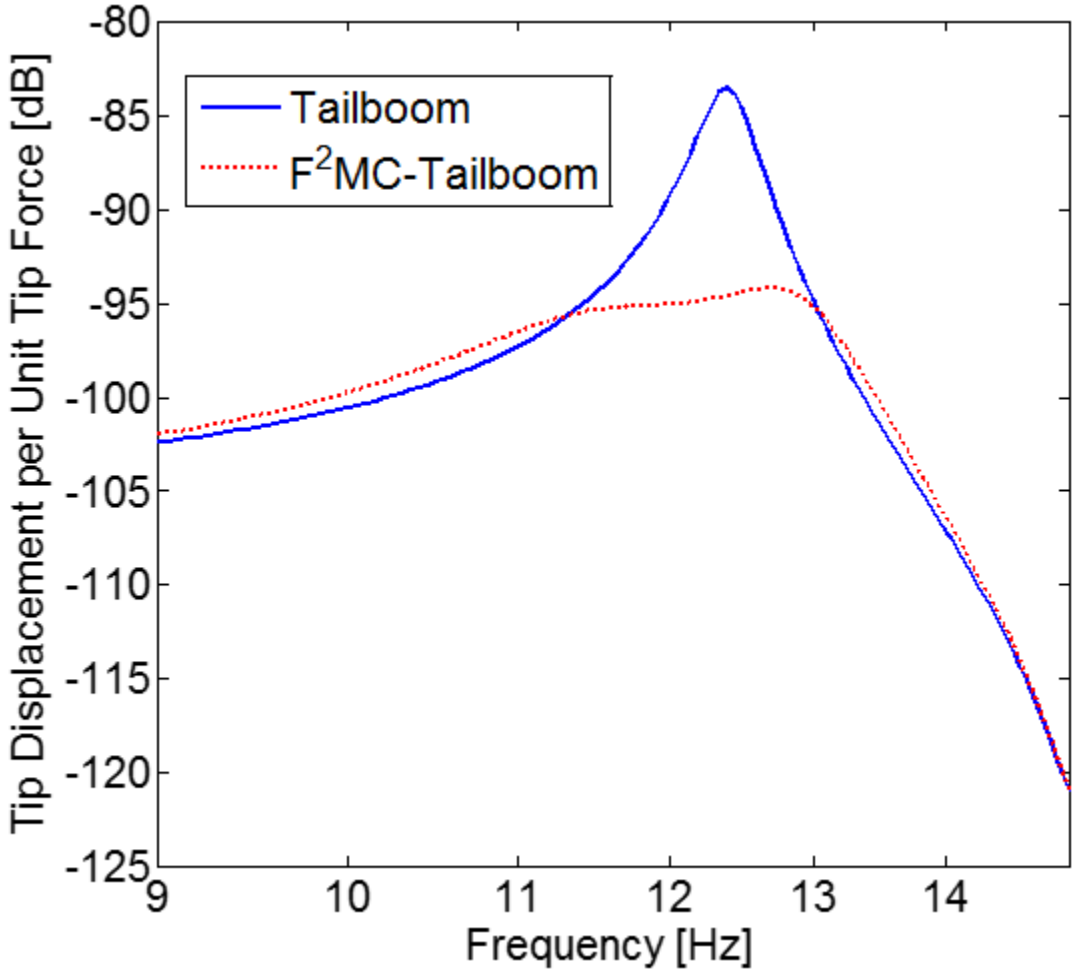

Figure 3-24. Simulated frequency response of a tailboom with F^2MC tubes with 1.59 mm (1/16 in) Latex Rubber bladder.

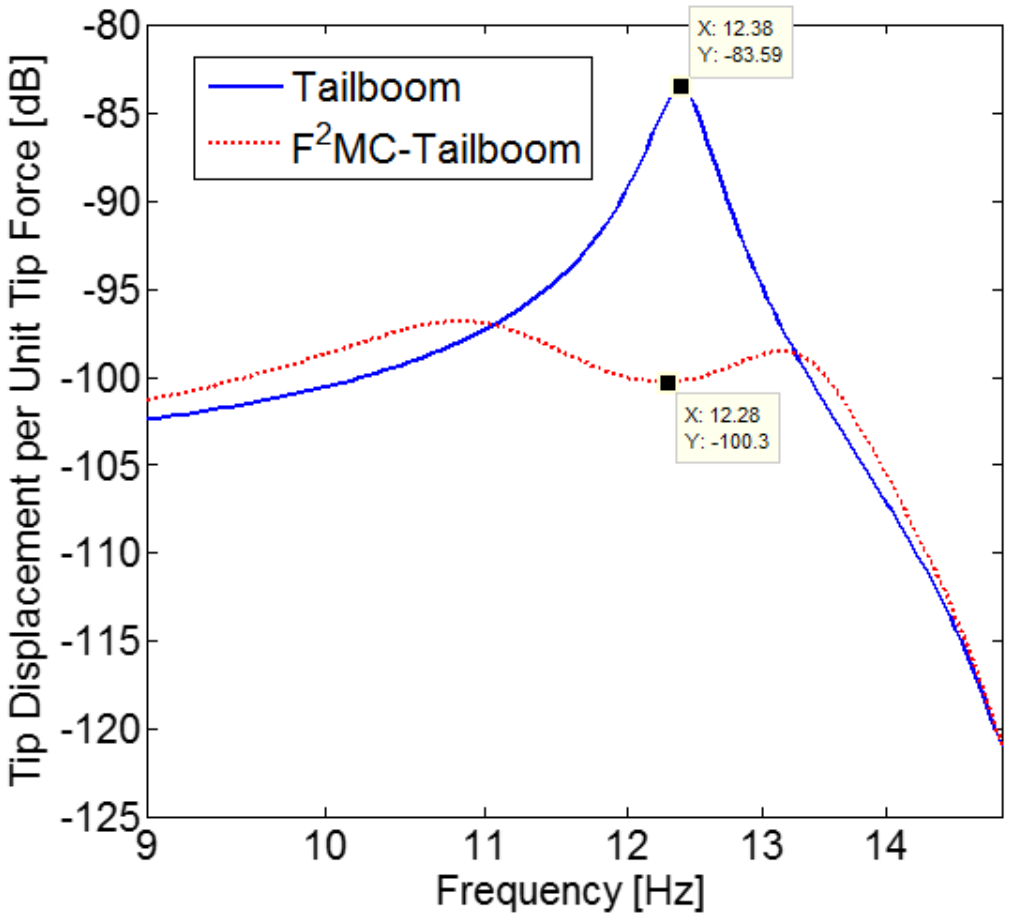

Figure 3-25. Simulated frequency response of a tailboom with F^2MC tubes with 0.79 mm (1/32 in) Latex Rubber bladder.

Based on Figures 3-21 through 3-25, the 0.79 mm (1/32 in) Latex Rubber bladder is clearly superior to the other bladders tested. The absorber using the 0.79 mm (1/32 in) Latex Rubber can produce a visible anti-resonance in the frequency response, whereas the 1.59 mm (1/16 in) Latex Rubber, Santoprene, and Polyethylene bladders fail to do so. Of the bladders tested, the 0.79 mm (1/32 in) Latex Rubber ranks second in compliance and first in pumping coefficient. Polyethylene, which ranked first in compliance but last in pumping, shows the worst overall performance. The Masterkleer bladder, which ranked third in both categories, showed the second best overall performance. Thus, pumping coefficient and compliance are both important determinants of the F^2MC tube's performance.

3.4 Dynamic Model Validation of the Lab-Scale Tailboom Structure

With the knowledge gained from the experiments described in the previous sections, a high-performance absorber is designed for and tested on the lab-scale tailboom structure. Additional validation experiments are conducted to study the effect of pre-pressurization on performance and to test the linearity of the system.

3.4.1 Experimental Demonstration of Absorption and Damping

The vibration absorber uses the parameters listed in Table 3-4, targeting the first vertical tailboom bending mode. New F^2MC tubes are fabricated, pre-pressurized, and pre-tensioned to a nominal center diameter of 11 mm (0.435 in). The coefficients c_3 and c_4 are measured using the same procedure outlined in Section 3.3. Orifice resistance R_{orf} is tuned using the experimental frequency response plot, to approximately match the damping ratios of the absorber peaks, calculated using the half-power method.

Table 3-4. Absorber parameters used for the dynamic tests.

Characteristic	Imperial	Metric
Fluid		
ρ_f	$112\ lb/ft^3$	$1800\ kg/m^3$
μ	$1.3 \times 10^{-7} psi \cdot s$	$9 \times 10^{-4} Pa \cdot s$
- pre-pressure	$100\ psi$	$690\ kPa$
F^2MC tube		
- length	$15\ in$	$0.38\ m$
- diameter	$0.375\ in$	$9.5\ mm$
- weight	$0.17\ lb$	$75.8\ g$
α	$18\ deg$	
c_1	$259\ lb/in$	$45.35\ kN/m$
c_2	$3.41\ in^2$	$0.0022\ m^2$
c_3	$-2.54\ in^2$	$-16.4\ cm^2$
c_4	$4.6 \times 10^{-4} in^5/lb$	$1090\ mm^5/N$
Inertia track		
l_p	$110.2\ in$	$2.88\ m$
$2r_p$	$0.277\ in$	$7\ mm$
R_{orf}	$2930 \frac{lb}{in^4 s}$	$3.19 \times 10^9 \frac{kg}{m^4 s}$
Total weight	$6\ lb$	$2.7\ kg$

A high-density, low-viscosity fluid provided by LORD Corporation is used in this experiment. A dense fluid with minimal viscosity is desirable because denser fluids provide higher fluid inertance, and lower viscosity results in more efficient fluid pumping and therefore a deeper anti-resonance.

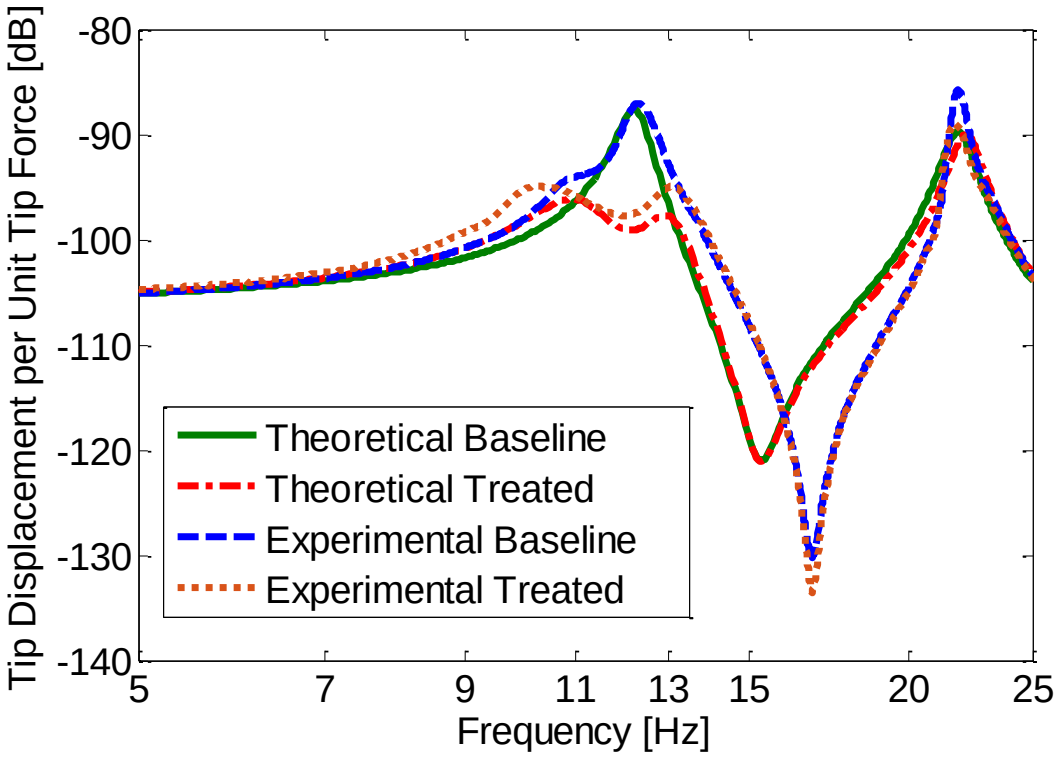

Figure 3-26. Theoretical and experimental frequency response of the baseline and absorber-treated tailboom.

Figure 3-26 shows the theoretical and experimental frequency responses from tip force to tip displacement of the untreated baseline tailboom and of the tailboom with F^2MC tubes. The theoretical and experimental frequency responses of the baseline structure show good correlation; the correct static behavior is predicted, and the first two natural frequencies match. The Open Valve curves represent the tailboom response with the absorber orifice completely open (i.e. $R_{orf} = 0$). Based on pre-test predictions, the inertia track is tuned such that the absorption frequency matches the tailboom's first natural frequency. The absorber reduces vibration amplitude at the first mode by 10.6 dB (70.5%). The model predicts a reduction of 13.1 dB (77.8%).

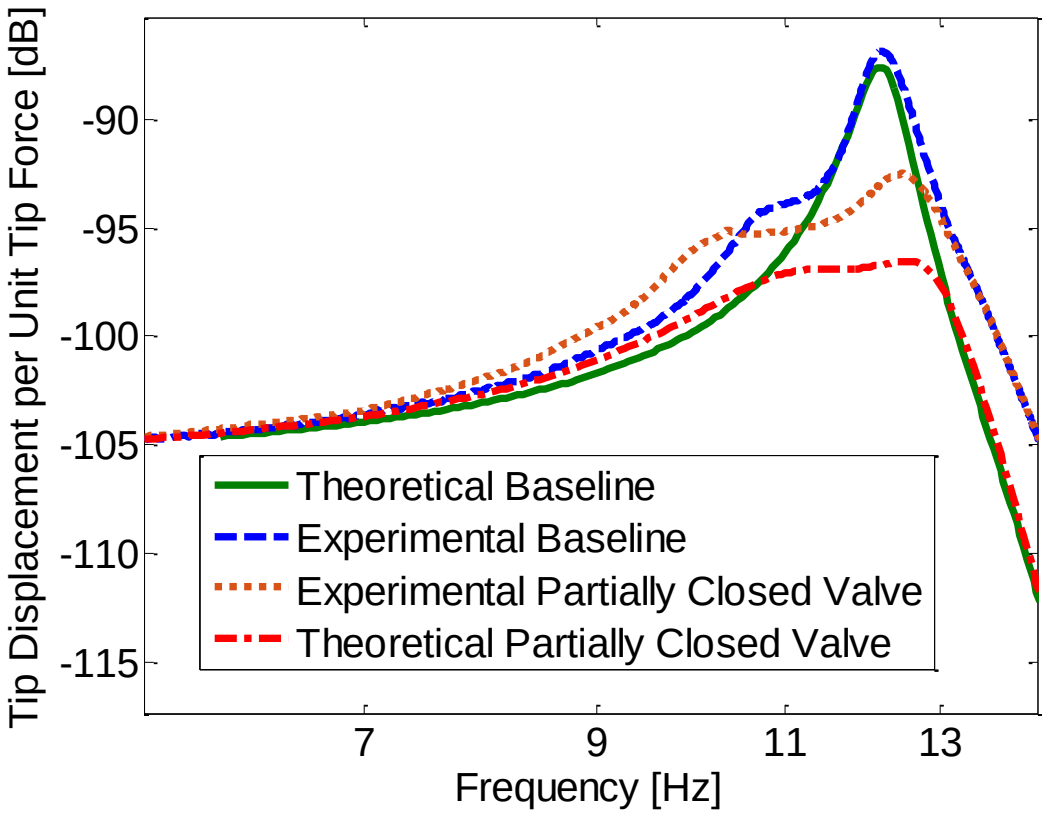

Figure 3-27. Experimental frequency response of the baseline and damped absorber-treated tailboom.

As shown in Figure 3-27, partially closing the orifice eliminates the two flanking resonant peaks of the absorber, producing a damped absorber. The experimental and theoretical damped absorbers provide a damping ratio of 10.8%, a 7.8% improvement over that of the baseline structure. Model predictions indicate that an optimally tuned orifice with $R_{orf} = 3.05 \times \frac{10^7 kg}{m^4 s}$ can add up to 11% damping to the first mode. This configuration can be useful against impulsive, broadband, or time-varying harmonic inputs that may be amplified by the flanking resonant peaks of an absorber.

Finite element analysis of the lab-scale tailboom structure indicates the presence of a torsional mode around 11.7 Hz. The shaker may not be aligned perfectly, thereby exciting the

torsional and lateral bending modes. Furthermore, due to spatial constraints the laser vibrometer is not pointed along the centerline of the tailboom. Thus, unmodeled tailboom dynamics, particularly from the resonance observed at 10.5 Hz, may explain the discrepancy in vibration amplitude between the experimental and theoretical absorber frequency responses.

The total added weight due to the absorber and fluid is 2.7 kg (7 lb), which is below the initial 4.55 kg (10 lb) weight target for 2% damping added set by the sponsor. The F^2MC-absorber adds 7.8% damping at the first mode, or 2.9% per kg, over 6.5 times the sponsor goal. The experimentally measured absorber and damped absorber performances are summarized along with model predictions in Table 3-5. The results shown indicate that F^2MC-based absorbers can significantly reduce vibration in realistic aerospace structures for a low weight penalty.

Table 3-5. Summary of absorber and damped absorber performance.

Characteristic	Value
Amplitude Reduction	
- Theory	77.8%
- Experiment	70.5%
Damping Added	
- Theory	7.8%
- Experiment	7.8%
Theoretical Max Damping Added	11%

3.4.2 Pre-Pressurization and Linearity Study

To study the effect of pre-pressurization on absorber performance, experiments are run for various nominal fluid pressures. The influence of fluid pre-pressure on F^2MC tube performance is shown in Figure 3-28. Increasing the pre-pressure applied to the fluid results in a deeper absorber valley and lower resonance peak. Applying pre-pressure to the fluid helps

collapse air bubbles that increase fluid compliance and reduce performance. Furthermore, pre-pressurization improves the mechanical engagement between the F^2MC fibers and bladder. This coupling is needed to ensure that axial displacement in the F^2MC tubes is converted into fluid pumping. The marginal benefit of further pressurization decreases with increasing pressure.

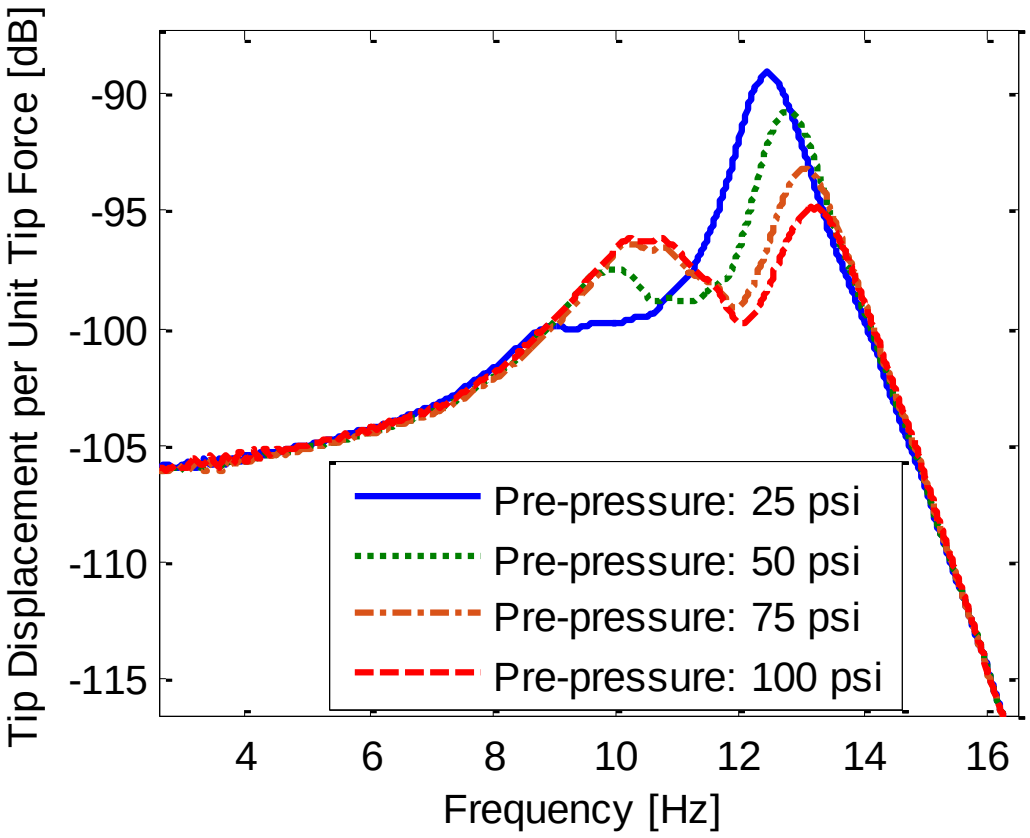

Figure 3-28. Experimental frequency response various fluid pre-pressures.

The effect of tailboom forcing amplitude is illustrated in Figures 3-29 and 3-30. In a perfectly linear system, the frequency response from tip force to tip displacement should be independent of forcing amplitude. However, increasing shaker forcing amplitude causes a decrease in the resonance frequencies of the structure. Consequently, at higher forcing amplitudes

the absorber is not tuned to the correct resonance frequency, and performance declines. Thus, the effect of forcing amplitude must be considered when designing an absorber.

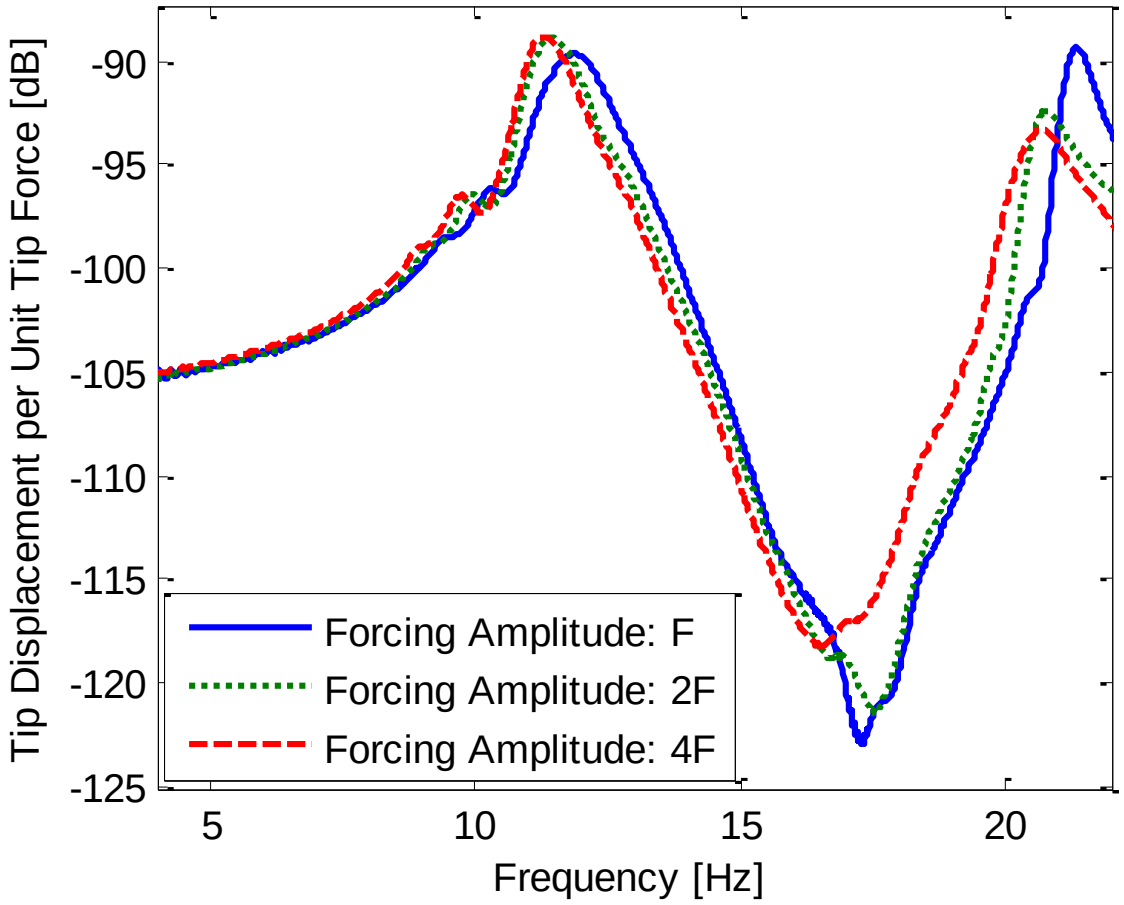

Figure 3-29. Experimental frequency response of the baseline tailboom for various forcing amplitudes.

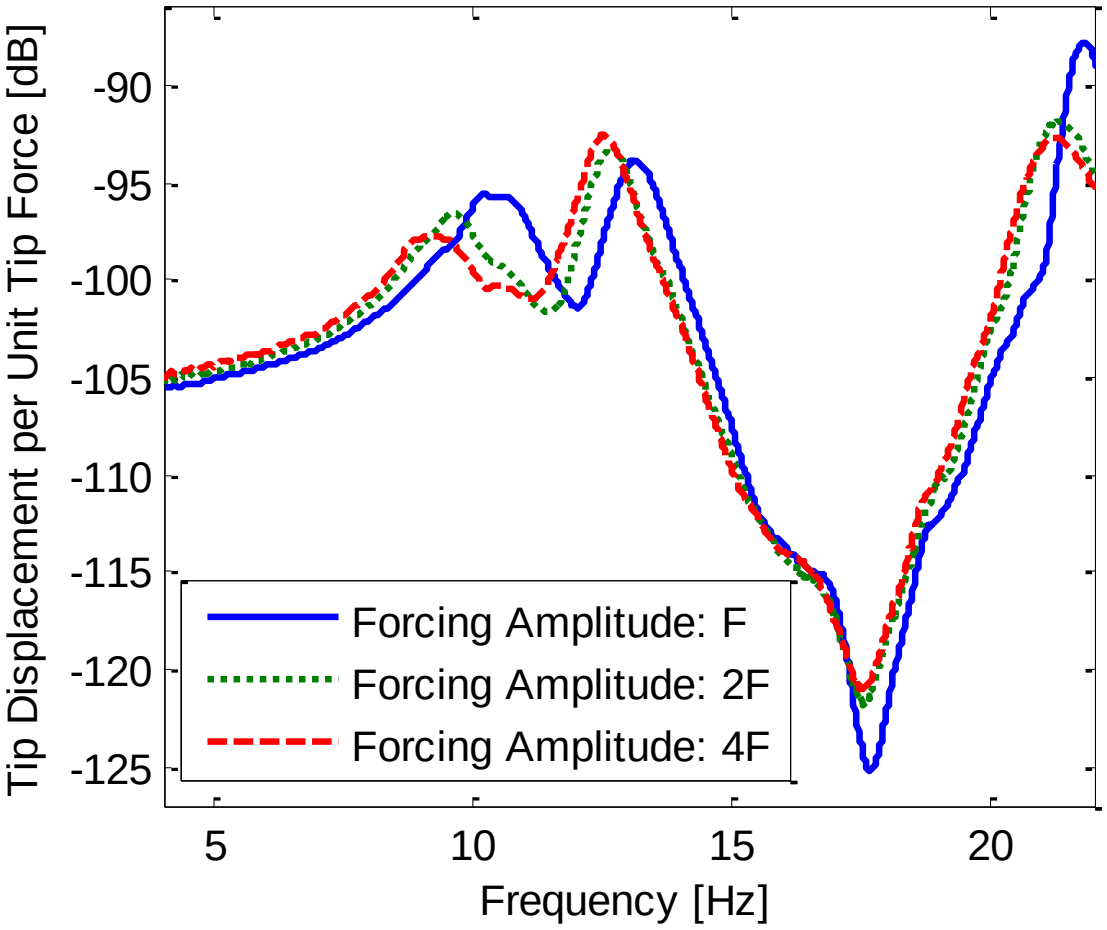

Figure 3-30. Experimental frequency response of the treated tailboom for various forcing amplitudes.

Chapter 4

F^2MC-Absorber Design Process

A general design process for F^2MC-based vibration treatments is developed and tested. A fluidic circuit with two parallel inertia tracks is proposed and analyzed to enable an absorber to be tuned to multiple frequencies. Finally, the performance per added weight of the F^2MC-based absorber is compared against an equivalent piston-based pumper and an active control scheme.

4.1 F^2MC-Absorber Design Process

To build an effective absorber, appropriate F^2MC tube parameters must be selected. These include the length and attachment location of the tubes, the tube diameter, fiber winding angle, inertia track dimensions, working fluid, and bladder material and thickness. Various design considerations are discussed and then summarized in a design process.

4.1.1 F^2MC Tube Placement

The optimal attachment location of the F^2MC tubes depends on the vibration frequency targeted and the response shape of the underlying structure at the target frequency. Integrating Eq. (2.5.3) reveals that F^2MC tube axial displacement, and therefore fluid pumping, is maximized when the slope difference between the attachment points is maximized. Through a parametric study for a simple cantilever beam with F^2MC tubes, Zhu et al. confirm that designs with F^2MC tubes attached in high-slope-difference and thus high-strain regions of the beam outperform others [42]. As shown in Figure 4-1, the theoretical first full-scale tailboom mode shape slope monotonically increases from the root to the tip. From this plot, it is clear that maximum strain is

achieved by attaching F^{2}MC tubes at the tailboom root and tip. A weight-efficient solution may be obtained by attaching F^{2}MC tubes at the root and at approximately 30% station.

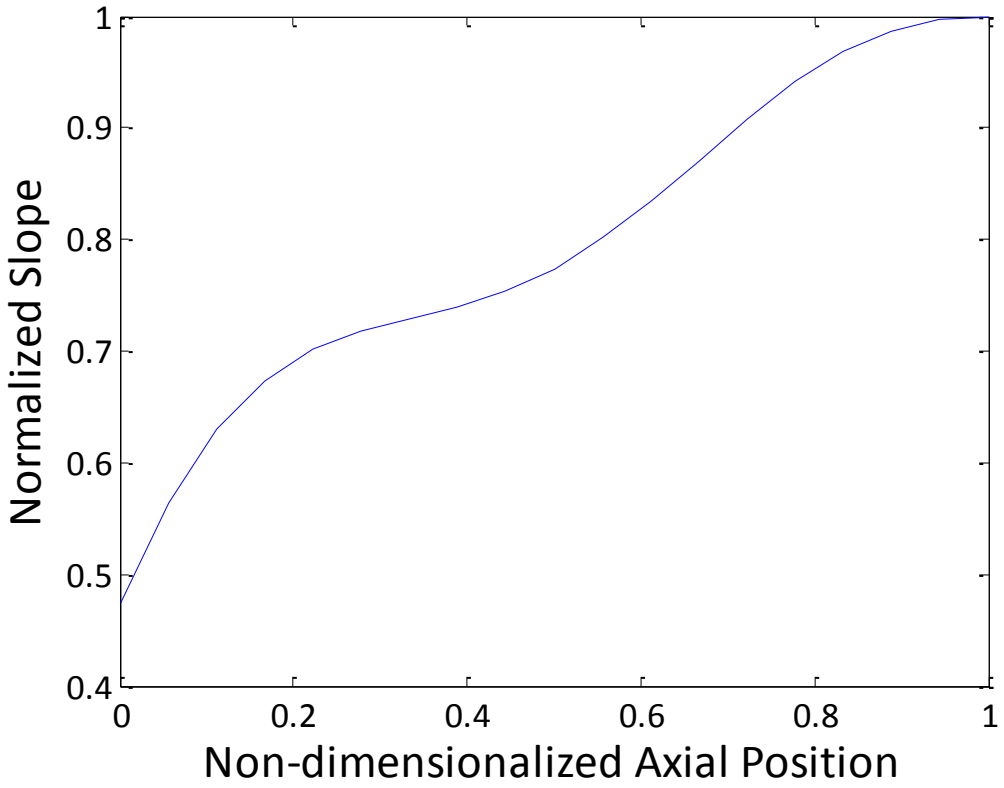

Figure 4-1. Slope of first vertical bending mode shape of the full-scale tailboom.

4.1.2 F^{2}MC Tube Sizing and Number

As discussed in the previous section, F^{2}MC tube length is determined largely by the response shape of the structure at the target frequency. The tube diameter governs the control authority of the vibration control treatment. Larger diameter tubes pump more fluid and produce more force for a given pressure, magnifying the vibration damping effect.

Using multiple F^{2}MC tubes has a similar effect to increasing the tube cross-sectional area. More or higher diameter tubes reduce the required inertance. A simplified F^{2}MC tube pumping model is shown in Figure 4-2.

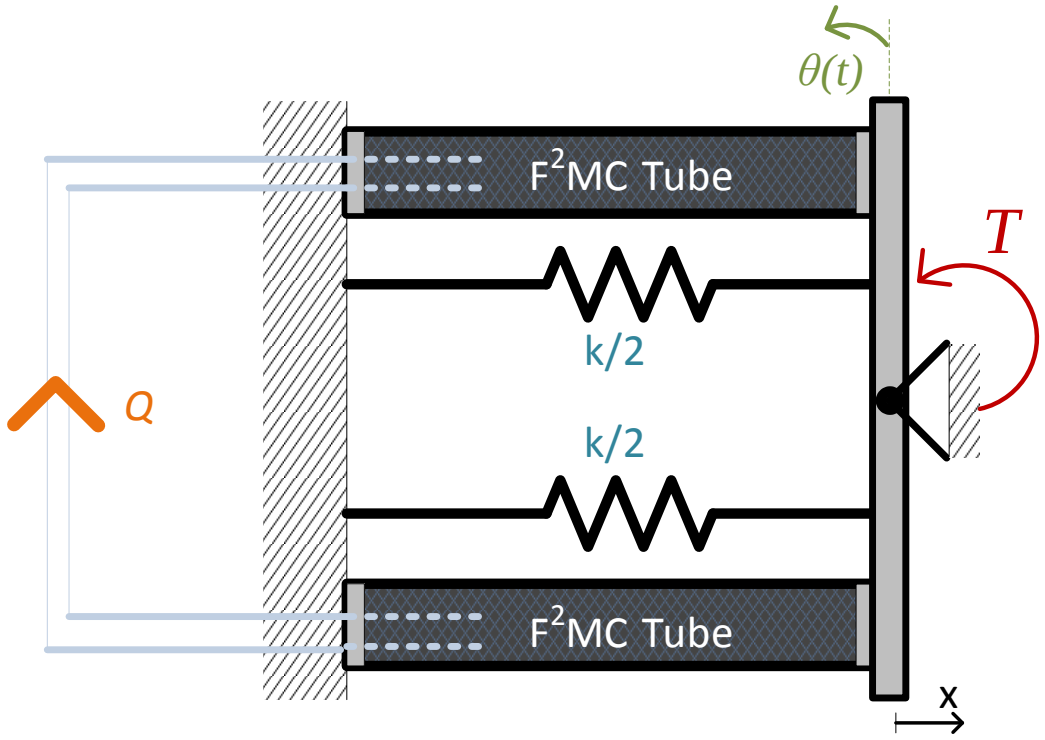

Figure 4-2. Simplified F^{2}MC tube pumping model.

The cantilevered tailboom is simulated as two parallel springs anchored to the wall and attached to a rigid rod that rotates with angle θ about its center at a fixed hinge. An external torque T is applied to the rod. The n_p pairs of F^{2}MC tubes on the top and bottom are interconnected via an inertia track that allows volumetric flow rate Q. number of F^{2}MC tube pairs affects the tuning of the inertia track. The equation of motion for the system in Figure 4-2 is obtained via the moment balance

$$0 = -2n_p dF_t - 2d\left(\frac{k}{2} d \sin\theta\right), \tag{4.1}$$

where d is the distance between the hinge and the F^{2}MC tube axis, k is the spring constant, and F_t is the F^{2}MC tube axial force given by Eq. (2.43). Substituting Eq. (2.43) into Eq. (4.1), solving for pressure p, and using the small-angle approximation gives

$$p = \frac{1}{2n_p d c_2} T - \frac{d}{c_2}\left(c_1 + \frac{k}{2n_p}\right)\theta. \tag{4.2}$$

Substitute Eq. (4.2) into Eq. (2.44) and differentiate with respect to time to obtain

$$\dot{Q} = \left[\frac{d n_p c_4}{c_2}\left(c_1 + \frac{k}{2n_p}\right) - n_p c_3 d\right]\ddot{\theta} - \frac{c_4}{2dc_2}\ddot{T} \tag{4.3}$$

For simplicity, ignore the effects of fluid resistance and substitute Eq. (4.3) into Eq. (2.40):

$$I_f n_p\left[\frac{c_4}{c_2}\left(c_1 + \frac{k}{2n_p}\right) - c_3\right]\ddot{\theta} + \frac{2}{c_2}\left(c_1 + \frac{k}{2n_p}\right)\theta = \frac{c_4}{2n_p d^2 c_2}\left(I_f \ddot{T} + \frac{2}{c_4} T\right). \tag{4.4}$$

Eq. (4.4) can be transformed into the Laplace domain to derive the transfer function

$$\frac{\theta(s)}{T(s)} = \frac{\mathrm{c}_4}{2d^2 c_2} \frac{I_f s^2 + \dfrac{2}{c_4 n_p}}{I_f n_p\left[\dfrac{c_4}{c_2}\left(c_1 + \dfrac{k}{2n_p}\right) - c_3\right] s^2 + \dfrac{2}{c_2}\left(c_1 + \dfrac{k}{2n_p}\right)}, \tag{4.5}$$

where $\theta(s) = \mathcal{L}[\theta(t)]$ and $T(s) = \mathcal{L}[T(t)]$. To calculate the vibration response at the tailboom natural frequency, solve Eq. (4.5) for $s = j\sqrt{k/M}$:

$$\left|\frac{\theta}{T}\left(j\sqrt{\frac{k}{M}}\right)\right| = \frac{c_4 I_f k n_p - 2M}{d^2\left[c_4 I_f k^2 n_p - 4c_1 M n_p - 2k\left(M + (c_2 c_3 - c_1 c_4) I_f n_p^2\right)\right]}, \tag{4.6}$$

where M is the mass of the tailboom and $\sqrt{k/M}$ is the natural frequency. Setting the numerator of Eq. (4.6) to zero gives the approximate inertance required to minimize vibration in the structure at resonance:

$$I_f^* = \frac{M}{k}\frac{2}{c_4 n_p} = \frac{1}{\omega_n^2}\frac{2}{c_4 n_p}, \tag{4.7}$$

where ω_n is the natural frequency of the tailboom. The inertance requirement of a tuned absorber is inversely proportional to n_p. For instance, doubling the number of F^{2}MC tubes has the same effect as doubling the absorber inertia. Thus, using Eq. (4.7) one can evaluate the tradeoff between the number of F^{2}MC tubes and the inertia track size. A method for quantitatively sizing F^{2}MC tubes is discussed in Section 4.1.5.

All else being equal, a configuration using multiple F^{2}MC tubes is heavier compared to one using a single large tube since the total surface area of the F^{2}MC tubes is greater. Furthermore, the additional F^{2}MC tube end fittings and plumbing adds parasitic weight. However, if mounted inside the structure, the smaller-diameter tubes have a longer moment arm d compared to a single tube.

4.1.4 F^{2}MC Tube Bladder Design and Fiber Winding Angle

As discussed in Section 3.3, the properties of the bladder inside the F^{2}MC tube is a major determinant of F^{2}MC tube performance. Constants c_3 and c_4 from Eq. (2.44) are of particular interest when selecting a bladder. Figures 3-21 through 3-25 suggest that performance is maximized when pumping coefficient c_3 is maximized. Simultaneously, compliance c_4 must be minimized [54]. Both of these parameters are determined partially by the bladder material and thickness used in the F^{2}MC tube.

As summarized in Table 3-3, stiffer bladders offer lower compliance at the expense of reduced pumping. The lower-compliance benefit of thinner bladders can be seen in Figure 3-18; this reduced compliance is due to the smaller volume of bladder material compression [48]. Thicker bladders have the added disadvantage of reducing the volume of fluid inside of the F^{2}MC tube [58]. On the other hand, softer bladders expand and contract readily with the fiber reinforcement, enabling better pumping. Thin-walled bladders offer lower compliance without

sacrificing pumping, but may be less resistant to wear and leaking. Thus, the pumping and compliance coefficients of numerous candidate bladder materials should be measured. If a model of the system of interest is available, then the best-performing bladder can be determined through simulation. The results of Section 3.3.3 indicate that a thin and flexible bladder like the 0.79 mm (1/32 in) latex rubber provides the best results of the bladders tested.

Although tube compliance generally reduces absorber performance, in some cases added compliance may be desirable. As can be seen in Eq. (4.7), the inertance required to tune an absorber to a given frequency ω_n is inversely proportional to compliance c_4. A low-compliance system, while resulting in high performance, requires a large matching inertia, which requires a large, heavy inertia track. If a higher-compliance system is deemed to provide acceptable performance, then it may be preferable to a heavier system that provides more vibration reduction than is needed. Thus, in practice a tradeoff must be made between compliance-related performance and inertia track weight.

