Design and Development of a Force Control and Automation System for the VT-FRA Roller Rig

Jay Kailash Dixit

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science

In

Mechanical Engineering

Mehdi Ahmadian, Chair

Steve C Southward

Reza Mirzaeifar

June 19, 2018

Blacksburg, VA

Keywords: control systems, automation, roller rig, force control, cascaded loop control, motion control

Design and Development of a Force Control and Automation System for the VT-FRA Roller Rig

Jay Kailash Dixit

ABSTRACT

This study discusses the design of a force control strategy for reducing force disturbances in the Virginia Tech – Federal Railroad Administration (VT-FRA) Roller Rig. The VT-FRA Roller Rig is a state-of-the-art roller Rig for studying contact mechanics. It consists of a 0.2m diameter wheel and a 1m diameter roller in vertical configuration, which replicates the wheel-rail contact in a $^1/_4{}^{th}$ scale. The Rig has two 19.4 kW servo motors for powering the rotational bodies and six heavy-duty servo linear actuators that control other boundary conditions.

The Rig was operationalized successfully with all degrees of freedom working in the default position feedback control. During the Rig's commissioning, this approach was found to result in vertical force fluctuations that are larger than desired. Since the vertical force affects the longitudinal and lateral traction between the wheel and roller, keeping the fluctuations to a minimum provides a better test condition. Testing and data analysis revealed the issue to be in the control method. The relative position of the wheel and roller was being controlled instead of controlling the forces between them. The latter is a far more challenging control setup because it requires a faster dynamic response and full knowledge of forces at the interface. Additionally, force control could result in dynamic instability more readily than position control.

Multiple methods for force control are explored and documented. The most satisfactory solution is found in a cascaded loop force/position controller. The closed loop system is tested for stability and performance at various load, speed, and creepage conditions. The results indicate that the controller is able to reduce the standard deviation of vertical force fluctuations at the wheel-rail contact by a factor of four. In terms of power of the vertical force fluctuations, this corresponds to a 12 dB reduction with the force control when compared with the previous control method.

This study also explores the possibility of automating the tests in order to enable running a larger number of tests in a shorter period of time. A multi-thread software is developed in C++ for executing a user-defined position, velocity, or force vs. time trajectory, and for recording the data automatically. The software also provides continuous monitoring, and performs a safe shutdown if a fault is detected. An intuitive GUI is provided for constant data polling and ease of user operation. The code is modular in order to accommodate future modifications and additions for various testing needs.

The engineering upgrades included in this study, together with the baseline testing, complete the commissioning of the VT-FRA Roller Rig. With unparalleled parameter control and testing repeatability, the VT-FRA Roller Rig holds the promise of being used successfully for various contact mechanics needs that may arise in the railroad industry.

Design and Development of a Force Control and Automation System for the VT-FRA Roller Rig

Jay Kailash Dixit

GENERAL AUDIENCE ABSTRACT

Roller Rigs have seen widespread use around the world for research and development of railway vehicles. These test rigs are specialized machinery that provide the means to test a particular aspect of railroading in a controlled environment, allowing for thorough parametric analyses which aid in the design and development of railroad vehicles. One such test rig is the Virginia Tech – Federal Railroad Administration (VT-FRA) Roller Rig, located at the Railway Technologies Laboratory in Blacksburg, Virginia. It is a state-of-the-art test rig which is developed with the objective of providing a controlled test environment for studying railway contact mechanics. A good understanding of the wheel/rail contact is critical to railroad engineering, and this problem has been the subject of research for about a century now. Several compelling mathematical models have been proposed, but the experimental verification of those theories has proven to be difficult. Traditionally, field testing data has been utilized for comparison with prediction from the models. However, field tests are plagued by a low level of noise control and the inability to carry out sophisticated parametric analyses. VT-FRA Roller Rig holds the promise to fill this gap with its sophisticated electro-mechanical design and high precision instrumentation.

The VT-FRA Roller Rig replicates the wheel-rail contact in a $^{1}/_{4}^{th}$ scale by utilizing the INRETS scaling strategy. The locomotive wheel is replicated by a 0.2m diameter wheel and the tangent track is replicated by a 1m diameter roller. The relative size difference ensures that the contact distortion effects from the use of roller are kept to a minimum. The wheel and the roller are arranged in a vertical configuration, and are independently powered by two 19.4 kW servo motors. This enables the VT-FRA Roller Rig to achieve a precise creepage control of upto 0.1%. VT-FRA Roller Rig also has six heavy-duty servo linear actuators which are responsible for controlling four boundary conditions: cant angle, angle of attack, lateral displacement and vertical load. A sophisticated six-axis contact force-moment measurement system allows for precise measurements with a high dynamic bandwidth.

The Rig was operationalized successfully with all degrees of freedom working in the default position feedback control. During the Rig's commissioning, this approach was found to result in force fluctuations that were larger than desired. Since the vertical force affects the longitudinal and lateral traction between the wheel and roller, keeping the fluctuations to a minimum provides a better test condition. Testing and data analysis revealed the issue to be in the control method. The relative position of the wheel and roller was being controlled instead of controlling the forces between them.

This study documents the development process of a reliable force control methodology for the VT-FRA Roller Rig. Force control is a far more challenging control problem when compared to position control because it requires a faster dynamic response and full knowledge of forces at the interface. Additionally, force control could result in dynamic instability more readily than position control. Multiple methods for force control were implemented on the VT-FRA Roller Rig.

Satisfactory solution is achieved with the complicated cascaded loop force/position controller, and the stability and performance of the control system is ensured by a slew of tests at various operating conditions.

This study also explores the possibility of automating the tests in order to enable running a larger number of tests in a shorter period of time. A multi-thread software is developed in C++ for executing a user-defined position, velocity, or force vs. time trajectory, and for recording the data automatically. The software also provides continuous monitoring, and performs a safe shutdown if a fault is detected. An intuitive GUI is provided for constant data polling and ease of user operation. The code is modular in order to accommodate future modifications and additions for various testing needs.

The engineering upgrades included in this study, together with the baseline testing, complete the commissioning of the VT-FRA Roller Rig. With unparalleled parameter control and testing repeatability, the VT-FRA Roller Rig holds the promise of being used successfully for various contact mechanics needs that may arise in the railroad industry.

To my family

Acknowledgements

These two years of my graduate education have transformed me, both professionally and personally. The key enabler of this transformation has been my advisor, Dr. Mehdi Ahmadian. I deeply appreciate his support, frankness and the freedom to express and execute ideas while working under his guidance. The discussions with him have not only steered my project but have also shaped my personality, preparing me for the real world. I remain extremely thankful to him for being my advisor.

I would also like to thank the committee members, Dr. Steve Southward and Dr. Reza Mirzaeifar for their inputs and for taking the time to serve on my committee. I am especially thankful to Dr. Southward for his lessons and lectures that have played a key role in building my theoretical foundation and practical acumen in control systems and signal processing.

A very special thank you to the Federal Railroad Administration for providing me the required funding for this research. In particular, I would like to thank Mr. Ali Tajaddini for his time and support throughout the project. The discussions with him have been full of insights and ideas that have positively influenced the project.

I am very thankful to Karan Kothari who besides being my project partner, has been a wonderful and supportive friend. Dr. Milad Hosseinipour and Dr. Sajjad Meymand have been incredibly supportive and enthusiastic alumni and I'm thankful for their suggestions and inputs.

A big reason for the completion of this project is the unflinching and immense support of Dr. Andrew Peterson who has always remained accessible and has helped out with any difficulties that I have faced.

Sincere thanks go to other members of the CVeSS family for being awesome and providing a positive and cheerful work environment – Sara, Yang, Yash, Ahmad, Ashish, Dejah, Campbell, Timothy, Yongwen, Andrew, Nilesh, Yunbo, Iman, Xiaoyen, Zichen.

Most importantly, I would like to thank my family members, my mother, Hitesha Dixit and father, Kailash Dixit for standing strong in the face of the hardships. They have faced incredible hardship for the past two years but they made sure none of that got in the way of my graduate education. I am also deeply indebted to my cousin, Aalok Trivedi for providing me with emotional and economic support in the direst moments of need.

Jay Kailash Dixit
July 2018
Blacksburg, VA

Contents

List of Figures

1. Introduction

1.1 Overview of the VT-FRA Roller Rig

The VT-FRA Roller Rig is a state-of-the-art testing facility for testing of railway contact mechanics and dynamics. It consists of a wheel and a roller in a vertical configuration that replicates the wheel-rail contact in a $1/4^{th}$ scale. The dimensional scaling is done scientifically using the INRETS scaling strategy for a close correspondence with the contact in the field. Dimensionally, the roller is five times larger than the wheel which ensures that the contact distortion is minimum [1].

The sensing and actuation hardware on the Rig is unparalleled in sophistication as far as railway test rigs are concerned. The wheel and the roller are each powered by two 19.5 kW servo motors, which are the largest and the most powerful motors off-the-shelf catalog offerings from Kollmorgen. Six servo linear actuators with low-friction ballscrew drives are used for motion along the four linear degrees of freedom. The Rig can set control all the major contact boundary conditions as shown in Table 1-1.

Table 1-1: Controlled boundary conditions on the VT-FRA Roller Rig

Boundary Condition	Actuation Hardware
Locomotive Speed	Roller rotation by Kollmorgen AKM-84T servo motor
Creepage	Wheel rotation by Kollmorgen AKM-84T servo motor
Vertical Load	Cradle displacement by two Kollmorgen EC4 linear actuators in gantry
Rail Cant	Cradle rotation about cant pivot by two EC4 linear actuators in gantry
Angle of Attack	Roller turntable rotation via one off-center EC4 linear actuator
Lateral Displacement	Roller frame displacement by one EC4 linear actuator

The degrees of freedom are visually represented in Figure 1-1. The independent wheel-roller drivelines allow for accurately controlling creepages to within 0.01%. The angle of attack and the rail cant angle can be maintained within +/- 0.1^{o} , and the lateral position can be maintained to within 0.1 mm, even under high lateral loads.

The custom contact force-moment measurement system allows for measurements up to 32 kN of vertical load, and 16 kN of longitudinal and lateral loads. The load measurements are done with an accuracy of +/- 50N. The data acquisition system operates at a sample rate of 2000 Hz, which can be changed to a maximum of 16 kHz. This provides a high Nyquist frequency to detect and analyze very high frequency dynamics without aliasing effects.

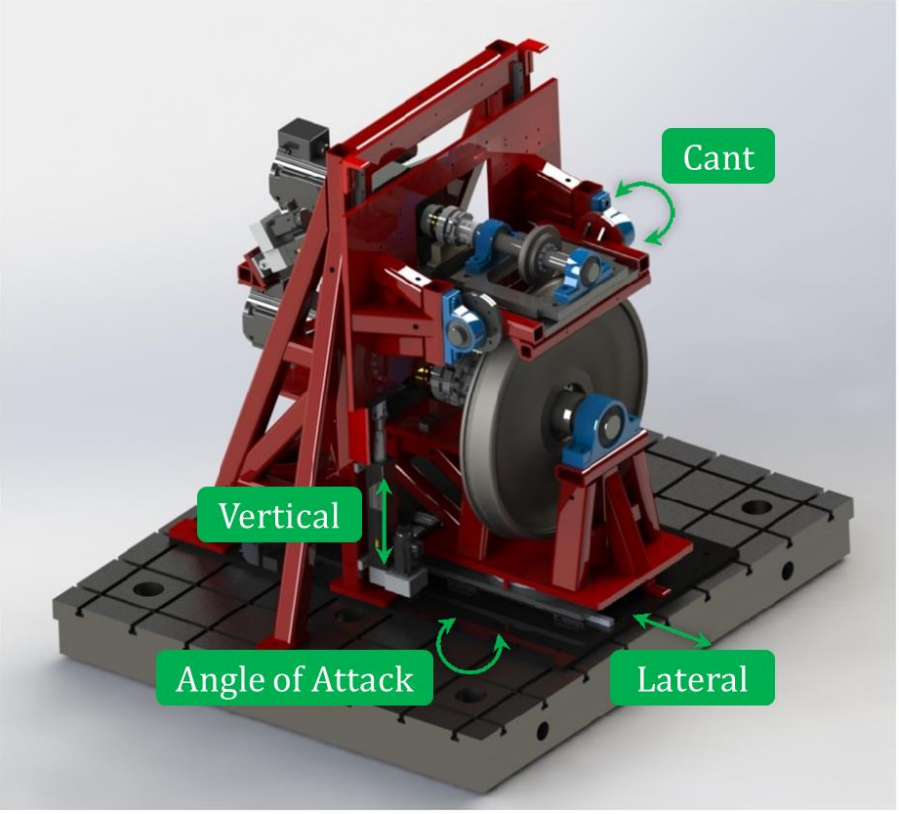


Figure 1-1 Visualization of the degrees of freedom on the VT-FRA Roller Rig

1.2 Motivation

Commissioning tests on the VT-FRA Roller Rig were initiated with the objective of generating creep-creepage curves with a high degree of parameter control and fine creepage resolution. These tests revealed some areas of improvement on the Roller Rig which include:

- Sensor drift in data from the contact force-moment measurement system,
- Need for a standard conversion and processing protocol for raw sensor data,
- Need for a standard operating procedure for testing on the Roller Rig,
- High vertical contact force fluctuations when wheel and roller were in motion, and
- High degree of manual labor involvement for creep-creepage testing.

A standard operating procedure was established after iterative experimentation to ensure statistically significant measurements and independence of boundary condition variables. A two-man team was set up for managing the experimentation, data collection, data processing and result reporting. However, engineering solutions were required for resolving sensor drift, vertical contact force fluctuations, and automation of the Roller Rig.

1.3 Objectives

The primary objectives of this study are to:

1. Improve the accuracy and repeatability of tests on the VT-FRA Roller Rig.
2. Implement force control methods to significantly reduce the vertical force fluctuations that commonly result from position control.

3. Evaluate and compare the proposed force control methods with position control to assess the extent of improvements.
4. Provide some level of automation in an effort to reduce the user involvement in setting up and running tests manually.
5. Suggest any additional improvements that could be made toward improving the accuracy and ease of use of the VT-FRA Roller Rig.

1.4 Approach

The general approach towards the solutions is mentioned as follows:

- Identify the end-goal requirements from the system,
- Conduct a system identification to identify the input-output relationship,
- Identify the potential sources of error through design examination and/or modeling,
- Conduct experiments to investigate the error sources and assign severity levels,
- Explore the available literature for solutions to similar problems,
- Design a solution based on the engineering first principles,
- Implement the solution and examine the effectiveness,
- Reiterate the previous two steps until a satisfactory performance is obtained.
- Integrate the solution into the testing workflow of the Rig, and
- Document the solution for future knowledge transfer.

