Modification of a 3D Haptic Robotic Device to Study Upper Limb Movement and Proprioceptive Reflexes

Philip Bernardus Leopold Raaphorst

TU Delft

Northwestern Medicine®
Feinberg School of Medicine

ME51035 ME-BMD MSc Thesis

Modification of a 3D Haptic Robotic Device to Study Upper Limb Movement and Proprioceptive Reflexes

by

Philip Bernardus Leopold Raaphorst

to obtain the degree of Master of Science for the Mechanical Engineering - Biomechanical Design Master at the Delft University of Technology, to be defended publicly on Tuesday September 22, 2025.

Student Number 4094395

Thesis Committee

Chair	Prof. Dr. Frans C.T. van der Helm, Biomechanical Engineering, TU Delft
Supervisor	Prof. Dr. Julius P.A. Dewald, Biomedical Engineering, Northwestern University
Mentor	Dr. Thomas Plaisier, Biomedical Engineering, Northwestern University
Committee Member	Prof. Dr. Ir. Alfred C. Schouten, Biomechanical Engineering, TU Delft

PREFACE

The completion of this thesis marks the end of a long and challenging journey, which would not have been possible without the support, guidance, and encouragement of many people.

First and foremost, I would like to express my deep gratitude to my supervisors for their unwavering commitment throughout these years. Their expertise and advice have been invaluable in helping me bring this project to completion. I am also very thankful to Northwestern University for their hospitality during my stay. Jules ensured I felt welcome and created opportunities for me to explore different parts of the department, allowing me to learn and experience as much as possible. Thomas was always ready to provide advice and generously carried out the experiments on my behalf while I was back in The Netherlands. Frans has been a constant source of guidance, always helping me stay focused on the end goal.

I owe a special thanks to my family for their patience, encouragement, and support over the years. Their belief in me made this long path less daunting, even when it felt endless at times. I am especially grateful to Vera, whose encouragement and support helped me push through the final stretch. The last months have been exhausting, and I couldn't have crossed the finish line without the steadfast support of Vera, Leonie, and Theodoor, who stood by me in every possible way.

To all who contributed to this journey, directly or indirectly, I extend my sincere appreciation.

P.B.L. Raaphorst
Delft, September 2025

SUMMARY

The New Arm Coordination Trainer in 3D (NACT-3D) is a mechatronic device developed to investigate upper limb movement and quantify human proprioceptive reflexes through haptic interaction. This thesis evaluates whether the NACT-3D meets the technical requirements necessary to perturb the human arm at frequencies suitable for system identification, while simultaneously generating realistic 3D haptic environments. The NACT-3D integrates admittance and impedance control with multimodal sensing, offering capabilities beyond existing robotic platforms. Experimental testing was conducted using perturbation signals and optical tracking to assess the device's bandwidth, while SPACAR simulations were used to model theoretical performance under various conditions. Results show that the NACT-3D's actuation system has sufficient force output and bandwidth; however, the current manipulator's high inertia and mechanical play significantly limit performance. Without the manipulator, the system achieves bandwidths above 40 Hz, but with the manipulator attached, performance drops below 6 Hz. Simulation results indicate that a redesigned manipulator can restore performance, enabling the system to meet its operational target. These findings demonstrate that although the current configuration is inadequate for high-speed perturbations, targeted improvements to the manipulator can enable the NACT-3D to become a robust tool for studying neuromechanical control and reflex modulation.

CONTENTS

1 INTRODUCTION

Quantifying human motor function and proprioceptive reflexes is critical to understanding neuromuscular control, especially in individuals with neurological impairments. The New Arm Coordination Trainer in 3D (NACT-3D) is a custom-designed, mechatronic device developed to generate virtual haptic environments, apply dynamic perturbations, and measure interactive forces and movements of the human upper limb. With its multimodal capabilities, including compatibility with electromyographic (EMG) and electroencephalographic (EEG) recordings, the NACT-3D supports experimental studies on the neuromechanics of movement and reflex modulation.

Unlike conventional devices that operate in only one or two dimensions and primarily use impedance control, the NACT-3D provides a large three-dimensional workspace and supports both admittance and impedance control. This dual-mode capability enables it to render complex haptic environments and deliver high-frequency perturbations suitable for system identification of the human arm. Many commercial systems fall short in delivering precise perturbations due to limitations inherent to impedance control. For instance, while the Haptic Master uses admittance control, its restricted workspace and limited ability to apply rapid position perturbations reduce its effectiveness. In contrast, the NACT-3D was purpose-built to overcome these constraints, combining high-performance actuation, adaptable control architecture, and an anatomically suitable range of motion.

Despite its promising design, the NACT-3D has not yet been experimentally validated for key performance requirements: namely, the ability to (1) generate realistic 3D haptic environments and (2) deliver perturbations at bandwidths sufficient to quantify human proprioceptive reflexes. Therefore, the central aim of this thesis is to evaluate whether the current configuration of the NACT-3D meets the necessary specifications for reflex characterization and dynamic task simulation.

To this end, a two-pronged approach is undertaken. First, physical experiments assess the current performance of the NACT-3D in terms of bandwidth, workspace coverage, and force capability. Second, a virtual model is developed using SPACAR to simulate system dynamics and predict the potential benefits of design modifications. Together, these methods will determine whether the NACT-3D can be used for high-fidelity perturbation experiments and support neuromechanical investigations across a clinically relevant workspace.

This thesis is structured as follows: A technical and clinical background is presented necessary to understand stretch reflex hyperexcitability and its measurement. Corresponding performance requirements for the NACT-3D are defined, and the experimental and simulation-based evaluation methods are outlined. The results of static analysis, dynamic perturbation experiments, and virtual simulations are presented. The findings, along with limitations and implications are discussed followed by a summary of key insights and recommendations for further development.

By systematically combining hardware testing and simulation, this study determines whether the NACT-3D can achieve the performance thresholds required for its intended research applications. The experimental evaluation includes five key investigations:

- Force calculations to estimate manipulator deflections under maximum human-generated loads.
- System identification experiments using multi-sine perturbations to quantify bandwidth with the manipulator.
- Actuation analysis, isolating the drive system and controller by means of system identification experiments without the manipulator.
- SPACAR simulations to model the manipulator's mechanical dynamics.
- SPACAR-based estimations to assess the performance of an improved manipulator configuration.

2 Background Information

This chapter provides a theoretical and clinical context for the NACT-3D as a tool to study human upper limb movement, proprioceptive reflexes, and stretch reflex hyperexcitability (spasticity). First, a brief explanation of a normal stretch reflex loop is provided followed by an explanation of how altered neural control leads to spasticity. The limitations of current clinical assessment techniques are noted as well as the shortcomings of most currently available measurement devices. The NACT-3D combines a large 3D workspace, high bandwidth perturbation capability, and multimodal sensing, making it uniquely suited to bridge the gaps in the current technology for quantitative human reflex research. These insights motivate the experimental and modelling studies that follow in later chapters.

2.1. The Stretch Reflex Loop

The stretch reflex is the fastest sensorimotor feedback mechanism. In the human arm, it responds within tens of milliseconds. A muscle stretch elongates fibres inside the muscle spindle, this mechanical event triggers signals to be sent via afferent sensory nerve fibres to the spinal cord: group Ia (length + velocity) and group II (length-only) fibres (Matthews & Bagby, 1974; Prochazka, 1996). Those afferent fibres make monosynaptic (and, for group II, polysynaptic) excitatory connections to homonymous alpha motoneurons, which rapidly return the muscle to its pre-stretch length. Gamma motor neurons adjust spindle sensitivity, pre-tuning the reflex gain for the current task. Descending pathways (corticospinal and brain stem tracts) superimpose task-dependent excitation and inhibition, enabling voluntary override and context-specific reflex modulation. The bandwidth and latency are important to understand the

stretch reflex. At the shoulder, the closed-loop bandwidth is roughly 2 Hz, at the elbow 2-6 Hz. Voluntary motions faster than 6 Hz are therefore essentially open-loop (Bennett et al., 1992; Shadmehr & Mussa-ivaldi, 1994).

2.2. Strech Reflex Hyperexcitability (Spasticity)

The stretch reflex is a closed-loop control system that maintains muscle position and stiffness through sensory feedback. The system's gain is finely tuned to ensure stable limb movements. However, neurological injuries such as stroke or spinal cord injury manifest as upper motor neuron lesions. This disrupts the descending pathways and hinders the brain's ability to modulate the gain.

In stroke, pathways that fine-tune spinal reflexes are primarily damaged while other, secondary pathways remain active. This resulting imbalance causes an increase in motor neuron activity, muscles respond too sharply to stretch (the system’s feedback gain is too high) (Li & Francisco, 2015; Sheean, 2002).

In spinal cord injury, the descending pathways from brain to muscle spindles are critically damaged or severed entirely. Spinal neurons gradually compensate for the missing neural activity by becoming extraordinarily sensitive. As a result, minimal muscle stretches illicit disproportionately strong muscle contractions, making the reflex loop hyper-responsive (an increase in sensitivity of the system’s feedback sensors) (Dietz & Sinkjaer, 2007; Heckman & Enoka, 2012).

In both scenarios, the firing threshold of the alpha motor neurons is lowered, and the system is essentially unable to moderate its response. This results in involuntary contractions and exaggerated muscle stiffness, characteristic of stretch reflex hyperexcitability (spasticity). Treating spasticity as a malfunctioning control system shows why precise assessment tools, such as the NACT-3D, are essential to objectively study and measure spasticity.

2.3. Clinical Assessment Tools and Limitations

The current gold standard for assessment are clinical scales such as the Modified Ashworth Scale (MAS) and Modified Tardieu Scale (MTS) (Guo et al., 2022). They are quick and equipment free but lack objectivity, reliability, repeatability, and sensitivity. The scores heavily rely on a clinician’s subjective judgement and observational skills, and the large steps between scores on the scales can correspond to vast changes in reflex torque. Objective approaches exist such as hand-held dynamometry, perturbation devices, EMG-only metrics, but most only partially capture the underlying mechanisms

(mechanical or neural). Additionally, most operate in 1D workspaces and below the bandwidth required to meaningfully perturb and measure reflexes.

Technology-Assisted Measurement

Recent systematic reviews highlight three broad measurement type categories (Measuring Stretch Reflex Hyperexcitability review):

- Mechanical - Kinetic and kinematic measures to evaluate musculoskeletal and musculotendon properties.
- Electrophysiological - Electrically based measurements that quantify the neurophysiological properties.
- Combined - A combination of mechanical and electrophysiological measures.

The measurement types were further classified based on four input types:

- No input - Measurements taken at rest.
- Voluntary - The patient actively moves during measurement.
- Manual - A clinician moves the patient for measurement.
- Controlled - A device is used to move/perturb the patient in a controlled manner.

Combined measurements made with controlled inputs consistently provide the highest accuracy, reliability, and repeatability. Such systems can disentangle neural and non-neural components over precisely defined inputs. The main drawbacks of existing systems are the limited dimensions (1D and 2D), the confined workspace, the restricted perturbation bandwidth, and poor integration with EEG.

The NACT-3D was conceived to overcome the limitations of currently available measurement devices.

- Workspace - The work area largely matches the 95th percentile of the human reach envelope.
- Control flexibility - Both impedance and admittance control are possible, allowing for haptic environment rendering as well as high bandwidth perturbations.
- Perturbation bandwidth - The targeted bandwidth of 20 Hz at the endpoint is an order of magnitude above a human arm's bandwidth, ensuring perturbations are appropriate.
- Multimodal data capture - The NACT-3D has 6-DOF endpoint force & torque measurements, measures joint kinematics & dynamics, and all data can be time-locked to EMG and/or EEG measurements.

- Clinical relevance - The NACT-3D is well equipped to systematically quantify reflexes and movement throughout the human arm's functional workspace, something no existing platform can currently do.

3 EXPERIMENTAL APPARATUS

This chapter describes the NACT-3D, its design purpose, and how it functions. The NACT-3D facilitates the study of a human upper limb by applying and measuring forces during movement. It is comprised of a manipulator that interacts with a participant's forearm, and it can generate virtual haptic environments. The major components and functionalities are described below.

3.1. NACT-3D Purpose

The NACT-3D is designed to measure the forces and torques generated by a human arm and to apply programmable forces. It can perform static force measurements, perturb the human arm during motion, and simulate virtual haptic environments. The system can switch between admittance and impedance control, allowing for position, velocity, and torque-based tasks or perturbations. It supports integration with electroencephalography (EEG) and electromyography (EMG) recording equipment. The simultaneous measurement of brain, muscle, and movement data allows for the study of reflex modulation during upper extremity movements.

3.2. Device Design

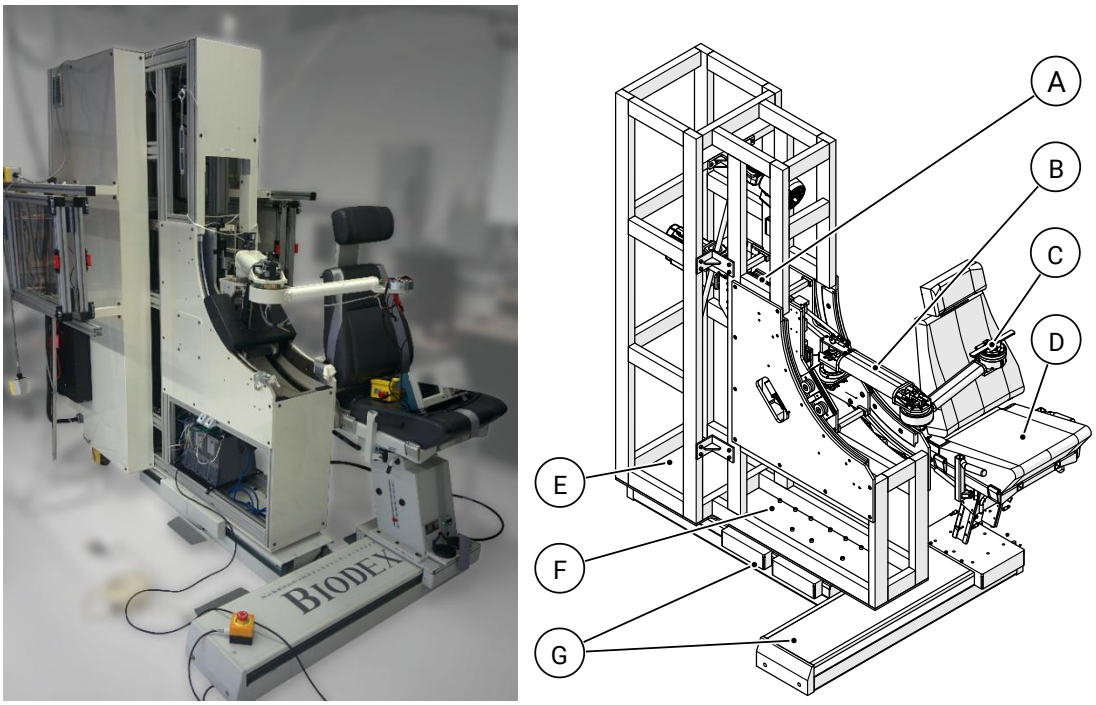


Fig. 1. Photo of the NACT-3D (left) and a labelled overview (right): (A) actuation linkage and servos, (B) manipulator, (C) point of interaction with study participant, (D) ergonomic chair, (E) space for the E-box with all electronics and safety circuit, (F) space for the servo drives, and (G) rails to allow for quick right/left handed configuration switching.

The NACT-3D is a floor-mounted mechatronic system with a four degree-of-freedom robotic manipulator. These four degrees of freedom correspond to: (1 & 2) the rotation of two manipulator segments in plane, enabling two-dimensional translation of the manipulator's end point in a horizontal plane; (3) vertical translation of the entire manipulator, enabling end point motion in the z-dimension; and (4) adjustment of the tilt angle of the manipulator's working plane, enabling the reorientation of the otherwise horizontal planar manipulator motion plane. A more detailed description of the manipulator is provided in 3.3. Manipulator Design.

Fig. 1 shows the main components, including the support rails, ergonomic chair (Sisto & Dyson-Hudson, 2007)(Biodex Medical Systems, Shirley, NY), electronics, servo drives, actuation system, and manipulator. A study participant's forearm interfaces with the manipulator via an orthosis. A six-degree-of-freedom load cell is located at the point of interface between the orthosis and manipulator to measure the interaction forces and torques. The support rails and quick-release mechanisms on the manipulator's joints allow the NACT-3D to be reconfigured to accommodate both left and right arm measurements (Fig. 2 and Fig. 3).

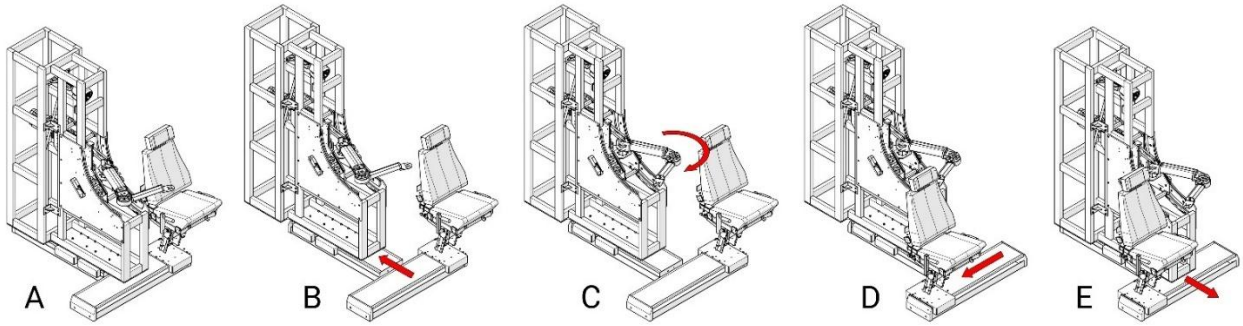


Fig. 2. Changing the NACT-3D's configuration: (A) right handed configuration, (B) slide the main frame backwards, (C) flip the manipulator configuration using the quick-release latches at the elbow and should joints, (D) slide the BioDex chair to the other side, and (E) slide the main frame forwards again.

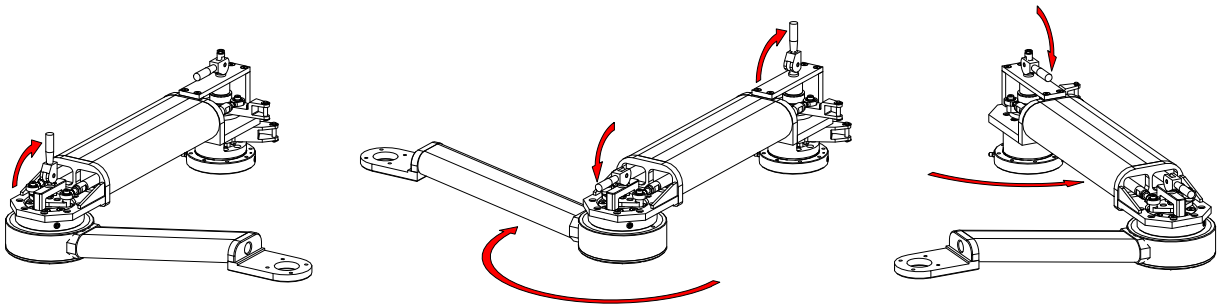


Fig. 3. Changing the NACT-3D Manipulator's configuration: (left) open the elbow latch, (middle) rotate the lower arm 180 degrees, close the elbow latch, open the shoulder latch, (right) rotate the upper arm, and close the shoulder latch.

3.3. Manipulator Design

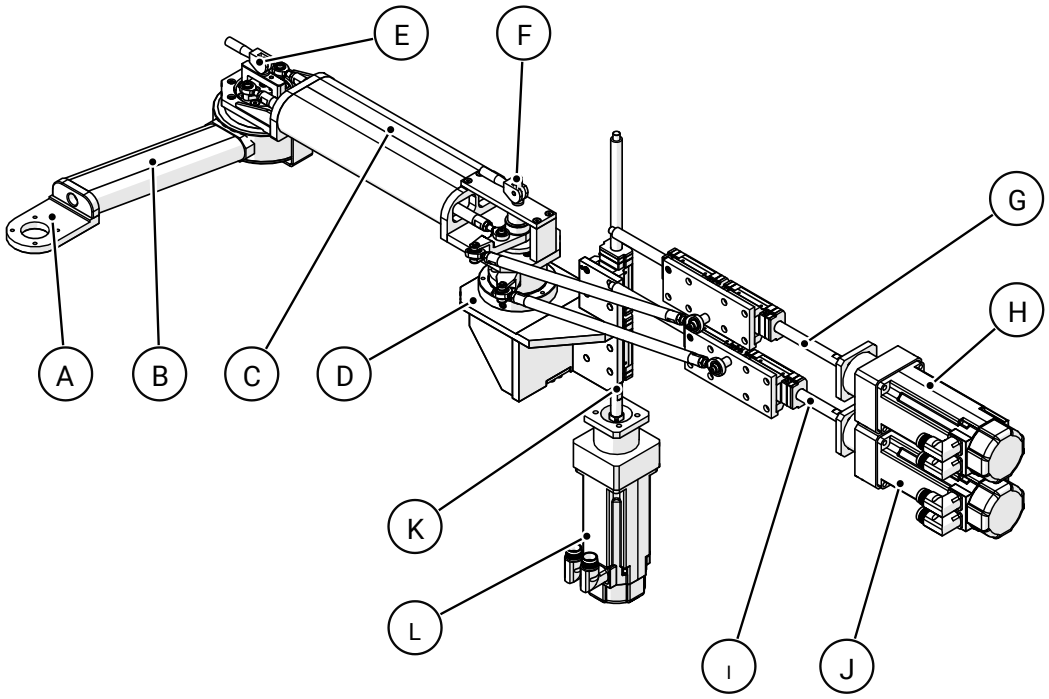


Fig. 4. Labelled overview of the NACT-3D's manipulator: (A) manipulator end point and attachment point for load cell and wrist orthosis, (B) manipulator lower arm, (C) manipulator upper arm, (D) base plate that supports the manipulator, (E) elbow joint quick release latch, (F) shoulder joint quick release latch, (G) spindle and carriage for lower arm actuation, (H) servo for lower arm, (I) spindle and carriage for upper arm actuation, (J) servo for upper arm, (K) spindle and carriage for vertical actuation of base plate, and (J) servo for vertical actuation.

The NACT-3D's manipulator, shown in Fig. 4, mimics a human arm and consists of two linked segments. The two segments are connected via joints and can each rotate in one dimension. Both segments rotate in the same plane and can be individually rotated. Two servo-driven spindles convert rotational motion to linear motion and move carriages which in turn are connected to push-pull rods. The push-pull rods convert the linear motion to rotational motion at the segment's joints via levers. This configuration enables two-dimensional translation of the manipulator's endpoint in a horizontal plane.

The manipulator segments are attached to the NACT-3D's frame via a plate connected to a vertical carriage at the manipulator's 'shoulder-joint'. Vertical adjustment of the manipulator is achieved with a third servo-driven spindle that moves the 'shoulder-joint' of the manipulator along the z-axis. The manipulator's push-pull rods are connected to the carriages and levers with ball-joints so that the actuation system does not also need to be moved vertically. As a result, a vertical displacement of the 'shoulder-joint' will also result in x- and y-translations of the end point if the manipulator servos remain stationary. A fourth servo sets the tilt angle of the manipulator's working plane by rotating the 'shoulder-joint' around the y-axis, but this motion is excluded from active control during experiments. Fig. 5 illustrates the motion envelope of the arm-like

manipulator, a detailed description of how the manipulator's upper arm and lower arm are actuated is given in Fig. 6 and Fig. 7 respectively.

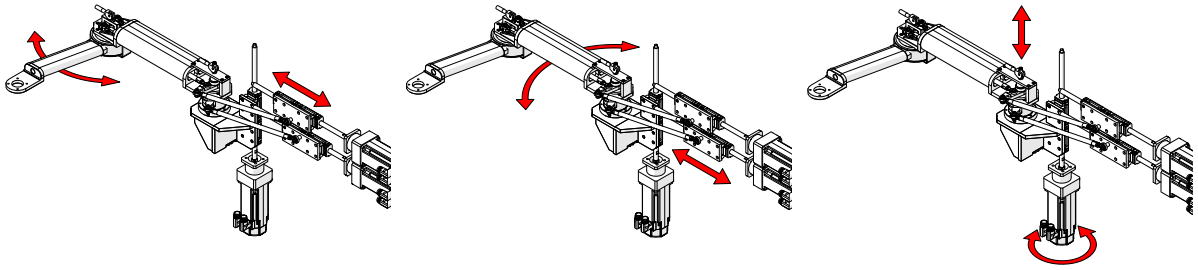


Fig. 5. Representation of the manipulator's acuation system: (left) lower arm orientation angle is controlled by driving a servo that uses a spindle to translate rotation into linear displacement, (middle) similarly the upper arm's orientation angle is controlled with a servo and spindle system, and (right) the entire manipulator can be translated vertically with a servo and spindle system.

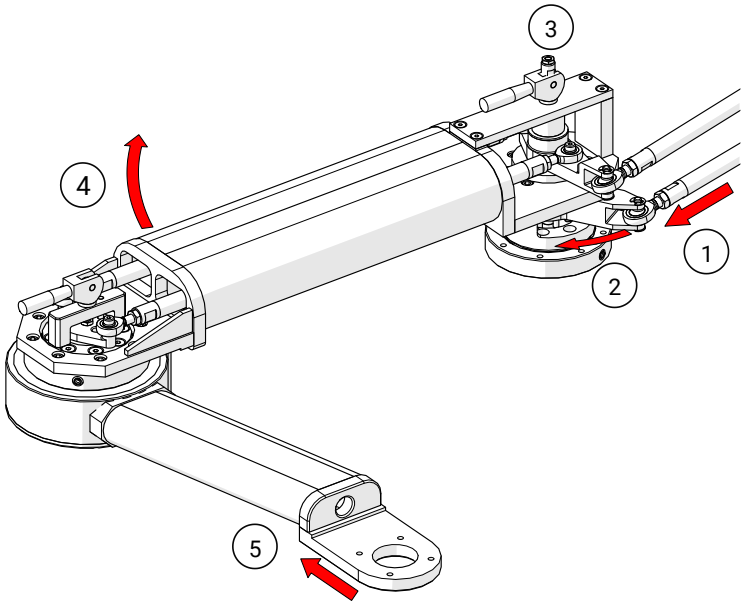


Fig. 6. Actuation of the manipulator's upper arm orientation angle: the linear displacement of the carriage on the shoulder spindle is transferred via a push-pull rod (1) to a lever that rotates (2) around the shoulder joint (3), resulting in rotation of the upper arm (4) and translation of the end point (5) while the lower arm orientation angle remains the same.

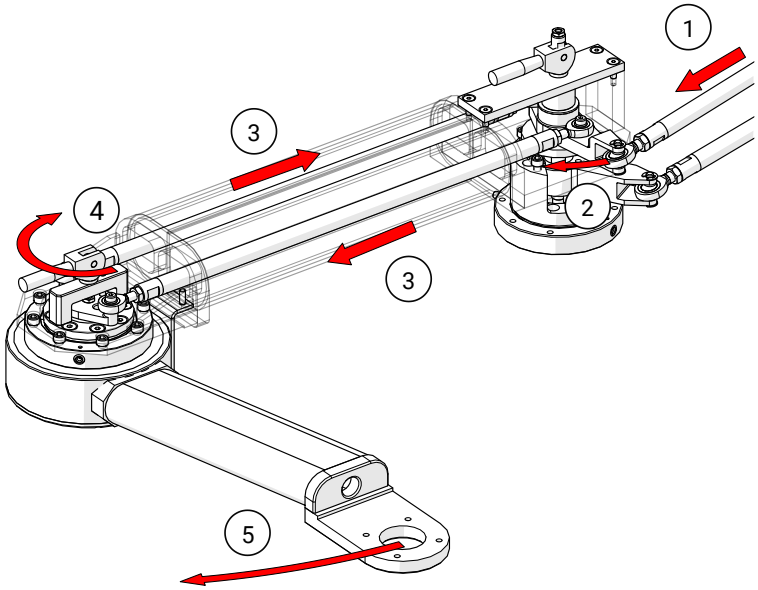


Fig. 7. Actuation of the manipulator's lower arm orientation angle: the linear displacement of the carriage on the elbow spindle is transferred via a push-pull rod (1) to a lever that rotates (2) around the shoulder joint, this lever is attached to rods in a parallelogram (3) that transfer the rotation of a lever that rotates around the elbow joint (4), resulting in rotation of the lower arm (5) and translation of the end point.

The manipulator was mechanically designed to have high stiffness. Actuation of the manipulator segments with push-pull rods prevents the need for servos within the manipulator. Direct spindle connections to the servos remove the need for high gearing, minimizing backlash. The servo's deliver up to 16.3 Nm (Automation & Ns, n.d.), corresponding to a theoretical maximum endpoint force of approximately 1,000 N. The maximal linear actuation input force applied by the carriage is calculated with the relationship between servo torque and spindle force in equation (1) with the torque T, force F, and spindle lead l. One full rotation of the servo results in 20 mm of linear carriage translation, the corresponding servo lead is 0.02 m. Assuming no friction and 100% efficiency, the maximal carriage force is 5,122.5 N.

$$T = F \cdot \frac{l}{2\pi} \tag{1}$$

3.4. Device Control

The NACT-3D can operate in both admittance and impedance control modes to create immersive virtual haptic environments. The system behaves as though physical objects (e.g., springs or walls) are present by generating forces that vary with movement. For example, a virtual spring is emulated by applying a directional force that increases with distance.

During admittance control, the controller adjusts the system's admittance to match that of the virtual environment. During unimpeded reaching, the manipulator should appear

'invisible' to the participant (Keemink et al., 2018). This requires its admittance to exceed that of the human limb. When a participant interacts with virtual objects, the perceived forces and compliance are generated by the controller, they do not originate from physical mechanics.

In impedance control, the controller operates using a velocity control loop. The haptic renderer processes the desired endpoint dynamics based on virtual object models and transforms these into reference velocity signals. These are passed through an internal kinematic model and PI controllers that compute joint velocity commands. Servo drives receive these commands and use feedback from encoder signals (servo position and velocity) to generate the appropriate motor torques by controlling the servo's current. This layered control approach ensures that the end effector behaviour closely matches the virtual environment. However, due to the need to integrate accelerations into velocities, any noise or drift in sensor readings can accumulate over time. This integration step introduces errors at higher frequencies. Robust tuning of the PI controllers can mitigate these errors to some extent. Still, at higher bandwidths, the actuator power and servo response become performance bottlenecks.

In addition to the existing velocity-based impedance control, computed torque control is promising for future implementation. This method combines feedback and feedforward control by directly calculating the required joint torques using dynamic models of the manipulator. While not currently implemented, it offers potential performance benefits, particularly in high-speed applications, by reducing lag and improving tracking accuracy.

A simplified control diagram is shown in Fig. 8, the full system control diagram is provided in APPENDIX A.

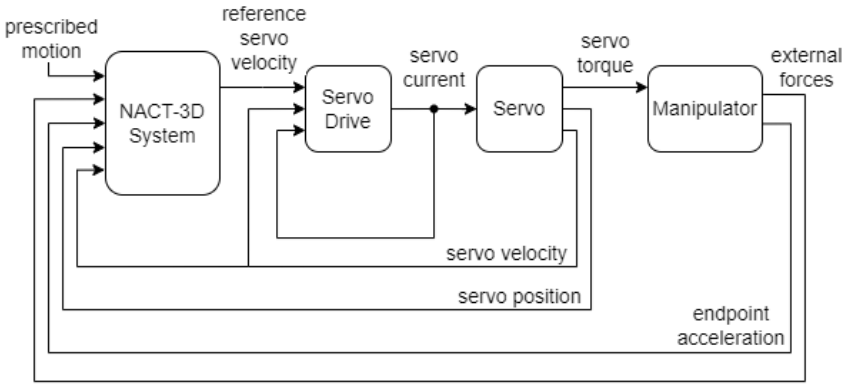


Fig. 8. Simplified control diagram for the NACT-3D; a prescribed motion is translated into prescribed servo velocities using feedback from the measured servo behaviour and manipulator forces and accelerations.

3.5. Safety

The NACT-3D integrates both hardware and software safety mechanisms to protect users (Fig. 9):

- Hardware: Emergency stop buttons (accessible for operator as well as participant), safety interlocks on plexiglass shields, and physical limit switches at both ends of each spindle.
- Software: Real-time monitoring the manipulator's position and restricting movement based on kinematic constraints and range-of-motion limits that are specific to the participant. Additionally, the admittance is set to zero at the workspace boundaries to prevent overextension or accidental collisions.

Together, these measures ensure user safety during all operating conditions and support use with impaired or vulnerable participants.

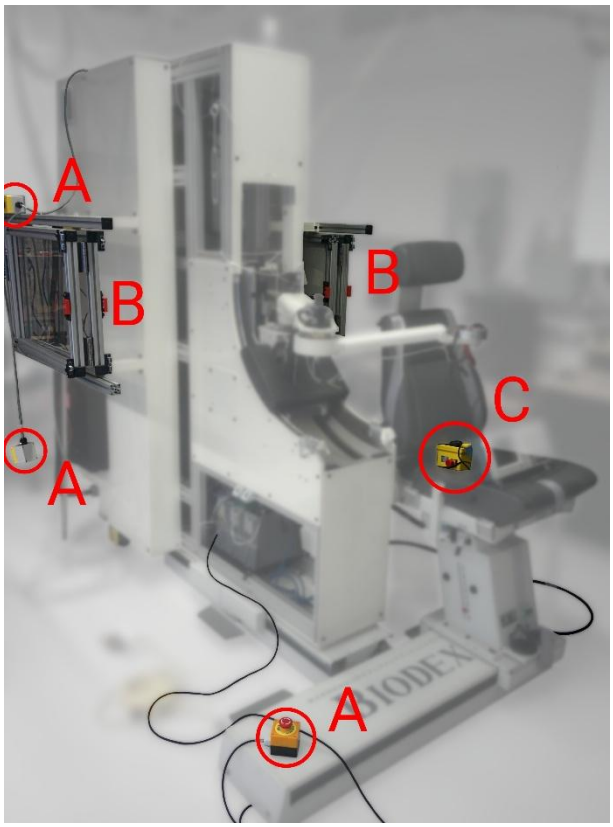


Fig. 9. NACT-3D's safety systems: (A) multiple emergency stop buttons that can be placed anywhere, (B) interlocks on hinged plexiglass sheets, and (C) a dedicated emergency stop for the operator.

4 REQUIREMENTS

This chapter defines performance and operational requirements necessary for the NACT-3D to effectively create controlled haptic environments, withstand maximum human-generated forces, and accurately replicate desired dynamic properties to apply perturbations during motion (Plaisier et al., 2023). These requirements must be met for the NACT-3D to precisely evaluate and characterize human upper limb movement and proprioceptive reflexes in a clinical research context.

4.1. Control and Haptic Capabilities

To simulate biomechanically plausible interactions, the NACT-3D must create responsive and realistic haptic environments. To achieve this, a high haptic transparency and force feedback fidelity are necessary.

To enable precise simulation of various haptic environments for a human arm, the NACT-3D must use an admittance control strategy. This allows for the modulation of the interaction forces experienced by the participant. Essential capabilities include generating virtual objects such as masses, dampers, springs, and force fields. The admittance should also be adjustable and the transitions between different admittance levels should be seamless. This enables participants to experience sensations ranging from strong damping to weightlessness.

High haptic transparency is essential to ensure the participant perceives the system as ‘invisible’. Therefore, the manipulator must respond swiftly and accurately to applied forces, ensuring the participant’s experience is dominated by the virtual environment rather than mechanical resistance from the manipulator. The system must: measure forces, generate reference trajectories, and execute accurate actuation.

While the controller can typically compensate for damping and stiffness, the physical mass of the manipulator is more challenging to overcome. The extent to which this mass can be effectively compensated serves as a key indicator of haptic capabilities. Therefore, reducing the manipulator's inertia and improving mass compensation through control are both essential to enhance haptic transparency.

Effectively, the controller must compensate for manipulator dynamics to ensure the perceived interactions match the virtually defined haptic environment.

4.2. Work Area

The NACT-3D's manipulator must have a sufficiently large work area to ensure the full range of human arm motion can be accessed. The human-device interface occurs at the forearm, and the participant's torso is fixed by an ergonomic chair. Based on human anatomy, a 3D motion envelope for the forearm relative to the shoulder is defined in device coordinates.

DINED data defines human reach envelopes for different populations. The maximum reach envelope corresponds to a frontal distance of 130.8 cm for the 95th percentile of adults aged 20–30 years with a stature of at least 1.75 m (Dirken et al., 2018). Although the NACT-3D is intended for clinical populations that likely have a reduced range of motion, it must also accommodate healthy participants.

The NACT-3D's manipulator dimensions were used to calculate its current workspace. Both the human reach envelope and NACT-3D work area were projected onto the global horizontal plane as shown in Fig. 10. The NACT-3D's work area projection was taken at $z = 0$ where push-pull rods are horizontal. Most of the human envelope fits within the NACT-3D's workspace, except for extreme shoulder abduction, which is not needed for typical experimental protocols.

Near the workspace limits, the manipulator operates close to full extension. This increases joint torques and mass moment of inertia, reducing dynamic responsiveness. High manipulator stiffness is needed in these zones to maintain control performance and avoid excessive torque demands. This high manipulator stiffness must be achieved while minimizing the weight to ensure the mass moment of inertia is sufficiently low to satisfy the operational bandwidth requirements.

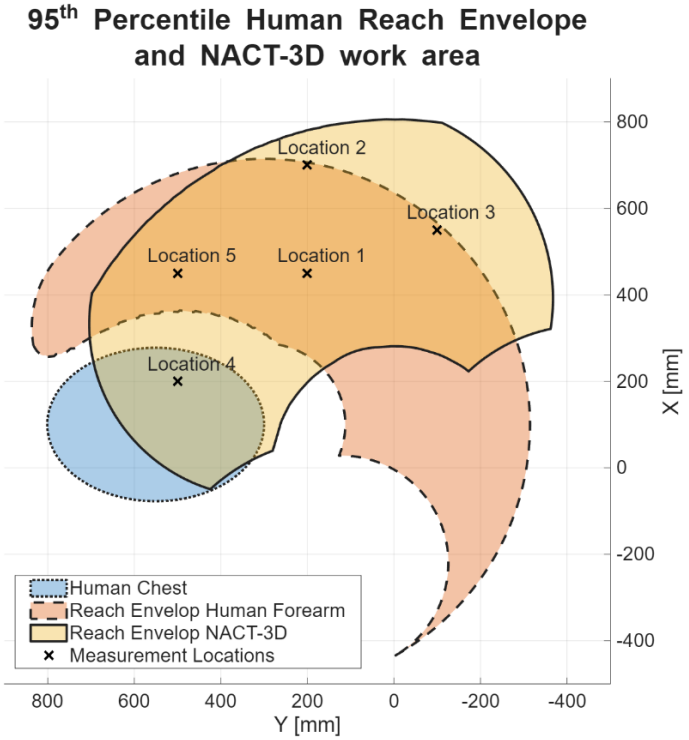


Fig. 10. The 2D human reach envelope is shown together with the work area of the NACT-3D's manipulator, taken at the z = 0 configuration.

As shown in illustrated in Manipulator Design, the NACT-3D's manipulator can extend beyond the human's reach envelope. Chapter 3.5 - Safety outlines what precautions were taken to prevent the manipulator from striking the participant or moving beyond the participant's reach and over-stretching their arm.

4.3. Maximum Forces

To ensure safety and durability, the NACT-3D must resist the full range of forces that a human arm may exert. Peak torques were measured in 2015 at the Northwestern University laboratory using an isometric setup. These values are listed in TABLE I. and are a reference for the structural stiffness and actuator sizing requirements. An overview of the naming each human-generated torque that was measured is provided in Fig. 11.

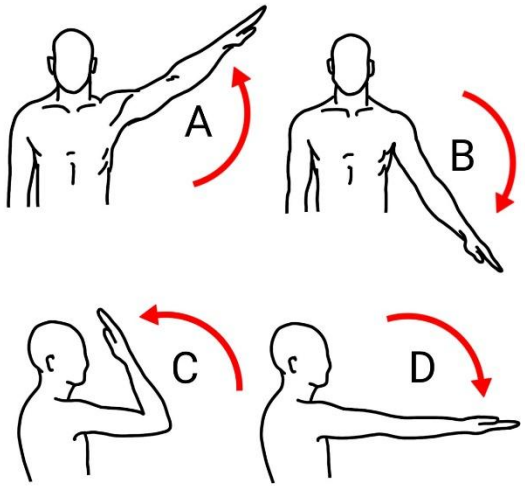


Fig. 11. Human arm torques: (A) shoulder abduction, (B) shoulder adduction, (C) elbow flexion, and (D) elbow extension.

TABLE I. MAXIMUM HUMAN UPPER ARM TORQUES AND CONFIGURATION

Measurement	Value
Shoulder Abduction	75 ° / 90 °
Elbow Flexion	90 °
Horizontal Shoulder Adduction	40 °
Shoulder Abduction Max Torque	110 Nm
Shoulder Adduction Max Torque	111.6 Nm
Elbow Flexion Max Torque	108.6 Nm
Elbow Extension Max Torque	73.7 Nm

Using the data from TABLE I. and the known positions of the human arm, corresponding manipulator joint torques were calculated based on the current dimensions of the NACT-3D's manipulator. The calculated manipulator joint torques are shown in 0and TABLE II. . An overview of the naming of the torques on the manipulator is provided in Fig. 12.

TABLE II. RESULTING MOMENTS AT THE MANIPULATOR'S ELBOW JOINT FROM THE DIFFERENT HUMAN JOINT TORQUES.

Applied Human Force	Twisting Torque [Nm]	Vertical Bending Moment [Nm]	Horizontal Bending Moment [Nm]
Shoulder Abduction	53	31	-16
Shoulder Adduction	-72	-42	15
Elbow Flexion	-21	-12	-109
Elbow Extension	15	9	74

TABLE III. RESULTING MOMENTS AT THE MANIPULATOR'S SHOULDER JOINT FROM DIFFERENT HUMAN JOINT TORQUES.

Applied Human Force	Twisting Torque [Nm]	Vertical Bending Moment [Nm]	Horizontal Bending Moment [Nm]
Shoulder Abduction	-27	-146	-37
Shoulder Adduction	26	180	35
Elbow Flexion	110	67	-223
Elbow Extension	-80	-49	158

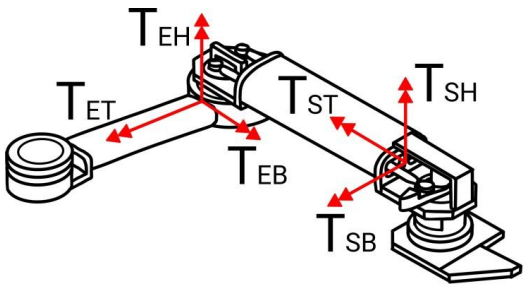


Fig. 12. Manipulator segment torques: (T_{EH}) bending torque in the horizontal plane of the lower arm at the elbow, (T_{EB}) vertical bending torque of the lower arm at the elbow, (T_{ET}) twisting torque around the length of the lower arm at the elbow, (T_{SH}) bending torque in the horizontal plane of the upper arm at the shoulder, (T_{SB}) vertical bending torque of the upper arm at the shoulder, and (T_{ST}) twisting torque around the length of the upper arm at the shoulder.

These values in 0and TABLE II. show that the manipulator shoulder must withstand up to 110 Nm twisting torque, 180 Nm vertical bending moment, and 223 Nm horizontal bending moment; and the elbow must tolerate 53 Nm torque, 31 Nm vertical bending moment, and 93 Nm horizontal bending moment. Additionally, dynamic motion tests done during baseball pitching (Leenen et al., 2020) demonstrate that the peak elbow flexion moments during movement (54.5 Nm) are below the static values observed above, confirming these estimates are conservative and sufficient.

4.4. Dynamic Properties

To evaluate proprioceptive reflex modulation, the NACT-3D must operate beyond the human neuromuscular bandwidth. Reflex responses at the shoulder occur around 2 Hz (Happee et al., 2008), and voluntary movements at the elbow range from 2-6 Hz (De Vlugt, 2004; Schouten et al., 2006). Any movement above 6 Hz are considered open loop from the human perspective as that is feedforward motion. Satisfying the bandwidth requirements is essential to ensure research validity and accurate measurement of human reflexes.

To identify reflex dynamics and conduct valid system identification, the manipulator must apply inputs and collect data at frequencies of at least 20 Hz (ten times the reflex

bandwidth). The system's mechanical and control bandwidth must exceed this value to ensure valid results. This bandwidth will also allow the NACT-3D to maintain control during feedforward movement of the participant as it is above 6 Hz.

Maintaining trajectory control during high-frequency motion requires minimizing deflections and eliminating mechanical play. Low manipulator inertia improves the ability of the controller to track desired trajectories. The manipulator's mass distribution and geometry must balance stiffness and inertia to support high-frequency responsiveness without introducing error.

Fine-tuned control parameters and suitable control architectures (e.g., computed torque control) may increase the NACT-3D's performance; however, computed torque control is not implemented in the current NACT-3D configuration.

5 METHODS

This chapter outlines the approach used to evaluate whether the NACT-3D meets operational requirements. A combination of static analyses, experiments, and simulations were used to assess mechanical performance, control effectiveness, and dynamic response bandwidth. The overall aim is to determine if the current configuration can reliably perturb the human arm and generate virtual haptic environments with sufficient accuracy and responsiveness for the intended purpose.

5.1. Evaluation Strategy

The evaluation strategy of the NACT-3D consists of two main approaches: physical experiments on the NACT-3D in its current configuration, and virtual simulations using a representative dynamic model. These approaches address the fundamental requirements: mechanical stiffness, dynamic responsiveness, and control accuracy. The methods are organized into the following categories:

- Static force and deflection analysis.
- Experimental perturbations of the system with and without the manipulator.
- Simulations using SPACAR to model the manipulator's dynamic properties and predict bandwidth.

5.2. Static Force and Deflection Analysis

Before evaluating the dynamic performance of the NACT-3D through experiments and simulations, it was necessary to confirm whether its mechanical structure could meet stiffness and deflection requirements under expected static loading. This was addressed using a static analysis tool.

A MATLAB-based tool was developed to evaluate whether selected cross-sectional geometries meet requirements. The tool used inputs including manipulator segment dimensions and material properties. The analysis aimed to determine if deflections, bending moments, and joint torques remained within allowable limits for the intended loading scenarios, and if the mechanical configuration would enable responsive and stable actuation without compromising safety or stiffness.

The tool was implemented with a graphical user interface (GUI), allowing users to input segment lengths, cross-sectional dimensions, applied force vectors, and material properties (e.g., Yong's modulus and density). Calculations were performed using beam theory, and outputs included: joint torque distributions, maximum endpoint deflections, and 3D visualizations of deflections for varying manipulator configurations and poses. The tool facilitates pose-specific analysis, enabling interactive evaluation of mechanical feasibility across the workspace (Fig. 13). Full implementation details are provided in Appendix B. , with the GUI layout in Appendix D. .

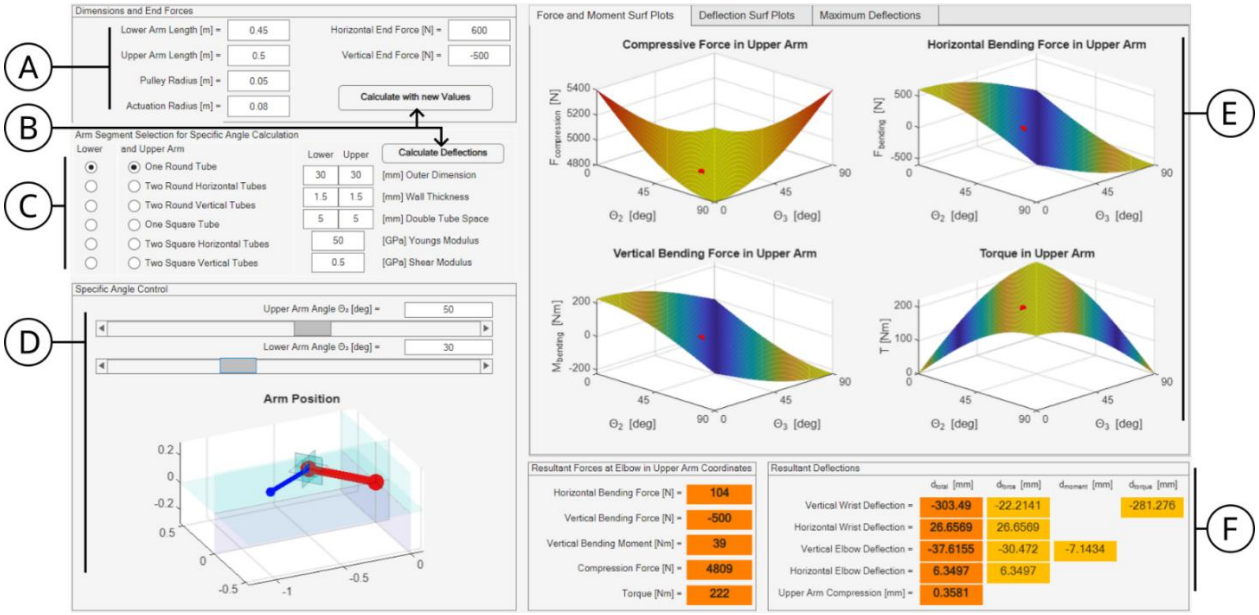


Fig. 13. GUI of the 'NACTArmStaticCalculator' tool; (A) manipulator dimensions and load inputs, (B) control to calculate resulting forces and deflections, (C) manipulator segment configuration and dimension inputs, (D) manipulator position configuration input sliders with visualization, (E) output plots, and (F) resultant forces and deflections for the defined inputs.

Three material types were compared based on their specific modulus (E/ρ) and specific strength (σ/ρ): aluminium, AISI 1045 steel, and unidirectional carbon epoxy (TABLE IV.). These metrics indicate how efficiently a material resists deformation and supports load

per unit mass, which are important for dynamic applications such as robotic manipulators. The dynamic performance benefits from reduced inertia and increased stiffness due to higher accelerations and smaller deflections directly improving control responsiveness.

TABLE IV. MATERIAL PROPERTIES FOR ALUMINIUM, STEEL, AND CARBON FIBRE (ENGINEERING TOOLBOX, 2008).

Material	Specific Modulus E / ρ	Specific Strength σ / ρ
Aluminium	27	0.17
Steel, AISI 1045	26.3	0.073
Unidirectional Carbon Epoxy (61%)	89.3	1.08

Carbon fiber was selected due to its favourable strength-to-weight and stiffness-to-weight ratios. However, due to its heterogeneous and anisotropic nature, its properties vary with fiber orientation, matrix content, and manufacturing quality. To accommodate this, conservative mechanical property estimates were used to ensure design robustness.

5.3. Experimental Setup

While the static analysis verified the mechanical feasibility of the manipulator, dynamic testing was performed to assess the system's performance under realistic operational conditions. The following section details the experimental procedures used to evaluate the system's dynamic response and control performance.

5.3.1. Manipulator Configuration

The experiments were designed to isolate key components of the NACT-3D's behaviour: actuation performance and manipulator-induced limitations. Trials were preformed using both full and reduced hardware configurations (with and without the manipulator attached) and measured the system's ability to follow input trajectories across a range of frequencies and spatial locations. Without the manipulator attached, the NACT-3D's actuation operated in an unloaded configuration. The resulting performance, as computed with the system identification approach, highlights the capabilities of the actuation system.

Although the NACT-3D supports actuation in three dimensions, the experimental signals were restricted to two dimensions (planar perturbations in x and y). This restriction was made to simplify the system identification process and align with anticipated clinical applications of the NACT-3D, which are likely to focus on planar upper-limb movement. This design choice balances analytical tractability with clinical relevance.

To gain insight into the manipulator's performance through the workspace, trials were conducted at five locations within the NACT-3D's workspace. The locations were selected for challenging manipulator poses that roughly coincide with the edges of the expected workspace. The coordinates of these locations are listed in TABLE V. and shown in Fig. 10. At each location, perturbations were applied separately along the x- and y-dimensions. To ensure statistical reliability, each perturbation was repeated three times for each configuration of NACT-3D (with or without the manipulator), workspace location, and perturbation direction.

TABLE V. EXPERIMENTAL LOCATIONS AND CORRESPONDING MANIPULATOR END POINT COORDINATES

Experimental Location	**Manipulator End Point Coordinates**	
	x [mm]	***y [mm]***
1	450	200
2	700	200
3	550	-100
4	200	500
5	450	500

The manipulator perturbations were executed while the NACT-3D manipulator was in the 'right-handed' configuration. It was assumed that the system's behavior is identical in both left/right-handed configurations as they are mirror images of one another.