While having a flexible bladder that conforms to the F^2MC fiber mesh is important, it is equally important that the fibers are wound at a low angle. Zhu et al. compute the pumping factor of the F^2MC tube as a function of fiber winding angle [46]. Pumping factor is the ratio of the volume change of a F^2MC tube to that of a piston-cylinder of the same diameter. For filament-wound tubes, pumping factor is maximized at small fiber winding angles, between approximately $5°$ and $20°$ depending on the thickness and material properties of the tube. The computation of c_3 for braid-sheathed tubes is based purely on the geometry of the fibers. According to this model, pumping $-c_3$ is maximized as the winding angle approaches zero (Figure 4-3). In this research, tubes with winding angles around 20 degrees are used because braided sleeves with that winding angle are readily available commercially.

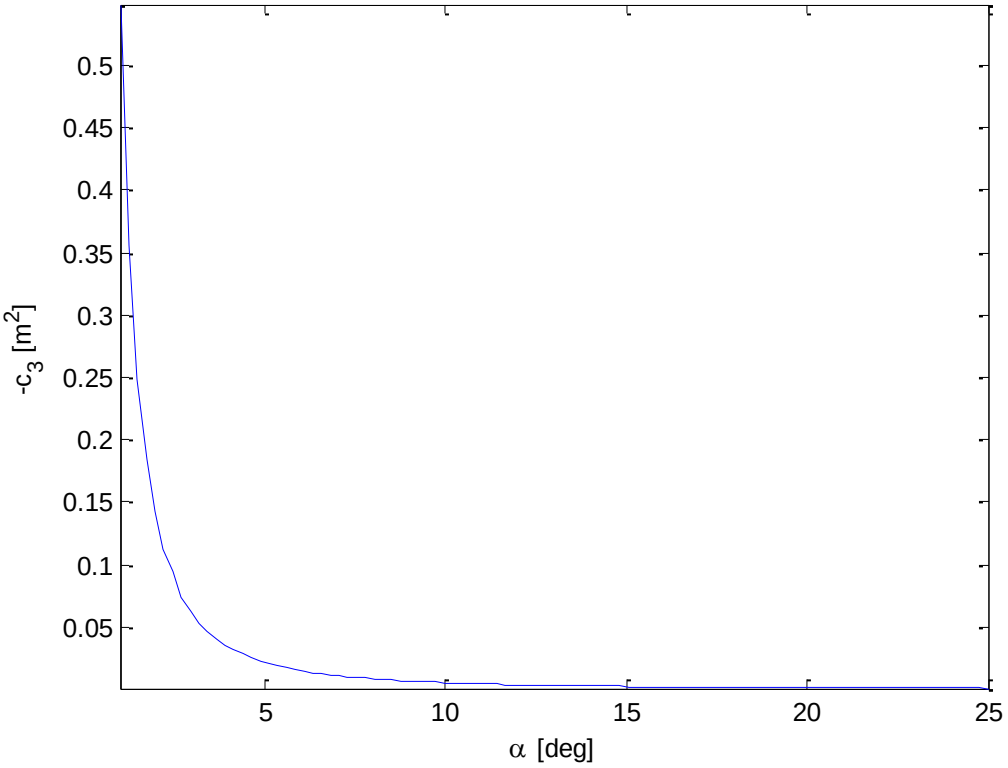

Figure 4-3. Pumping coefficient c_3 as a function of fiber winding angle α.

4.1.5 Inertia Track Dimensions, Orifice, and Fluid

The inertia track of the damped absorber must be tuned to a specific frequency. The simple approximation in Eq. (4.7) provides some insight into the relationship between tuning frequency, compliance, and inertance. The inertance tuning equation for any frequency in general is obtained by evaluating Eq. (4.5) for $s = j\omega$:

$$I_f^* = \frac{2}{\omega^2 c_4 n_p}. \tag{4.8}$$

The required inertance is inversely proportional to c_4. That is, stiffer systems (smaller values of c_4) require a larger fluid inertia. Due to the ω^2 term in the denominator, tuning to a lower frequency requires significantly more inertia.

Since Eq. (4.8) neglects losses in the fluid, it should be used only if the objective is to minimize the open-valve response amplitude at a specific frequency. If an orifice is used to moderate the absorber resonance and anti-resonances, then tuning rules accounting for fluid resistance must be used. To derive such tuning laws, the governing equations in Eqs. (2.34) and (2.45) are simplified. Recall that the tailboom differential equation Eq. (2.5) is discretized using the Rayleigh-Ritz method. Taking $N = 1$ in the Ritz series in Eq. (2.1), as well as substituting in the forcing term Eq. (2.57), the following scalar tailboom equation is obtained:

$$M\ddot{q} + C\dot{q} + Kq = \psi_L F_y - 2n_p F_t d_{21}\psi'_{21}. \tag{4.9}$$

To combine Eq. (4.9) with the fluidic system, the numerator and denominator of the transfer function in Eq. (2.45) are separated and transformed into the time domain such that

$$F_t = N_2\ddot{z}_t + N_1\dot{z}_t + N_0 z_t \tag{4.10}$$

and

$$I_f\ddot{z}_t + R\dot{z}_t + D_0 z_t - d_t q = 0. \tag{4.11}$$

Eqs. (4.10) and (4.11) can be combined to eliminate $\ddot{z}$ from Eq. (4.10). This is necessary to make the mass, stiffness, and damping matrices symmetric:

$$F_t = \left(N_0 - \frac{N_2 D_0}{I_f}\right) z_t + \left(N_1 - \frac{N_2 R}{I_f}\right)\dot{z}_t + \frac{N_2}{I_f} d_t q = \left(N_0 - \frac{N_2 D_0}{I_f}\right) z_t + \frac{N_2}{I_f} d_t q. \tag{4.12}$$

Substitute Eq. (4.12) into Eq. (4.9) to obtain

$$M\ddot{q} + C\dot{q} + \left(K + \frac{2n_p N_2}{I_f} d_{21}\psi'_{21} d_t\right) q + 2n_p d_{21}\psi'_{21}\left(N_0 - \frac{N_2 D_0}{I_f}\right) z_t = \psi_L F_y. \tag{4.13}$$

Substituting Eqs. (2.46), (2.48), and (2.49) into Eq. (4.13), and combining the result with Eq. (4.11) in matrix form, gives

$$\begin{bmatrix} M & 0 \\ 0 & I_f \end{bmatrix}\begin{bmatrix} \ddot{q} \\ \ddot{z}_t \end{bmatrix} + \begin{bmatrix} C & 0 \\ 0 & R \end{bmatrix}\begin{bmatrix} \dot{q} \\ \dot{z}_t \end{bmatrix} + \begin{bmatrix} K + 2n_p\left(c_1 - \dfrac{c_2c_3}{c_4}\right)d_{21}\psi'_{21}d_t & \dfrac{4c_2c_3}{c_4^2}d_{21}\psi'_{21} \\ -d_t & \dfrac{2}{n_pc_4} \end{bmatrix}\begin{bmatrix} q \\ z_t \end{bmatrix} = \begin{bmatrix} \psi_L \\ 0 \end{bmatrix}F_y. \tag{4.14}$$

Define a new state variable z based on the following transformation:

$$\begin{bmatrix} q \\ z_t \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & \dfrac{n_pc_4d_t}{2} \end{bmatrix}\begin{bmatrix} q \\ z \end{bmatrix}. \tag{4.15}$$

Substitute Eq. (4.15) into Eq. (4.14) to obtain

$$\begin{bmatrix} M & 0 \\ 0 & \dfrac{n_pc_4I_fd_t}{2} \end{bmatrix}\begin{bmatrix} \ddot{q} \\ \ddot{z} \end{bmatrix} + \begin{bmatrix} C & 0 \\ 0 & \dfrac{n_pc_4Rd_t}{2} \end{bmatrix}\begin{bmatrix} \dot{q} \\ \dot{z} \end{bmatrix} + \begin{bmatrix} K + 2n_p\left(c_1 - \dfrac{c_2c_3}{c_4}\right)d_{21}\psi'_{21}d_t & 2n_p\dfrac{c_2c_3}{c_4}d_td_{21}\psi'_{21} \\ -d_t & d_t \end{bmatrix}\begin{bmatrix} q \\ z \end{bmatrix} = \begin{bmatrix} \psi_L \\ 0 \end{bmatrix}F_y. \tag{4.16}$$

Finally, premultiply Eq. (4.16) by $\begin{bmatrix} 1 & 0 \\ 0 & -2n_p\frac{c_2c_3}{c_4}d_{21}\psi'_{21} \end{bmatrix}$ to ensure that the stiffness matrix is symmetric:

$$\begin{bmatrix} M & 0 \\ 0 & -n_p^2I_fc_2c_3d_{21}\psi'_{21}d_t \end{bmatrix}\begin{bmatrix} \ddot{q} \\ \ddot{z} \end{bmatrix} + \begin{bmatrix} C & 0 \\ 0 & -n_p^2Rc_2c_3d_{21}\psi'_{21}d_t \end{bmatrix}\begin{bmatrix} \dot{q} \\ \dot{z} \end{bmatrix} + \begin{bmatrix} K + 2n_p\left(c_1 - \dfrac{c_2c_3}{c_4}\right)d_td_{21}\psi'_{21} & 2n_p\dfrac{c_2c_3}{c_4}d_td_{21}\psi'_{21} \\ 2n_p\dfrac{c_2c_3}{c_4}d_td_{21}\psi'_{21} & -2n_p\dfrac{c_2c_3}{c_4}d_td_{21}\psi'_{21} \end{bmatrix}\begin{bmatrix} q \\ z \end{bmatrix} = \begin{bmatrix} \psi_L \\ 0 \end{bmatrix}F_y. \tag{4.17}$$

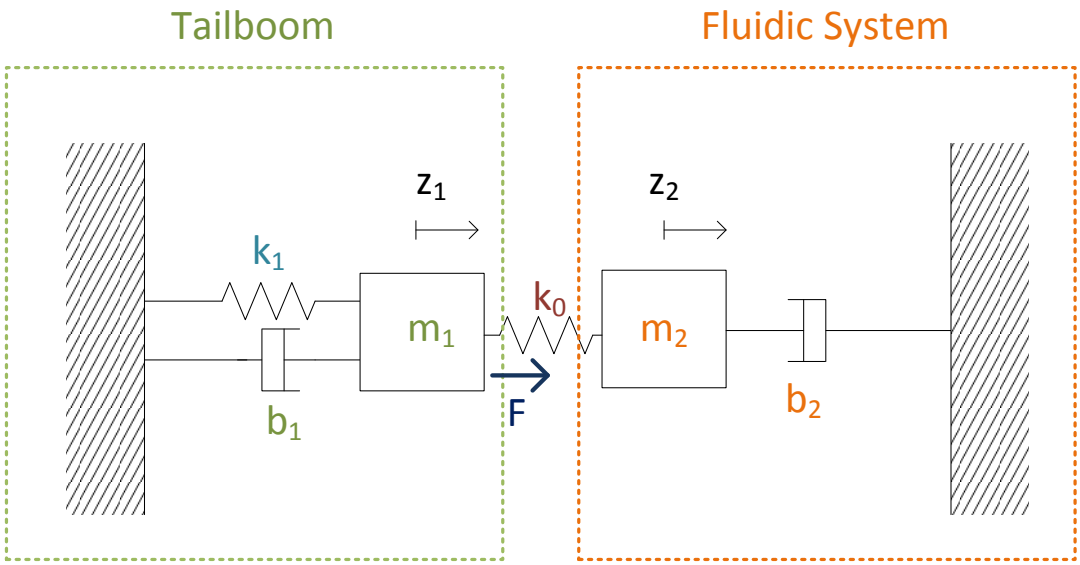

Figure 4-4. Simplified mechanical analogy of a tailboom with F^{2}MC tubes.

Equation (4.17) is significant because it takes the same form as the equations of motion for the simplified mechanical analogy shown in Figure 4-4:

$$\begin{bmatrix} m_1 & 0 \\ 0 & m_2 \end{bmatrix} \begin{bmatrix} \ddot{z}_1 \\ \ddot{z}_2 \end{bmatrix} + \begin{bmatrix} b_1 & 0 \\ 0 & b_2 \end{bmatrix} \begin{bmatrix} \dot{z}_1 \\ \dot{z}_2 \end{bmatrix} + \begin{bmatrix} k_1 + k_0 & -k_0 \\ -k_0 & k_2 + k_0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix}, \tag{4.18}$$

where $F = F_1 \sin \omega t$. By matching the mass, damping, and stiffness matrices of Eq. (4.17) to those of Eq. (4.18), the following equivalence relations can be made:

$$m_1 = M, \tag{4.19}$$

$$m_2 = -n_p^2 I_f c_2 c_3 d_{21} \psi'_{21} d_t, \tag{4.20}$$

$$b_1 = C, \tag{4.21}$$

$$b_2 = -n_p^2 R c_2 c_3 d_{21} \psi'_{21} d_t, \tag{4.22}$$

$$k_1 = K + 2 c_1 n_p d_t d_{21} \psi'_{21}, \tag{4.23}$$

and

$$k_0 = -2n_p \frac{c_2 c_3}{c_4} d_t d_{21} \psi'_{21}. \tag{4.24}$$

Note that, since $c_{1,2,4} \geq 0$ and $c_3 \leq 0$, Eqs. (4.19) through (4.24) are all non-negative.

The tailboom structure is cleanly represented by the oscillator on the left hand side, and the fluid by the oscillator on the right. Spring k_0 couples the fluid to the structure, and is inversely proportional to compliance c_4. The smaller the compliance coefficient, the stiffer k_0 is. It follows that lower-compliance systems better transmit the tailboom motion z_1 into the fluid, which can absorb the energy through motion z_2 of the fluid and dissipate it in dashpot b_2. The analogy in Figure 4-4 also helps to highlight the importance of properly tuning the absorber; to maximize fluid motion z_2 (and therefore energy absorption in m_2 and dissipation in b_2), the fluid inertia m_2 and compliance c_4 must be tuned to the desired forcing frequency.

Researchers have previously analyzed the system in Figure 4-4 with $b_1 = 0$ [58, 59]. For this analysis, $C = b_1 = 0$ is a reasonable assumption since helicopter tailbooms are very lightly damped, with damping ratios typically less than 2%. The following non-dimensional parameters are defined:

$$z_{st} = \frac{F_1}{z_1}, \tag{4.25}$$

$$\lambda = \frac{\omega}{\sqrt{\frac{k_1}{m_1}}}, \tag{4.26}$$

$$\gamma = \frac{\sqrt{\frac{k_0}{m_2}}}{\sqrt{\frac{k_1}{m_1}}}, \tag{4.27}$$

$$\mu = \frac{m_2}{m_1}, \tag{4.28}$$

and

$$\zeta = \frac{b_2}{2\sqrt{m_2 k_0}}, \tag{4.29}$$

where ω is the forcing frequency and z_{st} is the static displacement of m_1 under a static load of magnitude F_1.

Liu and Liu observe that the frequency response plots for the skyhook absorber configuration illustrated in Figure 4-4 intersect at two stationary points, P and Q, regardless of damping ratio ζ [60], as illustrated in Figure 4-5.

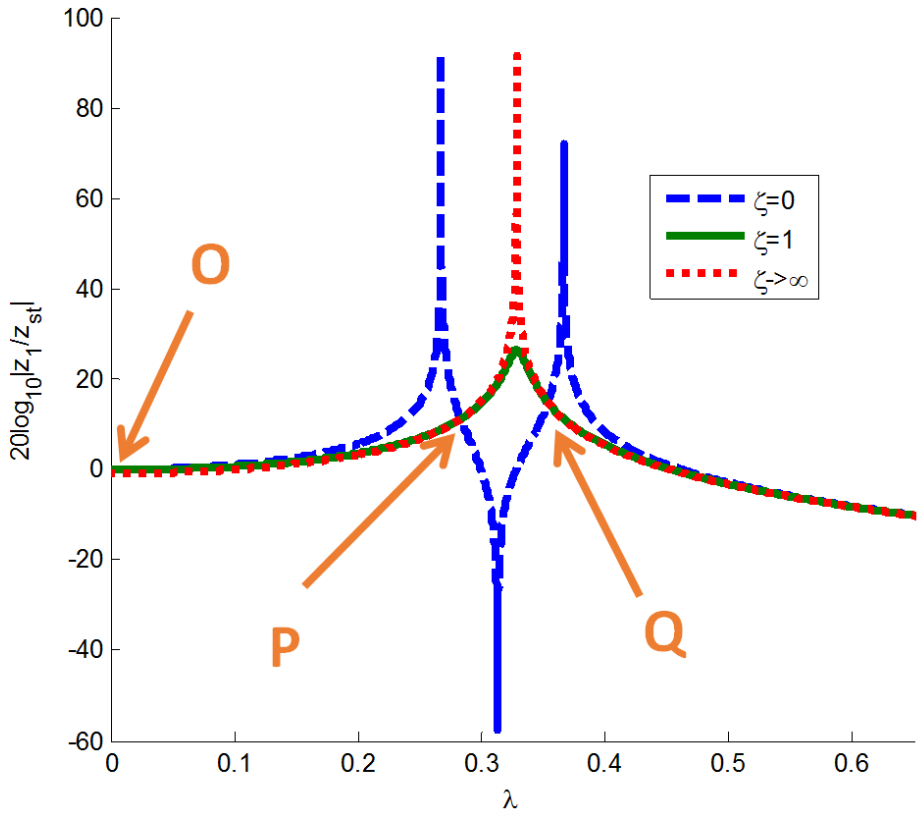

Figure 4-5. Sample frequency response of the system in Figure 4-4.

Very good approximations of the optimal tuning and damping ratios can be determined using fixed-point theory for traditional, non-skyhook absorbers [61]. Ren uses fixed-point theory to derive tuning laws for the skyhook absorber shown in Figure 4-4. In fixed-point theory, an absorber is deemed to be optimally tuned if fixed points P and Q have the same magnitude and zero slope [19]. By applying these conditions to the fixed points of the skyhook absorber, Ren concludes that the optimal non-dimensional tuning parameters are

$$\gamma_{FP} = \sqrt{\frac{1}{1-\mu}} \tag{4.30}$$

and

$$\zeta_{FP} = \sqrt{\frac{3\mu}{8-4\mu}}. \tag{4.31}$$

The higher the mass ratio, the higher the optimal tuning frequency and damping ratio. Substituting Eqs. (4.19) through (4.24) and Eqs. (4.27) through (4.29) into Eqs. (4.30) and (4.31), the optimal fluid inertance can be computed:

$$I_{f_{FP}} = \left(\frac{2}{c_4 n_p}\right) \frac{M}{K + 2n_p\left(c_1 - \frac{c_2 c_3}{c_4}\right) d_t d_{21} \psi'_{21}}. \tag{4.32}$$

Through a similar process, the optimal fluid resistance is found:

$$R_{FP} = I_{f_{FP}} \sqrt{\left(-6n_p \frac{c_2 c_3}{c_4}\right) \frac{d_t d_{21} \psi'_{21}}{2M + n_p^2 I_{f_{FP}} c_2 c_3 d_t d_{21} \psi'_{21}}}. \tag{4.33}$$

The laws in Eqs. (4.32) and (4.33) are tested in the OH-58C model with $N = 1$ and $N = 5$ in the Rayleigh-Ritz approximation; the result is plotted in Figure 4-6 and summarized in Table 4-1.

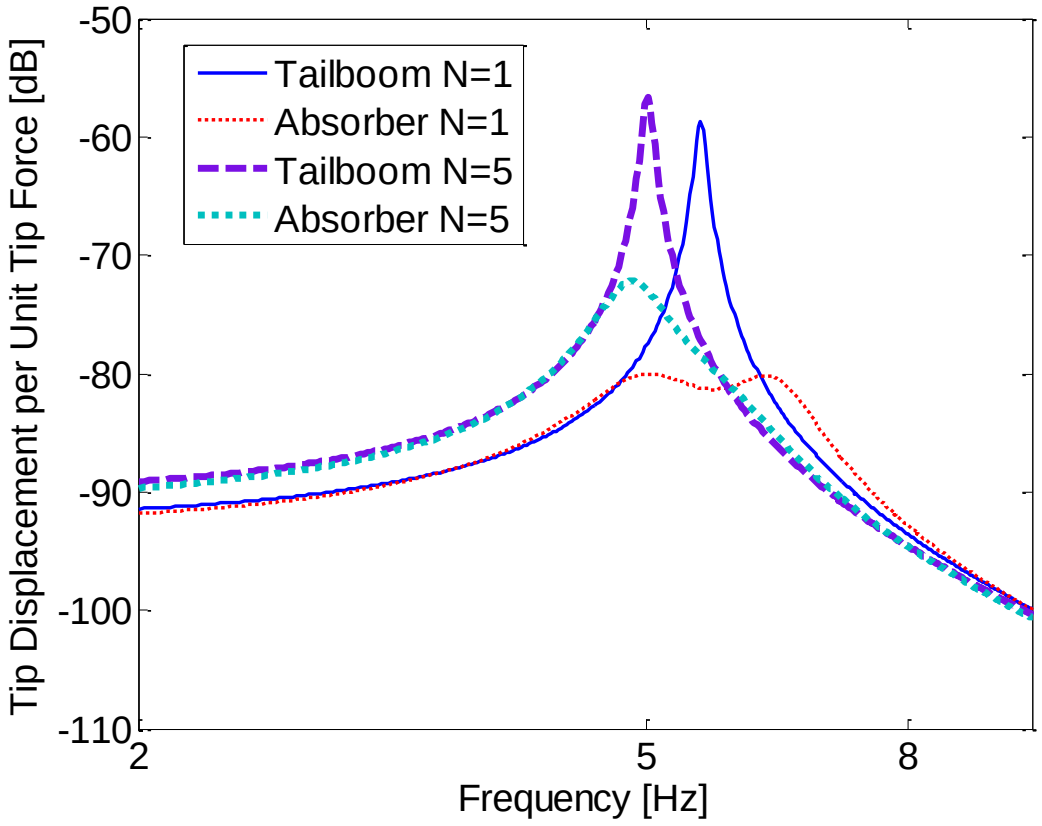

Figure 4-6. Tuned OH-58C absorber for Rayleigh-Ritz series $N = 1$ and $N = 5$ using Eqs. (4.32) and (4.33).

Table 4-1. Inertance tuning estimates.

Method	Base Inertance Required [kg/m^4]	
	N=1	N=5
Eq. 4.8 (Simple tuning rule)	1.8168x10^8	2.1984x10^8
Eq. 4.32 (Fixed points theory)	1.67711x10^8	N/A
Parameter search	1.68x10^8	2.02x10^8

For $N = 1$, the tuned absorber is essentially optimal according to the fixed-points criteria. The base inertance (inertance prior to oscillation frequency corrections) using the simple tuning rule of Eq. (4.8) is 8.3% higher than the optimal solution. For any structure that can reasonably be captured as a SDOF oscillator, Eqs. (4.32) and (4.33) are sufficient. As can be seen in the $N = 5$ case, the tuning is not perfectly optimal when the baseline structure has multiple degrees of freedom. The natural frequency of the system is 10% lower than in the $N = 1$ case. As can be seen from Table 4-1, for the $N = 5$ case the simple tuning rule overestimates required inertance by nearly 9%. However, there is a 20% discrepancy between the fixed-point theory prediction and the optimum determined via a parameter search. Thus, the fixed-point theory may not provide the best inertance tuning estimate for more complicated structures. Given that the system eigenvalues converge from above as more terms are used in the Rayleigh-Ritz series, the fixed-point estimate may at least provide a lower bound for the inertance requirement. Thus, an initial parameter search may reasonably span $I_{fixed\ point} \leq I \leq I_{simple\ tuning}$.

The base resistance (resistance value without frequency correction) estimate provided by the fixed-point method, $R_{N=1} = 2.08 \times 10^8 \frac{kg}{s \cdot m^4}$, essentially matches the parameter search result for $N = 1$. The parameter search result for $N = 5$, $R_{N=5} = 1.83 \times 10^8 \frac{kg}{s \cdot m^4}$, is approximately 12% lower than the fixed-point estimate. In this example, the optimal resistance value is less sensitive to N than is inertance.

Additionally, Ren approximates the maximum response amplitude as

$$\left|\frac{z_1}{z_{st}}\right| = (1 - \mu)\sqrt{\frac{2}{\mu}}. \tag{4.34}$$

Knowing that larger diameter tubes or more tubes effectively provides a higher mass ratio μ, Eq. (4.34) can be used to size the F^{2}MC tubes based on a response amplitude requirement.

Eqs. (4.8/4.32) and (4.33) are practical and useful tuning laws for the absorber inertia track and orifice, respectively. Such a set of tuning rules not only expedites the absorber design process, but enables deeper study of various F^{2}MC parameters. Previous studies of F^{2}MC-based absorbers and dampers have relied on Monte Carlo methods to test various fluidic circuit parameters and find high-performing designs [41, 42]. With tuning laws, more computational resources can be dedicated to studying other parameters instead of tuning the absorber.

Low baseline resistance and high inertance are desirable qualities in an absorber fluidic circuit. Fluid resistance can always be added to a circuit by restricting an orifice, but resistance cannot be removed from an inertia track. In practice, designing an inertia track of reasonable size and weight, with sufficient inertance and the optimal amount of resistance or less, can be difficult. As shown in Eqs. (2.41) and (2.42), inertance is inversely proportional to track radius r_p squared, whereas resistance is proportional to r_p^4. High inertia can be achieved by using a small-diameter inertia track, but the small diameter results in high flow resistance, which reduces absorber performance. One design strategy to avoid this issue is to use a long inertia track with an acceptably large diameter. However, since inertance is merely proportional to inertia track length, this approach results in a large inertia track. Another potential solution is to introduce compliance into the fluidic circuit, either by using a softer bladder material or by adding a fluid accumulator. This reduces the inertance requirement, though at the expense of absorber performance. Finally, a high-density fluid with low viscosity should be used to maximize inertance and minimize flow resistance.

4.1.6 Inertia Track Configuration

The inertia track analyzed in Chapter 2 can only be tuned to a single frequency. The ability to tune various system parameters like inertance would be useful not only for model validation studies, but also for potential future development of semi-active vibration controllers that adapt to varying flight conditions. For instance, a helicopter with variable rotor speed may require an absorber that can be tuned to two different frequencies. In this section, a fluidic circuit comprising two inertia tracks in parallel, each with a tunable orifice as illustrated in Figure 4-7, is proposed.

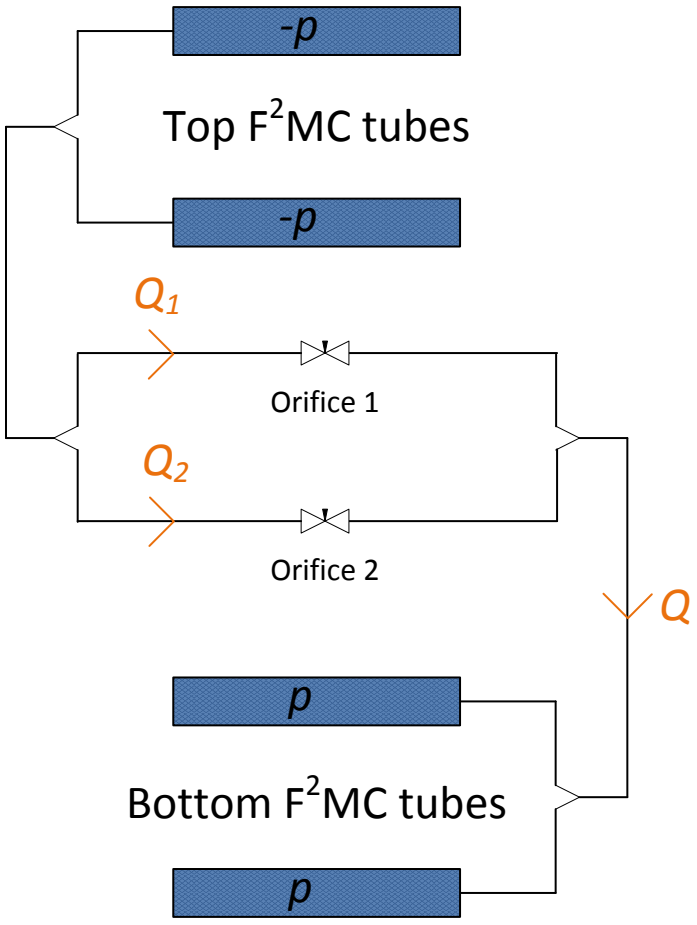

Figure 4-7. Diagram of proposed tunable inertia track.

The two tracks in Figure 4-7 are designed to have different inertances, I_1 and I_2. With Orifice 1 open and Orifice 2 closed, the system would have inertance I_1, and with Orifice 1 closed and Orifice 2 open the inertance would be I_2, as shown in Figure 4-8. Opening both valves adds an additional degree of freedom to the system, and a combination of two absorbers is achieved.

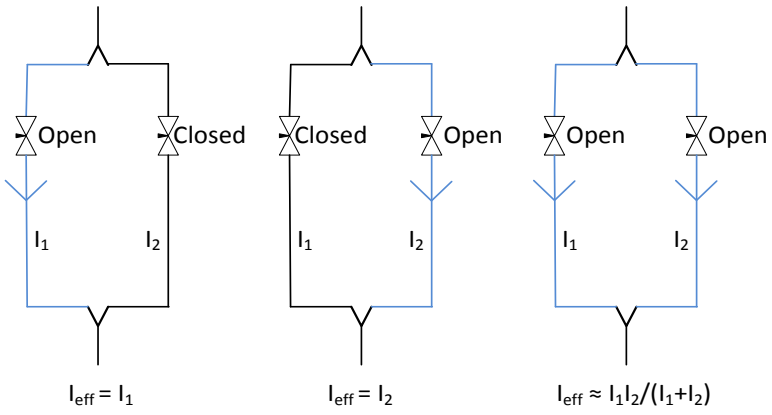

Figure 4-8. Illustration of the three parallel inertia track configurations.

The total volumetric flow rate Q of the fluid through the circuit is

$$Q = Q_1 + Q_2, \tag{4.35}$$

where Q_1 and Q_2 are the volumetric flow rates through inertia tracks 1 and 2, respectively. The pressure difference across both branches of the inertia track is

$$2p = I_1\dot{Q}_1 + R_1Q_1 = I_2\dot{Q}_2 + R_2Q_2, \tag{4.36}$$

where R_1 and R_2 are the flow resistances through orifices 1 and 2, respectively, and Q_1 and p is the fluid pressure developed in the bottom F^{2}MC tubes. For simplicity, losses in the flow through the inertia tracks are assumed to be negligible. Taking the Laplace transforms of Eqs. (4.35) and (4.36) and solving for $Q(s)$ gives

$$Q = 2P\left(\frac{(I_1 + I_2)s + R_1 + R_2}{I_1I_2s^2 + (I_1R_2 + I_2R_1)s + R_1R_2}\right). \tag{4.37}$$

Taking the Laplace transform of Eq. (2.44), solving for Q, and combining the result with Eq. (4.37) gives

$$-n_p c_3 X_t s - n_p c_4 P s = 2P\left(\frac{(I_1 + I_2)s + R_1 + R_2}{I_1 I_2 s^2 + (I_1 R_2 + I_2 R_1)s + R_1 R_2}\right). \tag{4.38}$$

Solve Eq. (4.38) for pressure P:

$$P = -\frac{c_3}{c_4}\frac{I_1 I_2 s^3 + (I_1 R_2 + I_2 R_1)s^2 + R_1 R_2 s}{I_1 I_2 s^3 + (I_1 R_2 + I_2 R_1)s^2 + \left(R_1 R_2 + \frac{2}{n_p c_4}(I_1 + I_2)\right)s + \frac{2}{n_p c_4}(R_1 + R_2)} X_t. \tag{4.39}$$

Taking the Laplace transform of Eq. (2.43) and combining with Eq. (4.39) results in the following transfer function, which is similar to Eq. (2.45):

$$\frac{F_t(s)}{X_t(s)} = \frac{N_3 s^3 + N_2 s^2 + N_1 s + N_0}{D_3 s^3 + D_2 s^2 + D_1 s + D_0} \tag{4.40}$$

where

$$N_0 = \frac{2c_1}{n_p c_4}(R_1 + R_2), \tag{4.41}$$

$$N_1 = \left(c_1 - \frac{c_2 c_3}{c_4}\right)R_1 R_2 + \frac{2c_1}{n_p c_4}(I_1 + I_2), \tag{4.42}$$

$$N_2 = \left(c_1 - \frac{c_2 c_3}{c_4}\right)(I_1 R_2 + I_2 R_1), \tag{4.43}$$

$$N_3 = \left(c_1 - \frac{c_2 c_3}{c_4}\right)I_1 I_2, \tag{4.44}$$

$$D_0 = \frac{2}{n_p c_4}(R_1 + R_2), \tag{4.45}$$

$$D_1 = R_1 R_2 + \frac{2}{n_p c_4}(I_1 + I_2), \tag{4.46}$$

$$D_2 = I_1 R_2 + I_2 R_1, \tag{4.47}$$

and

$$D_3 = I_1 I_2. \tag{4.48}$$

Through a process similar to that outlined in Sections 2.3 and 2.4, the fluid model Eq. (4.40) is integrated into the structural model. Eq. (4.40) is a more general form of Eq. (2.45), and the latter can be recovered by taking the limit as either R_1 or R_2 goes to infinity. This model of the parallel inertia tracks is implemented in MATLAB using the properties listed in Table 4-2 to obtain the frequency response plot in Figure 4-9.

Table 4-2. Inertia track dimensions used for tunable inertia track study.

Absorber Parameter	Value
Inertia Track 1	
- Inner Diameter [mm]	4.32
- Length [m]	1.7
Inertia Track 2	
- Inner Diameter [mm]	4.32
- Length [m]	0.365

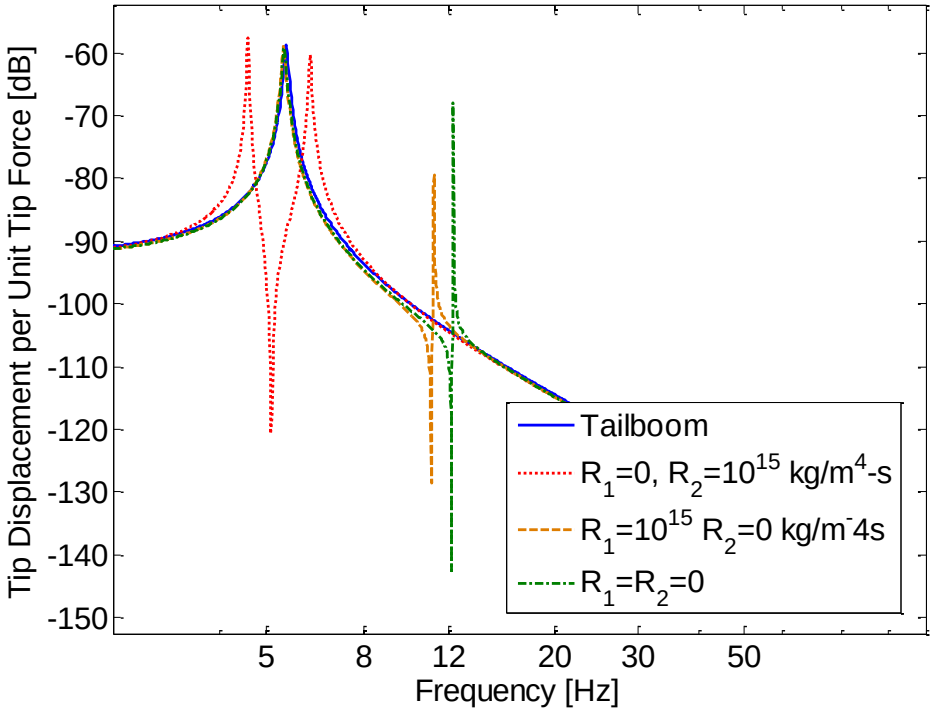

Figure 4-9. Frequency response for various orifice settings.

The $R_2 = 10^{15}\,\frac{kg}{m^4 s}$ (essentially closed orifice) response has a deep anti-resonance flanked by two resonance peaks characteristic of a vibration absorber. The opposite configuration, with $R_1 = 10^{15}\,\frac{kg}{m^4 s}$ and $R_1 = 0$, shows no response attenuation at the first mode and an additional "notch" in the response at 11 Hz, the frequency to which inertia track 2 is tuned. The third case, with no losses in either inertia track, shows a valley at a slightly higher frequency of 12.2 Hz. Since $R_1 = R_2 = 0$ in this case, the effective inertance

$$I_{eff} = \frac{I_1 I_2}{I_1 + I_2}. \tag{4.49}$$

Figure 4-8 shows that, with a parallel inertia track configuration, it is possible to tune an absorber to three distinct frequencies.

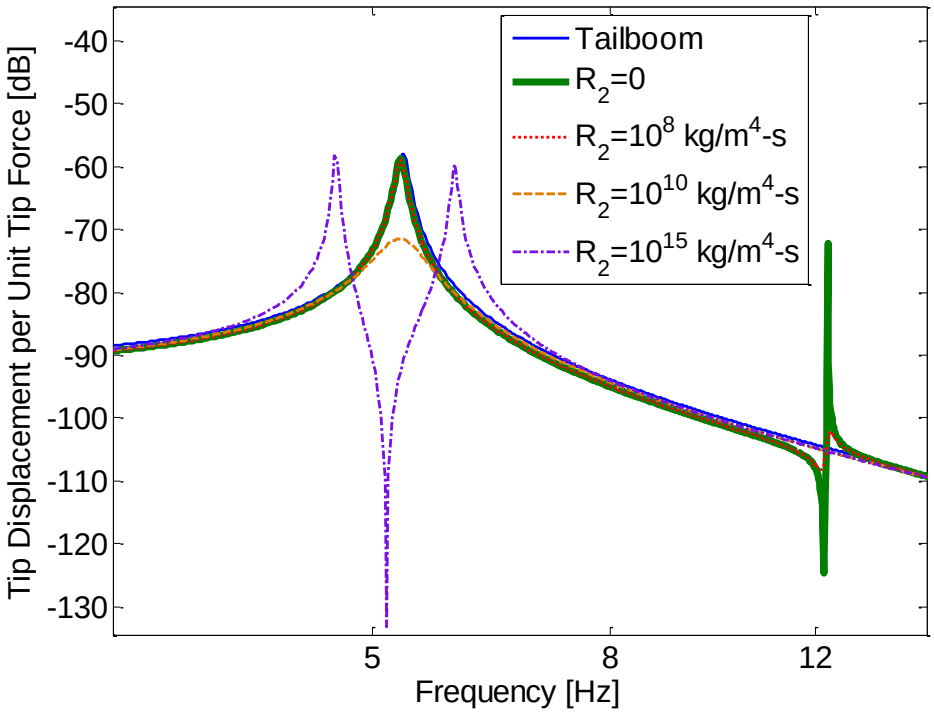

Figure 4-10. Frequency response for open Orifice 1 and partially open Orifice 2.

Figures 4-10 and 4-11 show the system response for partially closed orifices. In Figure 4-10, Orifice 1 is completely open, and the responses for various resistance values for orifice 2 are depicted. The response for $R_2 = 10^{15}\,\frac{kg}{m^4 s}$ is reproduced from Figure 4-9. As orifice 2 is opened such that $R_2 = 10^{10}\,\frac{kg}{m^4 s}$, the sharp resonances become heavily damped and damper-like behavior is observed at the first mode. Simultaneously, an additional "notch" is formed at 12.2 Hz. As R_2 is further reduced to $10^8\,\frac{kg}{m^4 s}$ and 0, the low-frequency absorber diminishes and the 12.2 Hz absorber becomes more prominent. Thus, different combinations of response attenuation at two frequencies can be achieved by partially closing both orifices. For instance, the $R_2 = 10^{10}\,\frac{kg}{m^4 s}$ case can reduce vibration amplitude at 12.2 Hz while still achieving significant damping at the first mode. Similarly, Figure 4-10 shows that different combinations of vibration reduction can be achieved between the two higher frequencies.

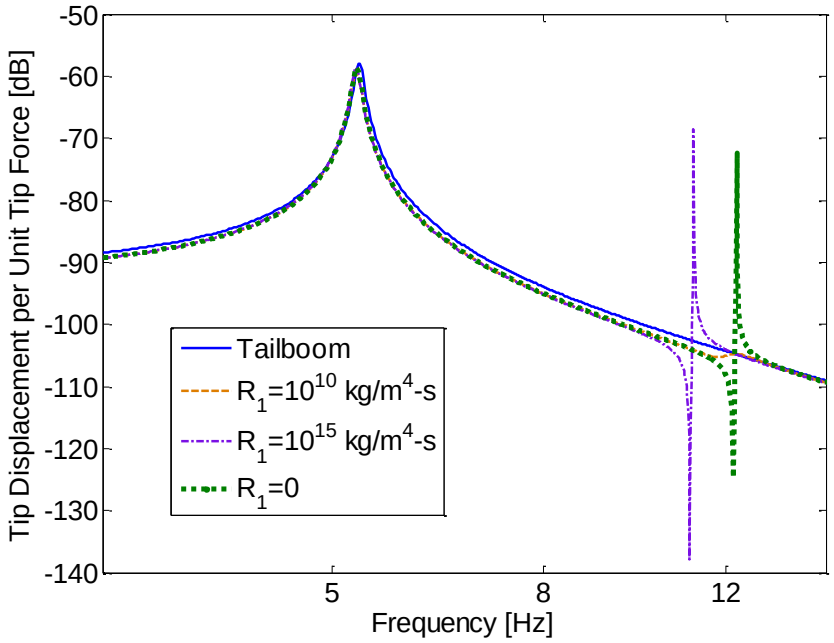

Figure 4-11. Frequency response for open Orifice 2 and partially open Orifice 1.