1.5 Contributions

The contributions of this study are as follows:

- Evaluated, identified, and successfully implemented a force control method on the VT-FRA Roller Rig for reducing the force fluctuations that result from minute surface imperfections of the roller and/or wheel. Standard deviation of vertical force fluctuations is reduced by a factor of four which corresponds to a 12 dB reduction in power level.
- Significantly revamped the MechaWare™ software that is used for controlling the Rig's servomotors to improve setting up tests, running multiple tests back to back, and collecting data with far more efficiency than the earlier setups.
- Established the robustness and stability of the proposed control method through a large slew of tests over the entire operational range of the Rig.
- Delivered a modular automation framework that can be scaled up for automatic motion execution and data collection while providing continuous monitoring of the Rig.

1.6 Outline

This document is divided into seven chapters, each of which fouceses on a particular aspect of the study.

Chapter 1 provides an overview of the VT-FRA Roller Rig that is the subject of this document. Objectives and contributions from this study are described.

Chapter 2 provides the description of the contact force-moment measurement system, with an analysis of the measurement noise and drift characteristics.

Chapter 3 explores the problem of vertical force control design and details all the methods that were employed, along with the pros and cons of each. The design and analysis procedure are presented, including the results from an exhaustive testing exercise.

Chapter 4 discusses the design and development of the automation software for the VT-FRA Roller Rig. The software control and logic flow diagram are presented, and the architecture is discussed with the usage expectations. Finally, a line-by-line code explanation is provided, emphasizing the modularity of code for easy understanding by the reader.

Chapter 5 provides the summary of the study and discusses the enhanced capabilities attained by this study. The chapter concludes by recommending future steps for development.

Appendix A details the features of the control system design software, MechaWare™, and provides a detailed guide about control system design for the Roller Rig.

Appendix B explains the Motion Programming Interface, which is the backbone of the control software for the VT-FRA Roller Rig. Control hierarchy is presented, and control software design methods are described in detail.

2. Force-Moment Measurement System

2.1 Introduction

The Rig features a custom designed force-moment measurement system to measure contact forces and moments to a high degree of accuracy. This is the most important component of the Roller Rig data acquisition system, and a proper understanding of the system is needed for conducting reliable measurements. Improper configuration of the system may lead to issues such as measurement drift and high frequency noise. Section 2.2 provides an overview of the piezo-electric force measurement concept to enable a better understanding of the system settings.

The novel design of the Rig's force-moment measurement system is discussed in Section 2.3. The discussion begins with an explanation of the load path, followed by the placement of the eight tri-axial load cells and summation of channels. A good understanding of these is required for choosing a proper data acquisition approach and experiment design. Section 2.4 explains the multi-channel charge amplifier. The amplifier contains numerous settings which affect the measurement characteristics, so a thorough understanding of the device is extremely important. Section 2.5 discusses the measurement channels and their function; refer to this Section to learn about acquiring raw data from the Rig's ADCs.

Typical measurements on the VT-FRA Roller Rig are quasi-static; therefore, the measurement settings on the charge amplifier must be configured accordingly. Sensor drift was a critical issue that was discovered during the commissioning process. Section 2.6 provides a comparison between the drift characteristics of the sensor, before and after it was resolved. This Section also provides guidelines about unloading and resetting the sensors, which should be factored in while conducting an experimental design.

2.2 Piezoelectric Load Sensing

Piezoelectric load cells work by utilizing the piezoelectric effect. The effect is shown by certain solid materials such as quartz where electric charge accumulates when the material is subjected to stress. The load cells used on the Rig dynamometer are 3-component Kistler piezoelectric load cells (type 9027C & 9028C) [1]. The load cell on the Rig and the principal measurement axes are shown in Figure 2-1.

Figure 2-1 Kistler piezoelectric load cell type 9027C with load directions shown [2]

The working principle of the load cell is shown in Figure 2-2. The load cell consists of three pairs of quartz plates in a cylindrical capacitor-like fashion. The force is applied on the top plate and is distributed to the quartz plate pairs. The disks are aligned in such a way that each pair reacts to the load in one direction. Thus, a load in any direction is resolved into three components: X, Y and Z. The exterior of the sensor is grounded and the sensing leads of each force component are placed in the gap between each pair of quartz plates (Figure 2-2). Application of force on the quartz plates creates a positive or a negative charge on them, depending on the stress direction. This creates an electric field in the gap between two plates and causes the conductor to develop a charge. As a result, the flow of electrons takes place in the wire, which becomes a measure of force.

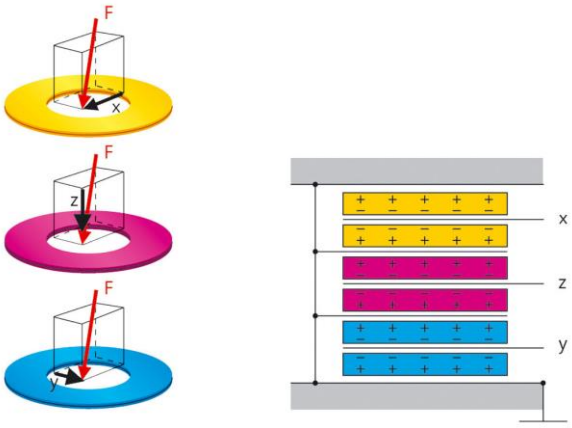


Figure 2-2: Working principle of the Kistler piezoelectric load cell [2]

A single piezo-electric sensor can be represented as a charge source with a shunt capacitor and a resistor (Figure 2-3). The charge produced depends on the force applied and the piezoelectric constant of the material. Some charge generated accumulates on the capacitance, producing voltage, while the remaining charge leaks through the resistor. It is to be noted that a static load produces a static charge. This means that if a static load is applied on a piezoelectric material, the qp will generate a definite amount of charge which will flow through the wires and either deposit on the capacitor or dissipate through the resistor. Unless the loading conditions change, the qp will not generate any more charge.

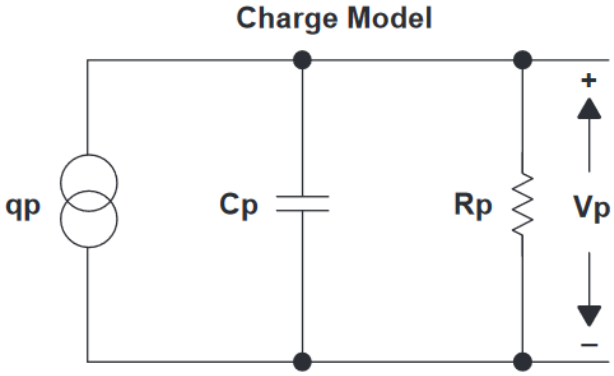


Figure 2-3: Model of a piezoelectric sensor

Thus, if a piezoelectric load cell is used to measure static force, then a certain amount of charge will deposit on the capacitor, and the capacitor will discharge through the resistor. This causes "charge leakage" to happen and we see a "sensor drift."

An important factoer to note is that the charge can also leak via environmental contact and hence, to ensure a noise-free measurement, Kistler recommends an insulation resistance of $10^{14}\ \Omega$ and not less than $10^{13}\ \Omega$ [2]. This also means that moisture should be avoided around piezoelectric sensors, and that the connections should be as dust-free as possible.

If the connection wires are well insulated and the contacts are clean and dry, the internal impedance of the sensor (Rp in Figure 2-3) will dictate the charge leakage rate. Choosing a high resistance value for Rp would decrease the charge leakage rate. Normally, a piezoelectric sensor generates so little charge that in order to obtain a meaningful measurement, a charge amplifier must be used. A typical charge amplifier circuit and its frequency response is shown in Figure 2-4. Notice the high pass behavior in the low frequency region.

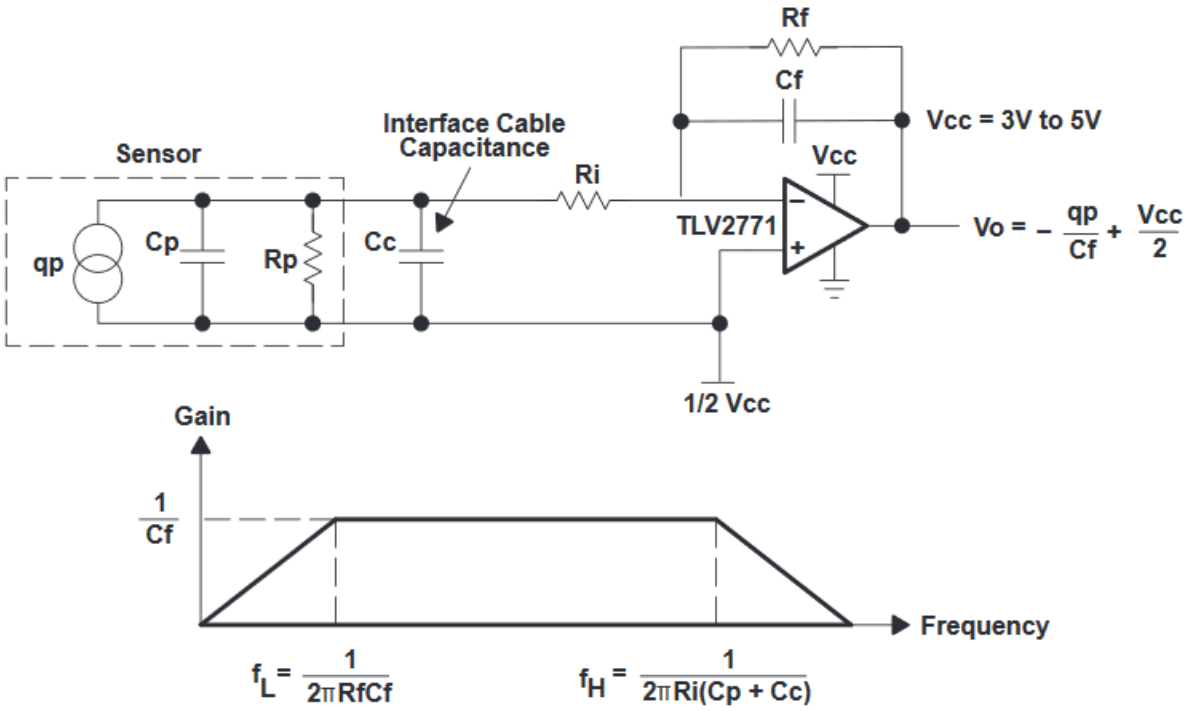


Figure 2-4: Charge amplifier circuit and frequency response [3]

The lower cut-off frequency depends on the resistor, Rf which is switched in parallel to the capacitance, Cf. The higher cut-off frequency depends on the capacitances Cp and Cc, and resistance Ri. The lower cut-off is commonly referred to as the "high pass" of charge amplifier, and the upper cut-off is commonly referred to as the "low pass" of the charge amplifier. Every charge amplifier provides the settings to change these. It is important to understand the type of measurement being done in order to configure these settings.

2.3 Load Measuring Platform Design

The load path for the wheel-roller contact forces is shown in Figure 2-5. The forces that propagate along the green load path react at the wheel dynamometer and the forces that propagate along the blue path get reacted at the motor dynamometer [1].

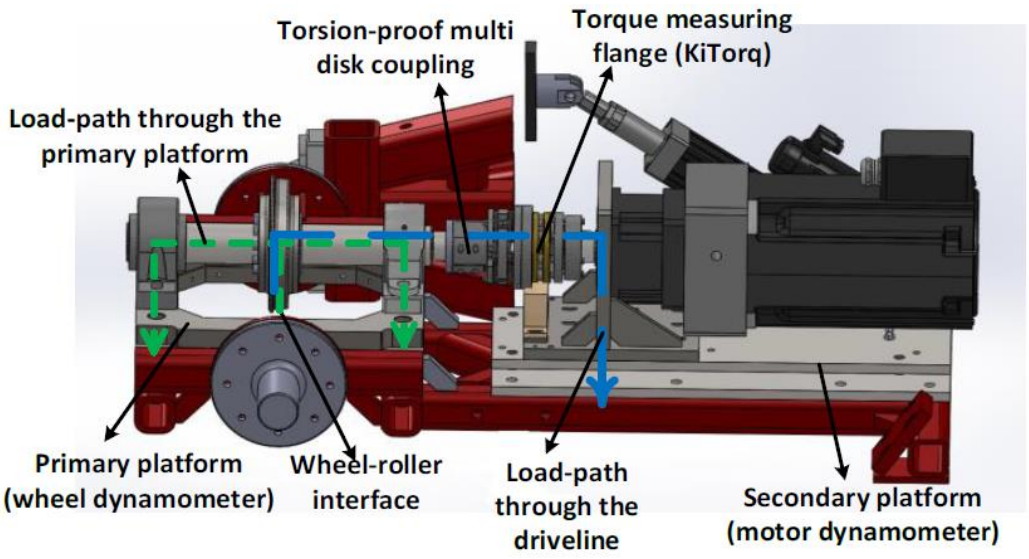


Figure 2-5: Load path for contact forces [1]

Hence, there are two dynamometers: one under the wheel bearings and another under the wheel motor. The force measurements from both the dynamometers must be added in order to obtain the total force acting on the contact. Assuming that the two dynamometer readings are in phase, this summation strategy also eliminates any imbalance forces caused due to angular misalignments in the driveline. The construction of the dynamometers is simple: there is one load cell at each corner of the dynamometer. The load cell arrangement is shown in Figure 2-6, which is the top view of the dynamometer. It should be noted that the load cell types 9027C and 9028C are identical in all technical specifications, except for the position of the wiring outlet relative to the sensor [2]. Each load cell is capable of measuring forces along the three directions – longitudinal, lateral and vertical. However, this arrangement of load cells provides the means to compute contact moments along the three axes. For example, if the component of moment along the X-axis, in z-direction is required, it can be computed as $M_{xz} = bF_{x,1} + bF_{x,2} + bF_{x,3} + bF_{x,4}$.