5.3.2. Experimental Input Signals

With the manipulator test configurations and workspace locations established, the input signals were defined to perturb the system in a controlled and measurable way. The NACT-3D was perturbed with two types of signals to perform qualitative and quantitative analyses:

- Ramp signals were employed to visualize system response and latency in the time domain, revealing system characteristics such as time delay, overshoot, and damping.

- Multi-sine signals were used to estimate the frequency response functions (FRFs) and assess the system's bandwidth. They were comprised of sine waves at discrete frequencies from 1 Hz to 128 Hz[1]. Multi-sine signals are suitable for system identification and provide robust estimates of magnitude, phase changes over frequency, resonance, and attenuation.

[1] The frequencies included in the signal were 1, 2, 3, 4, 5, 6, 7, 8, 12, 16, 20, 24, 28, 32, 40, 48, 56, 64, 80, 96, 112, and 128 Hz.

5.3.3. Data Collection

The system's response to input signals was captured using a combination of internal and external sensing systems. Experimental motion data were captured at 1,000 Hz from two sources:

- Servo Encoders: The internal (absolute) encoder data from the servos (Automation & Ns, n.d.)was used to calculate end effector positions via an internal rigid body kinematic model.

- OptoTrak Motion Capture: The OptoTrak motion tracking system (Derzsi & Volcic, 2018) recorded marker positions of infrared markers attached to the NACT-3D. The system used 4 marker types to track motion and establish the local coordinates. The marker placement on the NACT-3D's manipulator is shown in Fig. 14. A detailed explanation of the marker placement is provided in Appendix E.
  .

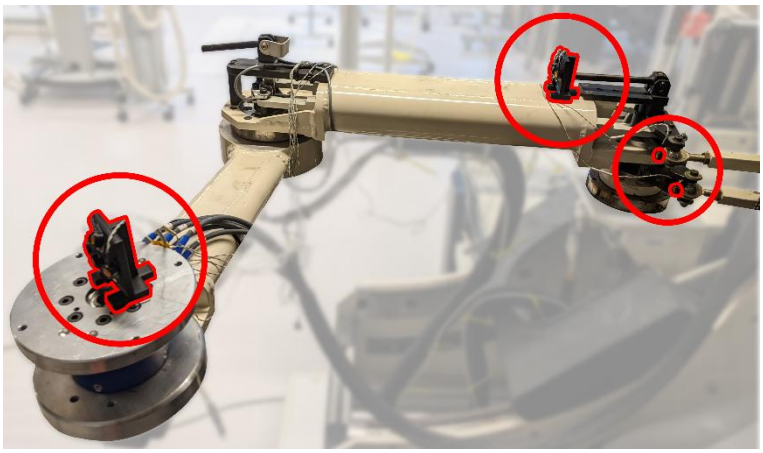


Fig. 14. Placement of infrared markers for the OptoTrak motion tracking system on the NACT-3D's manipulator, all markers are highlighted and circled in red.

All data collected during the experiments was categorized into three signals:

- Reference – The reference trajectory is the prescribed end effector trajectory that was provided as an input to the NACT-3D.

- Estimated – The estimated trajectory is the end point trajectory as projected from the servo's encoder data via the NACT-3D's internal kinematic rigid body model.

- Observed – The observed trajectory is the end point trajectory as recorded by the OptoTrak motion tracking system.

As described in 3.4. Device Control, the NACT-3D computes the necessary reference servo velocities required to achieve the reference trajectory using an internal kinematic rigid body model. The measured servo positions were used to calculate the estimated (projected) manipulator end point positions. Any kinematics and dynamics that aren't explicitly included in the servo drive's internal model were not included in the calculated

positions. An example of this is the play in the manipulator joints: this can cause discrepancies between the estimated position and the actual position of the manipulator. These discrepancies are shown by comparing the estimated positions with the observed positions as recorded with the OptoTrak motion tracking system.

Coordinate synchronization between these systems was necessary. The 3D marker data was aligned with the NACT-3D's coordinate system using markers placed at known reference points on the NACT-3D. Timestamp alignment was performed using extrapolated time sequences due to occasional gaps in the OptoTrak data. A manual correction method ensured alignment error remained smaller than three milliseconds. This error was deemed acceptable for the frequency response estimation.

During experiments without the manipulator, the OptoTrak markers were placed on the lever arms located at the 'shoulder-joint'. The experimental procedure was repeated, and the end point motion was projected based on the observed lever motion using a rigid body kinematic model.

Alignment of the OptoTrak's coordinate system with the NACT-3D's coordinate system was done by means of a plate of markers attached to the frame of the NACT-3D to use as a stationary reference during recording. Before the experimental trials, a digitizing probe with markers at the end of it was used to define the NACT-3D's manipulator. The system captured and recorded the locations of the probe when it was touched on the zero reference points which are the elbow and shoulder joint of the manipulator.

5.3.4. System Identification and Bandwidth Estimation

To interpret system behaviour, a system identification approach was applied. The NACT-3D exhibits characteristics of a multiple-input multiple-output (MIMO) system; two actuators simultaneously influence motion in both the x- and y-dimensions. However, due to limitations in the MATLAB system identification toolbox, the system was decomposed into multiple single-input single-output (SISO) subsystems. These approximations allowed for independent analysis of directional responses but inherently neglected inter-axis coupling.

To estimate the bandwidth, the frequency response function $H(f)$ must first be calculated, this is the system's transfer function in the frequency domain. This is done by dividing the cross spectral density $S_{uy}(f)$ between the input u and output y, by the auto spectral density $S_{uu}(f)$ of the input u as show in equation (2). The cross spectral density and auto spectral density are calculated withing MALTAB and averaged over all repetitions of each experimental trial.

$$H(f) = \frac{S_{uy}(f)}{S_{uu}(f)} \tag{2}$$

Ideally, to evaluate the performance of only the NACT-3D's manipulator, the input/outputs would be isolated. However, due to the design choices made, the inputs are in joint space, and the outputs are in cartesian space. Manipulator segment rotations

result in end point translations. Because of this, any single input will result in two outputs, and any single output requires two inputs.

If system identification were done from joint space to cartesian space, then the performance of the manipulator along with any artefacts resulting from the transfer function from joint- to cartesian-space would be evaluated. To evaluate the NACT-3D manipulator, all signals used for system identification were either recorded in the cartesian space or transformed to the cartesian space. Ideally, the NACT-3D's input joint angles would be transformed into cartesian space. However, these were not measured, but the servo positions were. As explained in 5.3.1 Manipulator Configuration, the system's input (servo positions) were transformed into cartesian space by means of the estimated end point coordinates.

System identification was performed by comparing the following trajectories: reference trajectory to observed trajectory, reference trajectory to estimated trajectory, and estimated trajectory to observed trajectory. Depending on the trajectories compared, the system identification results represent the performance of different components of the NACT-3D. An overview of the components represented by the results of each input-output combination of the system identification is given in TABLE VI. and TABLE VII.

TABLE VI. NACT-3D COMPONENTS EVALUATED WITH SYSTEM IDENTIFICATION RESULTS FOR EACH COMBINATION OF INPUT/OUTPUT FOR PERTURBATIONS WITH THE MANIPULATOR

System Identification Signals		**Components of NACT-3D Included in System Identification Results**			
Input	***Output***	***Controller***	***Servos***	***Actuation Linkage***	***Manipulator***
Reference	Observed	✓	✓	✓	✓
Reference	Estimated	✓	✓		
Estimated	Observed			✓	✓

TABLE VII. NACT-3D COMPONENTS EVALUATED WITH SYSTEM IDENTIFICATION RESULTS FOR EACH COMBINATION OF INPUT/OUTPUT FOR PERTURBATIONS WITHOUT THE MANIPULATOR

System Identification Signals		**Components of NACT-3D Included in System Identification Results**		
Input	***Output***	***Controller***	***Servos***	***Actuation Linkage***
Reference	Observed	✓	✓	✓
Reference	Estimated	✓	✓	
Estimated	Observed			✓

The frequency response function (FRF) was computed using averaged cross-spectral and auto-spectral densities. The -3 dB bandwidth threshold was used to define the operational bandwidth. The MATLAB code used to perform the system identification can be found in Appendix F. .

5.4. Simulation: SPACAR Model

The SPACAR-package (Aarts et al., 2011)can compute the dynamics of flexible multi-linked spatial mechanisms in MATLAB/SIMULINK using nonlinear finite element theory. Both motion and deformations can be simulated with one model. The ability to combine the analysis of the equations of motion with elastic deformations makes SPACAR a unique software and thus very suitable for simulating the performance of the NACT-3D manipulator. In most cases it would be sufficient to apply a rigid body model, but the bandwidth requirements of the NACT-3D manipulator require the dynamics of the manipulator segments to be included. An overview of the SPACAR model of the NACT-3D follows, a more detailed description of how SPACAR was used is provided in Appendix H. and Appendix I. .

5.4.1. Model Construction

In its simplest form, the NACT-3D manipulator can be modelled as two segments in two-dimensional space (Fig. 15). This is not an accurate representation of the system as the NACT-3D doesn't directly actuate the segment (joint) angles. Further expanding the virtual representation of the NACT-3D, actuation levers and push-pull rods are added (Fig. 16).

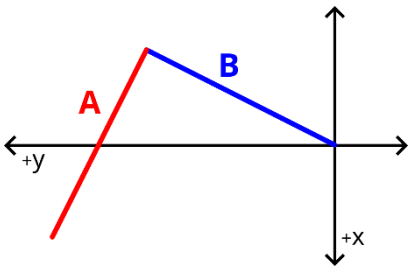


Fig. 15. Simplified 2D model of the NACT-3D's manipulator segments: (A) lower arm and (B) upper arm.

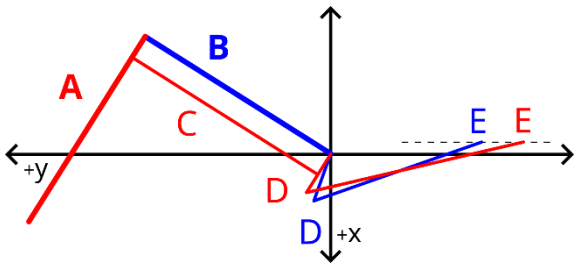


Fig. 16. Simplified 2D model of the NACT-3D's manipulator including the actuation linkage: (A) lower arm, (B) upper arm, (C) lower arm parallelogram, (D) levers for each manipulator segment, and (E) push-pull rods for each manipulator segment.

A two-dimensional model allows for planar analysis of the manipulator. However, the NACT-3D can be operated, and is loaded, in three-dimensional space. Additionally, the manipulator segments and actuation linkage are not all on the same plane. Although the experiments and simulations are two-dimensional, the vertical offset between the components also causes off-plane dynamics and stress. The virtual model of

manipulator was further expanded to better represent the NACT-3D in 3 dimensions (Fig. 17).

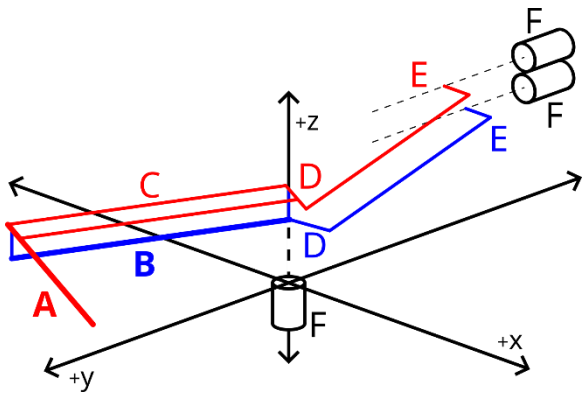


Fig. 17. Simplified 3D model of the NACT-3D's manipulator including the actuation linkage: (A) lower arm, (B) upper arm, (C) lower arm parallelogram, (D) levers for each manipulator segment, (E) push-pull rods on spindles for each manipulator segment, and (F) servos to actuate each spindle.

A complete virtual model of the NACT-3D manipulator was constructed using the SPACAR simulation toolbox as it supports both rigid and flexible multibody dynamics. To accurately represent the manipulator, the elements shown in Fig. 17 were further separated into smaller segments. The model was constructed using three different element types: beam, truss, and hinge. The element's deformation parameters are shown in TABLE VIII. with position nodes x and orientation nodes λ. The truss element was used to simulate the push-pull rods as well as the linear actuation inputs. By placing truss elements on the neutral axis of a spindle, spatially fixing one position node, and defining the other position node as an input, the single deformation node (tensile/compressive force) represents the carriage force as generated by the servo torque on the spindle. To capture structural compliance, elements in the manipulator were modelled as elastic beam elements with material and geometric properties corresponding to the manipulator components as modelled in SolidWorks. The approximation and extraction of component properties is outlined in Appendix G. SolidWorks NACT-3D Model Properties.

TABLE VIII. NODAL INFORMATION AND DEFORMATION PARAMETERS FOR SPACAR ELEMENTS USED TO CONSTRUCT NACT-3D MODEL.

Element	End Node p	End Node q	Generalized Deformation Modes
Spatial Beam	x^p, λ^p	x^q, λ^q	ε_1, ε_2, ε_3, ε_4, ε_5, ε_6
Spatial Truss	x^p	x^q	ε_1
Spatial Hinge	λ^p	λ^q	ε_1, ε_2, ε_3

The manipulator's kinematic and dynamic data were defined in a .dat file used as an input for the SPACAR simulation. The element and node data were defined, along with the initial node locations (in the 'right-handed' configuration) that were derived from the SolidWorks model. Boundary conditions defined in the .dat file replicate the device's mounting configuration and support constraints. The mass and inertial properties were also included to reflect the effects of the manipulator's load on the actuation system during operation. The full definition of all elements and nodes is summarized in Appendix H. . H.1. Kinematic Data and the boundary conditions are established in Appendix H. . H.2. Dynamic Data. A visualization of the entire SPACAR model is shown in Appendix I. .

5.4.2. Simulation Input Signals

Due to the limitations of SPACAR, the multi-sine signal used for the experiments could not be implemented as an input signal for the SPACAR model. Instead, the model was perturbed using chirp signals consisting of sinusoidal position inputs with linearly increasing frequency applied to the actuated joints. A chirp signal was chosen as it also spans a wide frequency range and similarly reveals frequency-dependent response characteristics of the modelled system. The chirp signal is a single sine wave that sweeps over multiple frequencies.

Like the experimental setup, the chirp signal perturbations were simulated at five locations within the NACT-3D's virtual workspace (TABLE V. and Fig. 10). At each location, perturbations were simulated separately along the x- and y-dimensions. The SPACAR simulations are run from at MATLAB script (Appendix J.) that automates the simulation and extracts the relevant data. The chirp signal is defined in a MATLAB file (Appendix H. - H.3.H.3.2 - MATLAB Signal) and is called from within the SPACAR input file (Appendix H. - H.3.H.3.1 - Trajctory and Nominal Inputs & Outputs).

5.4.3. Simulation Execution

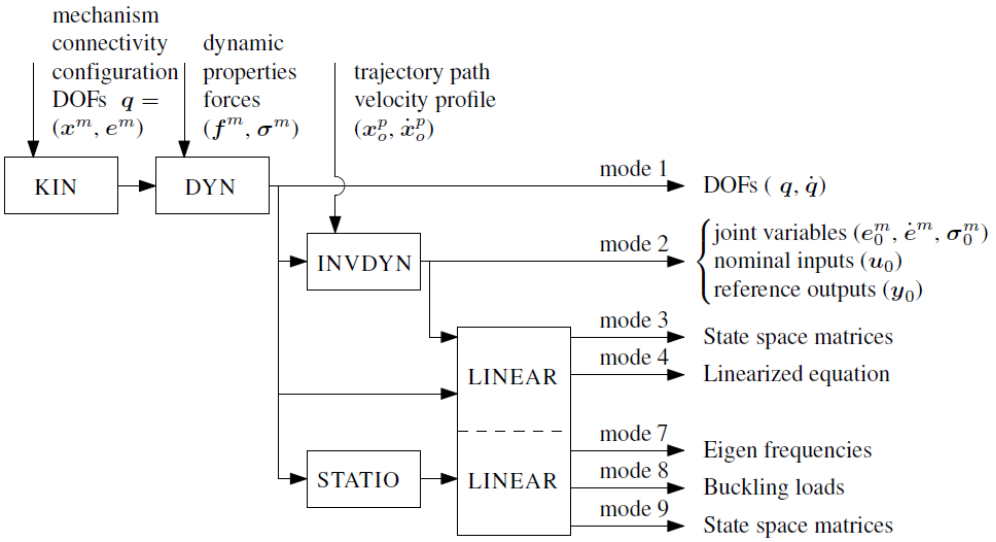


Fig. 18. Diagram portraying all the SPACAR modules (rectangles) and modes that the simulation can be run in (Aarts et al., 2011).

SPACAR requires a .dat text file as input and runs in the MATLAB environment, generating variables in the MATLAB workspace alongside SPACAR binary files containing the same information. The different modules and modes of SPACAR are illustrated in Fig. 18. There are five modules in SPACAR: kinematics, dynamics, inverse dynamics, stationary solutions, and (linearized) forward dynamics. These modules enable SPACAR to be run in 7 different modes to determine different solutions for the defined mechanism: generate equations of motion, numerical integration of forward dynamic analysis, inverse kinematics and dynamics, state space matrices, linearized equations as a function of the degrees of freedom, Eigen frequencies, buckling loads, and linearized state equations. The input file required consists of three blocks; the first block defines the kinematics of the mechanism, the second contains the dynamic properties, and the third contains the inverse dynamics and the setpoint generation (trajectory definition). The following three modes were used to simulate the NACT-3D's manipulator (listed in order):

- Mode 2 – Inverse kinematic model for transforming reference signal to actuator input. Mode 2 was employed for inverse dynamic simulations that computed the required actuator forces to follow a prescribed end-effector trajectory while accounting for link flexibility.

- Mode 1 – Linearized forward dynamic model using actuator velocity inputs. Mode 1 was used to simulate forward dynamics where joint torques were applied and resulting motion was observed.

- Mode 7 – Eigenfrequency simulation was performed in Mode 7 to estimate the system's natural frequencies and identify potential resonance points.

Theoretically the SPACAR model can be run with torque inputs, however this did not seem to be possible. The model was run using velocities as inputs and the actuator torques that would be required to achieve the end point velocities were computed by SPACAR and extracted. It was not possible to constrain maximum actuation (joint) torques in the model. As a result, to satisfy the input velocities, infinite accelerations and joint torques were possible, exceeding the servo's limitations by far. To account for this, all simulation results where the required spindles forces exceeded the maximum spindle force (calculated in 3.3 - Manipulator Design) was considered out of the simulated operating bandwidth of the manipulator.

5.4.4. System Identification and Bandwidth Estimation

Similar to the experimental system identification method, the system was decomposed into multiple SISO subsystems. In order to get insight into the inter-axis coupling, due to the system's MIMO nature, the system identification was performed using servo velocities as inputs and end point velocities as outputs (shown in Fig. 19).

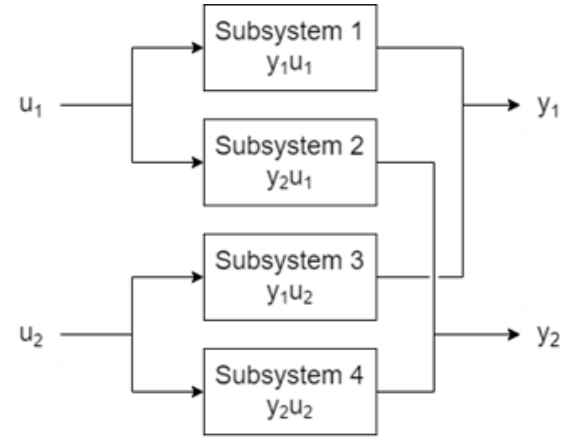


Fig. 19. Division of NACT-3D manipulator model into four subsystems. u1 and u2 represent the servo velocities. y1 and y2 represent the x and y coordinates of the manipulator's end point. The subsystems' in/out relations shown are as follows: (1) U1 to Y1, (2) U1 to Y2, (3) U2 to Y1, and (4) U2 to Y2.

Simulation outputs included time-domain profiles of joint angles, end-effector velocities, actuator torques, and internal link deformations. These responses were post-processed to extract frequency-domain information.

Bandwidth was assessed by analysing the simulated actuator torques as a function of excitation frequency. The cut-off frequency was defined as the point where the required actuator torque exceeded the rated continuous torque of the servo motors. This provided a conservative upper estimate of system performance, neglecting effects such as thermal limits or nonlinear saturation.

To support experimental validation, equivalent perturbation profiles were extracted from the chirp responses, enabling a one-to-one comparison between modelled and observed endpoint motion under identical loading conditions.

To estimate the system's frequency response function (FRF), the cross spectral density between input and output signals was divided by the auto spectral density of the input signal as shown in equation (2). As the SPACAR model is a MIMO system, the FRF was calculated for all inputs and outputs, equation (3). All spectral densities were averaged between the repetitions of the experiments for location and perturbations. Additionally, the spectral power of each subsystem $P_{i,j}$ was calculated, equation (5), by dividing the cross spectral density by the sum of the cross spectral densities of all subsystems, equation (4). The spectral power indicates the fraction of the total spectral density that is affected by each subsystem. Simply put, if all but one subsystem has a spectral power near zero, then the FRF of the entire subsystem is mostly determined by a single subsystem.

$$H_{i,j} = \frac{S_{u_i y_j}}{S_{u_i u_i}} \quad where \quad i,j \in \{1,2\} \tag{3}$$

$$\sum S_{uy} = \sum_{i=1}^{2}\sum_{j=1}^{2} S_{u_i y_j} \tag{4}$$

$$P_{i,j} = \frac{S_{u_i y_j}}{\sum S_{uy}} \quad where \quad i,j \in \{1,2\} \tag{5}$$

The bandwidth $B_{i,j}$ of each subsystem was estimated by plotting the FRF's magnitude and phase in bode plots, using only the perturbed frequencies. The point at which the magnitude crossed the -3dB threshold was interpolated and defined as the bandwidth of each identified subsystem. The entire system's bandwidth is estimated by taking a weighted average of the subsystem's bandwidths. This is done by taking the sum of each subsystem's bandwidth multiplied by the respective spectral power as shown in equation (6).

$$B = \sum_{i=1}^{2} \sum_{j=1}^{2} P_{i,j} \cdot B_{i,j} \tag{6}$$

6 RESULTS

This chapter provides the results of the experiments as outlined in section 5. Methods. The data is shown along with how it was processed and the subsequent conclusions. In section 4.1, the results of the static force calculations of the NACT-3D manipulator for potential designs are shown as manipulator deflections. In section 4.2, the signal alignment that was applied is illustrated with raw data results of perturbations. In sections 4.4 to 4.6, NACT-3D experimental results with and without the manipulator as well as SPACAR simulation results are shown. The raw data is shown for some experiments, and the results are quantified as bandwidths for measured and simulated systems. All bandwidth results are obtained from calculating the frequency response function (FRF) between the reference signal and the observed and estimated signal.

A brief overview of the NACT-3D manipulator experiments and simulations follows. In total five different experiments are run for the NACT-3D:

- Force Calculations - The deflections of the NACT-3D's manipulator due to a static load are calculated for various form factors and materials.

- Manipulator Experiments - Multi-sine perturbation inputs are applied to the NACT-3D with the manipulator attached. The inputs and outputs for system identification are reference end point trajectory and observed end point trajectory respectively.

- Actuation Experiments - Multi-sine perturbation inputs are applied to the NACT-3D without the manipulator attached, thereby measuring the performance of the NACT-3D's actuation linkage and controller. The inputs and outputs for system identification are reference end point trajectory and projected end point trajectory respectively.

- SPACAR Simulations - A chirp signal is simulated with the NACT-3D SPACAR model to establish the bandwidth of the virtual model and evaluate the validity of the system identification results of the virtual model.

Another detail that is essential for the context of the experiments and simulations with the manipulator is that they are all done in an unloaded state. This means that there is no additional weight or resistance added to the end point of the manipulator (i.e. the human arm that interfaces with the manipulator during standard operation).

6.1. Force Calculation

Carbon fibre was identified as the most suitable candidate due to its superior stiffness-to-weight and strength-to-weight ratios, despite known variability in composite properties. The static tool results indicated that a carbon fibre manipulator would achieve similar deflections to the current aluminium configuration while significantly reducing inertia.

TABLE IX. NODAL INFORMATION AND DEFORMATION PARAMETERS FOR SPACAR ELEMENTS USED TO CONSTRUCT NACT-3D MODEL.

Manipulator Segment Properties		**Manipulator End Point Deflection [mm]**		**Manipulator Elbow Deflection [mm]**		**Manipulator Upper Arm Compression [mm]**
Cross Section Shape	***Amount and Configuration***	***Vertical***	***Horizontal***	***Vertical***	***Horizontal***	
Cylinder	1	307.1	26.7	71.6	36.6	0.4
	2 Horizontal	72.5	3.3	35.8	4.6	0.2
	2 Vertical	64.2	13.3	8.9	18.3	0.2
Square	1	189.9	15.7	42.2	21.5	0.3
	2 Horizontal	49.2	2.4	21.1	3.3	0.2
	2 Vertical	44.6	7.9	6.5	10.8	0.2

6.2. Ramp Signal Alignment and Raw Data

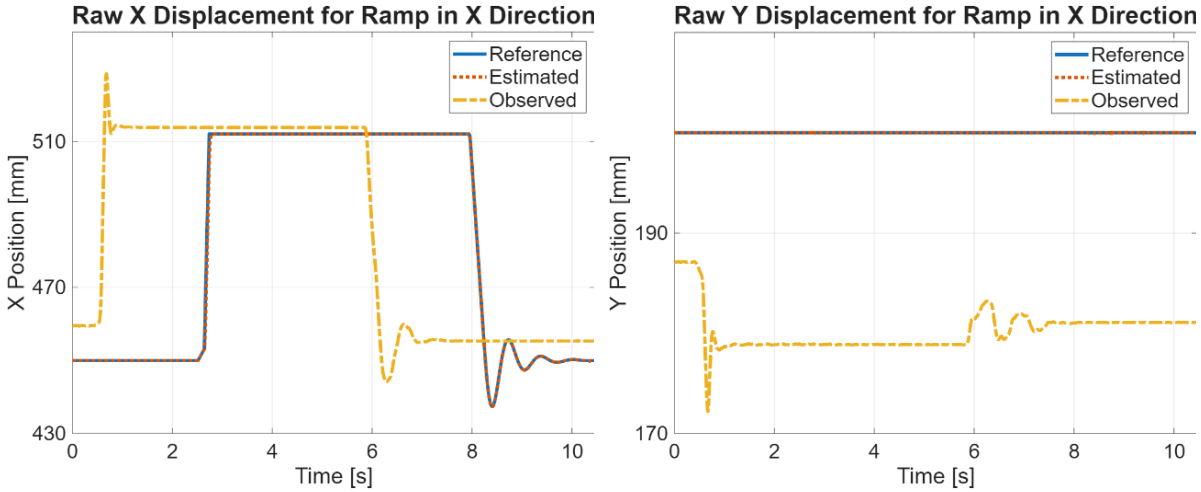


Fig. 20. Raw end point position data for a ramp perturbation with the manipulator in the positive x direction at location 1.

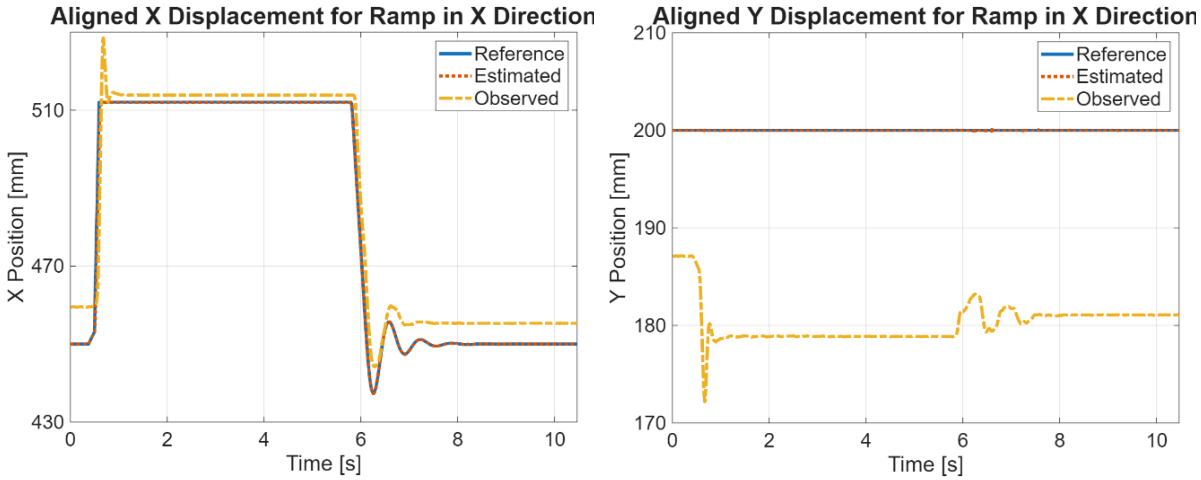


Fig. 21. Aligned end point position data for a ramp perturbation with the manipulator in the positive x direction at location 1.

Raw ramp signal before alignment shown in Fig. 20 shows significant deviation between reference and observed trajectories. This confirms the need for time synchronization between the NACT-3D system and OptoTrak data sources. After alignment, the NACT-3D's signal tracking is clearer as shown in Fig. 21. However, these results show position offset, response delay, signal overshoot, and cross-axis motion (displacement in the orthogonal direction to the perturbation). This indicates that the manipulator's mechanical properties significantly limited trajectory tracking, even at low frequencies.

6.3. Experimental Results

Raw data from the multi-sine perturbations contains three multi-sine perturbation trials (Fig. 22). The data was aligned, and the separate multi-sine perturbation trials were isolated (Fig. 23). The NACT-3D's frequency response function (FRF) was estimated and averaged for each location. The system's bandwidth is determined by the frequency at which the FRF's magnitude drops below the -3 dB threshold.

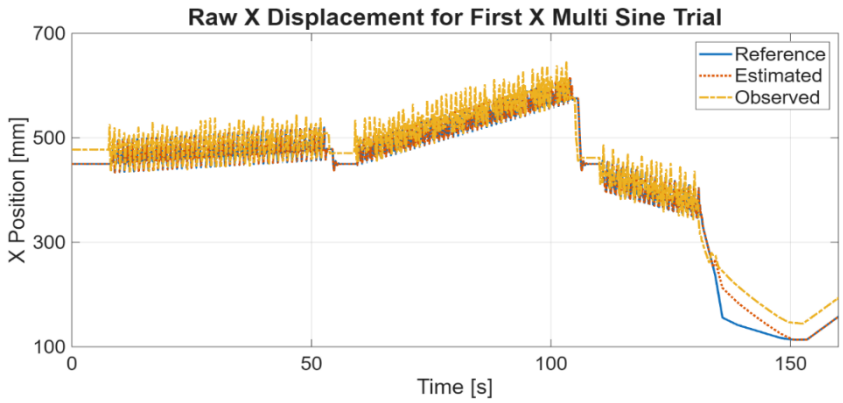


Fig. 22. Raw end point x position data for the x-direction multi sine trials with manipulator at location 1.

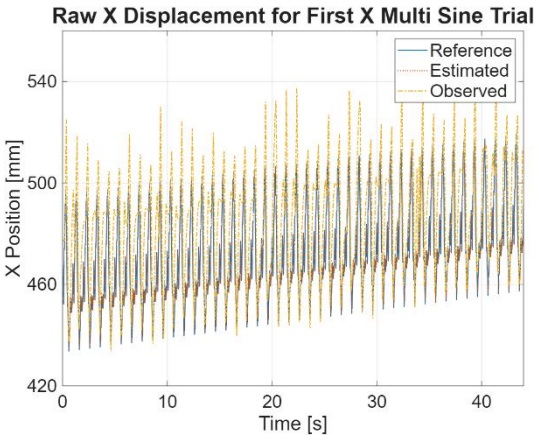


Fig. 23. Raw end point x position data for a single x-direction multi sine trial with manipulator at location 1.

6.3.1. Manipulator Experiments

The experimental data of the NACT-3D with the manipulator collected was averaged and used to calculate the frequency response function (FRF) for all perturbations at each location. The FRF is shown in bode plots for the frequencies that are included in the multi-sine input and shows the system's performance at different frequencies. Where the magnitude drops below the -3 dB threshold indicates the system's bandwidth. The NACT-3D perturbations were carried out for all five locations in both the 'x' direction and the 'y' direction. All plots for the experimental data will show three sets of data: reference, estimated, and observed.

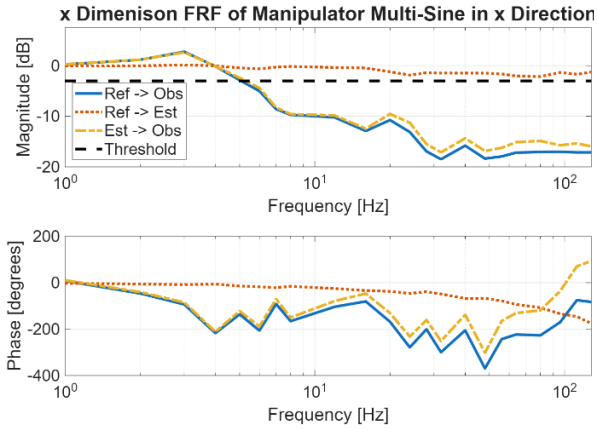


Fig. 24. Frequency response function in the x dimension for the x-direction multi-sine perturbation of the NACT-3D with the manipulator at location 1.

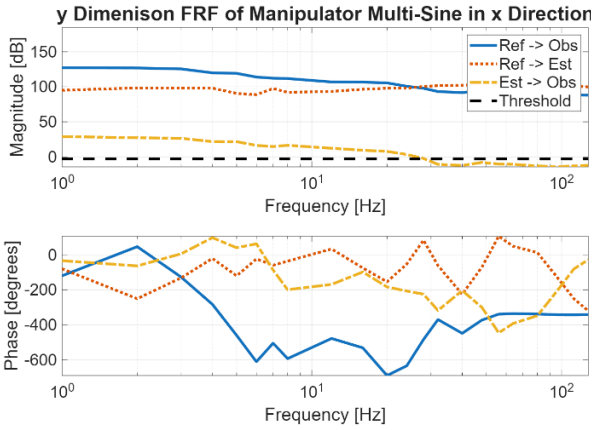


Fig. 25. Frequency response function in the y dimension for the x-direction multi-sine perturbation of the NACT-3D with the manipulator at location 1.

The FRF for the 'x' perturbation location '1' with the manipulator (Fig. 24) shows a bandwidth of 5.07 Hz for the reference to observed FRF. The FRF in the 'y' dimension for the same experiment (Fig. 25) a magnitude greater than 1 for the system. No movement is expected in the 'y' dimension during this perturbation. However, the FRF indicates that an output is generated without input, thereby showing the severity of the cross-axis motion. These results are confirmed when looking at the end point location data for the perturbation (Fig. 26 and Fig. 27). Additionally, the location data shows that the observed manipulator doesn't follow the reference trajectory very accurately whereas the controller estimates that the manipulator is tracking the reference trajectory very accurately. The large movements are somewhat recognizable, but even so there is quite some delay.

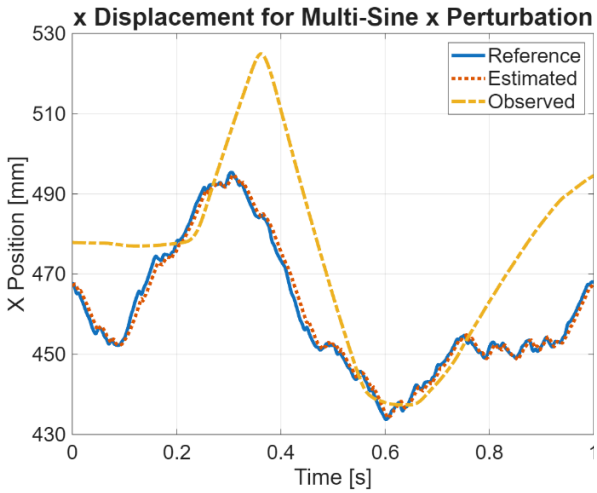


Fig. 26. Manipulator end point x-position data for the x-direction multi-sine perturbation of the NACT-3D at location 1.

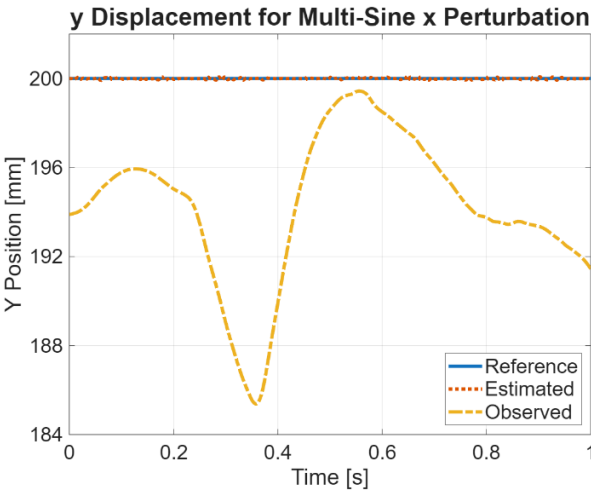


Fig. 27. Manipulator end point y-position data for the x-direction multi-sine perturbation of the NACT-3D at location 1.

The bandwidth for the reference to estimated FRF's bandwidth is greater than 128Hz as the magnitude does not drop below the -3dB threshold (Fig. 24). This indicates that the performance of the NACT-3D's actuation linkage, servos, and controller far exceed the performance of the manipulator. Furthermore, a slight drop in magnitude shows that there is a slight drop in performance in the higher frequencies.

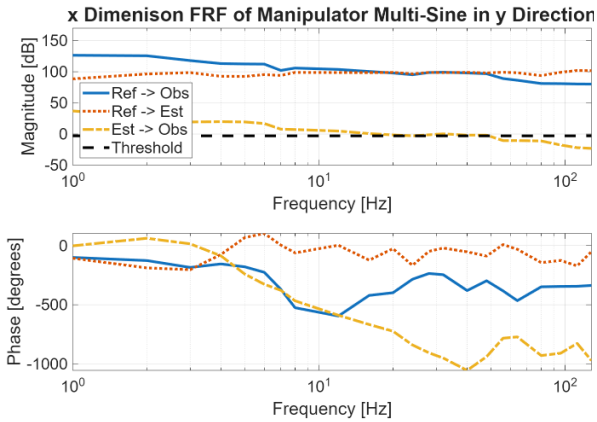


Fig. 28. Frequency response function in the y dimension for the y-direction multi-sine perturbation of the NACT-3D with the manipulator at location 1.

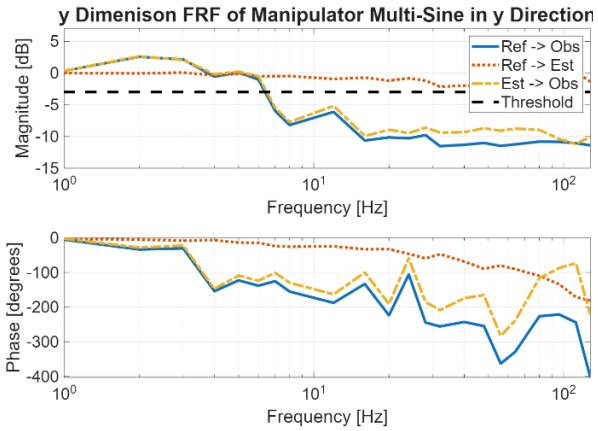


Fig. 29. Frequency response function in the x dimension for the y-direction multi-sine perturbation of the NACT-3D with the manipulator at location 1.

The y-perturbation's reference to observed FRF (Fig. 28 and Fig. 29) shows a bandwidth, in the perturbed direction, of 6.40 Hz. Like the x-perturbation, the reference to estimated FRF shows that the actuation system's bandwidth exceeds the highest perturbed frequency of 128 Hz. Again, the manipulator end point data (Fig. 30 and Fig. 31) shows that the reference trajectory is not accurately followed and there is cross-axis motion.

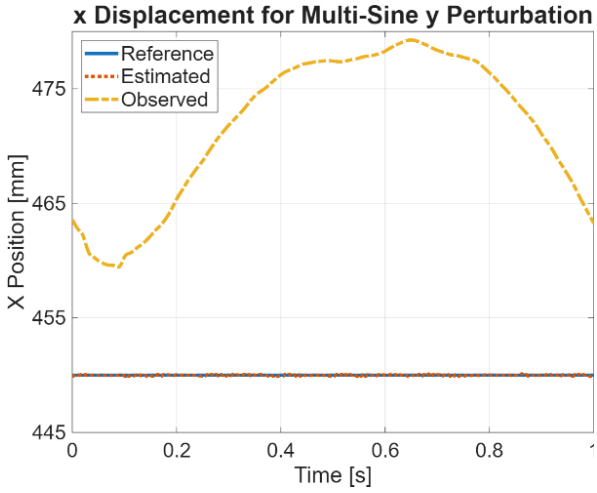


Fig. 30. Manipulator end point x-position data for the y-direction multi-sine perturbation of the NACT-3D at location 1.

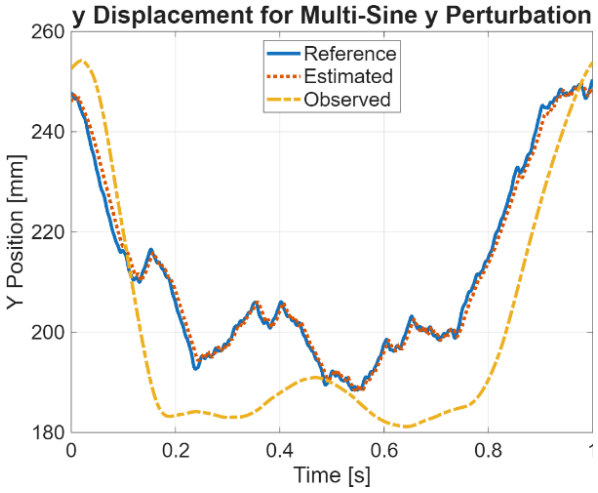


Fig. 31. Manipulator end point y-position data for the y-direction multi-sine perturbation of the NACT-3D at location 1.

All bandwidth estimations for the perturbations conducted with the manipulator are compiled in TABLE X. and show that the NACT-3D does not satisfy the bandwidth requirements in all locations of its work area.

TABLE X. BANDWIDTH OF NACT-3D WITH MANIPULATOR FROM REFERENCE TO OBSERVED DATA

Location	Perturbation Direction	Bandwidth
1	X	5.07
	Y	6.40
2	X	4.42
	Y	4.19
3	X	4.04
	Y	6.14
4	X	4.67
	Y	2.77
5	X	4.91
	Y	4.52

6.3.2. Actuation Experiments

The measured system bandwidth without the manipulator was significantly higher, ranging from 42 to 49 Hz. These results suggest that the NACT-3D's servo drives, control architecture, and transmission linkages are not the limiting factor. Instead, dynamic limitations originate from the current manipulator design, particularly its mass, compliance, and mechanical play.

The same experiments as were performed with the manipulator of the NACT-3D, were conducted for the NACT-3D without the manipulator attached. The OptoTrak data for the moment arms that actuate the manipulator segments was used to calculate the projected end point positions. The end point data was averaged and used to calculate the FRFs and is shown in bode plots.

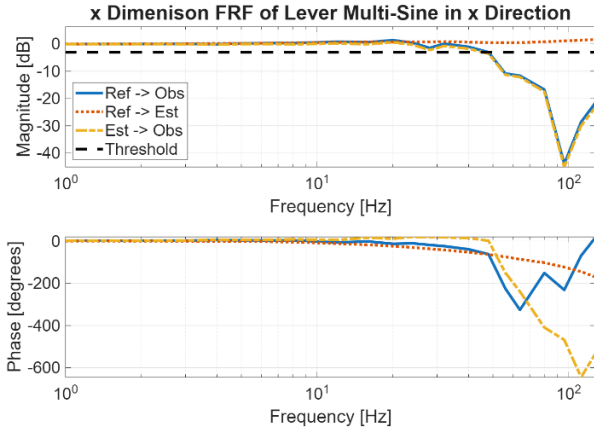


Fig. 32. Frequency response function in the x dimension for the x-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

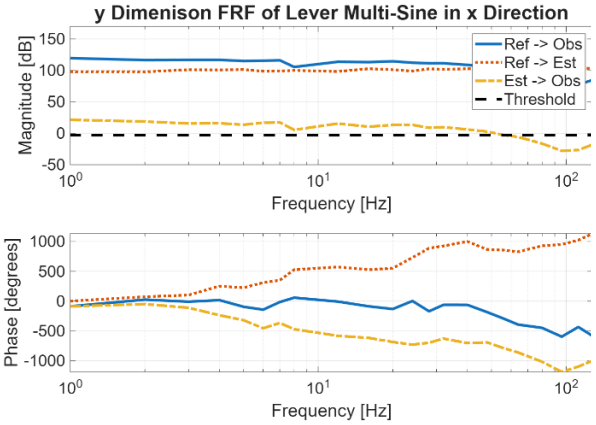


Fig. 33. Frequency response function in the y dimension for the x-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

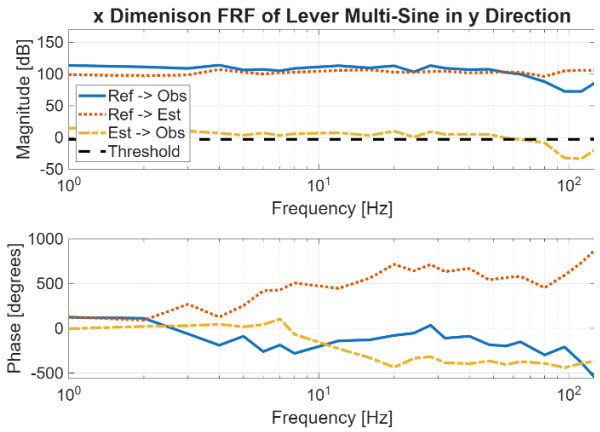


Fig. 34. Frequency response function in the x dimension for the y-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

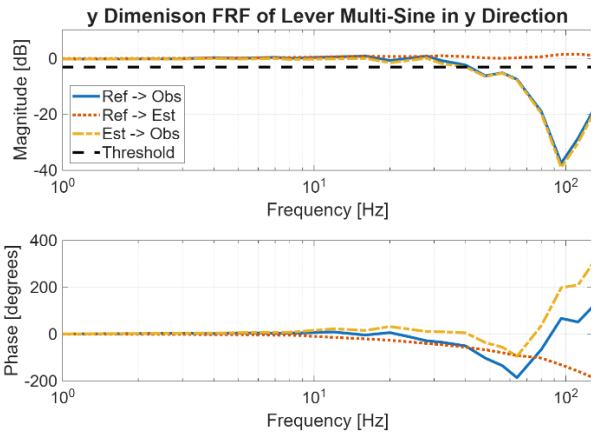


Fig. 35. Frequency response function in the y dimension for the y-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

The FRF for the experiments without the manipulator at location '1' in the 'x' dimension (Fig. 32 and Fig. 33) and 'y' dimension (Fig. 34 and Fig. 35) show that the NACT-3D's actuation linkage and controller perform roughly 8 times better than with the manipulator attached. The bandwidth without the manipulator at location '1' is 47.91 Hz in the 'x' dimension and 41.47 Hz in the 'y' dimension.

The FRF for the reference data input and estimated data output without the manipulator does not drop below the -3dB threshold. Similar to the experiments with the manipulator, this indicates that the bandwidth for the NACT-3D servo's and controller is higher than the highest tested frequency of 128 Hz. The magnitude does not drop below the −3 dB threshold, showing that the controller can perform sufficiently in the higher frequencies. However, there is a slight increase in magnitude. Ideally, the magnitude would be at 0 dB, indicating the output equals the input.

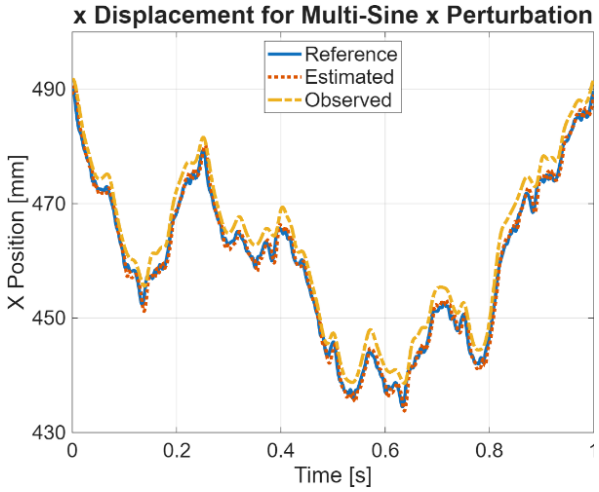


Fig. 36. Projected manipulator end point x-position data for the x-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

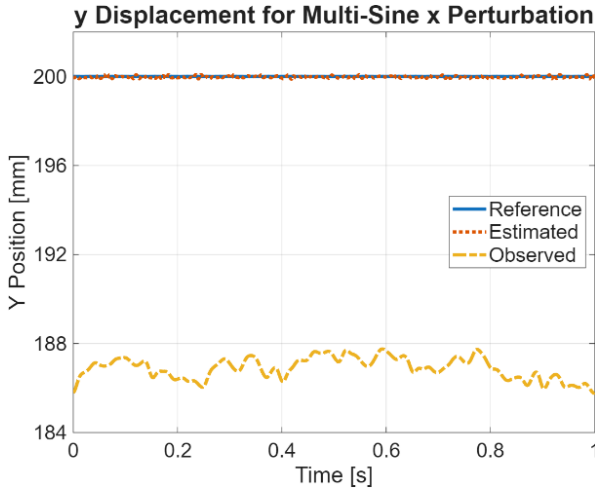


Fig. 37. Projected manipulator end point y-position data for the x-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

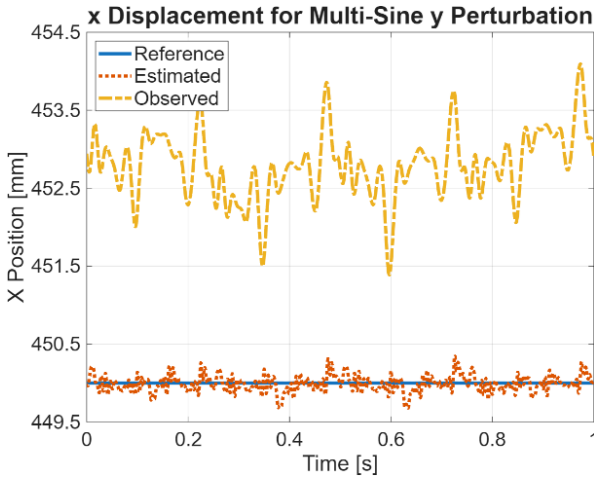


Fig. 38. Projected manipulator end point x-position data for the y-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

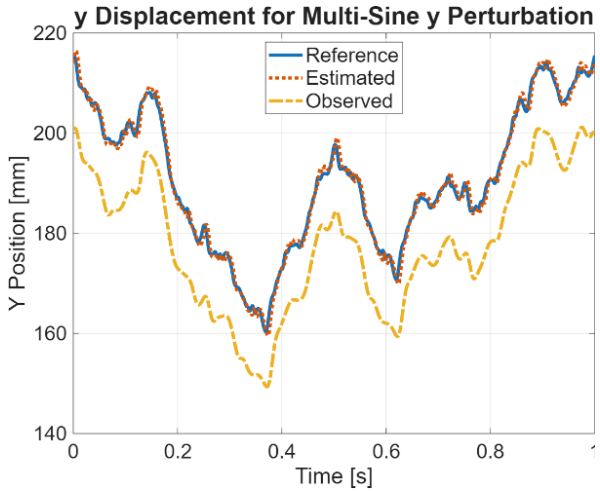


Fig. 39. Projected manipulator end point y-position data for the y-direction multi-sine perturbation of the NACT-3D without the manipulator at location 1.

The position data for the perturbations in the 'x' direction (Fig. 36 and Fig. 37) and 'y' direction (Fig. 38 and Fig. 39) at location '1' without the manipulator show that the NACT-3D's actuation is able to follow the reference signal quite accurately. Overall, the observed data seems a bit more smoothed than the reference signal, there is a small amount movement in the unperturbed dimension, and there is a slight offset in position.