4.1.8 Design Process Summary

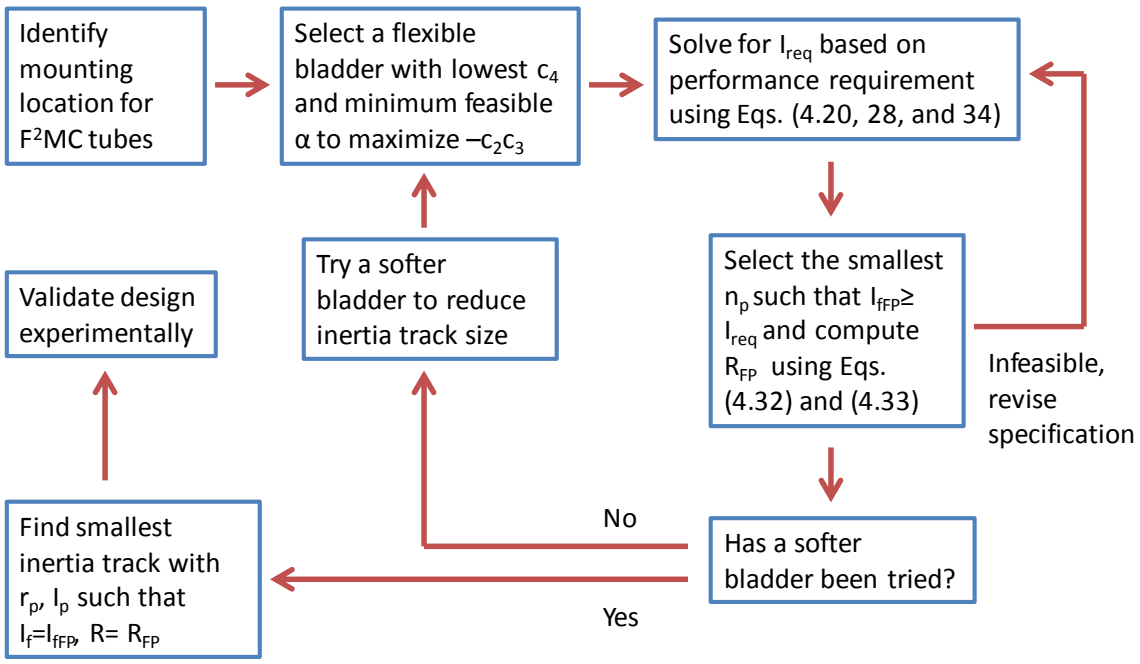

Figure 4-12. Flow chart summarizing absorber design process.

A general design process for F²MC-based absorbers is outlined in this section and summarized in the flowchart in Figure 4-12. The first step is to determine suitable mounting location for the F²MC tubes by identifying areas of high bending strain in the structure. F²MC tubes perform optimally when placed in regions of maximal bending strain, and therefore mode or response shapes of the structure should be obtained at the target frequency. Mode shape slope plots such as the one shown in Figure 4-1 can be used to determine F²MC tube placements that maximize fluid pumping. Care should be taken to ensure that the F²MC tubes are attached to solid structural elements to maximize the coupling between the fluid and the structure. Areas with bulkheads or stringers, for instance, are better candidates than areas with just skin.

Once suitable regions of maximum bending strain are identified, the F²MC bladder must be chosen. Initially, a flexible bladder with the lowest compliance c_4 should be selected, as lower

compliance results in better performance. Additionally, the largest practical F^{2}MC tube diameter and the minimum feasible fiber angle for the F^{2}MC tube should be selected. The effective absorber mass parameter $m_2 \propto -n_p^2 c_2 c_3 I_f$ illustrates the importance of selecting a large but practical tube diameter and small fiber winding angle. The larger the quantity $-c_2 c_3$, the smaller the inertance required to tune the absorber. As shown in Figures 4-13 and 4-14, $-c_2 c_3$ increases with F^{2}MC tube inner radius and lower fiber winding angle. That is, having larger F^{2}MC tubes with smaller fiber winding angles reduces the inertance required from the inertia track. In particular, smaller fiber winding angles can provide improved performance for no weight penalty. Initially, the largest feasible diameter and smallest winding angle should be chosen.

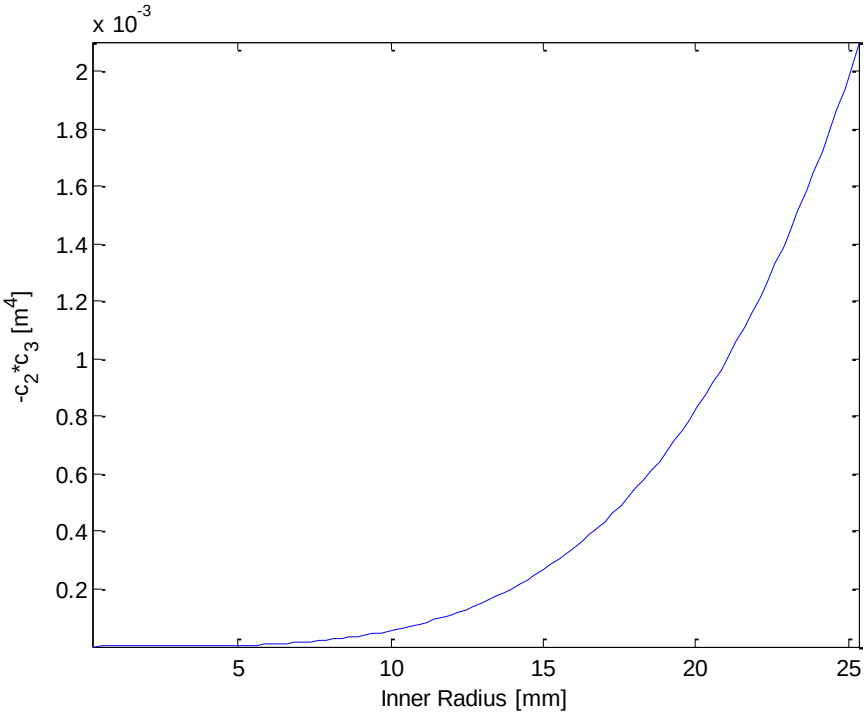

Figure 4-13. $-c_2 c_3$ as a function of F^{2}MC tube inner radius.

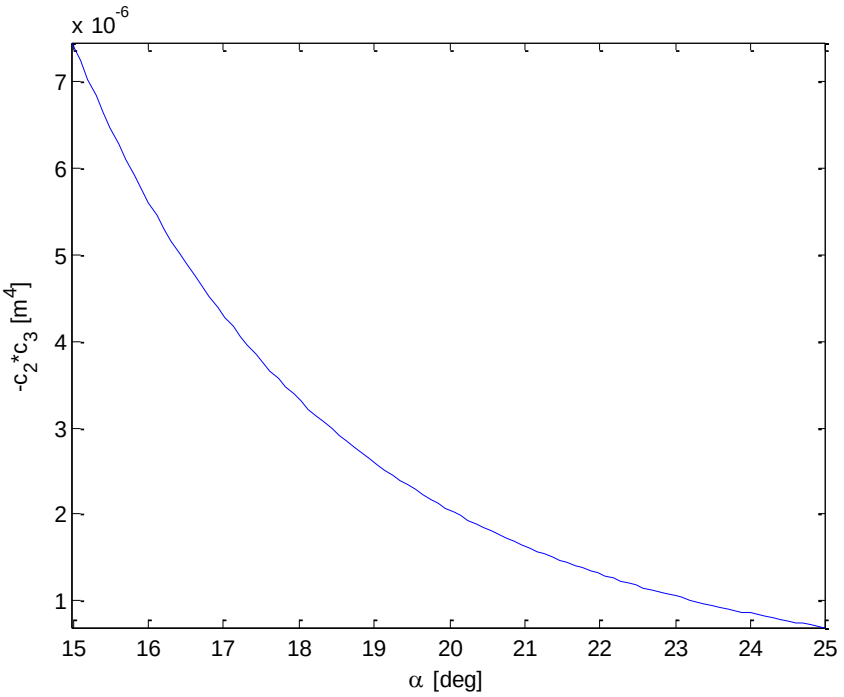

Figure 4-14. $-c_2 c_3$ as a function of fiber winding angle.

In the next step in Figure 4-12, the inertance required to achieve the performance specification is solved for. Given an amplitude requirement, Eq. (4.34) allows one to solve for the mass ratio μ, and therefore the required absorber mass parameter m_2. From Eq. (4.20), the inertance I_{req} required to meet the performance specification can be determined in terms of the number of F^{2}MC tube pairs n_p. Additionally, we solve for the inertance required to tune the absorber in terms of n_p using Eq. (4.32).

Next, the smallest n_p such that $I_{f_{FP}} \geq I_{req}$ is chosen. If this condition cannot be met for any practical value of n_p, then the performance specification is infeasible. That is, a specific inertance value $I_{f_{FP}}$ is needed to optimally tune the absorber, but this inertance value is not sufficient to meet the performance demands. On the other hand, if the $I_{f_{FP}} \geq I_{req}$ condition is met, then the resulting absorber will meet or exceed the performance requirement. From a purely

practical standpoint, single large tubes may save weight since less plumbing hardware is required to connect the top and bottom tubes. If the tubes are mounted internally, having multiple smaller-diameter tubes may be advantageous since a greater separation from the tailboom neutral axis is achieved.

After n_p is selected and the the $I_{f_{FP}} \geq I_{req}$ condition is met, then the resulting absorber will be heavier and over-perform. Therefore, in this case a softer bladder may be explored to reduce the optimal inertance. If a softer bladder has not been tried before, then one may return to the second step of Figure 4-12 and repeat the process with a softer bladder. If not, proceed to the next step.

Finally, the optimal resistance value is calculated according to Eq. (4.33). Inertia track dimensions can then be determined using Eqs. (4.32) and (2.41). The smallest diameter possible should be selected that still satisfy $I_f = I_{f_{FP}}$ and $R = R_{FP}$ to ensure that the inertia track is as light as possible. If desired, this process can be repeated for parallel inertia tracks.

Using this process, an approximately tuned and sized system can be designed, even without a full model of the structure. In practice, the inertia track in particular may need to be tuned to determine the optimal parameters empirically. If a system model is available, this design process can be used to generate approximate solutions, which can be used as starting points for parameter searches.

4.2 Comparison to Existing Technology

The main advantage of the F^{2}MC tube over existing fluidic technology is its pumping efficiency. A conventional piston exerts an axial force proportional to its cross-sectional area and the fluid pressure according to

$$F_p = pA, \tag{4.49}$$

where p is the fluid pressure, A is the piston cross-sectional area, and F_p is the piston axial force. This equation is identical to the F^{2}MC Eq. (2.43), except with F^{2}MC stiffness parameter $c_1 = 0$ and $c_2 = A$. Furthermore, the flow rate of fluid pumped by the piston is proportional to the time derivative its stroke distance x:

$$Q_p = A\dot{x}, \tag{4.50}$$

where Q_p is the volumetric flow rate out of the piston. Eq. (4.50) is identical to the F^{2}MC Eq. (2.44) except that pumping coefficient $c_3 = A$ and compliance $c_4 = 0$. With no compliance, a piston-based treatment may either be used as a pure dashpot or be connected to a fluid accumulator to create a tuned absorber. Combining Eqs. (4.49) and (4.50) and applying the Laplace transform to the result, the piston transfer function is obtained:

$$\frac{F_p(s)}{X(s)} = n_p \frac{A^2}{2}\left(I_f s^2 + Rs\right), \tag{4.51}$$

where $F_p(s) = \mathcal{L}\left[F_p\right]$ and $X(s) = \mathcal{L}[x]$. Note that the equivalent F^{2}MC transfer function from Eq. (2.45) with $c_1 = c_4 = 0$ is

$$\frac{F_t(s)}{X(s)} = -n_p \frac{c_2 c_3}{2}\left(I_f s^2 + Rs\right). \tag{4.52}$$

Typical values for c_2 and c_3, such as for the tubes used in the PSU tailboom structure [56], are $c_2 = 0.0022\ m^2$ and $c_3 = -0.00164\ m^2$. So, $c_2 c_3 = 3.61 \times 10^{-6}\ m^2$. For an equivalent piston of Eq. (4.51) to achieve the same performance, its diameter would need to be $49.2\ mm$, in comparison to the $9.5\ mm$ diameter of the F^{2}MC tubes. A conventional piston with a diameter of $9.5\ mm$ would have an A^2 value nearly 720 times smaller than $-c_2 c_3$. Comparing coefficient A from Eq. (4.50) to $-c_3$ in Eq. (2.44), one can see that, for a low-compliance F^{2}MC tube, a piston of the same diameter would require a stroke distance over 23 times longer than the F^{2}MC tube's to achieve the same fluid pumping.

The F^{2}MC tube fluid mass $m_{f_{tube}}$ is calculated by

$$m_{f_{tube}} = \pi r_t^2 l_t, \tag{4.53}$$

where r_t is the F^{2}MC tube inner radius and l_t is the F^{2}MC tube effective length. The remaining fluid mass in the fluidic circuit is estimated using

$$m_{f_{track}} = \pi r_p^2 l_p. \tag{4.54}$$

For the PSU tailboom experiment, $m_{f_{tube}} = 0.1955\ kg$ and $m_{f_{track}} = 0.2017\ kg$, for a total fluid mass of 0.397 kg. Given that an equivalent piston would require the same inertia track and 26.8 times the cross-sectional area as the F^{2}MC tube, the total fluid weight estimate would be $0.1955\ kg \times 26.8 + 0.2017\ kg = 5.45\ kg$, or nearly 14 times the fluid weight. Assuming that the remaining hardware weight is the same, the total weight for the piston absorber is approximately 7.7 kg. In fact, the weight of the fluid alone exceeds the weight penalty goal of 4.55 kg (10 lb). In practice, the piston system would be even heavier, as the piston and cylinder would be made of heavier materials than the F^{2}MC tube.

While performance/weight comparisons with other devices are difficult to make since they are tested on different structures and since much of the important data is proprietary. However, a comparison can be made between the F^{2}MC-absorber, an equivalent piston, and the active controller developed by Heverly [6], which was tested on the same PSU tailboom. Vibration amplitude reduction at the first mode per added weight is calculated for each system and summarized in Table 4-4:

Table 4-4. Efficiency of Various PSU Tailboom Vibration Treatments

Treatment	% Reduction/Mass [1/kg]
F^{2}MC Absorber	26.1
Piston Absorber	9.1
Piezo Actuators (without power supply)	259
Piezo Actuators (with power supply)	2.64

Active control offers a clear performance advantage over passive methods, as can be seen in Heverly's work, with up to 96% vibration suppression possible using $0.37\ kg$ of stack actuators. However, in practice the stack actuators require amplifiers to provide power. Given that the amplifier used in the experiments weigh 9 kg per channel, in the worst case 36 kg of amplifiers would need to be added. Moreover, these weight estimates do not account for the weight of the required attachment hardware, wiring, and computer. The significant weight added by these amplifiers may be mitigated to some extent by using existing power sources on the aircraft or perhaps through more advanced and lightweight electronics technology. The added weight may also be justified if significant vibration reduction is required over a broad frequency range or for different flight conditions.

Per added weight, the F^2MC-based absorber outperforms the conventional piston by a factor of three, and the worst-case active controller by an order of magnitude. The experimental F^2MC absorber could be made even lighter, as the prototype hardware is over-engineered and designed for flexibility rather than weight. For instance, the copper inertia track may be substituted for a rigid but lighter-weight plastic, and the travel distance of the end fittings may be reduced significantly. Regardless, it is clear that F^2MC tubes enable fluidic vibration treatment based on structural bending strain; with conventional pumpers, this is simply impractical.

Chapter 5

Conclusions and Future Work

5.1 Major Contributions

For the first time, a model of a F^2MC-based vibration absorber is integrated into a realistic tailboom structure. The novel design uses four F^2MC tubes in the coupled configuration, eliminating the need for the fluid accumulator used in previous experiments by Zhu et al. [41]-[42]. Scarborough's model of braid-sheathed F^2MC tubes [44] is modified to account for compliance of the tube inner bladder. A test stand comprising prototype F^2MC tubes integrated into a lab-scale tailboom structure was developed. Static tests performed on the test stand verify the pumping and actuation capability of the F^2MC tubes. Component-level tests are performed to characterize the tubes' pumping ability and compliance. In turn, these results are used to determine an appropriate F^2MC tube bladder design.

A tuned vibration absorber weighing 2.7 kg (6 lb) is shown experimentally to reduce vibration amplitude at the first vertical bending mode of the lab-scale tailboom structure by over 70%. This is the first demonstration of the feasibility of using F^2MC tubes to reduce vibration in a complex and realistic structure. Experiments also show that a partially closed orifice results in a damped absorber that adds 7.8% damping to the first mode. Importantly, the experimental results show excellent correlation with model predictions. For the first time, experiments at various fluid pre-pressures are performed to highlight the importance of sufficient pre-pressurization on absorber performance. Tests with various tailboom forcing amplitudes reveal the nonlinearity and excitation amplitude-dependence of the tailboom structure.

Through a simplified 2-DOF analogy and non-dimensionalization, approximate design rules using the fixed-point theory are presented for the F^2MC-absorber. The optimal fluid

inertance and resistance values are shown in Eqs. (4.32) and (4.33). The design rules and mechanical analogy are used to create a design process, summarized in Figure 4-12, for a F^2MC-based absorber, whereas previous studies by Zhu et al. relied on parameter searches [41]-[42]. Practical considerations regarding placement of the F^2MC tubes, as well as a method of identifying high-bending strain regions, are discussed. An iterative process is proposed in which the lowest-compliance bladder is initially selected, and lower-compliance bladders are tested until the resulting tuning inertance exceeds or meets the inertance required to satisfy the performance specification. It is revealed that the inertance required is inversely proportional to number of F^2MC tube pairs squared, that smaller fiber winding angles and larger tube diameters result in reduced inertance requirement. The tradeoffs between single and multiple tubes is discussed. Approximating the tailboom using only one term in the Ritz series discretization, the resulting optimal absorber parameters show excellent agreement with those obtained through a parameter search. Using five terms in the Ritz series revealed a 20% and 12 % discrepancy between the fixed-point and optimal values for inertance and resistance, respectively. Thus, for complicated structures like the tailboom, a parameter search in the neighborhood of the predicted optimum may be needed. The tuning rules presented greatly reduce the computational resources required to produce an optimal absorber design, whereas previous research by Zhu et al. [41]-[42] relied on tuning via Monte Carlo simulations over a wider parameter space. The design rules are tested via simulation by designing an absorber for the OH-58C tailboom.

A tunable fluidic circuit using two inertia tracks in parallel is proposed. Analysis of the circuit shows that, given negligible losses in the circuit, the absorber can be tuned to provide full absorption at one of three frequencies, or a combination of partial absorption at two different frequencies.

Finally, the F^2MC-absorber performance and weight are compared against an absorber using conventional pistons. The F^2MC-absorber is shown to outperform the piston-based device

on a per-added weight basis, conservatively, by a factor of three. A similar comparison is done with an active controller using piezoelectric actuators. While the active method could outperform the F^2MC-absorber by a factor of ten for the best-case scenario, this figure ignores the weight added from attachment hardware, wiring, computer, and most importantly, the amplifiers for the actuators. In the worst-case scenario, it is estimated that the F^2MC-absorber could outperform the active approach by a factor of ten.

5.2 Ongoing and Future Work

The end goal of this research is a demonstration of F^2MC technology on a full-scale, real-life structure. This involves designing, fabricating, and testing a F^2MC-absorber on an OH-58C tailboom at the U.S. Army Research Laboratory at Aberdeen Proving Ground. An analytical model of a full-scale OH-58C tailboom has been developed and used to produce preliminary F^2MC-asborber designs in preparation for future tests. Various dynamic experiments will be carried out not only to demonstrate vibration reduction performance but also to verify the predictive capability of the model. A successful full-scale demonstration will bring F^2MC technology closer to widespread adoption in aerospace applications, where reducing weight penalty is a major priority.

5.2.1 OH-58C Tailboom Model Validation and Future Improvement

The full-scale tailboom test stand has been built and modally tested. Using an impact hammer to excite the tailboom vertically at the tip, frequency response plots and mode shapes have been obtained. Preliminary experimental results, shown in Figure 5-1, show some agreement between model predictions and experiment. However, the OH-58C structure is far more

complicated than the PSU tailboom, and modifications could be made to the model to improve agreement. Furthermore, the OH-58C helicopter is resting on its landing gear, and therefore the tailboom is not cantilevered in the same fashion as the PSU tailboom. The mode shown in the experimental curve of Figure 5-1 is likely a rigid tailboom mode caused by rocking of the entire helicopter on its skids. In future work, the tailboom model could be modified to account for this, possibly by modeling the fuselage, to which the tailboom is cantilevered, as a large mass on a linear and/or torsional spring.

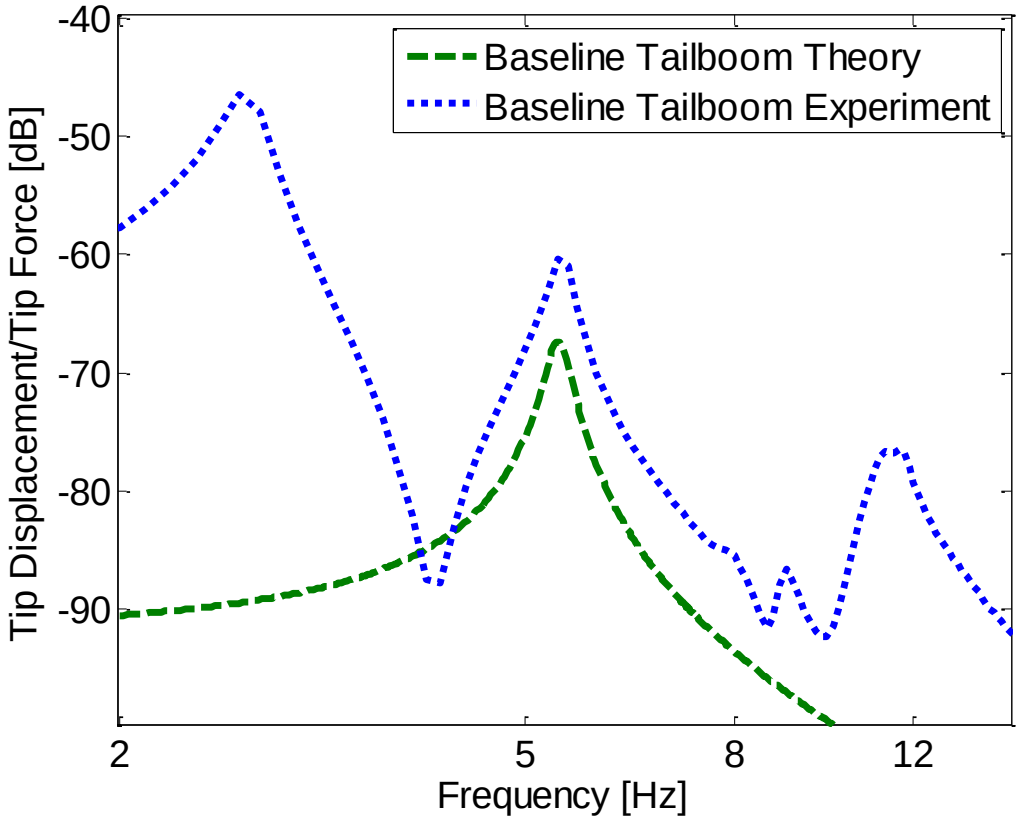

Figure 5-1. Comparison of experimental and theoretical untreated OH-58C tailboom frequency responses.

Additionally, a static actuation test has been performed on the tailboom. This test, identical to that performed on the PSU tailboom in Section 3.2.2, is conducted by closing the inertia track orifice and pressurizing the bottom F^2MC tubes. As the bottom tubes are pressurized, they contract axially and thus cause the tailboom to deflect downward. This deflection is measured using a laser vibrometer, and the result is shown in Figure 5-2. Based on Figure 5-2, it is clear that the F^2MC tubes have authority over the tailboom structure. While there is hysteresis in the structure as evidenced by the different paths taken by the loading and unloading curves, and the authority seems to taper off at higher pressures, the F^2MC tubes overall show more authority than predicted by the model. Indeed, as can be seen in Figure 5-1, the model is statically stiffer than the experiment. Based on these observations, it may be appropriate to slightly tune up the structural stiffness of the OH-58C model since the structure is clearly softer than initially thought.

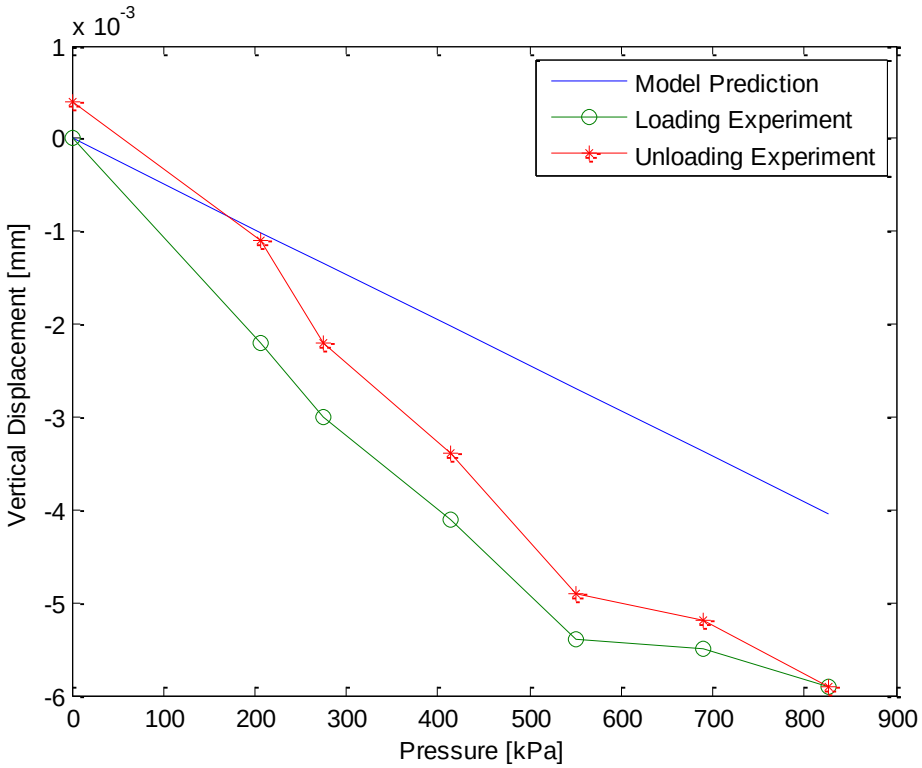

Figure 5-2. Experimental and theoretical OH-58C tailboom static actuation curves.

5.2.2 Full-Scale Dynamic Tests

The OH-58C tailboom has been built and is illustrated in Figure 5-3. In identical fashion to the PSU experiment, F^{2}MC tubes are crimped onto stainless steel end fittings, which in turn connect to an external fluidic circuit made of rigid plastic tubing. The threaded end fittings are tightened using nuts onto aluminum L-brackets, shown in Figure 5-4, which are bolted onto the outside of the tailboom for ease of installation and observation. The brackets are riveted onto structural elements on the tailboom to maximize coupling between the fluid and the structure.

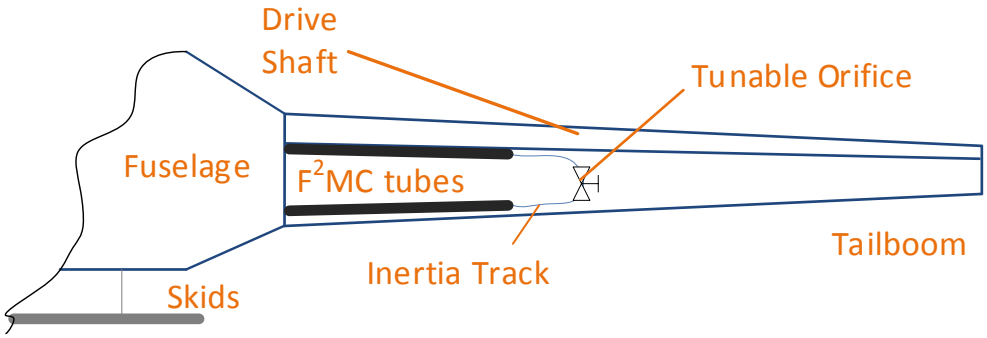

Figure 5-3. Schematic diagram of the OH-58C tailboom test stand.

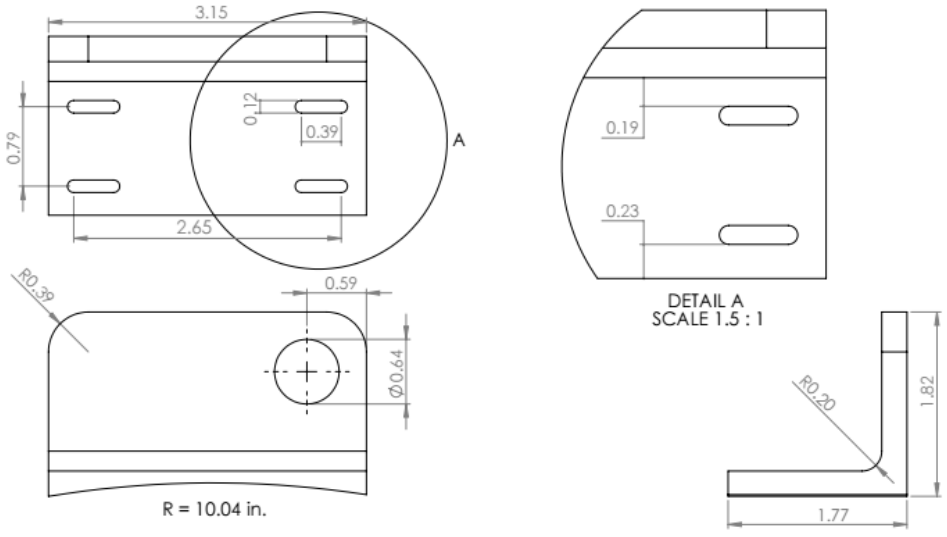

Figure 5-4. Sample drawing of L-brackets used to attach F^{2}MC tubes to the tailboom.

The F^{2}MC tube length is chosen based on geometric constraints of the tailboom as well as on experimentally obtained mode shapes, shown in Figure 5-5. For maximum bending strain at the first mode, F^{2}MC tubes extending toward the tip from the fuselage slightly forward of the root are most desirable. A distance of 1.18 m between the mounting L-brackets is selected to ensure that the brackets are secured onto structural elements instead of the skin only. The F^{2}MC tube diameter and inertia track dimensions are selected using the MATLAB model.

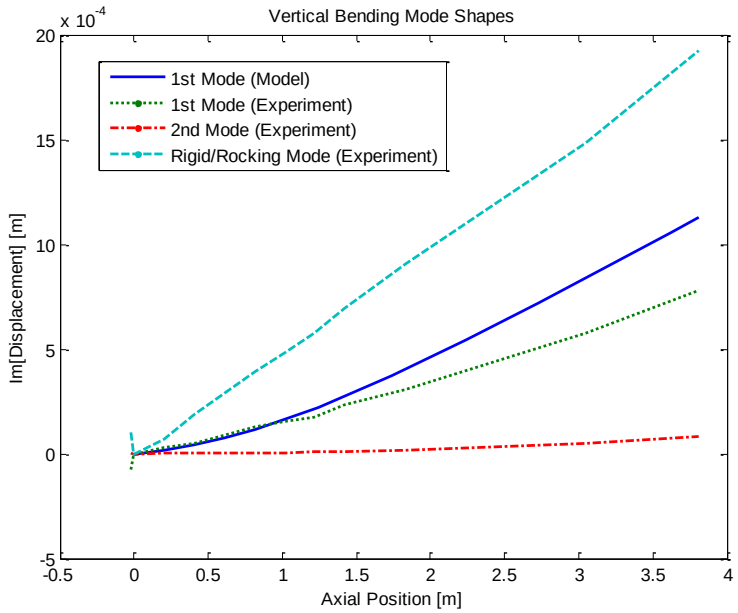

Figure 5-5. Experimental and theoretical (first mode only) mode shapes of the OH-58C tailboom.

Table 5-1. Summary of proposed full-scale absorber parameters.

Absorber Parameter	Value
F^{2}MC tube	
- Length [m]	1.05
- Outer Diameter [mm]	11.1
- Bladder Thickness [mm]	1.59
- Bladder Material	Latex Rubber
- Fiber Winding Angle [deg]	18
- Fiber Material	Stainless Steel
Inertia Track	
- Inner Diameter [mm]	4.32
- Length [m]	1.43
Weight Estimate [kg] (lb)	3.8 (8.4)

Using the design parameters from Table 5-1, the frequency response in Figure 5-6 is obtained. The tailboom has a sharp resonance peak around 5.5 Hz, its first vertical bending mode. With the F^2MC-based tuned absorber installed, this resonance peak amplitude is reduced by nearly 76%. Having demonstrated control authority and designed a promising absorber through simulation, the next logical step in this research is to demonstrate performance experimentally on the tailboom. The improved model proposed in Section 5.2.1 must also be validated against the experimental data.

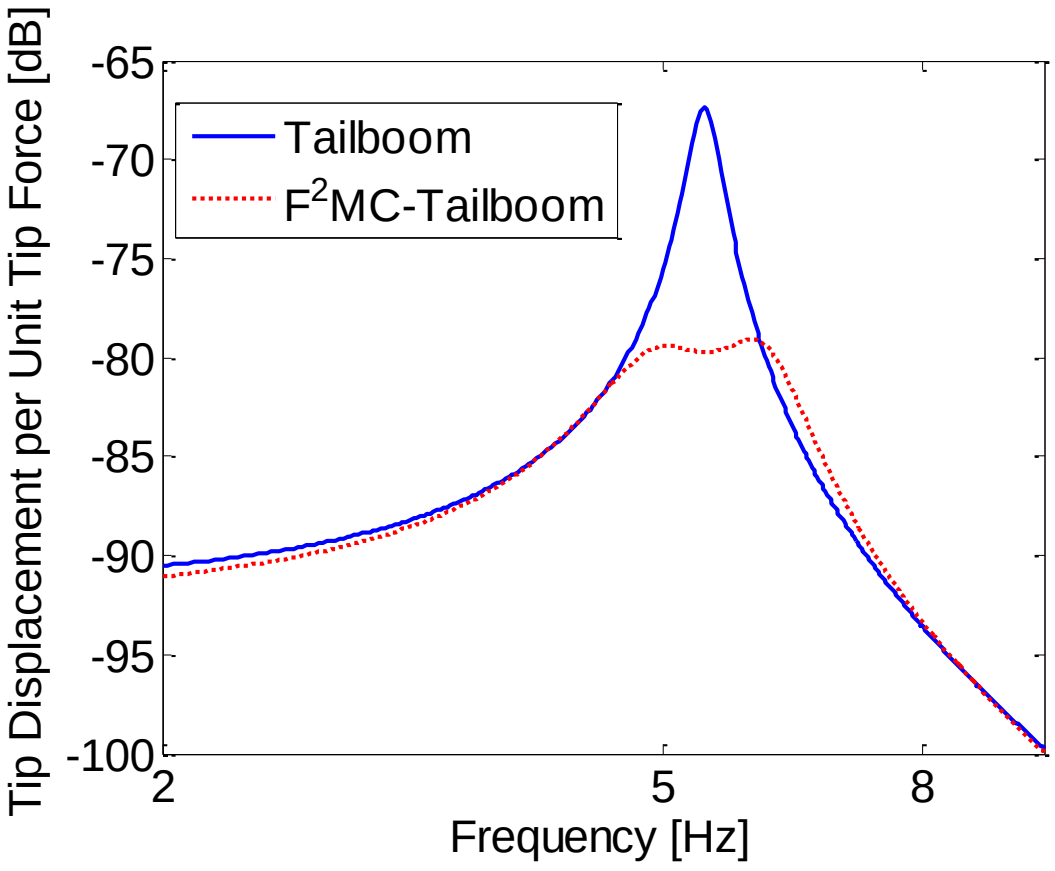

Figure 5-6. Simulated OH-58C frequency response plot with and without F^2MC tubes.

Longer and wider F^2MC tubes provide better performance, but add weight. The F^2MC tube length is constrained by the geometry of the tailboom, and the tube diameter is chosen to produce sufficient vibration control without exceeding the absorber weight limit of 10 lbs. The preliminary absorber design parameters are summarized in Table 5-1. The inertia track dimensions are selected to provide the inertance required for the fluid to absorb vibration at the first vertical bending mode. Furthermore, the same fluid used in the lab-scale tailboom dynamic experiments is used for its high density and relatively low viscosity.

5.2.3 Design Code to Minimize Weight

The design process described in Section 4.1.8 and summarized in Figure 4-12 does not necessarily result in the lowest-weight solution, or the most weight-effective one. To a limited extent, the process aims to reduce weight; for example, to minimize inertia track weight it calls for trying different bladder materials until one with the maximum possible compliance is found. However, some tradeoffs like the number of F^2MC tube pairs are not considered. For instance, having more pairs of tubes may save weight compared to fewer but with a larger track. It is not immediately clear how having multiple tubes will affect the system weight due to the added fittings and other hardware.

One potentially useful tool could consist of the tailboom simulation code and a database of various component weights. A tool could be developed that accounts for the weight of not just the fluid and F^2MC tubes but also components such as fittings, for any arbitrary configuration of F^2MC tubes and inertia tracks. Such a tool would allow one to obtain a clearer picture of the weight-related tradeoffs, whereas currently the design process simply meets a performance specification. Furthermore, the code could be used for parameter searches to obtain an optimally lightweight solution. For instance, such a code could determine the extent to which a lighter fiber

material could be used to replace the stainless steel fibers. In this case, a parameter search reveals that a fiber with an elastic modulus three orders of magnitude lower than stainless steel's could be used with no noticeable performance reduction. Much lighter fiber materials can thus safely be used. This type of analysis would be a tremendous benefit to implementing an absorber in real life applications. A weight-optimized F^2MC-absorber has the added benefit of adding minimal weight to the root of the tailboom. Unlike conventional mechanical absorbers, which would need to be placed as far to the rear as possible to capitalize on the larger vibration amplitudes there, the F^2MC-absorber minimally disrupts the center of gravity of the helicopter, requiring less counterweight in the front of the aircraft.

Bibliography

[1] Bielawa, R.L., 1992, *Rotary Wing Structural Dynamics and Aeroelasticity*, Washington DC, American Institute of Aeronautics and Astronautics, AIAA Education Series, pp. 163-208.

[2] Krysinski, T., and Malburet, F., 2007, *Mechanical Vibrations: Active and Passive Control*, ISTE, Newport, CA, pp. 253-318.

[3] Loewy, R.G., 1984, "Helicopter Vibrations: A Technological Perspective," *Journal of the American Helicopter Society*, 29(4), pp. 4-40.

[4] Bansemir, H., Bongers, B., 2001, Eurocopter Deutschland GmbH, Munich, Germany, U.S. Patent No. 6286782.

[5] duBois, J.L., Lieven, N.A., and Adhikari, S., 2007, "Adaptive Passive Control of Dynamic Response Through Structural Loading," *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, HI.

[6] Heverly, D.E., 2002, "Optimal Actuator Placement and Active Structure Design for Control of Helicopter Airframe Vibrations," PhD Thesis, Pennsylvania State University.

[7] de Silva, C.W., ed., 2005. *Vibration and Shock Handbook*, CRC Press, Boca Raton, FL, pp. 23-3.

[8] Ogata, K., 2004. *System Dynamics*, Pearson Prentice Hall, Upper Saddle River, NJ, p. 441.

[9] Flannelly, W. G., 1967, "Dynamic Antiresonant Vibration Isolator," U.S. Patent No. 3,322,379.

[10] Halwes, D. R., 1980. "LIVE – Liquid Inertia Vibration Eliminator," *Proc. American Helicopter Society 36th Annual Forum*, Washington, D.C.

[11] McGuire, D. P., 1994, "Fluidlastic Dampers and Isolators for Vibration Control in Helicopters," *Proc. American Helicopter Society* 5th *Annual Forum,* American Helicopter Society.

[12] Pfeiffer, F., 2008, *Mechanical System Dynamics*, Springer-Verlag, Berlin, p. 76.

[13] Johnson, C.D., 1995, "Design of passive damping systems," *Journal of Mechanical Design*, 117(B), pp. 171–176.

[14] Viana, F.A.C., Steffen Jr., V., 2006, "Multimodal vibration damping through piezoelectric patches and optimal resonant shunt circuits," Journal of the Brazilian Society of Mechanical Sciences and Engineering, 28(3), pp. 293-310.