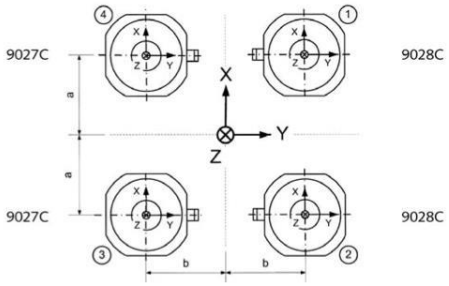


Figure 2-6: Load cell arrangement for the 3-axis force-torque measuring dynamometer [1]

Of course, the assumption here is that the contact is exactly in the middle of the dynamometer, which is not always the case on the Rig. If the distance along y-direction from contact is changed to b1 and b2, then the M_{xz} calculated earlier now becomes:

$$M_{xz} = b2(F_{x,1} + F_{x,2}) + b1(F_{x,3} + F_{x,4})$$

The distance from the center to the left load cells is b1, and the distance from the center to the right load cells is b2 (along the Y-axis). The equation above suggests that even if the contact

is not placed symmetrically with respect to the dynamometer, two force channels can be combined. This means less cabling and ultimately less noise corruption. Forces along the y-direction can be summated to in a manner similar to the x-channels. Hence, the outputs can be reduced to 8 channels from 12 channels, which is illustrated in Figure 2-7. The summation of signals is done by a component known as a summing box. The Roller Rig uses a summing box from Kistler, type 5417 [1]. There are two such summing boxes on top of the Rig, one for each dynamometer. From the summing box, the eight channels go to the charge amplifier (Kistler, type 5070A) where the charge in the channel cables is amplified and converted into an analog voltage signal. This analog voltage can be converted to a digital value recognizable by a computer by using an analog to digital converter (ADC). The charge amplifier contains all the configurable settings for the contact force-moment dynamometer.

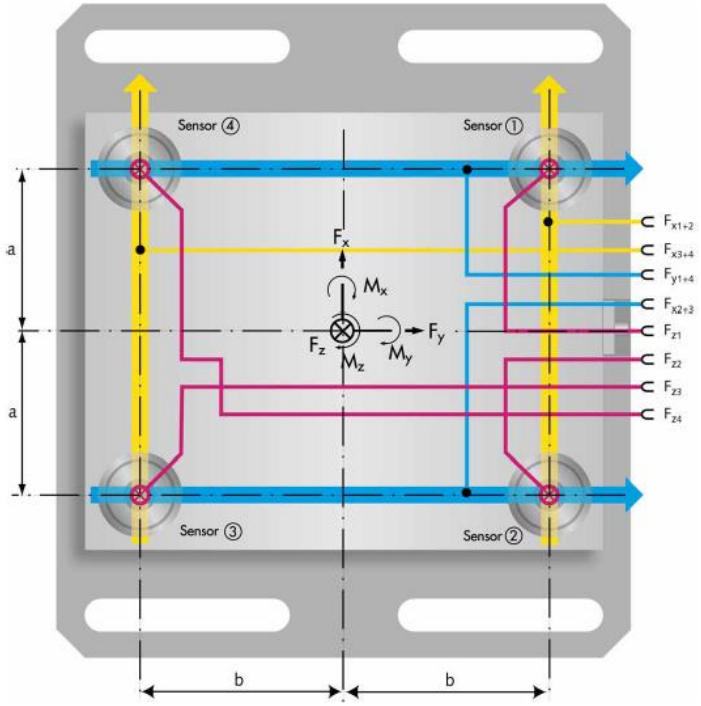


Figure 2-7: Channel outputs from a 4-post tri-axial load cell dynamometer [2]

2.4 Multichannel charge amplifier

Piezoelectric load cells generate very low amounts of charge when a load is applied to them. For example, the load cells in the Rig dynamometer have a sensitivity of about 7.5 pC/N. This means that 7.5×10^{-12} Coulombs of charge is generated when 1N of force is applied in a direction. Such low values need to be amplified and then converted into a signal which can be utilized for data acquisition. Charge amplifiers are used for this task, such as the one shown in Figure 2-8.

A charge amplifier is a powered (active) component that is connected to the main power supply. It uses the supply power to amplify the input charge from the piezoelectric sensor and convert it into a proportional analog voltage output. The Rig uses two 8-channel charge amplifiers, which are specifically designed for dynamometer systems like those on the Rig.

The charge from the summing box comes in a special 9-pin connector which is located in the back of the amplifier, as shown in Figure 2-9. This connection should never be handled unless

the entire system is being overhauled. Proper procedures [4] should be followed and Kistler support should be contacted before handling this connection. The output from the charge amplifier is provided via an RS-232 connection, and grey "printer-like" serial cables can be seen going out from the charge amplifier and into one of the SQIO boards.

Figure 2-8: Kistler type 5070A charge amplifier used in Roller Rig force-moment measuring system

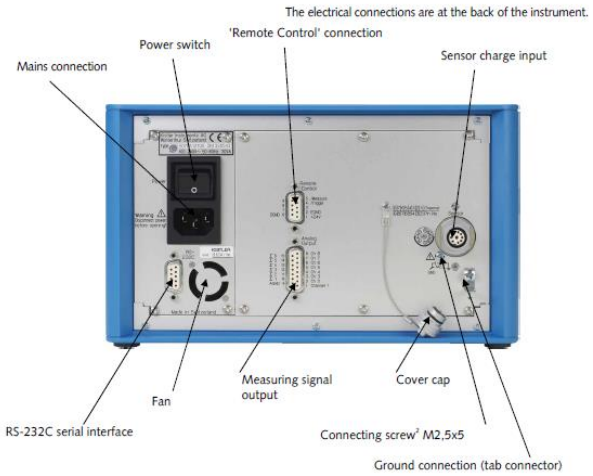


Figure 2-9: Rear side of the charge amplifier box

The charge amplifier has an LCD display which shows menu options along with the current value of force, the minimum and the maximum value of force measured on that channel. The menu options can be accessed by the blue dial supplied: turning it highlights the setting option, and pushing (clicking) on it selects the highlighted option. There are two push buttons adjacent to it, named "F" and "Meas." "F" is a programmable button which is currently programmed to change channels when pushed. The setting for "F" can be changed by the manual [4]. The green "Meas" button starts the force measurement.

Pressing the "Meas" button once causes the green "Meas" LED to light up, and the user can then see the force values being displayed by the LCD screen. Pressing it again causes the "Meas" LED to turn off and the measurement ceases. This is shown graphically in Figure 2-10. It is important to note that the measurement done by the charge amplifier is not absolute. Pressing the "Meas" button to start measurement also sets the reference point to zero. It is recommended that the user start the measurement when the Rig is not in motion and also not in contact so that the reference is set at the desired baseline.

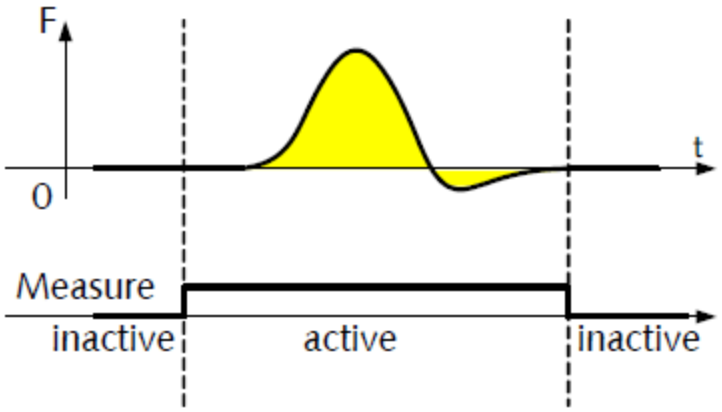


Figure 2-10: Typical measurement cycle. Measure inactive = Green LED OFF

The red LED lights up if there is any error with the charge amplifier. If this happens, the display should show an error message. The error can be reset by pressing the “Meas” button. The display and the description of menu items is shown in Figure 2-11 and detailed in Section 2.4.1.

2.4.1 Charge amplifier display settings

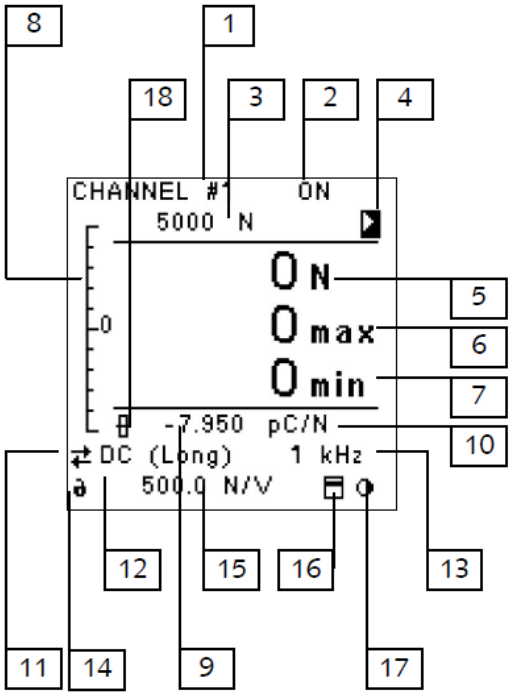


Figure 2-11: Charge amplifier display features marked with number tags

Table 2-1: Charge amplifier display items

Item	Description
1	Selected channel. Channels 1-8 can be selected. All settings and readings displayed on the screen are for the channel which is labeled here.
2	Actuate Output. This setting can be used to disable output from the current channel
3	Measuring range. The maximum absolute force that can be measured by the selected channel
4	Additional menus (refer to manual [4])
5	Current value measured by the channel
6	Maximum value measured by the channel since the “Meas” button was pressed
7	Minimum value measured by the channel since the “Meas” button was pressed
8	Bar graph display. Graphical bar showing the measured value with respect to the range of values measurable
9	Sensitivity value of channel. Calibration certificate of individual load cells provide sensitivity values for each force measurement axis.
10	Defining the unit of sensitivity. This is pC/N as given in calibration certificate
11	“Remote Control” status display: Not used for Roller Rig
12	Setting the measuring mode (time constant of high pass filter) (Section 2.4.2)
13	Low-pass filter setting (anti-alias filter) (Section 2.4.4)
14	Locking/unlocking the settings for adjustment
15	Voltage output scaling. Depends on the measuring range and cannot be modified

16	Not used for Roller Rig
17	Contrast setting of the LCD screen
18	Zero point setting of the bar graph display

The most important settings here are measuring range, measurement mode (high pass filter) and low pass filter (anti-aliasing filter). These should be changed and set for each channel.

2.4.2 Measurement range

The output from the charge amplifier is an analog signal which can have a value of +/-10V. The entire force measurement range is scaled in a way so as to generate a voltage that lies between -10V to +10V. So in Figure 2-11, for a range of 5000N means that an output of 1V would mean a force measurement of 500N. The maximum range available for our dynamometer is 8000N per channel. It is wise to choose a measuring range option that is higher than the highest force possible at contact, so as to avoid overloading.

However, setting the range too high would mean a less accurate and possibly a noisier force measurement. Every analog output has a fluctuation about a mean value. Suppose the output value fluctuates about +/-0.1V. A 0.1V fluctuation would appear as a +/-50N measurement noise for a voltage scaling of 500N/V (range of 5000N). The same would appear as a +/-5N measurement noise for a range of 500N at 50N/V. It is advised that the range be set just above the expected maximum force value during experiments.

2.4.3 Measuring mode (high pass filter time constant)

The frequency response for a piezoelectric sensor with a charge amplifier is shown in Figure 2-4. Piezoelectric sensors cannot measure DC values due to the charge leakage. The rate of this charge leakage depends on the impedance of the circuit. This charge leakage shows up as a high pass filter at low frequencies on a frequency response plot. The charge leakage practically can never be zero but it can be reduced by changing the value of the resistor Rf.

The measuring mode setting does that. There are two measuring modes available: short, which has a time constant τ = 340s and a DC (long) mode which has a time constant τ >= 10000s. Time constant is the amount of time elapsed for the measurement value to drop by 36.8% of the actual physical force measurement. These settings in the charge amplifier do not mean that the force measuring dynamometer will have such time constants. Actual decay rates have to be experimentally determined. This has been done for the Rig in Section 2.6. DC (long) measuring mode should be suitable for most applications on the Roller Rig.

2.4.4 Low pass filter (Anti-alias filter)

There is a roll-off in gain plot at high frequencies in frequency response of a piezoelectric sensor with a charge amplifier (Figure 2-4). Piezoelectric sensors generate a charge separation with virtually zero relative displacement. This high stiffness causes the roll-off break frequency to be high, enabling a high measurement bandwidth. The bandwidth can be set to a user-defined frequency by explicit addition of a low pass filter. An analog 2nd order Butterworth low pass filter comes with the charge amplifier with some preset options for break frequency. This filter can be used as an anti-alias filter to "band-limit" a signal.

A band-limited signal means that the signal does not have any significant spectral power content above a certain frequency which is less than the Nyquist frequency. Real-life signals are

seldom "band-limited" and an anti-alias filter is commonly used for such a task. If the signal is not "band-limited," aliasing may happen. Aliasing refers to the phenomenon where the signal of a particular frequency is misinterpreted for a signal of lower frequency due to a low digital sampling frequency. If Fourier transform of a non-band limited signal is taken, power peaks may show up at "aliased" frequencies if the sampling frequency is low.

An anti-alias filter should be used for "band-limiting" a signal. An appropriate anti-alias filter would be a low pass filter which causes an attenuation of 20dB or more at the Nyquist frequency. It should be noted that an anti-alias filter introduces a delay in feedback which manifests itself as phase decay in a feedback control system. This becomes an important consideration for the Rig's vertical force feedback controller. Phase decay causes a reduction of phase margins and can impact the stability of the closed loop control. This is discussed in detail in Section 3.12.3. The phase delay in force measurement could also be a cause for concern if the data is being collected for the purpose of machinery fault diagnosis.

2.5 Data Acquisition

The data acquisition from the charge amplifiers is undertaken via the analog RS-232 connection. The output from the charge amplifier is fed into the "analog input" pins of the SQIO-MIXEDMODULE nodes 10 and 11. The analog input is connected to an ADC: Analog to Digital Converter, which converts the analog signal into a series of digital bits which can be stored in the network memory and accessed by the controller or the workstation applications.

The SQIO-MIXEDMODULE board contains 16 differential analog inputs. The allowable analog signal value is +/-10V. Each analog signal is addressed by 16-bits, with one bit for sign [5]. Simply put, a +/-10V signal from the charge amplifier is converted into a digital 16-bit signed integer value ranging from -32768 to +32767. So the following conversion formula is used to obtain force from data acquisition counts:

$$Actual\ Force\ (N) = \{Channel\ Measurement\ Range\ (N)\} * \frac{Analog\ input\ in\ counts}{32767}$$

The channel measurement range is set up in the charge amplifier. For example, if the analog input reads a value of -4000 for a channel and the measurement range for that channel is 8000N, then -4000 "counts" of force is really $8000 * \left(\frac{-4000}{32767}\right) = 976.59\ N$. Each channel should be converted like this. The measurement ranges for channels measuring summation of forces ($\Sigma F_x, \Sigma F_y, \Sigma F_z$) is the sum of ranges of individual channels.