TABLE XI. SYSTEM BANDWIDTH OF NACT-3D WITHOUT MANIPULATOR FROM REFERENCE TO OBSERVED DATA

Location	Perturbation Direction	Bandwidth [Hz]
1	X	47.91
	Y	41.47
2	X	46.70
	Y	41.70
3	X	42.79
	Y	43.73
4	X	45.74
	Y	40.94
5	X	49.65
	Y	41.67

All bandwidth results for the perturbations conducted without the manipulator show that the NACT-3D has a sufficiently high bandwidth in all locations of its work area (TABLE XI.).

6.4. SPACAR Simulation

The SPACAR model predicted bandwidths that significantly exceeded experimental values, highlighting the omission of motor saturation, joint friction, and control loop delays. This validated the utility of SPACAR for idealized

performance analysis but underscored the importance of physical testing.

The SPACAR model of the manipulator and actuation linkage was perturbed with a chirp signal including the following frequencies: 5Hz, 10Hz, 15Hz, 20Hz, 25Hz, 30Hz, 40Hz, 50Hz, 60Hz, 80Hz, and 100Hz. This was done at a sample frequency of 8400 ms as this number is divisible by all perturbation frequencies. After the simulations, the end point positions were extracted and are shown in Fig. 40. A zoomed in plot of the interesting frequencies is shown in Fig. 41.

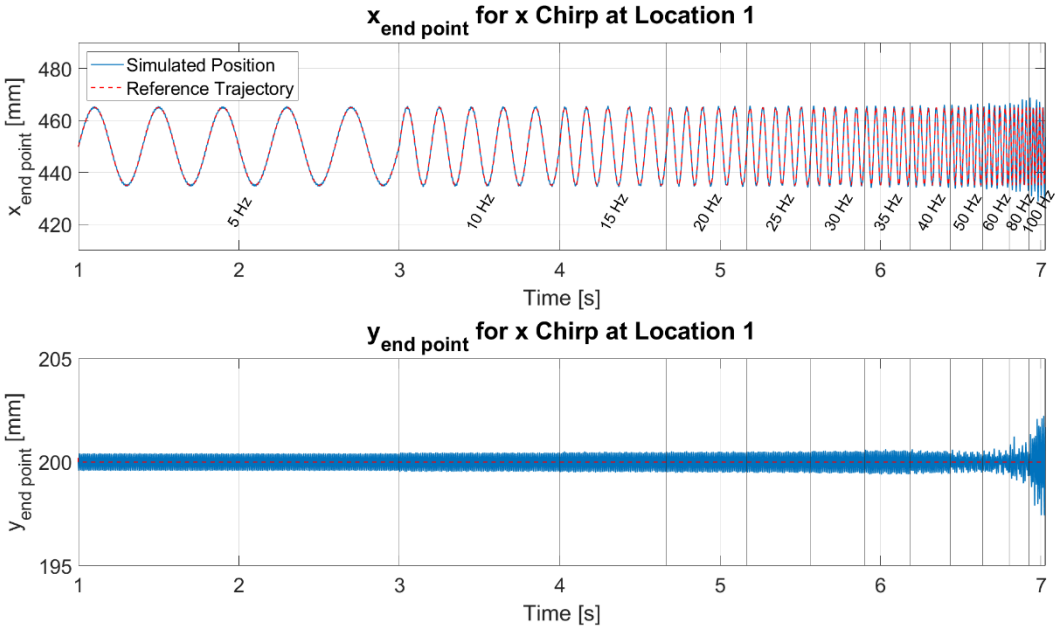


Fig. 40. SPACAR simulation end point positions for x-chirp perturbation at location 1.

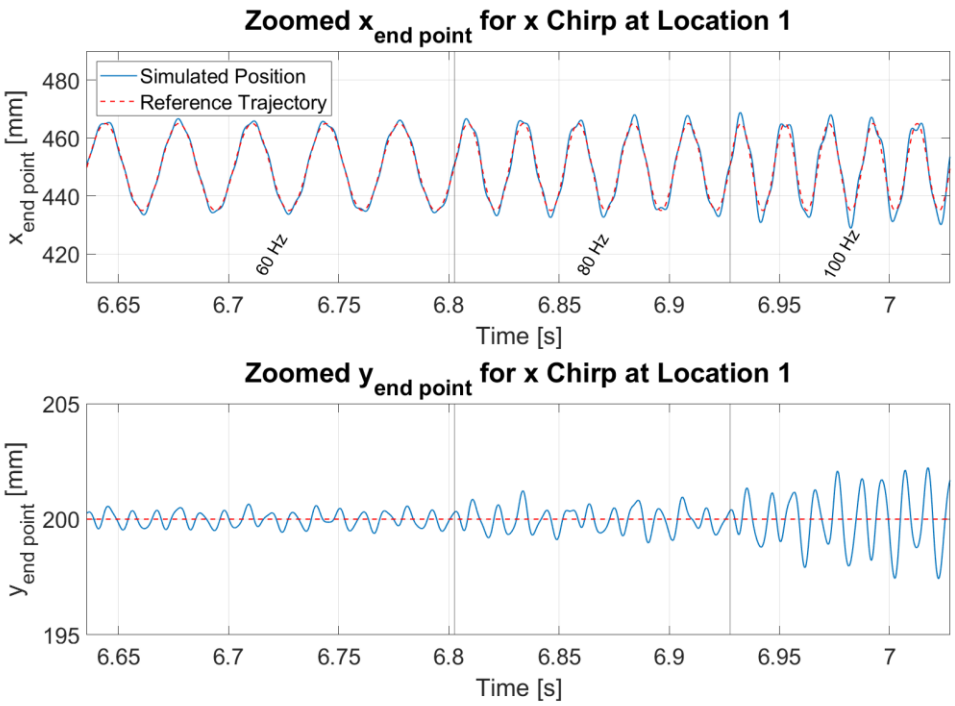


Fig. 41. SPACAR simulation end point positions for 60 Hz, 80 Hz, and 100 Hz x-chirp perturbation at location 1.

At 100Hz the end point is moving more than desired in the perturbed direction and there is also a notable amount of movement in the unperturbed direction.

The forces that the servo spindle must produce to realize the end point movement were also extracted from the simulation data and are shown in Fig. 42. The maximum force

that the NACT-3D actuation system can produce is also plotted. A zoomed in plot for the interesting frequencies is shown in Fig. 43.

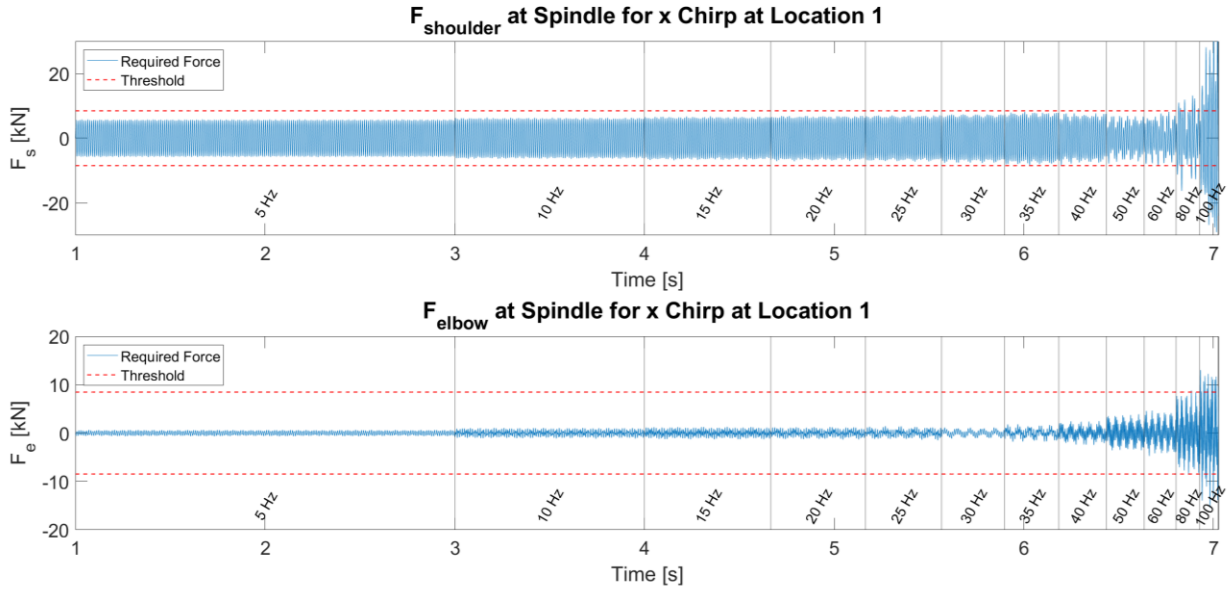


Fig. 42. SPACAR simulation spindle forces for x-chirp perturbation at location 1.

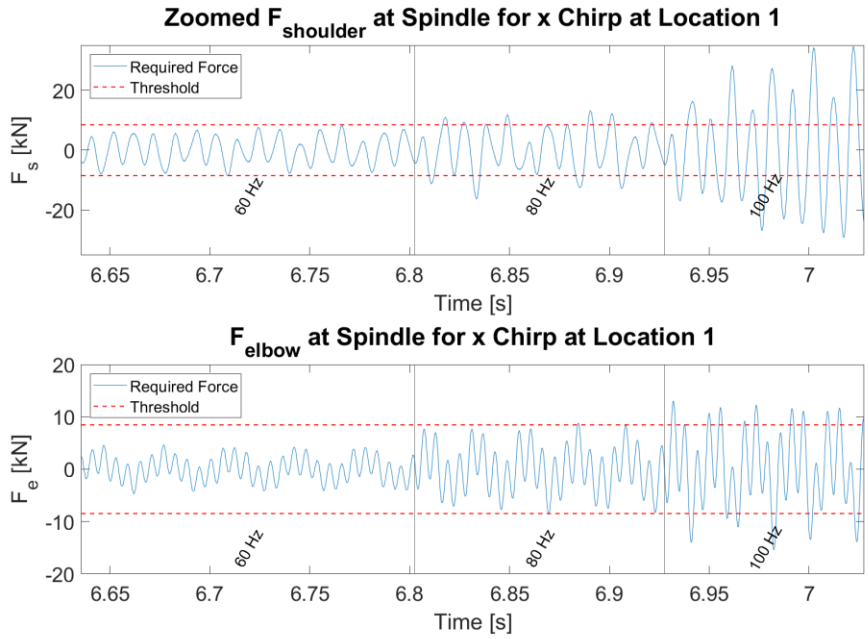


Fig. 43. SPACAR simulation spindle forces for 60 Hz, 80 Hz, and 100 Hz x-chirp perturbation at location 1.

Looking at the required servo spindle forces, at 100Hz the required spindle force exceeds the maximum forces that the NACT-3D can produce. In this simulation, the controller and inertia terms are not considered. Therefore, the required forces shown, and

necessary servo drive torques would be even higher. The NACT-3D cannot exceed these torques and forces during operation, and any estimated forces that are higher in the simulations can be regarded as unachievable. As such, the bandwidth of the system is less than the frequency at which the spindle force limits are exceeded.

The simulation data can be used for a similar system identification as in the experimental results and the FRF can be generated as shown in Fig. 44 and Fig. 45. Here the reference trajectory is used as the input and the simulated trajectory is used as the output for system identification.

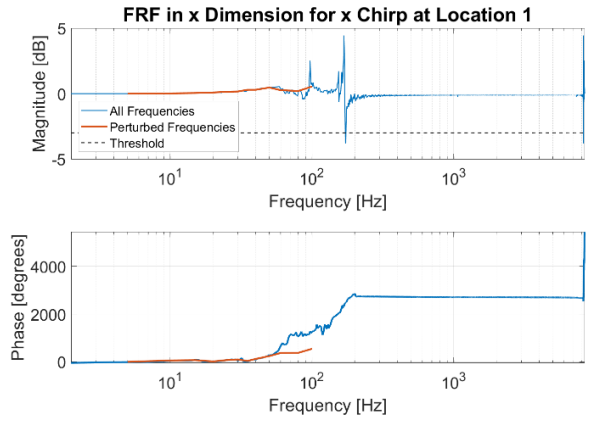


Fig. 44. SPACAR FRF in x domain for a chirp perturbation in the x direction at location 1.

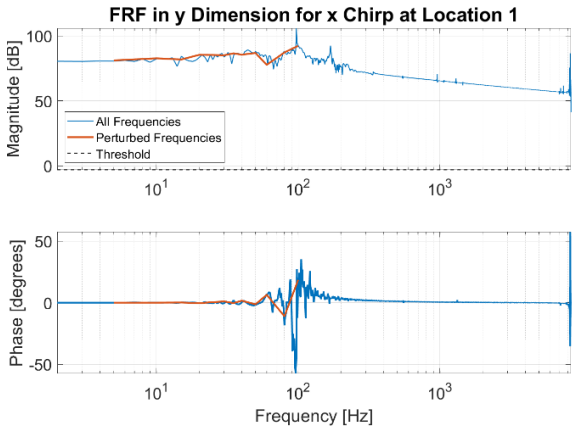


Fig. 45. SPACAR FRF in y domain for a chirp perturbation in the x direction at location 1.

The FRF of the simulation data shows that the magnitude of the system stays above 0. The simulated flexibility of the manipulator, and lack of limitations due to the servo's maximum torque, causes the manipulator to resonate. This resonation causes more movement than generated with the input signal which is reflected by an increase in system magnitude. The resonance can cause a clear difference between the reference positions and the observed positions. The lack of inclusion of servo limitations is reflected by the lack of magnitude drop off. The servo power limitations would cause less movement, as this is not included in the simulation the magnitude does not go down.

Looking at the highest frequencies where the required servo spindle force does not exceed the maximum servo spindle force, an approximation of the simulation bandwidths can be made for all simulations as shown in TABLE XII. . The data plots used to determine the simulated bandwidth estimates can be found in Appendix K. .

TABLE XII. SYSTEM BANDWIDTH OF NACT-3D SPACAR SIMULATIONS

Location	Perturbation Direction	Bandwidth [Hz]
1	X	< 80
	Y	< 10
2	X	< 40
	Y	< 15
3	X	< 15
	Y	< 15
4	X	< 25
	Y	< 15
5	X	< 80
	Y	< 15

The eigenfrequencies of the manipulator are also calculated with SPACAR. These eigenfrequencies show that the eigen modes caused by the manipulator's flexibility exceed the bandwidth performance of the NACT-3D. As the NACT-3D will not be able to generate these frequencies for the manipulator, the eigenmode will not be a limiting factor during operation.

It is important to note that these values may not accurately represent real-world behaviour due to the exclusion of control dynamics, sensor delays, and joint play. Nevertheless, these simulations provided valuable insights into the physical limitations of the manipulator and informed design adjustments aimed at increasing effective operational bandwidth.

7 CONCLUSION

This thesis sets out to evaluate whether the NACT-3D can generate high-fidelity virtual haptic environments and apply perturbations sufficient to characterize proprioceptive reflexes in the human arm. A structured approach was taken that included static force calculations, experimental perturbation testing of the manipulator as well as the actuation system, and dynamic simulations using SPACAR. These methods enabled assessment of both the current mechanical design and the control performance of the NACT-3D.

The experimental results demonstrate that the current manipulator imposes critical limitations on the system's dynamic performance. Specifically, bandwidth measurements during unloaded operation with the full manipulator attached revealed a system bandwidth of only 4-6 Hz, well below the 20 Hz threshold required for effective system identification of the human arm. However, actuation experiments conducted without the manipulator showed a significant increase in bandwidth of up to 49 Hz, indicating that the controller and actuation linkage are not the limiting factors. Instead, the manipulator's mass, inertia, and mechanical play substantially reduce overall system responsiveness.

SPACAR simulations further confirmed that the manipulator's physical structure could theoretically support higher bandwidth operation if actuator saturation and joint imperfections were mitigated. A lightweight carbon fibre manipulator was shown through static analysis to reduce mass without compromising stiffness, suggesting a viable design path for achieving the required bandwidth.

In summary, the following conclusions can be drawn:

- The current NACT-3D manipulator can withstand human-generated forces but does not meet the required dynamic performance criteria to yield meaningful clinical research results.
- The actuation and control system, independent of the manipulator, satisfy the bandwidth requirements.
- The manipulator's inertia and mechanical play severely limit the NACT-3D's bandwidth.
- A redesign of the manipulator using lightweight, high-stiffness materials such as carbon fiber can significantly improve dynamic performance.
- The NACT-3D has the potential to meet the design requirements if improvements are made to the manipulator design, without changes to the actuation system.

These findings support the feasibility of using the NACT-3D for advanced neurophysiological experiments, provided that mechanical refinements are implemented. Once the manipulator is redesigned and validated, the system will be well-positioned for future studies investigating human sensorimotor control and reflex modulation.

8 Discussion

The purpose of the experimental and simulation studies was to determine whether the NACT-3D satisfies the operational requirements for high-bandwidth, 3D haptic control and perturbation of the human upper limb. This discussion evaluates the findings from physical experiments and SPACAR simulations, highlights limitations in both modelling and experimental approaches, and considers implications for system improvements.

8.1. Comparison of Experimental and Simulated Performance

The comparison between experimental results and SPACAR simulations reveals discrepancies which can be attributed to modelling simplifications. The SPACAR simulations exclude the dynamics of the NACT-3D's closed-loop controller and servo systems, both of which significantly affect performance. Additionally, actuator limitations, such as peak torque and friction, are not accounted for in the simulations. The manipulator's actuation rods are simulated as rigid bodies, and torque cannot be directly applied in the simulation environment. Consequently, the simulated system permits unrealistically high accelerations and forces, which artificially inflates the estimated bandwidth.

Despite these limitations, the simulation results are valuable for identifying theoretical system performance. They indicate that the manipulator's elastic properties do not introduce resonances within the operational frequency range and suggest a theoretical bandwidth of approximately 100 Hz. However, this value represents an upper limit under idealized conditions, not the actual performance of the integrated system.

8.2. Effect of Manipulator on System Bandwidth

Experimental system identification results demonstrate a substantial performance gap between the NACT-3D with and without the manipulator. Without the manipulator, the actuation system achieves a bandwidth of approximately 40 Hz. With the manipulator attached, however, the bandwidth drops to approximately 5 Hz. This dramatic reduction in responsiveness is primarily attributed to the manipulator's high inertia and mechanical play in the joints. This confirms the hypothesis that the manipulator is the limiting factor in the system's dynamic performance.

The presence of unintended motion in the unperturbed axis during experiments supports the hypothesis of mechanical play. While some cross-axis motion is expected due to the manipulator’s kinematics as a MIMO system, the magnitude of undesired motion suggests excessive joint play. Experiments conducted without the manipulator demonstrate significantly less coupling and more accurate trajectory tracking, reinforcing the conclusion that mechanical deficiencies in the manipulator are a key bottleneck.

8.3. Measurement Limitations and Sources of Error

Several sources of measurement error were identified. The OptoTrak system, which was used to observe endpoint positions, likely introduced smoothing artifacts due to the amplification of small marker position errors during data extrapolation. These effects were particularly evident in trials with the manipulator, where the observed signals had slight offsets from the reference trajectories in the initial position.

Moreover, coordinate system alignment errors between the OptoTrak and the NACT-3D may have introduced systematic offsets. These errors become more pronounced at larger distances from the origin and cannot be conclusively disentangled from manipulator-induced discrepancies. Therefore, while the offset patterns observed in the data may indicate measurement misalignment, they are dwarfed by the poor mechanical performance of the current manipulator.

8.4. Implications for Future Improvements

The findings point toward clear avenues for performance enhancement. The actuation system, as tested in isolation, meets the bandwidth requirements, indicating that improvements to the manipulator alone could ensure the system meets requirements. Simulations of alternative manipulator designs using carbon fiber suggest that

significant reductions in mass and inertia are achievable without sacrificing stiffness. A redesigned manipulator with improved material properties could theoretically increase system bandwidth to 20 Hz, deemed sufficient for system identification of proprioceptive reflexes.

Improvements to the controller architecture, such as implementing computed torque control, could yield further gains. However, given that the actuation system already surpasses required performance metrics, these enhancements are not strictly necessary to meet the system's design goals.

BIBLIOGRAPHY

Aarts, R. G. K. M., Meijaard, J. P., & Jonker, J. B. (2011). *SPACAR User Manual*.

Automation, P. R. O. G. R. A. M. M. A., & Ns, S. (n.d.). *Kollmorgen Automation and Motion Control Kollmorgen : Your partner . In Motion . Every solution comes from a real understanding of*. *500*.

Bennett, D. J., Hollerbach, J. M., Xu, Y., & Hunter, I. W. (1992). Time-varying stiffness of human elbow joint during cyclic voluntary movement. *Experimental Brain Research*, *88*(2), 433–442. https://doi.org/10.1007/BF02259118

De Vlugt, E. (2004). Identification of Spinal Reflexes. In *Thesis*. http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Identification+of+Spinal+Reflexes#0

Derzsi, Z., & Volcic, R. (2018). MOTOM toolbox: MOtion Tracking via Optotrak and Matlab. *Journal of Neuroscience Methods*, *308*(August), 129–134. https://doi.org/10.1016/j.jneumeth.2018.07.007

Dietz, V., & Sinkjaer, T. (2007). Spastic movement disorder: impaired reflex function and altered muscle mechanics. *Lancet Neurology*, *6*(8), 725–733. https://doi.org/10.1016/S1474-4422(07)70193-X

Dirken, J. M., Steenbekkers, L. P. A., Daanen, Voorbij, A. I. M., & Molenbroek, J. F. M. (Johan). (2018). *Dutch adults (DINED - anthropometric database)*. 4TU.Centre for Research Data. https://doi.org/10.4121/uuid:1b214a8f-f59c-460f-8eb9-3ef8db5e85ee

Guo, X., Wallace, R., Tan, Y., Oetomo, D., Klaic, M., & Crocher, V. (2022). Technology-assisted assessment of spasticity: a systematic review. *Journal of NeuroEngineering and Rehabilitation*, *19*(1), 1–17. https://doi.org/10.1186/s12984-022-01115-2

Happee, R., De Vlugt, E., & Schouten, A. C. (2008). Posture maintenance of the human upper extremity; Identification of intrinsic and reflex based contributions. *SAE Technical Papers*, *April*. https://doi.org/10.4271/2008-01-1888

Heckman, C. J., & Enoka, R. M. (2012). Motor Unit. In *Comprehensive Physiology* (pp. 2629–2682). https://doi.org/https://doi.org/10.1002/cphy.c100087

Keemink, A. Q. L., van der Kooij, H., & Stienen, A. H. A. (2018). Admittance control for physical human–robot interaction. *International Journal of Robotics Research*, *37*(11), 1421–1444. https://doi.org/10.1177/0278364918768950

Leenen, T. (A. J. R.), Trigt, B. Van, Hoozemans, M. (M. J. M.), & Veeger, D. (H. E. J.). (2020). *Effects of a Disturbed Kinetic Chain in the Fastball Pitch on Elbow Kinetics and Ball Speed*. 67. https://doi.org/10.3390/proceedings2020049067

Li, S., & Francisco, G. E. (2015). New insights into the pathophysiology of post-stroke spasticity. *Frontiers in Human Neuroscience*, *9*(APRIL), 1–9. https://doi.org/10.3389/fnhum.2015.00192

Matthews, P. B. C., & Bagby, R. M. (1974). Mammalian Muscle Receptors and Their Central Actions. In *Medicine & Science in Sports & Exercise* (Vol. 6, Issue 2, p. iv). https://doi.org/10.1249/00005768-197400620-00005

Plaisier, T. A. M., Acosta, A. M., & Dewald, J. P. A. (2023). A Method for Quantification of Stretch Reflex Excitability During Ballistic Reaching. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, *31*, 2698–2704. https://doi.org/10.1109/TNSRE.2023.3283861

Prochazka, A. (1996). Proprioceptive Feedback and Movement Regulation. *Comprehensive Physiology*, *January*, 89–127. https://doi.org/10.1002/cphy.cp120103

Schouten, A. C., de Vlugt, E., van Hilten, J. J. B., & van der Helm, F. C. T. (2006). Design of a torque-controlled manipulator to analyse the admittance of the wrist joint. *Journal of Neuroscience Methods*, *154*(1–2), 134–141. https://doi.org/10.1016/j.jneumeth.2005.12.001

Shadmehr, R., & Mussa-ivaldi, F. A. (1994). Adaptive Task of Dynamics during Learning of a Motor. *The Journal of Neuroscience : The Official Journal of the Society for Neuroscience*, *14*(5), 3208–3224.

Sheean, G. (2002). Pathophysiology of spasticity. *European Journal of Neurology : The Official Journal of the European Federation of Neurological Societies*, *9 Suppl 1*, 3–9; dicussion 53. https://doi.org/10.1046/j.1468-1331.2002.0090s1003.x

Sisto, S. A., & Dyson-Hudson, T. (2007). Dynamometry testing in spinal cord injury. *Journal of Rehabilitation Research and Development*, *44*(1), 123–136. https://doi.org/10.1682/JRRD.2005.11.0172

TU Delft

Northwestern Medicine®
Feinberg School of Medicine

APPENDICES

ME51035 ME-BMD MSc Thesis

Modification of a 3D Haptic Robotic Device to Study Upper Limb Movement and Proprioceptive Reflexes

Philip Bernardus Leopold Raaphorst
4094395

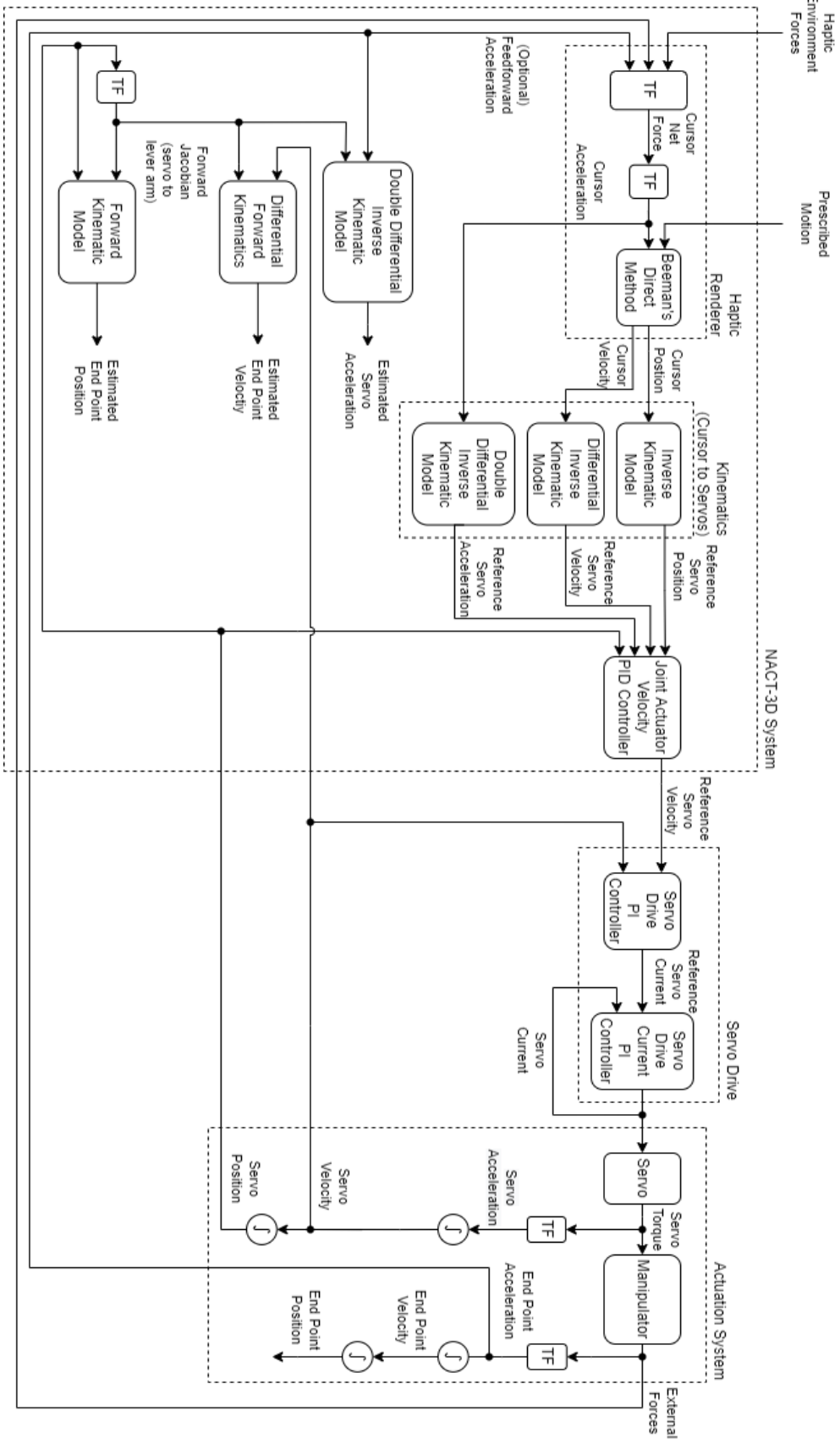
Thesis Committee

Prof. Dr. Frans C.T. van der Helm, Biomechanical Engineering, TU Delft
Prof. Dr. Julius P.A. Dewald, Biomedical Engineering, Northwestern University
Dr. Thomas Plaisier, Biomedical Engineering, Northwestern University
Prof. Dr. Ir. Alfred C. Schouten, Biomechanical Engineering, TU Delft

TU Delft

Northwestern Medicine®
Feinberg School of Medicine

APPENDIX A. NACT-3D Control Diagram



APPENDIX B. Maximum Human Static Force Trials

TABLE XIII. THOMAS' HUMAN ARM CONFIGURATION AND MEASUREMENTS (1 TRIAL)

Measurement	Value
Shoulder Abduction	75 °
Elbow Flexion	90 °
Horizontal Shoulder Adduction	40 °
Upper Arm Length	354 mm
Elbow to Attachment	204 mm
Vertical Offset Attachment	93 mm
Shoulder Abduction Max Torque	110 Nm
Shoulder Adduction Max Torque	103 Nm
Elbow Flexion Max Torque	96.5 Nm
Elbow Extension Max Torque	70.4 Nm

TABLE XIV. RESULTING MOMENTS AT THE MANIPULATOR'S ELBOW JOINT DURING THOMAS' TRIAL

Applied Human Force	Twisting Torque [Nm]	Vertical Bending Moment [Nm]	Horizontal Bending Moment [Nm]
Shoulder Abduction	53	31	-16
Shoulder Adduction	-49	-29	15
Elbow Flexion	-21	-12	-93
Elbow Extension	15	9	68

TABLE XV. RESULTING MOMENTS AT THE MANIPULATOR'S SHOULDER JOINT DURING THOMAS' TRIAL

Applied Human Force	Twisting Torque [Nm]	Vertical Bending Moment [Nm]	Horizontal Bending Moment [Nm]
Shoulder Abduction	-27	-146	-37
Shoulder Adduction	26	136	35
Elbow Flexion	110	67	-217
Elbow Extension	-80	-49	158

TABLE XVI. JULES' HUMAN ARM CONFIGURATION AND MEASUREMENTS (BEST OF 3 TRIALS)

Measurement	Value
Shoulder Abduction	90 °
Elbow Flexion	90 °
Horizontal Shoulder Adduction	40 °
Upper Arm Length	340 mm
Elbow to Attachment	255 mm
Vertical Offset Attachment	101 mm
Shoulder Abduction Max Torque	78.8 Nm
Shoulder Adduction Max Torque	111.6 Nm
Elbow Flexion Max Torque	108.6 Nm
Elbow Extension Max Torque	73.7 Nm

TABLE XVII. RESULTING MOMENTS AT THE MANIPULATOR'S ELBOW JOINT DURING JULES' TRIAL

Applied Human Force	Twisting Torque [Nm]	Vertical Bending Moment [Nm]	Horizontal Bending Moment [Nm]
Shoulder Abduction	51	30	0
Shoulder Adduction	-72	-42	0
Elbow Flexion	0	0	-109
Elbow Extension	0	0	74

TABLE XVIII. RESULTING MOMENTS AT THE MANIPULATOR'S SHOULDER JOINT DURING JULES' TRIAL

Applied Human Force	Twisting Torque [Nm]	Vertical Bending Moment [Nm]	Horizontal Bending Moment [Nm]
Shoulder Abduction	-16	-127	0
Shoulder Adduction	22	180	0
Elbow Flexion	85	17	-223
Elbow Extension	-58	-12	152

APPENDIX C. Static Force and Deflection Calculation Tool Code

This appendix contains the MATLAB code used to create the GUI with which the static forces, moments, and deflections are calculated.