[15] Preumont, A., 2011, *Vibration Control of Active Structures*, Springer-Verlag, Berlin, p. 5.

[16] Frahm, H., 1911, "Device for Damping Vibrations of Bodies," U.S. Patent No. 989,958.

[17] Roberson, R. E., 1952, "Synthesis of a Nonlinear Dynamic Vibration Absorber," *Journal of The Franklin Institute*, 254(3), pp. 205–220.

[18] Hunt, J. B., and Nissen, J. C., 1982, "The Broadband Dynamic Vibration Absorber," *Journal of Sound Vibration*, 83(4), pp. 573–578.

[19] Den Hartog, J. P., 1985, *Mechanical Vibrations*, Dover Publication, Mineola, NY.

[20] Rusovici, R., Dosch, J. J., and Lesieutre, G. A., 2002, "Design of a Single-Crystal Piezoceramic Vibration Absorber," *Journal of Intelligent Material Systems and Structures*, 13(11), pp. 705–712.

[21] Igusa, T., and Xu, K., 1994, "Vibration Control Using Multiple Tuned Mass Dampers," *Journal of Sound and Vibration*, 175(4), pp. 491–503.

[22] Rana, R., and Soong, T. T., 1998, "Parametric Study and Simplified Design of Tuned Mass Dampers," *Engineering Structures*, 20(3), pp. 193–204.

[23] Sun, J. Q., Jolly, M. R., and Norris, M. A., 1995, "Passive, Adaptive and Active Tuned Vibration Absorbers: A Survey," *ASME Journal of Mechanical Design*, 117(B), pp. 234–242.

[24] Soong, T. T., and Dargush, G. F., 1997, *Passive Energy Dissipation Systems in Structural Engineering*, Wiley, New York, p. 3.

[25] Karnopp, D., Crosby, M.J., and Harwood, R.A., 1974, "Vibration control using semi-active force generators," *ASME Journal of Engineering for Industry*, 96, pp. 619-626.

[26] Walsh, P. L., and Lamancusa, J. S., 1992, "A Variable Stiffness Vibration Absorber for Minimization of Transient Vibrations," *Journal of Sound and Vibration*, 158(2), pp. 195–211.

[27] Koo, J. H., Ahmadian, M., Setareh, M., and Murray, T., 2004, "In Search of Suitable Control Methods for Semi-Active Tuned Vibration Absorbers," *Journal of Vibration Control*, 10(2), pp. 163–174.

[28] Housner, G. W., Bergman, L. A., Caughey, T. K., Chassiakos, A. G., Claus, R. O., Masri, S. F., Skelton, R. E., Soong, T. T., Spencer, B. F., and Yao, J. T. P., 1997, "Structural Control: Past, Present, and Future," *Journal of Engineering Mechanics*, 123(9), pp. 897–971.

[29] Williams, K., Chiu, G., and Bernhard, R., 2002, "Adaptive-Passive Absorbers Using Shape-Memory Alloys," *Journal of Sound and Vibration*, 249(5), pp. 835–848.

[30] Hagood, N. W., and von Flotow, A., 1991, "Damping of Structural Vibrations With Piezoelectric Materials and Passive Electrical Networks," *Journal of Sound and Vibration*, 146(2), pp. 243–268.

[31] Hollkamp, J. J., and Starchville, T. F. Jr., 1994, "A Self-Tuning Piezoelectric Vibration Absorber," *Journal of Intelligent Material Systems and Structures*, 5(4), pp. 559–566.

[32] Davis, C. L., Lesieutre, G. A., and Dosch, J., 1997. "A Tunable Electrically Shunted Piezoceramic Vibration Absorber," *Proc. SPIE*, 3045, pp. 51–59.

[33] Deng, H., Gong, X., and Wang, L., 2006, "Development of an Adaptive Tuned Vibration Absorber With Magnetorheological Elastomer," *Smart Materials and Structures*, 15(5), pp. 111–116.

[34] Yalla, S. K., and Kareem, A., 2003, "Semiactive Tuned Liquid Column Dampers: Experimental Study," *Journal of Structural Engineering*, 129(7), pp. 960–971.

[35] Gao, H., Kwok, K. C. S., and Samali, B., 1997, "Optimization of Tuned Liquid Column Dampers," *Engineering Structures*, 19(6), pp. 476–486.

[36] Heverly, D.E., Wang, K.W., and Smith, E.C., 2001, "An Optimal Actuator Placement Methodology for Active Control of Helicopter Airframe Vibrations," *Journal of the American Helicopter Society* 46(4), pp. 251-261.

[37] Strehlow, H., Rottmayr, H., Duerr, J.K., and Zaglauer, H.W., 2008, "Method for Damping Rear Extension Arm Vibrations of Rotorcraft and Rotorcraft with a Rear Extension Arm Vibration Damping Device," U.S. Patent No. 2008/0173754A1.

[38] Staple, A.E., and MacDonald, B.A., 1993, "Active Vibration Control Systems," U.S. Patent No. 5219143.

[39] Gaffey, T.M., Kidd, D.L., and Grimes, M.L., 1988, "Nonlinear Vibration Absorber," U.S. Patent No. 4766984.

[40] Vuillet, Alain, and Zoppitelli, Elio, 1998, "Device for Reducing the Vibration on the Structure of a Helicopter," U.S. Patent No. 5775637.

[41] Zhu, B., Rahn, C. D., and Bakis, C. E., 2015, "Fluidic Flexible Matrix Composite Damping Treatment for a Cantilever Beam," *Journal of Sound and Vibration*, 340, pp. 80-94.

[42] Zhu, B., Rahn, C. D., and Bakis, C. E., 2015, "Fluidic Flexible Matrix Composite Vibration Absorber for a Cantilever Beam," *Journal of Vibration and Acoustics*, 137(2):021005, 11 p, 2015.

[43] Lotfi-Gaskarimahalle A., Scarborough III, L. H., Rahn, C. D., and Smith, E. C., 2009. "Fluidic Composite Tuned Vibration Absorbers," *Proc. ASME Conference on Smart Materials, Adaptive Structures and Intelligent Systems*, SMASIS2009-1349, pp. 501-508.

[44] Scarborough III, L.H., Rahn, C.D., Smith, E.C., 2011, "Fluidic Composite Tunable Vibration Isolators," *Journal of Vibration and Acoustics*, 134 (1).

[45] Shan, Y., Philen, M.P., Bakis, C.E., Wang, K.W., and Rahn, C.D., 2006. "Nonlinear-Elastic Finite Axisymmetric Deformation of Flexible Matrix Composite Membranes under Internal Pressure and Axial Force," *Composites Science and Technology*, 66(15), pp. 3053-3063.

[46] Zhu, B., Rahn, C.D., and Bakis, C.E., 2011. "Tailored Fluidic Composites for Stiffness or Volume Change," *Proc. ASME 2011 Conference on Smart Materials, Adaptive Structures and Intelligent Systems*, SMASIS 2011-4962, Scottsdale, A.Z.

[47] Liu, W. and Rahn, C. D., 2003. "Fiber-Reinforced Membrane Models of McKibben Actuators," *Journal of Applied Mechanics, Transactions ASME*, 70(6), pp. 853-859.

[48] Shan, Y., Philen, M., Lotfi, A., Li, S., Bakis, C. E., Rahn, C. D., and Wang, K. W., 2009. "Variable Stiffness Structures Utilizing Fluidic Flexible Matrix Composites," *Journal of Intelligent Material Systems and Structures*, 20, pp. 443-456.

[49] Zhu, B., Rahn, C. D., and Bakis, C. E., 2012. "Actuation of Fluidic Flexible Matrix Composites in Structural Media," *Journal of Intelligent Material Systems and Structures*, 23(3), pp. 269-278.

[50] Philen, M., Phillips, D., and Baur, J., 2009, "Variable Modulus Materials based upon F^2MC Reinforced Shape Memory Polymers," *Proc. 50th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference.*, Palm Springs, C.A., pp. 461-470.

[51] Li, S., and Wang, K.W., 2012, "On the dynamic characteristics of biological inspired multicellular fluidic flexible matrix composite structures," *Journal of Intelligent Material Systems and Structures*, 23(3), pp. 291-300.

[52] Miura, K., Rahn, C.D., and Smith, E.C., 2014. "Passive Tailboom Vibration Control Using Fluidic Flexible Matrix Composite Tubes," *Proc. 55th AIAA/ASME/ASCE/AHS/SC Structures, Structural Dynamics, and Materials Conference*, 2014-1368, National Harbor, M.D.

[53] Miura, K., Krott, M.J., Rahn, C.D., Smith, E.C., 2014, "Experimental Characterization of a Tailboom with Fluidic Flexible Matrix Composite Tubes," *Proc. AHS 70th Annual Forum,* Montréal, QC, CA.

[54] Krott, M.J., Miura, K., Rahn, C.D., and Smith, E.C., 2015, "Tube Compliance Effects on Fluidic Flexible Matrix Composite Devices for Rotorcraft Vibration Control," *Proc. 56th AIAA/ASME/ASCE/AHS/SC Structures, Structural Dynamics, and Materials Conference*, 2015-1416, Kissimmee, FL.

[55] Miura, K., Krott, M.J., Rahn, C.D., Smith, E.C., and Romano, P.Q., 2015, "Experimental Validation of Tailboom Vibration Control Using Fluidic Flexible Matrix Composite Tubes," *Proc. AHS 71st Annual Forum,* Virginia Beach, VA.

[56] Ginsberg, J.H., 2001, *Mechanical and Structural Vibrations: Theory and Applications*, Wiley, New York, pp. 25-26.

[57] Donovan, F.M., Taylor, B.C., and Su, M.C., 1991, "One-Dimensional Computer Analysis of Oscillatory Flow in Rigid Tubes," *Journal of Biomechanical Engineering*, 113(4), pp. 476-484.

[58] Chou, C.P., Hannaford, B., 1994. "Static and Dynamic Characteristics of McKibben Pneumatic Artificial Muscles," *Proc. IEEE International Conference on Robotics and Automation*, 1994.350977.

[59] Ren, M.Z., 2001. "A Variant Design of the Dynamic Vibration Absorber," *Journal of Sound and Vibration*, 245(4), pp. 762-770.

[60] Liu, K. and Liu, J., 2004. “The Damped Dynamic Vibration Absorbers: Revisited and New Result,” *Journal of Sound and Vibration*, 284, pp. 1181-1189.

[61] Cheung, Y.L. and Wong, W.O., 2011. “H-infinity Optimization of a Variant Design of the Dynamic Vibration Absorber – Revisited and New Results,” *Journal of Sound and Vibration*, 330, pp. 3901-3912.

[62] Scarborough III, L.H., 2014, “Dynamics of Fluidic Devices with Applications to Rotor Pitch Links,” PhD Thesis, Pennsylvania State University.

Appendix A

Basis Functions for a Spring-Hinged Beam - Derivation

The spring-hinged beam is modeled as an Euler-Bernoulli beam:

$$EIw'''' + \rho A\ddot{w} = 0, \tag{A.1}$$

where $w(x,t)$ is the beam vertical displacement, EI is flexural rigidity, ρ is the beam density, and A is the cross-sectional area. The displacement solution is of the form

$$w(x,t) = W(x)e^{i\omega t}, \tag{A.2}$$

where ω is frequency and W is the shape function. Substituting the solution Eq. (A.2) into (A.1), we obtain

$$W_n(x) = B_1\sin(\alpha_n x) + B_2\cos(\alpha_n x) + B_3\sinh(\alpha_n x) + \mathrm{B}_4\cosh(\alpha_n x), \tag{A.3}$$

with

$$\alpha = \left(\omega_n^2\frac{\rho A}{EI}\right)^{1/4}, \tag{A.4}$$

where n denotes the n-th mode. The boundary conditions are

$$W(0) = 0, \tag{A.5}$$

$$W''(L) = 0, \tag{A.6}$$

$$W'''(L) = 0, \tag{A.7}$$

and

$$K_t W'(0) = EIW''(0), \tag{A.8}$$

where K_t is the root spring stiffness. By applying the boundary conditions from Eq. (A.5) – (A.7), three of the coefficients B can be determined as

$$B_2 = -B_4, \tag{A.9}$$

$$B_4 = \frac{B_1\sin(\alpha_n L) - B_3\sinh(\alpha_n L)}{\cos(\alpha_n L) + \cosh(\alpha_n L)}, \tag{A.10}$$

and

$$B_1 = B_3 \frac{1 + \cos(\alpha_n L)\cosh(\alpha_n L) + \sin(\alpha_n L)\sinh(\alpha_n L)}{1 + \cos(\alpha_n L)\cosh(\alpha_n L) - \sin(\alpha_n L)\sinh(\alpha_n L)}. \quad (A.11)$$

The last boundary condition, Eq. (A.8), reveals the characteristic equation

$$1 + \cos(\alpha_n L)\cosh(\alpha_n L) = \frac{EI\alpha_n}{K_t}[\cosh(\alpha_n L)\sin(\alpha_n L) - \cos(\alpha_n L)\sinh(\alpha_n L)]. \quad (A.12)$$

Taking $B_3 = 1$, the basis functions for the spring-hinged beam are

$$\psi_n(x) = \frac{C_1 + C_2}{1 + \cos(\alpha_n L)\cosh(\alpha_n L) - \sin(\alpha_n L)\sinh(\alpha_n L)}. \quad (A.13)$$

where

$$\begin{aligned} C_1 = {} & \sin(\alpha_n x) + \sinh(\alpha_n x) \\ & + \cosh(\alpha_n L)\left(\cosh(\alpha_n x)\sin(\alpha_n L) - \sin\big(\alpha_n(L - x)\big)\right. \\ & \left. + \cos(\alpha_n L)\sinh(\alpha_n x)\right) \end{aligned} \quad (A.14)$$

and

$$C_2 = \sinh(\alpha_n L)\left(\cos\big(\alpha_n(L - x)\big) - \cos(\alpha_n L)\cosh(\alpha_n x) - \sin(\alpha_n L)\sinh(\alpha_n x)\right). \quad (A.15)$$