2.5.1 Current list of measurement channels in Roller Rig data acquisition

The individual load cells on the Roller Rig contact force-moment measurement are numbered as shown in Figure 2-12.

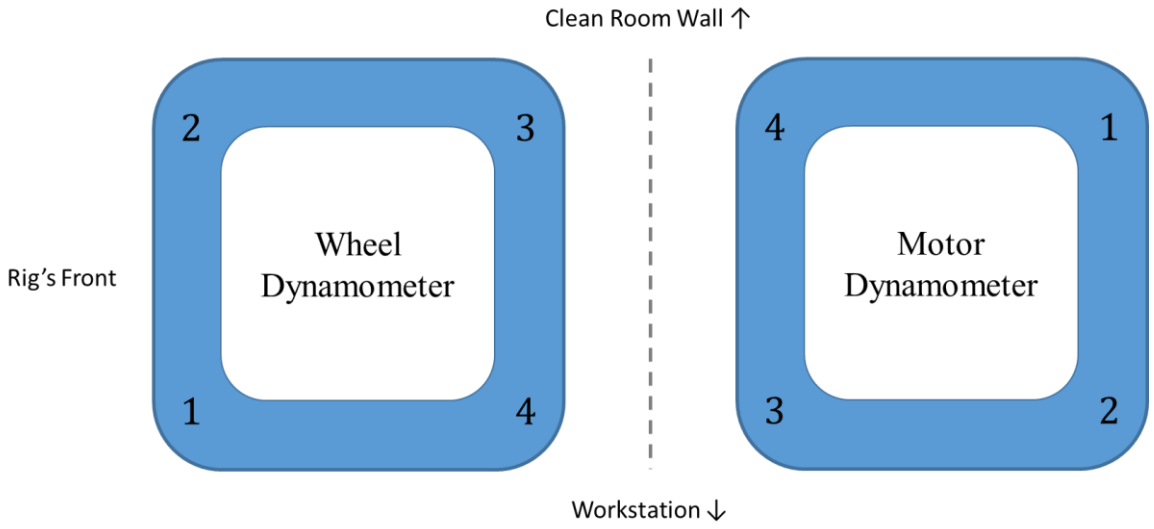


Figure 2-12: Arrangement and numbering of load cells in the Rig's contact force-moment measurement system

The sensor part number is required to identify the appropriate calibration certificate in order to obtain the proper load cell sensitivity value. The part numbers are described in Table 2-2.

Table 2-2: Part serial numbers for load cell calibration certificate identification

Motor Dynamometer		
Load Cell Number	Part Number	Kistler model type
1	4710660	9028C
2	4710665	9028C
3	4713449	9027C
4	4713445	9027C
wheel Dynamometer		
Load Cell Number	Part Number	Kistler model type
1	4710653	9028C
2	4710657	9028C
3	4663921	9027C
4	4663924	9027C

The data acquisition software and hardware access points are mentioned in Table 2-3. Note that although the moment channels are provided by the charge amplifier, they should not be blindly trusted to be valid because as mentioned before, the charge amplifier assumes that the contact is exactly in the middle of the dynamometer. To calculate contact moments, it is advised to post process the data with individual channel readings.

Table 2-3: Hardware and software access points for different force & moment channels

Motor Dynamometer		
Signal	SynqNet I/O	Hardware/Node
F_{x1+x2}	analogIn_0	MIXEDMODULE #1 P2 (node 10)
F_{x3+x4}	analogIn_1	MIXEDMODULE #1 P2 (node 10)

F_{y1+y4}	analogIn_2	MIXEDMODULE #1 P2 (node 10)
F_{y2+y3}	analogIn_3	MIXEDMODULE #1 P2 (node 10)
F_{z1}	analogIn_4	MIXEDMODULE #1 P2 (node 10)
F_{z2}	analogIn_5	MIXEDMODULE #1 P2 (node 10)
F_{z3}	analogIn_6	MIXEDMODULE #1 P2 (node 10)
F_{z4}	analogIn_7	MIXEDMODULE #1 P2 (node 10)
ΣF_x	analogIn_8	MIXEDMODULE #1 P2 (node 10)
ΣF_y	analogIn_9	MIXEDMODULE #1 P2 (node 10)
ΣF_z	analogIn_10	MIXEDMODULE #1 P2 (node 10)
ΣM_x	analogIn_11	MIXEDMODULE #1 P2 (node 10)
ΣM_y	analogIn_12	MIXEDMODULE #1 P2 (node 10)
ΣM_z	analogIn_13	MIXEDMODULE #1 P2 (node 10)
wheel Dynamometer		
Signal	SynqNet I/O	Hardware/Node
F_{x1+x2}	analogIn_0	MIXEDMODULE #1 P1 (node 11)
F_{x3+x4}	analogIn_1	MIXEDMODULE #1 P1 (node 11)
F_{y1+y4}	analogIn_2	MIXEDMODULE #1 P1 (node 11)
F_{y2+y3}	analogIn_3	MIXEDMODULE #1 P1 (node 11)
F_{z1}	analogIn_4	MIXEDMODULE #1 P1 (node 11)
F_{z2}	analogIn_5	MIXEDMODULE #1 P1 (node 11)
F_{z3}	analogIn_6	MIXEDMODULE #1 P1 (node 11)
F_{z4}	analogIn_7	MIXEDMODULE #1 P1 (node 11)
ΣF_x	analogIn_8	MIXEDMODULE #1 P1 (node 11)
ΣF_y	analogIn_9	MIXEDMODULE #1 P1 (node 11)
ΣF_z	analogIn_10	MIXEDMODULE #1 P1 (node 11)
ΣM_x	analogIn_11	MIXEDMODULE #1 P1 (node 11)
ΣM_y	analogIn_12	MIXEDMODULE #1 P1 (node 11)
ΣM_z	analogIn_13	MIXEDMODULE #1 P1 (node 11)

2.6 Sensor Drift Analysis

The sensor measurement drift is a critical issue that was addressed during the Rig's commissioning. The problem lay in the charge amplifier's "measurement mode" setting, which was improperly configured to "short" mode. As mentioned in Section 2.4.3, short mode corresponds to a low bypass resistance which increases the "leakage rate" of charge and also the high-pass cut-off frequency.

Figure 2-13 shows the drift characteristics when all load cells are configured to the "short" measurement mode. The time constant for a channel in short measurement mode is computed to be τ = 340s, i.e. it should take about 340s for the force value to drop to 36.8% of its actual value, which in this case is 2208 N. The curve-fit relationship for the short mode drift, as seen in Figure 2-13, is:

$$f_z = 5980e^{-0.003t}$$

Substituting t = 340s yields f_z = 2156.4 N. The observed decay is more than what is calculated from the short measurement mode resistance value. This is expected, as charge amplifier

measurement modes are dependent on time constants calculated from circuit elements. There is no guarantee that the dynamometer will represent the same rates of decay. In real life, the time constants are expected to be lower, i.e. the rate of charge leakage is expected to be higher due to less than ideal contact and insulation conditions.

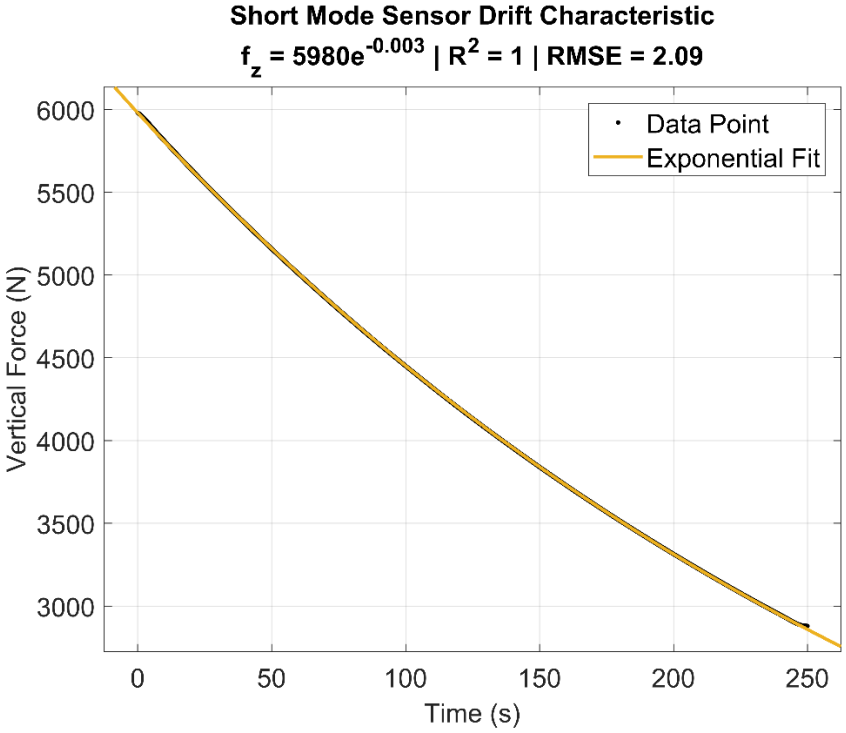


Figure 2-13: Short mode vertical force drift characteristic

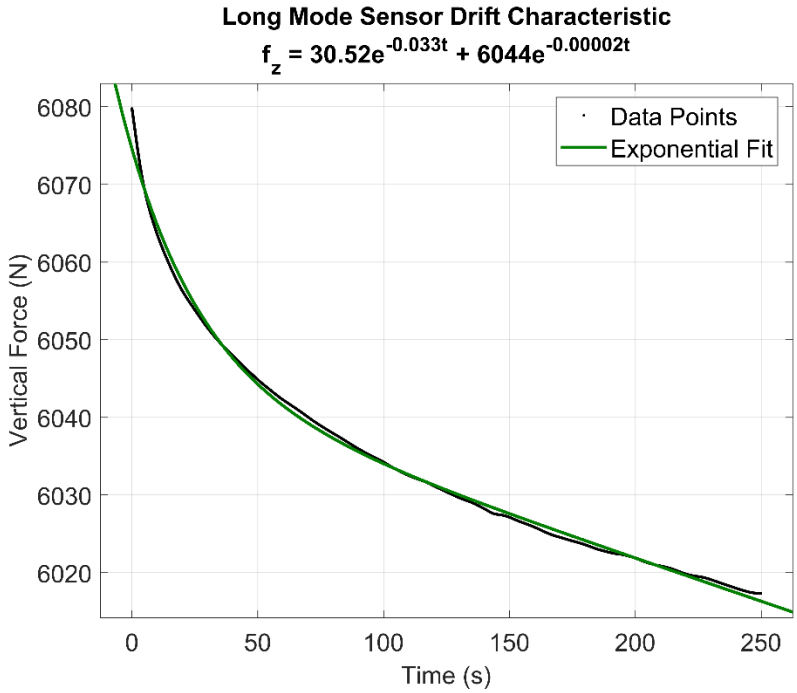


Figure 2-14: Long mode vertical force drift characteristic

It should be noted that the low difference in measured decay rate versus ideal decay rate indicates that the charge leakage from insulation resistance and contact resistance is low, thereby implying good health of the wiring. The reader is advised to perform such decay rate tests periodically to monitor insulation health and uncontrolled charge decay rates.

Figure 2-14 shows the sensor drift characteristic after the charge amplifier was configured for "long" measurement mode. The manual mentions the time constant to be more than 10000 seconds. The curve-fit expression for drift in long measurement mode is

$$f_z = 30.52e^{-0.033t} + 6044e^{-0.00002t}$$

The force at t = 0 is 6085N. The time taken for the reading to decay to 36.8% of its value (2516 N) will be 50000 seconds. This is suitably high, and hence long measurement mode is suitable for quasi-static measurements. It is recommended to reset the force measurement system every 500s since the force readings can drift by about 100 N by that time. This should be incorporated in the experimental design.

The DC (long) measurement mode decreases the cut-off frequency of the high pass filter, but does not affect high frequency measurements, as they remain in the "pass-band" region of the sensor response.

2.7 Summary

A systematic methodology for force data acquisition was developed by taking the physics-based fundamentals into consideration. The data acquisition system was fine-tuned, and a repeatable and reliable low-drift force data measurement was obtained. Additionally, this chapter documents the suggested configuration and data acquisition methodology, and provides an investigative data-based rationale for the same.

3. Vertical Axis Force Control Design

3.1 Introduction

Vertical force is an important boundary condition for wheel-rail contact mechanics. Hertzian contact theory states that the dimensions of the contact patch rely on the contact stress, which in turn depends on the load with which the two bodies are pressed together [6]. Hosseinipour [7] notes that unbalanced rotation of the wheel and roller can result in force measurements that would be picked up by the force measuring system. These disturbances would have the same frequency as the rotational frequency of the wheel, thereby making passive filtering of data impossible. He established that the Roller Rig setup would not have a chaotic vibration behavior, and thus the disturbance will be deterministic and can be removed in data post-processing. These conclusions were made for off-contact measurements.

During the commissioning exercises on the Rig, the wheel and the roller were pressed against each another and rotated. High vertical force fluctuations were seen with a peak-to-peak amplitude up to 100% of the mean force. The time waveform pattern appeared to be synchronous with the wheel and roller rotation. These fluctuations were deterministic to some degree, but could not be removed in data post-processing because vertical force fluctuations appeared to influence longitudinal and lateral force readings in almost perfect correlation. This implied that the vertical force fluctuations impacted the contact patch physics and a tighter control was desired in order to maintain as close to ideal testing conditions as possible.

This chapter covers the steps taken for reducing the force disturbance. Section 0 describes the vertical degree of freedom and how the vertical force is exerted. This is the "plant" under consideration for the control system. Section 3.2 elaborates the problem in greater detail and provides a motivating hypothesis which was pursued for solution design. A thorough data-based investigation was done for root cause analysis of the fluctuations, which was based on the hypothesis and is elaborated in Section 3.3. Section 3.4 describes the physics of contact and interaction with the environment, and reviews some established literature regarding force control in robotic manipulators. Sections 3.5 to 3.7 explain the design of passive compliance control, the physics-based reasoning behind the concept, the implemented design and the results that were obtained. The passive approach was less than satisfactory and the reasons for this were investigated and documented in Section 3.8. It was determined that active control methods are necessary to realize any reduction in the power of vertical force fluctuations.

The single loop PID compensation was implemented and is described in Section 3.9. The difficulties in single loop PID compensation led to the investigation of a deterministic feedforward-type control as the vertical force disturbance appeared to be synchronous with wheel and roller rotation. This feedforward control concept was first explored based on single wheel rotation and is elaborated in Section 3.10. The promising results of the feedforward-type control led to the conceptualization of the cascaded loop force/position feedback control where the feedforward part was replaced by a controller that predicted displacement based on force feedback error, detailed in Section 3.11, The results were promising and extensive system testing was carried out for robustness and performance analysis, as detailed in Section 3.12.