A.1. NACT_StaticCalculatorGUI.m

```
% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

% This is the main script that generates a GUI which calculates forces,
% moments, and the corresponding deflections for a humanoid robot arm. The
% GUI includes interactive plots that visualize the calculated results and
% input fields which can be used to alter the arm specifications and
% orientation.

clearvars
close all

simple_gui

function simple_gui
%  Create and then hide the UI as it is being constructed.
    gui = figure('Visible','on',...
        'Name','Simple NACT Upper Arm Force Calculator',...
        'units','normalized','Position',[.1 .05 .8 .85]);

%% Panel - Plot Tab Group
tabgp = uitabgroup('Parent',gui,...
                'units','normalized',...
                'Position',[.41 .26 .58 .73]);
tabFandM = uitab(tabgp,'Title','Force and Moment Surf Plots');
tabDefl = uitab(tabgp,'Title','Deflection Surf Plots');
tabMax = uitab(tabgp,'Title','Maximum Deflections');

%% Tab - Force and Moment Surf Plots
    pFSurf = uipanel('Parent',tabFandM,...
                'Title','',...
                'Position',[0 0 1 1]);
        % compression 3d plot
        axFComp = axes('Parent',pFSurf,...
                    'View',[45 35],...
                    'XGrid','on','YGrid','on','ZGrid','on',...
                    'XLim',[0 90],'XTick',[0 45 90],...
                    'YLim',[0 90],'YTick',[0 45 90],...
                    'Position',[.1 .59 .36 .35]);
        axFComp.Title.String = 'Compressive Force in Upper Arm';
        axFComp.XLabel.String = '\Theta_2 [deg]';
        axFComp.YLabel.String = '\Theta_3 [deg]';
        axFComp.ZLabel.String = 'F_{compression} [N]';
        colormap(axFComp,flip(autumn()))
        setappdata(gui,'axFComp',axFComp);
        hold on
        % horizontal bending force 3d plot
        axFBendHor = axes('Parent',pFSurf,...
                    'View',[45 35],...
                    'XGrid','on','YGrid','on','ZGrid','on',...
                    'XLim',[0 90],'XTick',[0 45 90],...
                    'YLim',[0 90],'YTick',[0 45 90],...
                    'Position',[.59 .59 .36 .35]);
        axFBendHor.Title.String = 'Horizontal Bending Force in Upper Arm';
        axFBendHor.XLabel.String = '\Theta_2 [deg]';
        axFBendHor.YLabel.String = '\Theta_3 [deg]';
        axFBendHor.ZLabel.String = 'F_{bending} [N]';
        colormap(axFBendHor,[flip(parula()); parula()])
        setappdata(gui,'axFBendHor',axFBendHor);
        hold on
```

```
        % Vertical bending moment 3d plot
        axMBendVer = axes('Parent',pFSurf,...
                    'View',[45 35],...
                    'XGrid','on','YGrid','on','ZGrid','on',...
                    'XLim',[0 90],'XTick',[0 45 90],...
                    'YLim',[0 90],'YTick',[0 45 90],...
                    'Position',[.1 .1 .36 .35]);
        axMBendVer.Title.String = 'Vertical Bending Force in Upper Arm';
        axMBendVer.XLabel.String = '\Theta_2 [deg]';
        axMBendVer.YLabel.String = '\Theta_3 [deg]';
        axMBendVer.ZLabel.String = 'M_{bending} [Nm]';
        colormap(axMBendVer,[flip(parula()); parula()])
        setappdata(gui,'axMBendVer',axMBendVer);
        hold on
        % Vertical bending moment 3d plot
        axTorq = axes('Parent',pFSurf,...
                    'View',[45 35],...
                    'XGrid','on','YGrid','on','ZGrid','on',...
                    'XLim',[0 90],'XTick',[0 45 90],...
                    'YLim',[0 90],'YTick',[0 45 90],...
                    'Position',[.59 .1 .36 .35]);
        axTorq.Title.String = 'Torque in Upper Arm';
        axTorq.XLabel.String = '\Theta_2 [deg]';
        axTorq.YLabel.String = '\Theta_3 [deg]';
        axTorq.ZLabel.String = 'T [Nm]';
        colormap(axTorq,[flip(parula()); parula()])
        setappdata(gui,'axTorq',axTorq);
        hold on

%% Tab - Deflection Surf Plots
    pdSurf = uipanel('Parent',tabDefl,...
                'Title','',...
                'Position',[0 0 1 1]);
        % compression 3d plot
        axdLowVer = axes('Parent',pdSurf,...
                    'View',[45 35],...
                    'XGrid','on','YGrid','on','ZGrid','on',...
                    'XLim',[0 90],'XTick',[0 45 90],...
                    'YLim',[0 90],'YTick',[0 45 90],...
                    'Position',[.1 .59 .36 .35]);
        axdLowVer.Title.String = 'Lower Arm Vertical Deflection';
        axdLowVer.XLabel.String = '\Theta_2 [deg]';
        axdLowVer.YLabel.String = '\Theta_3 [deg]';
        axdLowVer.ZLabel.String = '\delta [mm]';
        colormap(axdLowVer,flip(autumn()))
        setappdata(gui,'axdLowVer',axdLowVer);
        hold on
        % legend and change highlighted surf plot
        pTubeSurf = uibuttongroup('Parent',pdSurf,...
                    'Title','Legend',...
                    'Position',[.59 .59 .36 .35]);
        setappdata(gui,'pTubeUpp',pTubeSurf);
            uicontrol(pTubeSurf,'Style','radiobutton',...
                        'String',' One Round Tube',...
                        'BackgroundColor','#0072BD',...
                        'Tag','TubeSurfCy_1',...
                        'Callback',@eTubeSurf_Callback,...
                        'units','normalized',...
                        'Position',[.05 .82 .9 .15]);
            uicontrol(pTubeSurf,'Style','radiobutton',...
                        'String',' Two Round Horizontal Tubes',...
                        'BackgroundColor','#D95319',...
                        'Tag','TubeSurfCy_2h',...
                        'Callback',@eTubeSurf_Callback,...
                        'units','normalized',...
                        'Position',[.05 .67 .9 .15]);
            uicontrol(pTubeSurf,'Style','radiobutton',...
                        'String',' Two Round Vertical Tubes',...
                        'BackgroundColor','#EDB120',...
                        'Tag','TubeSurfCy_2v',...
                        'Callback',@eTubeSurf_Callback,...
                        'units','normalized',...
                        'Position',[.05 .52 .9 .15]);
            uicontrol(pTubeSurf,'Style','radiobutton',...
```

```
                    'String',' One Square Tube',...
                    'BackgroundColor','#7E2F8E',...
                    'Tag','TubeSurfSq_1',...
                    'Callback',@eTubeSurf_Callback,...
                    'units','normalized',...
                    'Position',[.05 .37 .9 .15]);
        uicontrol(pTubeSurf,'Style','radiobutton',...
                    'String',' Two Square Horizontal Tubes',...
                    'BackgroundColor','#77AC30',...
                    'Tag','TubeSurfSq_2h',...
                    'Callback',@eTubeSurf_Callback,...
                    'units','normalized',...
                    'Position',[.05 .22 .9 .15]);
        uicontrol(pTubeSurf,'Style','radiobutton',...
                    'String',' Two Square Vertical Tubes',...
                    'BackgroundColor','#A2142F',...
                    'Tag','TubeSurfSq_2v',...
                    'Callback',@eTubeSurf_Callback,...
                    'units','normalized',...
                    'Position',[.05 .077 .9 .15]);
    % Vertical bending moment 3d plot
    axdUppVer = axes('Parent',pdSurf,...
                'View',[45 35],...
                'XGrid','on','YGrid','on','ZGrid','on',...
                'XLim',[0 90],'XTick',[0 45 90],...
                'YLim',[0 90],'YTick',[0 45 90],...
                'Position',[.1 .1 .36 .35]);
    axdUppVer.Title.String = 'Upper Arm Vertical Deflection';
    axdUppVer.XLabel.String = '\Theta_2 [deg]';
    axdUppVer.YLabel.String = '\Theta_3 [deg]';
    axdUppVer.ZLabel.String = '\delta [mm]';
    colormap(axdUppVer,[flip(parula()); parula()])
    setappdata(gui,'axdUppVer',axdUppVer);
    hold on
    % Vertical bending moment 3d plot
    axdUppHor = axes('Parent',pdSurf,...
                'View',[45 35],...
                'XGrid','on','YGrid','on','ZGrid','on',...
                'XLim',[0 90],'XTick',[0 45 90],...
                'YLim',[0 90],'YTick',[0 45 90],...
                'Position',[.59 .1 .36 .35]);
    axdUppHor.Title.String = 'Upper Arm Horizontal Deflection';
    axdUppHor.XLabel.String = '\Theta_2 [deg]';
    axdUppHor.YLabel.String = '\Theta_3 [deg]';
    axdUppHor.ZLabel.String = '\delta [mm]';
    colormap(axdUppHor,[flip(parula()); parula()])
    setappdata(gui,'axdUppHor',axdUppHor);
    hold on

%% Tab - Max Deflections Results
    pMax = uipanel('Parent',tabMax,...
                'Title','Maximum Absolute Deflections',...
                'units','normalized',...
                'Position',[0 .5 1 .3]);
    pM1x = 0;   pM1w = .27; pM2x = .28;pM2w = .11;  pMh = .155;
        % titles
        uicontrol(pMax,'Style','text',...
                    'HorizontalAlignment','center',...
                    'FontWeight','bold',...
                    'String','1 Cylinder',...
                    'units','normalized',...
                    'Position',[(pM2x+0*(pM2w+.01)) .835 pM2w pMh]);
        uicontrol(pMax,'Style','text',...
                    'HorizontalAlignment','center',...
                    'FontWeight','bold',...
                    'String','2 Hor. Cy.',...
                    'units','normalized',...
                    'Position',[(pM2x+1*(pM2w+.01)) .835 pM2w pMh]);
        uicontrol(pMax,'Style','text',...
                    'HorizontalAlignment','center',...
                    'FontWeight','bold',...
                    'String','2 Ver. Cy.',...
                    'units','normalized',...
                    'Position',[(pM2x+2*(pM2w+.01)) .835 pM2w pMh]);
```

```
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','center',...
            'FontWeight','bold',...
            'String','1 Square',...
            'units','normalized',...
            'Position',[(pM2x+3*(pM2w+.01)) .835 pM2w pMh]);
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','center',...
            'FontWeight','bold',...
            'String','2 Hor. Sq.',...
            'units','normalized',...
            'Position',[(pM2x+4*(pM2w+.01)) .835 pM2w pMh]);
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','center',...
            'FontWeight','bold',...
            'String','2 Ver. Sq.',...
            'units','normalized',...
            'Position',[(pM2x+5*(pM2w+.01)) .835 pM2w pMh]);

% vertical wrist deflection
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','right',...
            'String','Vertical Wrist Deflection [mm] = ',...
            'units','normalized',...
            'Position',[pM1x .67 pM1w pMh]);
tdmLowVCy1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+0*(pM2w+.01)) .67 pM2w pMh]);
setappdata(gui,'tdmLowVCy1',tdmLowVCy1);
tdmLowVCy2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+1*(pM2w+.01)) .67 pM2w pMh]);
setappdata(gui,'tdmLowVCy2h',tdmLowVCy2h);
tdmLowVCy2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+2*(pM2w+.01)) .67 pM2w pMh]);
setappdata(gui,'tdmLowVCy2v',tdmLowVCy2v);
tdmLowVSq1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+3*(pM2w+.01)) .67 pM2w pMh]);
setappdata(gui,'tdmLowVSq1',tdmLowVSq1);
tdmLowVSq2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+4*(pM2w+.01)) .67 pM2w pMh]);
setappdata(gui,'tdmLowVSq2h',tdmLowVSq2h);
tdmLowVSq2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+5*(pM2w+.01)) .67 pM2w pMh]);
setappdata(gui,'tdmLowVSq2v',tdmLowVSq2v);

% horizontal wrist deflection
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','right',...
            'String','Horizontal Wrist Deflection [mm] = ',...
            'units','normalized',...
            'Position',[pM1x .505 pM1w pMh]);
tdmLowHCy1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+0*(pM2w+.01)) .505 pM2w pMh]);
setappdata(gui,'tdmLowHCy1',tdmLowHCy1);
tdmLowHCy2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+1*(pM2w+.01)) .505 pM2w pMh]);
setappdata(gui,'tdmLowHCy2h',tdmLowHCy2h);
tdmLowHCy2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+2*(pM2w+.01)) .505 pM2w pMh]);
setappdata(gui,'tdmLowHCy2v',tdmLowHCy2v);
tdmLowHSq1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+3*(pM2w+.01)) .505 pM2w pMh]);
setappdata(gui,'tdmLowHSq1',tdmLowHSq1);
tdmLowHSq2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
```

```
            'Position',[(pM2x+4*(pM2w+.01)) .505 pM2w pMh]);
setappdata(gui,'tdmLowHSq2h',tdmLowHSq2h);
tdmLowHSq2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+5*(pM2w+.01)) .505 pM2w pMh]);
setappdata(gui,'tdmLowHSq2v',tdmLowHSq2v);

% vertical elbow deflection
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','right',...
            'String','Vertical Elbow Deflection [mm] = ',...
            'units','normalized',...
            'Position',[pM1x .34 pM1w pMh]);
tdmUppVCy1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+0*(pM2w+.01)) .34 pM2w pMh]);
setappdata(gui,'tdmUppVCy1',tdmUppVCy1);
tdmUppVCy2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+1*(pM2w+.01)) .34 pM2w pMh]);
setappdata(gui,'tdmUppVCy2h',tdmUppVCy2h);
tdmUppVCy2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+2*(pM2w+.01)) .34 pM2w pMh]);
setappdata(gui,'tdmUppVCy2v',tdmUppVCy2v);
tdmUppVSq1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+3*(pM2w+.01)) .34 pM2w pMh]);
setappdata(gui,'tdmUppVSq1',tdmUppVSq1);
tdmUppVSq2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+4*(pM2w+.01)) .34 pM2w pMh]);
setappdata(gui,'tdmUppVSq2h',tdmUppVSq2h);
tdmUppVSq2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+5*(pM2w+.01)) .34 pM2w pMh]);
setappdata(gui,'tdmUppVSq2v',tdmUppVSq2v);

% horizontal elbow deflection
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','right',...
            'String','Horizontal Elbow Deflection [mm] = ',...
            'units','normalized',...
            'Position',[pM1x .175 pM1w pMh]);
tdmUppHCy1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+0*(pM2w+.01)) .175 pM2w pMh]);
setappdata(gui,'tdmUppHCy1',tdmUppHCy1);
tdmUppHCy2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+1*(pM2w+.01)) .175 pM2w pMh]);
setappdata(gui,'tdmUppHCy2h',tdmUppHCy2h);
tdmUppHCy2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+2*(pM2w+.01)) .175 pM2w pMh]);
setappdata(gui,'tdmUppHCy2v',tdmUppHCy2v);
tdmUppHSq1 = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+3*(pM2w+.01)) .175 pM2w pMh]);
setappdata(gui,'tdmUppHSq1',tdmUppHSq1);
tdmUppHSq2h = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+4*(pM2w+.01)) .175 pM2w pMh]);
setappdata(gui,'tdmUppHSq2h',tdmUppHSq2h);
tdmUppHSq2v = uicontrol(pMax,'Style','text',...
            'units','normalized',...
            'Position',[(pM2x+5*(pM2w+.01)) .175 pM2w pMh]);
setappdata(gui,'tdmUppHSq2v',tdmUppHSq2v);

% upper arm compression
uicontrol(pMax,'Style','text',...
            'HorizontalAlignment','right',...
            'String','Upper Arm Compression [mm] = ',...
            'units','normalized',...
```

```
                'Position',[pM1x .01 pM1w pMh]);
        tdmUppCompCy1 = uicontrol(pMax,'Style','text',...
                'units','normalized',...
                'Position',[(pM2x+0*(pM2w+.01)) .01 pM2w pMh]);
        setappdata(gui,'tdmUppCompCy1',tdmUppCompCy1);
        tdmUppCompCy2h = uicontrol(pMax,'Style','text',...
                'units','normalized',...
                'Position',[(pM2x+1*(pM2w+.01)) .01 pM2w pMh]);
        setappdata(gui,'tdmUppCompCy2h',tdmUppCompCy2h);
        tdmUppCompCy2v = uicontrol(pMax,'Style','text',...
                'units','normalized',...
                'Position',[(pM2x+2*(pM2w+.01)) .01 pM2w pMh]);
        setappdata(gui,'tdmUppCompCy2v',tdmUppCompCy2v);
        tdmUppCompSq1 = uicontrol(pMax,'Style','text',...
                'units','normalized',...
                'Position',[(pM2x+3*(pM2w+.01)) .01 pM2w pMh]);
        setappdata(gui,'tdmUppCompSq1',tdmUppCompSq1);
        tdmUppCompSq2h = uicontrol(pMax,'Style','text',...
                'units','normalized',...
                'Position',[(pM2x+4*(pM2w+.01)) .01 pM2w pMh]);
        setappdata(gui,'tdmUppCompSq2h',tdmUppCompSq2h);
        tdmUppCompSq2v = uicontrol(pMax,'Style','text',...
                'units','normalized',...
                'Position',[(pM2x+5*(pM2w+.01)) .01 pM2w pMh]);
        setappdata(gui,'tdmUppCompSq2v',tdmUppCompSq2v);

%% Panel - Dimension Text boxes
    pDim = uipanel('Parent',gui,...
                'Title','Dimensions and End Forces',...
                'Position',[.01 .8 .39 .19]);
        uicontrol(pDim,'Style','text',...
                'HorizontalAlignment','right',...
                'String','Lower Arm Length [m] = ',...
                'units','normalized',...
                'Position',[0 .7 .34 .25]);
        eLow = uicontrol(pDim,'Style','edit',...
                'String','0.45',...
                'Callback',@eDim_Callback,...
                'units','normalized',...
                'Position',[.35 .785 .14 .21]);
        setappdata(gui,'eLow',eLow);
        uicontrol(pDim,'Style','text',...
                'HorizontalAlignment','right',...
                'String','Upper Arm Length [m] = ',...
                'units','normalized',...
                'Position',[0 .45 .34 .25]);
        eUpp = uicontrol(pDim,'Style','edit',...
                'String','0.5',...
                'Callback',@eDim_Callback,...
                'units','normalized',...
                'Position',[.35 .535 .14 .21]);
        setappdata(gui,'eUpp',eUpp);
        uicontrol(pDim,'Style','text',...
                'HorizontalAlignment','right',...
                'String','Pulley Radius [m] = ',...
                'units','normalized',...
                'Position',[0 .2 .34 .25]);
        ePul = uicontrol(pDim,'Style','edit',...
                'String','0.05',...
                'Callback',@eDim_Callback,...
                'units','normalized',...
                'Position',[.35 .285 .14 .21]);
        setappdata(gui,'ePul',ePul);
        uicontrol(pDim,'Style','text',...
                'HorizontalAlignment','right',...
                'String','Actuation Radius [m] = ',...
                'units','normalized',...
                'Position',[0 -.05 .34 .25]);
        eArm = uicontrol(pDim,'Style','edit',...
                'String','0.08',...
                'Callback',@eDim_Callback,...
                'units','normalized',...
                'Position',[.35 .035 .14 .21]);
        setappdata(gui,'eArm',eArm);
```

```
        uicontrol(pDim,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Horizontal End Force [N] = ',...
                    'units','normalized',...
                    'Position',[.5 .7 .34 .25]);
        eFHor = uicontrol(pDim,'Style','edit',...
                    'String','600',...
                    'Callback',@eDim_Callback,...
                    'units','normalized',...
                    'Position',[.85 .785 .14 .21]);
        setappdata(gui,'eFHor',eFHor);
        uicontrol(pDim,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Vertical End Force [N] = ',...
                    'units','normalized',...
                    'Position',[.5 .45 .34 .25]);
        eFVer = uicontrol(pDim,'Style','edit',...
                    'String','-500',...
                    'Callback',@eDim_Callback,...
                    'units','normalized',...
                    'Position',[.85 .535 .14 .21]);
        setappdata(gui,'eFVer',eFVer);
        % Button
        pbFCalc = uicontrol('Parent',pDim,'Style','pushbutton',...
                    'String','Calculate with new Values',...
                    'FontWeight','bold',...
                    'Callback',@pbCalc_Callback,...
                    'units','normalized',...
                    'Position',[.6 .1 .35 .25]);
        setappdata(gui,'pbFCalc',pbFCalc);

%% Panel - Segment tube selection
    pTube = uibuttongroup('Parent',gui,...
                'Title','Arm Segment Selection for Specific Angle Calculation',...
                'units','normalized',...
                'Position',[.01 .55 .39 .24]);
        pTubeLow = uibuttongroup('Parent',pTube,...
                    'Title','Lower',...
                    'units','normalized',...
                    'Position',[0 0 .1 1]);
        setappdata(gui,'pTubeLow',pTubeLow);
            uicontrol(pTubeLow,'Style','radiobutton',...
                        'Tag','TubeLowCy_1',...
                        'Callback',@eTube_Callback,...
                        'units','normalized',...
                        'Position',[.25 .835 1 .155]);
            uicontrol(pTubeLow,'Style','radiobutton',...
                        'Tag','TubeLowCy_2h',...
                        'Callback',@eTube_Callback,...
                        'units','normalized',...
                        'Position',[.25 .67 1 .155]);
            uicontrol(pTubeLow,'Style','radiobutton',...
                        'Tag','TubeLowCy_2v',...
                        'Callback',@eTube_Callback,...
                        'units','normalized',...
                        'Position',[.25 .505 1 .155]);
            uicontrol(pTubeLow,'Style','radiobutton',...
                        'Tag','TubeLowSq_1',...
                        'Callback',@eTube_Callback,...
                        'units','normalized',...
                        'Position',[.25 .34 1 .155]);
            uicontrol(pTubeLow,'Style','radiobutton',...
                        'Tag','TubeLowSq_2h',...
                        'Callback',@eTube_Callback,...
                        'units','normalized',...
                        'Position',[.25 .175 1 .155]);
            uicontrol(pTubeLow,'Style','radiobutton',...
                        'Tag','TubeLowSq_2v',...
                        'Callback',@eTube_Callback,...
                        'units','normalized',...
                        'Position',[.25 .01 1 .155]);

        pTubeUpp = uibuttongroup('Parent',pTube,...
```

```
            'Title','and Upper Arm',...
            'units','normalized',...
            'Position',[.1 0 .4 1]);
    setappdata(gui,'pTubeUpp',pTubeUpp);
        uicontrol(pTubeUpp,'Style','radiobutton',...
                'String',' One Round Tube',...
                'Tag','TubeUppCy_1',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.05 .835 1 .155]);
        uicontrol(pTubeUpp,'Style','radiobutton',...
                'String',' Two Round Horizontal Tubes',...
                'Tag','TubeUppCy_2h',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.05 .67 1 .155]);
        uicontrol(pTubeUpp,'Style','radiobutton',...
                'String',' Two Round Vertical Tubes',...
                'Tag','TubeUppCy_2v',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.05 .505 1 .155]);
        uicontrol(pTubeUpp,'Style','radiobutton',...
                'String',' One Square Tube',...
                'Tag','TubeUppSq_1',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.05 .34 1 .155]);
        uicontrol(pTubeUpp,'Style','radiobutton',...
                'String',' Two Square Horizontal Tubes',...
                'Tag','TubeUppSq_2h',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.05 .175 1 .155]);
        uicontrol(pTubeUpp,'Style','radiobutton',...
                'String',' Two Square Vertical Tubes',...
                'Tag','TubeUppSq_2v',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.05 .01 1 .155]);
    % button
    pbdCalc = uicontrol('Parent',pTube,'Style','pushbutton',...
            'String','Calculate Deflections',...
            'FontWeight','bold',...
            'Callback',@pbDeflCalc_Callback,...
            'units','normalized',...
            'Position',[.7 .845 .275 .15]);

    % text
    uicontrol(pTube,'Style','text',...
            'HorizontalAlignment','center',...
            'String','Lower',...
            'units','normalized',...
            'Position',[.52 .805 .08 .15]);
    uicontrol(pTube,'Style','text',...
            'HorizontalAlignment','center',...
            'String','Upper',...
            'units','normalized',...
            'Position',[.6 .805 .08 .15]);
    eLowDout = uicontrol(pTube,'Style','edit',...
            'String','30',...
            'Callback',@eTube_Callback,...
            'units','normalized',...
            'Position',[.52 .68 .08 .15]);
    setappdata(gui,'eLowDout',eLowDout);
    eUppDout = uicontrol(pTube,'Style','edit',...
            'String','30',...
            'Callback',@eTube_Callback,...
            'units','normalized',...
            'Position',[.6 .68 .08 .15]);
    setappdata(gui,'eUppDout',eUppDout);
    uicontrol(pTube,'Style','text',...
            'HorizontalAlignment','left',...
            'String','[mm] Outer Dimension',...
```

```
                'units','normalized',...
                'Position',[.7 .65 .3 .15]);
        eLowThick = uicontrol(pTube,'Style','edit',...
                'String','1.5',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.52 .515 .08 .15]);
        setappdata(gui,'eLowThick',eLowThick);
        eUppThick = uicontrol(pTube,'Style','edit',...
                'String','1.5',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.6 .515 .08 .15]);
        setappdata(gui,'eUppThick',eUppThick);
        uicontrol(pTube,'Style','text',...
                'HorizontalAlignment','left',...
                'String','[mm] Wall Thickness',...
                'units','normalized',...
                'Position',[.7 .485 .3 .15]);
        eLowSpace = uicontrol(pTube,'Style','edit',...
                'String','5',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.52 .35 .08 .15]);
        setappdata(gui,'eLowSpace',eLowSpace);
        eUppSpace = uicontrol(pTube,'Style','edit',...
                'String','5',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.6 .35 .08 .15]);
        setappdata(gui,'eUppSpace',eUppSpace);
        uicontrol(pTube,'Style','text',...
                'HorizontalAlignment','left',...
                'String','[mm] Double Tube Space',...
                'units','normalized',...
                'Position',[.7 .32 .3 .15]);
        eE = uicontrol(pTube,'Style','edit',...
                'String','50',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.54 .185 .12 .15]);
        setappdata(gui,'eE',eE);
        uicontrol(pTube,'Style','text',...
                'HorizontalAlignment','left',...
                'String','[GPa] Youngs Modulus',...
                'units','normalized',...
                'Position',[.7 .155 .3 .15]);
        eG = uicontrol(pTube,'Style','edit',...
                'String','0.5',...
                'Callback',@eTube_Callback,...
                'units','normalized',...
                'Position',[.54 .02 .12 .15]);
        setappdata(gui,'eG',eG);
        uicontrol(pTube,'Style','text',...
                'HorizontalAlignment','left',...
                'String','[GPa] Shear Modulus',...
                'units','normalized',...
                'Position',[.7 -.01 .3 .15]);

%% Panel - Angle Control
    pSpec = uipanel('Parent',gui,...
                'Title','Specific Angle Control',...
                'Position',[.01 .01 .39 .53]);
        chars2 = [char(hex2dec("398")) char(hex2dec("2082"))];
        uicontrol(pSpec,'Style','text',...
                'HorizontalAlignment','right',...
                'String',['Upper Arm Angle ' chars2 ' [deg] = '],...
                'units','normalized',...
                'Position',[.05 .93 .65 .05]);
        eTheta2 = uicontrol(pSpec,'Style','edit',...
                'String','0',...
                'units','normalized',...
                'Position',[.75 .935 .2 .05]);
        setappdata(gui,'eTheta2',eTheta2);
```

```
chars3 = [char(hex2dec("398")) char(hex2dec("2083"))];
uicontrol(pSpec,'Style','text',...
            'HorizontalAlignment','right',...
            'String',['Lower Arm Angle ' chars3 ' [deg] = '],...
            'units','normalized',...
            'Position',[.05 .81 .65 .05]);
eTheta3 = uicontrol(pSpec,'Style','edit',...
            'String','0',...
            'units','normalized',...
            'Position',[.75 .815 .2 .05]);
setappdata(gui,'eTheta3',eTheta3);
slTheta2 = uicontrol('Parent',pSpec,'Style','slider',...
            'Min',0,'Max',90,'Value',0,...
            'SliderStep',[1/90 1/9],...
            'units','normalized',...
            'Position',[.05 .875 .9 .05],...
            'callback',{@slTheta2_Callback,gui});
setappdata(gui,'slTheta2',slTheta2);
slTheta3 = uicontrol('Parent',pSpec,'Style','slider',...
            'Min',0,'Max',90,'Value',0,...
            'SliderStep',[1/90 1/9],...
            'units','normalized',...
            'Position',[.05 .755 .9 .05],...
            'callback',{@slTheta3_Callback,gui});
setappdata(gui,'slTheta3',slTheta3);
% visual arm representation
axArm = axes('Parent',pSpec,...
            'View',[-25 25],...
            'DataAspectRatio',[1 1 1],...
            'XGrid','on','YGrid','on',...
            'XLim',[-1.2 0.1],...
            'YLim',[-.55 0.55],...
            'ZLim',[-.3 0.3],...
            'XAxisLocation','origin',...
            'YAxisLocation','origin',...
            'OuterPosition',[0 0 1 .7]);  % [660 38 232 228]
axArm.Title.String = 'Arm Position';
hold on
axArmUpp = plot([0 -0.5],[0 0],'-ro',...
                'LineWidth',7,...
                'MarkerSize',7,...
                'MarkerFaceColor','r');
setappdata(gui,'axArmUpp',axArmUpp);
axArmLow = plot([-0.5 -0.95],[0 0],'-bo',...
                'LineWidth',4,...
                'MarkerSize',4,...
                'MarkerFaceColor','b');
setappdata(gui,'axArmLow',axArmLow);
% basic plane matrices
s.XY = cat(3,zeros(3)+[-1 0 1],zeros(3)+[-1 0 1]',zeros(3))*.1;
s.YZ = cat(3,zeros(3),zeros(3)+[-1 0 1],zeros(3)+[-1 0 1]')*.1;
s.ZX = cat(3,zeros(3)+[-1 0 1],zeros(3),zeros(3)+[-1 0 1]')*.1;
setappdata(gui,'sXY',s.XY);
setappdata(gui,'sYZ',s.YZ);
setappdata(gui,'sZX',s.ZX);
% null planes
axArmXY = surf(s.XY(:,:,1)*20,s.XY(:,:,2)*20,s.XY(:,:,3)*20,...
            'EdgeAlpha',0.3,'FaceAlpha',0.1);
axArmYZ = surf(s.YZ(:,:,1)*20,s.YZ(:,:,2)*20,s.YZ(:,:,3)*20,...
            'EdgeAlpha',0.3,'FaceAlpha',0.1);
axArmZX = surf(s.ZX(:,:,1)*20,s.ZX(:,:,2)*20,s.ZX(:,:,3)*20,...
            'EdgeAlpha',0.3,'FaceAlpha',0.1);
setappdata(gui,'axArmXY',axArmXY);
setappdata(gui,'axArmYZ',axArmYZ);
setappdata(gui,'axArmZX',axArmZX);
% mini plane positions
s.XY_ = s.XY + cat(3,-.5,0,0);
s.YZ_ = s.YZ + cat(3,-.5,0,0);
s.ZX_ = s.ZX + cat(3,-.5,0,0);
setappdata(gui,'sXY_',s.XY_);
setappdata(gui,'sYZ_',s.YZ_);
setappdata(gui,'sZX_',s.ZX_);
% plot mini planes
axArmXY = surf(s.XY_(:,:,1),s.XY_(:,:,2),s.XY_(:,:,3),...
```

```
                    'EdgeAlpha',0.4,'FaceAlpha',0.3);
        axArmYZ = surf(s.YZ_(:,:,1),s.YZ_(:,:,2),s.YZ_(:,:,3),...
                    'EdgeAlpha',0.4,'FaceAlpha',0.3);
        axArmZX = surf(s.ZX_(:,:,1),s.ZX_(:,:,2),s.ZX_(:,:,3),...
                    'EdgeAlpha',0.4,'FaceAlpha',0.3);
        setappdata(gui,'axArmXY',axArmXY);
        setappdata(gui,'axArmYZ',axArmYZ);
        setappdata(gui,'axArmZX',axArmZX);

%% Panel - Force and Moment Results
    pRes = uipanel('Parent',gui,...
                'Title','Resultant Forces at Elbow in Upper Arm Coordinates',...
                'units','normalized',...
                'Position',[.41 .01 .2 .24]);
    pR1x = 0;   pR1w = .69; pR2x = .7;  pR2w = .29; pRh = .15;
        uicontrol(pRes,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Horizontal Bending Force [N] = ',...
                    'units','normalized',...
                    'Position',[pR1x .77 pR1w pRh]);
        tFBendHor = uicontrol(pRes,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .79 pR2w pRh]);
        setappdata(gui,'tFBendHor',tFBendHor);
        uicontrol(pRes,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Vertical Bending Force [N] = ',...
                    'units','normalized',...
                    'Position',[pR1x .58 pR1w pRh]);
        tFBendVer = uicontrol(pRes,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .6 pR2w pRh]);
        setappdata(gui,'tFBendVer',tFBendVer);
        uicontrol(pRes,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Vertical Bending Moment [Nm] = ',...
                    'units','normalized',...
                    'Position',[pR1x .39 pR1w pRh]);
        tMBendVer = uicontrol(pRes,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .41 pR2w pRh]);
        setappdata(gui,'tMBendVer',tMBendVer);
        uicontrol(pRes,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Compression Force [N] = ',...
                    'units','normalized',...
                    'Position',[pR1x .2 pR1w pRh]);
        tFComp = uicontrol(pRes,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .22 pR2w pRh]);
        setappdata(gui,'tFComp',tFComp);
        uicontrol(pRes,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Torque [Nm] = ',...
                    'units','normalized',...
                    'Position',[pR1x .01 pR1w pRh]);
        tTorq = uicontrol(pRes,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .03 pR2w pRh]);
```

```
        setappdata(gui,'tTorq',tTorq);

%% Panel - Deflection Results
    pDefl = uipanel('Parent',gui,...
                'Title','Resultant Deflections',...
                'units','normalized',...
                'Position',[.62 .01 .37 .24]);
    pR1x = 0;   pR1w = .35; pR2x = .36;pR2w = .15;  pRh = .155;
        axes('Parent',pDefl,'Visible','off',...
                    'Position',[(pR2x+0*(pR2w+.01)) .835 pR2w pRh]);
            text('String','d_{total} [mm]','FontSize',8,...
                        'Position',[.15 .6]);
        axes('Parent',pDefl,'Visible','off',...
                    'Position',[(pR2x+1*(pR2w+.01)) .835 pR2w pRh]);
            text('String','d_{force} [mm]','FontSize',8,...
                        'Position',[.13 .6]);
        axes('Parent',pDefl,'Visible','off',...
                    'Position',[(pR2x+2*(pR2w+.01)) .835 pR2w pRh]);
            text('String','d_{moment} [mm]','FontSize',8,...
                        'Position',[.06 .6]);
        axes('Parent',pDefl,'Visible','off',...
                    'Position',[(pR2x+3*(pR2w+.01)) .835 pR2w pRh]);
            text('String','d_{torque} [mm]','FontSize',8,...
                        'Position',[.10 .6]);

        uicontrol(pDefl,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Vertical Wrist Deflection = ',...
                    'units','normalized',...
                    'Position',[pR1x .67 pR1w pRh]);
        tdLowVtot = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .69 pR2w pRh]);
        setappdata(gui,'tdLowVtot',tdLowVtot);
        tdLowVF = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'BackgroundColor',[1 .75 0],...
                    'units','normalized',...
                    'Position',[(pR2x+1*(pR2w+.01)) .69 pR2w pRh]);
        setappdata(gui,'tdLowVF',tdLowVF);
        tdLowVT = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'BackgroundColor',[1 .75 0],...
                    'units','normalized',...
                    'Position',[(pR2x+3*(pR2w+.01)) .69 pR2w pRh]);
        setappdata(gui,'tdLowVT',tdLowVT);

        uicontrol(pDefl,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Horizontal Wrist Deflection = ',...
                    'units','normalized',...
                    'Position',[pR1x .505 pR1w pRh]);
        tdLowHtot = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .525 pR2w pRh]);
        setappdata(gui,'tdLowHtot',tdLowHtot);
        tdLowHF = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'BackgroundColor',[1 .75 0],...
                    'units','normalized',...
                    'Position',[(pR2x+1*(pR2w+.01)) .525 pR2w pRh]);
        setappdata(gui,'tdLowHF',tdLowHF);

        uicontrol(pDefl,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Vertical Elbow Deflection = ',...
                    'units','normalized',...
                    'Position',[pR1x .34 pR1w pRh]);
```

```
        tdUppVtot = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .36 pR2w pRh]);
        setappdata(gui,'tdUppVtot',tdUppVtot);
        tdUppVF = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'BackgroundColor',[1 .75 0],...
                    'units','normalized',...
                    'Position',[(pR2x+1*(pR2w+.01)) .36 pR2w pRh]);
        setappdata(gui,'tdUppVF',tdUppVF);
        tdUppVM = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'BackgroundColor',[1 .75 0],...
                    'units','normalized',...
                    'Position',[(pR2x+2*(pR2w+.01)) .36 pR2w pRh]);
        setappdata(gui,'tdUppVM',tdUppVM);

        uicontrol(pDefl,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Horizontal Elbow Deflection = ',...
                    'units','normalized',...
                    'Position',[pR1x .175 pR1w pRh]);
        tdUppHtot = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .195 pR2w pRh]);
        setappdata(gui,'tdUppHtot',tdUppHtot);
        tdUppHF = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'BackgroundColor',[1 .75 0],...
                    'units','normalized',...
                    'Position',[(pR2x+1*(pR2w+.01)) .195 pR2w pRh]);
        setappdata(gui,'tdUppHF',tdUppHF);

        uicontrol(pDefl,'Style','text',...
                    'HorizontalAlignment','right',...
                    'String','Upper Arm Compression [mm] = ',...
                    'units','normalized',...
                    'Position',[pR1x .01 pR1w pRh]);
        tdUppComp = uicontrol(pDefl,'Style','text',...
                    'FontSize',10,...
                    'FontWeight','bold',...
                    'BackgroundColor',[1 .5 0],...
                    'units','normalized',...
                    'Position',[pR2x .03 pR2w pRh]);
        setappdata(gui,'tdUppComp',tdUppComp);

%% Main
% Make the UI visible and run initial functions
    gui.Visible = 'on';
    init = true(1);
    [data.t2mat,data.t3mat] = meshgrid(0:90);
    [spec] = getInput();
    [data] = FCalc(spec,data);
    [mySurf] = updateFSurfPlots(init,gui,data);
    [point] = updatePoints(init,gui,spec,data);
    s = updateArm(gui,spec,s);
    [data] = updateResults(gui,spec,data);
    [spec] = updateDeflection(gui,spec,data);
    [data] = dCalc(spec,data);
    [mySurf] = updatedSurfPlots(init,gui,data,mySurf);
    updateMaxDeflection(gui,data);
    eTubeSurf_Callback();
    init = false(1);

%% Functions - Callbacks
    % Dimension edit box callback
    function [] = eDim_Callback(~,~)
        set(pbFCalc,'BackgroundColor',[1 0.2 0.2])
```

```
    set(pbdCalc,'BackgroundColor',[1 0.2 0.2])
end

% Tube edit box callback
function [] = eTube_Callback(~,~)
    set(pbdCalc,'BackgroundColor',[1 0.2 0.2])
end

% theta2 slider Callback
function [] = slTheta2_Callback(varargin)
    [spec] = getInput();
    % display current angle
    set(eTheta2,'String',round(slTheta2.Value))
    % update angle variable and index
    spec.t2 = round(slTheta2.Value);
    spec.t2i = spec.t2 + 1;
    % update arm position and plot new points
    s = updateArm(gui,spec,s);
    [point] = updatePoints(init,gui,spec,data,point);
    [data] = updateResults(gui,spec,data);
    [spec] = updateDeflection(gui,spec,data);
end

% theta3 slider Callback
function [] = slTheta3_Callback(varargin)
    [spec] = getInput();
    % display current angle
    set(eTheta3,'String',round(slTheta3.Value))
    % update angle variable and index
    spec.t3 = round(slTheta3.Value);
    spec.t3i = spec.t3 + 1;
    % update arm position and plot new points
    s = updateArm(gui,spec,s);
    [point] = updatePoints(init,gui,spec,data,point);
    [data] = updateResults(gui,spec,data);
    [spec] = updateDeflection(gui,spec,data);
end

% global calc button callback
function [] = pbCalc_Callback(source,~,varargin)
    % get user input variables and recalculate the forces and moments
    [spec] = getInput();
    [data] = FCalc(spec,data);
    % make new surf plots
    [mySurf] = updateFSurfPlots(init,gui,data,mySurf);
    s = updateArm(gui,spec,s);
    [point] = updatePoints(init,gui,spec,data,point);
    [data] = updateResults(gui,spec,data);
    [spec] = updateDeflection(gui,spec,data);
    set(source,'BackgroundColor',[.94 .94 .94])
end

% deflection calculation button callback
function [] = pbDeflCalc_Callback(source,~,varargin)
    [spec] = getInput();
    [data] = dCalc(spec,data);
    [mySurf] = updatedSurfPlots(init,gui,data,mySurf);
    eTubeSurf_Callback();
    [spec] = updateDeflection(gui,spec,data);
    updateMaxDeflection(gui,data);
    set(source,'BackgroundColor',[.94 .94 .94])
end

% Deflection surf plot button group selection callback
function [] = eTubeSurf_Callback(~,~)

    if ~init
        % remove previous plotted points
        uAlpha = 0.3;
        oldTubeSurf = getappdata(gui,'selectedTubeSurf');
        switch oldTubeSurf
            case 'TubeSurfCy_1'
                set(mySurf.dCy1LowV,'FaceAlpha',uAlpha);
                set(mySurf.dCy1UppV,'FaceAlpha',uAlpha);
```

```
                    set(mySurf.dCy1UppH,'FaceAlpha',uAlpha);
                case 'TubeSurfCy_2h'
                    set(mySurf.dCy2hLowV,'FaceAlpha',uAlpha);
                    set(mySurf.dCy2hUppV,'FaceAlpha',uAlpha);
                    set(mySurf.dCy2hUppH,'FaceAlpha',uAlpha);
                case 'TubeSurfCy_2v'
                    set(mySurf.dCy2vLowV,'FaceAlpha',uAlpha);
                    set(mySurf.dCy2vUppV,'FaceAlpha',uAlpha);
                    set(mySurf.dCy2vUppH,'FaceAlpha',uAlpha);
                case 'TubeSurfSq_1'
                    set(mySurf.dSq1LowV,'FaceAlpha',uAlpha);
                    set(mySurf.dSq1UppV,'FaceAlpha',uAlpha);
                    set(mySurf.dSq1UppH,'FaceAlpha',uAlpha);
                case 'TubeSurfSq_2h'
                    set(mySurf.dSq2hLowV,'FaceAlpha',uAlpha);
                    set(mySurf.dSq2hUppV,'FaceAlpha',uAlpha);
                    set(mySurf.dSq2hUppH,'FaceAlpha',uAlpha);
                case 'TubeSurfSq_2v'
                    set(mySurf.dSq2vLowV,'FaceAlpha',uAlpha);
                    set(mySurf.dSq2vUppV,'FaceAlpha',uAlpha);
                    set(mySurf.dSq2vUppH,'FaceAlpha',uAlpha);
            end
        end

        fAlpha = 0.8;
        newTubeSurf = pTubeSurf.SelectedObject.Tag;
        switch newTubeSurf
            case 'TubeSurfCy_1'
                set(mySurf.dCy1LowV,'FaceAlpha',fAlpha);
                set(mySurf.dCy1UppV,'FaceAlpha',fAlpha);
                set(mySurf.dCy1UppH,'FaceAlpha',fAlpha);
            case 'TubeSurfCy_2h'
                set(mySurf.dCy2hLowV,'FaceAlpha',fAlpha);
                set(mySurf.dCy2hUppV,'FaceAlpha',fAlpha);
                set(mySurf.dCy2hUppH,'FaceAlpha',fAlpha);
            case 'TubeSurfCy_2v'
                set(mySurf.dCy2vLowV,'FaceAlpha',fAlpha);
                set(mySurf.dCy2vUppV,'FaceAlpha',fAlpha);
                set(mySurf.dCy2vUppH,'FaceAlpha',fAlpha);
            case 'TubeSurfSq_1'
                set(mySurf.dSq1LowV,'FaceAlpha',fAlpha);
                set(mySurf.dSq1UppV,'FaceAlpha',fAlpha);
                set(mySurf.dSq1UppH,'FaceAlpha',fAlpha);
            case 'TubeSurfSq_2h'
                set(mySurf.dSq2hLowV,'FaceAlpha',fAlpha);
                set(mySurf.dSq2hUppV,'FaceAlpha',fAlpha);
                set(mySurf.dSq2hUppH,'FaceAlpha',fAlpha);
            case 'TubeSurfSq_2v'
                set(mySurf.dSq2vLowV,'FaceAlpha',fAlpha);
                set(mySurf.dSq2vUppV,'FaceAlpha',fAlpha);
                set(mySurf.dSq2vUppH,'FaceAlpha',fAlpha);
        end

        setappdata(gui,'selectedTubeSurf',newTubeSurf)
    end

%% Get information from gui

    function [spec] = getInput()
        % dimension and force inputs
        spec.FHor       = str2double(get(eFHor,'String'));
        spec.FVer       = str2double(get(eFVer,'String'));
        spec.rPulley    = str2double(get(ePul,'String'));
        spec.Lowerarm   = str2double(get(eLow,'String'));
        spec.Upperarm   = str2double(get(eUpp,'String'));
        spec.Lmoment    = str2double(get(eArm,'String'));

        % angle inputs
        spec.t2         = round(slTheta2.Value);
        spec.t3         = round(slTheta3.Value);
        spec.t2i = spec.t2 + 1;
        spec.t3i = spec.t3 + 1;

        % tube dimensions
```

```
        spec.Low.Do     = str2double(get(eLowDout,'String'))/1000;
        spec.Low.t      = str2double(get(eLowThick,'String'))/1000;
        spec.Low.S      = str2double(get(eLowSpace,'String'))/1000;
        spec.Low.Di     = spec.Low.Do - 2*spec.Low.t;
        spec.Low.A.Cy1 = (pi*(spec.Low.Do^2 - spec.Low.Di^2))/4;
        spec.Low.A.Sq1 = spec.Low.Do^2 - spec.Low.Di^2;

        spec.Upp.Do     = str2double(get(eUppDout,'String'))/1000;
        spec.Upp.t      = str2double(get(eUppThick,'String'))/1000;
        spec.Upp.S      = str2double(get(eUppSpace,'String'))/1000;
        spec.Upp.Di     = spec.Upp.Do - 2*spec.Upp.t;
        spec.Upp.A.Cy1 = (pi*(spec.Upp.Do^2 - spec.Upp.Di^2))/4;
        spec.Upp.A.Cy2h = 2*spec.Upp.A.Cy1;
        spec.Upp.A.Cy2v = spec.Upp.A.Cy2h;
        spec.Upp.A.Sq1 = spec.Upp.Do^2 - spec.Upp.Di^2;
        spec.Upp.A.Sq2h = 2*spec.Upp.A.Sq1;
        spec.Upp.A.Sq2v = spec.Upp.A.Sq2h;

        % material properties
        spec.E          = str2double(get(eE,'String'))*10^9;
        spec.G          = str2double(get(eG,'String'))*10^9;
    end

end
```

A.1. calcIJ.m

```
function [I,J,spec] = calcIJ(spec)
% Calculate all second planar moments of area and inertia. The input is the
% specifications structure and the outputs are the second planar moment of
% area structure, second planar moment of area structure, and appended
% specifications structure.
%
%   [I,J,spec] = calcIJ(spec)
%
% The results arse saved in a structure, the naming of these variables is
% explained below.
%   The variables are calculated and in the same line they are stored
%   in a structure. The codes for the variables within the function and
%   the structure are similar. Below a description of the code is given
%   where the first and second columns are the character code in the
%   function and structure respectively (some character codes are the
%   same for both):
%       First character
%           d   = dim   =       dimension of tube
%           I   = I     = [m^4] second planar moment of area
%           J   = J     = [m^4] polar moment of inertia
%       Second character
%           L   = Low   = lower arm
%           U   = Upp   = upper arm
%       Third character or set of characters
%           Do  = Dout  = [m]   outer dimension/diameter
%           t   = Thick = [m]   tube wall thickness
%           Di  = Din   = [m]   inner dimension/diameter
%           S   = Space = [m]   space between tubes
%           A           = [m^2] segment cross sectional area
%           c           = [m]   distance neutral axes of tube and arm
%               Cy..    =       round tube
%               Sq..    =       square tube
%               ..1     =       one tube
%               ..2v    =       two tubes in vertical configuration
%               ..2h    =       two tubes in horizontal configuration
%       Final character (only examine if ending in vv, hh, vh, or hv)
%           v = Ver     =       vertical component
%           h = Hor     =       horizontal component

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020
```

```
    % lower arm
    dLDo    = spec.Low.Do;
    dLS     = spec.Low.S;
    dLDi    = spec.Low.Di;
    dLCyA   = spec.Low.A.Cy1;
    dLSqA   = spec.Low.A.Sq1;

    ILOneCy = (pi*(dLDo^4 - dLDi^4))/64;       I.Low.OneCy     = ILOneCy;
    ILOneSq = (dLDo^4 - dLDi^4)/12;            I.Low.OneSq     = ILOneSq;

    % lower arm round tube 1
    ILCy1v  = ILOneCy;                         I.Low.Cy1.Ver   = ILCy1v;
    ILCy1h  = ILOneCy;                         I.Low.Cy1.Hor   = ILCy1h;
    % lower arm round tube 2 horizontal
    ILCy2hv = 2*ILOneCy;                       I.Low.Cy2h.Ver  = ILCy2hv;
    dLCy2hc = dLDo/2 + dLS/2;                  spec.Low.c_Cy2h = dLCy2hc;
    ILCy2hh = 2*(ILOneCy + dLCyA*dLCy2hc^2);   I.Low.Cy2h.Hor  = ILCy2hh;
    % lower arm round tube 2 vertical
    dLCy2vc = dLDo/2 + dLS/2;                  spec.Low.c_Cy2v = dLCy2vc;
    ILCy2vv = 2*(ILOneCy + dLCyA*dLCy2vc^2);   I.Low.Cy2v.Ver  = ILCy2vv;
    ILCy2vh = 2*ILOneCy;                       I.Low.Cy2v.Hor  = ILCy2vh;
    % lower arm square tube 1
    ILSq1v  = ILOneSq;                         I.Low.Sq1.Ver   = ILSq1v;
    ILSq1h  = ILOneSq;                         I.Low.Sq1.Hor   = ILSq1h;
    % lower arm square tube 2 horizontal
    ILSq2hv = 2*ILOneSq;                       I.Low.Sq2h.Ver  = ILSq2hv;
    dLSq2hc = dLDo/2 + dLS/2;                  spec.Low.c_Sq2h = dLSq2hc;
    ILSq2hh = 2*(ILOneSq + dLSqA*dLSq2hc^2);   I.Low.Sq2h.Hor  = ILSq2hh;
    % lower arm square tube 2 vertical
    dLSq2vc = dLDo/2 + dLS/2;                  spec.Low.c_Sq2v = dLSq2vc;
    ILSq2vv = 2*(ILOneSq + dLSqA*dLSq2vc^2);   I.Low.Sq2v.Ver  = ILSq2vv;
    ILSq2vh = 2*ILOneSq;                       I.Low.Sq2v.Hor  = ILSq2vh;

    % upper arm
    dUDo    = spec.Upp.Do;
    dUS     = spec.Upp.S;
    dUDi    = spec.Upp.Di;
    dUCyA   = spec.Upp.A.Cy1;
    dUSqA   = spec.Upp.A.Sq1;

    IUOneCy = (pi*(dUDo^4 - dUDi^4))/64;        I.Upp.OneCy       = IUOneCy;
    IUOneSq = (dUDo^4 - dUDi^4)/12;             I.Upp.OneSq       = IUOneSq;

    % upper arm round tube 1
    IUCy1v  = IUOneCy;                         I.Upp.Cy1.Ver   = IUCy1v;
    IUCy1h  = IUOneCy;                         I.Upp.Cy1.Hor   = IUCy1h;
    JUCy1   = (pi*(dUDo^4 - dUDi^4))/32;       J.Upp.Cy1       = JUCy1;
    % upper arm round tube 2 horizontal
    IUCy2hv = 2*IUOneCy;                       I.Upp.Cy2h.Ver  = IUCy2hv;
    dUCy2hc = dUDo/2 + dUS/2;                  spec.Upp.c_Cy2h = dUCy2hc;
    IUCy2hh = 2*(IUOneCy + dUCyA*dUCy2hc^2);   I.Upp.Cy2h.Hor  = IUCy2hh;
    JUCy2h  = IUCy2hv + IUCy2hh;               J.Upp.Cy2h      = JUCy2h;
    % upper arm round tube 2 vertical
    dUCy2vc = dUDo/2 + dUS/2;                  spec.Upp.c_Cy2v = dUCy2vc;
    IUCy2vv = 2*(IUOneCy + dUCyA*dUCy2vc^2);   I.Upp.Cy2v.Ver  = IUCy2vv;
    IUCy2vh = 2*IUOneCy;                       I.Upp.Cy2v.Hor  = IUCy2vh;
    JUCy2v  = IUCy2vv + IUCy2vh;               J.Upp.Cy2v      = JUCy2v;
    % upper arm square tube 1
    IUSq1v  = IUOneSq;                         I.Upp.Sq1.Ver   = IUSq1v;
    IUSq1h  = IUOneSq;                         I.Upp.Sq1.Hor   = IUSq1h;
    JUSq1   = 2*IUOneSq;                       J.Upp.Sq1       = JUSq1;
    % upper arm square tube 2 horizontal
    IUSq2hv = 2*IUOneSq;                       I.Upp.Sq2h.Ver  = IUSq2hv;
    dUSq2hc = dUDo/2 + dUS/2;                  spec.Upp.c_Sq2h = dUSq2hc;
    IUSq2hh = 2*(IUOneSq + dUSqA*dUSq2hc^2);   I.Upp.Sq2h.Hor  = IUSq2hh;
    JUSq2h  = IUSq2hv + IUSq2hh;               J.Upp.Sq2h      = JUSq2h;
    % upper arm square tube 2 vertical
    dUSq2vc = dUDo/2 + dUS/2;                  spec.Upp.c_Sq2v = dUSq2vc;
    IUSq2vv = 2*(IUOneSq + dUSqA*dUSq2vc^2);   I.Upp.Sq2v.Ver  = IUSq2vv;
    IUSq2vh = 2*IUOneSq;                       I.Upp.Sq2v.Hor  = IUSq2vh;
    JUSq2v  = IUSq2vv + IUSq2vh;               J.Upp.Sq2v      = JUSq2v;
end
```

A.2. dCalc.m

```
function [data] = dCalc(spec,data)
% Calculate all deflections at all given angle configurations. Input is the
% specifications structure and data structure, the output is the data
% structure with appended new deflection data.
%
%   [data] = dCalc(spec,data)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    % calculate all I's and J's
    [I,J,spec] = calcIJ(spec);

    for i = ["Cy1" "Cy2h" "Cy2v" "Sq1" "Sq2h" "Sq2v"]
        % lower arm vertical
        eval(strcat(...
            'data.d.',i,'.Low.F.Ver = dFeq(spec.FVer,spec.Lowerarm,I.Low.',i,'.Ver,spec.E);',...
            'data.d.',i,'.Upp.T.phi = phiTeq(data.TorqMat,spec.Upperarm,J.Upp.',i,',spec.G);',...
            'data.d.',i,'.Low.T.Ver = spec.Lowerarm * sin(data.d.',i,'.Upp.T.phi);',...
            'data.d.',i,'.Low.tot.Ver = data.d.',i,'.Low.F.Ver + data.d.',i,'.Low.T.Ver;'))

        % lower arm horizontal
        eval(strcat(...
            'data.d.',i,'.Low.tot.Hor = dFeq(spec.FHor,spec.Lowerarm,I.Low.',i,'.Hor,spec.E);'))

        % upper arm vertical
        eval(strcat(...
            'data.d.',i,'.Upp.F.Ver = dFeq(data.FUppV,spec.Upperarm,I.Upp.',i,'.Ver,spec.E);',...
            'data.d.',i,'.Upp.M.Ver = dMeq(data.MUppVMat,spec.Upperarm,I.Upp.',i,'.Ver,spec.E);',...
            'data.d.',i,'.Upp.tot.Ver = data.d.',i,'.Upp.F.Ver + data.d.',i,'.Upp.M.Ver;'))

        % upper arm horizontal
        eval(strcat(...
            'data.d.',i,'.Upp.tot.Hor = dFeq(data.FUppHMat,spec.Upperarm,I.Upp.',i,'.Hor,spec.E);'))

        % upper arm compression
        eval(strcat(...
            'data.d.',i,'.Upp.Comp = (data.FCompMat*spec.Upperarm)/(spec.E*spec.Upp.A.',i,');'));
%         dUppComp = (FUppComp*L.Upperarm)/(E*AUpp);
    end
end
```

A.3. dFeq.m

```
function [d] = dFeq(V,L,I,E)
% Calculate deflection from cantilever force. The inputs are the bending
% force, length of beam, second mass moment of area, and Young's modulus.
% The output is the deflection.
%
%   [d] = dFeq(V,L,I,E)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    d = (V*(L)^3)/(3*E*I);
end
```

A.4. dFunc.m

```
function [dLowV,dLowH,dUppV,dUppH,dUppComp,d,spec] = dFunc(gui,spec,data)
% Calculate all deflections in the current arm configuration. The inputs
% are the handle for the GUI, the specificaitons structure, and data
```

```
% structure. The outputs are the total defelctions relative to the
% segment's coordinate system for lower arm vertical, lower arm horizontal,
% upper arm verticcal, upper arm horizontal, upper arm compression, and a
% deflection structure with all sub-deflections along with the updated
% specifications structure.
%
%   [dLowV,dLowH,dUppV,dUppH,dUppComp,d,spec] = dFunc(gui,spec,data)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020
    [data] = FFunc(spec,data);
    [ILowVer,ILowHor,IUppVer,IUppHor,JUpp,AUpp,spec] = getIJ(gui,spec);

    dLowFv = dFeq(spec.FVer,spec.Lowerarm,ILowVer,spec.E);
    dLowMv = 0;
    UppTphi = phiTeq(data.TUpp,spec.Upperarm,JUpp,spec.G);
    d.UppTphi = UppTphi;
    dLowTv = spec.Lowerarm * sin(UppTphi);
    dLowV = dLowFv + dLowMv + dLowTv;
    d.LowFv = dLowFv;
    d.LowMv = dLowMv;
    d.LowTv = dLowTv;

    dLowFh = dFeq(spec.FHor,spec.Lowerarm,ILowHor,spec.E);
    dLowMh = 0;
    dLowH = dLowFh + dLowMh;
    d.LowFh = dLowFh;
    d.LowMh = dLowMh;

    dUppFv = dFeq(data.FUppV,spec.Upperarm,IUppVer,spec.E);
    dUppMv = dMeq(data.MUppV,spec.Upperarm,IUppVer,spec.E);
    dUppV = dUppFv + dUppMv;
    d.UppFv = dUppFv;
    d.UppMv = dUppMv;

    dUppFh = dFeq(data.FUppH,spec.Upperarm,IUppHor,spec.E);
    dUppMh = 0;
    dUppH = dUppFh + dUppMh;
    d.UppFh = dUppFh;
    d.UppMh = dUppMh;

    dUppComp = (data.FUppComp*spec.Upperarm)/(spec.E*AUpp);
end
```

A.5. dMeq.m

```
function [d] = dMeq(M,L,I,E)
% Calculate deflection from cantilever moment. The inputs are the bending
% moment, length of beam, second mass moment of area, and Young's modulus.
% The output is the deflection.
%
%   [d] = dMeq(M,L,I,E)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    d = -(M*(L)^2)/(2*E*I);
end
```

A.6. FCalc.m

```
function [data] = FCalc(spec,data)
% Calculate all forces and moments at all given angle configurations. The
% inputs are is the specifications structure and data structure, the output
% is the data structure with appended new deflection data.
```

```
%
%   [data] = FCalc(spec,data)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    % convert angles to radians
    t2matrad = data.t2mat*pi/180;
    t3matrad = data.t3mat*pi/180;

    % calculate forces that cause bending in horizontal plane
    data.Ft34 = (spec.FHor*spec.Lowerarm)/spec.rPulley;
    data.Fd3x = (data.Ft34*spec.rPulley)./(spec.Lmoment*sin(t3matrad));
    data.Fb4x = data.Ft34*cos(t2matrad)-spec.FHor*sin(t3matrad);
    data.Fb4y = data.Ft34*sin(t2matrad)-spec.FHor*cos(t3matrad);
    data.Fa3x = data.Ft34*cos(t2matrad)-data.Fd3x;
    data.Fa3y = data.Ft34*sin(t2matrad);
    data.Fb2x = data.Fb4x;
    data.Fb2y = data.Fb4y;
    data.Fc2x = (data.Fb2x.*spec.Upperarm.*sin(t2matrad)-...
                 data.Fb2y.*spec.Upperarm.*cos(t2matrad))./(spec.Lmoment*cos(t2matrad));
    data.Fa2x = data.Fb2x + data.Fc2x;
    data.Fa2y = data.Fb2y;

    % distance from elbow to end effector
    data.r(:,:,1) = -spec.Lowerarm*cos(t3matrad);
    data.r(:,:,2) = spec.Lowerarm*sin(t3matrad);
    data.r(:,:,3) = 0;

    % calculate forces moments that cause torsion and vertical bending
    spec.FVerMat = zeros(size(data.r)) + cat(3,0,0,spec.FVer); % vertical forces in 3d
    data.Mb3 = cross(spec.FVerMat,data.r); % 3d moments in world coordinates
    for j = 1:length(t2matrad)
        Rz = rotz(t2matrad(1,j)*180/pi); % rotation matrix to get to elbow coordinates
        for i = 1:length(t3matrad)
            data.Mb3_(i,j,:) = reshape(data.Mb3(i,j,:),[1,3])*Rz; % moments in elbow coordinates
        end
    end

    % main outputs
    data.FUppHMat = data.Fb2x.*sin(t2matrad)-data.Fb2y.*cos(t2matrad);
    data.FCompMat = data.Fb2x.*cos(t2matrad)+data.Fb2y.*sin(t2matrad);
    data.MUppVMat = data.Mb3_(:,:,2);
    data.TorqMat = data.Mb3_(:,:,1);
end
```

A.7. FFunc.m

```
function [data] = FFunc(spec,data)
% Get the forces and moments for the current arm position from the already
% calculated matrices. The inputs are the specifications structure and data
% structure, the output is the data structure with appended new deflection
% data.
%
%   [data] = FCalc(spec,data)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    % extract corresponding values
    data.FUppH = data.FUppHMat(spec.t2i,spec.t3i);
    data.FUppComp = data.FCompMat(spec.t2i,spec.t3i);
    data.FUppV = spec.FVer;
    data.MUppV = data.MUppVMat(spec.t2i,spec.t3i);
    data.TUpp = data.TorqMat(spec.t2i,spec.t3i);

end
```

A.8. getIJ.m

```
function [ILowVer,ILowHor,IUppVer,IUppHor,JUpp,AUpp,spec] = getIJ(gui,spec)
% Calculate the second planar moments of area and inertia using the
% calcIJ() function and extract the second planar moments that are selected
% by the user in the GUI. The inputs are the handle for the GUI and the
% specifications structure. The outputs are the second planar moment of
% area (I) for the lower arm in the vertical direction, I lower arm
% horizontal, I upper arm vertical, I upper arm horizontal, second planar
% moment of inertia (J) for the upper arm, crosssectional area structure,
% and the updated specifications structure.
%
%   [ILowVer,ILowHor,IUppVer,IUppHor,JUpp,AUpp,spec] = getIJ(gui,spec)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    % calculate all I's and J's
    [I,J,spec] = calcIJ(spec);

    % lower arm
    pTubeLow = getappdata(gui,'pTubeLow');
    TubeLow = pTubeLow.SelectedObject.Tag;
    switch TubeLow
        case 'TubeLowCy_1'
            ILowVer = I.Low.Cy1.Ver;
            ILowHor = I.Low.Cy1.Hor;
        case 'TubeLowCy_2h'
            ILowVer = I.Low.Cy2h.Ver;
            ILowHor = I.Low.Cy2h.Hor;
        case 'TubeLowCy_2v'
            ILowVer = I.Low.Cy2v.Ver;
            ILowHor = I.Low.Cy2v.Hor;
        case 'TubeLowSq_1'
            ILowVer = I.Low.Sq1.Ver;
            ILowHor = I.Low.Sq1.Hor;
        case 'TubeLowSq_2h'
            ILowVer = I.Low.Sq2h.Ver;
            ILowHor = I.Low.Sq2h.Hor;
        case 'TubeLowSq_2v'
            ILowVer = I.Low.Sq2v.Ver;
            ILowHor = I.Low.Sq2v.Hor;
    end

    % upper arm
    pTubeUpp = getappdata(gui,'pTubeUpp');
    TubeUpp = pTubeUpp.SelectedObject.Tag;
    switch TubeUpp
        case 'TubeUppCy_1'
            AUpp    = spec.Upp.A.Cy1;
            IUppVer = I.Upp.Cy1.Ver;
            IUppHor = I.Upp.Cy1.Hor;
            JUpp    = J.Upp.Cy1;
        case 'TubeUppCy_2h'
            AUpp    = 2*spec.Upp.A.Cy1;
            IUppVer = I.Upp.Cy2h.Ver;
            IUppHor = I.Upp.Cy2h.Hor;
            JUpp    = J.Upp.Cy2h;
        case 'TubeUppCy_2v'
            AUpp    = 2*spec.Upp.A.Cy1;
            IUppVer = I.Upp.Cy2v.Ver;
            IUppHor = I.Upp.Cy2v.Hor;
            JUpp    = J.Upp.Cy2v;
        case 'TubeUppSq_1'
            AUpp    = spec.Upp.A.Sq1;
            IUppVer = I.Upp.Sq1.Ver;
            IUppHor = I.Upp.Sq1.Hor;
            JUpp    = J.Upp.Sq1;
        case 'TubeUppSq_2h'
            AUpp    = 2*spec.Upp.A.Sq1;
            IUppVer = I.Upp.Sq2h.Ver;
```

```
            IUppHor = I.Upp.Sq2h.Hor;
            JUpp    = J.Upp.Sq2h;
        case 'TubeUppSq_2v'
            AUpp    = 2*spec.Upp.A.Sq1;
            IUppVer = I.Upp.Sq2v.Ver;
            IUppHor = I.Upp.Sq2v.Hor;
            JUpp    = J.Upp.Sq2v;
    end
end
```

A.9. phiTeq.m

```
function [phi] = phiTeq(T,L,J,G)
% deflection from cantilever torque
% Calculate the angular deflection from cantilever torque. The inputs are
% the torque, length of beam, second mass moment of inertia, and the shear
% modulus. The output is the angular deflection.
%
%   [phi] = phiTeq(T,L,J,G)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020
    phi = -(T*(L)^3)/(3*G*J);
end
```

A.10.updateArm.m

```
function [s] = updateArm(gui,spec,s)
% Update the 3D plot of that indicates the user's input arm position. The
% inputs are the handle for the GUI, the specifications structure, and the
% structure with the vectors that represent the arm position. The output is
% an updated structure with the representative arm vectors.
%
%   [s] = updateArm(gui,spec,s)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    % get angles and lengths
    t2rad = (pi/180)*round(spec.t2);
    t3rad = (pi/180)*round(spec.t3);
    % calculate joint positions
    xElbow = -spec.Upperarm*cos(t2rad);
    yElbow = spec.Upperarm*sin(t2rad);
    zElbow = 0;
    xWrist = xElbow - spec.Lowerarm*cos(t3rad);
    yWrist = yElbow - spec.Lowerarm*sin(t3rad);
    % rotation matrix around z axis
    Rz = rotz(spec.t2);
    % update arm plot

    axArmUpp = getappdata(gui,'axArmUpp');
    axArmLow = getappdata(gui,'axArmLow');

    set(axArmUpp,'xdata',[0 xElbow],...
            'ydata',[0 yElbow]);
    set(axArmLow,'xdata',[xElbow xWrist],...
            'ydata',[yElbow yWrist]);
    for i=1:3
        for j=1:3
            s.XY_(i,j,:) = reshape(s.XY(i,j,:),[1,3])*Rz;
            s.YZ_(i,j,:) = reshape(s.YZ(i,j,:),[1,3])*Rz;
            s.ZX_(i,j,:) = reshape(s.ZX(i,j,:),[1,3])*Rz;
        end
    end
```

```
    s.XY_ = s.XY_ + cat(3,xElbow,yElbow,zElbow);
    s.YZ_ = s.YZ_ + cat(3,xElbow,yElbow,zElbow);
    s.ZX_ = s.ZX_ + cat(3,xElbow,yElbow,zElbow);

    axArmXY = getappdata(gui,'axArmXY');
    axArmYZ = getappdata(gui,'axArmYZ');
    axArmZX = getappdata(gui,'axArmZX');

    set(axArmXY,'xdata',s.XY_(:,:,1),...
            'ydata',s.XY_(:,:,2),...
            'zdata',s.XY_(:,:,3));
    set(axArmYZ,'xdata',s.YZ_(:,:,1),...
            'ydata',s.YZ_(:,:,2),...
            'zdata',s.YZ_(:,:,3));
    set(axArmZX,'xdata',s.ZX_(:,:,1),...
            'ydata',s.ZX_(:,:,2),...
            'zdata',s.ZX_(:,:,3));
end
```

A.11.updateDeflection.m

```
function [spec] = updateDeflection(gui,spec,data)
% Update the deflection data shown in the results table by calculating the
% deflections and changing the relevant text fields. The inputs are the
% handle for the GUI, the specifications structure, and the data structure.
% The output is an updated specifications structure.
%
%   [spec] = updateDeflection(gui,spec,data)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    tdLowVtot = getappdata(gui,'tdLowVtot');
    tdLowVF = getappdata(gui,'tdLowVF');
    tdLowVT = getappdata(gui,'tdLowVT');

    tdLowHtot = getappdata(gui,'tdLowHtot');
    tdLowHF = getappdata(gui,'tdLowHF');

    tdUppVtot = getappdata(gui,'tdUppVtot');
    tdUppVF = getappdata(gui,'tdUppVF');
    tdUppVM = getappdata(gui,'tdUppVM');

    tdUppHtot = getappdata(gui,'tdUppHtot');
    tdUppHF = getappdata(gui,'tdUppHF');

    tdUppComp = getappdata(gui,'tdUppComp');

    % calculate and display deflections
    [dLowV,dLowH,dUppV,dUppH,dUppComp,d,spec] = dFunc(gui,spec,data);

    set(tdLowVtot,'String',round(dLowV*1000,4))
    set(tdLowVF,'String',round(d.LowFv*1000,4))
    set(tdLowVT,'String',round(d.LowTv*1000,4))

    set(tdLowHtot,'String',round(dLowH*1000,4))
    set(tdLowHF,'String',round(d.LowFh*1000,4))

    set(tdUppVtot,'String',round(dUppV*1000,4))
    set(tdUppVF,'String',round(d.UppFv*1000,4))
    set(tdUppVM,'String',round(d.UppMv*1000,4))

    set(tdUppHtot,'String',round(dUppH*1000,4))
    set(tdUppHF,'String',round(d.UppFh*1000,4))

    set(tdUppComp,'String',round(dUppComp*1000,4))
end
```