Appendix B

MATLAB Code

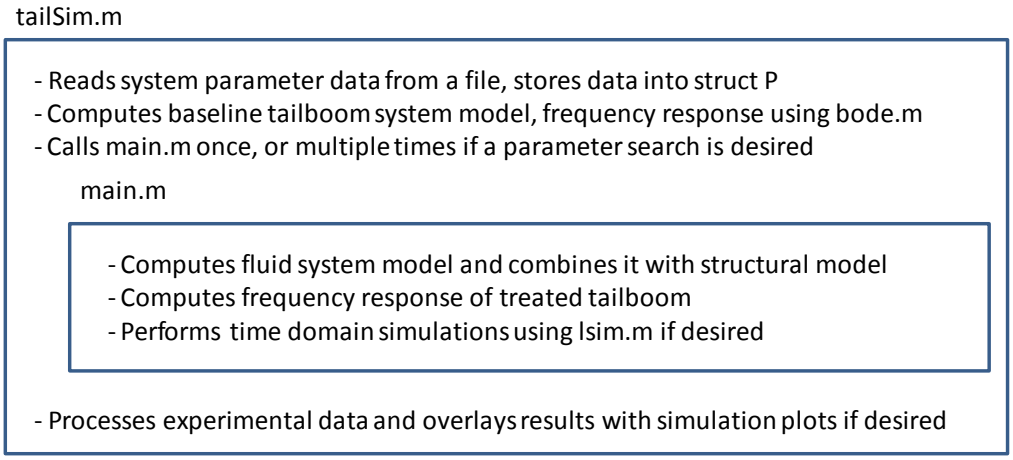

Figure B-1. Overview of the MATLAB code structure.

```
function tailSim
% tailSim.m
% For simulating a a helicopter tailboom with fluidic devices integrated
% into the structure. Performs parameter sweeps on main.m and plots
% the resulting frequecy responses.
%
% Parameters may be passed onto main.m through the cell array sweepParam.
% If a parameter, such as inertance, is specified below, then main.m will
% run using this overridden inertance value rather than the one specified
% in the F2MC tube data spreadsheet.
%
% If an override value is not specified, then main.m will use the values
% specified in the F2MC tube data spreadsheet by default.
%
% Kentaro Miura
% Mechatronics Research Laboratory
% Vertical Lift Research Center of Excellence
% Pennsylvania State University
%
% Last updated: 13 February, 2016

clc
clear all
close all

%% System configuration.
% Choose system to simulate (full-scale tailboom or experiment):
% scaleTB= 'full';
scaleTB= 'OH-58';
% scaleTB= 'small';
```

```
% Choose whether to account for frequency-dependence of fluid resistance
% and inertance:
res= 'const';
% res= 'freqDep';

% Choose whether to estimate the inertance required to tune an absorber
% (1: estimate 0: don't estimate):
inertEst= 1;

% Choose whether to optimize resistance (1: optimize 0: don't optimize):
resOpt= 0;

% Choose tube type:
% tubeType= 'F2MC';
tubeType= 'McKibben';

% Choose bladder type:
if (strcmp(tubeType,'McKibben'))
%     bladderType= 'unobtainium';
    bladderType= '1/32 soft rubber';
%     bladderType= 'piston';
%     bladderType= '1/16 soft rubber';
%     bladderType= '1/8 soft rubber';
%     bladderType= '1/16 durable rubber';
%     bladderType= '1/16 soft PVC';
%     bladderType= '1/16 hard PVC';
end

% Choose system type:
sysType= 'Fluidlastic';
% sysType= 'Tailboom';

% Choose tube and track configuration:
config= 'coupled';
% config= 'uncoupled';
% config= 'coupled tunable';

% Choose whether to compute mode shape (1: compute 0: don't compute):
modeShape= 0;
modeShape2= 0;
responseShapeFrequency= 5.5;

% Choose whether to debug tailboom structure mass:
massDebug= 0;

%% Experimental data processing setup.
% Specify whether an accelerometer or laser vibrometer was used:
measurement= 'laser';
% measurement= 'accelerometer';

% Specify number of data points over which to perform sliding window
% filtering for experimental data (use -1 to use no averaging):
avgWindow= -1;

% Sensor amplifier gains:
gainAccel= 100;
gainVibe= 1;
gainLoad= 1;

% Add path in which the data files are stored:
addpath('C:\Users\Kentaro\Dropbox\Data Files\4-10-2015')
```

```
% Specify names of files to be read:
filename{1}= 'empty with foam at valve.lvm';
filename{2}= 'open.lvm';
% filename{3}= 'r8.lvm';
% filename{4}= 'open.lvm';

%% Initial setup.
% Initialize variable override vector:
sweepParam= cell(1,28);
sweepParam(:)={-1};

% Initialize vector to record first two variables to be overridden:
overrideVec= [0 0];
overrideInd= 1;

%% Define overriding parameters.

% F2MC tube attachment points ---------------------------------------- %
% x1= 0;
% x2= linspace(0.2,0.25,10);
% x2= 0.3;
% x2= [10.25*0.0254 0.4064];

% F2MC tube material properties -------------------------------------- %
% E11= ;
% E22= ;
% v12= ;
% v23= ;
% G12= ;
% G23= ;

% F2MC tube geometry ------------------------------------------------- %
% wallThickness= [1 1]'*linspace(0.002,0.006,10);
% wallThickness= [1 1]'*[.003 .004];
% innerRadius= [1 1]'*linspace(0.005,0.01,5);
% innerRadius= [1 1]'*[.0127 .014];

% Inertia track ------------------------------------------------------ %
% portLength= linspace(2.5,2.7,10);
% portRadius= ;

% Fiber angle -------------------------------------------------------- %
% fiberAngle= linspace(1,50,51);

% Fluid properties --------------------------------------------------- %
% waterModulus= ;
% waterDensity= ;
% viscosity= ;
% reynoldsNumber= ;

% Accumulator -------------------------------------------------------- %
% capacitance= ;

% Inertance ---------------------------------------------------------- %
% inertance= linspace(8.53e7,1.02e8,5);

% Orifice Resistance ------------------------------------------------- %
% orfRes= 0;

% # of Tubes Per Side ------------------------------------------------ %
```

```
% numTubePairs= [2 4];

%% Override F2MC variables as needed.
switch config
    case {'coupled','uncoupled'}
        if (exist('x1','var'))
            sweepParam{1}= x1;
            overrideVec(overrideInd)= 1;
            overrideInd= overrideInd+1;
        end
        if (exist('x2','var'))
            sweepParam{2}= x2;
            overrideVec(overrideInd)= 2;
            overrideInd= overrideInd+1;
        end
        if (exist('E11','var'))
            sweepParam{3}= E11;
            overrideVec(overrideInd)= 3;
            overrideInd= overrideInd+1;
        end
        if (exist('E22','var'))
            sweepParam{4}= E22;
            overrideVec(overrideInd)= 4;
            overrideInd= overrideInd+1;
        end
        if (exist('v12','var'))
            sweepParam{5}= v12;
            overrideVec(overrideInd)= 5;
            overrideInd= overrideInd+1;
        end
        if (exist('v23','var'))
            sweepParam{6}= v23;
            overrideVec(overrideInd)= 6;
            overrideInd= overrideInd+1;
        end
        if (exist('G12','var'))
            sweepParam{7}= G12;
            overrideVec(overrideInd)= 7;
            overrideInd= overrideInd+1;
        end
        if (exist('G23','var'))
            sweepParam{8}= G23;
            overrideVec(overrideInd)= 8;
            overrideInd= overrideInd+1;
        end
        if (exist('wallThickness','var'))
            sweepParam{9}= wallThickness;
            overrideVec(overrideInd)= 9;
            overrideInd= overrideInd+1;
        end
        if (exist('innerRadius','var'))
            sweepParam{10}= innerRadius;
            overrideVec(overrideInd)= 10;
            overrideInd= overrideInd+1;
        end
        if (exist('outerRadius','var'))
            sweepParam{11}= outerRadius;
            overrideVec(overrideInd)= 11;
            overrideInd= overrideInd+1;
        end
        if (exist('portLength','var'))
```

```
            sweepParam{12}= portLength;
            overrideVec(overrideInd)= 12;
            overrideInd= overrideInd+1;
        end
        if (exist('portRadius','var'))
            sweepParam{13}= portRadius;
            overrideVec(overrideInd)= 13;
            overrideInd= overrideInd+1;
        end
        if (exist('fiberAngle','var'))
            sweepParam{14}= fiberAngle;
            overrideVec(overrideInd)= 14;
            overrideInd= overrideInd+1;
        end
        if (exist('waterModulus','var'))
            sweepParam{15}= waterModulus;
            overrideVec(overrideInd)= 15;
            overrideInd= overrideInd+1;
        end
        if (exist('waterDensity','var'))
            sweepParam{16}= waterDensity;
            overrideVec(overrideInd)= 16;
            overrideInd= overrideInd+1;
        end
        if (exist('viscosity','var'))
            sweepParam{17}= viscosity;
            overrideVec(overrideInd)= 17;
            overrideInd= overrideInd+1;
        end
        if (exist('capacitance','var'))
            sweepParam{18}= capacitance;
            overrideVec(overrideInd)= 19;
            overrideInd= overrideInd+1;
        end
        if (exist('inertance','var'))
            sweepParam{19}= inertance;
            overrideVec(overrideInd)= 20;
            overrideInd= overrideInd+1;
        end
        if (exist('resistance','var'))
            sweepParam{20}= resistance;
            overrideVec(overrideInd)= 21;
            overrideInd= overrideInd+1;
        end
        if (exist('orfRes','var'))
            sweepParam{21}= resistance;
            overrideVec(overrideInd)= 21;
            overrideInd= overrideInd+1;
        end
        if (exist('numTubePairs','var'))
            sweepParam{22}= numTubePairs;
            overrideVec(overrideInd)= 22;
            overrideInd= overrideInd+1;
        end
        if (exist('operatingPressure','var'))
            sweepParam{23}= operatingPressure;
            overrideVec(overrideInd)= 23;
            overrideInd= overrideInd+1;
        end
    case 'coupled tunable'
        if (exist('x1','var'))
```

```
    sweepParam{1}= x1;
    overrideVec(overrideInd)= 1;
    overrideInd= overrideInd+1;
end
if (exist('x2','var'))
    sweepParam{2}= x2;
    overrideVec(overrideInd)= 2;
    overrideInd= overrideInd+1;
end
if (exist('E11','var'))
    sweepParam{3}= E11;
    overrideVec(overrideInd)= 3;
    overrideInd= overrideInd+1;
end
if (exist('E22','var'))
    sweepParam{4}= E22;
    overrideVec(overrideInd)= 4;
    overrideInd= overrideInd+1;
end
if (exist('v12','var'))
    sweepParam{5}= v12;
    overrideVec(overrideInd)= 5;
    overrideInd= overrideInd+1;
end
if (exist('v23','var'))
    sweepParam{6}= v23;
    overrideVec(overrideInd)= 6;
    overrideInd= overrideInd+1;
end
if (exist('G12','var'))
    sweepParam{7}= G12;
    overrideVec(overrideInd)= 7;
    overrideInd= overrideInd+1;
end
if (exist('G23','var'))
    sweepParam{8}= G23;
    overrideVec(overrideInd)= 8;
    overrideInd= overrideInd+1;
end
if (exist('wallThickness','var'))
    sweepParam{9}= wallThickness;
    overrideVec(overrideInd)= 9;
    overrideInd= overrideInd+1;
end
if (exist('innerRadius','var'))
    sweepParam{10}= innerRadius;
    overrideVec(overrideInd)= 10;
    overrideInd= overrideInd+1;
end
if (exist('outerRadius','var'))
    sweepParam{11}= outerRadius;
    overrideVec(overrideInd)= 11;
    overrideInd= overrideInd+1;
end
if (exist('portLength1','var'))
    sweepParam{12}= portLength;
    overrideVec(overrideInd)= 12;
    overrideInd= overrideInd+1;
end
if (exist('portRadius1','var'))
    sweepParam{13}= portRadius;
```

```
    overrideVec(overrideInd)= 13;
    overrideInd= overrideInd+1;
end
if (exist('portLength1','var'))
    sweepParam{14}= portLength;
    overrideVec(overrideInd)= 14;
    overrideInd= overrideInd+1;
end
if (exist('portRadius1','var'))
    sweepParam{15}= portRadius;
    overrideVec(overrideInd)= 15;
    overrideInd= overrideInd+1;
end
if (exist('fiberAngle','var'))
    sweepParam{16}= fiberAngle;
    overrideVec(overrideInd)= 16;
    overrideInd= overrideInd+1;
end
if (exist('waterModulus','var'))
    sweepParam{17}= waterModulus;
    overrideVec(overrideInd)= 17;
    overrideInd= overrideInd+1;
end
if (exist('waterDensity','var'))
    sweepParam{18}= waterDensity;
    overrideVec(overrideInd)= 18;
    overrideInd= overrideInd+1;
end
if (exist('viscosity','var'))
    sweepParam{19}= viscosity;
    overrideVec(overrideInd)= 19;
    overrideInd= overrideInd+1;
end
if (exist('capacitance','var'))
    sweepParam{20}= capacitance;
    overrideVec(overrideInd)= 20;
    overrideInd= overrideInd+1;
end
if (exist('inertance1','var'))
    sweepParam{21}= inertance;
    overrideVec(overrideInd)= 21;
    overrideInd= overrideInd+1;
end
if (exist('resistance1','var'))
    sweepParam{22}= resistance;
    overrideVec(overrideInd)= 22;
    overrideInd= overrideInd+1;
end
if (exist('orfRes1','var'))
    sweepParam{23}= resistance;
    overrideVec(overrideInd)= 23;
    overrideInd= overrideInd+1;
end
if (exist('inertance2','var'))
    sweepParam{24}= inertance;
    overrideVec(overrideInd)= 24;
    overrideInd= overrideInd+1;
end
if (exist('resistance2','var'))
    sweepParam{25}= resistance;
    overrideVec(overrideInd)= 25;
```

```
            overrideInd= overrideInd+1;
        end
        if (exist('orfRes2','var'))
            sweepParam{26}= resistance;
            overrideVec(overrideInd)= 26;
            overrideInd= overrideInd+1;
        end
        if (exist('numTubePairs','var'))
            sweepParam{27}= numTubePairs;
            overrideVec(overrideInd)= 27;
            overrideInd= overrideInd+1;
        end
        if (exist('operatingPressure','var'))
            sweepParam{28}= operatingPressure;
            overrideVec(overrideInd)= 28;
            overrideInd= overrideInd+1;
        end
end

%% Initialize system parameter struct.

% Pack up simulation parameters:
P= struct;
P.Simulation.scaleTB= scaleTB;
P.Simulation.config= config;
P.Simulation.tubeType= tubeType;
P.Simulation.inertEst= inertEst;
P.Simulation.res= res;
P.Simulation.modeShape= modeShape;
P.Tube.bladderType= bladderType;
P.SysType= sysType;
P.Data.measurement= measurement;
P.Data.avgWindow= avgWindow;
P.Data.filename= filename;
P.Data.gainAccel= gainAccel;
P.Data.gainVibe= gainVibe;
P.Data.gainLoad= gainLoad;
P.Simulation.debug= massDebug;

% Initialize beam parameters:
P= initializeTailboom(P);

% If F2MC tubes exist, initialize those:
P= initializeF2MC(P,sweepParam);

% Define frequency span [Rad/s]:
fmin= P.Frequency.fmin*2*pi;
fmax= P.Frequency.fmax*2*pi;

fres= P.Frequency.fres;
w= linspace(fmin,fmax,ceil((fmax-fmin)*fres/(2*pi)));
P.Frequency.w= w;

% Initialize system basis functions:
if (modeShape2)
    x_temp= [0 8 16 24 32 49 68.5 88 107.5 142.5 150]*0.0254;

    modeShapeVec= zeros(size(x_temp));
    P_temp= P;

    for i= 1:length(x_temp)
```

```
        outLoc_temp= x_temp(i);
        P_temp.Simulation.outLoc= outLoc_temp;

        % Compute beam system matrices for temporary output location:
        P_temp= sysInit(P_temp);
        [mag ph]= bode(P_temp.beamSys_v,2*pi*responseShapeFrequency);
        modeShapeVec(i)= mag*sin(ph);
    end
    fig1= figure;
    P.Figure.fig1= fig1;
    plot(x_temp,-modeShapeVec)

    % Compute slope difference between F2MC tube attachment points:
    slopeVec= diff(modeShapeVec)./diff(x_temp);
    rootSlope= slopeVec(1);
    endSlope= interp1(x_temp(1:end-1),slopeVec,P_temp.Tube.length);
    slopeDiff= endSlope-rootSlope;

    fprintf('Theoretical Slope Difference: %g\n\n',slopeDiff)

    if (0)
        fig2= figure;
        magSlope= diff(magBeamPlot)./diff(x_temp);
        plot(x_temp(1:end-1),magSlope/(max(abs(magSlope))))
        P.Figure.fig2= fig2;
    end
end

P= sysInit(P);
P= computeBeamFRF(P,P.beamSys_v,P.Frequency.w);

%% Iterate through parameter values to peform sweep.
% Initialize timing:
tStart= tic;

% Determine the length of each sweep vector:
if (overrideInd>1)
    [~,nI]= size(sweepParam{overrideVec(1)});
end
if (overrideInd==3)
    [~,nJ]= size(sweepParam{overrideVec(2)});
elseif (overrideInd>3)
    fprintf('Error: Cannot vary more than two parameters simultaneously.')
    return
elseif (overrideInd==2)
    nJ= 1;
else
    nI= 1; nJ= 1;
end

% Sweep through each overriden parameter value:
if (overrideInd~=1)
    for i= 1:nI
        for j= 1:nJ
            % Temporarily modify sweepParam to be inputted into main.m:
            sweepParamsIn= sweepParam;

            % Extract ith entry of the i-variable:
            iVar= sweepParam{overrideVec(1)};
            sweepParamsIn{overrideVec(1)}= iVar(:,i)';
```

```
            % Extract jth entry of the j-variable:
            if (overrideInd>=3)
                jVar= sweepParam{overrideVec(2)};
                sweepParamsIn{overrideVec(2)}= jVar(:,j)';
            end

            % Feed overridden parameters into main.m:
            P.Var1Index= i;  P.Var2Index= j;
            trialNum= (j-1)*nI + i;
            P.trialNum= trialNum;
            P= updateF2MC(P,sweepParamsIn);

            switch resOpt
                case 1
                    P= resistanceOptimizer(P);
                case 0
                    P= main(P,0);
                otherwise
                    error('Select resOpt')
            end

            % Print sweep status to command window:
            statusUpdate(i,j,nI,nJ,tStart)
        end
    end
else
    % Feed parameters into main.m:
    P.trialNum= 1;

    % Simulate system:
    tic
    switch resOpt
        case 1
            P= resistanceOptimizer(P);
        case 0
            if (strcmp(sysType,'Fluidlastic'))
                P= main(P);
            else
                P= convertUnits(P);
            end
        otherwise
            error('Select resOpt')
    end
    toc

    % Plot the resulting frequency response:
    plotFRF(P,3)

    switch scaleTB
        case 'full'
        case 'small'
            if ((P.TFmode<4 || P.TFmode>8))
                % Plot experimental data for comparison:
                experimentalDataProcessor(P)
            end
        case 'OH-58'
                 testData58(P)
    end
end

end
```

```
%% List of subfunctions
% --------------------------------------------------------------------- %
% Determine timestamp string
function time_string= sec2hms(t)
% Convert timestamp from seconds to hours-minutes-seconds format.
% INPUT  1:  t - Time [s].
% OUTPUT 1:  time_string  - String with properly formatted time.

% --------------------------------------------------------------------- %

% Determine the number of hours, minutes, and seconds:
numHours = floor(t/3600);
numMins = floor((t-3600*numHours)/60);
numSecs = round(t-3600*numHours-60*numMins);

% Format hours:
if (numHours<10)
    hourString = ['0' num2str(numHours)];
else
    hourString = num2str(numHours);
end

% Format minutes:
if (numMins<10)
    minString = ['0' num2str(numMins)];
else
    minString = num2str(numMins);
end

% Format seconds:
if (numSecs<10)
    secString= ['0' num2str(numSecs)];
else
    secString= num2str(numSecs);
end

time_string = [hourString ':' minString ':' secString];
end
% --------------------------------------------------------------------- %
% Display iteration status update
function statusUpdate(i,j,nI,nJ,tStart)
% Print a status update with estimated simulation time remaining.
% INPUT  1:  i - First indexing variable.
% INPUT  2:  j - Second indexing variable.
% INPUT  3: nI - Number of iterations for the first index variable.
% INPUT  4: nJ - Number of iterations for the second index variable.
% INPUT  5: tStart - Time at which simulation was started.

% --------------------------------------------------------------------- %
% Compute the total elapsed time so far:
tElapsTotal= toc(tStart);

% Total number of iterations completed:
nTotal= nJ*(i-1)+j;

% Average time spent per iteration:
avgTime= tElapsTotal/nTotal;

% Number of iterations remaining:
nRem= nI*nJ-nTotal;
```

```
% Estimated time remaining based on average iteration time:
estRem= avgTime*nRem;

% Reformat time into hms format:
tElapsTotal= sec2hms(tElapsTotal);
estRem= sec2hms(estRem);

% Print simulation status:
text1= ['Time Elapsed: ' tElapsTotal];
text2= ['  Estimated Time Remaining: ' estRem];
text3= ['  i= ' num2str(i) ' j= ' num2str(j) '\n\n'];
fprintf([text1 text2 text3])
end
% ------------------------------------------------------------------------- %
% Extract Beam data
function P= initializeTailboom(P)
% Initialize tailboom properties and store them into struct P.
% INPUT  1:  Struct P.
% OUTPUT 1:  Struct P with all fields populated.

% ------------------------------------------------------------------------- %
% Unpack variables:
scaleTB= P.Simulation.scaleTB;

% Determine which spreadsheets to read:
switch scaleTB
    case 'full'
        spreadName= 'FS';
    case 'OH-58'
        spreadName= 'OH';
    case 'small'
        spreadName= 'SS';
    otherwise
        error('Please select tailboom scale.\n')
end
fileName= 'System.xlsx';

% Extract simulation parameters. Remove NaNs:
sysProps= xlsread(fileName,['SIM-' spreadName],'D2:D102');
sysProps(isnan(sysProps))= [];

% Extract tailboom parameters. Remove NaNs:
tailProps= xlsread(fileName,['TB-' spreadName],'D2:D102');
tailProps(isnan(tailProps))= [];

% Extract stringer parameters. Remove NaNs:
switch P.Simulation.scaleTB
    case 'full'
        stringProps= xlsread(fileName,['TB-' spreadName],'D22:DD24');
        stringProps(any(isnan(stringProps),2),:)= [];
    case 'OH-58'
        stringProps= xlsread(fileName,['TB-' spreadName],'D22:DD24');
        stringProps(any(isnan(stringProps),2),:)= [];
    case 'small'
        stringProps= xlsread(fileName,['TB-' spreadName],'D28:D30');
        stringProps(any(isnan(stringProps),2),:)= [];
    otherwise
        error('Select a tailboom scale.')
end
```

```
% Simulation parameters ------------------------------------------------- %
% Output units:
sp1= sysProps(1);
switch sp1
    case 1
        P.Simulation.units= 'SI';
    case 2
        P.Simulation.units= 'English';
end

% Simulation output axial position [m]:
P.Simulation.outLoc= sysProps(2);

% Dimension of basis function space:
P.Simulation.nDimensions= sysProps(3);

% Numerical differentiation step size:
P.Simulation.stepSize= sysProps(4);

% Transfer function to compute.
P.TFmode= sysProps(5);

% Frequency response parameters ----------------------------------------- %
% Minimum frequency [Hz]:
P.Frequency.fmin= sysProps(6);

% Maximum frequency [Hz]:
P.Frequency.fmax= sysProps(7);

% Frequency resolution [1/Hz]:
P.Frequency.fres= sysProps(8);

% Output frequency:
P.Frequency.outFreq= [num2str(sysProps(9)) '/rev'];

% Tailboom properties --------------------------------------------------- %
% Tailboom length [m]:
P.Beam.length= tailProps(1);

% Tailboom damping coefficient:
P.Beam.dampingRatio= tailProps(2);

% Thickness of tailboom skin [m]:
P.Beam.skinThickness= tailProps(3);

switch scaleTB
    case 'full'
        % Tailboom geometry ------------------------------------------- %
        % Ratio of tailboom tip radius to base radius:
        P.Beam.taper= tailProps(4);

        % Radius of tailboom at base [m]:
        P.Beam.baseRadius= tailProps(5);

        % Vertical loading properties --------------------------------- %
        % Vertical load on tailboom:
        P.Beam.verticalLoad= tailProps(6);

        % Axial position of vertical load [m]:
        P.Beam.verticalLoadPos= tailProps(7);
```

```
        % Masses ------------------------------------------------------- %
        % Tail rotor mass [kg]:
        P.Beam.massRotor= tailProps(8);

        % Tail rotor axial position [m]:
        P.Beam.rotorPosition= tailProps(9);

        % Horizontal stabilizer mass [kg]:
        P.Beam.horizStabMass= tailProps(10);

        % Horizontal stabilizer position [m]:
        P.Beam.horizStabPosition= tailProps(11);

        % Concentrated mass [kg]:
        P.Beam.mass= tailProps(12);

        % Concentrated mass position [m]:
        P.Beam.massPosition= tailProps(13);

        % Driveshaft mass [kg/m]:
        P.Beam.driveShaftMass= tailProps(14);

        % Tailboom stringer properties ---------------------------------- %
        % Radial position of each stringer center [rad]:
        P.Stringer.radialPosition= stringProps(1,2:end);

        % Number of stringers:
        P.Stringer.number= length(P.Stringer.radialPosition);

        % Cross-sectional area of each stringer [m]:
        P.Stringer.area= stringProps(2,2:end);

        % "Radius" of each stringer [m]:
        P.Stringer.radius= stringProps(3,2:end);
    case 'OH-58'
        % Tailboom geometry ---------------------------------------------- %
        % Ratio of tailboom tip radius to base radius:
        P.Beam.taper= tailProps(4);

        % Radius of tailboom at base [m]:
        P.Beam.baseRadius= tailProps(5);

        % Vertical loading properties ----------------------------------- %
        % Vertical load on tailboom:
        P.Beam.verticalLoad= tailProps(6);

        % Axial position of vertical load [m]:
        P.Beam.verticalLoadPos= tailProps(7);

        % Masses ------------------------------------------------------- %
        % Tail rotor mass [kg]:
        P.Beam.massRotor= tailProps(8);

        % Tail rotor axial position [m]:
        P.Beam.rotorPosition= tailProps(9);

        % Horizontal stabilizer mass [kg]:
        P.Beam.horizStabMass= tailProps(10);

        % Horizontal stabilizer position [m]:
        P.Beam.horizStabPosition= tailProps(11);
```

```
        % Concentrated mass [kg]:
        P.Beam.mass= tailProps(12);

        % Concentrated mass position [m]:
        P.Beam.massPosition= tailProps(13);

        % Driveshaft mass [kg/m]:
        P.Beam.driveShaftMass= tailProps(14);

        % Tailboom stringer properties ---------------------------------- %
        % Radial position of each stringer center [rad]:
        P.Stringer.radialPosition= stringProps(1,2:end);

        % Number of stringers:
        P.Stringer.number= length(P.Stringer.radialPosition);

        % Cross-sectional area of each stringer [m]:
        P.Stringer.area= stringProps(2,2:end);

        % "Radius" of each stringer [m]:
        P.Stringer.radius= stringProps(3,2:end);
    case 'small'
        % Tailboom geometry ---------------------------------------------- %
        % Beam taper ratio (width):
        P.Beam.taperWidth= tailProps(4);

        % Beam taper ratio (height):
        P.Beam.taperHeight= tailProps(5);

        % Beam base width [m]:
        P.Beam.baseWidth= tailProps(6);

        % Beam base height [m]:
        P.Beam.baseHeight= tailProps(7);

        % Vertical loading properties ---------------------------------- %

        % Vertical load on tailboom:
        P.Beam.verticalLoad= tailProps(8);

        % Axial position of vertical load [m]:
        P.Beam.verticalLoadPos= tailProps(9);

        % Masses ------------------------------------------------------- %
        % Vertical stabilizer point mass [kg]:
        P.Beam.vertStabMass= tailProps(10);

        % Vertical stabilizer tube length [m]:
        P.Beam.vertStabLength= tailProps(11);

        % Vertical stabilizer tube mass [kg]:
        P.Beam.vertStabTubeMass= tailProps(12);

        % Horizontal stabilizer mass [kg]:
        P.Beam.horizStabMass= tailProps(12);

        % Horizontal stabilizer side mass [kg]:
        P.Beam.horizStabMass= tailProps(13);

        % Horizontal stabilizer side mass [kg]:
```

```
        P.Beam.horizStabTubeMass= tailProps(14);

        % Stack mass [kg]:
        P.Beam.stackMass= tailProps(15);

        % Stack mass position [m]:
        P.Beam.stackMassLoc= tailProps(16);

        % Rear plate mass [kg]:
        P.Beam.tailPlateMass= tailProps(17);

        % Tuning springs ------------------------------------------------ %
        % Linear spring constant [N/m]:
        P.Beam.k= tailProps(18);

        % Torsional spring constant [N/rad]:
        P.Beam.Kt= tailProps(19);

        % Tailboom stringer properties ---------------------------------- %
        P.Stringer.thickness= stringProps(1);
        P.Stringer.cornerWidth= stringProps(2);
        P.Stringer.sideWidth= stringProps(3);
end

% Axial positions over which simulation takes place:
P.Beam.x= ...
    linspace(0,P.Beam.length,(1/P.Simulation.stepSize)*P.Beam.length+1);
end
% ------------------------------------------------------------------------- %
% Extract F2MC data
function P= initializeF2MC(P,sweepParam)
% Initialize F2MC tube properties and store them into struct P.
% INPUT  1:  Struct P.
% INPUT  2:  Cell array containing parameters through which to sweep.
% OUTPUT 1:  Struct P with all fields populated.

% ------------------------------------------------------------------------- %
% Unpack variables:
config= P.Simulation.config;
scaleTB= P.Simulation.scaleTB;
tubeType= P.Simulation.tubeType;

% Determine which spreadsheet to read.
switch config
    case 'coupled'
        configName= 'C';
    case 'uncoupled'
        configName= 'UC';
    case 'coupled tunable'
        configName= 'CT';
    otherwise
        error('Please select F2MC configuration.\n')
end

switch scaleTB
    case 'full'
        spreadName= 'FS';
    case 'OH-58'
        spreadName= 'OH';
    case 'small'
        spreadName= 'SS';
```

```
    otherwise
        error('Please select tailboom scale.\n')
end

fileName= 'System.xlsx';

% Extract F2MC tube data from spreadsheet. Remove NaNs.
tubeProps= xlsread(fileName,[tubeType '-' spreadName '-' configName]);
if (size(tubeProps,2)>1)
    tubeProps(:,2:3)= [];
    tubeProps(any(isnan(tubeProps),2),:)= [];
    switch configName
        case {'C','UC'}
            % F2MC tube positions ---------------------------------------- %

            % Fuselage-end positions of tubes [m]:
            if (sweepParam{1}<0)
                P.Tube.x1= tubeProps(1,2:end);
            else
                P.Tube.x1= sweepParam{1};
            end

            % Tail-end positions of tubes [m]:
            if (sweepParam{2}<0)
                P.Tube.x2= tubeProps(2,2:end);
            else
                P.Tube.x2= sweepParam{2};
            end

            % Material properties ---------------------------------------- %

            if (sweepParam{3}<0)
                P.Tube.E11= tubeProps(3,2:end);
            else
                P.Tube.E11= sweepParam{3};
            end

            if (sweepParam{4}<0)
                P.Tube.E22= tubeProps(4,2:end);
            else
                P.Tube.E22= sweepParam{4};
            end

            if (sweepParam{5}<0)
                P.Tube.v12= tubeProps(5,2:end);
            else
                P.Tube.v12= sweepParam{5};
            end

            if (sweepParam{6}<0)
                P.Tube.v23= tubeProps(6,2:end);
            else
                P.Tube.v23= sweepParam{6};
            end

            if (sweepParam{7}<0)
                P.Tube.G12= tubeProps(7,2:end);
            else
                P.Tube.G12= sweepParam{7};
            end
```

```
if (sweepParam{8}<0)
    P.Tube.G23= tubeProps(8,2:end);
else
    P.Tube.G23= sweepParam{8};
end

% Tube geometry ------------------------------------------- %

% Tube thickness [m]:
if (sweepParam{9}<0)
    P.Tube.thickness= tubeProps(9,2:end);
else
    P.Tube.thickness= sweepParam{9};
end

% Tube inner radius [m]:
if (sweepParam{10}<0)
    P.Tube.innerRadius= tubeProps(10,2:end);
else
    P.Tube.innerRadius= sweepParam{10};
end

% Tube outer radius [m]:
if (sweepParam{11}<0)
    P.Tube.outerRadius= tubeProps(11,2:end);
else
    P.Tube.outerRadius= sweepParam{11};
end

% Inertia track length [m]:
if (sweepParam{12}<0)
    P.Tube.portLength= tubeProps(12,2:end);
else
    P.Tube.portLength= sweepParam{12};
end

% Inertia track radius [m]:
if (sweepParam{13}<0)
    P.Tube.portRadius= tubeProps(13,2:end);
else
    P.Tube.portRadius= sweepParam{13};
end

% Tube fiber winding angle [deg]:
if (sweepParam{14}<0)
    P.Tube.fiberAngle= tubeProps(14,2:end);
else
    P.Tube.fiberAngle= sweepParam{14};
end

% Working fluid properties -------------------------------- %

% Water bulk modulus [Pa]:
if (sweepParam{15}<0)
    P.Tube.fluidModulus= tubeProps(15,2:end);
else
    P.Tube.fluidModulus= sweepParam{15};
end

% Water density [kg*m^-3]:
if (sweepParam{16}<0)
```

```
            P.Tube.fluidDensity= tubeProps(16,2:end);
        else
            P.Tube.fluidDensity= sweepParam{16};
        end

        % Water viscosity [Pa*s]:
        if (sweepParam{17}<0)
            P.Tube.viscosity= tubeProps(17,2:end);
        else
            P.Tube.fiberAngle= sweepParam{17};
        end

        % Fluid capacitance [m^4-s^2/kg]:
        if (sweepParam{18}<0)
            P.Tube.capacitance= tubeProps(18,2:end);
        else
            P.Tube.capacitance= sweepParam{18};
        end

        % Fluid inertance [kg/m^4]:
        if (sweepParam{19}<0 && sweepParam{12}<0 && sweepParam{11}<0)
            P.Tube.inertance= tubeProps(19,2:end);
        else
            area= pi*P.Tube.portRadius^2;
            P.Tube.inertance= P.Tube.fluidDensity*sweepParam{12}/area;
        end

        % Fluid resistance [kg/s-m^3]:
        P.Tube.resistance= tubeProps(20,2:end);

        % Orifice resistance [kg/s-m^3]:
        if (sweepParam{21}<0)
            P.Tube.orificeResistance= tubeProps(21,2:end);
        else
            P.Tube.orificeResistance= sweepParam{21};
        end

        % Number of tubes per side:
        if (sweepParam{22}<0)
            P.Tube.numTubePairs= tubeProps(22,2:end);
        else
            P.Tube.numTubePairs= sweepParam{22};
        end

        % Operating pressure:
        if (sweepParam{23}<0 && strcmp(tubeType,'McKibben'))
            P.Tube.operatingPressure= tubeProps(23,2:end);
        elseif (strcmp(tubeType,'McKibben'))
            P.Tube.operatingPressure= sweepParam{23};
        end
    case 'CT'
        % F2MC tube positions ---------------------------------------- %

        % Fuselage-end positions of tubes [m]:
        if (sweepParam{1}<0)
            P.Tube.x1= tubeProps(1,2:end);
        else
            P.Tube.x1= sweepParam{1};
        end

        % Tail-end positions of tubes [m]:
```

```
if (sweepParam{2}<0)
    P.Tube.x2= tubeProps(2,2:end);
else
    P.Tube.x2= sweepParam{2};
end

% Material properties ------------------------------------- %

if (sweepParam{3}<0)
    P.Tube.E11= tubeProps(3,2:end);
else
    P.Tube.E11= sweepParam{3};
end

if (sweepParam{4}<0)
    P.Tube.E22= tubeProps(4,2:end);
else
    P.Tube.E22= sweepParam{4};
end

if (sweepParam{5}<0)
    P.Tube.v12= tubeProps(5,2:end);
else
    P.Tube.v12= sweepParam{5};
end

if (sweepParam{6}<0)
    P.Tube.v23= tubeProps(6,2:end);
else
    P.Tube.v23= sweepParam{6};
end

if (sweepParam{7}<0)
    P.Tube.G12= tubeProps(7,2:end);
else
    P.Tube.G12= sweepParam{7};
end

if (sweepParam{8}<0)
    P.Tube.G23= tubeProps(8,2:end);
else
    P.Tube.G23= sweepParam{8};
end

% Tube geometry ------------------------------------------- %

% Tube thickness [m]:
if (sweepParam{9}<0)
    P.Tube.thickness= tubeProps(9,2:end);
else
    P.Tube.thickness= sweepParam{9};
end

% Tube inner radius [m]:
if (sweepParam{10}<0)
    P.Tube.innerRadius= tubeProps(10,2:end);
else
    P.Tube.innerRadius= sweepParam{10};
end

% Tube outer radius [m]:
```

```
if (sweepParam{11}<0)
    P.Tube.outerRadius= tubeProps(11,2:end);
else
    P.Tube.outerRadius= sweepParam{11};
end

% Inertia track 1 length [m]:
if (sweepParam{12}<0)
    P.Tube.portLength1= tubeProps(12,2:end);
else
    P.Tube.portLength1= sweepParam{12};
end

% Inertia track 1 radius [m]:
if (sweepParam{13}<0)
    P.Tube.portRadius1= tubeProps(13,2:end);
else
    P.Tube.portRadius1= sweepParam{13};
end

% Inertia track 2 length [m]:
if (sweepParam{14}<0)
    P.Tube.portLength2= tubeProps(14,2:end);
else
    P.Tube.portLength2= sweepParam{14};
end

% Inertia track 2 radius [m]:
if (sweepParam{15}<0)
    P.Tube.portRadius2= tubeProps(15,2:end);
else
    P.Tube.portRadius2= sweepParam{15};
end

% Tube fiber winding angle [deg]:
if (sweepParam{16}<0)
    P.Tube.fiberAngle= tubeProps(16,2:end);
else
    P.Tube.fiberAngle= sweepParam{16};
end

% Working fluid properties ---------------------------------- %

% Water bulk modulus [Pa]:
if (sweepParam{17}<0)
    P.Tube.fluidModulus= tubeProps(17,2:end);
else
    P.Tube.fluidModulus= sweepParam{17};
end

% Water density [kg*m^-3]:
if (sweepParam{18}<0)
    P.Tube.fluidDensity= tubeProps(18,2:end);
else
    P.Tube.fluidDensity= sweepParam{18};
end

% Water viscosity [Pa*s]:
if (sweepParam{19}<0)
    P.Tube.viscosity= tubeProps(19,2:end);
else
```

```
            P.Tube.fiberAngle= sweepParam{19};
        end

        % Fluid capacitance [m^4-s^2/kg]:
        if (sweepParam{20}<0)
            P.Tube.capacitance= tubeProps(20,2:end);
        else
            P.Tube.capacitance= sweepParam{20};
        end

        % Fluid inertance 1 [kg/m^4]:
        if (sweepParam{21}<0 && sweepParam{12}<0 && sweepParam{11}<0)
            P.Tube.inertance1= tubeProps(21,2:end);
        else
            area= pi*P.Tube.portRadius^2;
            P.Tube.inertance1= P.Tube.fluidDensity*sweepParam{12}/area;
        end

        % Fluid resistance 1 [kg/s-m^3]:
        P.Tube.resistance1= tubeProps(22,2:end);

        % Orifice resistance 1 [kg/s-m^3]:
        if (sweepParam{23}<0)
            P.Tube.orificeResistance1= tubeProps(23,2:end);
        else
            P.Tube.orificeResistance1= sweepParam{23};
        end

        % Fluid inertance 2 [kg/m^4]:
        if (sweepParam{24}<0 && sweepParam{14}<0 && sweepParam{11}<0)
            P.Tube.inertance2= tubeProps(24,2:end);
        else
            area= pi*P.Tube.portRadius^2;
            P.Tube.inertance2= P.Tube.fluidDensity*sweepParam{14}/area;
        end

        % Fluid resistance 2 [kg/s-m^3]:
        P.Tube.resistance2= tubeProps(25,2:end);

        % Orifice resistance 2 [kg/s-m^3]:
        if (sweepParam{26}<0)
            P.Tube.orificeResistance2= tubeProps(26,2:end);
        else
            P.Tube.orificeResistance2= sweepParam{26};
        end

        % Number of tubes per side:
        if (sweepParam{27}<0)
            P.Tube.numTubePairs= tubeProps(27,2:end);
        else
            P.Tube.numTubePairs= sweepParam{27};
        end

        % Operating pressure:
        if (sweepParam{28}<0 && strcmp(tubeType,'McKibben'))
            P.Tube.operatingPressure= tubeProps(28,2:end);
        elseif (strcmp(tubeType,'McKibben'))
            P.Tube.operatingPressure= sweepParam{28};
        end
end
```

```
    % Initialize result storage vector:
    P.Results.irVec= [];
    P.Results.x2Vec= [];
    P.Results.sysMassVec= [];
    P.Results.firstModeDampingVec= [];
    P.Results.secondModeDampingVec= [];
end
end
% ------------------------------------------------------------------------ %
% Update F2MC data as needed
function P= updateF2MC(P,sweepParam)
% Update F2MC tube properties and store them into struct P as needed.
% INPUT  1:  P - Empty struct P.
% OUTPUT 1:  P - Struct containing tailboom parameters.

% F2MC tube positions ----------------------------------------------------%

% Fuselage-end positions of tubes [m]:
if (sweepParam{1}>=0)
    P.Tube.x1= sweepParam{1};
end

% Tail-end positions of tubes [m]:
if (sweepParam{2}>=0)
    P.Tube.x2= sweepParam{2};
end

% Material properties ----------------------------------------------------%

if (sweepParam{3}>=0)
    P.Tube.E11= sweepParam{3};
end

if (sweepParam{4}>=0)
    P.Tube.E22= sweepParam{4};
end

if (sweepParam{5}>=0)
    P.Tube.v12= sweepParam{5};
end

if (sweepParam{6}>=0)
    P.Tube.v23= sweepParam{6};
end

if (sweepParam{7}>=0)
    P.Tube.G12= sweepParam{7};
end

if (sweepParam{8}>=0)
    P.Tube.G23= sweepParam{8};
end

% Tube geometry ----------------------------------------------------------%

% Tube thickness [m]:
if (sweepParam{9}>=0)
    P.Tube.thickness= sweepParam{9};
end

% Tube inner radius [m]:
```

```
if (sweepParam{10}>=0)
    P.Tube.innerRadius= sweepParam{10};
end

% Tube outer radius [m]:
if (sweepParam{11}>=0)
    P.Tube.outerRadius= sweepParam{11};
end

switch config
    case {'coupled','uncoupled'}
        % Port length [m]:
        if (sweepParam{12}>=0)
            P.Tube.portLength= sweepParam{12};
        end

        % Port radius [m]:
        if (sweepParam{13}>=0)
            P.Tube.portRadius= sweepParam{13};
        end

        % Tube fiber winding angle [deg]:
        if (sweepParam{14}>=0)
            P.Tube.fiberAngle= sweepParam{14};
        end

        % Working fluid properties ------------------------------------ %

        % Water bulk modulus [Pa]:
        if (sweepParam{15}>=0)
            P.Tube.fluidModulus= sweepParam{15};
        end

        % Water density [kg*m^-3]:
        if (sweepParam{16}>=0)
            P.Tube.fluidDensity= sweepParam{16};
        end

        % Water viscosity [Pa*s]:
        if (sweepParam{17}>=0)
            P.Tube.fiberAngle= sweepParam{17};
        end

        % Fluid capacitance [m^4-s^2/kg]:
        if (sweepParam{18}>=0)
            P.Tube.capacitance= sweepParam{18};
        end

        % Fluid inertance [kg/m^4]:
        if (sweepParam{12}>=0 || sweepParam{13}>=0)
            area= pi*P.Tube.portRadius^2;
            P.Tube.inertance= P.Tube.fluidDensity*sweepParam{12}/area;
        end

        % Orifice resistance [kg/s-m^3]:
        if (sweepParam{21}>=0)
            P.Tube.orificeResistance= sweepParam{21};
        end

        % # of tubes per setup:
        if (sweepParam{22}>=0)
```

```
            P.Tube.numTubePairs= sweepParam{22};
        end
    case 'coupled tunable'
        % Port length 1 [m]:
        if (sweepParam{12}>=0)
            P.Tube.portLength1= sweepParam{12};
        end

        % Port radius 1 [m]:
        if (sweepParam{13}>=0)
            P.Tube.portRadius1= sweepParam{13};
        end

        % Port length 2 [m]:
        if (sweepParam{14}>=0)
            P.Tube.portLength1= sweepParam{14};
        end

        % Port radius 2 [m]:
        if (sweepParam{15}>=0)
            P.Tube.portRadius1= sweepParam{15};
        end

        % Tube fiber winding angle [deg]:
        if (sweepParam{16}>=0)
            P.Tube.fiberAngle= sweepParam{16};
        end

        % Working fluid properties ------------------------------------- %

        % Water bulk modulus [Pa]:
        if (sweepParam{17}>=0)
            P.Tube.fluidModulus= sweepParam{17};
        end

        % Water density [kg*m^-3]:
        if (sweepParam{18}>=0)
            P.Tube.fluidDensity= sweepParam{18};
        end

        % Fluid viscosity [Pa*s]:
        if (sweepParam{19}>=0)
            P.Tube.fiberAngle= sweepParam{19};
        end

        % Fluid capacitance [m^4-s^2/kg]:
        if (sweepParam{20}>=0)
            P.Tube.capacitance= sweepParam{20};
        end

        % Fluid inertance 1[kg/m^4]:
        if (sweepParam{12}>=0 || sweepParam{13}>=0)
            area= pi*P.Tube.portRadius1^2;
            P.Tube.inertance1= P.Tube.fluidDensity*sweepParam{12}/area;
        end

        % Fluid resistance 1[kg/m^4]:
        if (sweepParam{12}>=0 || sweepParam{13}>=0)
            P.Tube.resistance1= 8*P.Tube.fluidViscosity*sweepParam{12}...
                /(pi*P.Tube.portRadius1^4);
        end
```

```
        % Orifice resistance 1[kg/s-m^3]:
        if (sweepParam{23}>=0)
            P.Tube.orificeResistance1= sweepParam{23};
        end

        % Fluid inertance 2[kg/m^4]:
        if (sweepParam{14}>=0 || sweepParam{15}>=0)
            area= pi*P.Tube.portRadius2^2;
            P.Tube.inertance1= P.Tube.fluidDensity*sweepParam{14}/area;
        end

        % Fluid resistance 2[kg/m^4]:
        if (sweepParam{14}>=0 || sweepParam{15}>=0)
            P.Tube.resistance1= 8*P.Tube.fluidViscosity*sweepParam{14}...
                /(pi*P.Tube.portRadius2^4);
        end

        % Orifice resistance 2[kg/s-m^3]:
        if (sweepParam{26}>=0)
            P.Tube.orificeResistance2= sweepParam{26};
        end

        % # of tubes per setup:
        if (sweepParam{27}>=0)
            P.Tube.numTubePairs= sweepParam{27};
        end
end
end
% ------------------------------------------------------------------------ %
% Compute basis functions, tailboom system, and tailboom FRF
function P= sysInit(P)
% Compute basis functions, tailboom model, and compute tailboom frequency
% response.
% INPUT  1:  P - Empty struct P.
% OUTPUT 1:  P - Struct containing tailboom parameters.

% ------------------------------------------------------------------------ %
% Unpack variables:
scaleTB= P.Simulation.scaleTB;
outerRadius= P.Tube.outerRadius;
skinThickness= P.Beam.skinThickness;
x= P.Beam.x;

% Compute basis function values ------------------------------------------ %
% This block of code computes basis functions that are used throughout the
% tailboom simulation code package. The resulting vectors are then stored
% in struct P to avoid making redundant calculations.

% Compute eigenvalues for hinge-springed boundary condition:
if (strcmp('small',scaleTB) || strcmp('OH-58',scaleTB))
    % Compute eigenvalues:
    a= linspace(0,10,1000);
    kn= zeros(size(a));

    for j= 1:length(a)
        kn(j)= fzero(@(a) charEqn(a,P),a(j));
    end
    kn= unique(kn);
    kn= sort(kn);
```

```
    % Set tolerance for numerical differences:
    kn= round(1e9*kn)*1e-9;
    kn= unique(kn);
    kn(kn<0)= [];

    % Store eigenvalues in struct P:
    P.Simulation.kn= kn;
end

% Compute tube effective length [m]:
switch scaleTB
    case 'full'
        P.Tube.length= P.Tube.x2-P.Tube.x1;
    case 'OH-58'
        P.Tube.length= P.Tube.x2-P.Tube.x1 - 5*.0254;
    case 'small'
        if ((P.Tube.x2 - P.Tube.x1)==10.25*0.0254)
            P.Tube.length= 4.5*.0254;
        elseif ((P.Tube.x2 - P.Tube.x1)==22*0.0254)
            P.Tube.length= (22-7)*0.0254;
        elseif ((P.Tube.x2 - P.Tube.x1)==13*0.0254)
            P.Tube.length= 6*0.0254;
        else
            error('Tube length must be defined.\n\n')
        end
end

% Compute initial tube volume [m^3]:
P.Tube.initialVolume= pi*(P.Tube.innerRadius.^2).*P.Tube.length;

% Count the number of F2MC tubes:
nTubes= length(P.Tube.capacitance);
P.Tube.nTubes= nTubes;

% Unpack variables and initialize:
outLoc= P.Simulation.outLoc;
N= P.Simulation.nDimensions;
psi_L_prime= zeros(P.Simulation.nDimensions,1);
[psi_L,psi_Out]= deal(zeros(P.Simulation.nDimensions,1));
[d21,d21Psi21p]= deal(zeros(N,nTubes));
d= cell(nTubes,1);
[psi_0_doublePrime,psi_0_triplePrime]= ...
    deal(zeros(P.Simulation.nDimensions,1));

% Compute F2MC tube strain:
for k= 1:nTubes
    % Determine x-span of each F2MC tube:
    xUP= x(x>=P.Tube.x1(k));
    xTube= xUP(xUP<=P.Tube.x2(k));
    P.Tube.xTube= xTube;

    switch scaleTB
        case 'full'
            effRadius= outerRadius(k)+skinThickness;
            d{k,1}= beamRadius(P,xTube)-effRadius;
        case 'OH-58'
            d{k,1}= beamRadius(P,xTube);
        case 'small'
            d{k,1}= beamHeight(P,xTube)/2+1*.0254;
    end
    P.Tube.offset= d;
```

```
end

% Iterate through each dimension and tube to populate basis function
% vectors:
for j=1:P.Simulation.nDimensions
    psi_L(j)= Psi(P.Beam.length,j,P);
    psi_Out(j)= Psi(outLoc,j,P);
    psi_L_prime(j)= d1Psi(P.Beam.length,j,P);

    psi_0_doublePrime(j)= d2Psi(0,j,P);
    psi_0_triplePrime(j)= d3Psi(0,j,P);
    for k= 1:nTubes
        d21(j,k)= integ(j,k,'T',P);
        d2= d{k,1};
        d1= d2(1);
        d2= d2(end);
        d21Psi21p(j,k)= d2*d1Psi(xTube(end),j,P)-d1*d1Psi(xTube(1),j,P);
    end
end

% Basis function values at L (N x 1):
P.Basis.psi_L= psi_L;
P.Basis.psi_L_prime= psi_L_prime;

% Basis function values at desired output location (N x 1):
P.Basis.psi_Out= psi_Out;

% Basis function values at 0 (N x 1):
P.Basis.psi_0_doublePrime= psi_0_doublePrime;
P.Basis.psi_0_triplePrime= psi_0_triplePrime;

% eps_21= {d21}'*{q} (N x 1):
P.Basis.d21= d21/P.Tube.length;
P.Basis.d21Psi21p= d21Psi21p;

% EI(0), EI'(0):
P.EI_0= E(P,0)*Izz(P,0);

% Compute tailboom system and frequency response ----------------------- %
% The system analyzed consists of two subsystems: the tailboom and the F2MC
% tubes. This segment of code computes the tailboom (beam) state space
% model, and uses it to compute its frequency response.
[M_v,K_v,C_v,F_v,P]= computeBeamMatrices(P);
P.Beam.M_v= M_v; P.Beam.K_v=K_v; P.Beam.C_v= C_v; P.Beam.F_v= F_v;
[P.Abeam_v,P.Bbeam_v,P.Cbeam_v,P.Dbeam_v]= ...
    beam2StateSpace(P);
P.beamSys_v= ss(P.Abeam_v,P.Bbeam_v,P.Cbeam_v,P.Dbeam_v);
end
% ------------------------------------------------------------------------ %
% Compute beam mode shapes
function calcBeamModeShape(M,Phi,s,figNum,modeNum,P,plotType)
% Compute and plot the beam's mode shapes over position vector s.
% INPUT  1:  Mass matrix M.
% INPUT  2:  Mass-normalized modal matrix Phi.
% INPUT  3:  Position vector s.
% INPUT  4:  Figure number.
% INPUT  5:  Mode number of interest.
% INPUT  7:  Struct containing all passing parameters.
% INPUT  8:  Plot type. Displacement if 'disp', slope if 'slope'.
% OUTPUT 1:  N/A
```

```
% ------------------------------------------------------------------------ %
% Compute basis functions:
[N,~]= size(M);
psi= zeros(N,length(s));

switch plotType
    case 'disp'
        for j= 1:N
            psi(j,:)= Psi(s,j,P);
        end
    case 'slope'
        for j= 1:N
            psi(j,:)= d1Psi(s,j,P);
        end
end

% Construct mode shape:
Phi= Phi(:,modeNum);

y= Phi'*psi;

% Normalize response:
if (modeNum==1)
    y= y/max(abs(y));
end

% Approximate slope in rads/N:
figure(figNum)
plot(s,y)
xlabel('x [m]')
ylabel('w [m/N]')
end
% ------------------------------------------------------------------------ %
% Plot frequency response
function plotFRF(P,figNum)
% Subfunction for plotting beam and fluidlastic tailboom FRFs.
% INPUT  1:  P - Data structure containing parameter values. Populated by
% main() on the initial run, and then fed back into main() to avoid
% re-reading parameters from System_Data.xlsx.
%
% OUTPUT 1:  N/A
% ------------------------------------------------------------------------ %

% Unpack variables:
TFmode= P.TFmode;
scaleTB= P.Simulation.scaleTB;
w= P.Frequency.w;

figure(figNum)
if (TFmode<4 || TFmode>8)
    magBeam= P.Frequency.magBeam;
    semilogx(w/(2*pi),20*log10(magBeam),'b-','linewidth',2)
    hold on
end

if (isfield(P.Frequency,'magBeamTube'))
    magBeamTube= P.Frequency.magBeamTube;
end

% Plot F2MC-beam frequency response:
switch P.SysType
```

```
    case 'Fluidlastic'
        switch TFmode
            case {4,5,6,7,8,10}
                loglog(w/(2*pi),magBeamTube,'r--','linewidth',2)
            otherwise
                semilogx(w/(2*pi),20*log10(magBeamTube),'r:','linewidth',2)
                legStr= {'Tailboom';'F^2MC-Tailboom'};
                legend(legStr)
        end
    case 'Tailboom'
        legStr= {'Baseline Model'; 'Experiment'};
        legend(legStr)
end

% Tailor frequency response plots based on transfer function type:
y_str= P.Plot.ylabel;

set(gca, 'fontSize',16)
set(findall(gcf,'type','text'),'fontSize',16)
ylabel(y_str)
xlim([w(1) w(end)]/(2*pi))
xlabel('Frequency [Hz]')

switch scaleTB
    case 'small'
        set(gca,'XTick',[5 7 9 11 13 15 20 25])
    case 'full'
        set(gca,'XTick',[3 4 5 6 7 8 9])
    case 'OH-58'
        set(gca,'XTick',[1 2 5 8 12 20 30 50])
end
end
% ------------------------------------------------------------------------ %
% Process and plot experimental data
function experimentalDataProcessor(P)
% Process and plot experimental data.
% INPUT  1:  P - Empty struct P.

% ------------------------------------------------------------------------ %
% Sensor data:
% Accelerometer: PCB Model 353B02
% Sensitivity (+-15%): 20mV/g, 2.04mV/(m/s^2)
% http://www.pcb.com/Products.aspx?m=353B02#.UuAhqRAo5ph

% Load cell: PCB Model 208C02
% Sensitivity (+-15%): 50mV/lbf, 11241mV/kN
% http://www.pcb.com/Products.aspx?m=208C02#.UuAhlBAo5ph

% Vibrometer: Polytech OFV 5000
% Sensitivity: 640 micron/V

% Sensor amplifier gains:
gainAccel= P.Data.gainAccel;
gainVibe= P.Data.gainVibe;
gainLoad= P.Data.gainLoad;

% Sensor sensitivity:
senAccel= 2.04/1000; % V per m-s^-2
senLoad= 11241/1000000; % V/N
senVibe= (1e6)/640; % V/m
```

```
% Unpack variables:
TFmode= P.TFmode;
units= P.Simulation.units;
measurement= P.Data.measurement;
avgWindow= P.Data.avgWindow;
filename= P.Data.filename;

% Count the number of data files to be plotted:
[~,n]= size(filename);

% Initialize frequency, phase, and magnitude data cell arrays:
[freq,phas,mags]= deal(cell(n,1));

% Process data files:
for i= 1:n
    % Read the i-th data file:
    a= dlmread(filename{i});

    % Unpack data.
    % Col 1: frequency (Hz)
    % Col 2: amplitude (V/V)
    % Col 3: phase (deg)
    f= a(:,1); mag= a(:,2); ph= a(:,3);

    % Scale measurements according to sensor:
    switch measurement
        case 'accelerometer'
            mag= mag*gainLoad/gainAccel;
            mag= mag*senLoad/senAccel; % m-s^-2/N

            % Convert acceleration to displacement:
            if (TFmode==1)
                w= f*2*pi;
                mag= mag./(w.^2);
            end
        case 'laser'
            if (strcmp(filename{i},'empty with foam at valve.lvm'))
                mag= mag*gainLoad/gainVibe*10;
            elseif (strcmp(filename{i},'open.lvm'))
                mag= mag*gainLoad/gainVibe*10;
            else
                mag= mag*gainLoad/gainVibe;
            end
            mag= mag*senLoad/senVibe;
        otherwise
            error('Select a vibration measurement device.')
    end

    % Sliding window average:
    if (avgWindow<=0)
        magAvg= mag;
    else
        magAvg= filter(ones(1,avgWindow)/avgWindow,1,mag);
    end

    % Trim data:
    upF= 40; lowF= 1;
    magAvg= magAvg(f<=upF & f>=lowF);
    ph= ph(f<=upF & f>=lowF);
    f= f(f<=upF & f>=lowF);
```

```
    % Perform unit conversions:
    if (strcmp(units,'English'))
        % Convert m/N to in/lbf:
        magAvg= 175.126835*magAvg;
    end

    freq{i}= f;
    phas{i}= ph;
    mags{i}= magAvg;
end

% Plot data
hold all
switch n
    case 1
        plot(freq{1},20*log10(mags{1}))
    case 2
        plot(freq{1},20*log10(mags{1}),freq{2},20*log10(mags{2}))
    case 3
        plot(freq{1},20*log10(mags{1}),freq{2},20*log10(mags{2}),...
            freq{3},20*log10(mags{3}))
    case 4
        plot(freq{1},20*log10(mags{1}),freq{2},20*log10(mags{2}),...
            freq{3},20*log10(mags{3}),freq{4},20*log10(mags{4}))
    case 5
    case 6