System Description

Figure 3-1 shows the actual view of the vertical degree of freedom in reference to the entire Rig [1]. One of the vertical actuators is shown in the image, with another actuator installed on the other side. These two actuators connect to the ground on one end, and the actuator pushrods are connected to the cradle. Cradle holds the wheel driveline and the Rail Cant assembly. The construction of the cradle is rigid and bulky as it sustains a high load. Cradle moves up and down by sliding on two linear motion guides which constrains any yaw rotational degree of freedom.

The load measuring platform is located below the blue wheel shaft bearings as highlighted in the image. The load measuring platform assembly is made by sandwiching four tri-axial load cells between two steel plates, one load cell on each corner. The forces on the wheel react on the load measuring platforms by the bolted connections of wheel bearings and wheel motor mount.

Figure 3-2 shows the actual measurement setup. When the linear actuators contract, the cradle slides down. Eventually the wheel comes in contact and presses against the roller. When that happens, the wheel shaft experiences a force vertically upwards, but that shaft is constrained by the bolted connection with the top plate of the dynamometer. This is how vertical force is measured by the dynamometer. A similar load path can be traced for longitudinal forces.

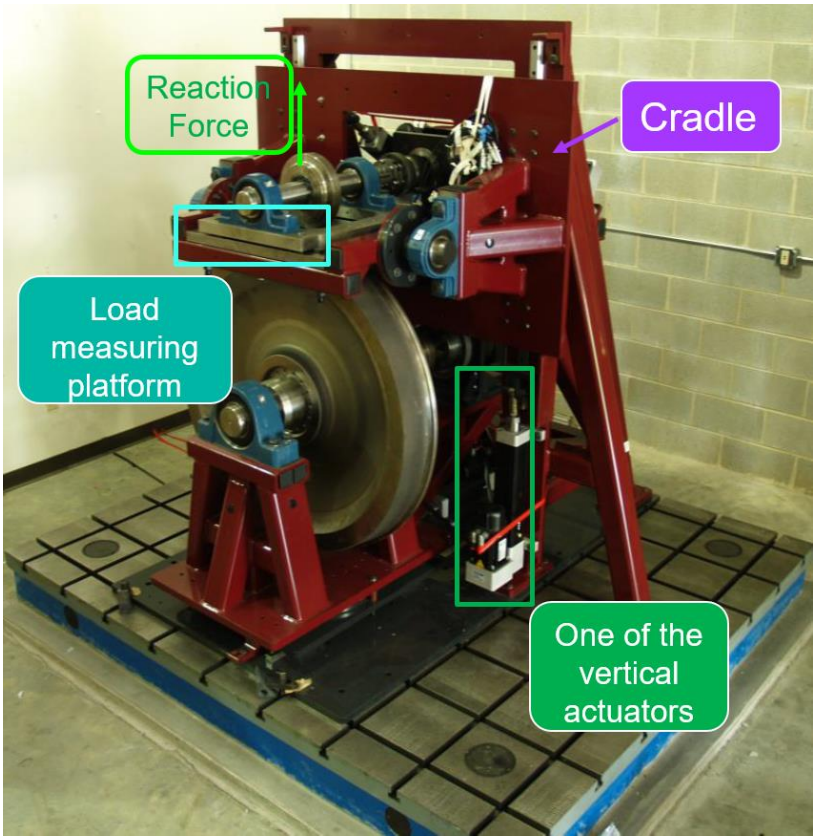


Figure 3-1 Vertical degree of freedom with respect to the VT-FRA Roller Rig

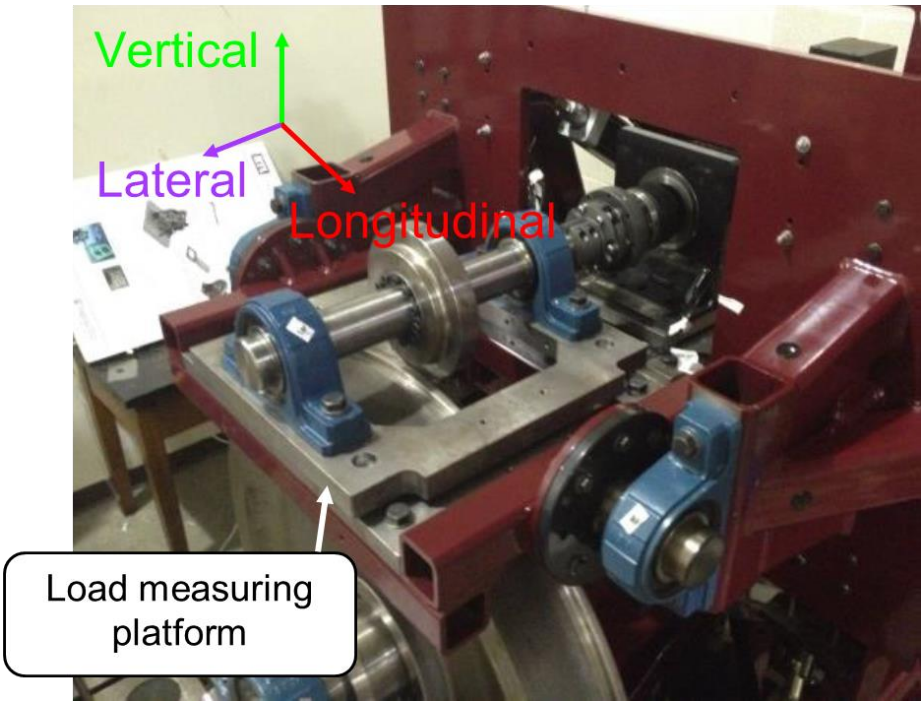


Figure 3-2: Measurement of contact forces on the Roller Rig

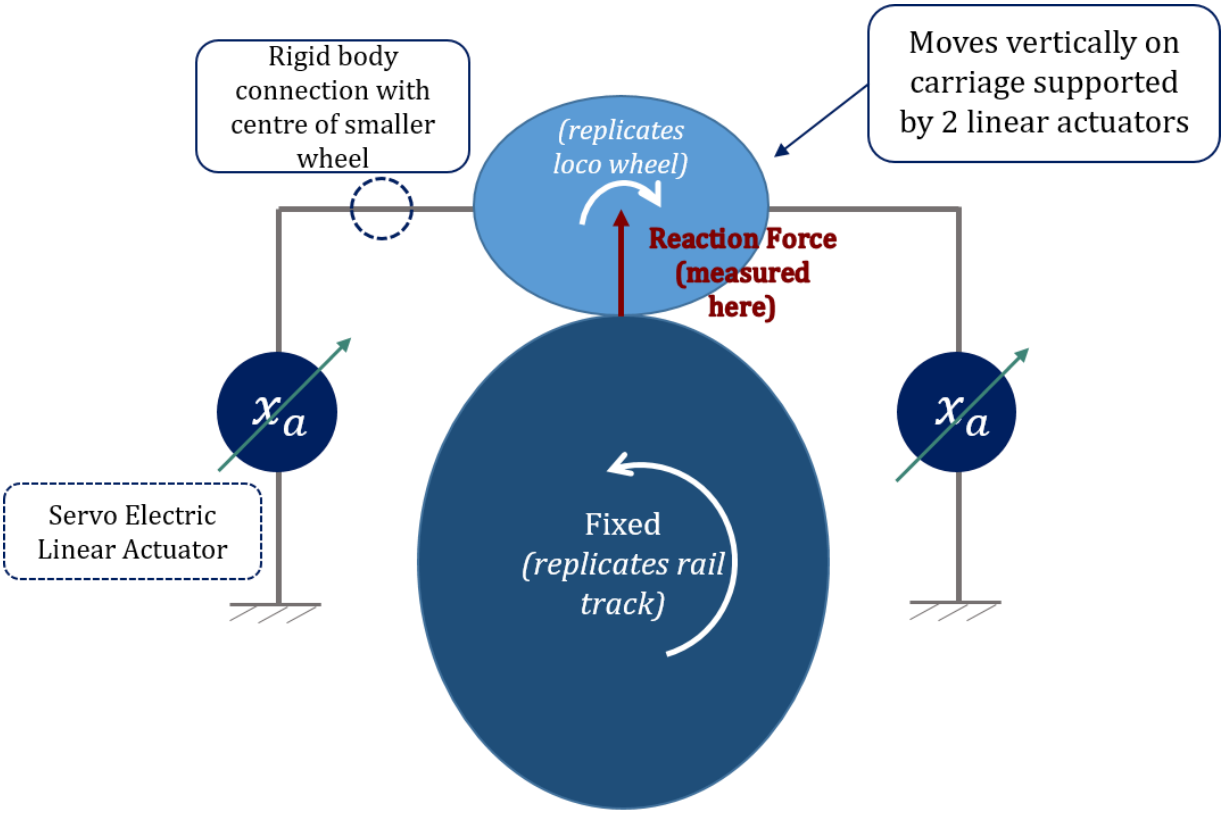


Figure 3-3: Simplified schematic of the vertical degree of freedom

Figure 3-3 shows a simplified schematic of the entire vertical degree of freedom. The linear actuators are shown in a notation consistent with the vehicle dynamics' textbooks. The cradle is constrained to move only in the vertical direction by two linear motion guides. No yaw angle is permissible while moving the vertical axis. Each servo axis has its own controller and actuator. Such a control scheme, where two servo actuators control the motion along a single direction and each servo is controlled by its own feedback controller, is called gantry control.

In case of the Rig, no yaw is permissible therefore both servo actuators are commanded the same physical command position [8]. It is also necessary that both closed loop systems have almost the same dynamics of motion, i.e. if one vertical actuator is faster than the other vertical actuator, then there is a change of mechanical locking interference or *crabbing* [7].

Figure 3-4 shows the construction of the linear actuators that actuate the cradle in vertical direction. A brushless AC servo motor connects at point 1 which provides the actuation power in the form of rotational motion. This power passes through a 5:1 belt-pulley reducer at 2 and rotates the ballscrew shaft. A recirculating ball nut slides on the ballscrew, thereby converting the rotational velocity to linear velocity. The ballnut is attached to the push rod which in turn retracts or expands depending on the direction of rotation of the ballscrew shaft.

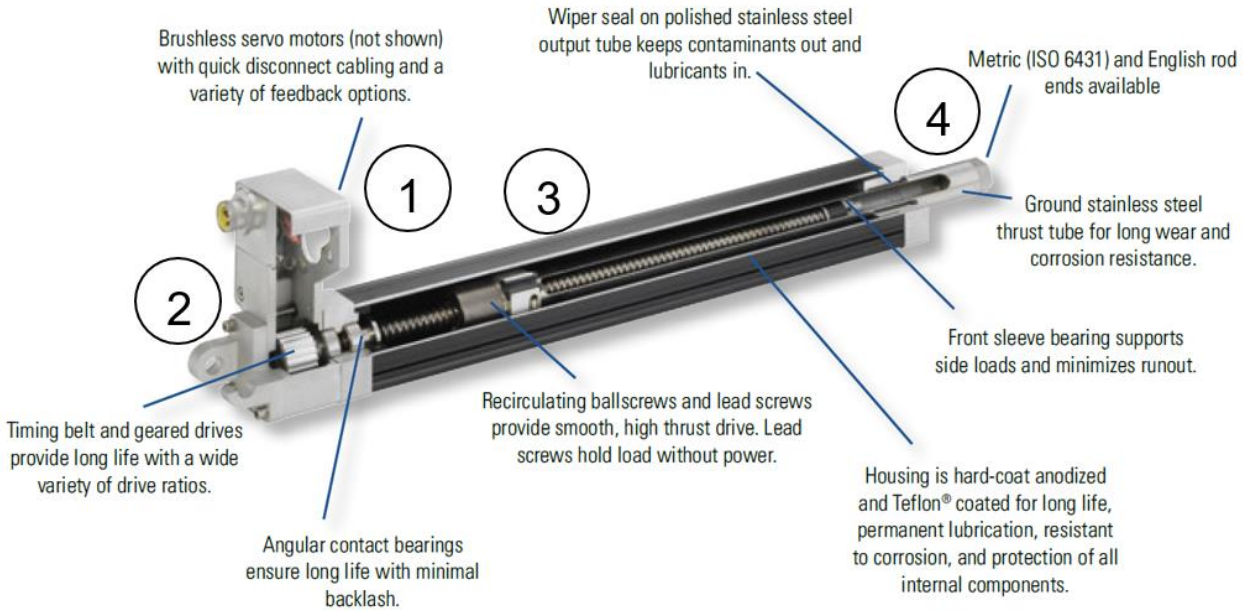


Figure 3-4: Cut-away diagram of the vertical linear actuators

From the discussion above, it is apparent that the by virtue of its assembly, entire vertical powertrain is mechanically stiff. Meymand notes in [1] that the construction was kept rigid so as to prevent low frequency resonant peaks that may interfere in data measurements. This stiffness has allowed for a high control bandwidth for positioning systems.

3.2 Vertical Degree of Freedom in Position Control

The Rig is constructed to be mechanically stiff. If any compliance is not explicitly added, then vertical actuators are the only source of any relative motion in the entire powertrain. Positioning accuracy parameters were calculated for the Rig and summarized in Table 3-1.

Table 3-1: Positioning accuracy of vertical actuators when in motion and at rest

Vertical positioning error type	Reported values
In air and in motion	0.00358 mm [7]
In air and at rest	8.567e-6 mm [7]
In contact at rest	1.2e-5 mm (experimental)
In contact with wheels moving	2.3e-5 mm (experimental)

The performance of the position control loop in the presence of disturbances such as the contact stiffness is better represented by looking at the closed loop frequency response. Higher bandwidth implies good disturbance rejection characteristics. The closed loop position control has a bandwidth of 120 Hz with a DC gain of 0 dB. The resonant peak gain is at 6.5 dB. These gains are kept the same as suggested by Hosseinipour [7].

The closed loop position control performance is expected to drop when the wheel and the roller are in contact. This is because the steel-steel contact has a high stiffness and will present itself as a disturbance for the position control loop. Figure 3-5 shows the comparative drop in gains when the wheel and the roller are in contact at 3000N versus when the wheel and the roller are not in contact. The drop in gains is expected to increase with increasing load due to the non-linear nature of the stiffness curve. Nevertheless, the closed loop position control still has a bandwidth of more than 100 Hz across all vertical load operating points.