A.12.updatedSurfPlots.m

```
function [mySurf] = updatedSurfPlots(init,gui,data,mySurf)
% Update the deflection surf plots. The inputs are the initiation variable,
% the handle for the GUI, the data structure, and the handle for the surf
% plots. The output is the handle for the surf plots.
%
%   [mySurf] = updatedSurfPlots(init,gui,data,mySurf)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    axdLowVer = getappdata(gui,'axdLowVer');
    axdUppVer = getappdata(gui,'axdUppVer');
    axdUppHor = getappdata(gui,'axdUppHor');

    if ~init
        % delete previous surf plots
        delete(mySurf.dCy1LowV)
        delete(mySurf.dCy2hLowV)
        delete(mySurf.dCy2vLowV)
        delete(mySurf.dSq1LowV)
        delete(mySurf.dSq2hLowV)
        delete(mySurf.dSq2vLowV)

        delete(mySurf.dCy1UppV)
        delete(mySurf.dCy2hUppV)
        delete(mySurf.dCy2vUppV)
        delete(mySurf.dSq1UppV)
        delete(mySurf.dSq2hUppV)
        delete(mySurf.dSq2vUppV)

        delete(mySurf.dCy1UppH)
        delete(mySurf.dCy2hUppH)
        delete(mySurf.dCy2vUppH)
        delete(mySurf.dSq1UppH)
        delete(mySurf.dSq2hUppH)
        delete(mySurf.dSq2vUppH)
    end

    uAlpha = 0.3;
    % Lower arm vertical bending
    mySurf.dCy1LowV = surf(axdLowVer,data.t2mat,data.t3mat,...
                data.d.Cy1.Low.tot.Ver*1000,...
                'FaceColor','#0072BD',...
                'EdgeColor','#0072BD',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dCy2hLowV = surf(axdLowVer,data.t2mat,data.t3mat,...
                data.d.Cy2h.Low.tot.Ver*1000,...
                'FaceColor','#D95319',...
                'EdgeColor','#D95319',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dCy2vLowV = surf(axdLowVer,data.t2mat,data.t3mat,...
                data.d.Cy2v.Low.tot.Ver*1000,...
                'FaceColor','#EDB120',...
                'EdgeColor','#EDB120',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq1LowV = surf(axdLowVer,data.t2mat,data.t3mat,...
                data.d.Sq1.Low.tot.Ver*1000,...
                'FaceColor','#7E2F8E',...
                'EdgeColor','#7E2F8E',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq2hLowV = surf(axdLowVer,data.t2mat,data.t3mat,...
                data.d.Sq2h.Low.tot.Ver*1000,...
                'FaceColor','#77AC30',...
                'EdgeColor','#77AC30',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq2vLowV = surf(axdLowVer,data.t2mat,data.t3mat,...
                data.d.Sq2v.Low.tot.Ver*1000,...
                'FaceColor','#A2142F',...
                'EdgeColor','#A2142F',...
```

```
                'EdgeAlpha',0,'FaceAlpha',uAlpha);

    % Upper arm vertical bending
    mySurf.dCy1UppV = surf(axdUppVer,data.t2mat,data.t3mat,...
                data.d.Cy1.Upp.tot.Ver*1000,...
                'FaceColor','#0072BD',...
                'EdgeColor','#0072BD',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dCy2hUppV = surf(axdUppVer,data.t2mat,data.t3mat,...
                data.d.Cy2h.Upp.tot.Ver*1000,...
                'FaceColor','#D95319',...
                'EdgeColor','#D95319',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dCy2vUppV = surf(axdUppVer,data.t2mat,data.t3mat,...
                data.d.Cy2v.Upp.tot.Ver*1000,...
                'FaceColor','#EDB120',...
                'EdgeColor','#EDB120',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq1UppV = surf(axdUppVer,data.t2mat,data.t3mat,...
                data.d.Sq1.Upp.tot.Ver*1000,...
                'FaceColor','#7E2F8E',...
                'EdgeColor','#7E2F8E',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq2hUppV = surf(axdUppVer,data.t2mat,data.t3mat,...
                data.d.Sq2h.Upp.tot.Ver*1000,...
                'FaceColor','#77AC30',...
                'EdgeColor','#77AC30',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq2vUppV = surf(axdUppVer,data.t2mat,data.t3mat,...
                data.d.Sq2v.Upp.tot.Ver*1000,...
                'FaceColor','#A2142F',...
                'EdgeColor','#A2142F',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);

    % Upper arm horizontal bending
    mySurf.dCy1UppH = surf(axdUppHor,data.t2mat,data.t3mat,...
                data.d.Cy1.Upp.tot.Hor*1000,...
                'FaceColor','#0072BD',...
                'EdgeColor','#0072BD',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dCy2hUppH = surf(axdUppHor,data.t2mat,data.t3mat,...
                data.d.Cy2h.Upp.tot.Hor*1000,...
                'FaceColor','#D95319',...
                'EdgeColor','#D95319',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dCy2vUppH = surf(axdUppHor,data.t2mat,data.t3mat,...
                data.d.Cy2v.Upp.tot.Hor*1000,...
                'FaceColor','#EDB120',...
                'EdgeColor','#EDB120',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq1UppH = surf(axdUppHor,data.t2mat,data.t3mat,...
                data.d.Sq1.Upp.tot.Hor*1000,...
                'FaceColor','#7E2F8E',...
                'EdgeColor','#7E2F8E',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq2hUppH = surf(axdUppHor,data.t2mat,data.t3mat,...
                data.d.Sq2h.Upp.tot.Hor*1000,...
                'FaceColor','#77AC30',...
                'EdgeColor','#77AC30',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
    mySurf.dSq2vUppH = surf(axdUppHor,data.t2mat,data.t3mat,...
                data.d.Sq2v.Upp.tot.Hor*1000,...
                'FaceColor','#A2142F',...
                'EdgeColor','#A2142F',...
                'EdgeAlpha',0,'FaceAlpha',uAlpha);
end
```

A.13.updateFSurfPlots.m

```
function [mySurf] = updateFSurfPlots(init,gui,data,mySurf)
```

```
% Update the force and moment surf plots. The inputs are the initiation
% variable, the handle for the GUI, the data structure, and the handle for
% the surf plots. The output is the handle for the surf plots.
%
%   [mySurf] = updateFSurfPlots(init,gui,data,mySurf)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    axFBendHor = getappdata(gui,'axFBendHor');
    axFComp = getappdata(gui,'axFComp');
    axMBendVer = getappdata(gui,'axMBendVer');
    axTorq = getappdata(gui,'axTorq');

    if ~init
        % delete previous surf plots
        delete(mySurf.FBend)
        delete(mySurf.Comp)
        delete(mySurf.MBend)
        delete(mySurf.Torq)
    end

    mySurf.FBend = surf(axFBendHor,data.t2mat,data.t3mat,data.FUppHMat,...
                'EdgeAlpha',0.2,'FaceAlpha',1);
    mySurf.Comp = surf(axFComp,data.t2mat,data.t3mat,data.FCompMat,...
                'EdgeAlpha',0.2,'FaceAlpha',1);
    mySurf.MBend = surf(axMBendVer,data.t2mat,data.t3mat,data.MUppVMat,...
                'EdgeAlpha',0.2,'FaceAlpha',1);
    mySurf.Torq = surf(axTorq,data.t2mat,data.t3mat,data.TorqMat,...
                'EdgeAlpha',0.2,'FaceAlpha',1);
end
```

A.14.updateMaxDeflection.m

```
function [] = updateMaxDeflection(gui,data)
% Find the maximum deflection in each direction for each tube selection and
% print it to the corresponding text fields. The inputs are the handle for
% the GUI and the data structure. There is no output.
%
%   [] = updateMaxDeflection(gui,data)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    for i = ["Cy1" "Cy2h" "Cy2v" "Sq1" "Sq2h" "Sq2v"]
        % get the text box handles
        eval(strcat(...
            'tdmLowV',i,' = getappdata(gui,''tdmLowV',i,''');',...
            'tdmLowH',i,' = getappdata(gui,''tdmLowH',i,''');',...
            'tdmUppV',i,' = getappdata(gui,''tdmUppV',i,''');',...
            'tdmUppH',i,' = getappdata(gui,''tdmUppH',i,''');',...
            'tdmUppComp',i,' = getappdata(gui,''tdmUppComp',i,''');'));

        % calculate and write to text box
        eval(strcat(...
            'set(tdmLowV',i,','''String'',max(abs(data.d.',i,'.Low.tot.Ver),[],''all'')*1000);',...
            'set(tdmLowH',i,','''String'',max(abs(data.d.',i,'.Low.tot.Hor),[],''all'')*1000);',...
            'set(tdmUppV',i,','''String'',max(abs(data.d.',i,'.Upp.tot.Ver),[],''all'')*1000);',...
            'set(tdmUppH',i,','''String'',max(abs(data.d.',i,'.Upp.tot.Hor),[],''all'')*1000);',...
            'set(tdmUppComp',i,','''String'',max(abs(data.d.',i,'.Upp.Comp),[],''all'')*1000);'));
    end
end
```

A.15.updatePoints.m

```
function [point] = updatePoints(init,gui,spec,data,point)
% Update the points in the force and moment surf plots that indicate the
% force and moment value for the corresponding user input arm angles. The
% inputs are the initiation variable, the handle for the GUI, the
% specification structure, the data structure, and the handle for the
% points. The output is the handle for the points.
%
%   [point] = updatePoints(init,gui,spec,data,point)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    axFBendHor = getappdata(gui,'axFBendHor');
    axFComp = getappdata(gui,'axFComp');
    axMBendVer = getappdata(gui,'axMBendVer');
    axTorq = getappdata(gui,'axTorq');

    if ~init
        % remove previoius plotted points
        delete(point.FBendHor)
        delete(point.Comp)
        delete(point.MBendVer)
        delete(point.Torq)
    end

    % plot points in 3d plots
    point.FBendHor = plot3(axFBendHor,spec.t2,spec.t3,...
        data.FUppHMat(spec.t2i,spec.t3i),...
        'ro','LineWidth',3,'MarkerSize',3);
    point.Comp = plot3(axFComp,spec.t2,spec.t3,...
        data.FCompMat(spec.t2i,spec.t3i),...
        'ro','LineWidth',3,'MarkerSize',3);
    point.MBendVer = plot3(axMBendVer,spec.t2,spec.t3,...
        data.MUppVMat(spec.t2i,spec.t3i),...
        'ro','LineWidth',3,'MarkerSize',3);
    point.Torq = plot3(axTorq,spec.t2,spec.t3,...
        data.TorqMat(spec.t2i,spec.t3i),...
        'ro','LineWidth',3,'MarkerSize',3);
end
```

A.16.updateResults.m

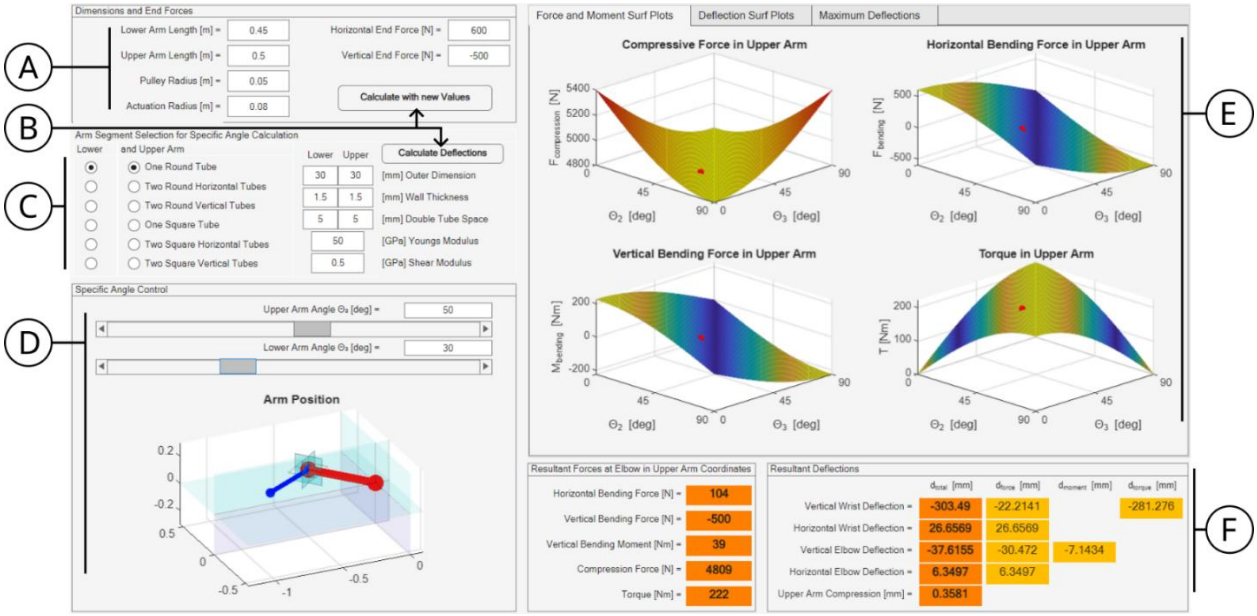
```
function [data] = updateResults(gui,spec,data)
% Update the force and moment data shown in the results table by getting
% the forces and moments for the current user input arm angles and changing
% the relevant text fields. The inputs are the handle for the GUI, the
% specifications structure, and the data structure. The output is an
% updated data structure.
%
%   [data] = updateResults(gui,spec,data)

% NACT Static Calculator
% Philip BL Raaphorst
% 10-04-2020

    tFBendHor = getappdata(gui,'tFBendHor');
    tFComp = getappdata(gui,'tFComp');
    tFBendVer = getappdata(gui,'tFBendVer');
    tMBendVer = getappdata(gui,'tMBendVer');
    tTorq = getappdata(gui,'tTorq');

    % calculate and display forces
    [data] = FFunc(spec,data);
    set(tFBendHor,'String',round(data.FUppH))
    set(tFComp,'String',round(data.FUppComp))
    set(tFBendVer,'String',round(data.FUppV))
```

```
    set(tMBendVer,'String',round(data.MUppV))
    set(tTorq,'String',round(data.TUpp))
end
```

APPENDIX D. Static Force and Deflection Calculation Tool Interface

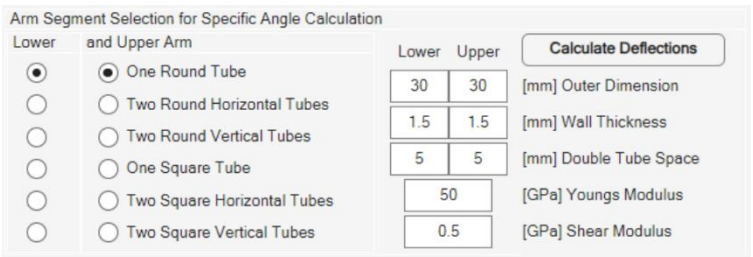
This appendix contains an overview of the static force and deflection calculation tool's interface.

GUI of the 'NACTArmStaticCalculator' tool; (A) manipulator dimensions and load inputs, (B) control to calculate resulting forces and deflections, (C) manipulator segment configuration and dimension inputs, (D) manipulator position configuration input sliders with visualization, (E) output plots, and (F) resultant forces and deflections for the defined inputs:

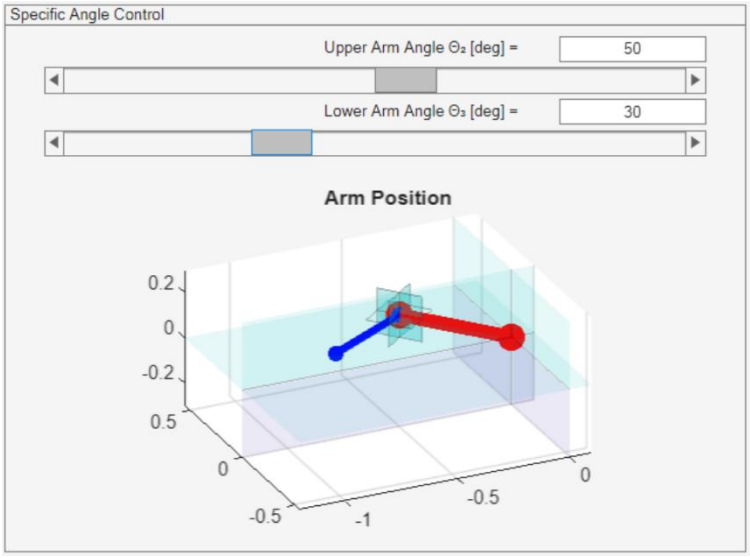


Dimension and end force input fields with button to perform calculation:

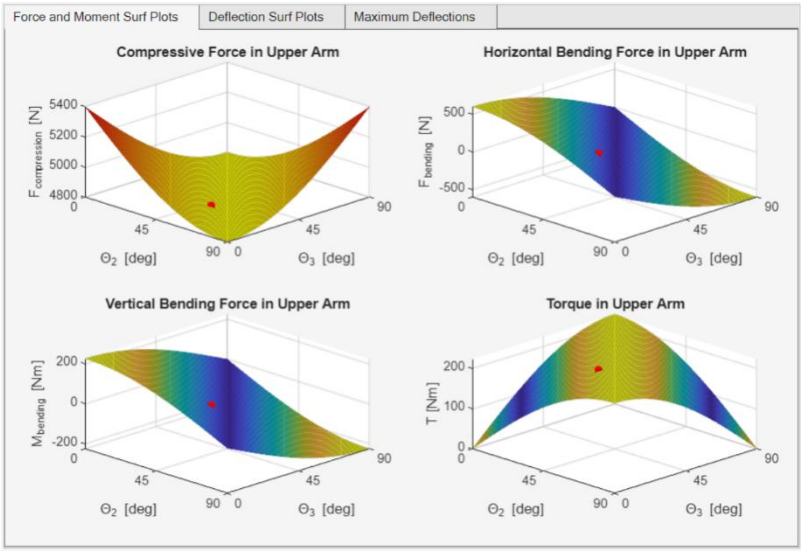
Options and input fields to determine the cross sectional shape, dimension, and properties of the manipulator segments as well as a button to perform the deflection calculations with the given input values.



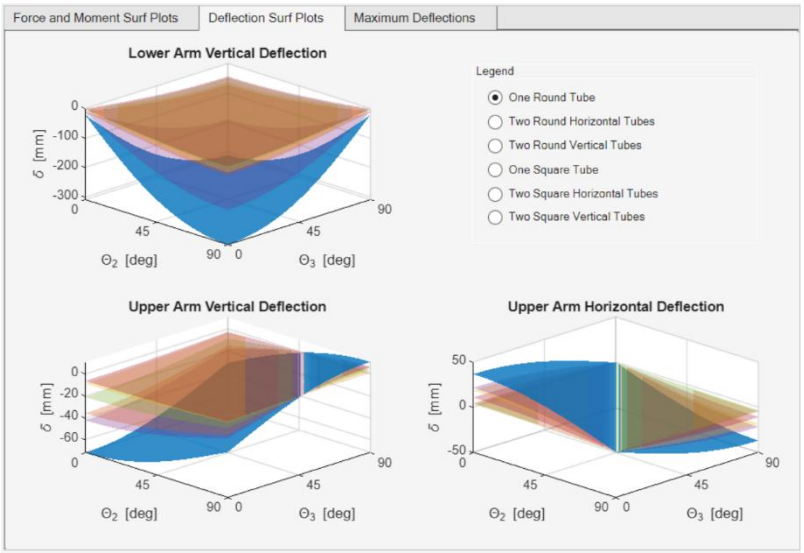
Interactable sliders and input to control the orientation of the manipulator segments:



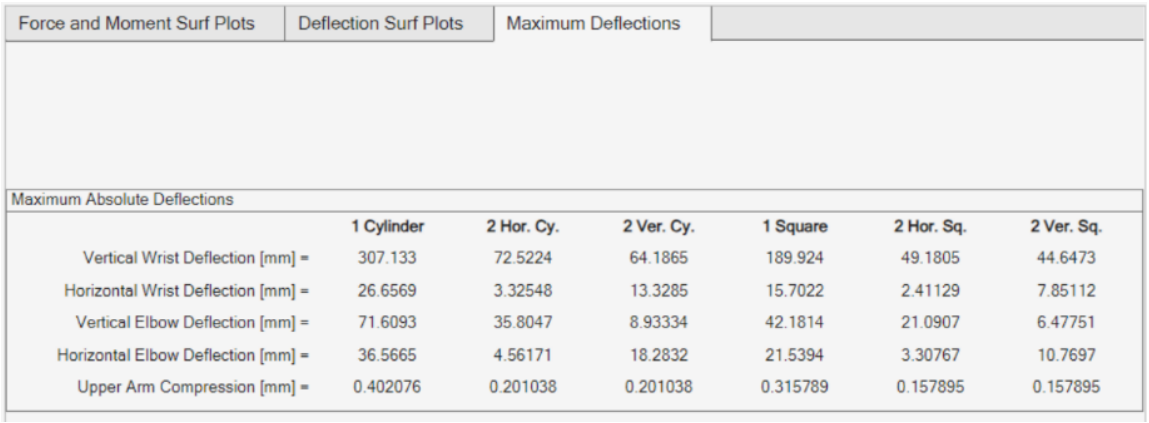
3D surf plots to show the forces and moments for all manipulator segment orientations, a red point indicated the values of the current user-defined orientations:



3D plots to show the deflections of various points on the manipulator for all manipulator segment orientations:



Results table show all maximum calculated deflections for different manipulator segment configurations:



Maximum Absolute Deflections

	1 Cylinder	2 Hor. Cy.	2 Ver. Cy.	1 Square	2 Hor. Sq.	2 Ver. Sq.
Vertical Wrist Deflection [mm] =	307.133	72.5224	64.1865	189.924	49.1805	44.6473
Horizontal Wrist Deflection [mm] =	26.6569	3.32548	13.3285	15.7022	2.41129	7.85112
Vertical Elbow Deflection [mm] =	71.6093	35.8047	8.93334	42.1814	21.0907	6.47751
Horizontal Elbow Deflection [mm] =	36.5665	4.56171	18.2832	21.5394	3.30767	10.7697
Upper Arm Compression [mm] =	0.402076	0.201038	0.201038	0.315789	0.157895	0.157895

Results table showing important forces and moments as well as the deflections and the composition of each deflection:

Resultant Forces at Elbow in Upper Arm Coordinates

Horizontal Bending Force [N] =	104
Vertical Bending Force [N] =	-500
Vertical Bending Moment [Nm] =	39
Compression Force [N] =	4809
Torque [Nm] =	222

Resultant Deflections

	d_{total} [mm]	d_{force} [mm]	d_{moment} [mm]	d_{torque} [mm]
Vertical Wrist Deflection =	-303.49	-22.2141		-281.276
Horizontal Wrist Deflection =	26.6569	26.6569		
Vertical Elbow Deflection =	-37.6155	-30.472	-7.1434	
Horizontal Elbow Deflection =	6.3497	6.3497		
Upper Arm Compression [mm] =	0.3581			

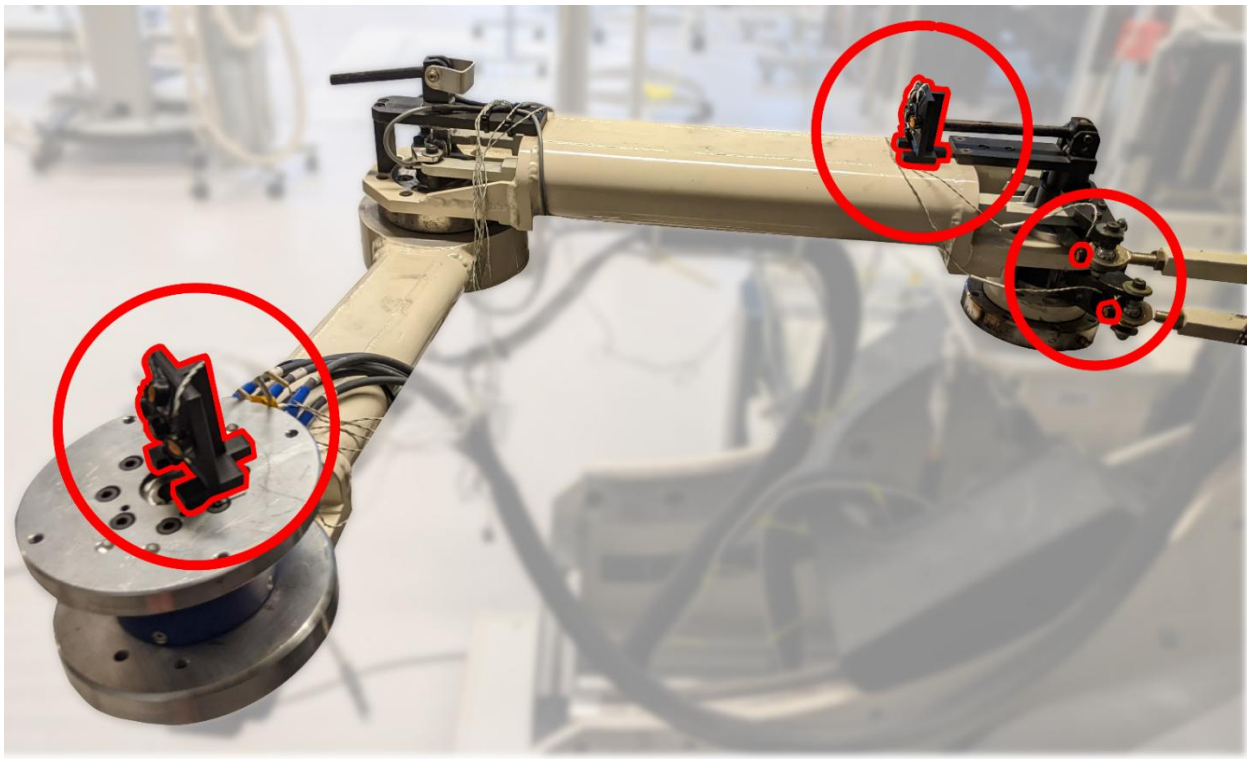
APPENDIX E. Experimental Marker Placements

This appendix contains images showing where the OptoTrak system's IR markers were placed on the NACT-3D's manipulator.

The types of markers used consisted of:

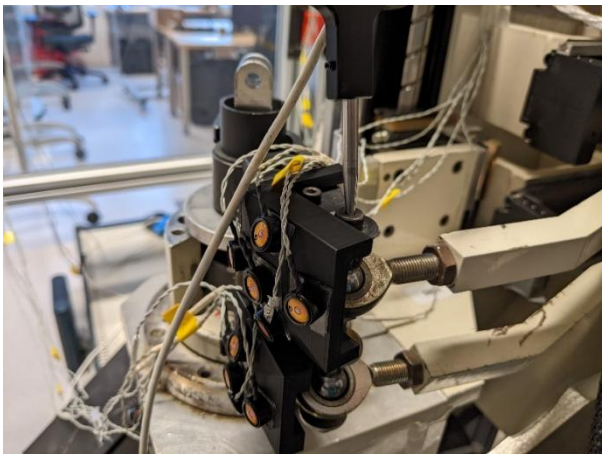
- Single Marker – only one marker which can only be used to track position.
- Small Orb – consists of 5 markers and can track position as well as orientation.
- Digitizing Probe – used to defined the coordinate system.
- Marker Plate – placed on a stationary object as a reference.

Marker placements on the manipulator:



Zoomed in view of markers placed on the actuation levers during testing with the manipulator:

Marker placement on the actuation levers during testing without the manipulator:



Definition of marker positions as found in the raw data:

- Attached: small orb, digitizing probe, 4 extra markers.
- Marker 1 through 9 are on the small orb.
- Marker 10 through 15 are the digitizing probe.
- Marker 16 through 18 are fixed to the NACT face plate.
- Marker 19 is on the NACT endpoint, roughly at -70 degrees relative to upper arm.
- Digitized position 1 and 2 are the elbow point of rotation.
- Analog 1 is a dud.
- Digital 1-4 and 6-8 are duds.
- Digital 5 is the cycle time telegraph input from the NACT.

APPENDIX F. MATLAB

This appendix contains the MATLAB code used to process the data from the manipulator experimental perturbations. The MATLAB code consists of six main code blocks:

- mainDataProcessing.m – This script calls all other functions to perform the data processing for all experimental data.
- runConfig.m – This script defines configuration settings that enable/disable the printing of the script's results in the console and the progress in console and dialogues.
- importDataFunc.m – This script imports all experimental data from the NACT-3D and OptoTrak. The data is sorted into a logical data structure.
- preProcessing.m – This script contains many secondary functions that:
    - remove broken data,
    - calculate lever angles and projected endpoint positions for lever experiments,
    - determines data type for separate ramp trials,
    - performs signal time alignment,
    - calculates velocities and accelerations,
    - splits data into separate ramp and multi-sine signals,
    - and adjusts for time alignment offsets.
- sysID.m – This script performs the system identification for the NACT-3D manipulator multi-sine experimental data.
- sys2excel.m – This script saves all calculated bandwidths to an excel file.

F.1. mainDataProcessing.m

```
close all force
clear

config = runConfig();

%% Import all data

% Define locations and naming structure of raw data
str.dirManipRamp    = 'C:\Users\flipr\Documents\NU\Measurements\PhilipData\2021_12_14';
str.dirManipSine    = 'C:\Users\flipr\Documents\NU\Measurements\PhilipData\2021_12_15';
str.dirLeverRamp    = 'C:\Users\flipr\Documents\NU\Measurements\PhilipData\2022_02_23_RampTrials';
str.dirLeverSine    =
'C:\Users\flipr\Documents\NU\Measurements\PhilipData\2022_02_23_MultiSineTrials';
str.NSVsub          = '\SubSave*.mat';
str.OptManip        = '\PhilipExperiment_*.mat';
str.OptLever        = '\LeverExperiment_*.mat';

[dataManipRamp,...
    dataManipSine,...
    dataLeverRamp,...
    dataLeverSine,...
    nExp]               = importDataFunc(str,config);

%% Data Preprocessing
% Get data types, align the data, calculate velocities and accelerations,
% and split the data into separate signals for the ramp- and multisine
% perturbations.
[dataManipRamp,...
    dataManipSine,...
    dataLeverRamp,...
    dataLeverSine,...
    offset]             = preProcessing(dataManipRamp,...
                                        dataManipSine,...
                                        dataLeverRamp,...
                                        dataLeverSine,...
                                        nExp,...
                                        config,...
                                        'All');

Sin_Freq = [1 2 3 4 5 6 7 8 12 16 20 24 28 32 40 48 56 64 80 96 112 128];   % Frequencies included in
MultiSine Signal

%% System Identification for NACT-3D experimental data
sys = struct;
sys = sysID(sys, dataManipSine, 'Manip', Sin_Freq);
sys = sysID(sys, dataLeverSine, 'Lever', Sin_Freq);

%% Write results to an excel document
sys2excel(sys);
```

F.2. runConfig.m

```
function config = runConfig()
% Ask the user whether or not the data processing steps should be printed
% to console and whether or not the data should be plotted/shown in between
% steps.

printConsole_ = questdlg('Do you want the data processing progression to be printed in the
console?',...
                    'Print in Console',...
                    'Yes','No','Yes');
switch printConsole_
    case 'Yes'
        config.printConsole = true;
    case 'No'
        config.printConsole = false;
end

showData_ = questdlg('Do you want to see all data plotted after each processing step?',...
                    'Display Data',...
                    'Yes','No','No');
switch showData_
    case 'Yes'
        config.showData = true;
    case 'No'
        config.showData = false;
end
```

F.3. importDataFunc.m

```
function [dataManipRamp,dataManipSine,dataLeverRamp,dataLeverSine,nExp] = importDataFunc(str,config)
% Import all data for NACT-3D experiments; the NSV data from the NACT-3D
% itself and the Opt data from the OptoTrak.

% Manipulator Ramp
strManipRampNSVsub  = dir([str.dirManipRamp str.NSVsub]);
strManipRampOpt     = dir([str.dirManipRamp str.OptManip]);
nExp.MR             = length({strManipRampOpt.name});                       % number of Ramp
Manipulator experiments
dataManipRamp       = struct;
dataManipRamp       = importNSVDataFunc(dataManipRamp,strManipRampNSVsub,'Manipulator','Ramp',config);
    % import NACT3D servo data
dataManipRamp       = importOptDataFunc(dataManipRamp,strManipRampOpt,'Manipulator','Ramp',config);
% import OptoTrak data (add additional input arguement to also import flat plate data)

% Manipulator Sine
strManipSineNSVsub  = dir([str.dirManipSine str.NSVsub]);
strManipSineOpt     = dir([str.dirManipSine str.OptManip]);
nExp.MS             = length({strManipSineOpt.name});                       % number of Sine
Manipulator experiments
dataManipSine       = struct;
dataManipSine       = importNSVDataFunc(dataManipSine,strManipSineNSVsub,'Manipulator','Sine',config);
    % import NACT3D servo data
dataManipSine       = importOptDataFunc(dataManipSine,strManipSineOpt,'Manipulator','Sine',config);
% import OptoTrak data

% Lever Ramp
strNSVsub       = dir([str.dirLeverRamp str.NSVsub]);
strLeverRampOpt = dir([str.dirLeverRamp str.OptLever]);
nExp.LR         = length({strLeverRampOpt.name});
dataLeverRamp   = struct;
dataLeverRamp   = importNSVDataFunc(dataLeverRamp,strNSVsub,'Lever','Ramp',config);         % import
NACT3D servo data
dataLeverRamp   = importOptDataFunc(dataLeverRamp,strLeverRampOpt,'Lever','Ramp',config);% import
OptoTrak data

% Lever Sine
strNSVsub       = dir([str.dirLeverSine str.NSVsub]);
strLeverSineOpt = dir([str.dirLeverSine str.OptLever]);
nExp.LS         = length({strLeverSineOpt.name});
dataLeverSine   = struct;
dataLeverSine   = importNSVDataFunc(dataLeverSine,strNSVsub,'Lever','Sine',config);         % import
NACT3D servo data
dataLeverSine   = importOptDataFunc(dataLeverSine,strLeverSineOpt,'Lever','Sine',config);% import
OptoTrak data


function [myDataStruct] = importNSVDataFunc(myDataStruct,DataName,Mode,Type,config)
% Import experiment data from the NACT-3D device

nExp = length({DataName.name});

for i = 1:nExp

    % import relevant optical and NSV data
    myData = importdata(DataName(i).name);

    % time data
    myDataStruct.Exp(i).NSV.TimeStamp = myData.TimeStamp;

    % import cursor positions
    myDataStruct.Exp(i).NSV.RefX = myData.P_Endef_Reference_0;
    myDataStruct.Exp(i).NSV.RefY = myData.P_Endef_Reference_1;
    myDataStruct.Exp(i).NSV.RefZ = myData.P_Endef_Reference_2;

    % import estiamted positions according to servos
    myDataStruct.Exp(i).NSV.EstX = myData.P_Endef_Actual_0;
    myDataStruct.Exp(i).NSV.EstY = myData.P_Endef_Actual_1;
    myDataStruct.Exp(i).NSV.EstZ = myData.P_Endef_Actual_2;
```

```
    % end point forces
    myDataStruct.Exp(i).NSV.rFx    = myData.F_Cursor_Actual_0;
    myDataStruct.Exp(i).NSV.rFy = myData.F_Cursor_Actual_1;
    myDataStruct.Exp(i).NSV.rFz = myData.F_Cursor_Actual_2;
    myDataStruct.Exp(i).NSV.Fx = myData.F_Endef_Actual_0;
    myDataStruct.Exp(i).NSV.Fy = myData.F_Endef_Actual_1;
    myDataStruct.Exp(i).NSV.Fz = myData.F_Endef_Actual_2;
    myDataStruct.Exp(i).NSV.Tx = myData.Tau_Endef_Actual_0;
    myDataStruct.Exp(i).NSV.Ty = myData.Tau_Endef_Actual_1;
    myDataStruct.Exp(i).NSV.Tz = myData.Tau_Endef_Actual_2;

    % Servo Slider positions and velocities
    myDataStruct.Exp(i).NSV.ServoElbowPRef          = myData.PQ_Elbow_Reference;
    myDataStruct.Exp(i).NSV.ServoElbowPActual       = myData.PQ_Elbow_Actual;
    myDataStruct.Exp(i).NSV.ServoElbowPLinear       = myData.PQ_Elbow_Linear;
    myDataStruct.Exp(i).NSV.ServoElbowPDrive        = myData.PQ_Elbow_Drive;
    myDataStruct.Exp(i).NSV.ServoElbowVRef          = myData.VQ_Elbow_Reference;
    myDataStruct.Exp(i).NSV.ServoElbowVActual       = myData.VQ_Elbow_Actual;

    myDataStruct.Exp(i).NSV.ServoShoulderPRef       = myData.PQ_Shoulder_Reference;
    myDataStruct.Exp(i).NSV.ServoShoulderPActual  = myData.PQ_Shoulder_Actual;
    myDataStruct.Exp(i).NSV.ServoShoulderPLinear  = myData.PQ_Shoulder_Linear;
    myDataStruct.Exp(i).NSV.ServoShoulderPDrive     = myData.PQ_Shoulder_Drive;
    myDataStruct.Exp(i).NSV.ServoShoulderVRef       = myData.VQ_Shoulder_Reference;
    myDataStruct.Exp(i).NSV.ServoShoulderVActual  = myData.VQ_Shoulder_Actual;

    myDataStruct.Exp(i).NSV.ServoVerticalPRef       = myData.PQ_Vertical_Reference;
    myDataStruct.Exp(i).NSV.ServoVerticalPActual  = myData.PQ_Vertical_Actual;
    myDataStruct.Exp(i).NSV.ServoVerticalPLinear  = myData.PQ_Vertical_Linear;
    myDataStruct.Exp(i).NSV.ServoVerticalPDrive     = myData.PQ_Vertical_Drive;
    myDataStruct.Exp(i).NSV.ServoVerticalVRef       = myData.VQ_Vertical_Reference;
    myDataStruct.Exp(i).NSV.ServoVerticalVActual  = myData.VQ_Vertical_Actual;

    if config.printConsole
        disp([Mode ' ' Type ' NSV Data set ' num2str(i) ' of ' num2str(nExp) ' imported.'])
    end

end


function [myDataStruct] = importOptDataFunc(myDataStruct,DataName,Mode,Type,config,varargin)
% Import experiment data from OptoTrak

nExp = length({DataName.name});

for i = 1:nExp
    % import relevant data
    myData = importdata(DataName(i).name);

    % replace all '-' with '_' to avoid errors
    myData.Properties.VariableNames = regexprep(myData.Properties.VariableNames,{'-',' '},'_');

    % create cell array with all field names of data timetable
    if exist('names','var') == 0; names = fieldnames( myData ); end

    % time data
%     myDataStruct.Exp(i).Opt.t = milliseconds(myData.Time);
    myDataStruct.Exp(i).Opt.TCCycle = myData.TCCycle;

    % remove NaN values in TCCycle
    % NOTE: I have tried to use fillmissing(), but some of the data points
    % in the TCCycle signal have outlying values. Mostly there are single
    % time values surrounded by many NaN's that have a timestamp that is
    % non-linear, but sometimes there's also a grouping of these abnormal
    % values and filloutliers() doesn't seem to always help. The method
    % below does make it so that the eventual time data can be a couple
    % milliseconds off.
    N = length(myData.TCCycle);
    idx = find(~isnan(myData.TCCycle),1);
    value = myData.TCCycle(find(~isnan(myData.TCCycle),1));
    myDataStruct.Exp(i).Opt.TCCycle_filled = (1:N)' + value - idx;

    % extract OptoTrack position data
    switch Mode
```

```
        case 'Manipulator'
            myDataStruct.Exp(i).Opt.X = -eval(['myData.' char(names(contains(names,'endpoint_x')))]);
            myDataStruct.Exp(i).Opt.Y = -eval(['myData.' char(names(contains(names,'endpoint_y')))]);
            myDataStruct.Exp(i).Opt.Z = eval(['myData.' char(names(contains(names,'endpoint_z')))]);
        case 'Lever'
            myDataStruct.Exp(i).Opt.elbowX = eval(['myData.' char(names(contains(names,'elbow_x')))]);
            myDataStruct.Exp(i).Opt.elbowY = eval(['myData.' char(names(contains(names,'elbow_y')))]);
            myDataStruct.Exp(i).Opt.elbowZ = eval(['myData.' char(names(contains(names,'elbow_z')))]);

            myDataStruct.Exp(i).Opt.shoulderX = eval(['myData.'
char(names(contains(names,'should_x')))]);
            myDataStruct.Exp(i).Opt.shoulderY = eval(['myData.'
char(names(contains(names,'should_y')))]);
            myDataStruct.Exp(i).Opt.shoulderZ = eval(['myData.'
char(names(contains(names,'should_z')))]);
    end

    if ~isempty(varargin)
        % flat plate data
        myDataStruct.Exp(i).Opt.fp1x = -eval(['myData.' char(names(contains(names,'fp_1_x')))]);
        myDataStruct.Exp(i).Opt.fp1y = -eval(['myData.' char(names(contains(names,'fp_1_y')))]);
        myDataStruct.Exp(i).Opt.fp1z =  eval(['myData.' char(names(contains(names,'fp_1_z')))]);
        myDataStruct.Exp(i).Opt.fp2x = -eval(['myData.' char(names(contains(names,'fp_2_x')))]);
        myDataStruct.Exp(i).Opt.fp2y = -eval(['myData.' char(names(contains(names,'fp_2_y')))]);
        myDataStruct.Exp(i).Opt.fp2z =  eval(['myData.' char(names(contains(names,'fp_2_z')))]);
        myDataStruct.Exp(i).Opt.fp3x = -eval(['myData.' char(names(contains(names,'fp_3_x')))]);
        myDataStruct.Exp(i).Opt.fp3y = -eval(['myData.' char(names(contains(names,'fp_3_y')))]);
        myDataStruct.Exp(i).Opt.fp3z =  eval(['myData.' char(names(contains(names,'fp_3_z')))]);
        myDataStruct.Exp(i).Opt.fp4x = -eval(['myData.' char(names(contains(names,'fp_4_x')))]);
        myDataStruct.Exp(i).Opt.fp4y = -eval(['myData.' char(names(contains(names,'fp_4_y')))]);
        myDataStruct.Exp(i).Opt.fp4z =  eval(['myData.' char(names(contains(names,'fp_4_z')))]);
        myDataStruct.Exp(i).Opt.fp5x = -eval(['myData.' char(names(contains(names,'fp_5_x')))]);
        myDataStruct.Exp(i).Opt.fp5y = -eval(['myData.' char(names(contains(names,'fp_5_y')))]);
        myDataStruct.Exp(i).Opt.fp5z =  eval(['myData.' char(names(contains(names,'fp_5_z')))]);
        myDataStruct.Exp(i).Opt.fp6x = -eval(['myData.' char(names(contains(names,'fp_6_x')))]);
        myDataStruct.Exp(i).Opt.fp6y = -eval(['myData.' char(names(contains(names,'fp_6_y')))]);
        myDataStruct.Exp(i).Opt.fp6z =  eval(['myData.' char(names(contains(names,'fp_6_z')))]);

        % quaternion data
        myDataStruct.Exp(i).Opt.fpq0 = eval(['myData.' char(names(contains(names,'flat_plate') &
contains(names,'q0')))]);
        myDataStruct.Exp(i).Opt.fpqx = eval(['myData.' char(names(contains(names,'flat_plate') &
contains(names,'qx')))]);
        myDataStruct.Exp(i).Opt.fpqy = eval(['myData.' char(names(contains(names,'flat_plate') &
contains(names,'qy')))]);
        myDataStruct.Exp(i).Opt.fpqz = eval(['myData.' char(names(contains(names,'flat_plate') &
contains(names,'qz')))]);

        % flat plate data processing
        myDataStruct.Exp(i).meanfp1 = [mean(myDataStruct.Exp(i).Opt.fp1x),
mean(myDataStruct.Exp(i).Opt.fp1y), mean(myDataStruct.Exp(i).Opt.fp1z)];
        myDataStruct.Exp(i).meanfp2 = [mean(myDataStruct.Exp(i).Opt.fp2x),
mean(myDataStruct.Exp(i).Opt.fp2y), mean(myDataStruct.Exp(i).Opt.fp2z)];
        myDataStruct.Exp(i).meanfp3 = [mean(myDataStruct.Exp(i).Opt.fp3x),
mean(myDataStruct.Exp(i).Opt.fp3y), mean(myDataStruct.Exp(i).Opt.fp3z)];
        myDataStruct.Exp(i).meanfp4 = [mean(myDataStruct.Exp(i).Opt.fp4x),
mean(myDataStruct.Exp(i).Opt.fp4y), mean(myDataStruct.Exp(i).Opt.fp4z)];
        myDataStruct.Exp(i).meanfp5 = [mean(myDataStruct.Exp(i).Opt.fp5x),
mean(myDataStruct.Exp(i).Opt.fp5y), mean(myDataStruct.Exp(i).Opt.fp5z)];
        myDataStruct.Exp(i).meanfp6 = [mean(myDataStruct.Exp(i).Opt.fp6x),
mean(myDataStruct.Exp(i).Opt.fp6y), mean(myDataStruct.Exp(i).Opt.fp6z)];

        % quaternions as array
        myDataStruct.Exp(i).Opt.Quat = [myDataStruct.Exp(i).Opt.fpq0, myDataStruct.Exp(i).Opt.fpqx,
myDataStruct.Exp(i).Opt.fpqy, myDataStruct.Exp(i).Opt.fpqz];
    end

    if config.printConsole
        disp([Mode ' ' Type ' Opt Data set ' num2str(i) ' of ' num2str(nExp) ' imported.'])
    end
end
```