        plot(freq{1},20*log10(mags{1}),freq{2},20*log10(mags{2}),...
            freq{3},20*log10(mags{3}),freq{4},20*log10(mags{4}),...
            20*log10(mags{5}),20*log10(mags{6}))
end
legend('Theoretical Baseline','Theoretical F^2MC',...
    'Experimental Baseline','Experimental Open Valve',...
    'Experimental Closed Valve')
end
% ------------------------------------------------------------------------ %
% Compute beam frequency response function
function P= computeBeamFRF(P,beamSys,w)
% Subfunction for computing tailboom frequency responses.
% INPUT  1:  P - Data structure containing parameter values.
% INPUT  2:  beamSys - State space model of the tailboom structure.
% INPUT  3:  w - Vector of frequencies at which to compute FRFs [rad/s].
%
% OUTPUT 1:  P - Data structure containing parameter values.
% ------------------------------------------------------------------------ %
magBeam= bode(beamSys,w);
magBeam= squeeze(magBeam(1,1,:));
P.Frequency.magBeam= magBeam;
end
% ------------------------------------------------------------------------ %

%% List of matrix computation subfunctions
% ------------------------------------------------------------------------ %
% Transform system into first-order form
function [A,B,C,D]= beam2StateSpace(P)
% beam2StateSpace.m
% Function converts the function space representation of the tailboom model
% into a state space representation.
% INPUT  1:  P - Struct containing all passing parameters.
% OUTPUT 1:  [A B C D] - State space representation of tailboom system
% ------------------------------------------------------------------------%
```

```
% Unpack parameters:
N= P.Simulation.nDimensions;
psi_Out= P.Basis.psi_Out;
psi_0_doublePrime= P.Basis.psi_0_doublePrime;
psi_0_triplePrime= P.Basis.psi_0_triplePrime;
EI_0= P.EI_0;

% Unpack beam matrices:
Mbeam= P.Beam.M_v; Kbeam= P.Beam.K_v;
Cbeam= P.Beam.C_v; Fbeam= P.Beam.F_v;

switch P.Simulation.scaleTB
    case 'full'
        % Convert tailboom system to state space
        % Cantilever beam "A" matrix:
        A= [zeros(N)                    eye(N);
            -(Mbeam^-1)*Kbeam      -(Mbeam^-1)*Cbeam];

        % Cantilever beam "B" matrix:
        B= [zeros(N,1);
            (Mbeam^-1)*Fbeam];
        % Cantilever beam "C" matrix:
        switch P.TFmode
            case {1,4,5,6,7,8,10}
                % Tip displacement
                C= [psi_Out' zeros(1,N)];
                D= 0;
            case 2
                % Root moment
                C= [-EI_0*psi_0_doublePrime' zeros(1,N)];
                D= 0;
            case 3
                % Root shear
                c11= -EI_0*psi_0_triplePrime'-EI_0p*psi_0_doublePrime';
                C= [c11 zeros(1,N)];
                D= 0;
            case 9
                % Tip acceleration [g]
                a= zeros(1,2*N);
                a(N+1:2*N)= psi_Out';

                C= a*A;
                D= a*B;
            otherwise
                error('Choose a transfer function.')
        end
    case 'OH-58'
        % Convert tailboom system to state space
        % Cantilever beam "A" matrix:
        A= [zeros(N)                    eye(N);
            -(Mbeam^-1)*Kbeam      -(Mbeam^-1)*Cbeam];

        % Cantilever beam "B" matrix:
        B= [zeros(N,1);
            (Mbeam^-1)*Fbeam];
        % Cantilever beam "C" matrix:
        switch P.TFmode
            case {1,4,5,6,7,8,10}
                % Tip displacement
                C= [psi_Out' zeros(1,N)];
                D= 0;
```

```
            case 2
                % Root moment
                C= [-EI_0*psi_0_doublePrime' zeros(1,N)];
                D= 0;
            case 3
                % Root shear
                c11= -EI_0*psi_0_triplePrime'-EI_0p*psi_0_doublePrime';
                C= [c11 zeros(1,N)];
                D= 0;
            case 9
                % Tip acceleration [g]
                a= zeros(1,2*N);
                a(N+1:2*N)= psi_Out';

                C= a*A;
                D= a*B;
            otherwise
                error('Choose a transfer function.')
        end
    case 'small'
        % Unpack parameters
        [N,~]= size(Mbeam);

        % Convert tailboom system to state space
        % Cantilever beam "A" matrix:
        A= [zeros(N)                    eye(N);
            -(Mbeam^-1)*Kbeam      -(Mbeam^-1)*Cbeam];

        % Cantilever beam "B" matrix:
        B= [zeros(N,1);
            (Mbeam^-1)*Fbeam];

        % Cantilever beam "C" matrix:
        switch P.TFmode
            case {1,4,5,6,7,8,10}
                % Tip displacement
                C= zeros(1,2*N);
                C(1:length(psi_Out))= psi_Out;
                D= 0;
            case 2
                % Root moment
                C= [-EI_0*psi_0_doublePrime' zeros(1,N)];
                D= 0;
            case 3
                % Root shear
                c11= -EI_0*psi_0_triplePrime'-EI_0p*psi_0_doublePrime';
                C= [c11 zeros(1,N)];
                D= 0;
            case 9
                % Tip acceleration [g]
                a= zeros(1,2*N);
                a(N+1:2*N)= [psi_Out; 0; 0]';

                C= a*A;
                D= a*B;
            otherwise
                error('Choose a transfer function.')
        end
    otherwise
        error('Select a tailboom scale.')
end
```

```
end
% ------------------------------------------------------------------------ %
% Compute Beam Matrices
function [M,K,C,F,P]= computeBeamMatrices(P)
% computeBeamMatrices.m
% Function computing system matrices of a helicopter tailboom.
% INPUT  1:  P - Struct containing all passing parameters.
% OUTPUT 1: [M K C F] - Function space representation of tailboom
% ------------------------------------------------------------------------%

% Compute beam system matrices

% Forcing vector:
F= force(P);

% Inertia matrix:
[M,P]= mass(P);

% Stiffness matrix:
[K,P]= stiff(P);
P.Beam.K_orig= K;

switch P.Simulation.scaleTB
    case 'full'
        Meff= M;
        Keff= K;
        Feff= F;
    case 'OH-58'
        Meff= M;
        Keff= K;
        Feff= F;
    case 'small'
        % Unpack variables:
        Psi_L= P.Basis.psi_L;
        Psi_Lp= P.Basis.psi_L_prime;
        m1= P.Beam.m1;
        m2= P.Beam.m2;
        m3= P.Beam.m3;
        R= P.Beam.R;

        % Initialize mass and stiffness matrices, and forcing vector:
        N= P.Simulation.nDimensions;
        [Meff,Keff]= deal(zeros(N+2,N+2));
        Feff= zeros(N+2,1);

        % Populate mass matrix:
        Meff(1:N,1:N)= M;
        Meff(N+1,N+1)= m2;
        Meff(N+2,N+2)=(m1+m3/4)*R^2;
        Meff(1:N,N+2)= R*cosd(69.18)*Psi_L*(m1+m3/2);
        Meff(N+2,1:N)= Meff(1:N,N+2)';

        % Define spring tuning parameters:
        k= P.Beam.k;
        Kt= P.Beam.Kt;

        % Populate stiffness matrix:
        Keff(1:N,1:N)= K + Kt*(Psi_Lp*Psi_Lp') + k*(Psi_L*Psi_L');
        Keff(N+1,N+1)= k;
        Keff(N+2,N+2)= Kt;
        Keff(1:N,N+1)= -k*Psi_L;
```

```
        Keff(1:N,N+2)= -Kt*Psi_Lp;
        Keff(N+1,1:N)= Keff(1:N,N+1)';
        Keff(N+2,1:N)= Keff(1:N,N+2)';

        % Populate forcing matrix:
        Feff(1:N,1)= F;
    otherwise
        error('Select tailboom scale.')
end

% Compute eigensolutions:
[Phi,eigMat]= eig(Keff,Meff);

% Damping matrix:
if (P.Beam.dampingRatio==-1)
    Ceff= damping(P);
else
    % Impose uniform damping for all modes:
    CModal= 2*P.Beam.dampingRatio*sqrt(eigMat);
    Ceff= ((Phi')^-1)*CModal*(Phi^-1);
end

% Pack variables:
M= Meff;
C= Ceff;
K= Keff;
F= Feff;

% Compute mode shapes if desired:
if (P.Simulation.modeShape)
    modePos= linspace(0,P.Beam.length,200);
    calcBeamModeShape(M,Phi,modePos,1,1,P,'disp')
    hold all
    calcBeamModeShape(M,Phi,modePos,1,2,P,'disp')

    % Compute mode shape slopes:
    calcBeamModeShape(M,Phi,modePos,2,1,P,'slope')
    hold all
    calcBeamModeShape(M,Phi,modePos,2,2,P,'slope')
end

end
% ------------------------------------------------------------------------ %
% Compute Inertia Matrix
function [mass_out,P]= mass(P)
% Function computing entries for the tailboom mass matrix.
% INPUT  1:  P - Struct containing all passing parameters.
% OUTPUT 1: mass_out - NxN mass matrix.

% ------------------------------------------------------------------------ %

% Unpack parameters:
N= P.Simulation.nDimensions;

% Initialize mass matrix:
mass_out= zeros(N,N);

switch P.Simulation.scaleTB
    case 'full'
        % Unpack parameters:
        x_r= P.Beam.rotorPosition;
```

```
    % Compute each entry of the mass matrix:
    for a= 1:N
        for n= a:N
            m_r= P.Beam.massRotor;
            m_s= P.Beam.horizStabMass;
            x_s= P.Beam.horizStabPosition;

            % Integrate mass properties:
            if (a==1 && n==1 && P.Simulation.debug==0)
                [integral,P]= integ(a,n,'M',P);
            elseif (P.Simulation.debug==1)
                % If desired, compute tailboom mass and exit program
                % for debugging purposes:
                mStruct= 2.2*integ(a,n,'M',P);

                mD= 2.2*P.Beam.driveShaftMass*P.Beam.length;
                m_dead= P.Beam.mass;

                fprintf('TB skin + stringer mass: %g lb\n',mStruct)
                fprintf('TB drive shaft mass: %g lb\n',mD)
                fprintf('TB rotor mass: %g lb\n',2.2*m_r)
                fprintf('TB horiz. stabilizer mass: %g lb\n',2.2*m_s)
                fprintf('TB unmodeled mass: %g lb\n',2.2*m_dead)

                m_tot= mStruct+mD+2.2*(m_r+m_s+m_dead);
                fprintf('TB total mass: %g lb\n',m_tot)
                error('Debugging tailboom mass!')
            else
                integral= integ(a,n,'M',P);
            end

            % Compute effect of rotor mass:
            rotorMass= m_r*PsiJN(x_r,a,n,P);

            % Compute effect of horizontal stabilizer:
            stabMass= m_s*PsiJN(x_s,a,n,P);

            % Compute effect of unmodeled structural mass:
            m_d= P.Beam.mass;
            x_d= P.Beam.massPosition;
            deadMass= m_d*PsiJN(x_d,a,n,P);

            % Compute rotational inertia of vertical stabilizer:
            % Fins extend about 42in on top and bottom. They are shaped
            % like parallelograms.
            r_fin= 21*.0254; % Estimate of vertical fin centroid [m].
            I_r= 2*(15/2.2)*r_fin^2;
            mRot= I_r*d1PsiJN(x_r,a,n,P);

            % Sum all mass components:
            mass_out(a,n)=integral+rotorMass+mRot+stabMass+deadMass;

            % Duplicate off-diagonal terms (symmetry).
            mass_out(n,a)= mass_out(a,n);
        end
    end
case 'OH-58'
    % Unpack parameters:
    x_r= P.Beam.rotorPosition;
```

```
    % Compute each entry of the mass matrix:
    for a= 1:N
        for n= a:N
            m_r= P.Beam.massRotor;
            m_s= P.Beam.horizStabMass;
            x_s= P.Beam.horizStabPosition;

            % Integrate mass properties:
            if (a==1 && n==1 && P.Simulation.debug==0)
                [integral,P]= integ(a,n,'M',P);
            elseif (P.Simulation.debug==1)
                % If desired, compute tailboom mass and exit program
                % for debugging purposes:
                mStruct= 2.2*integ(a,n,'M',P);

                mD= 2.2*P.Beam.driveShaftMass*P.Beam.length;

                fprintf('TB skin + stringer mass: %g lb\n',mStruct)
                fprintf('TB drive shaft mass: %g lb\n',mD)
                fprintf('TB rotor mass: %g lb\n',2.2*m_r)
                fprintf('TB horiz. stabilizer mass: %g lb\n',2.2*m_s)

                m_tot= mStruct+mD+2.2*(m_r+m_s);
                fprintf('TB total mass: %g lb\n',m_tot)
                error('Debugging tailboom mass!')
            else
                integral= integ(a,n,'M',P);
            end

            % Compute effect of rotor mass:
            rotorMass= m_r*PsiJN(x_r,a,n,P);

            % Compute effect of horizontal stabilizer:
            stabMass= m_s*PsiJN(x_s,a,n,P);

            % Compute rotational inertia of vertical stabilizer:
            % Fins extend about 42in on top and bottom. They are shaped
            % like parallelograms.
            r_fin= 21*.0254; % Estimate of vertical fin centroid [m].
            I_r= 2*(15/2.2)*r_fin^2;
            mRot= I_r*d1PsiJN(x_r,a,n,P);

            % Sum all mass components:
            mass_out(a,n)=integral+rotorMass+1*mRot+stabMass;

            % Duplicate off-diagonal terms (symmetry).
            mass_out(n,a)= mass_out(a,n);
        end
    end
case 'small'
    % Unpack parameters:
    L= P.Beam.length;

    if (N==3)
        % Compute rotational effect of top mass:
        topMass= P.Beam.vertStabMass;
        tubeL= P.Beam.vertStabLength;
        rotTop= topMass*(tubeL+0.072591)^2;

        % Compute rotational effect of vertical tube:
        mTube= P.Beam.vertStabTubeMass;
```

```
    ITube= (1/3)*mTube*tubeL^2;
    I1= rotTop + ITube;
    m1= I1;
else
    % Compute rotational effect of top mass:
    topMass= P.Beam.vertStabMass;
    m1= topMass;
    tubeL= P.Beam.vertStabLength;
    mTube= P.Beam.vertStabTubeMass;
end

% Compute mass of side masses + horizontal tubes:
massSide= P.Beam.horizStabMass;
massSideBar= P.Beam.horizStabTubeMass;
m2= massSide+massSideBar;

% Piezo stacks:
ms= P.Beam.stackMass;
stacLoc= P.Beam.stackMassLoc;

% Tail plate mass:
mp= P.Beam.tailPlateMass;

% Compute each entry of the mass matrix:
for a= 1:N
    for n= a:N
        % Integrate mass properties:
        if (a==1 && n==1)
            [integral,P]= integ(a,n,'M',P);
        elseif (P.Simulation.debug==1)
            % If desired, compute tailboom mass and exit program
            % for debugging purposes:
            mStruct= 2.2*integ(a,n,'M',P);

            fprintf('TB skin + stringer mass: %g lb\n',mStruct)
            fprintf('TB tail plate mass: %g lb\n',2.2*mp)
            fprintf('TB stack mass: %g lb\n',2.2*ms)
            fprintf('TB stab mass: %g lb\n',2.2*(m2+mTube+topMass))

            m_tot= mStruct+2.2*(ms+m2+mp+mTube+topMass);
            fprintf('TB total mass: %g lb\n',m_tot)
            error('Debugging tailboom mass!')
        else
            integral= integ(a,n,'M',P);
        end

        % Compute stack inertia:
        stacMass= ms*PsiJN(stacLoc,a,n,P);

        % Compute tail plate inertia:
        platMass= mp*PsiJN(L,a,n,P);

        % Sum all mass components:
        if (N==3)
            % Compute vertical stabilizer inertia:
            vertMass= m1*PsiJN(L,a,n,P);
            mass_out(a,n)= integral + stacMass + platMass + vertMass;
        else
            mass_out(a,n)= integral + stacMass + platMass;
        end
        % Duplicate off-diagonal terms (symmetry).
```

```
                mass_out(n,a)= mass_out(a,n);
            end
        end

        % Pack variables:
        P.Beam.R= tubeL+0.072591;
        P.Beam.m1= m1;
        P.Beam.m2= m2;
        P.Beam.m3= mTube;
    otherwise
        error('Select a tailboom scale.')
end

end
% ------------------------------------------------------------------------ %
% Compute Damping Matrix
function [damp_out,P]= damping(P)
% Function computing entries for the tailboom damping matrix.
% INPUT  1:  P - Struct containing all passing parameters.
% OUTPUT 1:  damp_out - NxN damping matrix.

% ------------------------------------------------------------------------ %
% Unpack parameters:
N= P.Simulation.nDimensions;

% Initialize damping matrix:
damp_out= zeros(N,N);

% Compute each entry of the damping matrix:
for b= 1:N
    for n= b:N
        if (b==1 && n==1)
            [integral,P]= integ(b,n,'C',P);
        else
            integral= integ(b,n,'C',P);
        end
        damp_out(b,n)= integral;

        % Duplicate off-diagonal terms (symmetry).
        damp_out(n,b)= integral;
    end
end
end
% ------------------------------------------------------------------------ %
% Compute Stiffness Matrix
function [stiff_out,P]= stiff(P)
% Function computing entries for the tailboom stiffness matrix.
% INPUT  1:  P - Struct containing all passing parameters.
% OUTPUT 1: stiff_out - NxN stiffness matrix.

% ------------------------------------------------------------------------ %
% Unpack parameters:
N= P.Simulation.nDimensions;

% Initialize stiffness matrix:
stiff_out= zeros(N,N);

% Compute each entry of the stiffness matrix:
for c= 1:N
    for n= c:N
        if (c==1 && n==1)
```

```
            [integral,P]= integ(c,n,'K',P);
        else
            integral= integ(c,n,'K',P);
        end
        stiff_out(c,n)= integral;

        % Duplicate entires:
        stiff_out(n,c)= stiff_out(c,n);
    end
end
end
% ------------------------------------------------------------------------ %
% Compute Forcing Vector
function force_out= force(P)
% Function computing entries for the generalized forcing matrix.
% INPUT  1:  P - Struct containing all passing parameters.
% OUTPUT 1:  force_out - N x # of inputs, partial generalized forcing.

% ------------------------------------------------------------------------ %
% Unpack parameters:
F= P.Beam.verticalLoad;
N= P.Simulation.nDimensions;
x_F= P.Beam.verticalLoadPos;

% Initialize forcing vector:
force_out= zeros(N,1);

% Compute each entry of the forcing vector:
for k= 1:N
    fSum= 0;
    for i= 1:length(F)
        fSum= fSum + F(i)*Psi(x_F(i),k,P);
    end
    force_out(k)= fSum;
end
end
% ------------------------------------------------------------------------ %

%% List of parameter subfunctions
% Some system parameters are functions of position and therefore must be
% defined separately in nested functions below.

% ------------------------------------------------------------------------ %
function R_out= stringRadius(P,s,i)
% Function describing tailboom radius as a function of position
% along tailboom central axis.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  R_out - Tailboom radius at position s [m]
% ------------------------------------------------------------------------ %
rStringerBase= P.Stringer.radius(i);
taper= P.Beam.taper;
L= P.Beam.length;

if (taper~=1)
    R_out= rStringerBase*(1-(1-taper)*s/L);
else
    R_out= rStringerBase*ones(size(s));
end
end
% ------------------------------------------------------------------------ %
```

```
function R_out= beamRadius(P,s)
% Function describing tailboom radius as a function of position
% along tailboom central axis.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  R_out - Tailboom radius at position s [m]
% ------------------------------------------------------------------------ %
rBase= P.Beam.baseRadius;
taper= P.Beam.taper;
L= P.Beam.length;

if (taper~=1)
    R_out= rBase*(1-(1-taper)*s/L);
else
    R_out= rBase*ones(size(s));
end
end
% ------------------------------------------------------------------------ %
function R_out= beamHeight(P,s)
% Function describing tailboom radius as a function of position
% along tailboom central axis.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  R_out - Tailboom radius at position s [m]
% ------------------------------------------------------------------------ %
tH= P.Beam.taperHeight;
bH= P.Beam.baseHeight;
L= P.Beam.length;

if (tH~=1)
    R_out= bH*(1-(1-tH)*s/L);
else
    R_out= bH*ones(size(s));
end
end
% ------------------------------------------------------------------------ %
function R_out= beamWidth(P,s)
% Function describing tailboom radius as a function of position
% along tailboom central axis.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  R_out - Tailboom radius at position s [m]
% ------------------------------------------------------------------------ %
tW= P.Beam.taperWidth;
bW= P.Beam.baseWidth;
L= P.Beam.length;

if (tW~=1)
    R_out= bW*(1-(1-tW)*s/L);
else
    R_out= bW*ones(size(s));
end
end
% ------------------------------------------------------------------------ %
function A_out= beamArea(P,s)
% Function describing tailboom cross-sectional area as a function
% of position along tailboom central axis.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  A_out - Tailboom cross-sectional area at s [m^2]
% ------------------------------------------------------------------------ %
```

```
switch P.Simulation.scaleTB
    case 'full'
        t= P.Beam.skinThickness;

        % Compute cross-sectional area of cone:
        coneArea= pi*t*(2*beamRadius(P,s)-t);

        % Compute stringer area:
        A_stringers= 0;
        numStringers= length(P.Stringer.area);
        for i= 1:numStringers
            A_stringers= A_stringers + pi*stringRadius(P,s,i).^2;
        end

        % Compute sum of stringer and skin areas:

        % Make sure to update length in the area moment calculations too.
        A_stringers(s>=12*.0254)= 0;
        A_out= A_stringers + coneArea;
    case 'OH-58'
        t= P.Beam.skinThickness;

        % Compute cross-sectional area of cone:
        coneArea= pi*t*(2*beamRadius(P,s)-t);

        % Compute stringer area:
        A_stringers= 0;
        numStringers= length(P.Stringer.area);
        for i= 1:numStringers
            A_stringers= A_stringers + pi*stringRadius(P,s,i).^2;
        end

        % Compute sum of stringer and skin areas:

        % Make sure to update length in the area moment calculations too.
        A_stringers(s>=12*.0254)= 0;
        A_out= A_stringers + coneArea;
    case 'small'
        % Unpack variables:
        t= P.Beam.skinThickness;
        tS= P.Stringer.thickness;
        cW= P.Stringer.cornerWidth;
        sW= P.Stringer.sideWidth;

        % Compute cross-sectional area of skin:
        skinOut= beamWidth(P,s).*beamHeight(P,s);
        skinIn= (beamWidth(P,s)-2*t).*(beamHeight(P,s)-2*t);
        skinArea= skinOut-skinIn;

        % Compute stringer area:
        cornerArea= cW^2 - (cW-tS)^2;
        sideArea= sW*tS;

        stringArea= 4*cornerArea + 4*sideArea;

        % Compute sum of stringer and skin areas:
        A_out= skinArea + stringArea;
    otherwise
        error('Choose a tailboom scale.')
end
```

```
end
% ------------------------------------------------------------------------ %
function E_out= E(~,~)
% Function describing tailboom material modulus as a function of
% position along tailboom central axis.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  E_out - Tailboom Young's modulus at position s [GPa]
% ------------------------------------------------------------------------ %

% Elastic modulus of aluminum:
E_out= 70e9;
end
% ------------------------------------------------------------------------ %
function I_out= Izz(P,s)
% Function describing tailboom z-moment of inertia as a function of
% position along tailboom central axis. For lateral flexure.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  I_out - Tailboom moment of inertia [kg*m^2]
% ------------------------------------------------------------------------ %

switch P.Simulation.scaleTB
    case 'full'
        % Compute skin moment of inertia:
        % Beam skin thickness:
        t= P.Beam.skinThickness;

        % Stringer radial position:
        stiffAng= P.Stringer.radialPosition;

        % Skin area moment of inertia:
        I_skin= 0.25*pi*(beamRadius(P,s).^4 - (beamRadius(P,s)-t).^4);

        % Compute stringer moment of inertia:
        if (isempty(stiffAng))
            I_stiff= 0;
        else
            % Calculate distance of each stringer center from the neutral
            % axis of the beam:
            numStringers= length(P.Stringer.area);
            I_stiff= 0;
            for i= 1:numStringers
                % Calculate ith stringer area:
                A_stringer= pi*stringRadius(P,s,i).^2;

                % Calculate radial distance of ith stringer:
                radi= beamRadius(P,s)-t-stringRadius(P,s,i)/2;

                % Calculate offset distnace of ith stringer:
                offDist= radi*sin(stiffAng(i));

                % Area moment of the ith stringer about its center:
                I_center_stiff= 0.25*pi*stringRadius(P,s,i).^4;

                I_stiff= I_stiff + I_center_stiff+A_stringer.*offDist.^2;
            end
        end

        % Make sure to update length in the area calculations too.
        I_stiff(s>=12*.0254)= 0;
```

```
        I_out= sum(I_stiff,1)' + I_skin';
    case 'OH-58'
        % Compute tailboom area moment.
        E0= E(0);
        EI_out= zeros(size(s));

        % OH-58C representative data:
        removed

        % Convert units:
        EIzz= EIzz*4.44822162*0.00064516;

        for j= 1:length(s)
            sj= s(j);
            EI_out(j)= EIzz(length(x(sj>=x)));
        end

        I_out= EI_out'/E0;
    case 'small'
        % Unpack variables:
        t= P.Beam.skinThickness;
        tS= P.Stringer.thickness;
        cW= P.Stringer.cornerWidth;
        sW= P.Stringer.sideWidth;

        % Compute skin moment of inertia:
        I_skinOut= (beamWidth(P,s).*(beamHeight(P,s).^3))/12;
        I_skinIn= (beamWidth(P,s)-2*t).*((beamHeight(P,s)-2*t).^3)/12;
        I_skin= I_skinOut - I_skinIn;

        % Side stringers:
        I_side= tS*(sW^3)/12;

        % Top stringers:
        I_top= sW*(tS^3)/12 + (tS*sW)*(0.5*beamHeight(P,s)-tS/2-t).^2;

        % Corner stringers:
        a= cW;
        A_corn= tS*(2*a-tS);
        Cy= (a^2+a*tS-tS^2)/(2*(2*a-tS));
        y= a-Cy;
        I_corn_0= (1/3)*(tS*y^3 + a*(a-y)^3 - (a-tS)*(a-y-tS)^3);
        I_corn= I_corn_0 + A_corn*(0.5*beamHeight(P,s)-t-Cy).^2;

        % Assemble stringers + skin:
        I_string= 2*I_side + 2*I_top + 4*I_corn;
        I_out= I_string' + I_skin';

    otherwise
        error('Choose a tailboom scale.')
end
end
% ------------------------------------------------------------------------- %
function rho_out= rho(~,~)
% Function describing tailboom density as a function of position
% along tailboom central axis.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  s - Position along tailboom [m]
% OUTPUT 1:  rho_out - Tailboom density at position s [kg*m^-3]
% ------------------------------------------------------------------------- %
```

```
% Density of aluminum:
rho_out= 2700;

end
% ------------------------------------------------------------------------ %

%% List of lower-level computation subfunctions
% ------------------------------------------------------------------------ %
% Numerical Integrator
function [integ_out,P]= integ(j,n,mat,P)
% Computes integrals to determine entries for inertia, damping, and
% stiffness matrices.
% INPUT  1:  j - Integer indexing value.
% INPUT  2:  n - Integer indexing value.
% INPUT  3:  mat - Char specifying which integrand to compute:
%               'M' == Inertia matrix
%               'K' == Stiffness matrix
%               'C' == Damping matrix
%               'T' == Compute F2MC tube strain
% INPUT  4:  P - Struct containing all passing parameters.
% OUTPUT 1:  integ_out - Resulting integral.
% OUTPUT 2:  P - Struct containing all system parameters.

% ------------------------------------------------------------------------ %
% Unpack parameters:
x= P.Beam.x;

% ------------------------------------------------------------------------ %
% Define integrals for mass matrix:
% ------------------------------------------------------------------------ %
    function y= f(s)
        switch P.Simulation.scaleTB
            case 'full'
                % Tailboom mass known, but calculation does not account for
                % structural elements like bulkheads. Assume weight
                % distribution matches that calculated.
                sff= 2.4207;

                if (~isfield(P,'tailboomStruct'))
                    P.tailboomStruct= sff*rho(P,s).*(beamArea(P,s));
                end

                % m_temp= rho*A(x):
                m_temp= P.tailboomStruct+P.Beam.driveShaftMass;

                % Mass matrix integrand:
                if (P.Simulation.debug==1)
                    y= P.tailboomStruct;
                else
                    y= PsiJN(s,j,n,P).*m_temp;
                end
            case 'OH-58'
                % Tailboom mass known, but calculation does not account for
                % structural elements like bulkheads. Assume weight
                % distribution matches that calculated.
                sff= 2.6196;

                if (~isfield(P,'tailboomStruct'))
                    P.tailboomStruct= sff*rho(P,s).*(beamArea(P,s));
                end
```

```
                % m_temp= rho*A(x):
                m_temp= P.tailboomStruct+P.Beam.driveShaftMass;

                % Mass matrix integrand:
                if (P.Simulation.debug==1)
                    y= P.tailboomStruct;
                else
                    y= PsiJN(s,j,n,P).*m_temp;
                end
            case 'small'
                if (~isfield(P,'tailboomStruct'))
                    P.tailboomStruct= rho(P,s).*(beamArea(P,s));
                end

                % Mass matrix integrand:
                if (P.Simulation.debug==1)
                    y= P.tailboomStruct;
                else
                    y= PsiJN(s,j,n,P).*P.tailboomStruct;
                end
            otherwise
                error('Select tailboom scale.')
        end
    end

% ------------------------------------------------------------------------ %
% Define integral for damping matrix:
% ------------------------------------------------------------------------ %
    function y= g(s)
        if (~isfield(P,'structDamp'))
            P.structDamp= Gam.*E(P,s).*Izz(P,s);
        end

        y= Gam(P,s).*E(P,s).*Izz(P,s).*d2PsiJN(s,j,n,P)' +...
            P.Beam.dampingCoeff*PsiJN(s,j,n,P)';
    end

% ------------------------------------------------------------------------ %
% Define integral for stiffness matrix:
% ------------------------------------------------------------------------ %
switch P.Simulation.scaleTB
    case 'full'
        % TB mass is known, but based on predicted stiffness the model
        % natural frequencies end up being too high. There must be some
        % unmodeled compliance in the structure.
        sf= 0.831;
    case 'OH-58'
        sf= 1; % EI(x) data given, no correction needed.
    case 'small'
        sf= 1;
    otherwise
        error('Select a tailboom scale.')
end
    function y= h(s)
        if (~isfield(P,'EIV'))
            % Model prediction:
            P.EIV= E(P,s).*Izz(P,s);
        end
        y= sf*P.EIV.*d2PsiJN(s,j,n,P)';
    end
```

```
% ------------------------------------------------------------------------ %
% Define integral for F2MC tube strain:
% ------------------------------------------------------------------------ %
    function y= t(s)
        y= P.Tube.offset{n}.*d2Psi(s,j,P);
    end

% ------------------------------------------------------------------------ %
% Compute integral:
% ------------------------------------------------------------------------ %
switch mat
    case 'M'
        integ_out= trapz(x,f(x));
    case 'C'
        integ_out= trapz(x,g(x));
    case 'K'
        integ_out= trapz(x,h(x));
    case 'T'
        xTube= P.Tube.xTube;
        integ_out= trapz(xTube,t(xTube));
end
end
% ------------------------------------------------------------------------ %
% Basis Function
function Psi_out= Psi(s,j,P)
% Family of basis functions used for the Ritz series method.
% INPUT  1:  s - Position along tailboom segment [m].
% INPUT  2:  j - Integer indexing value.
% INPUT  3:  P - Struct containing all passing parameters.
% OUTPUT 1:  Psi_out - The jth basis function evaluated at x meters.

% ------------------------------------------------------------------------ %
% Unpack parameters:
L= P.Beam.length;
scaleTB= P.Simulation.scaleTB;

switch scaleTB
    case {'small','OH-58'}
        % Define family of basis functions Psi:
        sig_j= P.Simulation.kn;
        a= sig_j(j);

        Psi_out= sin(a*s)+sinh(a*s)+cosh(a*L)*(cosh(a*s)*sin(a*L)-...
            sin(a*(L-s))+cos(a*L)*sinh(a*s))+sinh(a*L)*(cos(a*(L-s))-...
            cos(a*L)*cosh(a*s)-sin(a*L)*sinh(a*s));

        Psi_out= Psi_out/(1+cos(a*L)*cosh(a*L)-sin(a*L)*sinh(a*L));
    case 'full'
        % Define family of basis functions Psi:
        [sig_j,R_j]= sig(j);
        k= sig_j/L;

        Psi_out= sin(k*s) - sinh(k*s) + R_j*(cos(k*s) - cosh(k*s));
end
end
% ------------------------------------------------------------------------ %
% First Positional Derivative of Basis Function
function d1Psi_out= d1Psi(s,j,P)
% Calculates numerically the first derivative of Psi_j.
% INPUT  1:  s - Position along tailboom segment [m].
```

```
% INPUT  2:  j - Integer indexing value.
% INPUT  3:  P - Struct containing all passing parameters.
% OUTPUT 1:  Psi_out - Numerical output of product evaluated at x.

% ------------------------------------------------------------------------ %
% Unpack parameters:
L= P.Beam.length;
scaleTB= P.Simulation.scaleTB;

switch scaleTB
    case {'small','OH-58'}
        sig_j= P.Simulation.kn;
        a= sig_j(j);

        d1Psi_out= a*cos(a*L)*cos(a*s)*cosh(a*L)+a*cos(a*s)*cosh(a*L)^2+...
            a*(cos(a*L)^2)*cosh(a*s)+a*cos(a*L)*cosh(a*L)*cosh(a*s) + ...
            a*cosh(a*s)*sin(a*L)^2 + a*cosh(a*L)*sin(a*L)*sin(a*s) + ...
            a*cos(a*s)*sin(a*L)*sinh(a*L) - ...
            a*cosh(a*s)*sin(a*L)*sinh(a*L) - ...
            a*cos(a*L)*sin(a*s)*sinh(a*L) - a*cos(a*s)*sinh(a*L)^2 + ...
            a*cosh(a*L)*sin(a*L)*sinh(a*s) - ...
            a*cos(a*L)*sinh(a*L)*sinh(a*s);

        d1Psi_out= d1Psi_out/(1+cos(a*L)*cosh(a*L)-sin(a*L)*sinh(a*L));
    case 'full'
        [sig_j,R_j]= sig(j);
        k= sig_j/L;

        d1Psi_out= k*(cos(k*s) - cosh(k*s)+R_j*(-sin(k*s)-sinh(k*s)));
end
end
% ------------------------------------------------------------------------ %
% Second Positional Derivative of Basis Function
function d2Psi_out= d2Psi(s,j,P)
% Calculates analytically the first derivative of Psi_j.
% INPUT  1:  s - Position along tailboom segment [m].
% INPUT  2:  j - Integer indexing value.
% INPUT  3:  P - Struct containing all passing parameters.
% OUTPUT 1:  Psi_out - Numerical output of product evaluated at x.

% ------------------------------------------------------------------------ %
% Unpack parameters:
L= P.Beam.length;
scaleTB= P.Simulation.scaleTB;

switch scaleTB
    case {'small','OH-58'}
        sig_j= P.Simulation.kn;
        a= sig_j(j);

        d2Psi_out= (a^2)*(-sin(a*s)+sinh(a*s)+...
            cosh(a*L)*(cosh(a*s)*sin(a*L)+sin(a*(L-s))+ ...
            cos(a*L)*sinh(a*s)) - sinh(a*L)*(cos(a*(L-s)) + ...
            cos(a*L)*cosh(a*s) + sin(a*L)*sinh(a*s)));

        d2Psi_out= d2Psi_out/(1+cos(a*L)*cosh(a*L)-sin(a*L)*sinh(a*L));
    case 'full'
        [sig_j,R_j]= sig(j);
        k= sig_j/L;

        d2Psi_out= -(k^2)*(sin(k*s)+sinh(k*s)+R_j*(cos(k*s)+cosh(k*s)));
```

```
end
end
% ------------------------------------------------------------------------ %
% Third Positional Derivative of Basis Function
function d3Psi_out= d3Psi(s,j,P)
% Calculates analytically the first derivative of Psi_j.
% INPUT  1:  s - Position along tailboom segment [m].
% INPUT  2:  j - Integer indexing value.
% INPUT  3:  P - Struct containing all passing parameters.
% OUTPUT 1:  Psi_out - Numerical output of product evaluated at x.

% ------------------------------------------------------------------------ %
% Unpack parameters:
L= P.Beam.length;
scaleTB= P.Simulation.scaleTB;

switch scaleTB
    case {'small','OH-58'}
        sig_j= P.Simulation.kn;
        a= sig_j(j);

        d3Psi_out= -(a^3)*(cos(a*s)+cos(a*s)*sin(a*L)*sinh(a*L) - ...
            cos(a*L)*sin(a*s)*sinh(a*L) + ...
            cosh(a*s)*(-1 + sin(a*L)*sinh(a*L)) + ...
            cos(a*L)*sinh(a*L)*sinh(a*s) + cosh(a*L)*(cos(a*(L-s))-...
            cos(a*L)*cosh(a*s)-sin(a*L)*sinh(a*s)));

        d3Psi_out= d3Psi_out/(1+cos(a*L)*cosh(a*L)-sin(a*L)*sinh(a*L));
    case 'full'
        [sig_j,R_j]= sig(j);
        k= sig_j/L;

        d3Psi_out= -(k^3)*(cos(k*s)+cosh(k*s)-...
            R_j*(sin(k*s)-sinh(k*s)));
end
end
% ------------------------------------------------------------------------ %
% Product of Basis Functions
function PsiJN_out= PsiJN(s,j,n,P)
% Calculates symbolically the product of two basis functions Psi_j
% and Psi_n.
% INPUT  1:  s - Axial position along tailboom [m].
% INPUT  2:  j - Integer indexing value.
% INPUT  3:  n - Integer indexing value.
% INPUT  4:  P - Struct containing all passing parameters.
% OUTPUT 1:  Psi_out - Numerical output of product evaluated at x.

% ------------------------------------------------------------------------ %
% Multiply Psi_j and Psi_n:
PsiJN_out= Psi(s,j,P).*Psi(s,n,P);
end
% ------------------------------------------------------------------------ %
% Product of First Positional Derivative of Basis Functions
function d1Psi_out= d1PsiJN(s,j,n,P)
% Calculates numerically the product of the first derivative of two
% basis functions Psi_j and Psi_n.
% INPUT  1:  s - Position along tailboom segment [m].
% INPUT  2:  j - Integer indexing value.
% INPUT  3:  n - Integer indexing value.
% INPUT  4:  P - Struct containing all passing parameters.
% OUTPUT 1:  Psi_out - Numerical output of product evaluated at x.
```

```
% ------------------------------------------------------------------------ %

% Differentiate once with respect to s:
d1Psi_j= d1Psi(s,j,P);

d1Psi_n= d1Psi(s,n,P);

% Compute product of derivatives:
d1Psi_out= d1Psi_j.*d1Psi_n;
end
% ------------------------------------------------------------------------ %
% Product of Second Positional Derivative of Basis Functions
function d2Psi_out= d2PsiJN(s,j,n,P)
% Calculates numerically the product of the second derivative of two
% basis functions Psi_j and Psi_n.
% INPUT  1:  s - Position along tailboom segment [m].
% INPUT  2:  j - Integer indexing value.
% INPUT  3:  n - Integer indexing value.
% INPUT  4:  P - Struct containing all passing parameters.
% OUTPUT 1:  Psi_out - Numerical output of product evaluated at x.

% ------------------------------------------------------------------------ %

% Differentiate twice with respect to x:
d2Psi_j= d2Psi(s,j,P);

d2Psi_n= d2Psi(s,n,P);

% Compute product of derivatives:
d2Psi_out= d2Psi_j.*d2Psi_n;
end
% ------------------------------------------------------------------------ %
% Beam Characteristic Equation
function out= charEqn(a,P)
% Characteristic equation for a spring-hinged free beam.
% EI= 1.0613e6 for small-scale TB.
% Kt= K*EI;

scaleTB= P.Simulation.scaleTB;
L= P.Beam.length;

switch scaleTB
    case 'small'
        K= 2.85;
    case 'OH-58'
        K= 5;
    otherwise
        error('Characteristic equation not needed.\n')
end

aL= a*L;
out= 1 + cos(aL).*cosh(aL) - (a/K).*(cosh(aL).*sin(aL)-cos(aL).*sinh(aL));
end
% ------------------------------------------------------------------------ %
% Mode Coefficient
function [k_n,R_n]= sig(n)
% Coefficients for various modal solutions.
% INPUT  1:  n - Integer indexing value.
% OUTPUT 1:  k_n - nth coefficient.
% OUTPUT 2:  R_n - Second nth coefficient, if used by modal solution.
```

```
%
% ------------------------------------------------------------------------ %
kn= [1.875104068711961;  4.694091132974175;  7.854757438237613;
    10.995540734875465; 14.137168391046471; 17.278759532088234;
    20.420352251041251; 23.561944901806445; 26.703537555518299;
    29.845130209102816; 32.986722862692837; 36.128315516282619;
    39.269908169872416; 42.411500823462205; 45.553093477052002;
    48.694686130641799; 51.836278784231588; 54.977871437821385;
    58.119464091411174; 61.261056745000971; 64.402649398590754;
    67.544242052180550; 70.685834705770347; 73.827427359360144;
    76.969020012949940; 80.110612666539723; 83.252205320129519;
    86.393797973719316; 89.535390627309113; 92.676983280898895;
    95.818575934488692; 98.960168588078488; 102.1017612416683;
    105.2433538952581;  108.3849465488479;  111.5265392024377;
    114.6681318560274;  117.8097245096172;  120.9513171632070;
    124.0929098167968;  127.2345024703866;  130.3760951239764;
    133.5176877775662;  136.6592804311560;  139.8008730847458;
    142.9424657383356;  146.0840583919254;  152.3672436991050];
if (n<=length(kn))
    k_n= kn(n);
else
    k_n= (2*n-1)*pi/2;
end

num= sin(k_n) + sinh(k_n);
den= cos(k_n) + cosh(k_n);
R_n= -num/den;
end
% ------------------------------------------------------------------------ %
```

```
function P= main(P)
%% Main function for fluidlastic tailboom simulation.
% This function computes frequency response for a cantilevered beam model
% of a helicopter tailboom, as well as for a fluidlastic tailboom model. If
% desired, this code can also simulate the tailboom/fluidlastic tailboom in
% the time domain.
%
% This function reads parameter values from a Microsoft Excel spreadsheet
% (System.xlsx). If proper input arguments are given, the function
% overrides up to two of the extracted parameter values.
%
% INPUT  1:  P - Data structure containing parameter values. Populated by
% main() on the initial run, and then fed back into main() to avoid
% re-reading parameters from System_Data.xlsx.
%
% OUTPUT 1:  P - Data structure containing parameter values.
%
% Kentaro Miura
% Mechatronics Research Laboratory
% Vertical Lift Research Center of Excellence
% Pennsylvania State University

%% Compute F2MC-tailboom system and frequency response
% This segment of code computes the F2MC-tailboom state space model, and
% uses it to compute its frequency response. If resEst==1, the
% F2MC-tailboom computation will include resistance estimation.

% Define frequency vector [Hz]:
freq_vec= P.Frequency.w/(2*pi);
P.freq_vec= freq_vec;

% Compute F2MC-beam frequency response:
w= P.Frequency.w;
P.Frequency.freqInd= length(w);
magBeamTube= zeros(size(w));
P.Simulation.inertanceYesNo= 1;

resistanceModel= P.Simulation.res;
config= P.Simulation.config;

switch resistanceModel
    case 'freqDep'
        for i= 1:length(w)
            ww= w(i);
            f= ww/2/pi;
            P= fluidlasticSys(P,f,i);

            beamTubeSys= P.beamTubesys{i};
            magBeamTube(i)= abs(evalfr(beamTubeSys,ww*1i));

            freqInd= P.Frequency.freqInd;
            if (i==freqInd)
                P.Frequency.magBeamTubeRes= magBeamTube(i);
            end
        end
    case 'const'
        % Use averaged inertance and resistance values:
        switch P.Simulation.scaleTB
            case 'OH-58'
                f1= 3;
                f2= 8;
```

```
    case 'small'
        f1= 10;
        f2= 14;
    otherwise
        f1= w(1)/2/pi;
        f2= w(end)/2/pi;
end

% Averaged resistance, inertance values to be used in simulation:
switch config
    case {'coupled','uncoupled'}
        % Compute resistance factor at frequency extremes:
        [Rf1,alph1]= computeResistance(P,f1,1);
        [Rf2,alph2]= computeResistance(P,f2,1);

        % Correct resistance:
        Rf_avg= mean([Rf1,Rf2]);
        P.Tube.Rf_avg= Rf_avg;

        % Compute inertance factor at frequency extremes:
        [If1,~]= computeInertance(alph1);
        [If2,~]= computeInertance(alph2);

        % Correct inertance:
        I0= P.Tube.inertance(1);
        If_avg= mean([If1 If2]);
        I_avg= I0*If_avg;
        P.Tube.I_avg= I_avg;
        P.Tube.If_avg= If_avg;
    case 'coupled tunable'
        % Compute resistance factor at frequency extremes:
        [Rf1,alph1]= computeResistance(P,f1,1);
        R1f1= Rf1(1);
        R2f1= Rf1(2);

        alph1f1= alph1(1);
        alph2f1= alph1(2);

        [Rf2,alph2]= computeResistance(P,f2,1);
        R1f2= Rf2(1);
        R2f2= Rf2(2);

        alph1f2= alph2(1);
        alph2f2= alph2(2);

        % Correct resistance:
        R01= P.Tube.resistance1(1);
        R02= P.Tube.resistance2(1);

        R1_mean= mean([R1f1,R1f2]);
        R2_mean= mean([R2f1,R2f2]);

        R1_avg= R01*R1_mean;
        R2_avg= R02*R2_mean;

        P.Tube.R1_avg= R1_avg;
        P.Tube.R2_avg= R2_avg;

        % Compute inertance factor at frequency extremes:
        [I1f1,~]= computeInertance(alph1f1);
        [I2f1,~]= computeInertance(alph2f1);
```

```
                [I1f2,~]= computeInertance(alph1f2);
                [I2f2,~]= computeInertance(alph2f2);

                % Correct inertance:
                I01= P.Tube.inertance1(1);
                I02= P.Tube.inertance2(1);

                I1_mean= mean([I1f1 I1f2]);
                I2_mean= mean([I2f1 I2f2]);

                I1_avg= I01*I1_mean;
                I2_avg= I02*I2_mean;

                P.Tube.I1_avg= I1_avg;
                P.Tube.I2_avg= I2_avg;
        end

        % Frequency index:
        freqInd= P.Frequency.freqInd;

        P= fluidlasticSys(P,-1,1);
        beamTubeSys= P.beamTubesys{1};
        magBeamTube= bode(beamTubeSys,w);
        magBeamTube= squeeze(magBeamTube(1,1,:));
    otherwise
        error('Select a resistance model in tailSim.m.')
end

P.Frequency.magBeamTube= magBeamTube;

% Convert response to desired output units:
P= convertUnits(P);

%% Simulate beam-tube system
% In this cell block we simulate the beam-tube system using lsim.m.

% Determine which input method to use:
% inputType= 'Sinusoidal Input';
% inputType= 'Step Input';
% inputType= 'Impulse Input';
inputType= '0';

% Determine response type:
% responseType= 'Tailboom';
responseType= 'Fluidlastic Tailboom';

% Simulate system using lsim.m:
if (strcmp('0',inputType)~=1)
    % Define the simulation time span vector [s]:
    endTime= 5;
    tSpan= linspace(0,endTime,10000*endTime);

    % Define system and initial conditions:
    switch responseType
        case 'Tailboom'
            q0= 0*P.Bbeam_v;
            sys= P.beamSys_v;
        case 'Fluidlastic Tailboom'
            q0= 0*P.Bbeamtube{1};
            sys= P.beamTubesys{1};
    end
```

```
    % Define forcing input time history:
    switch inputType
        case 'Step Input'
            fora= ' for a ';
            stepMag= 1;
            u= stepMag*ones(1,length(tSpan));
            u(1)= 0;

            inputType= [num2str(stepMag) 'N Step Input'];

            % Simulate system:
            [Z,t]= lsim(sys,u,tSpan,q0);
        case 'Sinusoidal Input'
            fora= ' for a ';
            stepMag= 27.7;
            fr= 6.88;
            u= stepMag*sin(fr*2*pi*tSpan);

            inputType= [num2str(stepMag) 'N Sinusoid, ' num2str(fr) 'Hz'];

            % Simulate system:
            [Z,t]= lsim(sys,u,tSpan,q0);
        case 'Impulse Input'
            fora= ' for an ';
            [Z,t]= impulse(sys,tSpan);
    end

    % Plot figures
    figure(2)
    plot(t,Z)
    title({['Time History of ' responseType]; [fora inputType]})
    xlabel('Time [s]')
    ylabel('Tip Displacement [m]')
    switch P.Simulation.res
        case 'const'
        case 'freqDep'
            error('System frequency-dependent\n\n')
        otherwise
            error('Select fluid model.\n\n')
    end
end

%% Debugging
% Compute and display damping estimate of first two resonance peaks:
[wnVec,~]= damp(P.beamSys_v);
zet= halfPowerEst(P.Frequency.w,magBeamTube);

% Display zero-frequency asymptote and resonances:
fprintf('\n------------------------------------------------ \n')
fprintf('START')
fprintf('\n------------------------------------------------ \n')
if (length(wnVec)>2)
    wnVec= wnVec(1:2:4)/(2*pi);
end
fprintf(['Peaks at:   ' num2str(wnVec') ' Hz\n'])
fprintf(['Zeta: ' num2str(zet(1)) '\n\n'])
fprintf(['Added: ' num2str(100*(zet(1)-P.Beam.dampingRatio)) 'pct \n'])

% Display vibration reduction at the first resonance:
if (isfield(P,'Frequency.magBeamTubeRes'))
```

```
    magBeam= P.Frequency.magBeam;
    magBeamRes= magBeam(freqInd);
    magBeamTubeRes= P.Frequency.magBeamTubeRes;
    magReduction= 20*log10(magBeamRes)-20*log10(magBeamTubeRes);
    fprintf(['Reduction: ' num2str(abs(magReduction)) ' dB \n\n'])
end

% Estimate component mass:
P.SystemMass= massF2MC(P);

%% Save data to struct P
% Save results into struct P:
trialNum= P.trialNum;
firstModeDampingVec= P.Results.firstModeDampingVec;
firstModeDampingVec(trialNum)= zet(1);
P.Results.firstModeDampingVec= firstModeDampingVec;

end

%% List of subfunctions:
% --------------------------------------------------------------------- %
function P= fluidlasticSys(P,f,i)
% Compute system parameters and matrices for various F2MC tube
% configurations.
% INPUT  1:  P - Data structure containing parameter values.
% INPUT  2:  f - Frequency [Hz] at which to compute system matrices.
% INPUT  3:  i - Index denoting the i-th entry of the frequency vector.
% OUTPUT 1:  P - Data structure containing parameter values.
%
% --------------------------------------------------------------------- %

% Compute tailboom-F2MC tube system matrices:
[Abt,Bbt,Cbt,Dbt,P]= computeBeamTubeMatrices(P,f,i);

% Pack state matrices into struct P:
P.Abeamtube{i}= Abt;
P.Bbeamtube{i}= Bbt;
P.Cbeamtube{i}= Cbt;
P.Dbeamtube{i}= Dbt;

P.beamTubesys{i}= ss(Abt,Bbt,Cbt,Dbt);
end
% --------------------------------------------------------------------- %
% Estimate F2MC tube mass
function systemMass= massF2MC(P)
% Estimates the mass of an individual F2MC tube filled with water. Does not
% take into account fittings, inertia track, and accumulator.
% INPUT  1:  P - Vector containing frequency values.
% OUTPUT 1:  systemtMass - Estimate of F2MC system mass.

% --------------------------------------------------------------------- %
% Unpack variables:
config= P.Simulation.config;
tubeType= P.Simulation.tubeType;
fluidDensity= P.Tube.fluidDensity;
numTubePairs= P.Tube.numTubePairs;

nTubes= 2*sum(numTubePairs);

switch config
    case 'coupled'
```

```
innerRadius= P.Tube.innerRadius;
outerRadius= innerRadius + P.Tube.thickness;
tubeLength= P.Tube.length;

switch tubeType
    case 'F2MC'
        % Define F2MC density:
        laminateDensity= 1014; % Density of Reoflex 200 [kg/m^3]
        fiberDensity= 1740; % Density of carbon fiber [kg/m^3]
        tubeDensity= 0.5*laminateDensity + 0.5*fiberDensity;

        % Compute F2MC tube volume:
        tubeVolume= pi*(outerRadius.^2-innerRadius.^2).*tubeLength;
        tubeMass= 2*dot(tubeVolume,numTubePairs)*tubeDensity;
    case 'McKibben'
        % Estimate of mass based on measurements for a 4" tube:
        measF2MCmass= 0.029/6.5; % [kg/in]
        lengthF2MC= tubeLength/0.0254 + 2;
        tubeMass= nTubes*measF2MCmass*lengthF2MC;
end

% Compute F2MC fluid mass [kg]:
fluidVolume= tubeLength*pi.*innerRadius.^2;
tubeFluidMass= 2*dot(fluidVolume,numTubePairs)*fluidDensity;

% Compute inertia track fluid mass [kg]:
rp= P.Tube.portRadius;
lp= P.Tube.portLength;
trackFluidMass= fluidDensity*lp*pi*rp^2;

% Compute inertia track mass [kg]:
tCu= (3/8-0.277)/2; % Based on 0.277" Cu tubing from McMaster-Carr
ACu= pi*((rp+tCu*0.0254)^2 - rp^2); % [m^2]

switch P.Simulation.scaleTB
    case 'OH-58'
        rhoCu= 8960/9; % [kg/m^3], typical plastic
    otherwise
        rhoCu= 8960; % [kg/m^3]
end
trackMass= ACu*rhoCu;
valveMass= 0.1075;
circuitMass= trackMass + 3*valveMass;

% Total fluid mass [kg]:
fluidMass= trackFluidMass + tubeFluidMass;

% End fitting mass [kg]:
rodMass= 2*nTubes*0.17; % Estimated 157g, 170 is more conservative

% L-bracket mass [kg]:
switch P.Simulation.scaleTB
    case 'small'
        bracketMass= 2*(0.9/2.2);
    otherwise
        bracketMass= 2*((0.9/2.2)/13)*12;
end

% Estimated total mass [kg]:
totalMass= rodMass+tubeMass+bracketMass*1+fluidMass+circuitMass;
```

```
    % Print results:
    txt1= 'End fittings + nuts + washers+barbs: %g [kg], %g [lb]\n';
    fprintf(txt1,rodMass,rodMass*2.2)
    txt2= 'F2MC tubes x%g mass: %g [kg], %g [lb]\n';
    fprintf(txt2,nTubes,tubeMass,tubeMass*2.2)
    txt3= 'L-brackets x2 mass: %g [kg], %g [lb]\n';
    fprintf(txt3,bracketMass,bracketMass*2.2)
    txt4= 'Fluid mass: %g [kg], %g [lb]\n';
    fprintf(txt4,fluidMass,fluidMass*2.2)
    txt5= 'Cu tubing + valves x3: %g [kg], %g [lb]\n';
    fprintf(txt5,circuitMass,circuitMass*2.2)
    txt6= 'Total system mass: %g [kg], %g [lb]\n';
    fprintf(txt6,totalMass,totalMass*2.2)
case 'uncoupled'
    error('INCOMPLETE')
case 'pure'
    error('INCOMPLETE')
case 'coupled tunable'
    innerRadius= P.Tube.innerRadius;
    outerRadius= innerRadius + P.Tube.thickness;
    tubeLength= P.Tube.length;

    switch tubeType
        case 'F2MC'
            % Define F2MC density:
            laminateDensity= 1014; % Density of Reoflex 200 [kg/m^3]
            fiberDensity= 1740; % Density of carbon fiber [kg/m^3]
            tubeDensity= 0.5*laminateDensity + 0.5*fiberDensity;

            % Compute F2MC tube volume:
            tubeVolume= pi*(outerRadius.^2-innerRadius.^2).*tubeLength;
            tubeMass= 2*dot(tubeVolume,numTubePairs)*tubeDensity;
        case 'McKibben'
            % Estimate of mass based on measurements for a 4" tube:
            measF2MCmass= 0.029/6.5; % [kg/in]
            lengthF2MC= tubeLength/0.0254 + 2;
            tubeMass= nTubes*measF2MCmass*lengthF2MC;
    end

    % Compute F2MC fluid mass [kg]:
    fluidVolume= tubeLength*pi.*innerRadius.^2;