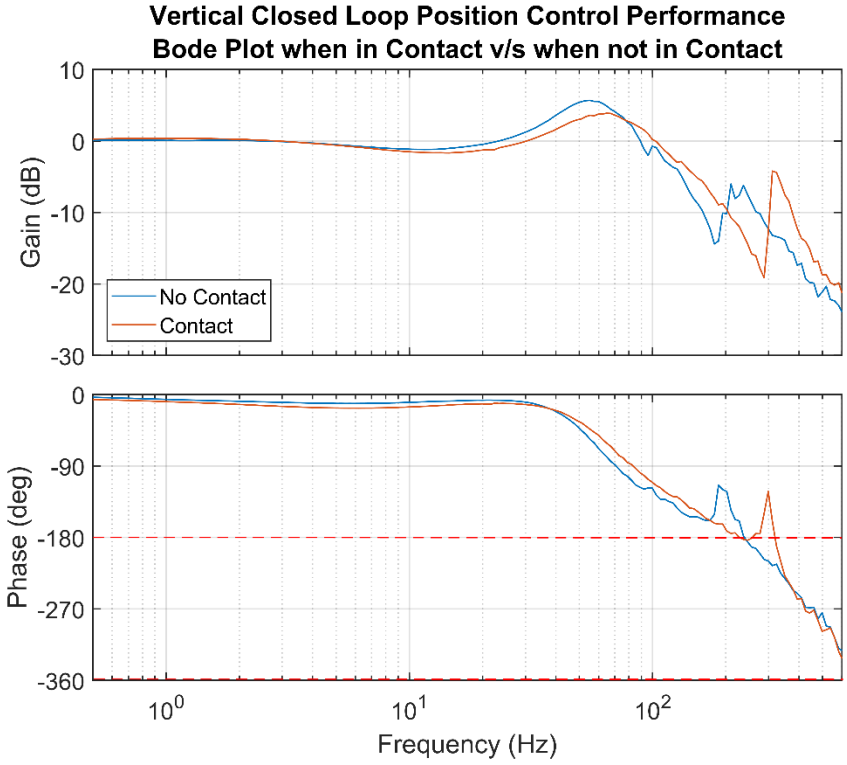


Figure 3-5: Position closed loop performance comparison - when not in contact versus when in contact at 3000N

Due to good tracking characteristics and a high bandwidth, the vertical actuators, when operated in closed loop positon control, are able to behave like a rigid link connecting the centers

of the wheel and the roller. This becomes a problem for the Rig, no surface can be machined to be perfectly circular. Machining methods and tolerances cause deviations from circularity. Physical examination of the wheel and roller surfaces show the presence of pits, notches, crests, and troughs. These deviations are aggravated by wear on the running surfaces of the wheel and roller, which is an inevitable consequence of conducting multiple creep-creepage experiments. When these deviations combine with a rigid position control, large vertical contact force fluctuations take place.

This can be understood by a simple model. Figure 3-6 shows the model where deviations in the wheel and roller are exaggerated to consider them as ellipses. They are pressed against one another and their centers are connected by a rigid link which constrains any vertical relative motion. This rigid link replicates the position-controlled linear actuators. When these imperfect circles move in this configuration, the wheel and roller experience time-varying compression at the contact patch. Since is a steel-on-steel contact, even small compressive strains will result in large reaction forces at the contact. Figure 3-7 shows the time series plot of vertical force fluctuations.

The peak-to-peak distance is roughly about 1000 N for a mean force of about 2800 N. Vertical force fluctuations of this magnitude are undesirable, as vertical force is an important contact parameter. Variations in vertical force also cause correlated variations in lateral and longitudinal traction forces. The size of the contact patch is also dependent on the contact pressure, which in turn depends on the contact force. In order to provide an accurate testing environment, it is necessary to reduce the vertical force fluctuations. The hypothesis presented in this section was tested via data analysis and is reported in the next section.

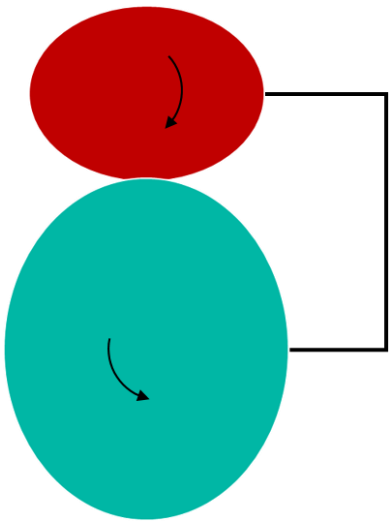


Figure 3-6: Position control scenario visualized with the exaggerated ovality of wheel and.

roller

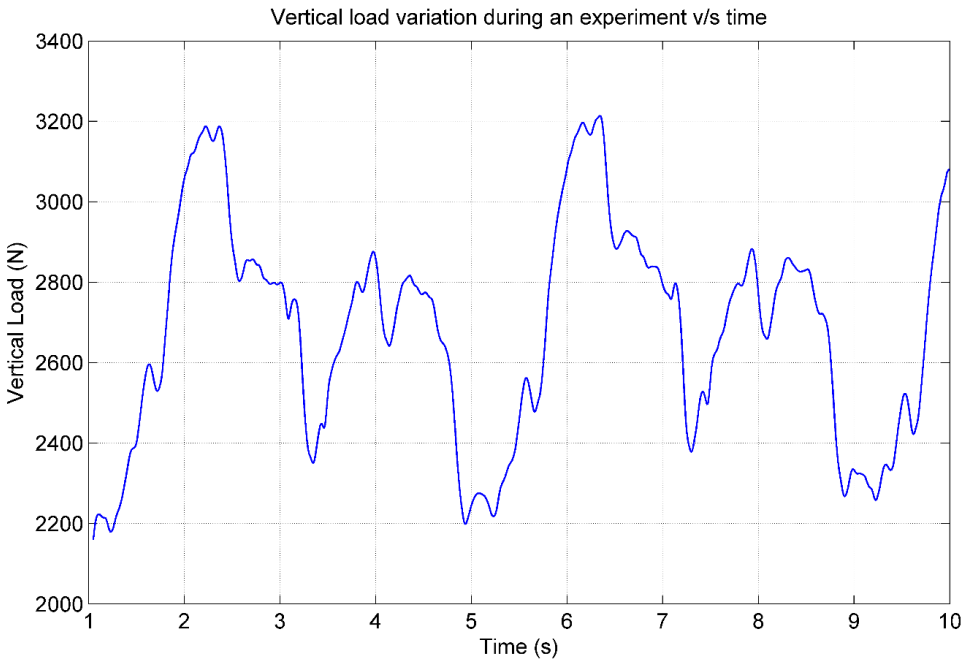


Figure 3-7: Vertical load variation due to geometric imperfections of wheel & roller when vertical degree of freedom is operated in position control

3.3 Causality Analysis of Vertical Force Fluctuation Data

A data-based investigation was conducted to verify the hypothesis that the vertical force fluctuations are a result of the minor geometric imperfections of the wheel and roller combined with the stiff position control. The vertical force time series data was collected, along with the wheel and roller encoder feedback time series. The encoder feedbacks were converted into rotation angles, and the time parameter was eliminated from the synchronous data by plotting the vertical force vector against the wheel and roller rotational angle vectors.

$$f_z = F(\theta_{wheel}, \theta_{roller})$$

Figure 3-8 shows the surface plot representative of the functional relationship shown above. The experiment was conducted by pressing the wheel and roller against each other in position control. The motion was initiated and data was collected for 250s, which corresponds to 500,000 samples at a sample rate of 2000 Hz. The wheel rotational frequency is 1.15 Hz, and the roller frequency is 0.25 Hz. At these rotational frequencies, both the wheel and the roller can return to the exact same starting configuration after 348s. The recording buffer capacity was limited to 500,000 samples of data, so only 250s of data was collected. This covered 62.5 full rotations of the roller and 287.5 full rotations of the wheel, covering a wide range of wheel and roller angle pairs. The surface is a simple interpolant fit generated from this data. Before generating the fit, the vertical force data was low pass filtered using the anti-causal filtfilt() function of MATLAB. This was done to ensure minimal phase distortion of the data. The filter used was a 2nd order Butterworth filter having a break frequency of 10 Hz.

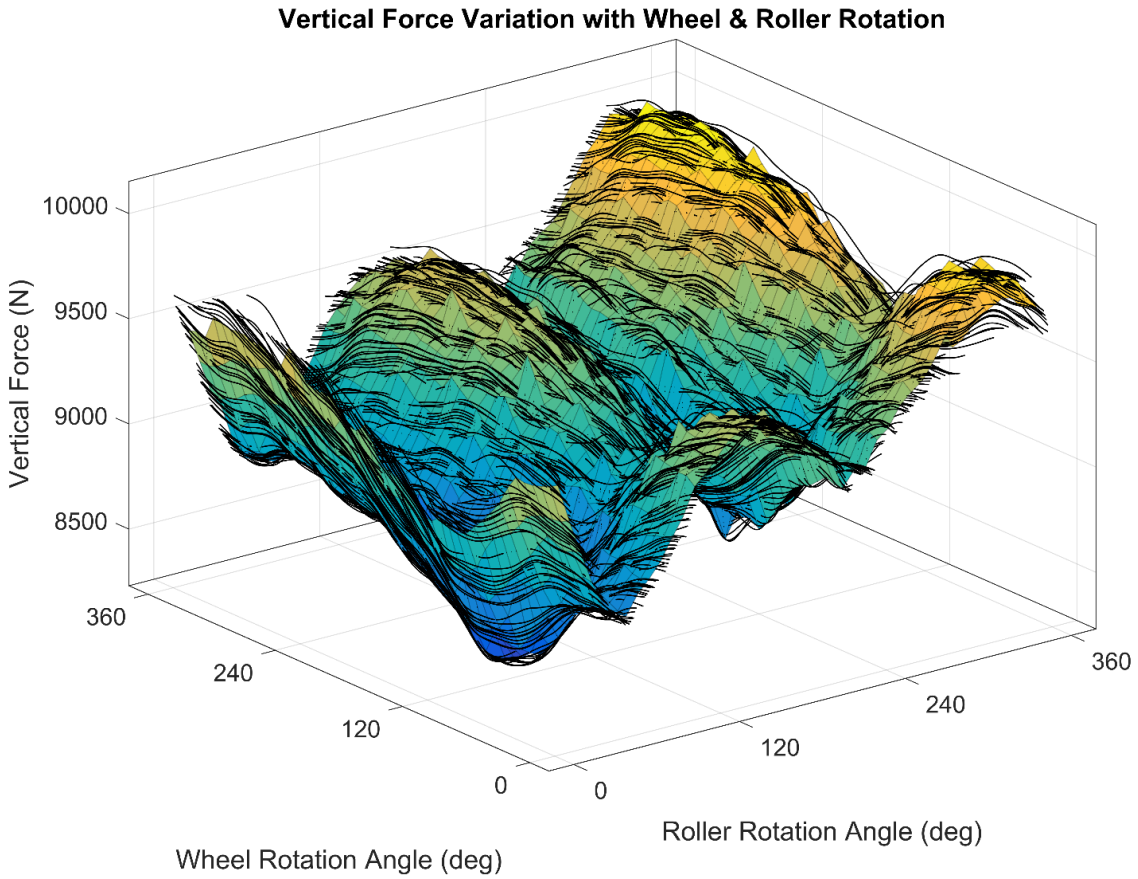


Figure 3-8 Surface plot showing dependence of vertical force with wheel and roller rotation angles

The actual data points are shown by the black dots. It can be seen that there is a pattern between the rotation angles of wheel and roller and the vertical force value. Looking at the projection plot of this surface plot should give an indication of the strength of the relationship between the two variables plotted. Figure 3-9 shows the projection plot of the surface along the vertical force - roller rotation plane. Figure 3-10 shows the projection plot of the surface along the vertical force – wheel rotation plane. Looking at these projections, it can be inferred that the vertical force depends on both of the variables, the wheel and the roller. The strength of relationship can be inferred by looking at the heat map of the data. Figure 3-11 shows the top projection of the surface plot shown in Figure 3-8. The heat bar on the right shows the value of vertical force attained. The strength of relationship can be seen by color changes along the X or Y axes. It can be seen that there is a sharper gradient when moving along the roller rotation when compared to wheel rotation. This indicates that vertical force varies more strongly with the roller surface profile than with the wheel surface profile.

This strength of relationship can be utilized for force control system design like feedforward control, which would be based on angular rotation value of the wheel and the roller. One such feedforward control method based on the wheel rotation angle has been implemented with promising results; details are covered in Section 3.10.