F.4. preProcessing.m

```
function [dataMR,dataMS,dataLR,dataLS,varargout] =
preProcessing(dataMR,dataMS,dataLR,dataLS,nExp,config,exec)
% Main data pre processing function that calls all secondary functions
% contained in this file.

varargout{1} = 0;
if strcmp(exec,'All') || strcmp(exec,'Broken Data')
    if config.printConsole; disp('Removing broken data ...'); end
    % Remove broken data from Manipulator Sine Trial 10
    fnNSV = fieldnames(dataMS.Exp(10).NSV);    % field names for NSV data
    fnOpt = fieldnames(dataMS.Exp(10).Opt);    % field names for Opt data
    for i = 1:numel(fnNSV)
        eval(['dataMS.Exp(10).NSV.' fnNSV{i} ' = dataMS.Exp(10).NSV.' fnNSV{i} '(1:44400);']);
    end
    for i = 1:numel(fnOpt)
        eval(['dataMS.Exp(10).Opt.' fnOpt{i} ' = dataMS.Exp(10).Opt.' fnOpt{i} '(1:44400);']);
    end
    if config.printConsole; disp('Broken data removed.'); end
end
if strcmp(exec,'All') || strcmp(exec,'Lever Angles') || strcmp(exec,'Lever')
    if config.printConsole; disp('Calculating lever angles ...'); end
    % Calculate actuation lever angles and projected manipulator points for
    % lever experiments
    dataLR  = calcLeverProjection(dataLR,nExp.LR);
    dataLS  = calcLeverProjection(dataLS,nExp.LS);
    if config.printConsole; disp('Lever angles calcualted.'); end
end
if strcmp(exec,'All') || strcmp(exec,'Perturbation Type')
    if config.printConsole; disp('Determining data types ...'); end
    % Determine data type for ramp experiments (+x, +y, -x, or -y)
    dataMR  = dataTypeFunc(dataMR,nExp.MR,'Manipulator','Ramp',config);     % determine the
perturbation types
    if config.showData; guiPlotData(dataMR,'Raw','Manip'); end              % Show Raw Series Data
    dataLR  = dataTypeFunc(dataLR,nExp.LR,'Lever','Ramp',config);           % determine the
perturbation types
    dataMS  = dataTypeFunc(dataMS,nExp.MS,'Manipulator','Sine',config);          % determine the
perturbation types
    dataLS  = dataTypeFunc(dataLS,nExp.LS,'Lever','Sine',config);           % determine the
perturbation types
    if config.printConsole; disp('Data types determined.'); end
end
if strcmp(exec,'All') || strcmp(exec,'Align') || strcmp(exec,'Lever')
    if config.printConsole; disp('Aligning Data ...'); end
    % Data Time Alignment
    dataMR  = alignDataFunc(dataMR,nExp.MR);                                    % align the manipulator
ramp perturbation data
    if config.showData; guiPlotData(dataMR,'Aligned','Manip'); end              % Show aligned data
series
    dataMS  = alignDataFunc(dataMS,nExp.MS);                                    % align the manipulator
sine perturbation data
    if config.showData; guiPlotData(dataMS,'Aligned','Manip'); end              % Show aligned data
series
    dataLR  = alignDataFunc(dataLR,nExp.LR);                                    % align the lever ramp
perturbation data
    dataLS  = alignDataFunc(dataLS,nExp.LS);                                    % align the lever sine
perturbation data
    if config.printConsole; disp('Data aligned.'); end
end
if strcmp(exec,'All') || strcmp(exec,'Velocity') || strcmp(exec,'Lever')
    if config.printConsole; disp('Calculating velocities and accelerations ...'); end
    % Velocity and Acceleration calculation
    dataMR  = calcVelAcc(dataMR,nExp.MR,'Manip');
    dataMS  = calcVelAcc(dataMS,nExp.MS,'Manip');
    dataLR  = calcVelAcc(dataLR,nExp.LR,'Lever');
    dataLS  = calcVelAcc(dataLS,nExp.LS,'Lever');
    if config.printConsole; disp('Velocities and accelerations calculated.'); end
end
if strcmp(exec,'All') || strcmp(exec,'Split') || strcmp(exec,'Lever')
    if config.printConsole; disp('Splitting separate signals ...'); end
    % Split experiment data into separate ramps or multisine signals
```

```
    dataMR  = splitRampsFunc(dataMR,nExp.MR,'Manip',config);                    % split each ramp into
separate variables
    if config.showData; guiPlotData(dataMR,'Signals','Manip'); end              % Show aligned
manipulator ramp data series
    dataMS  = splitSineFunc(dataMS,nExp.MS,'Manip',config);                     % split each multisine
signal into separate variables
    dataLR  = splitRampsFunc(dataLR,nExp.LR,'Lever',config);                    % split each ramp into
separate variables
    dataLS  = splitSineFunc(dataLS,nExp.LS,'Lever',config);                     % split each multisine
signal into separate variables
    if config.printConsole; disp('Signals split.'); end
end
if strcmp(exec,'All') || strcmp(exec,'Offset')
    if config.printConsole; disp('Calculating alignment offset ...'); end
    % Calculate data alignment offset
    offset.ManipRamp   = calcOffset(dataMR,'Manip','Ramp');
    offset.ManipSine   = calcOffset(dataMS,'Manip','Sine');
    offset.LeverRamp   = calcOffset(dataLR,'Lever','Ramp');
    offset.LeverSine   = calcOffset(dataLS,'Lever','Sine');
    varargout{1} = offset;
    if config.printConsole; disp('Alignment offset calculated.'); end
end
if strcmp(exec,'All') || strcmp(exec,'Iddata')
    if config.printConsole; disp('Creating iddata object ...'); end
    % Miscellaneous
    dataMR      = rampIddata(dataMR,'Manipulator',config);                      % Create separate
iddata objects for ramps
    if config.printConsole; disp('Iddata object created.'); end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%                        SECONDARY FUNCTIONS BELOW                        %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [myDataStruct] = dataTypeFunc(myDataStruct,nExp,mode,expType,config)

% Stitch all reference data together in one array
stitchedXRef = [];
stitchedYRef = [];
for i = 1:nExp
    stitchedXRef = [stitchedXRef; myDataStruct.Exp(i).NSV.RefX];
    stitchedYRef = [stitchedYRef; myDataStruct.Exp(i).NSV.RefY];
end

switch expType
    case 'Ramp'     % determine data type for ramp experiments
        for i = 1:nExp

            % extract initial value of measurement series
            iniRefX = round(myDataStruct.Exp(i).NSV.RefX(1),-1);
            iniRefY = round(myDataStruct.Exp(i).NSV.RefY(1),-1);

            % extract middle value of measurement series
            midRefX = round(myDataStruct.Exp(i).NSV.RefX(ceil(end/2)),-1);
            midRefY = round(myDataStruct.Exp(i).NSV.RefY(ceil(end/2)),-1);

            % average measurement series without first and last 10% of data
            avgRefX = round(mean(myDataStruct.Exp(i).NSV.RefX(ceil(.1*end):ceil(.9*end))),-1);
            avgRefY = round(mean(myDataStruct.Exp(i).NSV.RefY(ceil(.1*end):ceil(.9*end))),-1);

            % determine perturbation type; find if perturbations are x+, y-, x-, or y+
            if      (avgRefX-iniRefX) > 0  &&  avgRefY == midRefY
                if config.printConsole; disp([mode ' ramp experiment ' num2str(i) ' is type 1:
perturbations in the positive x direction']); end
                myDataStruct.Exp(i).Type = 1;
            elseif  avgRefX == midRefX &&  (avgRefY-iniRefY) < 0
                if config.printConsole; disp([mode ' ramp experiment ' num2str(i) ' is type 2:
perturbations in the negative y direction']); end
                myDataStruct.Exp(i).Type = 2;
            elseif  (avgRefX-iniRefX) < 0  &&  avgRefY == midRefY
                if config.printConsole; disp([mode ' ramp experiment ' num2str(i) ' is type 3:
perturbations in the negative x direction']); end
                myDataStruct.Exp(i).Type = 3;
```

```
            elseif  avgRefX == midRefX  && (avgRefY-iniRefY) > 0
                if config.printConsole; disp([mode ' ramp experiment ' num2str(i) ' is type 4:
perturbations in the positive y direction']); end
                myDataStruct.Exp(i).Type = 4;
            else
                currentExpType = guiType(myDataStruct,i,stitchedXRef,stitchedYRef);
                if config.printConsole; disp([mode ' ramp experiment ' num2str(i) ' is type '
num2str(currentExpType) ' (user input)']); end
                myDataStruct.Exp(i).Type = currentExpType;
            end

        end
    case 'Sine'     % determine data type for sine experiments
        for i = 1:nExp
            nx = sum(ischange(myDataStruct.Exp(i).NSV.RefX));
            ny = sum(ischange(myDataStruct.Exp(i).NSV.RefY));
            % determine perturbation type; find if perturbations are x or y
            if nx > ny
                if config.printConsole; disp([mode ' sine experiment ' num2str(i) ' is type 1: x
perturbations']); end
                myDataStruct.Exp(i).Type = 1;
            elseif nx < ny
                if config.printConsole; disp([mode ' sine experiment ' num2str(i) ' is type 2: y
perturbations']); end
                myDataStruct.Exp(i).Type = 2;
            end
        end
end

function myData = calcLeverProjection(myData,nExp)
% calculate
LUpperArm   = 500;  % [mm]
LLowerArm   = 450;  % [mm]

for i = 1:nExp
    myData.Exp(i).Opt.elbowAngleSin     = asin(myData.Exp(i).Opt.elbowX/85);
    myData.Exp(i).Opt.elbowAngleCos     = acos(myData.Exp(i).Opt.elbowY/85);
    myData.Exp(i).Opt.elbowAngleTan     = atan(myData.Exp(i).Opt.elbowX./myData.Exp(i).Opt.elbowY);
    myData.Exp(i).Opt.elbowLeverLength  =
sqrt(myData.Exp(i).Opt.elbowX.^2+myData.Exp(i).Opt.elbowY.^2);

    myData.Exp(i).Opt.shoulderAngleSin      = asin(myData.Exp(i).Opt.shoulderX/85);
    myData.Exp(i).Opt.shoulderAngleCos      = acos(myData.Exp(i).Opt.shoulderY/85);
    myData.Exp(i).Opt.shoulderAngleTan      =
atan(myData.Exp(i).Opt.shoulderX./myData.Exp(i).Opt.shoulderY);
    myData.Exp(i).Opt.shoulderLeverLength   =
sqrt(myData.Exp(i).Opt.shoulderX.^2+myData.Exp(i).Opt.shoulderY.^2);

    myData.Exp(i).Opt.elbowProjX    = LUpperArm * sin(myData.Exp(i).Opt.shoulderAngleTan + 2*pi/3);
    myData.Exp(i).Opt.elbowProjY    = LUpperArm * cos(myData.Exp(i).Opt.shoulderAngleTan + 2*pi/3);
    myData.Exp(i).Opt.elbowProjXSin    = LUpperArm * sin(myData.Exp(i).Opt.shoulderAngleSin + 2*pi/3);
    myData.Exp(i).Opt.elbowProjYSin    = LUpperArm * cos(myData.Exp(i).Opt.shoulderAngleSin + 2*pi/3);
    myData.Exp(i).Opt.elbowProjXCos    = LUpperArm * sin(myData.Exp(i).Opt.shoulderAngleCos + 2*pi/3);
    myData.Exp(i).Opt.elbowProjYCos    = LUpperArm * cos(myData.Exp(i).Opt.shoulderAngleCos + 2*pi/3);

    myData.Exp(i).Opt.wristProjX    = myData.Exp(i).Opt.elbowProjX + LLowerArm *
sin(myData.Exp(i).Opt.elbowAngleTan);
    myData.Exp(i).Opt.wristProjY    = myData.Exp(i).Opt.elbowProjY + LLowerArm *
cos(myData.Exp(i).Opt.elbowAngleTan);
    myData.Exp(i).Opt.wristProjXSin    = myData.Exp(i).Opt.elbowProjXSin + LLowerArm *
sin(myData.Exp(i).Opt.elbowAngleSin);
    myData.Exp(i).Opt.wristProjYSin    = myData.Exp(i).Opt.elbowProjYSin + LLowerArm *
cos(myData.Exp(i).Opt.elbowAngleSin);
    myData.Exp(i).Opt.wristProjXCos    = myData.Exp(i).Opt.elbowProjXCos + LLowerArm *
sin(myData.Exp(i).Opt.elbowAngleCos);
    myData.Exp(i).Opt.wristProjYCos    = myData.Exp(i).Opt.elbowProjYCos + LLowerArm *
cos(myData.Exp(i).Opt.elbowAngleCos);
end

function myData = alignDataFunc(myData,nExp)

% align data according to NSV TimeStamp and Opt TCCylce

fnNSV = fieldnames(myData.Exp(1).NSV);
```

```
fnOpt = fieldnames(myData.Exp(1).Opt);

for i = 1:nExp
    iDiff = diff([true;isnan(myData.Exp(i).Opt.TCCycle);true]);            % array with zeros, -1 for
transition from NaN to valid number, and 1 for transition from valid number to NaN
    iOptNV = find(iDiff<0);                                                % indexes of all NaN to
valid number transitions
    iOptBeg = iOptNV(1);

    iOptVN = find(iDiff>0)-1;                                              % indexes of all valid
number to NaN transitions
    iOptfinal = iOptVN(end);

    iNSVBeg = find(myData.Exp(i).NSV.TimeStamp == myData.Exp(i).Opt.TCCycle(iOptBeg));
%     iNSVEnd = iNSVBeg + length(myData.Exp(i).Opt_.t) - 1;

    nOpt = length(myData.Exp(i).Opt.TCCycle(iOptBeg:end));                 % length of adjusted Opt
signal
    nNSV = length(myData.Exp(i).NSV.TimeStamp(iNSVBeg:end));               % length of adjusted NSV
signal
    nSig = min(nOpt,nNSV);                                                 % length of adjusted
signal
    iOptEnd = iOptBeg + nSig - 1;
    iNSVEnd = iNSVBeg + nSig - 1;

    for j = 1:numel(fnOpt)
        myData.Exp(i).Opt_.(char(fnOpt(j))) = myData.Exp(i).Opt.(char(fnOpt(j)))(iOptBeg:iOptEnd);
    end
    % make time series starting at t=0
    myData.Exp(i).Opt_.t = myData.Exp(i).Opt_.TCCycle - myData.Exp(i).Opt_.TCCycle(1);
    myData.Exp(i).Opt_.t_filled = myData.Exp(i).Opt_.TCCycle_filled -
myData.Exp(i).Opt_.TCCycle_filled(1);

    finalValid = min(iOptfinal-iOptBeg+1,nSig);
    myData.Exp(i).Opt_off = myData.Exp(i).Opt_.t_filled(finalValid) -
myData.Exp(i).Opt_.t(finalValid);

    for j = 1:numel(fnNSV)
        myData.Exp(i).NSV_.(char(fnNSV(j))) = myData.Exp(i).NSV.(char(fnNSV(j)))(iNSVBeg:iNSVEnd);
    end
    % make time series starting at t=0
    myData.Exp(i).NSV_.t = myData.Exp(i).NSV_.TimeStamp - myData.Exp(i).NSV_.TimeStamp(1);
end

function myData = calcVelAcc(myData,nExp,mode)
% calculate velocities and accelerations

varNSV = {'RefX','RefY','RefZ','EstX','EstY','EstZ'};
switch mode
    case 'Manip'
        varOpt = {'X','Y','Z'};
        txt = 'manipulator end point';
    case 'Lever'
        varOpt =
{'elbowAngleTan','elbowProjX','elbowProjY','shoulderAngleTan','wristProjX','wristProjY'};
        txt = 'push pull rod lever';
end

nNSV = numel(varNSV);
nOpt = numel(varOpt);
nTot = nExp*(nNSV+nOpt);

H = waitbar(0,['Calculating ' mode ' ' txt ' data velocities...']);

for i = 1:nExp
    for j = 1:nNSV
        myData.Exp(i).NSV_.(sprintf('v%s',char(varNSV(j)))) =
dx(myData.Exp(i).NSV_.(char(varNSV(j))))*1000;
        myData.Exp(i).NSV_.(sprintf('a%s',char(varNSV(j)))) = dx(myData.Exp(i).NSV_.(['v'
char(varNSV(j))]));
        temp = (i-1)*(nNSV+nOpt)+j;
        waitbar(temp/nTot,H,sprintf('Calculating %s %s data velocities... %d/%d',mode,txt,temp,nTot));
    end
```

```
    myData.Exp(i).NSV_.ServoElbowAActual = dx(myData.Exp(i).NSV_.ServoElbowVActual);
    myData.Exp(i).NSV_.ServoShoulderAActual = dx(myData.Exp(i).NSV_.ServoShoulderVActual);

    for j = 1:nOpt
        myData.Exp(i).Opt_.(sprintf('v%s',char(varOpt(j)))) =
dx(myData.Exp(i).Opt_.(char(varOpt(j))))*1000; % times by 1000 so that it's mm/s
        myData.Exp(i).Opt_.(sprintf('a%s',char(varOpt(j)))) = dx(myData.Exp(i).Opt_.(['v'
char(varOpt(j))]));
        temp = (i-1)*(nNSV+nOpt)+nNSV+j;
        waitbar(temp/nTot,H,sprintf('Calculating %s %s data velocities... %d/%d',mode,txt,temp,nTot));
    end
end

close(H);

function v = dx(x)
incr = 1;
n = length(x);
v_ = filter([-2 -1 0 1 2],1,[x;0;0;0]);
v = -(v_(3:n+2))/(10.*incr);
v(1) = (-21.*x(1)+13.*x(2)+17.*x(3)-9.*x(4))/(20.*incr);
v(2) =  (-11.*x(1) + 3.*x(2) + 7.*x(3) + x(4))/(20.*incr);
v(n) = (21.*x(n)-13.*x(n-1)-17.*x(n-2)+9.*x(n-3))/(20.*incr);
v(n-1) =  (11.*x(n) - 3.*x(n-1) - 7.*x(n-2) - x(n-3))/(20.*incr);

function myData = splitRampsFunc(myData,nExp,mode,config)

switch mode
    case 'Manip'
        varOpt = {'vX','vY'};
        gain = 1;
    case 'Lever'
        varOpt = {'vwristProjX','vwristProjY'};
        gain = -1;
end

fnNSV = fieldnames(myData.Exp(1).NSV_);
fnOpt = fieldnames(myData.Exp(1).Opt_);

% Split each separate ramp response
for i = 1:nExp

    switch myData.Exp(i).Type
        case 1
            % The data in which peaks will be found is in the positive x
            % direction, save this as such.
            pksRefData = myData.Exp(i).NSV_.vRefX;
            pksOptData = myData.Exp(i).Opt_.(char(varOpt(1)));

        case 2
            % The data in which peaks will be found is in the negative y
            % direction, save this as such.
            pksRefData = -myData.Exp(i).NSV_.vRefY;
            pksOptData = -myData.Exp(i).Opt_.(char(varOpt(2))) * gain;

        case 3
            % The data in which peaks will be found is in the negative x
            % direction, save this as such.
            pksRefData = -myData.Exp(i).NSV_.vRefX;
            pksOptData = -myData.Exp(i).Opt_.(char(varOpt(1)));

        case 4
            % The data in which peaks will be found is in the positive y
            % direction, save this as such.
            pksRefData = myData.Exp(i).NSV_.vRefY;
            pksOptData = myData.Exp(i).Opt_.(char(varOpt(2))) * gain;

        case 5

        otherwise

    end

    % Find peaks in relevant data with at least a 9 second spacing and with
```

```
    % an amplitude of at least half of the max value.
    [pksRef,locsRef] =
findpeaks(pksRefData,'MinPeakDistance',9000,'MinPeakHeight',.5*max(pksRefData));
    [pksOpt,locsOpt] =
findpeaks(pksOptData,'MinPeakDistance',9000,'MinPeakHeight',.5*max(pksOptData));

    if config.printConsole
        disp([mode ' ramp experiment ' num2str(i) ' has ' ...
              num2str(numel(pksRef)) ' cursor peaks and ' ...
              num2str(numel(pksOpt)) ' OptoTrack peaks.'])
    end

    % Save each Ramp Signal separately
    tBefore = 0;
    tAfter = 1000;
    for j = 1:length(pksRef)
        k = j + (i-1)*5;
        iEnd = min(locsRef(j)+tAfter,length(myData.Exp(i).NSV_.t));

        for m = 1:numel(fnNSV)
            txt = char(fnNSV(m));
            if txt(1) == 'v'
                iEnd_ = iEnd-1;
            else
                iEnd_ = iEnd;
            end
            myData.Signal(k).NSV.(txt) = myData.Exp(i).NSV_.(txt)(locsRef(j)-tBefore:iEnd_);
%             eval(['myData.Signal(k).NSV.' fnNSV{m} ' = myData.Exp(i).NSV_.' fnNSV{m} '(locsRef(j)-'
num2str(tBefore) ':locsRef(j)+' num2str(tAfter) ');']);
        end
        for m = 1:numel(fnOpt)
            txt = char(fnOpt(m));
            if txt(1) == 'v'
                iEnd_ = iEnd-1;
            else
                iEnd_ = iEnd;
            end
            myData.Signal(k).Opt.(txt) = myData.Exp(i).Opt_.(txt)(locsRef(j)-tBefore:iEnd_);
%             eval(['myData.Signal(k).Opt.' fnOpt{m} ' = myData.Exp(i).Opt_.' fnOpt{m} '(locsRef(j)-'
num2str(tBefore) ':locsRef(j)+' num2str(tAfter) ');']);
        end

        myData.Signal(k).Type = myData.Exp(i).Type;
        myData.Signal(k).t = myData.Signal(k).Opt.t - myData.Signal(k).Opt.t(1);
        myData.Signal(k).t_filled = myData.Signal(k).Opt.t_filled - myData.Signal(k).Opt.t_filled(1);
    end

    % Save relevant data in the 'Data.Exp(n)' structure
    myData.Exp(i).pksRefData = pksRefData;
    myData.Exp(i).pksRef = pksRef;
    myData.Exp(i).locsRef = locsRef;
    myData.Exp(i).pksOptData = pksOptData;
    myData.Exp(i).pksOpt = pksOpt;
    myData.Exp(i).locsOpt = locsOpt;

end

function myData = splitSineFunc(myData,nExp,mode,config)

fnNSV = fieldnames(myData.Exp(1).NSV_);    % field names for NSV data
fnOpt = fieldnames(myData.Exp(1).Opt_);    % field names for NSV data

switch mode
    case 'Manip'
        % Sine 5/12/2021
        idx =  [7431 51430  59440 103400    110900 129900;...
                6954 50950  58040 102000    108900 152900;...
                7804 51800  58640 102600    110800 148800;...
                5053 49050  56300 100300    107600 151600;...
                7440 51440  59190 102200    109700 153700;...
                1532 45530  52440 85440     103400 126400;...
                6569 50570  57760 101800    109300 153300;...
                5847 33850  57460 101500    109100 153100;...
                7484 51480  58560 102600    110300 154300;...
```

```
                6596 41600  0     0         0      0];
    case 'Lever'
        % LeverSine 23/02/2022
        idx =  [7129 51130  58450 102500    109700 153700;...
                6726 50730  58030 102000    109400 153400;...
                7242 51240  58540 102500    109900 153900;...
                6756 50760  58050 102100    109300 153300;...
                7129 51130  58420 102400    109700 153700;...
                6697 50700  58060 102100    109300 153300;...
                7120 51120  58420 102400    109700 153700;...
                6695 50700  58000 102000    109300 153300;...
                7133 51130  58450 102500    109800 153800;...
                6681 50680  58000 102000    109300 153300];
end

k = 0;
for i = 1:nExp
    nSigE = sum(any(idx(i,:),1))/2; % amount of sine signals in the current experiment/trial series
    for j = 1:nSigE
%         k = j + (i-1)*nSigE;
        k = k + 1;

        for fn = 1:numel(fnNSV)
            txt = char(fnNSV(fn));
            myData.Signal(k).NSV.(txt) = myData.Exp(i).NSV_.(txt)(idx(i,2*j-1):idx(i,2*j));
        end

        for fn = 1:numel(fnOpt)
            txt = char(fnOpt(fn));
            myData.Signal(k).Opt.(txt) = myData.Exp(i).Opt_.(txt)(idx(i,2*j-1):idx(i,2*j));
        end
        myData.Signal(k).t = myData.Signal(k).Opt.t_filled - myData.Signal(k).Opt.t_filled(1);

        myData.Signal(k).Type = myData.Exp(i).Type;
    end

    if config.printConsole
        disp([mode ' sine experiment ' num2str(i) ' has ' num2str(nSigE) ' multisine signals.']);
    end
end


function [data] = rampIddata(data,mode,config)
% Create 'in' and 'out' data variables according to the data type and also
% create iddata objects for the ramp data.

nSig = numel(data.Signal);

for i = 1:nSig

    % create iddata object by specifying output signal, input signal, and sampling time
%     Data.Signal(i).data = iddata([Data.Signal(i).NSV.vRefX Data.Signal(i).NSV.vRefY
Data.Signal(i).NSV.vRefZ],...
%                                  [Data.Signal(i).Opt.vX Data.Signal(i).Opt.vY
Data.Signal(i).Opt.vZ],...
%                                  .001);

%     sigName_ = ['_L' num2str(ceil(i/20)) '_R' num2str(rem(i-1,5)+1)];
    sigNotes_ = [' | Location ' num2str(ceil(i/20)) ...
                 ' | Ramp ' num2str(rem(i-1,5)+1) ...
                 ' | Series ' num2str(ceil(i/5)) ...
                 ' | Trial ' num2str(i)];

    switch data.Signal(i).Type
        case 1
%             sigName = ['Exp' num2str(i) '_X+' sigName_];
            sigNotes = ['posX' sigNotes_];
            data.Signal(i).in = data.Signal(i).NSV.vRefX;
            data.Signal(i).out = data.Signal(i).Opt.vX;
        case 2
%             sigName = ['Exp' num2str(i) '_Y-' sigName_];
            sigNotes = ['negY' sigNotes_];
            data.Signal(i).in = data.Signal(i).NSV.vRefY;
            data.Signal(i).out = data.Signal(i).Opt.vY;
```

```
        case 3
%             sigName = ['Exp' num2str(i) '_X-' sigName_];
            sigNotes = ['negX' sigNotes_];
            data.Signal(i).in = data.Signal(i).NSV.vRefX;
            data.Signal(i).out = data.Signal(i).Opt.vX;
        case 4
%             sigName = ['Exp' num2str(i) '_Y+' sigName_];
            sigNotes = ['posY' sigNotes_];
            data.Signal(i).in = data.Signal(i).NSV.vRefY;
            data.Signal(i).out = data.Signal(i).Opt.vY;
    end

    if config.printConsole; disp(sigNotes); end

    data.Signal(i).EndRefToOptData = iddata(data.Signal(i).NSV.vRefX,data.Signal(i).Opt.vX,.001,...
                                    'Domain','Time',...
                                    'Name','NACT3D Ramp Trials',...
...%                                     'ExperimentName',sigName,...
                                    'Notes',sigNotes,...
                                    'InputName','End Point Reference',...
                                    'InputUnit','mm',...
                                    'OutputName','End Point OptoTrack',...
                                    'OutputUnit','mm');

    if config.printConsole
        disp([mode ' ramp signal ' num2str(i) ' out of ' num2str(nSig) ' processed']);
    end
end

data.EndRefToOptData = merge(data.Signal(:).EndRefToOptData);

function [offset] = calcOffset(data,mode,type)
% Calculate the time offset of the ramp data after alignment.

nSig = numel(data.Signal);

offset = table('Size',[nSig 8],...

'VariableTypes',{'double','double','double','double','double','double','double','double'},...
                'VariableNames',{'x Position','y Position',...
                                 'x Perturbation Type','y Perturbation Type',...
                                 'Initial x Offset','Final x Offset',...
                                 'Initial y Offset','Final y Offset'});
for i = 1:nSig
    % Calculate x- and y-offsets
    switch mode
        case 'Manip'
            iniXoffset = data.Signal(i).Opt.X(1)-data.Signal(i).NSV.RefX(1);
            endXoffset = data.Signal(i).Opt.X(end)-data.Signal(i).NSV.RefX(end);
            iniYoffset = data.Signal(i).Opt.Y(1)-data.Signal(i).NSV.RefY(1);
            endYoffset = data.Signal(i).Opt.Y(end)-data.Signal(i).NSV.RefY(end);
        case 'Lever'
            iniXoffset = data.Signal(i).Opt.wristProjX(1)-data.Signal(i).NSV.RefX(1);
            endXoffset = data.Signal(i).Opt.wristProjX(end)-data.Signal(i).NSV.RefX(end);
            iniYoffset = data.Signal(i).Opt.wristProjY(1)-data.Signal(i).NSV.RefY(1);
            endYoffset = data.Signal(i).Opt.wristProjY(end)-data.Signal(i).NSV.RefY(end);
    end

    switch type
        case 'Ramp'
            xType = -rem(data.Signal(i).Type-2,2);
            yType = rem(data.Signal(i).Type-3,2);
        case 'Sine'
            xType = -rem(data.Signal(i).Type-2,2);
            yType = -rem(data.Signal(i).Type-3,2);
    end

    % Place offsets in table
    offset(i,:) = array2table(round(...
                  [data.Signal(i).NSV.RefX(1),...
                   data.Signal(i).NSV.RefY(1),...
                   xType,...
                   yType,...
                   iniXoffset,...
```

```
                    endXoffset,...
                    iniYoffset,...
                    endYoffset]));
end
```

F.5. sysID.m

```
function sys = sysID(sys, data, mode, fSin)

txtLoc = {'L1' 'L2' 'L3' 'L4' 'L5'};
txtDir = {'DX' 'DY'};

switch mode
    case 'Manip'
        txtMode = {'Manip'};
        % Rx & Ry = reference/cursor x & y
        % Ox & Oy = observed/OptoTrak x & y
        % Ex & Ey = estimated x & y based on measured servo positions
        txtIO   = {'RxOx'  'RyOy'  'RxEx'  'RyEy'  'ExOx'  'EyOy'  };
        inType  = {'NSV'    'NSV'   'NSV'   'NSV'   'NSV'   'NSV'   };
        outType = {'Opt'    'Opt'   'NSV'   'NSV'   'Opt'   'Opt'   };
%         in      = {'vRefX'  'vRefY' 'vRefX' 'vRefY' 'vEstX' 'vEstY' };
%         out     = {'vX'     'vY'    'vEstX' 'vEstY' 'vX'    'vY'    };
        in      = {'RefX'  'RefY' 'RefX' 'RefY' 'EstX' 'EstY' };
        out     = {'X'     'Y'    'EstX' 'EstY' 'X'    'Y'    };
        na = 9:18;
        nb = 0:4;
        nk = 0;
        nnRange = struc(na,nb,nk);
    case 'Lever'
        txtMode = {'Lever'};
        % Rx & Ry = reference/cursor x & y
        % Ox & Oy = observed/OptoTrak x & y
        % Ex & Ey = estimated x & y based on measured servo positions
        txtIO   = {'RxOx'           'RyOy'          'RxEx'  'RyEy'  'ExOx'          'EyOy'          };
        inType  = {'NSV'            'NSV'           'NSV'   'NSV'   'NSV'           'NSV'           };
        outType = {'Opt'            'Opt'           'NSV'   'NSV'   'Opt'           'Opt'           };
%         in      = {'vRefX'          'vRefY'         'vRefX' 'vRefY' 'vEstX'         'vEstY'
};
%         out     = {'vwristProjX'    'vwristProjY'   'vEstX' 'vEstY' 'vwristProjX'  'vwristProjY'   };
        in      = {'RefX'          'RefY'         'RefX' 'RefY' 'EstX'         'EstY'         };
        out     = {'wristProjX'    'wristProjY'   'EstX' 'EstY' 'wristProjX'  'wristProjY'    };
        na = 4:8;
        nb = 0:2;
        nk = 0;
        nnRange = struc(na,nb,nk);
    case 'SPACAR'
        txtMode = {'SPACAR'};
        txtIO = {'u1y1' 'u1y2' 'u2y1' 'u2y2'};
        in  = {'servoShoulderXd' 'servoShoulderXd' 'servoElbowXd' 'servoElbowXd'};
        out = {'endXd'           'endYd'           'endXd'        'endYd'};
        na = 9:18;
        nb = 0:4;
        nk = 0;
        nnRange = struc(na,nb,nk);
end


threshold = -3;  % [dB] FRF bandwidth above this magnitude

dt  = 0.001;    % [s] sample time
fs  = 1/dt;     % [Hz] sample frequency
f   = 0:(fs-1); % [Hz] frequency vector
% N=8000;         % # of samples
% T=N*dt;         % [s] observation time
% t=(0:N-1).'*dt; % time vector

% % do not forget to smooth: Welch' method!
% D=8;        % number of (independent) segments used for Welch
% nfft=N/D;   % length of the segments




i = 1;
```

```
for Loc = 1:numel(txtLoc)
    for Dir = 1:numel(txtDir)
        for k = 1:numel(txtIO)

            tM  = char(txtMode);
            tL  = char(txtLoc(Loc));
            tD  = char(txtDir(Dir));
            tIO = char(txtIO(k));
            tiT = char(inType(k));
            toT = char(outType(k));
            ti  = char(in(k));
            to  = char(out(k));

            for j = 1:3

                input   = data.Signal(i).(tiT).(ti);
                output  = data.Signal(i).(toT).(to);

                N   = numel(input); % [ ] # of samples
                T   = N*dt;           % [s] observation time
                t   = (0:N-1).'*dt;   % [s] time vector

                % Bandwidth ID
                CrossSD(j,:)   = cpsd(output,input, rectwin(fs), 0, f, fs);
                AutoSD(j,:)     = pwelch(input, rectwin(fs), 0, f, fs);
                H(j,:)          = CrossSD(j,:) ./ AutoSD(j,:);

                sys.(tM).(tL).(tD).sys.(tIO).Soi(j,:) = CrossSD(j,:);
                sys.(tM).(tL).(tD).sys.(tIO).Sii(j,:) = AutoSD(j,:);
                sys.(tM).(tL).(tD).sys.(tIO).H(j,:)     = H(j,:);
            end

            CrossSDavg = sum(CrossSD(:,:))/size(CrossSD,1);
            AutoSDavg  = sum(AutoSD(:,:))/size(AutoSD,1);
            Havg        = CrossSDavg ./ AutoSDavg;

            mag = pow2db(abs(Havg(1,fSin+1)));          % [dB] system magnitude for perturbed
frequencies
            ang = unwrap(angle(Havg(1,fSin+1)))*180/pi; % [degrees] system phase for perturbed
frequencies (unwrapped to stay within 360 degrees)



            % Find system bandwidth by interpolating where the magnitude
            % crosses the threshold.
            mag_    = mag - threshold;                  % Theshold value is now zero
            fZC     = @(x) find(diff(sign(mag_))<0);    % function to find indices of zero crossings
            iZC     = fZC(mag_);                        % indices of zero crossings
            fInterp = @(x1,y1,x2,y2) x1 - (y1.*(x1 - x2))./(y1 - y2);   %function to interpolate the
bandwidth frequency
            BW      = fInterp(fSin(iZC),mag_(iZC),fSin(iZC+1),mag_(iZC+1)); % [power] bandwidth

            % Only take the first bandwidth entry (in some cases the
            % magnitude crosses the threshold more than once)
            if numel(BW)>1; BW = BW(1); end

            % Save to data structure
            sys.(tM).(tL).(tD).sys.(tIO).CrossSDavg   = CrossSDavg;
            sys.(tM).(tL).(tD).sys.(tIO).AutoSDavg= AutoSDavg;
            sys.(tM).(tL).(tD).sys.(tIO).Havg       = Havg;

            sys.(tM).(tL).(tD).sys.(tIO).Mag   = mag;
            sys.(tM).(tL).(tD).sys.(tIO).Ang   = ang;
            sys.(tM).(tL).(tD).sys.(tIO).BW     = BW;
        end

        i = i + 3;
    end
end
```

F.6. sys2excel.m

```
function [bwTable,bwROTable,bwEOTable] = sys2excel(data)
% This function takes the data structure in which the system information is
% stored and writes the relevant information to excel files. Three tables
% are created:
%   bandwidthTable        - Contains the bandwidth for all calculated
%                           systems.
%   bandwidthRefObsTable  - Contains the bandwidth for the systems with the
%                           reference cursor signal as input and the
%                           observed optotrak signal as output. It also
%                           only contains the bandwidth in the x-direction
%                           for x-perturbation experiments and the
%                           bandwidth in the y-direction for y-perturbation
%                           experiments.
%   bandwidthEstObsTable  - Contains the bandwidth for the systems with the
%                           estimated end point signal, based on the servo
%                           signal, as input and the observed optotrak
%                           signal as output. It also only contains the
%                           bandwidth in the x-direction for x-perturbation
%                           experiments and the bandwidth in the
%                           y-direction for y-perturbation experiments.

% Define header row for all bandwidth tables
bwTableHeader = {'Mode' 'Location' 'Perturbation' 'System' 'Bandwidth'};

% Get and store the different experiment mode, location, perturbation, and
% identified system names in variables.
tM  = fieldnames(data);                                                  % modes
tL  = fieldnames(data.(char(tM(1))));                                    % locations
tD  = fieldnames(data.(char(tM(1))).(char(tL(1))));                      % perturbations
tIO = fieldnames(data.(char(tM(1))).(char(tL(1))).(char(tD(1))).sys);     % identified systems

%% Bandwidth table with all bandwidth information
% Define the format of the bandwidth table containing all calculated
% bandwidth information.
bwTable = table('Size',[1 5],'VariableTypes',{'string' 'string' 'string' 'string' 'string'});

% Write the header row
bwTable(1,:) = bwTableHeader;

% Copy the tables format with header row for all tables
bwROTable = bwTable;
bwEOTable = bwTable;

% Cycle through all different experiment modes
for M = 1:numel(tM)
    % Cycle through all different experiment locations
    for L = 1:numel(tL)
        % Cycle through all different experiment perturbations
        for D = 1:numel(tD)
            % Cycle through all different identified systems
            for IO = 1:numel(tIO)
                % Determine the index of the current bandwidth table entry
                % (add 1 for the header row)
                i = (M-1)*numel(tL)*numel(tD)*numel(tIO) + (L-1)*numel(tD)*numel(tIO) + (D-
1)*numel(tIO) + IO + 1;

                % Get the current bandwidth
                BandWidth = data.(char(tM(M))).(char(tL(L))).(char(tD(D))).sys.(char(tIO(IO))).BW;

                % Round the bandwidth to two numbers after the decimal
                BandWidth = round(BandWidth,2);

                % If there is no calcualted bandwidth define with '-',
                % otherwise convert calculated bandwidth to string
                if isempty(BandWidth)
                    BandWidth = '-';
                else
                    BandWidth = num2str(BandWidth);
                end
```

```
                % Define new table row entry
                newEntry = {tM(M),tL(L),tD(D),tIO(IO),BandWidth};

                % Write bandwidth to table
                bwTable(i,:) = newEntry;

                % If the perturbation is in the x-direction and the system
                % bandwidth is from reference to observed in the
                % x-direction then append to bwROTable. Do the same for
                % y-perturbations and bandwidths in the y-direction. Also
                % do the same two operations for system bandwidths from
                % estimated to observed signals.
                if strcmp(tD(D),'DX') && strcmp(tIO(IO),'RxOx')
                    bwROTable = [bwROTable;newEntry];
                end
                if strcmp(tD(D),'DY') && strcmp(tIO(IO),'RyOy')
                    bwROTable = [bwROTable;newEntry];
                end
                if strcmp(tD(D),'DX') && strcmp(tIO(IO),'ExOx')
                    bwEOTable = [bwEOTable;newEntry];
                end
                if strcmp(tD(D),'DY') && strcmp(tIO(IO),'EyOy')
                    bwEOTable = [bwEOTable;newEntry];
                end
            end
        end
    end
end

% Define the name for the excel document
bwTableName = 'bandwidthTable.xlsx';

% If excel document already exists remove the file to overwrite it
if exist(bwTableName, 'file')==2
  delete(bwTableName);
end

% Write bandwidth data to excel file
writetable(bwTable,bwTableName,'WriteVariableNames',false,'Sheet','All Bandwidths');
writetable(bwROTable,bwTableName,'WriteVariableNames',false,'Sheet','Reference to Observed');
writetable(bwEOTable,bwTableName,'WriteVariableNames',false,'Sheet','Estimated to Observed');
```

F.7. thesisPlots.m

```
%% General Figure Parameters
par = struct;

par.downSample      = false;    % Determine if data should be downsampled
par.downSampleRate  = 100;      % Downsample rate

par.alterFont   = true;    % Determine if font size should be changed

par.fontTitle   = 22;   % Font size for title
par.fontAxis    = 18;   % Font size for text along axis
par.fontLegend  = 18;   % Font size for legend

par.lineStyle = {'-' ':' '-.' '--'};    % Line styles for readability
par.lineWidth = 3;                      % Line thickness for readability

par.figPositionTime = [687   455   799   585];
par.figPositionFreq = [687   300   1000   650];


%% Ramp Perturbations

% Initialize ramp experiment data structure
if ~exist('rampPlot','var')
    rampPlot = struct;
end

% Get source data to plot
rampPlot.src     = dataManipRamp.Exp(1);

% Set domain setting of data for plotting;
rampPlot.domain = 'time';

% Determine indeces of data to plot
rampPlot.iNSV    = [rampPlot.src.locsRef(1)-500, rampPlot.src.locsRef(2)-500];
rampPlot.iOpt    = [rampPlot.src.locsOpt(1)-500, rampPlot.src.locsOpt(2)-500];

% Extract time
rampPlot.t   = rampPlot.src.NSV.TimeStamp(rampPlot.iNSV(1):rampPlot.iNSV(2));
rampPlot.t   = (rampPlot.t - rampPlot.t(1))/1000;  % Start at 0 and convert to seconds

% Extract data
rampPlot.name(1)        = {'Reference'};
rampPlot.xRaw(1,:)      = rampPlot.src.NSV.RefX(rampPlot.iNSV(1):rampPlot.iNSV(2));
rampPlot.yRaw(1,:)      = rampPlot.src.NSV.RefY(rampPlot.iNSV(1):rampPlot.iNSV(2));
rampPlot.xAlign(1,:)    = rampPlot.src.NSV_.RefX(rampPlot.iNSV(1):rampPlot.iNSV(2));
rampPlot.yAlign(1,:)    = rampPlot.src.NSV_.RefY(rampPlot.iNSV(1):rampPlot.iNSV(2));

rampPlot.name(2)        = {'Estimated'};
rampPlot.xRaw(2,:)      = rampPlot.src.NSV.EstX(rampPlot.iNSV(1):rampPlot.iNSV(2));
rampPlot.yRaw(2,:)      = rampPlot.src.NSV.EstY(rampPlot.iNSV(1):rampPlot.iNSV(2));
rampPlot.xAlign(2,:)    = rampPlot.src.NSV_.EstX(rampPlot.iNSV(1):rampPlot.iNSV(2));
rampPlot.yAlign(2,:)    = rampPlot.src.NSV_.EstY(rampPlot.iNSV(1):rampPlot.iNSV(2));

rampPlot.name(3)        = {'Observed'};
rampPlot.xRaw(3,:)      = [rampPlot.src.Opt.X(rampPlot.iOpt(1):rampPlot.iOpt(2)-1)]';
rampPlot.yRaw(3,:)      = [rampPlot.src.Opt.Y(rampPlot.iOpt(1):rampPlot.iOpt(2)-1)]';
rampPlot.xAlign(3,:)    = rampPlot.xRaw(3,:);
rampPlot.yAlign(3,:)    = rampPlot.yRaw(3,:);

% Set figure properties
par.xAxis = 'Time [s]';
par.locLegend = 'northeast';

% Ramp Raw x-Plot Location 1 X Perturbation
par.title = 'Raw X Displacement for Ramp in X Direction';
par.yAxis = 'X Position [mm]';
rampPlot = dataPlot(rampPlot,rampPlot.xRaw,'xRaw',par);
rampPlot.xRawax1.YLim = [rampPlot.xRawax1.YLim(1) (rampPlot.xRawax1.YLim(2)+10)];

% Ramp Raw y-Plot Location 1 X Perturbation
```

```
par.title = 'Raw Y Displacement for Ramp in X Direction';
par.yAxis = 'Y Position [mm]';
rampPlot = dataPlot(rampPlot,rampPlot.yRaw,'yRaw',par);
rampPlot.yRawax1.YLim = [rampPlot.yRawax1.YLim(1) (rampPlot.yRawax1.YLim(2)+5)];

% Ramp Aligned x-Plot Location 1 X Perturbation
par.title = 'Aligned X Displacement for Ramp in X Direction';
par.yAxis = 'X Position [mm]';
rampPlot = dataPlot(rampPlot,rampPlot.xAlign,'xAlign',par);
rampPlot.xAlignax1.YLim = [rampPlot.xAlignax1.YLim(1) (rampPlot.xAlignax1.YLim(2)+10)];

% Ramp Aligned y-Plot Location 1 X Perturbation
par.title = 'Aligned Y Displacement for Ramp in X Direction';
par.yAxis = 'Y Position [mm]';
rampPlot = dataPlot(rampPlot,rampPlot.yAlign,'yAlign',par);
rampPlot.yAlignax1.YLim = [rampPlot.yAlignax1.YLim(1) (rampPlot.yAlignax1.YLim(2)+5)];

%% MultiSine Perturbations
frequencies = [1 2 3 4 5 6 7 8 12 16 20 24 28 32 40 48 56 64 80 96 112 128];
threshold   = ones(1,length(frequencies))*-3;

%% Manipulator Time Data X Perturbation 3 Raw Trials

% Get source data to plot
multPlotMX.src     = dataManipSine.Exp(1);
n = min([length(multPlotMX.src.NSV.RefX) length(multPlotMX.src.Opt.X)]);

% Set domain setting of data for plotting;
multPlotMX.domain = 'time';

% Extract time
multPlotMX.t   = multPlotMX.src.NSV.TimeStamp(1:n);
multPlotMX.t   = (multPlotMX.t - multPlotMX.t(1))/1000;  % Start at 0 and convert to seconds

% Extract data
multPlotMX.name(1)        = {'Reference'};
multPlotMX.xRaw3(1,:)      = multPlotMX.src.NSV.RefX(1:n);

multPlotMX.name(2)        = {'Estimated'};
multPlotMX.xRaw3(2,:)      = multPlotMX.src.NSV.EstX(1:n);

multPlotMX.name(3)        = {'Observed'};
multPlotMX.xRaw3(3,:)      = [multPlotMX.src.Opt.X(1:n)]';

clear n

% Set figure properties
par.xAxis = 'Time [s]';
par.locLegend = 'northeast';

% MultiSine Raw x-Plot Location 1 X Perturbation
par.lineWidth = 2;                      % Line thickness for readability
par.title = 'Raw X Displacement for First X Multi Sine Trial';
par.yAxis = 'X Position [mm]';
multPlotMX = dataPlot(multPlotMX,multPlotMX.xRaw3,'xRaw3',par);
multPlotMX.xRaw3fig.Position = [687   300   1200   500];

%% Manipulator Time Data X Perturbation

% Get source data to plot
multPlotMX.src     = dataManipSine.Signal(1);

% Set domain setting of data for plotting;
multPlotMX.domain = 'time';

% Extract time
multPlotMX.t   = multPlotMX.src.NSV.t;
multPlotMX.t   = (multPlotMX.t - multPlotMX.t(1))/1000;  % Start at 0 and convert to seconds

% Extract data
multPlotMX.name(1)        = {'Reference'};
multPlotMX.xRaw(1,:)      = multPlotMX.src.NSV.RefX;
multPlotMX.yRaw(1,:)      = multPlotMX.src.NSV.RefY;
multPlotMX.xOne(1,:)      = multPlotMX.src.NSV.RefX(1:1001);
```

```
multPlotMX.yOne(1,:)      = multPlotMX.src.NSV.RefY(1:1001);

multPlotMX.name(2)        = {'Estimated'};
multPlotMX.xRaw(2,:)      = multPlotMX.src.NSV.EstX;
multPlotMX.yRaw(2,:)      = multPlotMX.src.NSV.EstY;
multPlotMX.xOne(2,:)      = multPlotMX.src.NSV.EstX(1:1001);
multPlotMX.yOne(2,:)      = multPlotMX.src.NSV.EstY(1:1001);

multPlotMX.name(3)        = {'Observed'};
multPlotMX.xRaw(3,:)      = [multPlotMX.src.Opt.X]';
multPlotMX.yRaw(3,:)      = [multPlotMX.src.Opt.Y]';
multPlotMX.xOne(3,:)      = [multPlotMX.src.Opt.X(1:1001)]';
multPlotMX.yOne(3,:)      = [multPlotMX.src.Opt.Y(1:1001)]';

% Set figure properties
par.xAxis = 'Time [s]';
par.locLegend = 'northeast';

% MultiSine Raw x-Plot Location 1 X Perturbation
par.lineWidth = 1;                      % Line thickness for readability
par.title = 'Raw X Displacement for First X Multi Sine Trial';
par.yAxis = 'X Position [mm]';
multPlotMX = dataPlot(multPlotMX,multPlotMX.xRaw,'xRaw',par);

% Multi Sine Raw x-Plot Location 1 X Perturbation for one period
multPlotMX.t = multPlotMX.t(1:1001);    % Amount of millisecond in one period of lowest freq
par.lineWidth = 3;                  % Line thickness for readability
par.title = 'x Displacement for Multi-Sine x Perturbation';
par.yAxis = 'X Position [mm]';
multPlotMX = dataPlot(multPlotMX,multPlotMX.xOne,'xOne',par);

% Multi Sine Raw y-Plot Location 1 X Perturbation for one period
par.title = 'y Displacement for Multi-Sine x Perturbation';
par.yAxis = 'Y Position [mm]';
par.locLegend = 'southeast';
multPlotMX = dataPlot(multPlotMX,multPlotMX.yOne,'yOne',par);

%% Manipulator Time Data Y Perturbation

% Get source data to plot
multPlotMY.src     = dataManipSine.Signal(4);

% Set domain setting of data for plotting;
multPlotMY.domain = 'time';

% Extract time
multPlotMY.t   = multPlotMY.src.NSV.t;
multPlotMY.t   = (multPlotMY.t - multPlotMY.t(1))/1000;  % Start at 0 and convert to seconds

% Extract data
multPlotMY.name(1)        = {'Reference'};
multPlotMY.xOne(1,:)      = multPlotMY.src.NSV.RefX(1:1001);
multPlotMY.yOne(1,:)      = multPlotMY.src.NSV.RefY(1:1001);

multPlotMY.name(2)        = {'Estimated'};
multPlotMY.xOne(2,:)      = multPlotMY.src.NSV.EstX(1:1001);
multPlotMY.yOne(2,:)      = multPlotMY.src.NSV.EstY(1:1001);

multPlotMY.name(3)        = {'Observed'};
multPlotMY.xOne(3,:)      = [multPlotMY.src.Opt.X(1:1001)]';
multPlotMY.yOne(3,:)      = [multPlotMY.src.Opt.Y(1:1001)]';

% Set figure properties
par.xAxis = 'Time [s]';

% Multi Sine Raw x-Plot Location 1 Y Perturbation for one period
multPlotMY.t = multPlotMY.t(1:1001);    % Amount of millisecond in one period of lowest freq
par.lineWidth = 3;                  % Line thickness for readability
par.title = 'x Displacement for Multi-Sine y Perturbation';
par.yAxis = 'X Position [mm]';
par.locLegend = 'northwest';
multPlotMY = dataPlot(multPlotMY,multPlotMY.xOne,'xOne',par);

% Multi Sine Raw y-Plot Location 1 Y Perturbation for one period
```

```
par.title = 'y Displacement for Multi-Sine y Perturbation';
par.yAxis = 'Y Position [mm]';
par.locLegend = 'north';
multPlotMY = dataPlot(multPlotMY,multPlotMY.yOne,'yOne',par);

%% Lever Time Data X Perturbation

% Get source data to plot
multPlotLX.src     = dataLeverSine.Signal(1);

% Set domain setting of data for plotting;
multPlotLX.domain = 'time';

% Extract time
multPlotLX.t   = multPlotLX.src.NSV.t;
multPlotLX.t   = (multPlotLX.t - multPlotLX.t(1))/1000;  % Start at 0 and convert to seconds

% Extract data
multPlotLX.name(1)        = {'Reference'};
multPlotLX.xOne(1,:)      = multPlotLX.src.NSV.RefX(1:1001);
multPlotLX.yOne(1,:)      = multPlotLX.src.NSV.RefY(1:1001);

multPlotLX.name(2)        = {'Estimated'};
multPlotLX.xOne(2,:)      = multPlotLX.src.NSV.EstX(1:1001);
multPlotLX.yOne(2,:)      = multPlotLX.src.NSV.EstY(1:1001);

multPlotLX.name(3)        = {'Observed'};
multPlotLX.xOne(3,:)      = [multPlotLX.src.Opt.wristProjX(1:1001)]';
multPlotLX.yOne(3,:)      = [multPlotLX.src.Opt.wristProjY(1:1001)]';

% Set figure properties
par.xAxis = 'Time [s]';

% Multi Sine Raw x-Plot Location 1 X Perturbation for one period
multPlotLX.t = multPlotLX.t(1:1001);    % Amount of millisecond in one period of lowest freq
par.lineWidth = 3;                  % Line thickness for readability
par.title = 'x Displacement for Multi-Sine x Perturbation';
par.yAxis = 'X Position [mm]';
par.locLegend = 'north';
multPlotLX = dataPlot(multPlotLX,multPlotLX.xOne,'xOne',par);

% Multi Sine Raw y-Plot Location 1 X Perturbation for one period
par.title = 'y Displacement for Multi-Sine x Perturbation';
par.yAxis = 'Y Position [mm]';
par.locLegend = 'east';
multPlotLX = dataPlot(multPlotLX,multPlotLX.yOne,'yOne',par);

%% Lever Time Data Y Perturbation

% Get source data to plot
multPlotLY.src     = dataLeverSine.Signal(4);

% Set domain setting of data for plotting;
multPlotLY.domain = 'time';

% Extract time
multPlotLY.t   = multPlotLY.src.NSV.t;
multPlotLY.t   = (multPlotLY.t - multPlotLY.t(1))/1000;  % Start at 0 and convert to seconds

% Extract data
multPlotLY.name(1)        = {'Reference'};
multPlotLY.xOne(1,:)      = multPlotLY.src.NSV.RefX(1:1001);
multPlotLY.yOne(1,:)      = multPlotLY.src.NSV.RefY(1:1001);

multPlotLY.name(2)        = {'Estimated'};
multPlotLY.xOne(2,:)      = multPlotLY.src.NSV.EstX(1:1001);
multPlotLY.yOne(2,:)      = multPlotLY.src.NSV.EstY(1:1001);

multPlotLY.name(3)        = {'Observed'};
multPlotLY.xOne(3,:)      = [multPlotLY.src.Opt.wristProjX(1:1001)]';
multPlotLY.yOne(3,:)      = [multPlotLY.src.Opt.wristProjY(1:1001)]';

% Set figure properties
par.xAxis = 'Time [s]';
```

```
% Multi Sine Raw x-Plot Location 1 Y Perturbation for one period
multPlotLY.t = multPlotLY.t(1:1001);    % Amount of millisecond in one period of lowest freq
par.lineWidth = 3;                  % Line thickness for readability
par.title = 'x Displacement for Multi-Sine y Perturbation';
par.yAxis = 'X Position [mm]';
par.locLegend = 'northwest';
multPlotLY = dataPlot(multPlotLY,multPlotLY.xOne,'xOne',par);

% Multi Sine Raw y-Plot Location 1 Y Perturbation for one period
par.title = 'y Displacement for Multi-Sine y Perturbation';
par.yAxis = 'Y Position [mm]';
par.locLegend = 'north';
multPlotLY = dataPlot(multPlotLY,multPlotLY.yOne,'yOne',par);

%% FRF Manipulator location 1 x perturbation
% MSML1X:
% MS = Multi-Sine
% M  = Manipulator
% L1 = Location 1
% X  = x-perturbation

par.xAxis = 'Frequency [Hz]';
par.yAxisMag = 'Magnitude [dB]';
par.yAxisPha = 'Phase [degrees]';

% Initialize Multi Sine Lever Location 1 X Perturbation structure
if ~exist('MSML1X','var')
    MSML1X = struct;
end

% Set domain and x-data
MSML1X.domain = 'frequency';
MSML1X.f      = frequencies;

% Get data
MSML1X.name(1)  = {'Ref -> Obs'};
MSML1X.xFRF.Havg(1,:) = sys.Manip.L1.DX.sys.RxOx.Havg(1,MSML1X.f+1);
MSML1X.yFRF.Havg(1,:) = sys.Manip.L1.DX.sys.RyOy.Havg(1,MSML1X.f+1);

MSML1X.name(2)  = {'Ref -> Est'};
MSML1X.xFRF.Havg(2,:) = sys.Manip.L1.DX.sys.RxEx.Havg(1,MSML1X.f+1);
MSML1X.yFRF.Havg(2,:) = sys.Manip.L1.DX.sys.RyEy.Havg(1,MSML1X.f+1);

MSML1X.name(3)  = {'Est -> Obs'};
MSML1X.xFRF.Havg(3,:) = sys.Manip.L1.DX.sys.ExOx.Havg(1,MSML1X.f+1);
MSML1X.yFRF.Havg(3,:) = sys.Manip.L1.DX.sys.EyOy.Havg(1,MSML1X.f+1);

MSML1X.name(4)  = {'Threshold'};
MSML1X.thres = threshold;

% Bode plot for x-direction
par.title = 'x Dimenison FRF of Manipulator Multi-Sine in x Direction';
par.locLegend = 'southwest';

MSML1X = dataPlot(MSML1X,MSML1X.xFRF.Havg,'msFRFx',par);
MSML1X.msFRFxax1.YLim = [MSML1X.msFRFxax1.YLim(1)...
    MSML1X.msFRFxax1.YLim(2)+sum(abs(MSML1X.msFRFxax1.YLim))*0.1];

% Bode plot for y-direction
par.title = 'y Dimenison FRF of Manipulator Multi-Sine in x Direction';
par.locLegend = 'northeast';

MSML1X = dataPlot(MSML1X,MSML1X.yFRF.Havg,'msFRFy',par);
MSML1X.msFRFyax1.YLim = [MSML1X.msFRFyax1.YLim(1)...
    MSML1X.msFRFyax1.YLim(2)+sum(abs(MSML1X.msFRFyax1.YLim))*0.1];

%% FRF Manipulator location 1 y perturbation
% MSML1Y:
% MS = Multi-Sine
% M  = Manipulator
% L1 = Location 1
% Y  = y-perturbation
```

```
par.xAxis = 'Frequency [Hz]';
par.yAxisMag = 'Magnitude [dB]';
par.yAxisPha = 'Phase [degrees]';

% Initialize Multi Sine Lever Location 1 X Perturbation structure
if ~exist('MSML1Y','var')
    MSML1Y = struct;
end

% Set domain and x-data
MSML1Y.domain = 'frequency';
MSML1Y.f      = frequencies;

% Get data
MSML1Y.name(1)  = {'Ref -> Obs'};
MSML1Y.xFRF.Havg(1,:) = sys.Manip.L1.DY.sys.RxOx.Havg(1,MSML1Y.f+1);
MSML1Y.yFRF.Havg(1,:) = sys.Manip.L1.DY.sys.RyOy.Havg(1,MSML1Y.f+1);

MSML1Y.name(2)  = {'Ref -> Est'};
MSML1Y.xFRF.Havg(2,:) = sys.Manip.L1.DY.sys.RxEx.Havg(1,MSML1Y.f+1);
MSML1Y.yFRF.Havg(2,:) = sys.Manip.L1.DY.sys.RyEy.Havg(1,MSML1Y.f+1);

MSML1Y.name(3)  = {'Est -> Obs'};
MSML1Y.xFRF.Havg(3,:) = sys.Manip.L1.DY.sys.ExOx.Havg(1,MSML1Y.f+1);
MSML1Y.yFRF.Havg(3,:) = sys.Manip.L1.DY.sys.EyOy.Havg(1,MSML1Y.f+1);

MSML1Y.name(4)  = {'Threshold'};
MSML1Y.thres = threshold;

% Bode plot for x-direction
par.title = 'x Dimenison FRF of Manipulator Multi-Sine in y Direction';
par.locLegend = 'southwest';

MSML1Y = dataPlot(MSML1Y,MSML1Y.xFRF.Havg,'msFRFx',par);
MSML1Y.msFRFxax1.YLim = [MSML1Y.msFRFxax1.YLim(1)...
    MSML1Y.msFRFxax1.YLim(2)+sum(abs(MSML1Y.msFRFxax1.YLim))*0.1];

% Bode plot for y-direction
par.title = 'y Dimenison FRF of Manipulator Multi-Sine in y Direction';
par.locLegend = 'northeast';

MSML1Y = dataPlot(MSML1Y,MSML1Y.yFRF.Havg,'msFRFy',par);
MSML1Y.msFRFyax1.YLim = [MSML1Y.msFRFyax1.YLim(1)...
    MSML1Y.msFRFyax1.YLim(2)+sum(abs(MSML1Y.msFRFyax1.YLim))*0.1];

%% FRF Lever location 1 x perturbation
% MSLL1X:
% MS = Multi-Sine
% L  = Lever
% L1 = Location 1
% X  = x-perturbation

par.xAxis = 'Frequency [Hz]';
par.yAxisMag = 'Magnitude [dB]';
par.yAxisPha = 'Phase [degrees]';

% Initialize Multi Sine Lever Location 1 X Perturbation structure
if ~exist('MSLL1X','var')
    MSLL1X = struct;
end

% Set domain and x-data
MSLL1X.domain = 'frequency';
MSLL1X.f      = frequencies;

% Get data
MSLL1X.name(1)  = {'Ref -> Obs'};
MSLL1X.xFRF.Havg(1,:) = sys.Lever.L1.DX.sys.RxOx.Havg(1,MSLL1X.f+1);
MSLL1X.yFRF.Havg(1,:) = sys.Lever.L1.DX.sys.RyOy.Havg(1,MSLL1X.f+1);

MSLL1X.name(2)  = {'Ref -> Est'};
MSLL1X.xFRF.Havg(2,:) = sys.Lever.L1.DX.sys.RxEx.Havg(1,MSLL1X.f+1);
MSLL1X.yFRF.Havg(2,:) = sys.Lever.L1.DX.sys.RyEy.Havg(1,MSLL1X.f+1);
```

```
MSLL1X.name(3)  = {'Est -> Obs'};
MSLL1X.xFRF.Havg(3,:) = sys.Lever.L1.DX.sys.ExOx.Havg(1,MSLL1X.f+1);
MSLL1X.yFRF.Havg(3,:) = sys.Lever.L1.DX.sys.EyOy.Havg(1,MSLL1X.f+1);

MSLL1X.name(4)  = {'Threshold'};
MSLL1X.thres = threshold;

% Bode plot for x-direction
par.title = 'x Dimenison FRF of Lever Multi-Sine in x Direction';
par.locLegend = 'southwest';

MSLL1X = dataPlot(MSLL1X,MSLL1X.xFRF.Havg,'msFRFx',par);
MSLL1X.msFRFxax1.YLim = [MSLL1X.msFRFxax1.YLim(1)...
    MSLL1X.msFRFxax1.YLim(2)+sum(abs(MSLL1X.msFRFxax1.YLim))*0.1];

% Bode plot for y-direction
par.title = 'y Dimenison FRF of Lever Multi-Sine in x Direction';
par.locLegend = 'northeast';

MSLL1X = dataPlot(MSLL1X,MSLL1X.yFRF.Havg,'msFRFy',par);
MSLL1X.msFRFyax1.YLim = [MSLL1X.msFRFyax1.YLim(1)...
    MSLL1X.msFRFyax1.YLim(2)+sum(abs(MSLL1X.msFRFyax1.YLim))*0.1];

%% FRF Lever location 1 y perturbation
% MSLL1Y:
% MS = Multi-Sine
% L  = Lever
% L1 = Location 1
% Y  = y-perturbation

par.xAxis = 'Frequency [Hz]';
par.yAxisMag = 'Magnitude [dB]';
par.yAxisPha = 'Phase [degrees]';

% Initialize Multi Sine Lever Location 1 X Perturbation structure
if ~exist('MSLL1Y','var')
    MSLL1Y = struct;
end

% Set domain and x-data
MSLL1Y.domain = 'frequency';
MSLL1Y.f      = frequencies;

% Get data
MSLL1Y.name(1)  = {'Ref -> Obs'};
MSLL1Y.xFRF.Havg(1,:) = sys.Lever.L1.DY.sys.RxOx.Havg(1,MSLL1Y.f+1);
MSLL1Y.yFRF.Havg(1,:) = sys.Lever.L1.DY.sys.RyOy.Havg(1,MSLL1Y.f+1);

MSLL1Y.name(2)  = {'Ref -> Est'};
MSLL1Y.xFRF.Havg(2,:) = sys.Lever.L1.DY.sys.RxEx.Havg(1,MSLL1Y.f+1);
MSLL1Y.yFRF.Havg(2,:) = sys.Lever.L1.DY.sys.RyEy.Havg(1,MSLL1Y.f+1);

MSLL1Y.name(3)  = {'Est -> Obs'};
MSLL1Y.xFRF.Havg(3,:) = sys.Lever.L1.DY.sys.ExOx.Havg(1,MSLL1Y.f+1);
MSLL1Y.yFRF.Havg(3,:) = sys.Lever.L1.DY.sys.EyOy.Havg(1,MSLL1Y.f+1);

MSLL1Y.name(4)  = {'Threshold'};
MSLL1Y.thres = threshold;

% Bode plot for x-direction
par.title = 'x Dimenison FRF of Lever Multi-Sine in y Direction';
par.locLegend = 'southwest';

MSLL1Y = dataPlot(MSLL1Y,MSLL1Y.xFRF.Havg,'msFRFx',par);
MSLL1Y.msFRFxax1.YLim = [MSLL1Y.msFRFxax1.YLim(1)...
    MSLL1Y.msFRFxax1.YLim(2)+sum(abs(MSLL1Y.msFRFxax1.YLim))*0.1];

% Bode plot for y-direction
par.title = 'y Dimenison FRF of Lever Multi-Sine in y Direction';
par.locLegend = 'southwest';

MSLL1Y = dataPlot(MSLL1Y,MSLL1Y.yFRF.Havg,'msFRFy',par);
MSLL1Y.msFRFyax1.YLim = [MSLL1Y.msFRFyax1.YLim(1)...
    MSLL1Y.msFRFyax1.YLim(2)+sum(abs(MSLL1Y.msFRFyax1.YLim))*0.1];
```

```
%% Functions

function [exp] = dataPlot(exp,data,figType,config)

    n = size(data,1);   % Determine amount of data lines to plot

    figName = [figType 'fig'];

    existPlot = isfield(exp,figName) && isvalid(exp.(figName)); % Boolen whether figure already exists

    if existPlot == 1
        fig = exp.(figName);        % Retrieve figure handle if it already exists
        ax = exp.([figType 'ax1']);  % Retrieve axes handle if it exists
        cla(ax)
    else
        fig = figure;   % Initiate figure if it doesn't exist yet
        ax = gca(fig);  % Handle of axes to plot on
    end

    switch exp.domain
        case 'time'
            % Time domain plotting logic

            for i = 1:n                         % Loop through all data lines
                if i == 2; hold on; end         % Hold plotted lines if more than one

                if config.downSample            % Downsample the data if required
                    x = downsample(exp.t, config.downSampleRate);
                    y = downsample(data(i,:), config.downSampleRate);
                else                            % Use original data if not downsampling
                    x = exp.t;
                    y = data(i,:);
                end

                plot(ax,x,y,'DisplayName',char(exp.name(i)))   % Plot data
            end

        case 'frequency'
            % Frequency domain plotting logic
            if existPlot == 1
                % Retrieve magnitude axes handle if exists
                axMag = exp.([figType 'ax1']);
                cla(axMag)
                % Retrieve phase axes handle if exists
                axPha = exp.([figType 'ax2']);
                cla(axPha)
            else
                % Handle of axes to plot on
                axMag = subplot(2,1,1);
                axPha = subplot(2,1,2);
            end

            for i = 1:n

                mag = pow2db(abs(data(i,:)));           % System magnitude

                pha = unwrap(angle(data(i,:)))*180/pi;  % System Phase

                % Plot data
                semilogx(axMag,exp.f,mag,'DisplayName',char(exp.name(i)))
                semilogx(axPha,exp.f,pha,'DisplayName',char(exp.name(i)))

                % Hold plotted lines if more than one
                if i == 1
                    hold(axMag,'on')
                    hold(axPha,'on')
                end
            end
            % Plot Threshold
            semilogx(axMag,exp.f,exp.thres,'k','DisplayName',char(exp.name(n+1)))
    end
```

```
    exp.(figName) = fig;  % Save figure handle to data structure

    exp = applyStyling(exp,figType,config);   % Apply styling

end


function [exp] = applyStyling(exp,figType,config)
    fig = exp.([figType 'fig']);
    switch exp.domain
        case 'time'
            fig.Position = config.figPositionTime;
        case 'frequency'
            fig.Position = config.figPositionFreq;
    end
    axAll = findobj( get(fig,'Children'), '-depth', 1, 'type', 'axes');
    axAll = flip(axAll);
    n = numel(axAll);

    for i = 1:n
        % ax = gca(fig);  % Get axis handle
        ax = axAll(i); % Get axis handle

        if i == 1
            lgd = legend(ax);                   % Enable legend
            lgd.Location = config.locLegend;    % Set legend location
            ax.Title.String     = config.title; % Set plot title text
        end

        grid(ax,'on')

        % Set axis label text
        ax.XLabel.String    = config.xAxis;
        if n == 1
            ax.YLabel.String    = config.yAxis;
        else
            if i == 1
                ax.YLabel.String    = config.yAxisMag;
            elseif i == 2
                ax.YLabel.String    = config.yAxisPha;
            end
        end

        % Set text format
        if config.alterFont == true
            ax.FontSize         = config.fontAxis;
            ax.Title.FontSize   = config.fontTitle;
            lgd.FontSize        = config.fontLegend;
        end

        % Apply line styling to all data lines
        m = size(ax.Children,1);
        for j = 1:m
            k = m-j+1; % reverse order because most recently added line is index 1
            ax.Children(k).LineStyle = char(config.lineStyle(j));
            ax.Children(k).LineWidth = config.lineWidth;
        end

        switch exp.domain
            case 'time'
                ax.XLim = [min(exp.t) max(exp.t)];

                ax.YTickMode = 'auto';
                ax.YTick = downsample(ax.YTick,2);
        end

        exp.([figType 'ax' num2str(i)]) = ax;

    end
end
```