    tubeFluidMass= 2*dot(fluidVolume,numTubePairs)*fluidDensity;

    % Compute inertia track fluid mass [kg]:
    rp1= P.Tube.portRadius1;
    lp1= P.Tube.portLength1;
    rp2= P.Tube.portRadius2;
    lp2= P.Tube.portLength2;
    track1FluidMass= fluidDensity*lp1*pi*rp1^2;
    track2FluidMass= fluidDensity*lp2*pi*rp2^2;

    % Compute inertia track mass [kg]:
    tCu= (3/8-0.277)/2; % Based on 0.277" Cu tubing from McMaster-Carr
    A1Cu= pi*((rp1+tCu*0.0254)^2 - rp1^2); % [m^2]
    A2Cu= pi*((rp2+tCu*0.0254)^2 - rp2^2); % [m^2]
    rhoCu= 8960; % [kg/m^3]
    trackMass= (A1Cu+A2Cu)*rhoCu;
    valveMass= 0.1075;
    circuitMass= trackMass + 3*valveMass;

    % Total fluid mass [kg]:
```

```
        fluidMass= track1FluidMass + track2FluidMass + tubeFluidMass;

        % End fitting mass [kg]:
        rodMass= 2*nTubes*0.17; % Estimated 157g, 170 is more conservative

        % L-bracket mass [kg]:
        bracketMass= 2*(0.9/2.2);

        % Estimated total mass [kg]:
        totalMass= rodMass+tubeMass+bracketMass*0+fluidMass+circuitMass;

        % Print results:
        txt1= 'End fittings + nuts + washers+barbs: %g [kg], %g [lb]\n';
        fprintf(txt1,rodMass,rodMass*2.2)
        txt2= 'F2MC tubes x%g mass: %g [kg], %g [lb]\n';
        fprintf(txt2,nTubes,tubeMass,tubeMass*2.2)
        txt3= 'L-brackets x2 mass: %g [kg], %g [lb]\n';
        fprintf(txt3,bracketMass,bracketMass*2.2)
        txt4= 'Fluid mass: %g [kg], %g [lb]\n';
        fprintf(txt4,fluidMass,fluidMass*2.2)
        txt5= 'Cu tubing + valves x3: %g [kg], %g [lb]\n';
        fprintf(txt5,circuitMass,circuitMass*2.2)
        txt6= 'Total system mass: %g [kg], %g [lb]\n';
        fprintf(txt6,totalMass,totalMass*2.2)
end
systemMass= totalMass;
end
% ------------------------------------------------------------------------ %
% Compute Tube Matrices
function [A,B,C,D,P]= computeTubeMatrices(P,f,tubeTypeNum,freqInd)
% computeTubeMatrices.m
% Function computing state space system matrices of an F2MC tube or a
% McKibben muscle.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  f - Frequency [Hz].
% INPUT  3:  tubeTypeNum - Index specifying tube type number.
% INPUT  4:  freqInd - Denotes the index of the frequency vector.
%
% OUTPUT 1:  [A B C D] - State space representation of system.
% OUTPUT 2:  P - Struct containing all passing parameters.

% ------------------------------------------------------------------------ %
% Unpack variables:
config= P.Simulation.config;
tubeType= P.Simulation.tubeType;
scaleTB= P.Simulation.scaleTB;

% Initialize storage variables:
tubeConst= cell(4,length(P.Tube.numTubePairs));

% Extract tube properties:
Lt= P.Tube.length(tubeTypeNum);
switch config
    case {'coupled','uncoupled'}
        I0= P.Tube.inertance(tubeTypeNum);
        R0= P.Tube.resistance(tubeTypeNum);
        R_orf= P.Tube.orificeResistance(tubeTypeNum);
    case 'coupled tunable'
        R1_orf= P.Tube.orificeResistance1(tubeTypeNum);
        R2_orf= P.Tube.orificeResistance2(tubeTypeNum);
end
```

```
% Compute inertance, resistance:
if (f<0)
    % Use averaged inertance, resistance:
    switch config
        case {'coupled','uncoupled'}
            R0= P.Tube.resistance(1);
            Rf_avg= P.Tube.Rf_avg;
            R_avg= (R0+P.Tube.orificeResistance(tubeTypeNum))*Rf_avg;
            P.Tube.R_avg= R_avg;
            R= P.Tube.R_avg;
            I= P.Tube.I_avg;
        case 'coupled tunable'
            R1= P.Tube.R1_avg + R1_orf;
            I1= P.Tube.I1_avg;

            R2= P.Tube.R2_avg + R2_orf;
            I2= P.Tube.I2_avg;
    end
else
    % Adjust inertance and resistance based on frequency:
    switch config
        case {'coupled','uncoupled'}
            [Rf,alph]= computeResistance(P,f,tubeTypeNum);
            [If,~]= computeInertance(alph);

            I= I0*If;
            R= R0*Rf + R_orf;
        case 'coupled tunable'
            error('incomplete')
    end
end

% Compute F2MC tube transfer function:
switch tubeType
    case 'F2MC'
        switch config
            case 'coupled'
                % Compute F2MC tube variables
                Amatrix_i= computeComplianceMatrix(P,tubeTypeNum);

                IR= P.Tube.innerRadius;
                IR= IR(tubeTypeNum);

                OR= P.Tube.outerRadius;
                OR= OR(tubeTypeNum);

                coef= tubeCoefficients(IR,OR,Amatrix_i);
                Sq= -P.Tube.initialVolume(tubeTypeNum)*(coef(1)+2*coef(4));
                Tq= -P.Tube.initialVolume(tubeTypeNum)*...
                    (1/P.Tube.fluidModulus(tubeTypeNum)+coef(2)+2*coef(5));

                Phi_1= coef(1);
                Phi_2= coef(2);

                P.Tube.phi1(tubeTypeNum)= Phi_1;
                P.Tube.phi2(tubeTypeNum)= Phi_2;
                P.Tube.Sq(tubeTypeNum)= Sq;
                P.Tube.Tq(tubeTypeNum)= Tq;

                % Axial force scale factor:
```

```
            numPairs= P.Tube.numTubePairs;
            numTubesTotal= numPairs*2;
            P.Tube.numTubes= numTubesTotal;

            % Define the following to simplify the equations of motion:
            A0= pi*IR^2;
            P.Tube.A0(tubeTypeNum)= A0;
            k1= Phi_2*Sq-Phi_1*Tq;
            D0= 2*Phi_1/k1;
            D1= R;
            D2= I;
            N0= -2/(Lt*k1);
            N1= R*(Sq*A0+Tq)/(Lt*k1);
            N2= I*(Sq*A0+Tq)/(Lt*k1);
        case 'uncoupled'
            error('INCOMPLETE')
        case 'pure'
            error('INCOMPLETE')
    end
case 'McKibben'
    switch config
        case {'coupled','coupled tunable'}
            % Compute McKibben tube variables:
            [C1,C2,C3]= mcKibbenTubeConstants(P);

            % Convert variables to SI units:
            C1= C1*175.2684;
            C2= C2*0.0254^2;
            C3= C3*0.0254^2;

            % Assign C3, C4 based on empirical data:
            switch P.Tube.bladderType
                case 'piston'
                    % Comparison against 1/32 soft bladder
                    C1= 0;
                    C2= pi*P.Tube.innerRadius^2;
                    C3= -C2;
                    C4= 0.15*2*5.45e-13;
                case 'unobtainium'
                    % The near-ideal damper bladder:
                    C3= -1.64e-3; % Unobtainium
                    C4= 0.15*2*5.45e-13;  % Unobtainium
                case '1/32 soft rubber'
                    C3= -1.64e-3;
                    switch scaleTB
                        case 'small'
                            C4= 2*5.45e-13; % Measured 2/11/2015
                        otherwise
                            C4= (Lt/(6*0.0254))*5.45e-13;
                    end
                case '1/16 soft rubber'
                    C3= -1.62e-3; % 1/16 soft rubber
                    C4= 2*10.55e-13; % 1/16 soft rubber
                case '1/8 soft rubber'
                    C3= C3*0.0254^2;
                    C4= 25.5e-13;
                case '1/16 durable rubber'
                    C3= -1.28e-3; % 1/16 durable rubber
                    C4= 7.89e-13; % 1/16 durable rubber
                case '1/16 soft PVC'
                    C3= -1.22e-3; % 1/16 soft PVC
```

```
                    C4= 5.31e-13; % 1/16 soft PVC
                case '1/16 hard PVC'
                    C3= -0.23e-3; % 1/16 hard PVC
                    C4= 2.81e-13; % 1/16 hard PVC
                otherwise
                    error('Select a bladder type.\n\n')
            end

            % Store tube constants for the tubeTypeNum-th tube:
            tubeConst{1,tubeTypeNum,freqInd}= C1;
            tubeConst{2,tubeTypeNum,freqInd}= C2;
            tubeConst{3,tubeTypeNum,freqInd}= C3;
            tubeConst{4,tubeTypeNum,freqInd}= C4;
            P.Tube.tubeConst= tubeConst;

            % Axial force scale factor:
            numPairs= P.Tube.numTubePairs;
            numTubesTotal= numPairs*2;
            P.Tube.numTubes= numTubesTotal;

            % Define the following to simplify the equations of motion:
            gam= C1-C2*C3/C4;

            switch config
                case 'coupled'
                    D0= 2/(C4*numPairs);
                    D1= R;
                    D2= I;
                    N0= D0*C1;
                    N1= gam*R;
                    N2= gam*I;
                case 'coupled tunable'
                    D0= 2*(R1+R2)/(C4*numPairs);
                    D1= R1*R2+2*(I1+I2)/(C4*numPairs);
                    D2= I1*R2+I2*R1;
                    D3= I1*I2;
                    N0= D0*C1;
                    N1= gam*R1*R2 + 2*C1*(I1+I2)/(C4*numPairs);
                    N2= gam*(I1*R2+I2*R1);
                    N3= gam*I1*I2;
            end
            % Note that the above assumes:
            % 1) 2p= IQdot + RQ
            % 2) c1*x + c2*p= Ft
            % 3) -c3*xdot - c4*pdot= Q
        case 'uncoupled'
            error('INCOMPLETE')
    end
otherwise
    error('Choose tube type.\n')
end

% Inertance tuning estimate:
switch scaleTB
    case 'full'
        f_target= [6 6.00218 6.03];
    case 'OH-58'
        f_target= [5.15 5.16 5.17];
    case 'small'
        f_target= [12.33 12.3624 12.38];
end
```

```
% Trigger the inertance tuning estimate at resonance. Throw a warning if
% the estimator is run more than once:
inertanceYesNo= P.Simulation.inertanceYesNo;
inertEst= P.Simulation.inertEst;

if (f>=f_target(1) && f<=f_target(3) && inertanceYesNo==0 && inertEst || 1)
    % Compute inertance tuning estimate:
    w_target= 2*pi*f_target(2);
    If= P.Tube.If_avg;
    I0_target= inertanceEstimator(P,tubeTypeNum,freqInd,w_target)/If;
    fprintf('Inertance tuning estimate: %g\n\n',I0_target)

    % Store index of the resonance estimate:
    P.f_targetNeighbor= f;
    P.Frequency.freqInd= freqInd;
    P.Simulation.inertanceYesNo= 1;
elseif (f>=f_target(1) && f<=f_target(3) && inertanceYesNo==1 && inertEst)
    error('Inertance estimator triggered twice. \n\n')
end

% Convert transfer function into state space (note that the realization is
% important for troubleshooting purposes, though tf2ss.m can be used):
switch config
    case {'coupled','uncoupled'}
        A= [-D1/D2 -D0/D2;1 0];
        B= [1/D2 0]';
        C= [N1-N2*D1/D2 N0-N2*D0/D2];
        D= N2/D2;
    case 'coupled tunable'
        A= [-D2/D3 -D1/D3 -D0/D3; 1 0 0 ; 0 1 0];
        B= [1/D3 0 0]';
        C= [N2-D2*N3/D3 N1-D1*N3/D3 N0-D0*N3/D3];
        D= N3/D3;
end
end
% ------------------------------------------------------------------------ %
% Compute Beam-Tube Matrices
function [A,B,C,D,P]= computeBeamTubeMatrices(P,f,freqInd)
% computeBeamTubeMatrices.m
% Function computing state space system matrices of a fluidlastic
% helicopter tailboom.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  freqInd - Denotes the index of the frequency vector.
% OUTPUT 1:  [A B C D] - State space representation of system.
% OUTPUT 2:  P - Struct containing all passing parameters.
% ------------------------------------------------------------------------ %
% Extract beam system matrices:
massMat= P.Beam.M_v;
stiffMat= P.Beam.K_v;
dampMat= P.Beam.C_v;

EI_0= P.EI_0;
[N,~]= size(massMat);

config= P.Simulation.config;
switch config
    case {'coupled','uncoupled'}
        m= 2;
    case 'coupled tunable'
        m= 3;
```

```
end

% Unpack basis functions:
switch P.Simulation.scaleTB
    case 'full'
        dof= 0;
    case 'OH-58'
        dof= 0;
    case 'small'
        dof= N-P.Simulation.nDimensions;
end

psi_L= [P.Basis.psi_L; zeros(dof,1)];
psi_Out= [P.Basis.psi_Out; zeros(dof,1)];
psi_0_doublePrime= [P.Basis.psi_0_doublePrime; zeros(dof,1)];
psi_0_triplePrime= [P.Basis.psi_0_triplePrime; zeros(dof,1)];
dtube= [P.Basis.d21*P.Tube.length; zeros(dof,1)];
d21Psi21p= [P.Basis.d21Psi21p; zeros(dof,1)];

% Calculate the number of different F2MC tube types:
nTubeTypes= length(P.Tube.numTubePairs);

% Combine tailboom and F2MC systems into single state-space model.
% Initialize varaibles:
beta_a_Sum= cell(1,nTubeTypes);

% Initialize beam-tube matrices:
N_eff= 2*N+m*nTubeTypes;
A= zeros(N_eff,N_eff);
B= zeros(N_eff,1);
C= zeros(1,N_eff);
D= 0;
tubeMatrices= cell(4,nTubeTypes);

for i=1:nTubeTypes
    % This loop will iterate through each tube and perform the following:
    % First, it will compute F2MC tube properties and package them into
    % variables alpha, beta, and gamma to simplify notation. Then, it will
    % populate the beam-tube A-matrix as needed.

    % Compute tube state space representation:
    jj= freqInd;
    [Atube_i,Btube_i,Ctube_i,Dtube_i,P]= computeTubeMatrices(P,f,i,jj);
    tubeMatrices{1,i}= Atube_i;
    tubeMatrices{2,i}= Btube_i;
    tubeMatrices{3,i}= Ctube_i;
    tubeMatrices{4,i}= Dtube_i;

    % To simplify notation, we define the following variables: beta, alpha,
    % and gamma. Compute beta-partial sum:
    d21= [P.Basis.d21(:,i); zeros(dof,1)];
    dtube= d21*P.Tube.length;
    numPairs= P.Tube.numTubePairs;
    beta_a_Sum{i}= 2*numPairs*Dtube_i*(d21Psi21p)*dtube';

    % Compute beta_i:
    beta_i= -2*numPairs*(massMat^-1)*d21Psi21p*Ctube_i;

    % Compute alpha_i:
    alpha_i= Btube_i*dtube';
```

```
    % Compute gamma_i:
    gamma_i= Atube_i;

    % Populate A-matrix as needed
    % beta_i:
    A(N+1:2*N, 2*N+m*(i-1)+1:2*N+m*i)= beta_i;

    % alpha_i:
    A(2*N+m*(i-1)+1:2*N+m*i, 1:N)= alpha_i;

    % gamma_i:
    A(2*N+m*(i-1)+1:2*N+m*i, 2*N+m*(i-1)+1:2*N+m*i)= gamma_i;

    if (P.TFmode>=4 && P.TFmode<=8)
        % Pack up cell arrays for pressure, force, and flow calculations:

        % Pressure:
        P.Tube.tubeMatrices= tubeMatrices;
    end
end

% Complete population of beam-tube matrices
% Sum all the beta parts:
beta_a_Sum= sum(cat(3,beta_a_Sum{:}),3);

% Populate the remainder of the beam-tube A-matrix:
A(1:N,N+1:2*N)= eye(N);
A(N+1:2*N, N+1:2*N)= -(massMat^-1)*dampMat;
A(N+1:2*N, 1:N)= -(massMat^-1)*(beta_a_Sum + stiffMat);

% Compute B-matrix:
B(N+1:2*N)= (massMat^-1)*psi_L;

switch P.TFmode
    case 1
        % Compute C-matrix (displacement at tip):
        C(1:N)= psi_Out';
    case 2
        % Compute C-matrix (moment at root):
        C(1:N)= -EI_0*psi_0_doublePrime';
    case 3
        % Compute C-matrix (shear at root):
        C(1:N)= -EI_0*psi_0_triplePrime' -EI_0p*psi_0_doublePrime';
    case 4
        % Compute C-matrix for tip force to pressure of jth tube.
        j= 1;

        if (nTubeTypes>1)
            d21= d21(:,1);
        end

        tubeConst= P.Tube.tubeConst;
        tubeMatrices= P.Tube.tubeMatrices;
        C1= tubeConst{1,j};
        C2= tubeConst{2,j};
        dtube= [d21*P.Tube.length; zeros(dof,1)];
        Ctube_i= tubeMatrices{3,j};
        Dtube_i= tubeMatrices{4,j};

        C(1:N)= (1/C2)*(Dtube_i - C1)*dtube';
        C(2*(N+j)-1:2*(N+j))= (1/C2)*Ctube_i;
```

```
    case 5
        % Compute C-matrix for tip force to axial force of jth tube.
        C(1:N)= Dtube_i*dtube';
        C(2*N+1:2*N+length(Ctube_i))= Ctube_i;
    case 6
        % Compute matrices for tip acceleration to pressure
        % of first tube:
        error('INCOMPLETE')
    case 7
        % Compute matrices for tip acceleration to axial force:
        Ca= psi_L'*A(N+1:2*N,:);
        Da= psi_L'*(massMat^-1)*psi_L;

        A= A-B*Ca;
        B= B/Da;

        if (nTubeTypes>1)
            d21= d21(:,1);
            dtube= d21*P.Tube.length;
        end
        C(1:N)= Dtube_i*dtube';
        C(2*N+1:2*N+2)= Ctube_i;
    case 8
        % Compute C-matrix for tip force to flow rate:
        j= 1;

        if (nTubeTypes>1)
            d21= d21(:,1);
            dtube= d21*P.Tube.length;
        else
            dtube= [P.Basis.d21*P.Tube.length; zeros(dof,1)];
        end
        tubeConst= P.Tube.tubeConst;
        tubeMatrices= P.Tube.tubeMatrices;
        C1= tubeConst{1,j};
        C2= tubeConst{2,j};
        C3= tubeConst{3,j};
        C4= tubeConst{4,j};

        Ctube_i= tubeMatrices{3,j};
        Dtube_i= tubeMatrices{4,j};

        Cp(1:N)= (1/C2)*(Dtube_i - C1)*dtube';
        Cp(2*(N+j)-1:2*(N+j))= (1/C2)*Ctube_i;

        C= -C4*Cp*A;
        C(1:2*N)= C(1:2*N) + [zeros(1,N) -C3*dtube'];
        D= -C4*Cp*B;
    case 9
        % C-matrix for tip force to tip acceleration [m-s^-2]:
        C= psi_Out'*A(N+1:2*N,:);
        D= psi_Out'*(massMat^-1)*psi_L;
    otherwise
        error('Choose a transfer function.')
end
end
% ------------------------------------------------------------------------ %
% Compute Compliance Matrix (Bin Zhu)
function complianceMatrix= computeComplianceMatrix(P,i)
% Bin Zhu's code for computing F2MC tube compliance matrix.
% Modified by Kentaro Miura.
```

```
%
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  i - Index variable denoting i-th F2MC tube.
%
% OUTPUT 1:  complianceMatrix - F2MC tube compliance matrix.

% ------------------------------------------------------------------------ %

E11= P.Tube.E11(i); E22= P.Tube.E22(i); v12= P.Tube.v12(i);
v23= P.Tube.v23(i); G12= P.Tube.G12(i); G23= P.Tube.G23(i);
angle= P.Tube.fiberAngle(i);

FiberAngle=[angle,-angle,-angle,+angle];
PlyNum=length(FiberAngle);
Layer_Thickness=[1,1,1,1];
Total_Thickness=sum(Layer_Thickness);
V=((1+v23)*(1-v23-2*v12^2*E22/E11));
c11=(1-v23^2)*E11/V;
c12=v12*(1+v23)*E22/V;
c13=c12;
c23=(v23+v12^2*E22/E11)*E22/V;
c22=(1-v12^2*E22/E11)*E22/V;
c33=c22;
c44=G23;
c55=G12;
c66=c55;
C0=[c11,c12,c13,0,0,0;
    c12,c22,c23,0,0,0;
    c13,c23,c33,0,0,0;
    0,0,0,c44,0,0;
    0,0,0,0,c55,0;
    0,0,0,0,0,c66];
a=0;b=0;c=0;
for k=1:PlyNum
    theta=FiberAngle(k)*pi/180;
    vk=Layer_Thickness(k)/Total_Thickness;
    m=cos(theta);
    n=sin(theta);
    T_s=[m^2, n^2, 0,0,0,2*m*n;
        n^2,m^2,0,0,0,-2*m*n;
        0,0,1,0,0,0;
        0,0,0,m,-n,0;
        0,0,0,n,m,0;
        -m*n,m*n,0,0,0,(m^2-n^2)];
    T_e=[m^2,n^2,0,0,0,m*n; % w.r.t. engineering strain
        n^2,m^2,0,0,0,-m*n;
        0,0,1,0,0,0;
        0,0,0,m,-n,0;
        0,0,0,n,m,0;
        -2*m*n,2*m*n,0,0,0,(m^2-n^2)];
    C=T_s\C0*T_e;
    delta_k=C(4,4)*C(5,5)-(C(4,5))^2;
    a=a+vk*C(4,4)/delta_k;
    b=b+vk*C(5,5)/delta_k;
    c=c+vk*C(4,5)/delta_k;
end % Just for calculation of delta
delta=a*b-c^2;
% The following is for the calculation of Cij
C_bar=[0,0,0,0,0,0;
    0,0,0,0,0,0;
    0,0,0,0,0,0;
```

```
    0,0,0,0,0,0;
    0,0,0,0,0,0;
    0,0,0,0,0,0];
for k=1:PlyNum
    theta=FiberAngle(k)*pi/180;
    vk=Layer_Thickness(k)/Total_Thickness;
    m=cos(theta);
    n=sin(theta);
    T_s=[m^2, n^2, 0,0,0,2*m*n;
        n^2,m^2,0,0,0,-2*m*n;
        0,0,1,0,0,0;
        0,0,0,m,-n,0;
        0,0,0,n,m,0;
        -m*n,m*n,0,0,0,(m^2-n^2)];
    T_e=[m^2,n^2,0,0,0,m*n;
        n^2,m^2,0,0,0,-m*n;
        0,0,1,0,0,0;
        0,0,0,m,-n,0;
        0,0,0,n,m,0;
        -2*m*n,2*m*n,0,0,0,(m^2-n^2)];
    C=T_s\C0*T_e;
    delta_k=C(4,4)*C(5,5)-(C(4,5))^2;
    C_bar(1,1)=C_bar(1,1)+vk*C(1,1);
    C_bar(1,2)=C_bar(1,2)+vk*C(1,2);
    C_bar(1,3)=C_bar(1,3)+vk*C(1,3);
    C_bar(1,6)=C_bar(1,6)+vk*C(1,6);
    C_bar(2,1)=C_bar(1,2);
    C_bar(2,2)=C_bar(2,2)+vk*C(2,2);
    C_bar(2,3)=C_bar(2,3)+vk*C(2,3);
    C_bar(2,6)=C_bar(2,6)+vk*C(2,6);
    C_bar(3,1)=C_bar(1,3);
    C_bar(3,2)=C_bar(2,3);
    C_bar(3,3)=C_bar(3,3)+vk*C(3,3);
    C_bar(3,6)=C_bar(3,6)+vk*C(3,6);
    C_bar(6,1)=C_bar(1,6);
    C_bar(6,2)=C_bar(2,6);
    C_bar(6,3)=C_bar(3,6);
    C_bar(6,6)=C_bar(6,6)+vk*C(6,6);
    C_bar(4,4)=C_bar(4,4)+vk*C(4,4)/delta_k/delta;
    C_bar(4,5)=C_bar(4,5)+vk*C(4,5)/delta_k/delta;
    C_bar(5,4)=C_bar(4,5);
    C_bar(5,5)=C_bar(5,5)+vk*C(5,5)/delta_k/delta;
end
% C_bar;
S_bar=inv(C_bar);

a11=S_bar(3,3);
a12=S_bar(2,3);
a13=S_bar(1,3);
a22=S_bar(2,2);
a23=S_bar(1,2);
a33=S_bar(1,1);
a66=S_bar(4,4);
a55=S_bar(5,5);
a44=S_bar(6,6);

complianceMatrix = [a11 a12 a13  0   0  0;
    a12 a22 a23  0   0  0;
    a13 a23 a33  0   0  0;
    0   0   0  a44  0  0;
    0   0   0   0  a55 0;
```

```
    0   0   0   0   0 a66];
end
% ------------------------------------------------------------------------ %
% Compute Tube Coefficients (Bin Zhu)
function tubeCoeff= tubeCoefficients(a,b,aMatrix)
% Bin Zhu's subroutine for generating F2MC tube equation coefficients.
% Modified by Kentaro Miura.
%
% INPUT  1:  a - F2MC tube inner radius [m].
% INPUT  2:  b - F2MC tube outer radius [m].
% INPUT  3:  aMatrix - F2MC tube compliance matrix.
%
% OUTPUT 1:  tubeCoeff - Vector of F2MC tube coefficients.

% ------------------------------------------------------------------------ %

a11= aMatrix(1,1);
a12= aMatrix(1,2);
a13= aMatrix(1,3);
a22= aMatrix(2,2);
a23= aMatrix(2,3);
a33= aMatrix(3,3);
a44= aMatrix(4,4);

beta11 = a11-a13^2/a33;
beta22 = a22-a23^2/a33;
beta44 = a44;
beta14 = 0; beta24 = 0;

k= sqrt(beta11/beta22);
kai= (a13-a23)*beta44/(beta22*beta44-beta24^2-beta11*beta44+beta14^2);
T= pi*(power(b,2)-power(a,2))-2*pi*kai/a33*((power(b,2)-power(a,2))...
    /2*(a13+a23)-(power(b,k+1)-power(a,k+1))*(a13+k*a23)/...
    (power(10*b,2*k)-power(10*a,2*k))/(k+1)*10^(2*k)*(power(b,k+1)...
    -power(a,k+1))-(power(b,k-1)-power(a,k-1))*power(a,2)*power(b,2)...
    *(a13-k*a23)/(power(10*b,2*k)-power(10*a,2*k))/(k-1)*10^(2*k)*...
    (power(b,k-1)-power(a,k-1)));

Coef_sr1= [kai/T; -kai*(power(b,k+1)-power(a,k+1))/(power(10*b,2*k)-...
    power(10*a,2*k))*10^(2*k)/T; -kai*(power(b,k-1)-power(a,k-1))...
    /(power(10*b,2*k)-power(10*a,2*k))*10^(2*k)/...
    T*power(a,k+1)*power(b,k+1)];
Coef_sq1= [kai/T; -kai*(power(b,k+1)-power(a,k+1))*k/(power(10*b,2*k)...
    -power(10*a,2*k))*10^(2*k)/T; kai*(power(b,k-1)-power(a,k-1))...
    *k/(power(10*b,2*k)-power(10*a,2*k))*10^(2*k)/T*power(a,k+1)...
    *power(b,k+1)];
Coef_sz1= -a13/a33*Coef_sr1 - a23/a33*Coef_sq1 + [1/T; 0; 0];

Coef_sr2= [0; power(a,k+1)/(power(10*b,2*k)-power(10*a,2*k))*10^(2*k);...
    -power(b,k-1)/(power(10*b,2*k)-power(10*a,2*k))*10^(2*k)...
    *power(a,k+1)*power(b,k+1)];
Coef_sr2c= kai*[1; -(power(b,k+1)-power(a,k+1))/(power(10*b,2*k)-...
    power(10*a,2*k))*10^(2*k); -(power(b,k-1)-power(a,k-1))/...
    (power(10*b,2*k)-power(10*a,2*k))*10^(2*k)*power(a,k+1)*power(b,k+1)];
Coef_sq2= [0; power(a,k+1)*k/(power(10*b,2*k)-power(10*a,2*k))*10^(2*k);...
    power(b,k-1)*k/(power(10*b,2*k)-power(10*a,2*k))*10^(2*k)*...
    power(a,k+1)*power(b,k+1)];
Coef_sq2c= kai*[1; -(power(b,k+1)-power(a,k+1))*k/(power(10*b,2*k)-...
    power(10*a,2*k))*10^(2*k); (power(b,k-1)-power(a,k-1))*k/...
    (power(10*b,2*k)-power(10*a,2*k))*10^(2*k)*power(a,k+1)*power(b,k+1)];
Coef_sz2= -a13/a33*Coef_sr2 - a23/a33*Coef_sq2;
```

```
Coef_sz2c= -a13/a33*Coef_sr2c - a23/a33*Coef_sq2c + [1; 0; 0];

FunP1= @(x)Coef_sz2(1)*x + Coef_sz2(2)*x.^k + Coef_sz2(3)*x.^-k;
P1= integral(FunP1,a,b)*2*pi;
FunP2= @(x)Coef_sz2c(1)*x + Coef_sz2c(2)*x.^k + Coef_sz2c(3)*x.^-k;
P2= integral(FunP2,a,b)*2*pi;

Coef_sr2= Coef_sr2 - P1/P2*Coef_sr2c;
Coef_sq2= Coef_sq2 - P1/P2*Coef_sq2c;
Coef_sz2= -a13/a33*Coef_sr2 - a23/a33*Coef_sq2 + [-P1/P2; 0; 0];

Coef_sr3= [0; -power(b,k+1)/(power(10*b,2*k)-power(10*a,2*k))*10^(2*k); ...
    power(a,k-1)/(power(10*b,2*k)-power(10*a,2*k))*...
    10^(2*k)*power(a,k+1)*power(b,k+1)];
Coef_sr3c= kai*[1; -(power(b,k+1)-power(a,k+1))/...
    (power(10*b,2*k)-power(10*a,2*k))*10^(2*k);...
    -(power(b,k-1)-power(a,k-1))/...
    (power(10*b,2*k)-power(10*a,2*k))*10^(2*k)*power(a,k+1)*power(b,k+1)];
Coef_sq3= [0; -power(b,k+1)*k/(power(10*b,2*k)-power(10*a,2*k))*...
    10^(2*k); -power(a,k-1)*k/(power(10*b,2*k)-power(10*a,2*k))*...
    10^(2*k)*power(a,k+1)*power(b,k+1)];
Coef_sq3c= kai*[1; -(power(b,k+1)-power(a,k+1))*k/(power(10*b,2*k)...
    -power(10*a,2*k))*10^(2*k); (power(b,k-1)-power(a,k-1))*k...
    /(power(10*b,2*k)-power(10*a,2*k))*10^(2*k)*power(a,k+1)*power(b,k+1)];
Coef_sz3= -a13/a33*Coef_sr3 - a23/a33*Coef_sq3;
Coef_sz3c= -a13/a33*Coef_sr3c - a23/a33*Coef_sq3c + [1; 0; 0];

FunQ1= @(x)Coef_sz3(1)*x + Coef_sz3(2)*x.^k + Coef_sz3(3)*x.^-k;
Q1= integral(FunQ1,a,b)*2*pi;
FunQ2= @(x)Coef_sz3c(1)*x + Coef_sz3c(2)*x.^k + Coef_sz3c(3)*x.^-k;
Q2= integral(FunQ2,a,b)*2*pi;

Coef_sr3= Coef_sr3 - Q1/Q2*Coef_sr3c;
Coef_sq3= Coef_sq3 - Q1/Q2*Coef_sq3c;
Coef_sz3= -a13/a33*Coef_sr3 - a23/a33*Coef_sq3 + [-Q1/Q2; 0; 0];

Coef_eq1= a12*Coef_sr1 + a22*Coef_sq1 + a23*Coef_sz1;
Coef_eq2= a12*Coef_sr2 + a22*Coef_sq2 + a23*Coef_sz2;
Coef_eq3= a12*Coef_sr3 + a22*Coef_sq3 + a23*Coef_sz3;
Coef_ez1= a13*Coef_sr1 + a23*Coef_sq1 + a33*Coef_sz1;
Coef_ez2= a13*Coef_sr2 + a23*Coef_sq2 + a33*Coef_sz2;
Coef_ez3= a13*Coef_sr3 + a23*Coef_sq3 + a33*Coef_sz3;

eq1a= Coef_eq1(1)+Coef_eq1(2)*a^(k-1)+Coef_eq1(3)*a^(-k-1);
eq2a= Coef_eq2(1)+Coef_eq2(2)*a^(k-1)+Coef_eq2(3)*a^(-k-1);
eq3a= Coef_eq3(1)+Coef_eq3(2)*a^(k-1)+Coef_eq3(3)*a^(-k-1);

ez1a= Coef_ez1(1)+Coef_ez1(2)*a^(k-1)+Coef_ez1(3)*a^(-k-1);
ez2a= Coef_ez2(1)+Coef_ez2(2)*a^(k-1)+Coef_ez2(3)*a^(-k-1);
ez3a= Coef_ez3(1)+Coef_ez3(2)*a^(k-1)+Coef_ez3(3)*a^(-k-1);

tubeCoeff= [ez1a ez2a ez3a eq1a eq2a eq3a];
end
% ------------------------------------------------------------------------ %
% Compute Tube Coefficients (Lloyd Scarborough)
function [C1,C2,C3]= mcKibbenTubeConstants(P)
% Lloyd Scarborough's code for computing braid-sheathed F2MC tube equation
% coefficients. Modified by Kentaro Miura.
%
% INPUT  1:  P - Struct containing all passing parameters.
%
```

```
% OUTPUT 1:  [C1 C2 C3] - F2MC tube coefficients.
%            C1 [lbf/in]
%            C2 [in^2]
%            C3 [in^2]
%
% Initial guesses are specified as the second argument of fsolve.m, and the
% options are specified in the variable: options
% ------------------------------------------------------------------------ %

% Unpack variables:
L= P.Tube.length/0.0254;
Ep= P.Tube.E11*0.000145037738;
Er= P.Tube.E22*0.000145037738;
rf= P.Tube.v12/0.0254;
nu= P.Tube.v23;
ds= P.Tube.G12/0.0254;
m= P.Tube.G23;
Rio= P.Tube.innerRadius/0.0254;
Roo= P.Tube.outerRadius/0.0254;
alpha= P.Tube.fiberAngle*pi/180;
Ls= L/cos(alpha);

% Define simulation parameters:
delta= 11111115/16;
p_res= 0.01;
lambda1_res= 0.001;

% Define operating fluid pressure [psi]:
p_op= P.Tube.operatingPressure;

% Define operating axial load [lbf]:
F_op= 0;

% Find lambda1_op
p=p_op;
F=F_op;

% Pack variables into struct P for linearization subroutines:
P.McKibben.Rio= Rio;
P.McKibben.Roo= Roo;
P.McKibben.L= L;
P.McKibben.alpha= alpha;
P.McKibben.Er= Er;
P.McKibben.nu= nu;
P.McKibben.Ep= Ep;
P.McKibben.ds= ds;
P.McKibben.Ls= Ls;
P.McKibben.m= m;
P.McKibben.rf= rf;
P.McKibben.delta= delta;
P.McKibben.p= p;
P.McKibben.F= F;

f1= @(l1) exprII_dyn_I(l1,P); % Refer to Scarborough [62]
options = optimset('Display','off');
lambda1_op=fsolve(f1,1,options);

% Compute C3 ------------------------------------------------------------- %
lambda1_op= lambda1_op-lambda1_res;
V_tube_b= pi .* ((0.1e1 - lambda1_op .^ 2 .* cos(alpha) .^ 2).^(1/2)...
    ./ sin(alpha) .* (Rio + Roo) ./ 0.2e1 - (Roo - Rio) ./ lambda1_op .*...
```

```
    (0.1e1 - lambda1_op .^ 2 .* cos(alpha) .^ 2) .^ (-0.1e1 ./ 0.2e1) .*...
    sin(alpha) ./ 0.2e1) .^ 2 .* lambda1_op .* L;
lambda1_op= lambda1_op+2*lambda1_res;
V_tube_a= pi .* ((0.1e1 - lambda1_op .^ 2 .* cos(alpha) .^ 2).^(1/2)...
    ./ sin(alpha) .* (Rio + Roo) ./ 0.2e1 - (Roo - Rio) ./ lambda1_op...
    .* (0.1e1 - lambda1_op .^ 2 .* cos(alpha) .^ 2) .^ (-0.1e1 ./ 0.2e1)...
    .* sin(alpha) ./ 0.2e1) .^ 2 .* lambda1_op .* L;
A3= (V_tube_a-V_tube_b)/(2*lambda1_res);
C3= A3/L;

% Reset lambda1_op:
lambda1_op= lambda1_op-lambda1_res;

% Compute C1 ------------------------------------------------------------- %
p= p_op;
lambda1= lambda1_op-lambda1_res;
P.McKibben.p= p;
P.McKibben.lambda1= lambda1;

f2= @(l2) exprII_dyn_II(l2,P);
Ftube_b= fsolve(f2,20,options);

lambda1= lambda1_op+2*lambda1_res;
P.McKibben.lambda1= lambda1;

f2= @(l2) exprII_dyn_II(l2,P);
Ftube_a= fsolve(f2,20,options);

A1= (Ftube_a-Ftube_b)/(2*lambda1_res);
C1= A1/L;

% Reset lambda1_op:
lambda1_op= lambda1_op-lambda1_res;

% Compute C2 ------------------------------------------------------------- %
lambda1= lambda1_op;
p= p_op-p_res;
P.McKibben.p= p;
P.McKibben.lambda1= lambda1;

f2= @(l2) exprII_dyn_II(l2,P); % Refer to Scarborough [62]
Ftube_b= fsolve(f2,20,options);

p= p_op+2*p_res;
P.McKibben.p= p;
f2= @(l2) exprII_dyn_II(l2,P);
Ftube_a= fsolve(f2,20,options);

C2= (Ftube_a-Ftube_b)/(2*p_res);
end
% ------------------------------------------------------------------------ %
% Inertance tuning estimator
function I0_target= inertanceEstimator(P,tubeTypeNum,freqInd,w_target)
% Subroutine for estimating the inertance required to tune an absorber.
% Uses a reduced-order model of the tailboom.
% INPUT  1:  P - Data structure containing parameter values.
% INPUT  2:  tubeTypeNum - Index specifying tube type number.
% INPUT  3:  freqInd - Denotes the index of the frequency vector.
% INPUT  4:  w_target - Target frequency [rad/s].
%
% OUTPUT 1:  I0_target - Target inertance [kg/m^4].
```

```
% ------------------------------------------------------------------------ %

% Unpack variables:
dtube= P.Basis.d21*P.Tube.length;
d21Psi21p= P.Basis.d21Psi21p;
numPairs= P.Tube.numTubePairs;
np= numPairs;
C1= P.Tube.tubeConst{1,tubeTypeNum,freqInd};
C2= P.Tube.tubeConst{2,tubeTypeNum,freqInd};
C3= P.Tube.tubeConst{3,tubeTypeNum,freqInd};
C4= P.Tube.tubeConst{4,tubeTypeNum,freqInd};
gam= C1-C2*C3/C4;
d211= d21Psi21p(1);
d212= d21Psi21p(2);
d213= d21Psi21p(3);
dt1= dtube(1); dt2= dtube(2); dt3= dtube(3);
massMat= P.Beam.M_v;
stiffMat= P.Beam.K_v;
M11= massMat(1,1);
M12= massMat(1,2);
M13= massMat(1,3);
M22= massMat(2,2);
M23= massMat(2,3);
M33= massMat(3,3);
k11= stiffMat(1,1);
k12= stiffMat(1,2);
k13= stiffMat(1,3);
k22= stiffMat(2,2);
k23= stiffMat(2,3);
k33= stiffMat(3,3);

% Compute optimal parameters for sdof oscillator + absorber:
k0= -2*np*C2*C3*d211*dt1/C4;
m1= M11;
m2= -(np^2)*C2*C3*d211*dt1*P.Tube.I_avg;
mu= m2/m1;
zetfp= sqrt(3*mu/(8-4*mu));
np= numPairs;
Iffp= (2/(C4*np))*M11/(k11+2*np*gam*d211*dt1)/P.Tube.If_avg;
b2opt= 2*sqrt(m2*k0)*(zetfp);
Ropt= -b2opt/((np^2)*C2*C3*d211*dt1);
Rffp= Ropt/P.Tube.Rf_avg;
Ifsimple= (2/(C4*np))/((w_target)^2)/P.Tube.If_avg;

% Print results:
fprintf('Inertance estimate using simple tuning rule: %g \n',Ifsimple)
fprintf('Inertance estimate using fixed points: %g \n',Iffp)
fprintf('Resistance estimate using fixed points: %g \n\n',Rffp)

if (0)
    % Authority test processing:
    J= (stiffMat+2*np*C1*(dtube*d21Psi21p')')^-1;
    pp= [0 30 40 60 80 100 120]*6.89476; % Pressure vector
    wL= -2*np*C2*P.Basis.psi_Out'*J*d21Psi21p*pp; % Displacement

    figure
    ppe1= [0 30 40 60 80 100 120]*6.89476; %Exp. pressure
    wLe1= 0.0001*[65 43 35 24 11 10 6]; % Exp. displ.
    dc= wLe1(1);
    wLe1= wLe1-dc;
```

```
    ppe2= [120 100 80 60 40 30 0]*6.89476; %Exp. pressure
    wLe2= 0.0001*[6 13 16 31 43 54 69]; % Exp. displ.
    wLe2= wLe2-dc;
    plot(pp,wL*1000,ppe1,wLe1,'o-',ppe2,wLe2,'*-')
    xlabel('Pressure [kPa]')
    ylabel('Vertical Displacement [mm]')
    legend('Model Prediction','Loading Experiment','Unloading Experiment')
end

% Compute inertance estimate:
I0_target= 2.*C4.^(-2).*numPairs.^(-1).*w_target.^(-2).*(k23.^2.*M11+...
    k12.*k33.*M12+(-2).*d212.*dt2.*gam.*k33.*M11.*numPairs+...
    2.*d213.*dt3.*gam.*k12.*M12.*numPairs+...
    (-2).*d213.*dt2.*gam.*k13.*M12.*numPairs+2.*d211.* ...
    dt2.*gam.*k33.*M12.*numPairs+(-2).*d212.*dt3.*gam.*k12.*M13.* ...
    numPairs+2.*d212.*dt2.*gam.*k13.*M13.*numPairs+(-1).*k33.*M12.^2.* ...
    w_target.^2+k33.*M11.*M22.*w_target.^2+...
    (-1).*k13.*M13.*M22.*w_target.^2+k13.*M12.*M23.* ...
    w_target.^2+k12.*M13.*M23.*w_target.^2+...
    (-1).*k12.*M12.*M33.*w_target.^2+(-2).*d213.* ...
    dt3.*gam.*M12.^2.*numPairs.*w_target.^2+...
    2.*d213.*dt2.*gam.*M12.*M13.* ...
    numPairs.*w_target.^2+...
    2.*d212.*dt3.*gam.*M12.*M13.*numPairs.*w_target.^2+(-2).* ...
    d212.*dt2.*gam.*M13.^2.*numPairs.*w_target.^2+...
    2.*d213.*dt3.*gam.*M11.* ...
    M22.*numPairs.*w_target.^2+...
    (-2).*d211.*dt3.*gam.*M13.*M22.*numPairs.* ...
    w_target.^2+(-2).*d213.*dt2.*gam.*M11.*M23.*numPairs.*w_target.^2+...
    (-2).*d212.*dt3.*gam.*M11.*M23.*numPairs.*w_target.^2+...
    2.*d211.*dt3.*gam.*M12.*M23.*numPairs.*w_target.^2+...
    2.*d211.*dt2.*gam.*M13.*M23.*numPairs.*w_target.^2+2.* ...
    d212.*dt2.*gam.*M11.*M33.*numPairs.*w_target.^2+...
    (-2).*d211.*dt2.*gam.* ...
    M12.*M33.*numPairs.*w_target.^2+M13.^2.*M22.*w_target.^4+...
    (-2).*M12.*M13.*M23.* ...
    w_target.^4+M11.*M23.^2.*w_target.^4+M12.^2.*M33.*w_target.^4+...
    (-1).*M11.*M22.*M33.* ...
    w_target.^4+(-1).*k23.*(k13.*M12+k12.*M13+...
    (-2).*d213.*dt2.*gam.*M11.* ...
    numPairs+(-2).*d212.*dt3.*gam.*M11.*numPairs+2.*d211.*dt3.*gam.* ...
    M12.*numPairs+2.*d211.*dt2.*gam.*M13.*numPairs+(-2).*M12.*M13.* ...
    w_target.^2+2.*M11.*M23.*w_target.^2)+k22.*((-1).*k33.*M11+k13.*M13+...
    (-2).*d213.* ...
    dt3.*gam.*M11.*numPairs+2.*d211.*dt3.*gam.*M13.*numPairs+(-1).* ...
    M13.^2.*w_target.^2+...
    M11.*M33.*w_target.^2)).^(-1).*(C4.*(k23.^2.*M11+k12.*k33.* ...
    M12+(-2).*d212.*dt2.*gam.*k33.*M11.*numPairs+2.*d213.*dt3.*gam.* ...
    k12.*M12.*numPairs+(-2).*d213.*dt2.*gam.*k13.*M12.*numPairs+2.* ...
    d211.*dt2.*gam.*k33.*M12.*numPairs+(-2).*d212.*dt3.*gam.*k12.* ...
    M13.*numPairs+2.*d212.*dt2.*gam.*k13.*M13.*numPairs+(-1).*k33.* ...
    M12.^2.*w_target.^2+k33.*M11.*M22.*w_target.^2+...
    (-1).*k13.*M13.*M22.*w_target.^2+k13.* ...
    M12.*M23.*w_target.^2+k12.*M13.*M23.*w_target.^2+...
    (-1).*k12.*M12.*M33.*w_target.^2+(-2) ...
    .*d213.*dt3.*gam.*M12.^2.*numPairs.*w_target.^2+...
    2.*d213.*dt2.*gam.*M12.* ...
    M13.*numPairs.*w_target.^2+...
    2.*d212.*dt3.*gam.*M12.*M13.*numPairs.*w_target.^2+( ...
    -2).*d212.*dt2.*gam.*M13.^2.*numPairs.*w_target.^2+...
    2.*d213.*dt3.*gam.* ...
```

```
    M11.*M22.*numPairs.*w_target.^2+(-2).*d211.*dt3.*gam.*M13.*M22.* ...
    numPairs.*w_target.^2+...
    (-2).*d213.*dt2.*gam.*M11.*M23.*numPairs.*w_target.^2+(-2) ...
    .*d212.*dt3.*gam.*M11.*M23.*numPairs.*w_target.^2+...
    2.*d211.*dt3.*gam.* ...
    M12.*M23.*numPairs.*w_target.^2+...
    2.*d211.*dt2.*gam.*M13.*M23.*numPairs.* ...
    w_target.^2+2.*d212.*dt2.*gam.*M11.*M33.*numPairs.*w_target.^2+...
    (-2).*d211.*dt2.* ...
    gam.*M12.*M33.*numPairs.*w_target.^2+M13.^2.*M22.*w_target.^4+...
    (-2).*M12.*M13.* ...
    M23.*w_target.^4+M11.*M23.^2.*w_target.^4+M12.^2.*M33.*w_target.^4+...
    (-1).*M11.*M22.* ...
    M33.*w_target.^4+(-1).*k23.*(k13.*M12+k12.*M13+...
    (-2).*d213.*dt2.*gam.* ...
    M11.*numPairs+(-2).*d212.*dt3.*gam.*M11.*numPairs+2.*d211.*dt3.* ...
    gam.*M12.*numPairs+2.*d211.*dt2.*gam.*M13.*numPairs+(-2).*M12.* ...
    M13.*w_target.^2+2.*M11.*M23.*w_target.^2)+...
    k22.*((-1).*k33.*M11+k13.*M13+(-2).* ...
    d213.*dt3.*gam.*M11.*numPairs+2.*d211.*dt3.*gam.*M13.*numPairs+( ...
    -1).*M13.^2.*w_target.^2+M11.*M33.*w_target.^2))+...
    (-2).*C2.*C3.*numPairs.*( ...
    d213.^2.*(k22.*M11+(-1).*k12.*M12+...
    (M12.^2+(-1).*M11.*M22).*w_target.^2)+ ...
    d213.*(d212.*((-2).*k23.*M11+k13.*M12+k12.*M13+(-2).*M12.*M13.* ...
    w_target.^2+2.*M11.*M23.*w_target.^2)+...
    d211.*(k23.*M12+(-1).*k22.*M13+(M13.*M22+( ...
    -1).*M12.*M23).*w_target.^2))+...
    d212.*(d212.*(k33.*M11+(-1).*k13.*M13+( ...
    M13.^2+(-1).*M11.*M33).*w_target.^2)+...
    d211.*((-1).*k33.*M12+k23.*M13+((-1) ...
    .*M13.*M23+M12.*M33).*w_target.^2))));
end
% ------------------------------------------------------------------------ %
% Compute resistance factor
function [Rf,alph]= computeResistance(P,f,tubeTypeNum)
% Computes resistance multiplier Rf as a function of fluid oscillation
% frequency.
% INPUT  1:  P - Struct containing all passing parameters.
% INPUT  2:  f - Frequency [Hz].
% INPUT  3:  tubeTypeNum - Number identifying F2MC tube.
% OUTPUT 1:  Rf - Resistance multiplier.
% OUTPUT 2:  alph - Non-dimensional frequency parameter.
% ------------------------------------------------------------------------ %
% Unpack variables:
config= P.Simulation.config;
mu= P.Tube.viscosity(tubeTypeNum);
rho= P.Tube.fluidDensity(tubeTypeNum);

nu= mu/rho;

switch config
    case {'coupled','uncoupled'}
        r= P.Tube.portRadius(tubeTypeNum);
        alph= r*sqrt(2*pi*f/nu);
        Rf= 0.166*alph^1.49;
    case 'coupled tunable'
        r1= P.Tube.portRadius1(tubeTypeNum);
        r2= P.Tube.portRadius2(tubeTypeNum);

        alph1= r1*sqrt(2*pi*f/nu);
```

```
        Rf1= 0.166*alph1^1.49;

        alph2= r2*sqrt(2*pi*f/nu);
        Rf2= 0.166*alph2^1.49;

        Rf= [Rf1 Rf2];
        alph= [alph1 alph2];
end
end
% ------------------------------------------------------------------------ %
% Compute inertance factor
function [I,b]= computeInertance(a)
% Estimate corrected inertance for a given non-dimensional parameter alpha.
% a= alpha

% Read lookup tables from files:
A= dlmread('alphabeta.txt');
B= dlmread('dimensionlessInertance.txt');

% Beta:
betA= A(:,1);
betB= B(:,1);

% Alpha/Beta curve:
aob= A(:,2);
Ivec= B(:,2);

% Compute Alpha/Beta using our alpha value:
betA_candidates= betA;
aob_candidates= a./betA_candidates;
c= aob_candidates - aob;

% Compare aob_candidates vs the aob curve:
b= interp1(c,betA,0);

% Find dimensionless inertance value corresponding to b:
I= interp1(betB,Ivec,b);
end
% ------------------------------------------------------------------------ %
% Estimate damping ratio using half-power method
function zet= halfPowerEst(w,mag)
% Estimates damping ratio by applying the half power method on a resonance
% peak.
% INPUT  1:  w - Vector containing frequency values [rad/s].
% INPUT  2:  mag - Vector containing response magnitude.
%
% OUTPUT 1:  zet - Estimate of damping ratio.

% ------------------------------------------------------------------------ %

% Find the frequency response peaks:
[~,locs]= findpeaks(mag);

% Extract ith natural frequency and peak magnitude:
wn= w(locs);
wn= wn(wn<16*2*pi);
peak= mag(locs);

% Trim frequency response:
w= w(w<16*2*pi);
mag= mag(w<16*2*pi);
```

```
% Make sure the system is not an absorber:
if (length(wn)>1)
elseif (length(wn)<1)
    zet= -1.2345;
    return
end

% Extract the peak and frequency of interest:
wn= wn(1);
peak= peak(1);
locs= locs(1);

% Determine half-power amplitude:
halfPowerAmp= peak/sqrt(2);

% Determine half-power crossings via linear interpolation:
segment1= mag(1:locs);
segment2= mag(locs+1:end);
w1= w(1:locs);
w2= w(locs+1:end);
w1= interp1(segment1,w1,halfPowerAmp);
w2= interp1(segment2,w2,halfPowerAmp);
w1= w1(1); w2= w2(1);

% Estimate damping ratio:
zet= (w2-w1)/(2*wn);
end
% ------------------------------------------------------------------------- %

function testData58(P)
% Process OH-58C vibration data.
% INPUT  2:  P - Struct containing all passing parameters.

% ------------------------------------------------------------------------- %
% Allow code to run without input:
if (~exist('P','var'))
    clc
    clear all
    close all
end

% Select mode shape type:
modeShapeType= 'baseline';
% modeShapeType= 'open';
% modeShapeType= 'closed';


% Unpack baseline data:
[~,~,A]= xlsread('baseline all.xlsx','BK.data');
freqVec= cell2mat(A(12:end,1));
freqVec= freqVec(2:end);
omegaVec= 2*pi*freqVec;

% Unpack open valve data:
[~,~,B]= xlsread('open all.xlsx','BK.data');
freqVecB= cell2mat(B(12:end,1));
freqVecB= freqVecB(2:end);
omegaVecB= 2*pi*freqVecB;

% Unpack closed valve data:
```

```
[~,~,C]= xlsread('closed all.xlsx','BK.data');
freqVecC= cell2mat(C(12:end,1));
freqVecC= freqVecC(2:end);
omegaVecC= 2*pi*freqVecC;

% Unpack nodes:
nodeList= A(2,:);
nodeList= nodeList(cell2mat(cellfun(@ischar,nodeList,'UniformOutput',0)));
numNodes= length(nodeList);

% Initialize data cell:
[dataCell dataCellB dataCellC]= deal(cell(4,numNodes));
[dataCell(1,:) dataCellB(1,:) dataCellC(1,:)]= deal(nodeList);

for k= 1:numNodes
    % Baseline
    reals= cell2mat(A(12:end,2*k));
    imags= cell2mat(A(12:end,2*k+1));
    reals= reals(2:end);
    imags= imags(2:end);
    mags= sqrt(reals.^2+imags.^2);
    mags= mags./(omegaVec.^2);
    phs= atan2(imags,reals);

    dataCell{2,k}= mags;
    dataCell{3,k}= phs;
    dataCell{4,k}= imags;

    % Open
    realsB= cell2mat(B(12:end,2*k));
    imagsB= cell2mat(B(12:end,2*k+1));
    realsB= realsB(2:end);
    imagsB= imagsB(2:end);
    magsB= sqrt(realsB.^2+imagsB.^2);
    magsB= magsB./(omegaVecB.^2);
    phsB= atan2(imagsB,realsB);

    dataCellB{2,k}= magsB;
    dataCellB{3,k}= phsB;
    dataCellB{4,k}= imagsB;

    % Closed
    realsC= cell2mat(C(12:end,2*k));
    imagsC= cell2mat(C(12:end,2*k+1));
    realsC= realsC(2:end);
    imagsC= imagsC(2:end);
    magsC= sqrt(realsC.^2+imagsC.^2);
    magsC= magsC./(omegaVecC.^2);
    phsC= atan2(imagsC,realsC);

    dataCellC{2,k}= magsC;
    dataCellC{3,k}= phsC;
    dataCellC{4,k}= imagsC;
end

% Split up data into x, y, z:
[xDataCell,yDataCell,zDataCell]= deal(cell(2,numNodes/3));
[xDataCellB,yDataCellB,zDataCellB]= deal(cell(2,numNodes/3));
[xDataCellC,yDataCellC,zDataCellC]= deal(cell(2,numNodes/3));
[temp_indX temp_indY temp_indZ]= deal(1);
for k= 1:numNodes
```

```
nodeName= dataCell{1,k};
nodeNum= str2double(nodeName(isstrprop(nodeName,'digit')));
if(~isempty(strfind(nodeName,'x')))
    % Baseline:
    xDataCell{1,temp_indX}= nodeNum;
    xDataCell{2,temp_indX}= dataCell{2,k};
    xDataCell{3,temp_indX}= dataCell{3,k};
    xDataCell{4,temp_indX}= dataCell{4,k};

    % Open:
    xDataCellB{1,temp_indX}= nodeNum;
    xDataCellB{2,temp_indX}= dataCellB{2,k};
    xDataCellB{3,temp_indX}= dataCellB{3,k};
    xDataCellB{4,temp_indX}= dataCellB{4,k};

    % Closed:
    xDataCellC{1,temp_indX}= nodeNum;
    xDataCellC{2,temp_indX}= dataCellC{2,k};
    xDataCellC{3,temp_indX}= dataCellC{3,k};
    xDataCellC{4,temp_indX}= dataCellC{4,k};

    temp_indX= temp_indX+1;
elseif(~isempty(strfind(nodeName,'y')))
    % Baseline:
    yDataCell{1,temp_indY}= nodeNum;
    yDataCell{2,temp_indY}= dataCell{2,k};
    yDataCell{3,temp_indY}= dataCell{3,k};
    yDataCell{4,temp_indY}= dataCell{4,k};

    % Open:
    yDataCellB{1,temp_indY}= nodeNum;
    yDataCellB{2,temp_indY}= dataCellB{2,k};
    yDataCellB{3,temp_indY}= dataCellB{3,k};
    yDataCellB{4,temp_indY}= dataCellB{4,k};

    % Closed:
    yDataCellC{1,temp_indY}= nodeNum;
    yDataCellC{2,temp_indY}= dataCellC{2,k};
    yDataCellC{3,temp_indY}= dataCellC{3,k};
    yDataCellC{4,temp_indY}= dataCellC{4,k};

    temp_indY= temp_indY+1;
else
    % Baseline:
    zDataCell{1,temp_indZ}= nodeNum;
    zDataCell{2,temp_indZ}= dataCell{2,k};
    zDataCell{3,temp_indZ}= dataCell{3,k};
    zDataCell{4,temp_indZ}= dataCell{4,k};

    % Open:
    zDataCellB{1,temp_indZ}= nodeNum;
    zDataCellB{2,temp_indZ}= dataCellB{2,k};
    zDataCellB{3,temp_indZ}= dataCellB{3,k};
    zDataCellB{4,temp_indZ}= dataCellB{4,k};

    % Closed:
    zDataCellC{1,temp_indZ}= nodeNum;
    zDataCellC{2,temp_indZ}= dataCellC{2,k};
    zDataCellC{3,temp_indZ}= dataCellC{3,k};
    zDataCellC{4,temp_indZ}= dataCellC{4,k};
```

```
        temp_indZ= temp_indZ+1;
    end
end


% Sort cell array
[~,ix]= sort([xDataCell{1,:}],'ascend');
xDataCell= xDataCell(:,ix);

[~,ix]= sort([xDataCellB{1,:}],'ascend');
xDataCellB= xDataCellB(:,ix);

[~,ix]= sort([xDataCellC{1,:}],'ascend');
xDataCellC= xDataCellC(:,ix);

[~,iy]= sort([yDataCell{1,:}],'ascend');
yDataCell= yDataCell(:,iy);

[~,iy]= sort([yDataCellB{1,:}],'ascend');
yDataCellB= yDataCellB(:,iy);

[~,iy]= sort([yDataCellC{1,:}],'ascend');
yDataCellC= yDataCellC(:,iy);

[~,iz]= sort([zDataCell{1,:}],'ascend');
zDataCell= zDataCell(:,iz);

[~,iz]= sort([zDataCellB{1,:}],'ascend');
zDataCellB= zDataCellB(:,iz);

[~,iz]= sort([zDataCellC{1,:}],'ascend');
zDataCellC= zDataCellC(:,iz);

% Node indices:
nodeVec= cell2mat(zDataCell(1,:));

% Mode shapes
if (1)
    % Frequency response
    if (1)
        magZ= zDataCell{2,length(nodeVec)};
        figure(3)
        semilogx(freqVec,20*log10(magZ),'LineWidth',3)
        xlim([1 100])
        xlabel('Frequency [Hz]')
        ylabel('Tip Displacement/Tip Force [dB]')
        hold all
        if (exist('B','var')&& 0)
            magB= zDataCellB{2,length(nodeVec)};
            semilogx(freqVecB,20*log10(magB),'LineWidth',3)
        end
        if (exist('C','var') && 0)
            magC= zDataCellC{2,length(nodeVec)};
            semilogx(freqVecC,20*log10(magC),'LineWidth',3)
        end
        legend('Baseline','Open Valve','Closed Valve')
    end

    modeInd2= freqVec==2.625;
    modeInd5= freqVec==5.375;
    modeInd11= freqVec==11.25;
```

```
    [msAmp2z,msAmp5z,msAmp11z]= deal(zeros(length(nodeVec),1));
    switch modeShapeType
        case 'baseline'
            zDC= zDataCell;
        case 'open'
            zDC= zDataCellB;
        case 'closed'
            zDC= zDataCellC;
        otherwise
            error('what')
    end
    for k= 1:length(nodeVec)
%         mags_kx= xDataCell{2,k};
%         mags_ky= yDataCell{2,k};
        mags_kz= zDC{2,k};

%         phs_kx= xDataCell{3,k};
%         phs_ky= yDataCell{3,k};
        phs_kz= zDC{3,k};

%         imags_kx= mags_kx.*sin(phs_kx);
%         imags_ky= mags_ky.*sin(phs_ky);
        imags_kz= mags_kz.*sin(phs_kz);

        msAmp11z(k)= imags_kz(modeInd11);
        msAmp5z(k)= imags_kz(modeInd5);
        msAmp2z(k)= imags_kz(modeInd2);
    end

    % Tailboom coordinates:
    xVec=     [-1 0 8  16 32 40 48 56 72 120 150]*0.0254;

    if (exist('P','var'))
        figure(P.Figure.fig1)
    else
        figure(1)
    end
    hold all

    % Shift mode shapes to zero at the root:
    if (0)
        msAmp5z= msAmp5z-msAmp5z(2);
        msAmp11z= msAmp11z-msAmp11z(2);
        msAmp2z= msAmp2z-msAmp2z(2);
    end

    plot(xVec,msAmp11z)
    xlabel('Axial Position [m]')
    ylabel('Im[Displacement] [m]')
    legend('Rigid Mode','1st Mode (5 Hz)','2nd Mode (11 Hz)')
    title('Vertical Bending Mode Shapes')

    if (exist('P','var'))
        % Compute slope difference between F2MC tube attachment points:
        slopeVec= diff(msAmp5z)./diff(xVec)';
        rootSlope= interp1(xVec(1:end-1),slopeVec,0);
        endSlope= interp1(xVec(1:end-1),slopeVec,P.Tube.length);
        slopeDiff= endSlope-rootSlope;

        fprintf('\n\nExperimental Slope Difference, Mode 1: %g\n\n',slopeDiff)
```

```
        slopeVec= diff(msAmp11z)./diff(xVec)';
        rootSlope= interp1(xVec(1:end-1),slopeVec,0);
        endSlope= interp1(xVec(1:end-1),slopeVec,P.Tube.length);
        slopeDiff= endSlope-rootSlope;

        fprintf('Experimental Slope Difference, Mode 2: %g\n\n',slopeDiff)
    end
end
end

function P_out= convertUnits(P)
% convertUnits.m
% Converts frequency magonse magnitude data to the proper display units.
% INPUT  1:  P - Data structure containing system parameter values.
% OUTPUT 1:  P_out - Data structure containing converted units.
%
% ------------------------------------------------------------------------ %

% Unpack variables:
outFreq= P.Frequency.outFreq;
sysType= P.SysType;
units= P.Simulation.units;
TFmode= P.TFmode;
if (isfield(P.Frequency,'magBeam'))
    magBeam= P.Frequency.magBeam;
end
if (isfield(P.Frequency,'magBeamTube'))
    magBeamTube= P.Frequency.magBeamTube;
end
if (isfield(P.Frequency,'magBeamTubeRes'))
    magBeamTubeRes= P.Frequency.magBeamTubeRes;
end

% Select forcing frequency at which to display output:
switch outFreq
    case '1/rev'
        ind= 10;
    case '4/rev'
        ind= 2400;
    otherwise
        error('Please select a forcing frequency')
end
P.Plot.ind= ind;

% Select system type, i.e. bare beam or fluidlastic:
switch sysType
    case 'Tailboom'
        mag_out= magBeam(ind);
    case 'Fluidlastic'
        mag_out= magBeamTube(ind);
    otherwise
        error('Please select a system.')
end

switch TFmode
    case 1
        % Tip force to tip displacement
        switch units
            case 'English'
                P.Frequency.magBeam= magBeam*175.126835;
```

```
            if (exist('magBeamTube','var'))
                P.Frequency.magBeamTube= magBeamTube*175.126835;
                P.Frequency.magBeamTubeRes= magBeamTubeRes*175.126835;
            end
            t_str= 'Tip Displacement Per Unit Tip Force [in/lbf]';
            mag_out= mag_out*175.126835;
            str_out= [num2str(mag_out) ' in/lbf'];
        case 'SI'
            t_str= 'Tip Displacement Per Unit Tip Force [m/N]';
            str_out= [num2str(mag_out) ' m/N'];
    end
    y_str= 'Tip Displacement per Unit Tip Force [dB]';
    plot_type= 'dB';
case 2
    % Tip force to root moment. Normalize by beam length:
    magBeam= magBeam/P.Beam.length;
    magBeamTube= magBeamTube/P.Beam.length;
    switch units
        case 'English'
            error('INCOMPLETE')
        case 'SI'
            error('INCOMPLETE')
    end
    y_str= 'M(0)/F(L) [dB]';
    tStr= 'Root Moment Per Unit Tip Force [N-m/N]';
case 3
    % Tip force to root shear:
    switch units
        case 'English'
            error('INCOMPLETE')
        case 'SI'
            error('INCOMPLETE')
    end
    y_str= 'V(0)/w(L) [dB]';
    tStr= 'Root Shear Per Unit Tip Force [N/N]';
case 4
    % Tip force to internal pressure:
    switch units
        case 'English'
            if (exist('magBeamTube','var'))
                sf4= 0.00014504/0.2248;
                P.Frequency.magBeamTube= magBeamTube*sf4;
                P.Frequency.magBeamTubeRes= magBeamTubeRes*sf4;
            end
            t_str= 'Fluid Pressure Per Unit Tip Force [psi/lbf]';
            mag_out= mag_out*sf4;
            str_out= [num2str(mag_out) ' psi/lbf'];
            y_str= 'log_{10}|p_1/F(L)| [psi/lbf]';
        case 'SI'
            t_str= 'Fluid Pressure Per Unit Tip Force [Pa/N]';
            str_out= [num2str(mag_out) ' Pa/N'];
            y_str= 'log_{10}|p_1/F(L)| [Pa/N]';
    end
    plot_type= 'loglog';
case 5
    % Tip force to axial force:
    switch units
        case 'English'
            t_str= 'Tube Axial Force Per Unit Tip Force [lbf/lbf]';
            str_out= [num2str(mag_out) ' lbf/lbf'];
```

```
            y_str= 'log_{10}|F_t/F(L)| [lbf/lbf]';
        case 'SI'
            t_str= 'Tube Axial Force Per Unit Tip Force [N/N]';
            str_out= [num2str(mag_out) ' N/N'];
            y_str= 'log_{10}|F_t/F(L)| [N/N]';
    end
    plot_type= 'loglog';
case 6
    % Tip acceleration to internal pressure:
    switch units
        case 'English'
            t_str= 'Tube Pressure Per Unit Tip Acceleration [psi/g]';
            str_out= [num2str(mag_out) ' psi/g'];
            y_str= 'log_{10}|F_t/F(L)| [psi/g]';
            sf6= 0.000145037738/0.101971621;
            P.Frequency.magBeamTube= magBeamTube*sf6;
        case 'SI'
            t_str= 'Tube Pressure Per Unit Tip Acceleration [psi/g]';
            str_out= [num2str(mag_out) ' psi/g'];
            y_str= 'log_{10}|F_t/F(L)| [psi/g]';
    end
    plot_type= 'loglog';
case 7
    % Tip acceleration to axial force:
    error('INCOMPLETE')
case 8
    % Tip force to volumetric flow rate:
    switch units
        case 'English'
            t_str= 'Flow Rate Per Unit Tip Force [in^3/lbf-s]';
            str_out= [num2str(mag_out) ' in^3/lbf-s'];
            y_str= 'log_{10}|Q/F(L)| [in^3/lbf-s]';
            sf8= 61023.7/0.224808943;
            P.Frequency.magBeamTube= magBeamTube*sf8;
            P.Frequency.magBeamTubeRes= magBeamTubeRes*sf8;
        case 'SI'
            t_str= 'Flow Rate Per Unit Tip Force [m^3/N-s]';
            str_out= [num2str(mag_out) ' m^3/N-s'];
            y_str= 'log_{10}|Q/F(L)| [m^3/N-s]';
    end
    plot_type= 'loglog';
case 9
    % Tip force to tip acceleration:
    P.Frequency.magBeam= magBeam/9.81;
    if (exist('magBeamTube','var'))
        P.Frequency.magBeamTube= magBeamTube/9.81;
        P.Frequency.magBeamTubeRes= magBeamTubeRes/9.81;
    end
    switch units
        case 'English'
            sf9= 0.2248089;
            P.Frequency.magBeamTube= magBeamTube/sf9;
            P.Frequency.magBeamTubeRes= magBeamTubeRes/sf9;
            t_str= 'Tip acceleration per Unit Tip Force [g/lbf]';
            str_out= [num2str(mag_out) ' g/lbf'];
        case 'SI'
            t_str= 'Tip acceleration per Unit Tip Force [g/N]';
            str_out= [num2str(mag_out) ' g/N'];
    end
    y_str= 'a(L)/F(L) [dB]';
    plot_type= 'dB';
```

```
    case 10
        % Tip force to accumulator pressure:
        switch units
            case 'English'
                sf10= 0.0001450377/0.224809;
                P.Frequency.magBeamTube= magBeamTube*sf10;
                P.Frequency.magBeamTubeRes= magBeamTubeRes*sf10;
                t_str= 'Accumulator Pressure Per Unit Tip Force [psi/lbf]';
                str_out= [num2str(mag_out) ' psi/lbf'];
                y_str= 'log_{10}|p_a/F(L)| [psi/lbf]';
            case 'SI'
                t_str= 'Accumulator Pressure Per Unit Tip Force [Pa/N]';
                str_out= [num2str(mag_out) ' Pa/N'];
                y_str= 'log_{10}|p_a/F(L)| [Pa/N]';
        end
        plot_type= 'loglog';
    otherwise
        error('Choose a transfer function')
end

P.Plot.title= t_str;
P.Plot.output= str_out;
P.Plot.ylabel= y_str;
P.Plot.type= plot_type;

P_out= P;
end
```