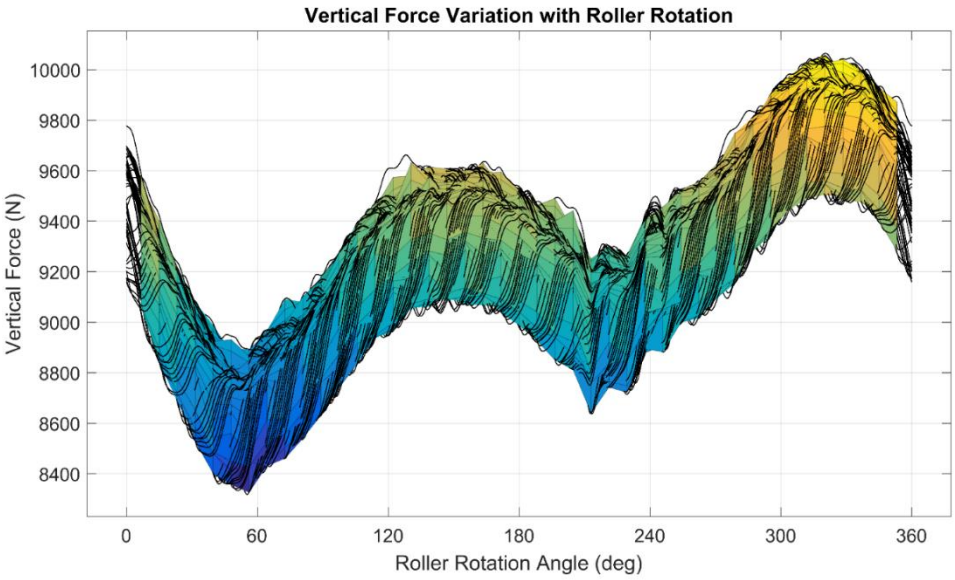


Figure 3-9: Projection plot showing dependence of vertical force with roller rotation

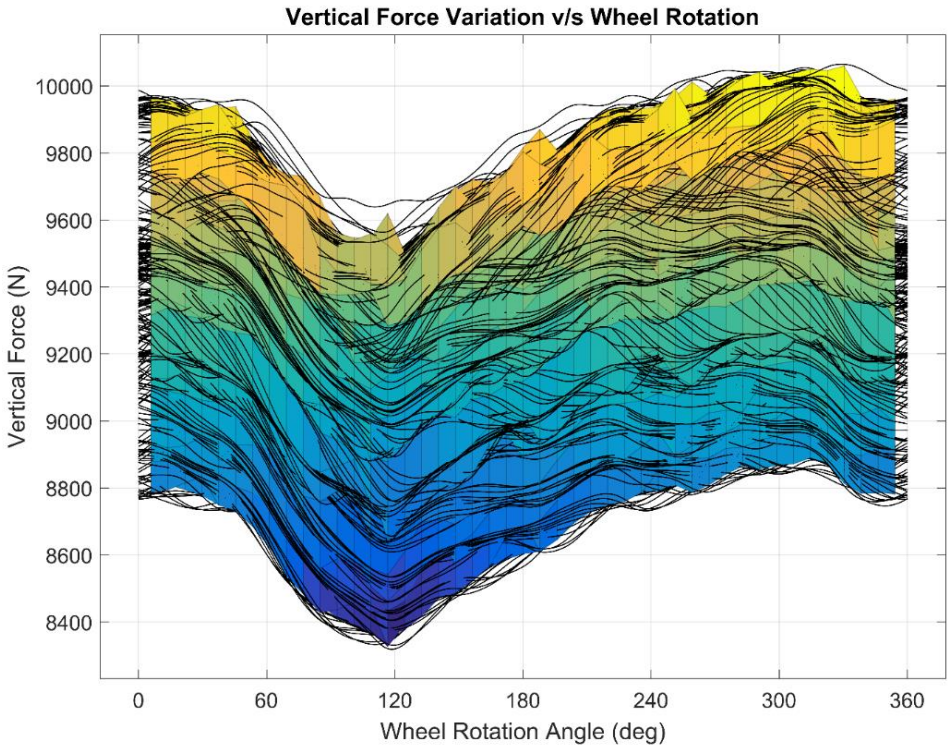


Figure 3-10: Projection plot showing dependence of vertical force with wheel rotation

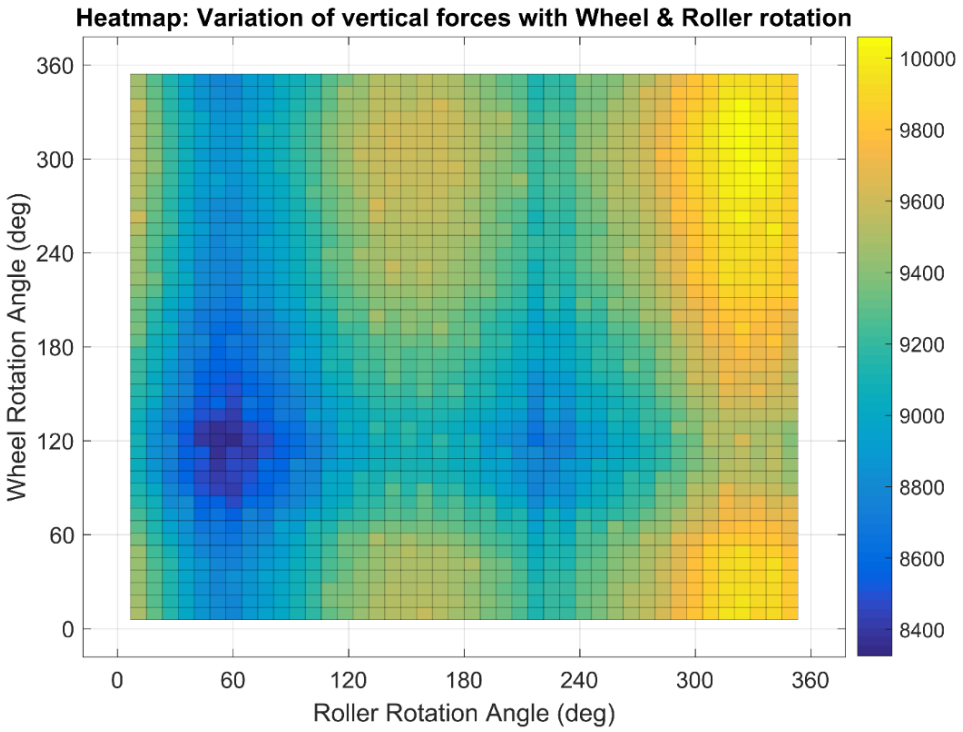


Figure 3-11: Heatmap of vertical force variation with wheel rotation and roller rotation

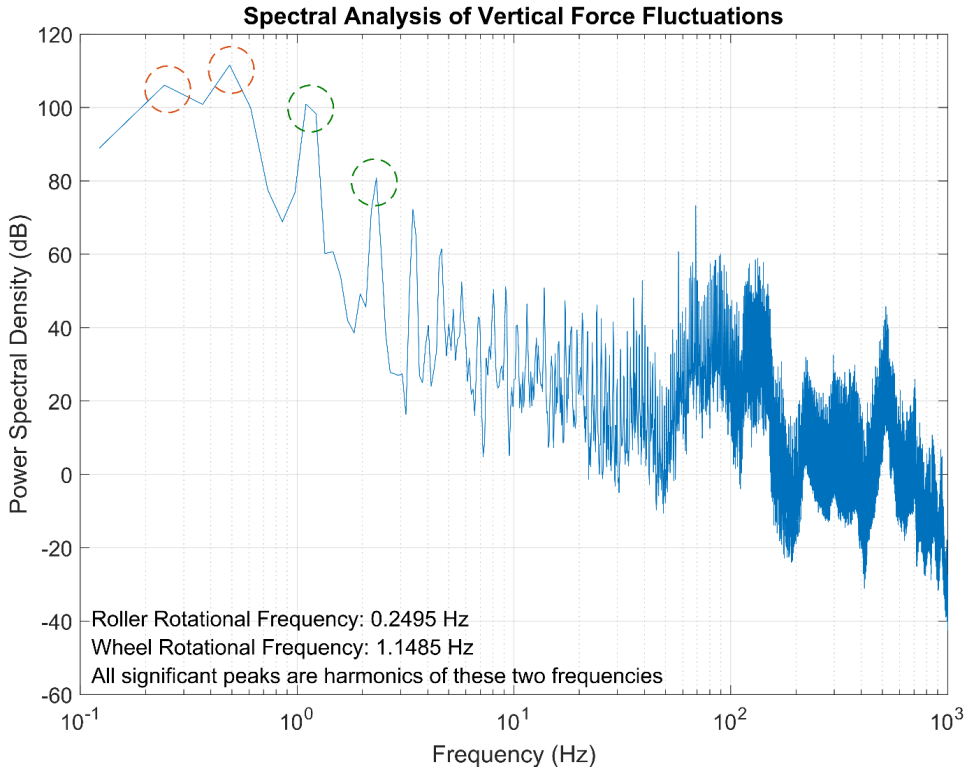


Figure 3-12: Welch's periodogram of the vertical force time series data shown earlier

The spatial correlation of vertical force data was supplemented by spectral analysis of the vertical force data. If the vertical force oscillations are caused by wheel and roller rotations, then

the vertical force data must be composed of forcing frequencies which correspond to the rolling frequencies of wheel and roller, and spectral analysis should show power peaks at those frequencies. Figure 3-12 shows the FFT of the vertical force fluctuation time series done by the Welch's periodogram method. This is a 16384-point FFT done with a Hanning window having an overlap of 50%. Sampling frequency is 2000 Hz. The vertical force data is unfiltered but was de-trended by the mean of time series so that the power spectrum is not dominated by the DC (mean) value.

The largest peaks in the spectrum correspond to the frequency of roller, frequency of wheel, and their harmonics. The peaks would be due to the geometries of the wheel and roller, which act as force excitation sources due to varying compression cycles. Figure 3-12 clearly shows the presence of fundamental frequencies and harmonics of the wheel and roller. The orange circles show roller frequency and it is first harmonic, while the green circles show wheel frequency and its first harmonic. All other peaks are lower by more than 20 dB and hence are insignificant for consideration. Such a plot by itself is extensively used in industries for vibration fault diagnosis in rotating machinery. The vertical force fluctuation data is very similar to an accelerometer reading for an unbalanced rotating machinery so the same principles can be utilized for analysis.

3.4 Force Control Theory

A control scheme purely based on position control is inadequate for the Rig, as it results in high contact force fluctuations that are undesirable. To avoid this, it was necessary to implement some sort of force control scheme for the vertical degree of freedom. Force control is a problem that has been extensively discussed in robotics. A manipulator arm which is actuated by servos operating purely in motion control is great for following a motion trajectory. However, for manipulation or interaction tasks with the environment, motion control would result in large contact forces which could either break through the environment or drive the actuators into saturation, thereby creating non-linear effects which could result in instability. Another application for force control is in applications that are highly dependent on applied force, such as the robots that cut, machine or polish objects. To realize the automatic high quality operation for such robots, the requirement is to not only maintain safety and stability of the system, but also to maintain a particular force set-point in the presence of disturbances. Both of these objectives gave rise to different methods of force control [9].

3.4.1 Contact and Interaction

Understanding and control of the interaction between two contacting surfaces is key to successful execution of manipulation or exertion tasks. No physical object is rigid or has infinite stiffness. To model the contact between two objects with high stiffness, a spring is inserted between two ideal rigid bodies which represents the compliance and damping of the contact (Figure 3-13). Compliance of a body is the measure of flexibility of a body under applied load. Stiffness is the inverse of compliance which is a measure of resistance of a body to any deformation/flexing. Compliance and stiffness are used interchangeably in literature without any confusion.

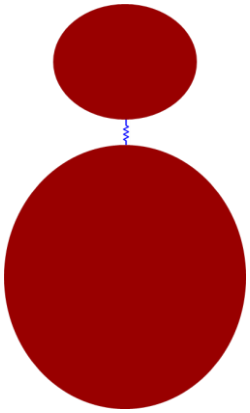


Figure 3-13: Schematic of the contact in the vertical direction between wheel and roller

From a system's modeling perspective, admittance and impedance are better terms with which to work. In the mechanical domain, admittance is a system that generates velocity (flow variable) when acted upon by a force (effort variable). Impedance is a system that generates force (effort variable) when acted upon by velocity (flow variable) (Figure 3-14).

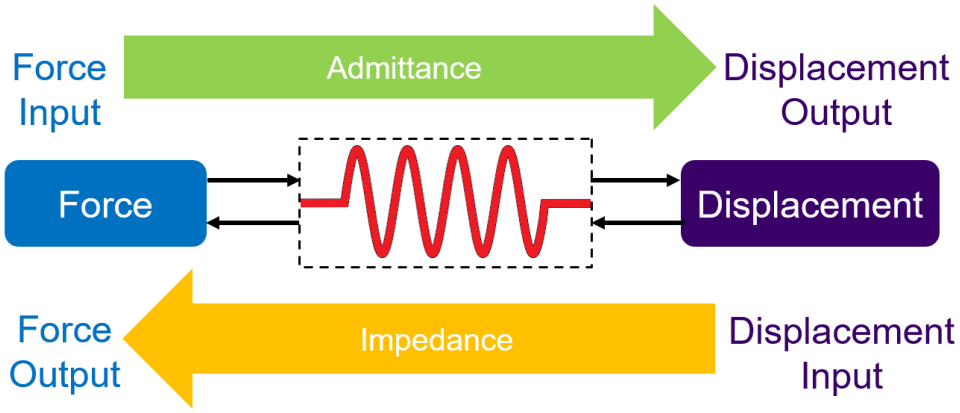


Figure 3-14: Compliance: Admittance and Impedance

During a contact, the environment sets kinematic constraints on the geometric paths that the end-effector can take. Such constraints can be classified as admittance or impedance depending on the inputs and outputs. The environment is an impedance and the end-effector is an admittance when the end-effector imposes motion on the environment. When the end-effector imposes force on the environment, then the end-effector becomes the impedance and the environment becomes the admittance [10]. A control or modification of this interaction results in a change of force/motion at the contact. Interaction control can be classified into two categories: passive interaction control and active interaction control [9]. Passive interaction control uses mechanical compliant elements to increase the inherent compliance of the system so as to limit or reduce the contact forces. Active interaction control uses a specially-designed control system to change the compliance of the system. Active interaction control again can be classified into two categories: Active direct force control uses sensors to monitor forces and moments and uses them to close a force feedback loop. Active indirect force control doesn’t explicitly carry out force measurements but it uses control loops to electronically change the compliance of the system. Indirect force control is normally used to prevent actuator saturation and damage prevention, whereas direct force control is used for force setpoint regulation and disturbance rejection [9]. It should be noted

that all the methods mentioned above change the compliance of the system. Such methods are often reported in the literature by names such as interaction control [10] or compliance control [11].

3.5 Passive Compliance Approach

It was established in Section 3.2 and Section 3.3 that the vertical force fluctuations were due to high rigidity of the system interacting with the inevitable minute geometric defects in wheel and roller. The first approach to reduce the force oscillations was to increase the compliance of the system by adding mechanical compliance elements in series along the load path. Figure 3-15 shows the schematic of the passive compliance approach.