APPENDIX G. SolidWorks NACT-3D Model Properties

This appendix contains the properties of the NACT-3D as estimated using the CAD model. Various (sub)assemblies and parts were isolated and the 'Mass Properties' tool within SolidWorks was used to extract the properties based on the assigned SolidWorks materials (Fig. 1). All aluminium components of the NACT-3D were assumed to be "7075-T6 (SN)" and all steel components of the NACT-3D were assumed to be "Plain Carbon Steel".

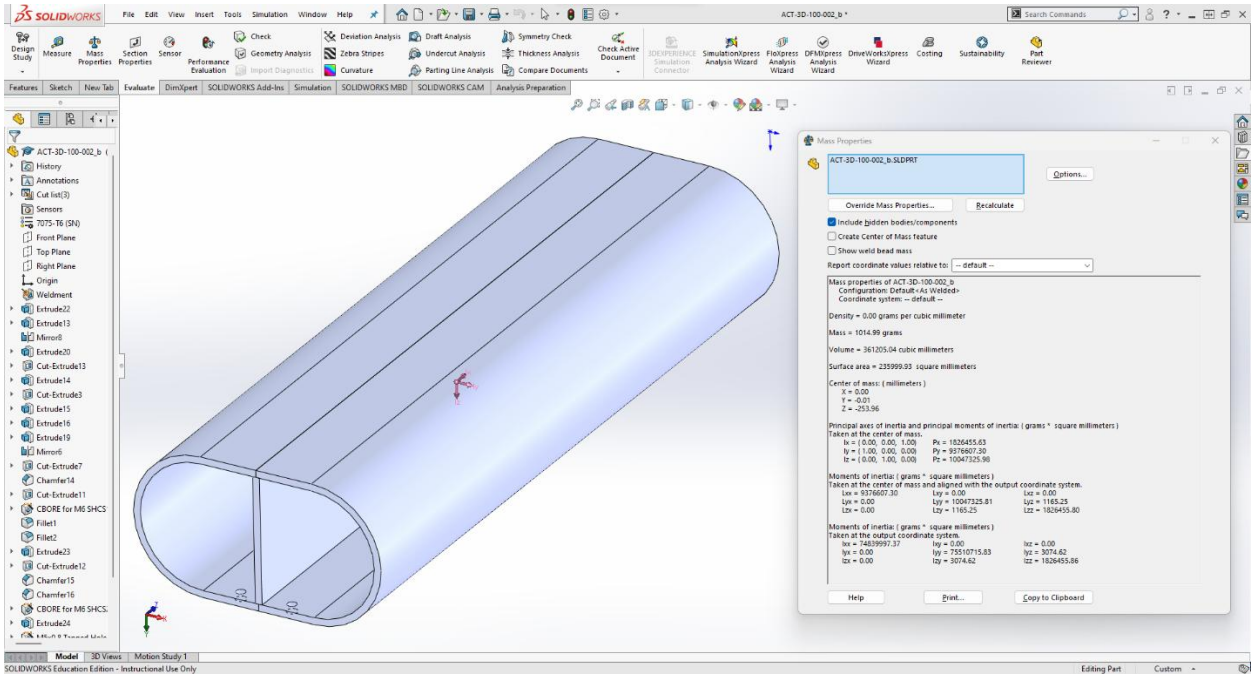


Fig. 1. Screenshot of an isolated part of the NACT-3D's SolidWorks CAD model. The central segment of the manipulator's upper arm is shown on the left and the output of the 'Mass Properties' tool is shown on the right.

All relevant mass properties for each isolated part were extracted from SolidWorks and collected:

G.1. Shoulder Up-Down around Shoulder Origin

```
Mass properties of selected components
     Coordinate system: Shoulder Origin

The center of mass and the moments of inertia are output in the coordinate system of Philip_ACT-
3D-000_Current
Mass = 2866.68 grams

Volume = 869563.55 cubic millimeters

Surface area = 234940.31  square millimeters

Center of mass: ( millimeters )
   X = -53.75
   Y = -5.07
   Z = -24.00

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
    Ix = ( 0.93,  0.29,  0.22)      Px = 9716901.07
```

```
    Iy = (-0.19,  0.90, -0.39)      Py = 13587531.72
    Iz = (-0.31,  0.33,  0.89)      Pz = 14591372.51

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
   Lxx = 10319743.36  Lxy = 1141311.40  Lxz = 1066914.52
   Lyx = 1141311.40   Lyy = 13373259.60 Lyz = -48836.76
   Lzx = 1066914.52   Lzy = -48836.76   Lzz = 14202802.35

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
   Ixx = 12044885.37  Ixy = 1921799.75  Ixz = 4765157.43
   Iyx = 1921799.75   Iyy = 23305993.24 Iyz = 299718.08
   Izx = 4765157.43   Izy = 299718.08   Izz = 22557514.11
```

G.2. Shoulder Rotation around Shoulder Actuation Arm Origin

```
Mass properties of selected components
     Coordinate system: Shoulder Actuation Arm Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Mass = 3716.47 grams

Volume = 929356.61 cubic millimeters

Surface area = 334872.51  square millimeters

Center of mass: ( millimeters )
    X = 5.54
    Y = 1.85
    Z = -5.02

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 0.23,  0.00,  0.97)     Px = 6271136.50
     Iy = ( 0.96, -0.18, -0.23)     Py = 15135976.69
     Iz = ( 0.18,  0.98, -0.04)     Pz = 16940324.54

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 14720923.83 Lxy = -321720.14  Lxz = 1999176.65
    Lyx = -321720.14  Lyy = 16882106.34 Lyz = 22027.38
    Lzx = 1999176.65  Lzy = 22027.38Lzz = 6744407.55

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 14827262.38 Ixy = -283600.82  Ixz = 1895846.28
    Iyx = -283600.82  Iyy = 17089778.56 Iyz = -12502.43
    Izx = 1895846.28  Izy = -12502.43   Izz = 6871217.81
```

G.3. Parallelogram Shoulder Moment Arm around Elbow Actuation Arm Origin

```
Mass properties of ACT-3D-100-015
     Configuration: Default
     Coordinate system: Elbow Actuation Arm Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Density = 0.01 grams per cubic millimeter

Mass = 285.10 grams

Volume = 36551.07 cubic millimeters

Surface area = 13718.21  square millimeters
```

```
Center of mass: ( millimeters )
    X = 0.00
    Y = 23.59
    Z = -3.60

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 0.00,  1.00,  0.02)     Px = 35460.64
     Iy = (-1.00,  0.00,  0.00)     Py = 292452.33
     Iz = ( 0.00, -0.02,  1.00)     Pz = 302442.47

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 292452.33   Lxy = -0.14   Lxz = -0.04
    Lyx = -0.14   Lyy = 35558.95Lyz = 5122.14
    Lzx = -0.04   Lzy = 5122.14 Lzz = 302344.16

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 454813.59   Ixy = -0.24   Ixz = -0.03
    Iyx = -0.24   Iyy = 39261.07Iyz = -19113.67
    Izx = -0.03   Izy = -19113.67   Izz = 461003.31
```

G.4. Elbow Move around Upper Arm Elbow Origin

```
Mass properties of selected components
     Coordinate system: Upper Arm Elbow Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Mass = 1123.48 grams

Volume = 265921.98 cubic millimeters

Surface area = 108749.60  square millimeters

Center of mass: ( millimeters )
    X = -21.19
    Y = -0.02
    Z = -29.34

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 0.99,  0.01, -0.14)     Px = 1770388.90
     Iy = (-0.01,  1.00,  0.00)     Py = 2565858.04
     Iz = ( 0.14,  0.00,  0.99)     Pz = 3536870.05

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 1803982.00  Lxy = 5975.70 Lxz = -241124.37
    Lyx = 5975.70 Lyy = 2565820.83  Lyz = 1237.74
    Lzx = -241124.37  Lzy = 1237.74 Lzz = 3503314.16

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 2771289.45  Ixy = 6535.67 Ixz = 457481.31
    Iyx = 6535.67 Iyy = 4037672.73  Iyz = 2013.08
    Izx = 457481.31   Izy = 2013.08 Izz = 4007859.86
```

G.5. Elbow Rotate around Elbow Origin

```
Mass properties of selected components
     Coordinate system: Elbow Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Mass = 2208.05 grams

Volume = 598983.08 cubic millimeters
```

```
Surface area = 236966.89  square millimeters

Center of mass: ( millimeters )
    X = 1.27
    Y = 1.68
    Z = 23.03

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = (-0.33,  0.79, -0.51)     Px = 4152199.39
     Iy = ( 0.60,  0.60,  0.53)     Py = 4288212.80
     Iz = ( 0.73, -0.13, -0.68)     Pz = 4624844.97

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 4450515.41  Lxy = -4728.76Lxz = 188303.79
    Lyx = -4728.76Lyy = 4208276.76  Lyz = -84095.24
    Lzx = 188303.79   Lzy = -84095.24   Lzz = 4406464.99

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 5627617.64  Ixy = 1.06 Ixz = 253019.86
    Iyx = 1.06 Iyy = 5382701.93  Iyz = 1476.94
    Izx = 253019.86   Izy = 1476.94 Izz = 4416296.13
```

G.6. Parallelogram +y Rod around Parallelogram +y Halfway Origin

```
Mass properties of selected components
     Coordinate system: Parallelogram +y Halfway Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Mass = 900.58 grams

Volume = 115458.50 cubic millimeters

Surface area = 32556.77  square millimeters

Center of mass: ( millimeters )
    X = 1.69
    Y = 0.07
    Z = 3.77

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 1.00,  0.00,  0.00)     Px = 37010.44
     Iy = ( 0.00,  1.00,  0.01)     Py = 16167182.00
     Iz = ( 0.00, -0.01,  1.00)     Pz = 16167918.98

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 37020.17Lxy = -10958.06   Lxz = -6070.95
    Lyx = -10958.06   Lyy = 16167174.62 Lyz = 10.84
    Lzx = -6070.95Lzy = 10.84   Lzz = 16167916.64

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 49857.76Ixy = -10854.37   Ixz = -311.53
    Iyx = -10854.37   Iyy = 16182592.78 Iyz = 241.88
    Izx = -311.53 Izy = 241.88  Izz = 16170505.53
```

G.7. Parallelogram -y Rod around Parallelogram -y Halfway Origin

```
Mass properties of selected components
     Coordinate system: Parallelogram -y Halfway Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
```

```
Mass = 900.58 grams

Volume = 115458.50 cubic millimeters

Surface area = 32556.77  square millimeters

Center of mass: ( millimeters )
    X = 1.69
    Y = -0.07
    Z = 3.77

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 1.00,  0.00,  0.00)     Px = 37010.44
     Iy = ( 0.00,  1.00, -0.01)     Py = 16167182.00
     Iz = ( 0.00,  0.01,  1.00)     Pz = 16167918.98

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 37020.17Lxy = 10958.06Lxz = -6070.95
    Lyx = 10958.06Lyy = 16167174.62 Lyz = -10.84
    Lzx = -6070.95Lzy = -10.84  Lzz = 16167916.64

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 49857.76Ixy = 10854.37Ixz = -311.53
    Iyx = 10854.37Iyy = 16182592.78 Iyz = -241.88
    Izx = -311.53 Izy = -241.88 Izz = 16170505.53
```

G.8. Wrist Move around Wrist Origin

```
Mass properties of ACT-3D-100-001_c
     Configuration: Default<As Welded>
     Coordinate system: Wrist Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Density = 0.00 grams per cubic millimeter

Mass = 369.26 grams

Volume = 131409.69 cubic millimeters

Surface area = 30610.97  square millimeters

Center of mass: ( millimeters )
    X = 0.00
    Y = -42.25
    Z = -1.46

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 0.00,  0.99, -0.14)     Px = 236699.21
     Iy = (-1.00,  0.00,  0.00)     Py = 539159.31
     Iz = ( 0.00,  0.14,  0.99)     Pz = 711287.73

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 539159.31   Lxy = 0.00 Lxz = 0.00
    Lyx = 0.00 Lyy = 246280.81   Lyz = -66749.61
    Lzx = 0.00 Lzy = -66749.61   Lzz = 701706.13

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 1199258.98  Ixy = 0.00 Ixz = 0.00
    Iyx = 0.00 Iyy = 247072.65   Iyz = -43900.79
    Izx = 0.00 Izy = -43900.79   Izz = 1361013.97
```

G.9. Upper Arm Section around Upper Arm Section Mid Origin

```
Mass properties of ACT-3D-100-002_b
     Configuration: Default<As Welded>
     Coordinate system: Upper Arm Section Mid Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Density = 0.00 grams per cubic millimeter

Mass = 1014.99 grams

Volume = 361205.04 cubic millimeters

Surface area = 235999.93  square millimeters

Center of mass: ( millimeters )
    X = -0.04
    Y = 0.00
    Z = 0.01

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 1.00,  0.00,  0.00)     Px = 1826455.63
     Iy = ( 0.00,  1.00,  0.00)     Py = 9376607.30
     Iz = ( 0.00,  0.00,  1.00)     Pz = 10047325.98

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 1826455.80  Lxy = 0.00 Lxz = 1165.25
    Lyx = 0.00 Lyy = 9376607.30  Lyz = 0.00
    Lzx = 1165.25  Lzy = 0.00 Lzz = 10047325.81

Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 1826455.86  Ixy = 0.00 Ixz = 1164.96
    Iyx = 0.00 Iyy = 9376608.80  Iyz = 0.00
    Izx = 1164.96  Izy = 0.00 Izz = 10047327.26
```

G.10.Lower Arm Section around Lower Arm Section Mid Origin

```
Mass properties of ACT-3D-100-001_b
     Configuration: Default<As Welded>
     Coordinate system: Lower Arm Section Mid Origin

The center of mass and the moments of inertia are output in the coordinate system of
Philip_ACT-3D-000_Current
Density = 0.00 grams per cubic millimeter

Mass = 653.04 grams

Volume = 232397.41 cubic millimeters

Surface area = 151722.73  square millimeters

Center of mass: ( millimeters )
    X = 0.00
    Y = 0.00
    Z = 0.00

Principal axes of inertia and principal moments of inertia: ( grams *  square millimeters )
Taken at the center of mass.
     Ix = ( 0.00,  0.98,  0.18)     Px = 464413.59
     Iy = (-1.00,  0.00,  0.00)     Py = 4809296.05
     Iz = ( 0.00, -0.18,  0.98)     Pz = 5035596.86

Moments of inertia: ( grams *  square millimeters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 4809296.05  Lxy = -0.59   Lxz = 2.73
    Lyx = -0.59   Lyy = 609527.87   Lyz = 801427.35
    Lzx = 2.73 Lzy = 801427.35   Lzz = 4890482.58
```

```
Moments of inertia: ( grams *  square millimeters )
Taken at the output coordinate system.
    Ixx = 4809296.05  Ixy = -0.59    Ixz = 2.73
    Iyx = -0.59    Iyy = 609527.87   Iyz = 801427.35
    Izx = 2.73 Izy = 801427.35   Izz = 4890482.58
```

APPENDIX H. SPACAR Model Code

SPACAR runs in the MATLAB environment, generating variables in the MATLAB workspace alongside SPACAR binary files. A .dat text file is required as in input for a simulation and consists of three blocks; kinematic definition of the model, dynamic properties, and the inverse dynamics with the setpoint generation (trajectory definition).

H.1. Kinematic Data

H.1.1. Elements and Nodes

To model the NACT-3D manipulator, spatial beam, truss, and hinge elements were used. Beams are deformable in all directions, allowing for the simulation of segment bending. Truss elements are only deformable in length; they represent the push-pull rods (small deformations) as well as the linear actuators (deformations as inputs). Hinges only allow rotational deformations and are used to represent the rotational joints in the manipulator. The interaction between elements is defined by the nodes that elements have in common. The elements and corresponding position and orientation nodes are defined in the following code:

```
TRUSS  1   1      2      # Upper Arm Actuation Slider
TRUSS  2   3      4      # Lower Arm Actuation Slider
TRUSS  3   5      6      # Shoulder Vertical Actuation Slider

BEAM   4   6   7   8   9   # Shoulder Axis from height of Bottom Plate to Shoulder Angle Moment
Arm

TRUSS  5   2      10     # Push-Pull Rod from Upper Arm Actuation Slider to Moment Arm for
Shoulder Angle
BEAM   6   10  11  8   12  # Shoulder Angle Moment Arm from shoulder push-pull rod to center of
shoulder
BEAM   7   8   12  13  14  # Shoulder Axis from height of Shoulder Angle Moment Arm to Elbow
Angle Moment Arm

TRUSS  8   4      15     # Push-Pull Rod from Lower Arm Actuation Slider to Moment Arm for
Elbow Angle

BEAM   9   15  16  17  18  # Elbow Angle Moment Arm from elbow push-pull rod to y+ shoulder
parallelogram
BEAM   10  17  18  13  19  # Elbow Angle Moment Arm from y+ shoulder parallelogram to shoulder
axis
BEAM   11  13  19  20  21  # Elbow Angle Moment Arm from shoulder axis to y- shoulder
parallelogram

BEAM   12  13  14  22  23  # Shoulder Axis from height of Elbow Angle Moment Arm to Upper Arm

BEAM   13  22  23  24  25  # Upper Arm Shoulder (proximal) Segment
BEAM   14  24  25  26  27  # Upper Arm Middle (main) Segment
BEAM   15  26  27  28  29  # Upper Arm Elbow (Distal) Segment

TRUSS  16  17     30     # Push-Pull Rod y+ Parallelogram
TRUSS  17  20     31     # Push-Pull Rod y- Parallelogram

BEAM   18  28  29  32  33  # Elbow Axis from height of Upper Arm to Parallelogram

BEAM   19  30  34  32  35  # Elbow Moment Arm from y+ Parallelogram to Elbow Axis
BEAM   20  32  35  31  36  # Elbow Moment Arm from  Elbow Axis to y- Parallelogram

BEAM   21  32  35  37  38  # Elbow Axis from height of Parallelogram to Lower Elbow

BEAM   22  37  38  39  40  # Lower Arm Elbow (proximal) Segment
BEAM   23  39  40  41  42  # Lower Arm Middle (main) Segment
BEAM   24  41  42  43  44  # Lower Arm Wrist (distal) Segment
```

```
HINGE  25     9      12     0   0   1
HINGE  26     14     19     0   0   1
HINGE  27     33     35     0   0   1
```

H.1.2. Initial Node Coordinates

The initial position of the NACT-3D manipulator was derived from the configuration in which: the push-pull rods were parallel to the horizontal plane, the upper arm segment coincident to the x-axis, and the lower arm parallel to the y-axis (at a 90-degree angle with the upper arm). The corresponding coordinates of the SPACAR model's position nodes are defined in the code below.

```
%  Node   x             y          z
X  1      -500          63         52.5082    # Upper Arm Slider Minimum
X  2      -472.3690     63         52.5082    # Upper Arm Spindle Actuation Point
X  3      -500          63         104.5082   # Lower Arm Slider Minimum
X  4      -429.4368     63         104.5082   # Lower Arm Spindle Actuation Point
X  5      0             0          -100       # Vertical Shoulder Slider Minimum
X  6      0             0          0          # Shoulder Plate Central Point
X  8      0             0          52.5082    # Shoulder Actuation Central Point
X  10     -42.5         73.6122    52.5082    # Shoulder Actuation Point
X  13     0             0          104.5082   # Shoulder Parallelogram Center Point
X  15     -0.00000002   85         104.5082   # Elbow Actuation Point
X  17     0             32         104.5082   # Shoulder Parallelogram +y Point
X  20     0             -32        104.5082   # Shoulder Parallelogram -y Point
X  22     0             0          106.0004   # Upper Arm Shoulder Point
X  24     92.7          0          106.0004   # Upper Arm Proximal Point
X  26     415.3         0          106.0004   # Upper Arm Distal Point
X  28     500           0          106.0004   # Upper Arm Elbow Point
X  32     500           0          104.5082   # Elbow Paralleogram Central Point
X  30     500           32         104.5082   # Elbow Parallelogram +y Point
X  31     500           -32        104.5082   # Elbow Parallelogram -y Point
X  37     500           0          38.3226    # Lower Arm Elbow Point
X  39     500           75         38.3226    # Lower Arm Proximal Point
X  41     500           364        89.2811    # Lower Arm Distal Point
X  43     500           450        75.45      # Wrist Point
```

H.1.3. System Constraints

To ensure the system was properly defined, fixed support coordinates, calculable deformations, prescribed degrees of freedom, and dynamic degrees of freedom were established in the code below:

```
FIX    1   1   2   3          # Shoulder Actuation Slider fixation point
FIX    3   1   2   3          # Elbow Actuation Slider fixation point
FIX    5   1   2   3          # Shoulder Vertical Actuation Slider fixation point

FIX    2   2   3              # Shoulder Actuation Slider can only move along x
FIX    4   2   3              # Elbow Actuation Slider can only move along x
FIX    6   1   2              # Shoulder Vertical Actuation Slider can only move along z

FIX    7                      # Shoulder Base Plate can not rotate

RLSE   25  1                  # Upper Arm can rotate around Shoulder
RLSE   26  1                  # Lower Arm Linkage can rotate around Shoulder
RLSE   27  1                  # Lower Arm can rotate around Elbow

RLSE   16     1               # Elbow Parallelogram y+ Truss can deform to avoid over-
constraining the system
```

For the inverse kinematic model, the following additional constraints were defined (NACT3DinvKin.dat):

```
FIX    43  3                  # End point can not move along z

RLSE   1   1
```

```
RLSE   2   1
RLSE   3   1

INPUTX 43  1                # End Point x input
INPUTX 43  2                # End Point y input
```

For the linearized forward dynamic model, the following additional constraints were defined (NACT3DinvKinJaap.dat):

```
DYNE   14
DYNE   23
DYNE   17

INPUTE 1   1                # Shoulder Actuation Slider Input
INPUTE 2   1                # Elbow Actuation Slider Input
```

The end of the kinematic data block is indicated with the following code:

```
END
HALT
```

H.2. Dynamic Data

H.2.1. Dynamic Properties

The dynamic properties of the NACT-3D manipulator were extracted from the SolidWorks CAD model and defined in the dynamic properties block. These properties were not needed for the inverse kinematic model simulation. For the linearized forward dynamic model simulation, the input signal and time signal were also defined in the following code (rearranged for ease of reading):

```
#   Node
XM  6       # Lumped Mass of translation at Shoulder Plate
XM  7       # Lumped Mass of translation at Shoulder Plate

XM  8       # Lumped Mass of rotation around Shoulder Axis at the height of Shoulder Angle
Moment Arm
XM  12      # Lumped Mass of rotation around Shoulder Axis at the height of Shoulder Angle
Moment Arm

XM  13      # Lumped Mass of Parallelogram Moment Arm at the Shoulder
XM  19      # Lumped Mass of Parallelogram Moment Arm at the Shoulder

XM  32      # Lumped Mass of translation at Upper Arm Elbow Origin
XM  33      # Lumped Mass of translation at Upper Arm Elbow Origin

XM  37      # Lumped Mass of rotation around Elbow Origin
XM  38      # Lumped Mass of rotation around Elbow Origin

XM  43      # Lumped Mass of Translation at Wrist Origin
XM  44      # Lumped Mass of Translation at Wrist Origin

#   Node    Mass        Ixx            Ixy            Ixz            Iyy            Iyz
    Izz
XM  6       2866.68
XM  7                   12044885.37    1921799.75     4765157.43     23305993.24    299718.08
    22557514.11

XM  8       3716.47
XM  12                  14827262.38    -283600.82     1895846.28     17089778.56    -12502.43
    6871217.81

XM  13      285.10
XM  19                  454813.59      -0.24          -0.03          39261.07       -19113.67
    461003.31
```

```
XM  32     1123.48
XM  33                 2771289.45     6535.67        457481.31      4037672.73     2013.08
    4007859.86

XM  37     2208.05
XM  38                 5627617.64     1.06           253019.86      5382701.93     1476.94
    4416296.13

XM  43     369.26
XM  44                 1199258.98     0.00           0.00           247072.65      -43900.79
    1361013.97

#   Element    m / L   Jxx            Jyy            Jzz            Jxz
EM  14         3.15    5662.494056    1791.994914    3870.49917     0          # Upper Arm
EM  23         2.23    1620.090186    425.058022     1195.032164    0.0034282  # Lower Arm
EM  16         1.9968  85.196826      42.598374      42.598374      0          # Parallelogram y+
Push Pull Rod
EM  17         1.9968  85.196826      42.598374      42.598374      0          # Parallelogram y-
Push Pull Rod

# Element  E*A               G*It              E*Iy              E*Iz
    E*Iy/(G*A*kz)  E*Iz/(G*A*ky)
ESTIFF 14  80635680000000    42654354068000000 45915883920000000 99172932480000000 2771.11
    8662.88
ESTIFF 23  57895920000000    35576535551000000 10891166400000000 30620041200000000 839.18
    4044.55
ESTIFF 16  53760000000000    728064000000000   1146879300000000  1146879300000000  66.81
    66.81
ESTIFF 17  53760000000000    728064000000000   1146879300000000  1146879300000000  66.81
    66.81

GRAVITY 0 0 -98.10
USERINP MYMOTSIM

TIMESTEP   7.0275 59031

END
HALT
```

H.2.2. MATLAB User Input

For the linearized forward dynamic model simulation, the output of the inverse kinematic model was used. The MATLAB variables are mapped to the corresponding SPACAR variables in the .m-file defined under '`USERINP`'. The "MYMOTSIM.m" MATLAB file defines the variable mapping with the following code:

```
function [t,e,x]  =  MYMOTSIM(t,is)

global tnom enom ednom eddnom

% search for right interval
for i=1:(numel(tnom)-1)
    if tnom(i+1)>=t
        break;
    end
end
iint=i;

einp   =  (1/(tnom(iint+1)-tnom(iint)))*( (tnom(iint+1)-t)*enom(iint,:)+(t-
tnom(iint))*enom(iint+1,:) );
edinp  =  (1/(tnom(iint+1)-tnom(iint)))*( (tnom(iint+1)-t)*ednom(iint,:)+(t-
tnom(iint))*ednom(iint+1,:) );
eddinp =  (1/(tnom(iint+1)-tnom(iint)))*( (tnom(iint+1)-t)*eddnom(iint,:)+(t-
tnom(iint))*eddnom(iint+1,:) );

e   =  [1  1   einp(133)  edinp(133) eddinp(133)
        2  1   einp(134)  edinp(134) eddinp(134) ];
x   =  [];
```

H.3. Setpoint Generation

H.3.1. Trajctory and Nominal Inputs & Outputs

For the inverse kinematic model simulation, the trajectory is initiated as a user-defined input trajectory in the following code:

```
TRAJ   1
USERTRAJ MYCHIRP
TRTIME     2.2639 18111

END
END
```

For the linearized forward dynamic model simulation, the input and output variables are defined. All variables are extracted from, and saved to, the MATLAB workspace. The nominal input and output variables are defined in the following code:

```
NOMS    1   1   1
NOMS    2   2   1

REFX    1   43  1
REFX    2   43  2
REFXP   3   43  1
REFXP   4   43  2
REFXDP  5   43  1
REFXDP  6   43  2

END
END
```

H.3.2. MATLAB Signal

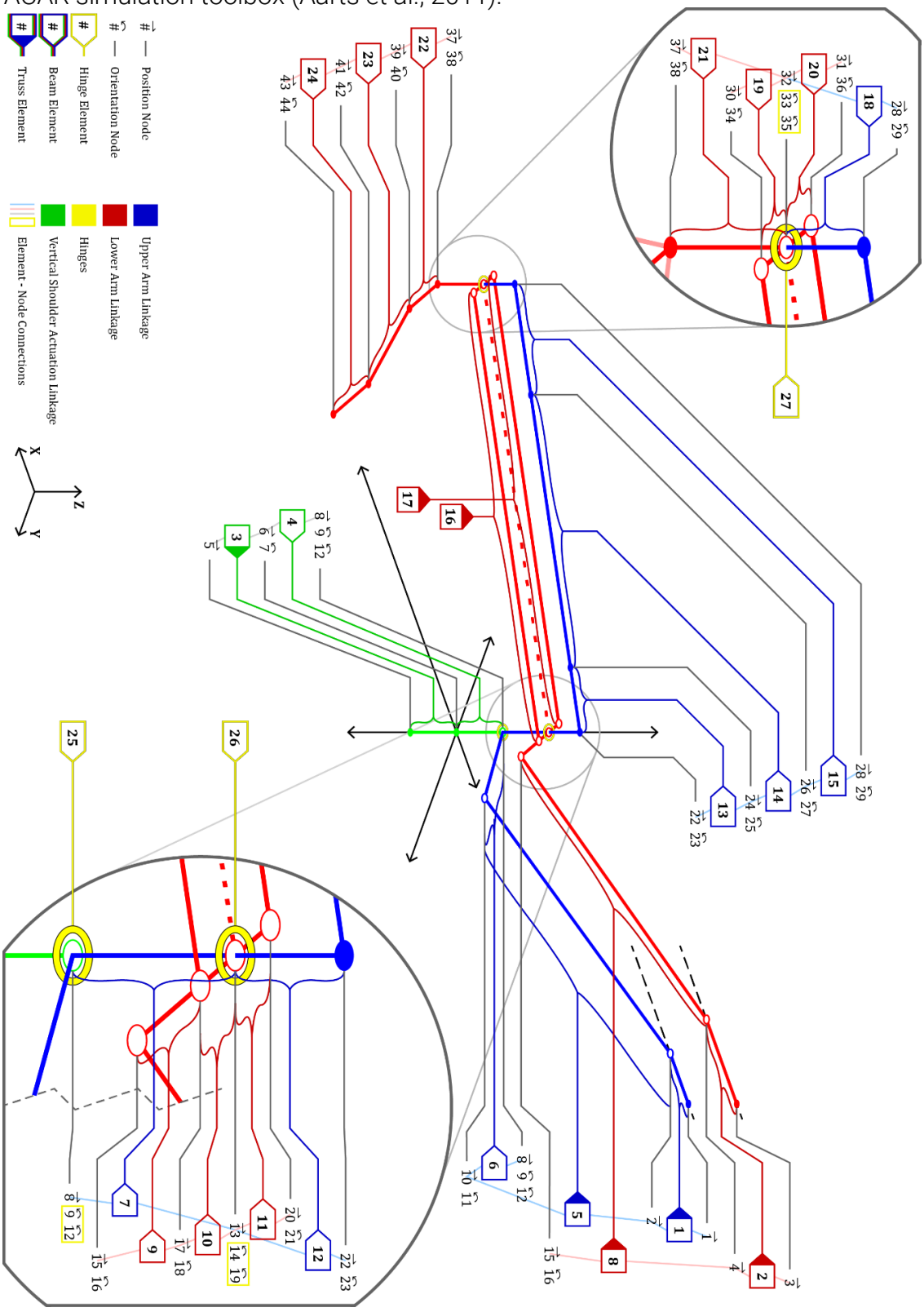
For the inverse kinematic model, the input user trajectory is stored in a MATLAB .m-file. The input chirp signal is defined in “MYCHIRP.m” with the following code:

```
function [t,e,x]  =   MYCHIRP(t,is)

global iniX iniY testDim sf amplitude freq

% Each frequency will have ten complete sine waves, total length of the
% signal is the sample frequency plus one to get to the initial position
% and plus the sum of the sample frequency divided by each frequency
% multiplied by the amount of periods, in this case 10 complete sine waves.

X = ones(1,ceil(sum(10*sf./freq)+sf+1));
Y = ones(1,ceil(sum(10*sf./freq)+sf+1));

if iniX == 500 && iniY == 450
    X(1:sf+1) = iniX;
    Y(1:sf+1) = iniY;
else
    X(1:sf+1) = (500:((iniX-500)/sf):iniX);
    Y(1:sf+1) = (450:((iniY-450)/sf):iniY);
end

tc_ = sf+1;    % time index for end of movement to initial position

for i = 1:numel(freq)
    nc = 10 * sf/freq(i);    % samples for one set of 10 periods
    tc = tc_ + nc;           % time index for end of current chirp

    % Generate the sine wave for the chirp signal in the x or y dimension
    % depending on which dimension is being tested.
    switch testDim
        case 'x'
            chirp = iniX + amplitude*sin((0:(2*pi/(2*sf)):(4*pi))*freq(i));
            X(tc_+1:tc) = chirp(1:nc);
```

```
            Y(tc_+1:tc) = Y(tc_+1:tc)*iniY;
        case 'y'
            chirp = iniY + amplitude*sin((0:(2*pi/(2*sf)):(4*pi))*freq(i));
            X(tc_+1:tc) = X(tc_+1:tc)*iniX;
            Y(tc_+1:tc) = chirp(1:nc);
    end
    tc_ = tc;               % time index for the end of previous chirp
end

X   = X';
Xd  = dx(X)*sf;
Xdd = dx(Xd)*sf;
Y   = Y';
Yd  = dx(Y)*sf;
Ydd = dx(Yd)*sf;

x   =   [43    1   X(is)  Xd(is) Xdd(is)
         43    2   Y(is)  Yd(is) Ydd(is)];
e   =   [];
```

APPENDIX I. SPACAR Model Visualization

This appendix contains a visualization of the NACT-3D's virtual model as created in the SPACAR simulation toolbox (Aarts et al., 2011).



APPENDIX J. SPACAR MATLAB Code

This appendix contains the MATLAB code used to simulate the chirp signal perturbations with the SPACAR manipulator model and extract all relevant data for all locations and perturbation directions.