VITA

Kentaro Miura

Education

The Pennsylvania State University, University Park, PA
Ph.D. in Mechanical Engineering (May 2016)
M.S. in Mechanical Engineering (August 2015)

Cornell University
B.S. in Mechanical Engineering, cum laude (August 2011)

Awards

Bell Graduate Fellowship (2012-2015)
First Place, US Army Research Laboratory Summer Symposium (August 2014)
Third Place, AIAA Region I Student Conference (2013)
Best Paper Award, Penn State College of Engineering Research Symposium (2013)
College of Engineering Fellowship (2011-2012)

Publications

[1] Krott, M.J., Miura, K, Rahn, C.D., Smith, E.C., "Finite Element Modeling of Fluidic Flexible Matrix Composite Treatments for Bending and Torsional Vibration Control," AIAA SciTech 2016, San Diego, CA, 2016.
[2] Miura, K, Zhu, B, Rahn, C.D., Smith, E.C., Bakis, C.E., "Vibration Isolation of a Cantilever Beam with Fluidic Flexible Matrix Composite Tubes," ASME IDETC – 27th Conference on Mechanical Vibration and Noise, Boston, MA, 2015.
[3] Miura, K, Krott, M.J., Smith, E.C., Rahn, C.D., Romano, P.Q., "Experimental Validation of Tailboom Vibration Control Using Fluidic Flexible Matrix Composite Tubes," AHS International 71st Annual Forum Proceedings, Virginia Beach, VA, 2015.
[4] Krott, M.J., Miura, K, LaBarge, S.M., Rahn, C.D., Smith, E.C., and Romano, P.Q., "Tube Compliance Effects on Fluidic Flexible Matrix Composite Devices for Rotorcraft Vibration Control," AIAA SciTech 2015, Kissimmee, FL, 2015.
[5] Miura, K, Krott, M.J., Smith, E.C., Rahn, C.D., and Romano, P.Q., "Experimental Demonstration of Tailboom Vibration Reduction Using Fluidic Flexible Matrix Composite Tubes," AHS International 70th Annual Forum Proceedings, Montreal, QC, 2014.
[6] Miura, K, Rahn, C.D., and Smith, E.C., "Passive Tailboom Vibration Control Using Fluidic Flexible Matrix Composite Tubes," AIAA SciTech 2014, National Harbor, MD, 2014.

Under preparation/review
[1] Miura, K, Smith, E.C., Rahn, C.D., "Modeling and Design of a Tailboom Vibation Absorber Using Fluidic Flexible Matrix Composite Tubes," Journal of the American Helicopter Society, under review.
[2] Miura, K, Krott, M.J., Smith, E.C., Rahn, C.D., "Experimental Validation of Tailboom Vibration Control Using Fluidic Flexible Matrix Composite Tubes," Journal of the American Helicopter Society, in preparation.