J.1. spacarChirp.m

```
% This script runs multiple SPACAR simulations of the NACT-3D manipulator
% using a multi-frequency chirp test signal for different starting
% locations of the manipulator end point. This script also creates plots
% for the results of the simulations.


%% Startup Variables
% Make the outputs of the SPACAR simulation global so that it can be
% accessed in all relevant MATLAB scripts.

% Make the starting (initial) positions of the manipulator input and the
% dimension in which the chirp test signal will be executed global so that
% the execution of the SPACAR simulation can be automated.
global iniX iniY testDim sf amplitude freq

global iSim

global testLocations testDimensions threshold

testLocations = [450 700  550 200 450 500;...
                 200 200 -100 500 500 450];
% testLocations = [500;...
%                  450];
% testLocations = [450;...
%                  200];
testDimensions = ['x' 'y'];
% testDimensions = ['x'];
% testDimensions = ['y'];

% sf = 8000;                                    % [Hz] sample frequency
sf = 8400;                                    % [Hz] sample frequency

amplitude = 15;                               % [mm] amplitude of chirp
% freq = [20 40 80 100 125 160 200 250 320 400];  % [Hz] frequencies that are tested in the chirp
% freq = [20 40 80 100 125 160 200];  % [Hz] frequencies that are tested in the chirp
% freq = [20 40];    % [Hz] frequencies that are tested in the chirp
% freq = [20 30 40];     % [Hz] frequencies that are tested in the chirp
freq = [5 10 15 20 25 30 35 40 50 60 80 100];   % [Hz] frequencies that are tested in the chirp
% freq = [5];   % [Hz] frequencies that are tested in the chirp

threshold = 3;  % [dB] FRF bandwidth above this magnitude

% dataSPACAR = struct;
% iSim = 0;

%% Run SPACAR Simulations
% iLoc = 6;
% iDim = 2;
% iSim = 2;
% iniX = testLocations(1,iLoc);
% iniY = testLocations(2,iLoc);
% testDim = testDimensions(iDim);
% spacarRunChirp;


% %%
% threshold = 3;
% for iSim = 1:5
%     spacarRunChirp;
% end
```

```
%%

iSim = 0;

tic
simulationTime = zeros(1,numel(testLocations));

for iLoc = 5%:numel(testLocations(1,:))
    for iDim = 1%:numel(testDimensions)
        iSim = iSim +1;

        iniX = testLocations(1,iLoc);
        iniY = testLocations(2,iLoc);
        testDim = testDimensions(iDim);

        fprintf(['Running SPACAR simulation for ' testDim ...
                 ' perturbation at location ' num2str(iLoc) '.\n']);

        spacarRunChirp;
%         spacar(7,'NACT3DinvKinLinJaap')

        pause(30)

        toc
        simulationTime(iSim) = toc;
    end
end

toc

%%
for iLoc = 1:numel(testLocations(1,:))
    for iDim = 1:numel(testDimensions)
        iSim = iSim +1;

        iniX = testLocations(1,iLoc);
        iniY = testLocations(2,iLoc);
        testDim = testDimensions(iDim);

        % Run SPACAR inverse kinematic simulation
        spacar(2,'NACT3DinvKin')

        % Extract calculated deformations of actuator elements
        % tnom = time(1:end-1);
        enom = e;
        ednom = ed;
        eddnom = edd;

        tnom = time(1:end);
        % Run SPACAR linearized forward dynamic model
        spacar(1,'NACT3DinvKinLinJaap')
        tnom = time(1:end-1);

        % Define the reference chirp signal in the same manner that the .m file
        % defines it for the SPACAR simulation.

        % Each frequency will have ten complete sine waves, total length of the
        % signal is the sample frequency plus one to get to the initial position
        % and plus the sum of the sample frequency divided by each frequency
        % multiplied by the amount of periods, in this case 10 complete sine waves.
        % The simulation is given one second to get to the starting position before
        % simulating the chirp signal.

        % Create the reference signal out of ones with the correct size
        X = ones(1,sum(10*sf./freq)+sf+1);
        Y = ones(1,sum(10*sf./freq)+sf+1);

        % Generate the first part of the trajectory to get to the initial part of
        % the chirp trajectory. The manipulator end point starts at (x,y)=(500,450)
        % according to it's configuration in the SPACAR file. It will be given 1
        % second to move to the starting point of the (chirp) test signal.
        X(1:sf+1) = (500:((iniX-500)/sf):iniX);
```

```
        Y(1:sf+1) = (450:((iniY-450)/sf):iniY);

        tc_ = sf+1;             % time index for end of movement to initial position
        freqLine(1) = tc_/sf;   % put the time value at the end of movement to the initial position in
an array for later use
        tFreqLine(1) = tc_;     % put the time index at the end of movement to the initial position in
an array for later use

        %% Generate the sine wave for each frequency in the chirp test signal.
        for i = 1:numel(freq)
            chirp = iniX + amplitude*sin((0:(2*pi/(2*sf)):(4*pi))*freq(i));
            nc = 10 * sf/freq(i);   % samples for one set of 10 periods
            tc = tc_ + nc;          % time index for end of current chirp
            % Generate the sine wave for the chirp signal in the x or y dimension
            % depending on which dimension is being tested.
            switch testDim
                case 'x'
                    X(tc_+1:tc) = chirp(1:nc);
                    Y(tc_+1:tc) = Y(tc_+1:tc)*iniY;
                case 'y'
                    X(tc_+1:tc) = X(tc_+1:tc)*iniX;
                    Y(tc_+1:tc) = chirp(1:nc);
            end
            tc_ = tc;               % time index for the end of previous chirp
            freqLine(i+1) = tc/sf;  % put the time value in an array
            tFreqLine(i+1) = tc;    % put the time index in an array
        end
        %% sysID of SPACAR Chirp

%         Xinput   = X(8001:end)';
%         Xoutput  = x(8002:end,lnp(43,1));
%         Yinput   = Y(8001:end)';
%         Youtput  = x(8002:end,lnp(43,2));
%
%         dt = 1/sf;
%         N   = numel(Xinput); % [ ] # of samples
%         T   = N*dt;          % [s] observation time
%         t   = (0:N-1).'*dt;  % [s] time vector
%         f   = 0:(sf-1); % [Hz] frequency vector
%
%         % Bandwidth ID
%         XCrossSD = cpsd(Xoutput,Xinput, rectwin(sf), 0, f, sf);
%         YCrossSD = cpsd(Youtput,Yinput, rectwin(sf), 0, f, sf);
%         XAutoSD     = pwelch(Xinput, rectwin(sf), 0, f, sf);
%         YAutoSD     = pwelch(Yinput, rectwin(sf), 0, f, sf);
%         XH          = XCrossSD ./ XAutoSD;
%         YH          = YCrossSD ./ YAutoSD;
%
%         Xmag  = pow2db(abs(XH(1,freq+1)));          % [dB] system magnitude for perturbed frequenties
%         Ymag  = pow2db(abs(YH(1,freq+1)));          % [dB] system magnitude for perturbed frequenties
%         Xang  = unwrap(angle(XH(1,freq+1)))*180/pi; % [degrees] system phase for perturbed
frequenties (unwrapped to stay within 360 degrees)
%         Yang  = unwrap(angle(YH(1,freq+1)))*180/pi; % [degrees] system phase for perturbed
frequenties (unwrapped to stay within 360 degrees)
%
%         % Find system bandwidth by interpolating where the magnitude
%         % crosses the threshold.
%         Xmag_ = Xmag - threshold;                  % Theshold value is now zero
%         Ymag_ = Ymag - threshold;                  % Theshold value is now zero
%         XfZC     = @(x) find(diff(sign(Xmag_))<0);    % function to find indices of zero crossings
%         YfZC     = @(x) find(diff(sign(Ymag_))<0);    % function to find indices of zero crossings
%         XiZC     = XfZC(Xmag_);                        % indices of zero crossings
%         YiZC     = YfZC(Ymag_);                        % indices of zero crossings
%         XfInterp = @(x1,y1,x2,y2) x1 - (y1.*(x1 - x2))./(y1 - y2);   %function to interpolate the
bandwidth frequency
%         YfInterp = @(x1,y1,x2,y2) x1 - (y1.*(x1 - x2))./(y1 - y2);   %function to interpolate the
bandwidth frequency
%         XBW      = XfInterp(freq(XiZC),Xmag_(XiZC),freq(XiZC+1),Xmag_(XiZC+1)); % [power] bandwidth
%         YBW      = YfInterp(freq(YiZC),Ymag_(YiZC),freq(YiZC+1),Ymag_(YiZC+1)); % [power] bandwidth
%
%
%         % Only take the first bandwidth entry (in some cases the
%         % magnitude crosses the threshold more than once)
%         if numel(XBW)>1; XBW = XBW(1); end
```

```
%         if numel(YBW)>1; YBW = YBW(1); end

        % store data in structure
        dataSPACAR.sim(iSim).xRef = X;
        dataSPACAR.sim(iSim).yRef = Y;
        dataSPACAR.sim(iSim).xSim = x(2:end,lnp(43,1));
        dataSPACAR.sim(iSim).ySim = x(2:end,lnp(43,2));
        dataSPACAR.sim(iSim).Fshoulder = sig(2:end,le(1,1))/1000000;
        dataSPACAR.sim(iSim).Felbow = sig(2:end,le(2,1))/1000000;
        dataSPACAR.sim(iSim).freqLine = freqLine;
        dataSPACAR.sim(iSim).tFreqLine = tFreqLine;

%         dataSPACAR.sim(iSim).t = t;
%         dataSPACAR.sim(iSim).f = f;
%         dataSPACAR.sim(iSim).XH = XH;
%         dataSPACAR.sim(iSim).YH = YH;
%         dataSPACAR.sim(iSim).XBW = XBW;
%         dataSPACAR.sim(iSim).YBW = YBW;

    end
end


%%
maxStress = 8465; % 36Nm torque? 1 rot. = 20mm
maxS = ones(length(tnom),1)*maxStress;
figure
    axX = subplot(2,1,1);
        plot(tnom,x(2:end,lnp(43,1)))
        hold on
        plot(tnom,X')
        title('SPACAR Chirp End Point Position')
%         xlabel('Time [s]')
        ylabel('End Point X [mm]')
        for i = 1:numel(freqLine)
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,370,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        legend({'Simulated Position' 'Reference'},'Location','northwest')
        ylim([350 540])
    axY = subplot(2,1,2);
        plot(tnom,x(2:end,lnp(43,2)))
        hold on
        plot(tnom,Y')
%         title('End Point Y Position')
%         xlabel('Time [s]')
        ylabel('End Point Y [mm]')
        for i = 1:numel(freqLine)
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,480,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        ylim([475 520])
    linkaxes([axX,axY],'x');
    xlim([tnom(sf) max(tnom)])
figure
    axS = subplot(2,1,1);
        plot(tnom,sig(2:end,le(1,1))/1000000)
        title('SPACAR Chirp Shoulder Servo Spindle Force')
%         xlabel('Time [s]')
        ylabel('Shoulder Servo Spindle Force [N]')
        hold on
        plot(tnom,maxS,'r--')
```

```
        plot(tnom,-maxS,'r--')
        for i = 1:numel(freqLine)
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,-140000,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        legend({'Spindle Force' 'Force Limit'},'Location','northwest')
        ylim([-180000 150000])
    axE = subplot(2,1,2);
        plot(tnom,sig(2:end,le(2,1))/1000000)
%         title('Elbow Servo Spindle Force')
        xlabel('Time [s]')
        ylabel('Elbow Servo Spindle Force [N]')
        hold on
        plot(tnom,maxS,'r--')
        plot(tnom,-maxS,'r--')
        for i = 1:numel(freqLine)
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,-340000,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        ylim([-400000 300000])
    linkaxes([axS,axE],'x');
    xlim([tnom(sf) max(tnom)])

%% Zoom Figure definition
zoomBegin = 4;
zoomEnd = 9;
zF = [tFreqLine(zoomBegin) tFreqLine(zoomEnd)];

%% Figure Position Zoom
figure
    axXz = subplot(2,1,1);
        plot(tnom(zF(1):zF(2)),x(zF(1)+1:zF(2)+1,lnp(43,1)))
        hold on
        plot(tnom(zF(1):zF(2)),X(zF(1):zF(2))')
        title('SPACAR Chirp End Point Position Zoomed')
%         xlabel('Time [s]')
        ylabel('End Point X [mm]')
        for i = zoomBegin:zoomEnd
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,430,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        legend({'Simulated Position' 'Reference'},'Location','northwest')
        ylim([420 470])
    axYz = subplot(2,1,2);
        plot(tnom(zF(1):zF(2)),x(zF(1)+1:zF(2)+1,lnp(43,2)))
        hold on
        plot(tnom(zF(1):zF(2)),Y(zF(1):zF(2))')
%         title('End Point Y Position')
%         xlabel('Time [s]')
        ylabel('End Point Y [mm]')
        for i = zoomBegin:zoomEnd
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,488,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
```

```
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        ylim([480 510])
    linkaxes([axXz,axYz],'x');
    xlim([freqLine(zoomBegin) freqLine(zoomEnd)]);

%% Figure Force Zoomed
figure
    axSz = subplot(2,1,1);
        plot(tnom(zF(1):zF(2)),sig(zF(1)+1:zF(2)+1,le(1,1))/1000000)
        title('SPACAR Chirp Shoulder Servo Spindle Force Zoomed')
%         xlabel('Time [s]')
        ylabel('Shoulder Servo Spindle Force [N]')
        hold on
        plot(tnom(zF(1):zF(2)),maxS(zF(1):zF(2)),'r--')
        plot(tnom(zF(1):zF(2)),-maxS(zF(1):zF(2)),'r--')
        for i = zoomBegin:zoomEnd
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,-120000,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        legend({'Spindle Force' 'Force Limit'},'Location','northwest')
        ylim([-150000 120000])
    axEz = subplot(2,1,2);
        plot(tnom(zF(1):zF(2)),sig(zF(1)+1:zF(2)+1,le(2,1))/1000000)
%         title('Elbow Servo Spindle Force')
        xlabel('Time [s]')
        ylabel('Elbow Servo Spindle Force [N]')
        hold on
        plot(tnom(zF(1):zF(2)),maxS(zF(1):zF(2)),'r--')
        plot(tnom(zF(1):zF(2)),-maxS(zF(1):zF(2)),'r--')
        for i = zoomBegin:zoomEnd
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,-35000,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontWeight','bold')
            end
        end
        ylim([-45000 40000])
    linkaxes([axSz,axEz],'x');
    xlim([freqLine(zoomBegin) freqLine(zoomEnd)]);

%% sysID of SPACAR Chirp

Xinput   = X(8001:end)';
Xoutput  = x(8002:end,lnp(43,1));
Yinput   = Y(8001:end)';
Youtput  = x(8002:end,lnp(43,2));

threshold = -3;  % [dB] FRF bandwidth above this magnitude
dt = 1/sf;
N   = numel(Xinput); % [ ] # of samples
T   = N*dt;          % [s] observation time
t   = (0:N-1).'*dt;% [s] time vector
f   = 0:(sf-1); % [Hz] frequency vector

% Bandwidth ID
XCrossSD    = cpsd(Xoutput,Xinput, rectwin(sf), 0, f, sf);
YCrossSD    = cpsd(Youtput,Yinput, rectwin(sf), 0, f, sf);
XAutoSD     = pwelch(Xinput, rectwin(sf), 0, f, sf);
YAutoSD     = pwelch(Yinput, rectwin(sf), 0, f, sf);
XH          = XCrossSD ./ XAutoSD;
YH          = YCrossSD ./ YAutoSD;
```

```
Xmag    = pow2db(abs(XH(1,freq+1)));          % [dB] system magnitude for perturbed frequenties
Ymag    = pow2db(abs(YH(1,freq+1)));          % [dB] system magnitude for perturbed frequenties
Xang    = unwrap(angle(XH(1,freq+1)))*180/pi; % [degrees] system phase for perturbed frequenties
(unwrapped to stay within 360 degrees)
Yang    = unwrap(angle(YH(1,freq+1)))*180/pi; % [degrees] system phase for perturbed frequenties
(unwrapped to stay within 360 degrees)

% Find system bandwidth by interpolating where the magnitude
% crosses the threshold.
Xmag_   = Xmag - threshold;                  % Theshold value is now zero
Ymag_   = Ymag - threshold;                  % Theshold value is now zero
XfZC     = @(x) find(diff(sign(Xmag_))<0);    % function to find indices of zero crossings
YfZC     = @(x) find(diff(sign(Ymag_))<0);    % function to find indices of zero crossings
XiZC     = XfZC(Xmag_);                        % indices of zero crossings
YiZC     = YfZC(Ymag_);                        % indices of zero crossings
XfInterp = @(x1,y1,x2,y2) x1 - (y1.*(x1 - x2))./(y1 - y2);   %function to interpolate the bandwidth
frequency
YfInterp = @(x1,y1,x2,y2) x1 - (y1.*(x1 - x2))./(y1 - y2);   %function to interpolate the bandwidth
frequency
XBW      = XfInterp(freq(XiZC),Xmag_(XiZC),freq(XiZC+1),Xmag_(XiZC+1)); % [power] bandwidth
YBW      = YfInterp(freq(YiZC),Ymag_(YiZC),freq(YiZC+1),Ymag_(YiZC+1)); % [power] bandwidth


% Only take the first bandwidth entry (in some cases the
% magnitude crosses the threshold more than once)
if numel(XBW)>1; XBW = XBW(1); end
if numel(YBW)>1; YBW = YBW(1); end

%% Bode Plots
figure
subplot(2,2,1)
    semilogx(1:sf,pow2db(abs(XH)))
    hold on
    grid on
    semilogx(freq,pow2db(abs(XH(1,freq+1))),'LineWidth',2)
    title('FRF x-direction NACT-3D SPACAR CHIRP - Location 1 - X Pertubation')
    ylabel('Magnitude [dB]')
    legend({'All Frequencies' 'Only Perturbed Frequencies'})
subplot(2,2,3)
    semilogx(1:sf,unwrap(angle(XH))*180/pi)
    hold on
    grid on
    semilogx(freq,unwrap(angle(XH(1,freq+1)))*180/pi,'LineWidth',2)
    ylabel('Phase [degrees]')
    xlabel('Frequency [Hz]')

subplot(2,2,2)
    semilogx(1:sf,pow2db(abs(YH)))
    hold on
    grid on
    semilogx(freq,pow2db(abs(YH(1,freq+1))),'LineWidth',2)
    title('FRF y-direction NACT-3D SPACAR CHIRP - Location 1 - X Pertubation')
    ylabel('Magnitude [dB]')
    legend({'All Frequencies' 'Only Perturbed Frequencies'})
subplot(2,2,4)
    semilogx(1:sf,unwrap(angle(YH))*180/pi)
    hold on
    grid on
    semilogx(freq,unwrap(angle(YH(1,freq+1)))*180/pi,'LineWidth',2)
    ylabel('Phase [degrees]')
    xlabel('Frequency [Hz]')

% Run SPACAR eigen frequency simulation

spacar(7,'NACT3DinvKinLinJaap')
logID = fopen('nact3dinvkinlinjaap.log','rt');
logLine = fgetl(logID);
lineCounter = 1;
idxLine = 0;
idxEigV = 1;

while ischar(logLine)
    if idxLine == 1
        fprintf('%s\n', logLine);
```

```
        txtEigV = extractBetween(logLine,': ',')');
        nEigV = numel(txtEigV)*2;
        for j = 1:nEigV
            EigV(idxEigV) = str2double(cell2mat(txtEigV(ceil(j/2))));
            if ~mod(idxEigV,2)
                EigV(idxEigV) = -EigV(idxEigV);
            end
            idxEigV = idxEigV + 1;
        end
    end
    % Print out what line we're operating on.
    % See if this line starts with CAM
    if startsWith(logLine, ' Eigenvalue')
        fprintf('%s\n', logLine);
        idxLine = 1;
    else
        idxLine = 0;
    end
    % Read the next line.
    logLine = fgetl(logID);
    lineCounter = lineCounter + 1;
end
% close log file
fclose(logID);

sys.(tM).eigenValues = EigV;
```

J.2. spacarChirpPlots.m

```
%% Initial Variables
global testLocations testDimensions freq sf threshold

maxStress = 8465; % 36Nm torque? 1 rot. = 20mm
% maxStress = 5122.5; % [N]
maxS = ones(length(tnom),1)*maxStress;
threshold = -3;

tc_ = sf+1;             % time index for end of movement to initial position
freqLine(1) = tc_/sf;   % put the time value at the end of movement to the initial position in an
array for later use
tFreqLine(1) = tc_;     % put the time index at the end of movement to the initial position in an
array for later use

% Generate the sine wave for each frequency in the chirp test signal.
for i = 1:numel(freq)
    nc = 10 * sf/freq(i);   % samples for one set of 10 periods
    tc = tc_ + nc;          % time index for end of current chirp
    tc_ = tc;               % time index for the end of previous chirp
    freqLine(i+1) = tc/sf;  % put the time value in an array
    tFreqLine(i+1) = tc;    % put the time index in an array
end

j1 = 1;
j2 = 10;

% Styling parameters
par.fontTitle   = 22;   % Font size for title
par.fontAxis    = 18;   % Font size for text along axis
par.fontLegend  = 14;   % Font size for legend
par.lineWidth   = 1;    % Line thickness for readability

%% End Point Location Plots

for j = j1:j2
    figPosition = figure('Name',['SPACAR Chirp Position for ' ...
                                 testDimensions(2-rem(j,2)) ...
                                 ' Perturbation at Location ' ...
                                 num2str(ceil(j/2))]);
        axX = subplot(2,1,1);
            plot(tnom,dataSPACAR.sim(j).xSim,'LineWidth',par.lineWidth)
            hold on
            grid on
            plot(tnom,dataSPACAR.sim(j).xRef,'r--','LineWidth',par.lineWidth)
            title(['x_{end point} for ' ...
                    testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' ...
                    num2str(ceil(j/2))])
            xlabel('Time [s]')
            ylabel('x_{end point} [mm]')
            for i = 1:numel(freqLine)
                xline(freqLine(i));
                if i > 1
                    text((freqLine(i)+freqLine(i-1))/2,...
                        testLocations(1,ceil(j/2))-25,...
                        [num2str(freq(i-1)) ' Hz'],...
                        'HorizontalAlignment','center',...
                        'Rotation',60,...
                        'FontSize',par.fontLegend)
                end
            end
            legend({'Simulated Position' 'Reference Trajectory'},...
                'Location','northwest')
            ylim([testLocations(1,ceil(j/2))-40 testLocations(1,ceil(j/2))+40])
        axY = subplot(2,1,2);
            plot(tnom,dataSPACAR.sim(j).ySim,'LineWidth',par.lineWidth)
            hold on
            grid on
            plot(tnom,dataSPACAR.sim(j).yRef,'r--','LineWidth',par.lineWidth)
            title(['y_{end point} for ' ...
```

```
                    testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' ...
                    num2str(ceil(j/2))])
            xlabel('Time [s]')
            ylabel('y_{end point} [mm]')
            for i = 1:numel(freqLine)
                xline(freqLine(i));
                if i > 1
                    text((freqLine(i)+freqLine(i-1))/2,...
                        425,...%testLocations(2,ceil(j/2))-25,...
                        [num2str(freq(i-1)) ' Hz'],...
                        'HorizontalAlignment','center',...
                        'Rotation',60,...
                        'FontSize',par.fontLegend)
                end
            end
            ylim([testLocations(2,ceil(j/2))-5 testLocations(2,ceil(j/2))+5])
        linkaxes([axX,axY],'x');
        xlim([tnom(sf) max(tnom)])

        % Styling
        figPosition.Position   = [300   150   1500   800];
        axX.FontSize        = par.fontAxis;
        axY.FontSize        = par.fontAxis;
        axX.Title.FontSize  = par.fontTitle;
        axY.Title.FontSize  = par.fontTitle;

    saveas(gcf,[figPosition.Name '.png'])
    saveas(gcf,[figPosition.Name '.fig'])
end

%% Spindle Force Plots
for j = j1:j2
    figForce = figure('Name',['SPACAR Chirp Force for ' ...
                              testDimensions(2-rem(j,2)) ...
                              ' Perturbation at Location ' ...
                              num2str(ceil(j/2))]);
        axS = subplot(2,1,1);
            plot(tnom,dataSPACAR.sim(j).Fshoulder/1000)
            title(['F_{shoulder} at Spindle for ' testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' num2str(ceil(j/2))])
            xlabel('Time [s]')
            ylabel('F_{s} [kN]')
            hold on
            plot(tnom,maxS/1000,'r--','LineWidth',par.lineWidth)
            plot(tnom,-maxS/1000,'r--','LineWidth',par.lineWidth)
            for i = 1:numel(freqLine)
                xline(freqLine(i));
                if i > 1
                    text((freqLine(i)+freqLine(i-1))/2,-20,...
                        [num2str(freq(i-1)) ' Hz'],...
                        'HorizontalAlignment','center',...
                        'Rotation',60,...
                        'FontSize',par.fontLegend)
                end
            end
            legend({'Required Force' 'Threshold'},'Location','northwest',...
                    'FontSize',par.fontLegend)
            ylim([-30000 30000]/1000)
        axE = subplot(2,1,2);
            plot(tnom,dataSPACAR.sim(j).Felbow/1000)
            title(['F_{elbow} at Spindle for ' testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' num2str(ceil(j/2))])
            xlabel('Time [s]')
            ylabel('F_{e} [kN]')
            hold on
            plot(tnom,maxS/1000,'r--','LineWidth',par.lineWidth)
            plot(tnom,-maxS/1000,'r--','LineWidth',par.lineWidth)
            for i = 1:numel(freqLine)
                xline(freqLine(i));
                if i > 1
                    text((freqLine(i)+freqLine(i-1))/2,-15,...
                        [num2str(freq(i-1)) ' Hz'],...
                        'HorizontalAlignment','center',...
```

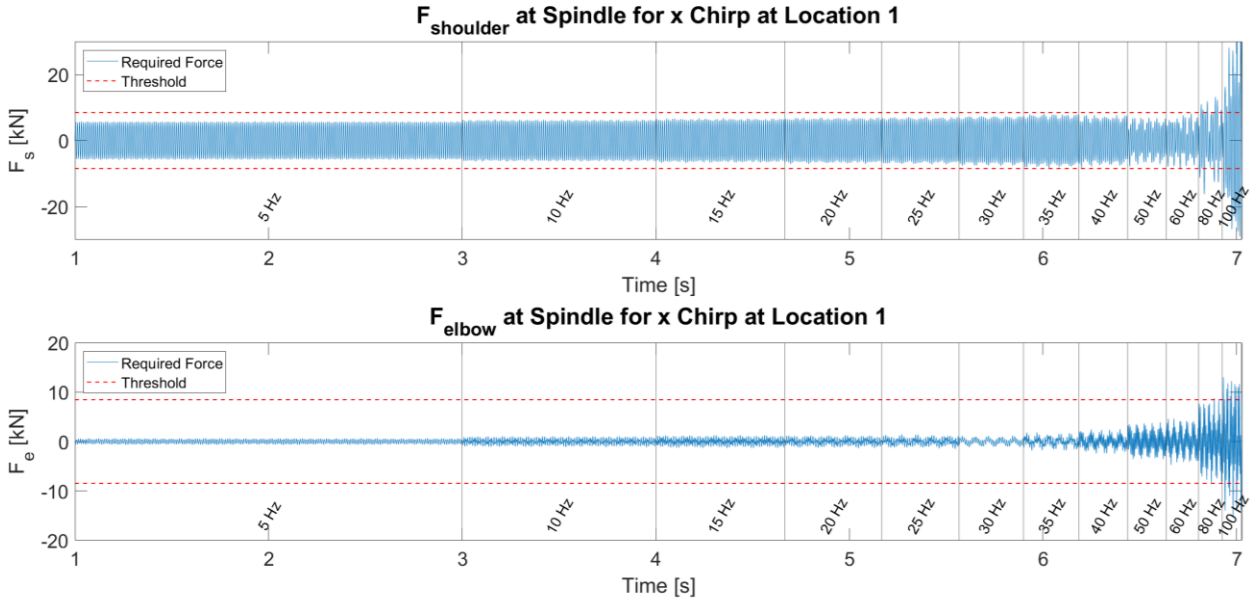
```
                        'Rotation',60,...
                        'FontSize',par.fontLegend)
                end
            end
            legend({'Required Force' 'Threshold'},'Location','northwest',...
                    'FontSize',par.fontLegend)
            ylim([-20000 20000]/1000)
        linkaxes([axS,axE],'x');
        xlim([tnom(sf) max(tnom)])

        % Styling
        figForce.Position  = [100   150   1900   800];
        axS.FontSize        = par.fontAxis;
        axE.FontSize        = par.fontAxis;
        axS.Title.FontSize  = par.fontTitle;
        axE.Title.FontSize  = par.fontTitle;

    saveas(gcf,[figForce.Name '.png'])
    saveas(gcf,[figForce.Name '.fig'])
end

%% Zoom Figure definition
zoomBegin = 10;
zoomEnd = 13;
zF = [tFreqLine(zoomBegin) tFreqLine(zoomEnd)];

%% Figure Position Zoom
figPosZoom = figure('Name',['Zoomed SPACAR Chirp Position for ' ...
                                testDimensions(2-rem(j,2)) ...
                                ' Perturbation at Location ' ...
                                num2str(ceil(j/2))]);
    axXz = subplot(2,1,1);
        plot(tnom(zF(1):zF(2)),dataSPACAR.sim(j).xSim(zF(1):zF(2)),...
            'LineWidth',par.lineWidth)
        hold on
        grid on
        plot(tnom(zF(1):zF(2)),dataSPACAR.sim(j).xRef(zF(1):zF(2)),...
            'r--','LineWidth',par.lineWidth)
        title(['Zoomed x_{end point} for ' ...
                    testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' ...
                    num2str(ceil(j/2))])
        xlabel('Time [s]')
        ylabel('x_{end point} [mm]')
        for i = zoomBegin:zoomEnd
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,420,...
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontSize',par.fontLegend)
            end
        end
        legend({'Simulated Position' 'Reference Trajectory'},...
            'Location','northwest')
        ylim([testLocations(1,ceil(j/2))-40 testLocations(1,ceil(j/2))+40])
    axYz = subplot(2,1,2);
        plot(tnom(zF(1):zF(2)),dataSPACAR.sim(j).ySim(zF(1):zF(2)),...
            'LineWidth',par.lineWidth)
        hold on
        grid on
        plot(tnom(zF(1):zF(2)),dataSPACAR.sim(j).yRef(zF(1):zF(2)),...
            'r--','LineWidth',par.lineWidth)
        title(['Zoomed y_{end point} for ' ...
                    testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' ...
                    num2str(ceil(j/2))])
        xlabel('Time [s]')
        ylabel('y_{end point} [mm]')
        for i = zoomBegin:zoomEnd
            xline(freqLine(i));
            if i > 1
                text((freqLine(i)+freqLine(i-1))/2,488,...
```

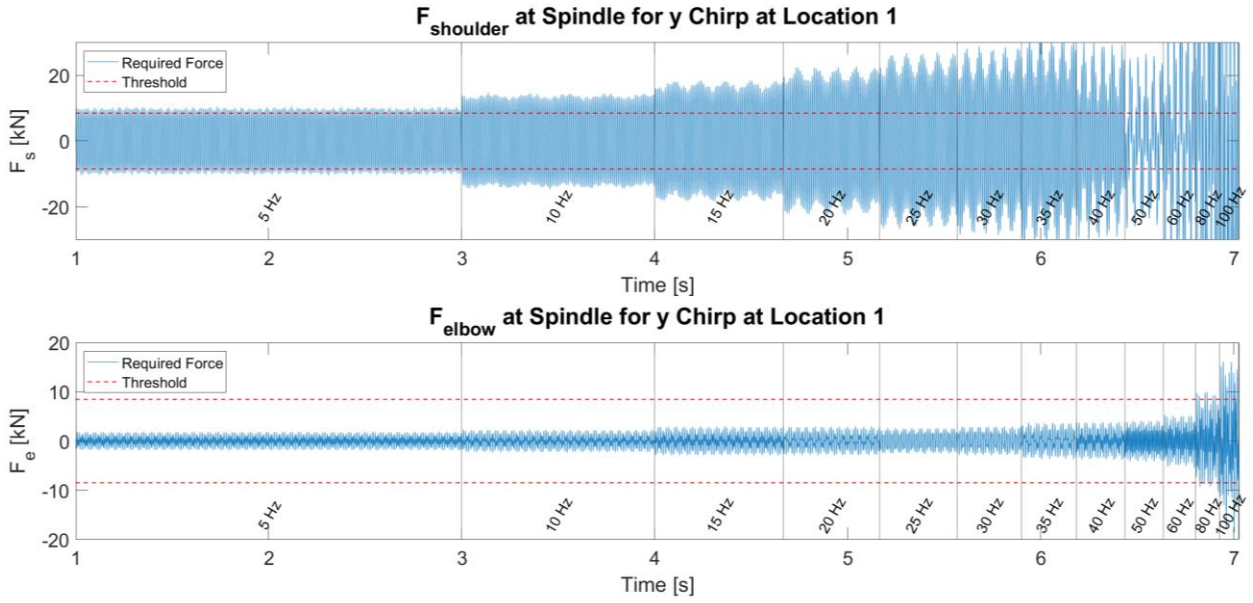
```
                    [num2str(freq(i-1)) ' Hz'],...
                    'HorizontalAlignment','center',...
                    'Rotation',60,...
                    'FontSize',par.fontLegend)
            end
        end
        ylim([testLocations(2,ceil(j/2))-5 testLocations(2,ceil(j/2))+5])
    linkaxes([axXz,axYz],'x');
    xlim([freqLine(zoomBegin) freqLine(zoomEnd)]);

    % Styling
    figPosZoom.Position = [300   150   1200   800];
    axXz.FontSize       = par.fontAxis;
    axYz.FontSize       = par.fontAxis;
    axXz.Title.FontSize = par.fontTitle;
    axYz.Title.FontSize = par.fontTitle;

saveas(gcf,[figPosZoom.Name '.png'])
saveas(gcf,[figPosZoom.Name '.fig'])

%% Figure Force Zoomed
figFZoom = figure('Name',['Zoomed SPACAR Chirp Force for ' ...
                          testDimensions(2-rem(j,2)) ...
                          ' Perturbation at Location ' ...
                          num2str(ceil(j/2))]);
        axSz = subplot(2,1,1);
            plot(tnom(zF(1):zF(2)),dataSPACAR.sim(j).Fshoulder(zF(1):zF(2))/1000)
            title(['Zoomed F_{shoulder} at Spindle for ' testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' num2str(ceil(j/2))])
            xlabel('Time [s]')
            ylabel('F_{s} [kN]')
            hold on
            plot(tnom(zF(1):zF(2)),maxS(zF(1):zF(2))/1000,...
                'r--','LineWidth',par.lineWidth)
            plot(tnom(zF(1):zF(2)),-maxS(zF(1):zF(2))/1000,...
                'r--','LineWidth',par.lineWidth)
            for i = 1:numel(freqLine)
                xline(freqLine(i));
                if i > 1
                    text((freqLine(i)+freqLine(i-1))/2,-15,...
                        [num2str(freq(i-1)) ' Hz'],...
                        'HorizontalAlignment','center',...
                        'Rotation',60,...
                        'FontSize',par.fontLegend)
                end
            end
            legend({'Required Force' 'Threshold'},'Location','northwest',...
                    'FontSize',par.fontLegend)
            ylim([-35000 35000]/1000)
        axEz = subplot(2,1,2);
            plot(tnom(zF(1):zF(2)),dataSPACAR.sim(j).Felbow(zF(1):zF(2))/1000)
            title(['F_{elbow} at Spindle for ' testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' num2str(ceil(j/2))])
            xlabel('Time [s]')
            ylabel('F_{e} [kN]')
            hold on
            plot(tnom(zF(1):zF(2)),maxS(zF(1):zF(2))/1000,...
                'r--','LineWidth',par.lineWidth)
            plot(tnom(zF(1):zF(2)),-maxS(zF(1):zF(2))/1000,...
                'r--','LineWidth',par.lineWidth)
            for i = 1:numel(freqLine)
                xline(freqLine(i));
                if i > 1
                    text((freqLine(i)+freqLine(i-1))/2,-15,...
                        [num2str(freq(i-1)) ' Hz'],...
                        'HorizontalAlignment','center',...
                        'Rotation',60,...
                        'FontSize',par.fontLegend)
                end
            end
            legend({'Required Force' 'Threshold'},'Location','northwest',...
                    'FontSize',par.fontLegend)
            ylim([-20000 20000]/1000)
        linkaxes([axSz,axEz],'x');
```

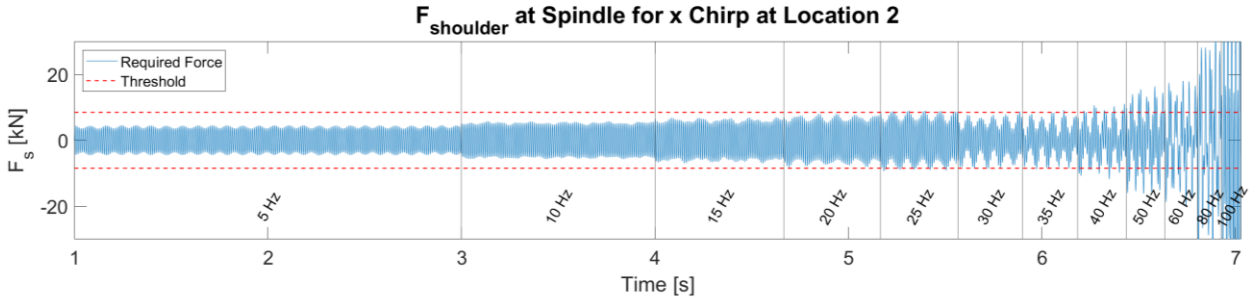
```
        xlim([freqLine(zoomBegin) freqLine(zoomEnd)]);

        % Styling
        figFZoom.Position  = [300   150   1200   800];
        axSz.FontSize        = par.fontAxis;
        axEz.FontSize        = par.fontAxis;
        axSz.Title.FontSize  = par.fontTitle;
        axEz.Title.FontSize  = par.fontTitle;

saveas(gcf,[figFZoom.Name '.png'])
saveas(gcf,[figFZoom.Name '.fig'])

%% Bode Plots
for j = j1:j2
    figBodeX = figure('Name',['SPACAR Chirp FRF in x for '  testDimensions(2-rem(j,2)) ...
                              ' Perturbation at Location '  num2str(ceil(j/2))]);
        axMagX = subplot(2,1,1);
            semilogx(1:sf,pow2db(abs(dataSPACAR.sim(j).XH)),...
                '-','LineWidth',par.lineWidth)
            hold on
            grid on
            semilogx(freq,pow2db(abs(dataSPACAR.sim(j).XH(1,freq+1))),...
                '-','LineWidth',par.lineWidth+1)
            semilogx(1:sf,ones(1,sf)*threshold,...
                'k--','LineWidth',par.lineWidth)
            title(['FRF in x Dimension for ' testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' num2str(ceil(j/2))])
            ylabel('Magnitude [dB]')
            xlabel('Frequency [Hz]')
            legend({'All Frequencies' 'Perturbed Frequencies' 'Threshold'},...
                'Location','southwest','FontSize',par.fontLegend)
        axPhaX = subplot(2,1,2);
            semilogx(1:sf,unwrap(angle(XH))*180/pi,...
                '-','LineWidth',par.lineWidth+1)
            hold on
            grid on
            semilogx(freq,unwrap(angle(XH(1,freq+1)))*180/pi,...
                '-','LineWidth',par.lineWidth+1)
            ylabel('Phase [degrees]')
            xlabel('Frequency [Hz]')
        linkaxes([axMagX,axPhaX],'x');
        xlim([2 sf]);

        % Styling
        figBodeX.Position  = [687   300   1000   650];
        axMagX.FontSize        = par.fontAxis;
        axPhaX.FontSize        = par.fontAxis;
        axMagX.Title.FontSize  = par.fontTitle;
        axPhaX.Title.FontSize  = par.fontTitle;

    saveas(gcf,[figBodeX.Name '.png'])
    saveas(gcf,[figBodeX.Name '.fig'])

    figBodeY = figure('Name',['SPACAR Chirp FRF in y for '  testDimensions(2-rem(j,2)) ...
                              ' Perturbation at Location '  num2str(ceil(j/2))]);
        axMagY = subplot(2,1,1);
            semilogx(1:sf,pow2db(abs(dataSPACAR.sim(j).YH)),...
                '-','LineWidth',par.lineWidth)
            hold on
            grid on
            semilogx(freq,pow2db(abs(dataSPACAR.sim(j).YH(1,freq+1))),...
                '-','LineWidth',par.lineWidth+1)
            semilogx(1:sf,ones(1,sf)*threshold,...
                'k--','LineWidth',par.lineWidth)
            title(['FRF in y Dimension for ' testDimensions(2-rem(j,2)) ...
                    ' Chirp at Location ' num2str(ceil(j/2))])
            ylabel('Magnitude [dB]')
            legend({'All Frequencies' 'Perturbed Frequencies' 'Threshold'},...
                'Location','southwest','FontSize',par.fontLegend)
        axPhaY = subplot(2,1,2);
            semilogx(1:sf,unwrap(angle(YH))*180/pi,...
                '-','LineWidth',par.lineWidth+1)
            hold on
            grid on
```

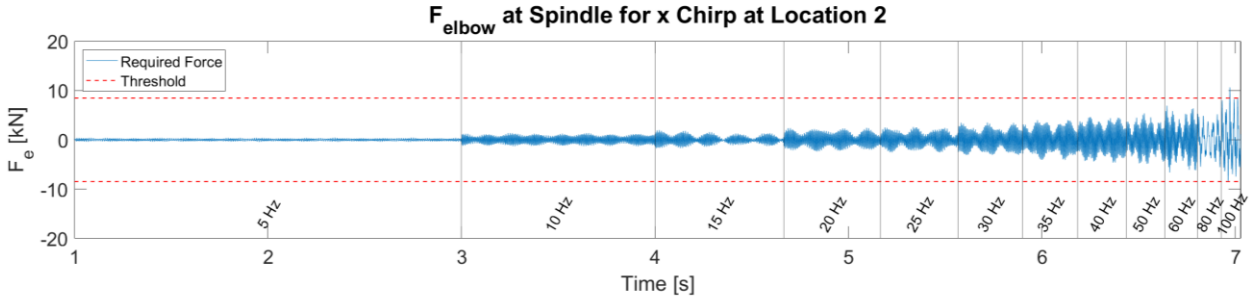
```
            semilogx(freq,unwrap(angle(YH(1,freq+1)))*180/pi,...
                '-','LineWidth',par.lineWidth+1)
            ylabel('Phase [degrees]')
            xlabel('Frequency [Hz]')
        linkaxes([axMagY,axPhaY],'x');
        xlim([2 sf]);

        % Styling
        figBodeY.Position  = [687   300   1000   650];
        axMagY.FontSize        = par.fontAxis;
        axPhaY.FontSize        = par.fontAxis;
        axMagY.Title.FontSize  = par.fontTitle;
        axPhaY.Title.FontSize  = par.fontTitle;

    saveas(gcf,[figBodeY.Name '.png'])
    saveas(gcf,[figBodeY.Name '.fig'])
end
```

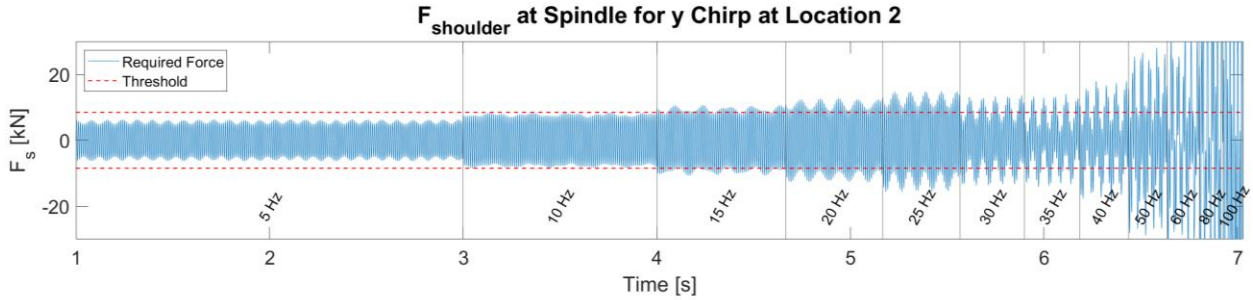
APPENDIX K. All SPACAR Simulation Force Plots

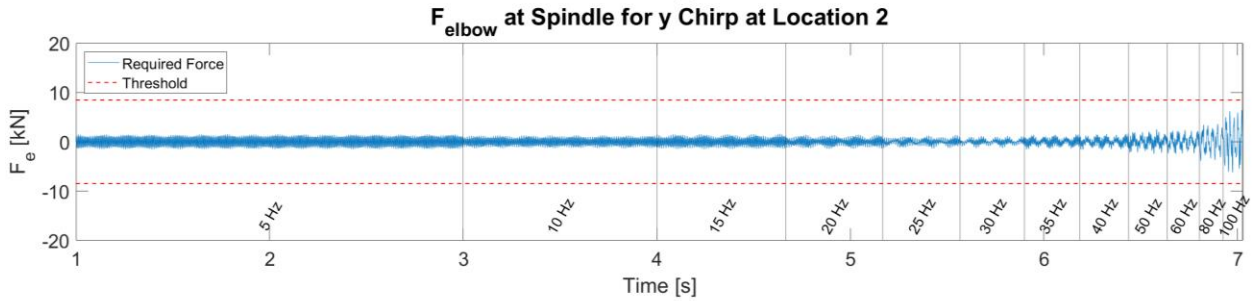


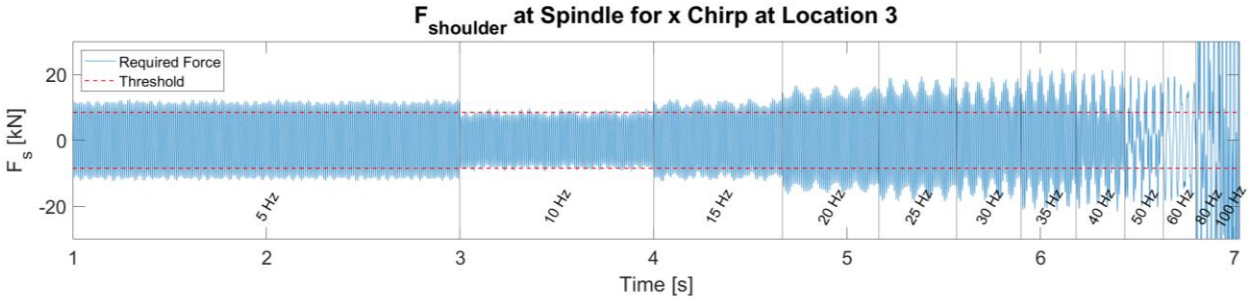


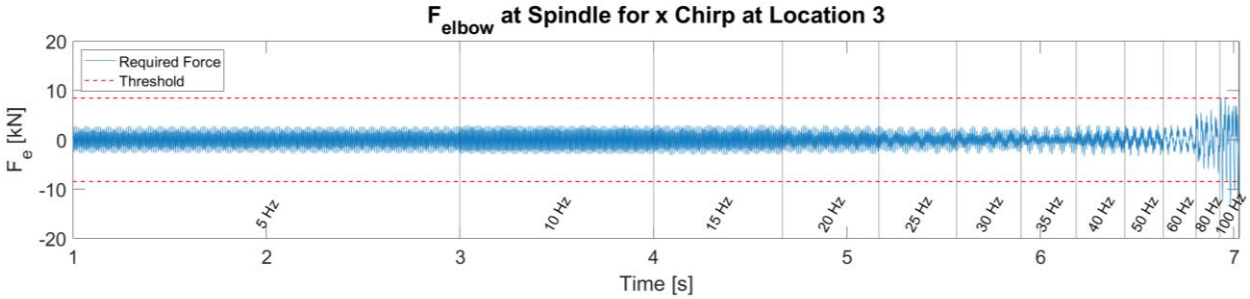

F shoulder at Spindle for x Chirp at Location 2
Required Force
Threshold
F s [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

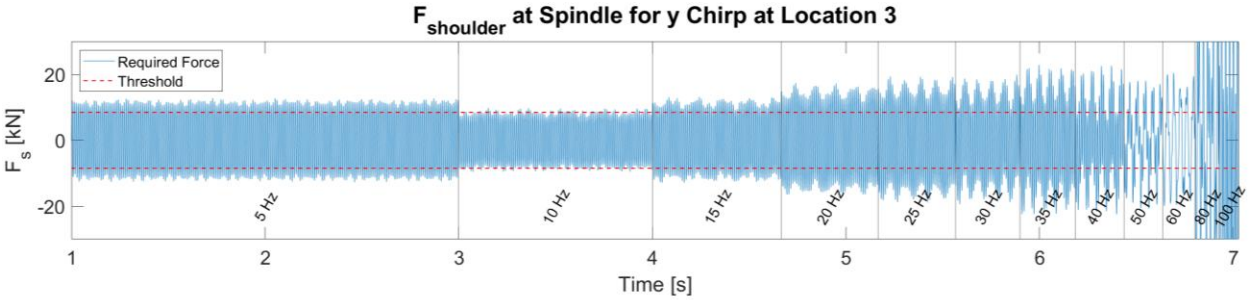

F elbow at Spindle for x Chirp at Location 2
Required Force
Threshold
F e [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

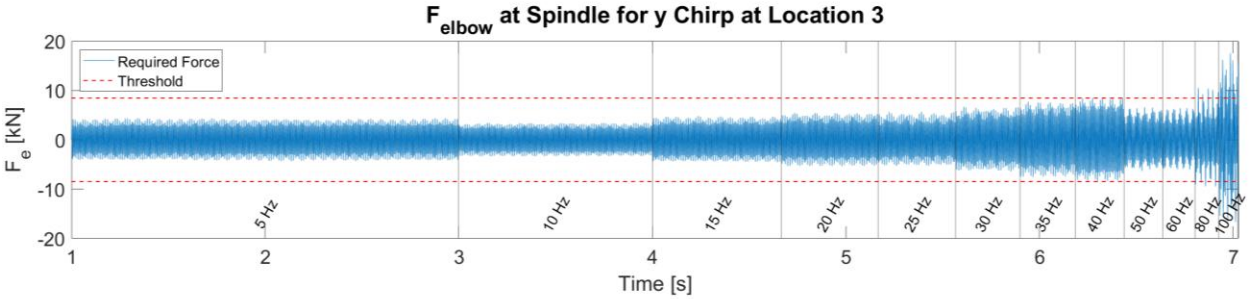

F shoulder at Spindle for y Chirp at Location 2
Required Force
Threshold
F s [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

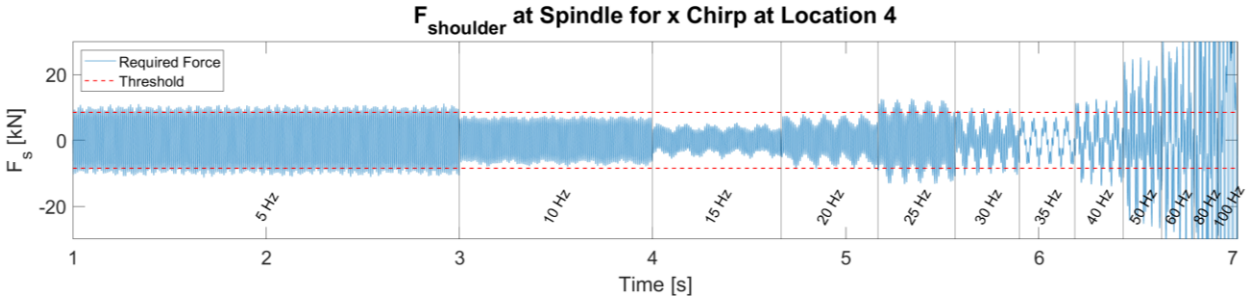

F elbow at Spindle for y Chirp at Location 2
Required Force
Threshold
F e [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

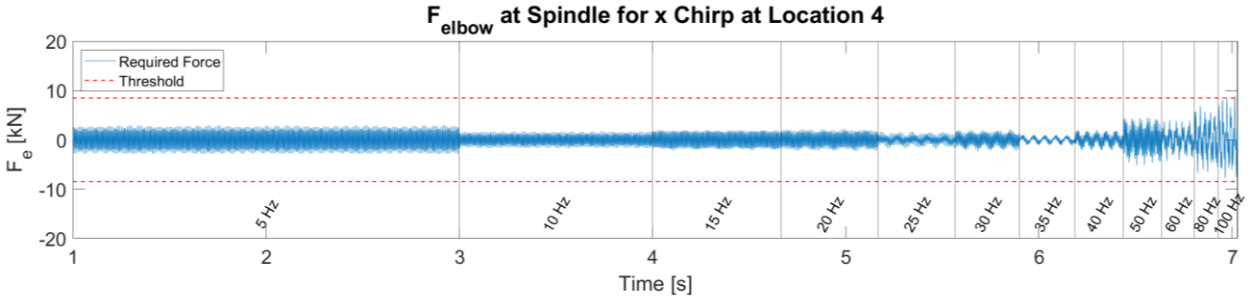

F shoulder at Spindle for x Chirp at Location 3
Required Force
Threshold
F s [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

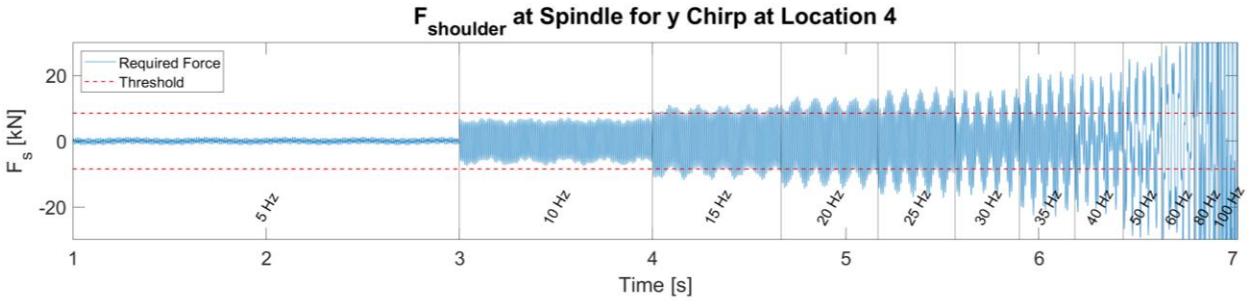

F elbow at Spindle for x Chirp at Location 3
Required Force
Threshold
F e [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

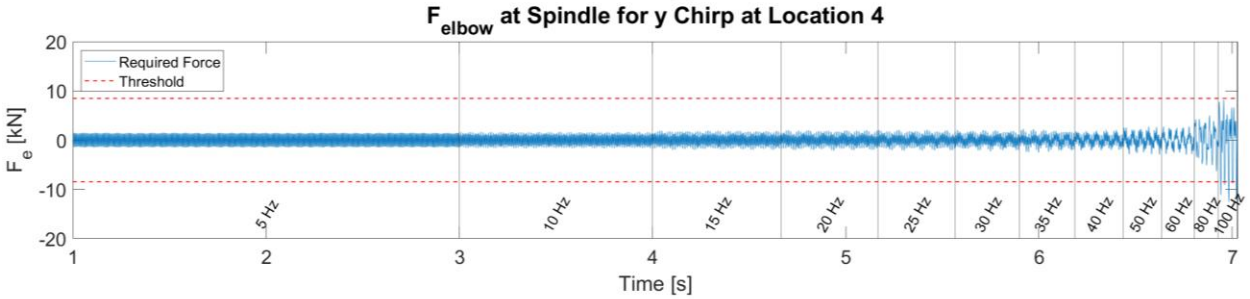

F shoulder at Spindle for y Chirp at Location 3
Required Force
Threshold
F s [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

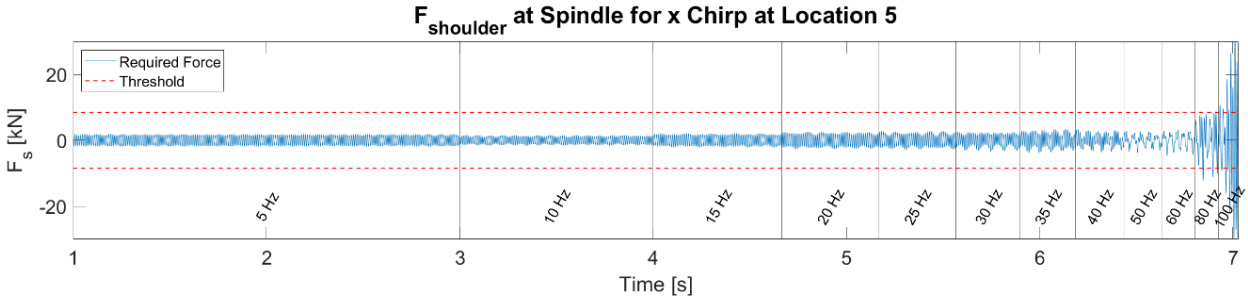

F elbow at Spindle for y Chirp at Location 3
Required Force
Threshold
F e [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz

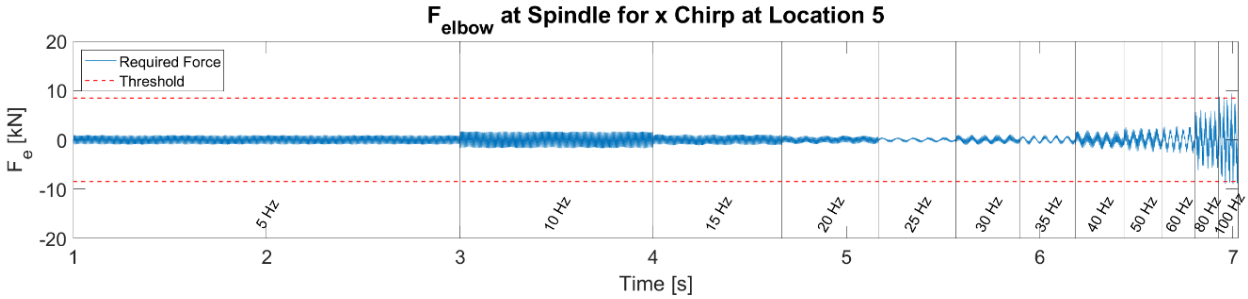

F shoulder at Spindle for x Chirp at Location 4
Required Force
Threshold
F s [kN]
20
0
-20
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz
1
2
3
4
5
6
7
Time [s]

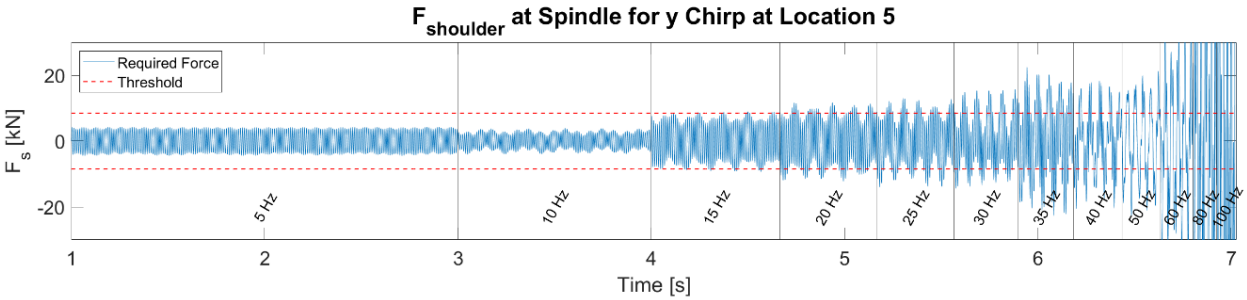

F elbow at Spindle for x Chirp at Location 4
Required Force
Threshold
F e [kN]
20
10
0
-10
-20
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz
1
2
3
4
5
6
7
Time [s]

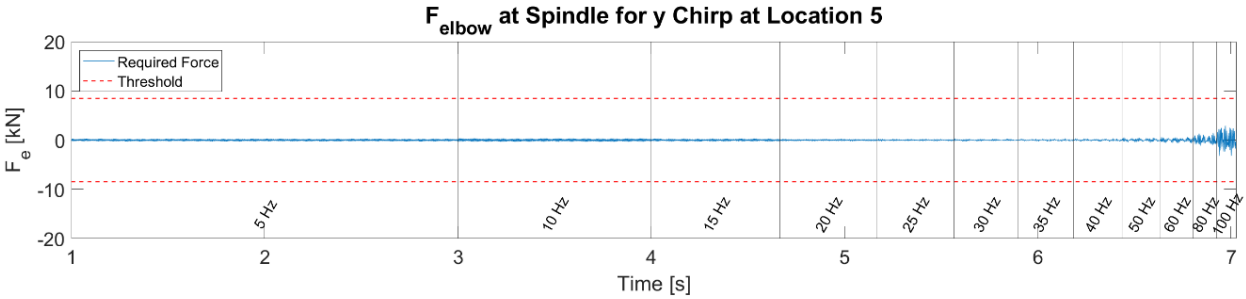

F shoulder at Spindle for y Chirp at Location 4
Required Force
Threshold
F s [kN]
20
0
-20
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz
1
2
3
4
5
6
7
Time [s]


F elbow at Spindle for y Chirp at Location 4
Required Force
Threshold
F e [kN]
20
10
0
-10
-20
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz
1
2
3
4
5
6
7
Time [s]


$F_{shoulder}$ at Spindle for x Chirp at Location 5
Required Force
Threshold
F_s [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz


F_{elbow} at Spindle for x Chirp at Location 5
Required Force
Threshold
F_e [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz


$F_{shoulder}$ at Spindle for y Chirp at Location 5
Required Force
Threshold
F_s [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz


F_{elbow} at Spindle for y Chirp at Location 5
Required Force
Threshold
F_e [kN]
Time [s]
5 Hz
10 Hz
15 Hz
20 Hz
25 Hz
30 Hz
35 Hz
40 Hz
50 Hz
60 Hz
80 Hz
100 Hz