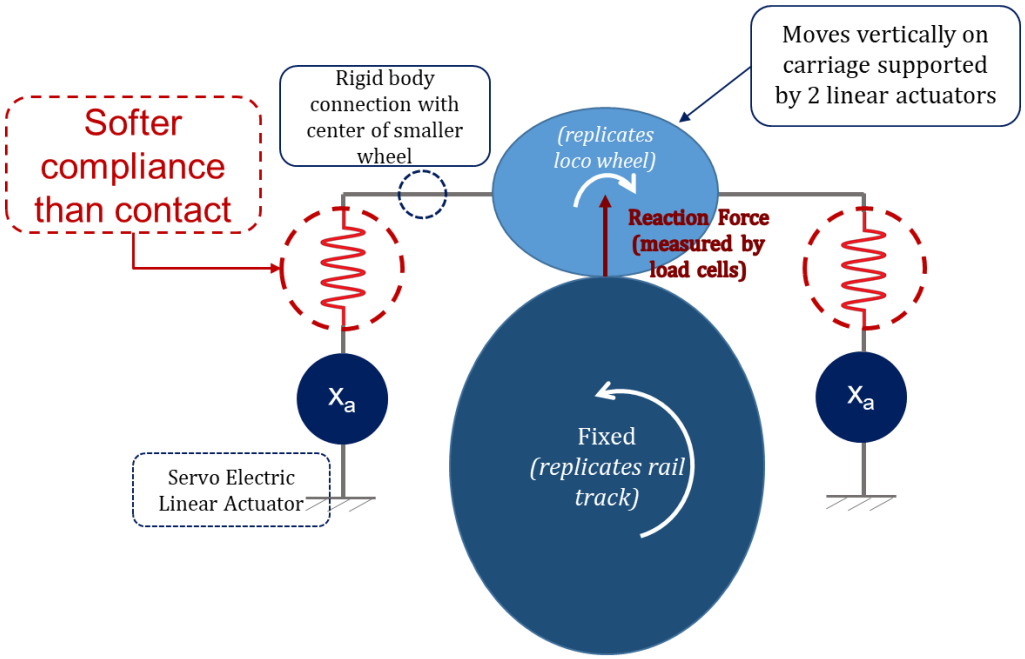


Figure 3-15: Concept schematic of adding compliance to the system

Load path before	Ground→ Actuators→ Cradle→ wheel→ Load Platform
Load path after	Ground→ Actuators→ Compliance→ Cradle→ wheel→ Load Platform

If the contact stiffness is represented as $k_{contact}$ and the stiffness of the external spring as $k_{external}$, then the following relationship holds if the two are in series:

$$\frac{1}{k_{equivalent}} = \frac{1}{k_{contact}} + \frac{1}{k_{external}}$$

If the external spring is much softer than the contact stiffness, then:

$$\frac{1}{k_{contact}} \ll \frac{1}{k_{external}}$$

$$\therefore \frac{1}{k_{equivalent}} \approx \frac{1}{k_{external}}$$

$$or\ k_{equivalent} \approx k_{external}$$

The rigidity along the vertical powertrain can be manipulated by adding an external compliance. There are two ways to add a compliance to the system – passive and active. Passive compliance refers to mechanical springs with some stiffness and damping. Active compliance refers to the idea of using a controller that produces displacement as an output when force is supplied to the controller.

3.6 Passive Compliance Design

The passive method involves adding physical compliance elements like springs and dampers in the load path to reduce the stiffness. Conceptually, the problem can be described by Figure 3-16, where a rigid circle is having a one-point contact with a wavy surface. The circle remains fixed in space and the wavy surface traverses horizontally, being in contact with the circle at all times. The circle is constrained to move in only one direction: vertically upwards. By adding an external compliance, it is as if a spring is added between the wheel and ground. It was hypothesized that for a softer spring, the vertical motion would be less constrained, which would cause lower vertical force fluctuations.

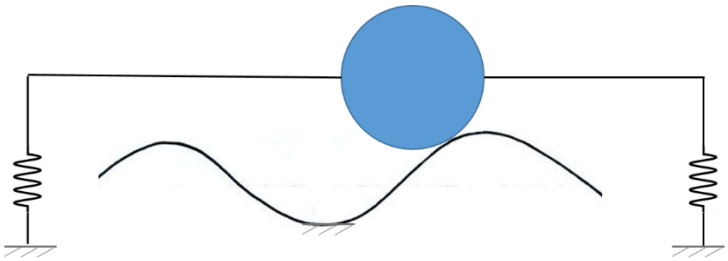


Figure 3-16: Rigid circle traversing on a rigid wavy surface

Figure 3-17 shows the compliance that was added on the Rig. The compliant element here is the black rubber bumper which flexes in shear. There are two threaded bolts on each side of the rubber bumper, but internally they are not connected. As a result, the bumper will flex when the bodies on the left and right move vertically in opposite directions. Figure 3-18 shows the schematic of the entire vertical assembly with the compliances added. Note that the cradle is suspended via the rubber compliant elements. There is no direct connection of the cradle with the actuator pushrod. Hence, the load path now involves a spring as intended.

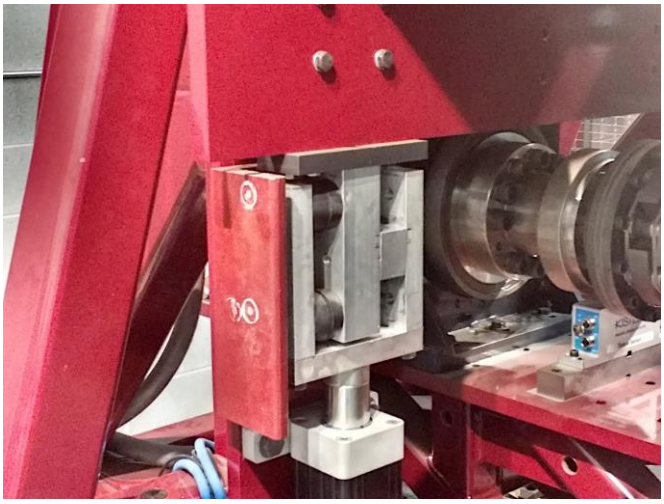


Figure 3-17: Passive rubber shear compliance on the VT-FRA Roller Rig

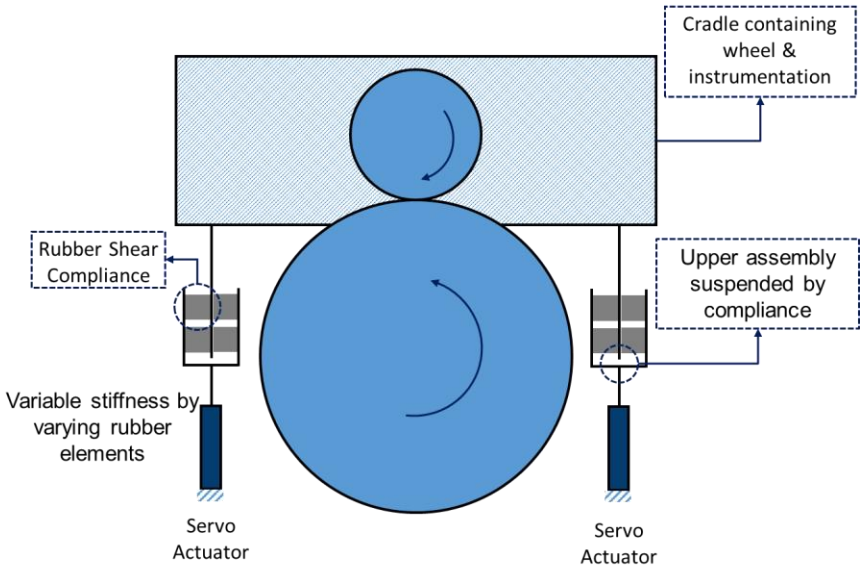


Figure 3-18: Schematic of the vertical assembly with passive compliance added

The compliance was designed so that the stiffness can be adjusted by adding or removing rubber compliant elements. The softest possible arrangement that could withstand static load of the cradle was accomplished by two rubber bumpers per compliant element. After the compliance was installed, the wheels were brought into contact and rotated. Observations are as reported in Figure 3-19. Under the same mean vertical load, there was only an 8.3% reduction in 99%ile data spread. 99%ile data spread is the 6σ spread about the mean, i.e. +/- 3σ limits. This was not an acceptable solution, and a better method was desired.

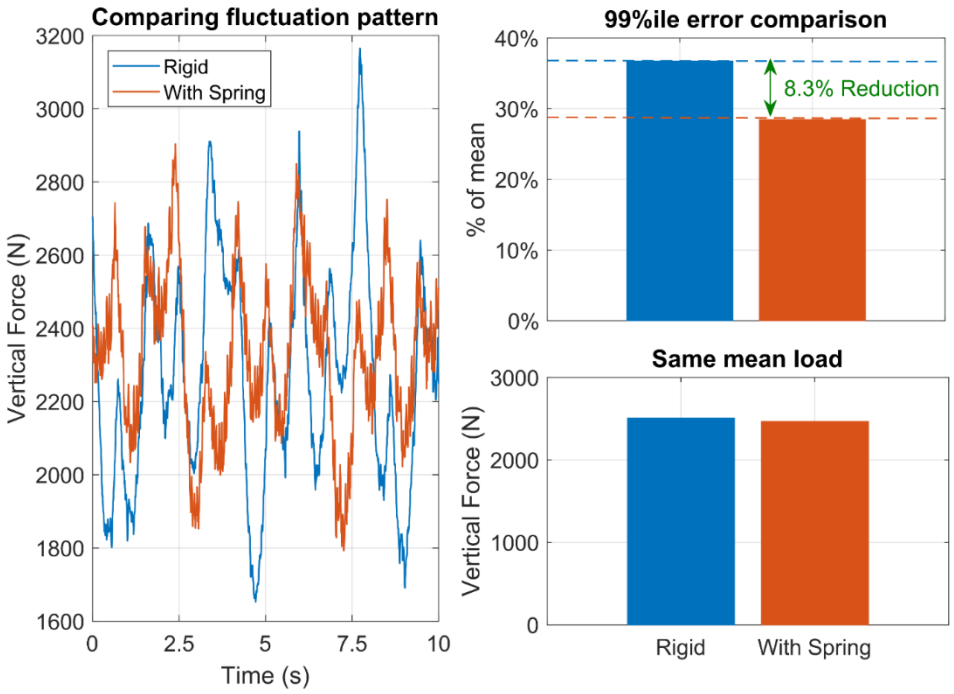


Figure 3-19: Vertical force fluctuation with passive compliance installed

3.7 Passive Compliance Effects on Position Control

Physically this becomes an inverse to the vibration isolation problem where the objective is to reduce the oscillations of the work-piece. Here the objective is to maximize the oscillations of the work-piece (cradle) in order to prevent a compression of the springs. Figure 3-20 shows a conceptual equivalent of our problem where the harmonic excitation of the base represents the combined sinusoidal profile of the wheel and roller and the mass represents the cradle. The spring-damper combination would represent the external compliance that should be added.

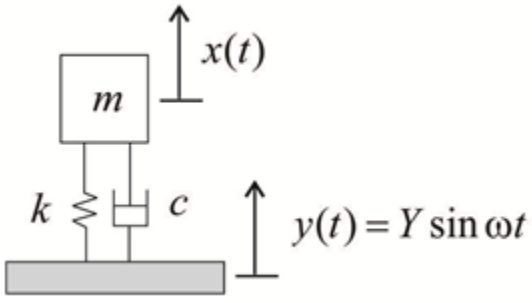


Figure 3-20: A typical base excitation problem

Assuming that the cradle moves without friction on the linear motion guides, we can write the dynamic equation as:

$$m\ddot{x} = F_{net}$$

$$m\ddot{x} = -k(x - y) - c(\dot{x} - \dot{y})$$

$$or\ m\ddot{x} + k(x - y) + c(\dot{x} - \dot{y}) = 0$$

$$Taking\ the\ laplace\ transform\ and\ collecting\ terms\ with\ X\ \&\ Y\ as\ coefficients$$

$$(ms^2 + k + cs)X = (k + cs)Y$$
$$\frac{X}{Y} = \frac{k + cs}{ms^2 + k + cs}$$

Transmissibility is defined as the ratio of signal amplitude transmitted to mass *m* to the base excitation, in the frequency domain. In this case, transmissibility becomes the magnitude of the above transfer function. However, the above transfer function also has a phase component associated with it. A phase lag between output and input signals in this case means a time lag between base excitation (surface profile) and the corresponding movement of the cradle. For a net zero compression of contact spring, the transmissibility should be one and the phase lag should be zero at forcing frequencies of interest. To achieve this with a massive cradle (m ≈ 1000 kg), k= $5.0x10^5$ N/m and c = 0.01 N-s/m. To put this in perspective, the estimated stiffness of the contact patch is $k_{contact} = 1.15e^6$. We need to add an external spring which is roughly equivalent to the contact patch stiffness. (Figure 3-21).This becomes equivalent to the rigid contact between the actuators and the cradle that existed previously.

So, if any active control method is used where servo actuators are used to compensate for force disturbances, the stiffness along the vertical drivetrain should be kept as high as possible so as to keep the transmissibility of compensation signal good.

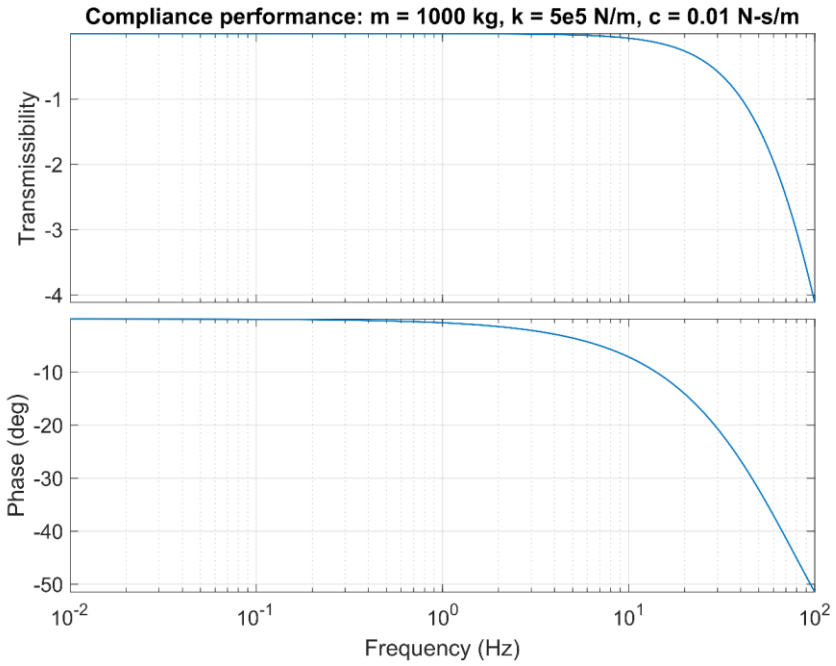


Figure 3-21: External compliance design using the base excitation transfer function

3.8 Passive Compliance Analysis and Conclusions

Passive compliance addition performed poorly than expected. The probable causes could be:

1. High friction along the linear motion guides
2. Large inertia of the cradle
3. Large gravitational force acting on the cradle due to the huge mass
4. Non-linearity in the spring

Meymand [1] notes that the guides used along the vertical axis are caged ball linear motion guides of model SHS provided by THK. Figure 3-22 shows the schematic of the drive. This motion system is similar to the ballscrew-ballnut mechanism found in our linear actuators and is considered the lowest friction mechanical motion option.

This brings us to another possible cause which is the large mass of the cradle. The cradle has an estimated mass of about 800 kg which moves in the vertical direction. So, the net force that can at least balance the weight of the cradle is 8000 N. The forces that would be balancing the cradle would come from flexing of the spring and the wheel-rail contact. Any contact load gets registered by the load measuring platform. If the spring (or any external agent) supports or exerts the load on the cradle, the vertical force decreases or increases respectively.

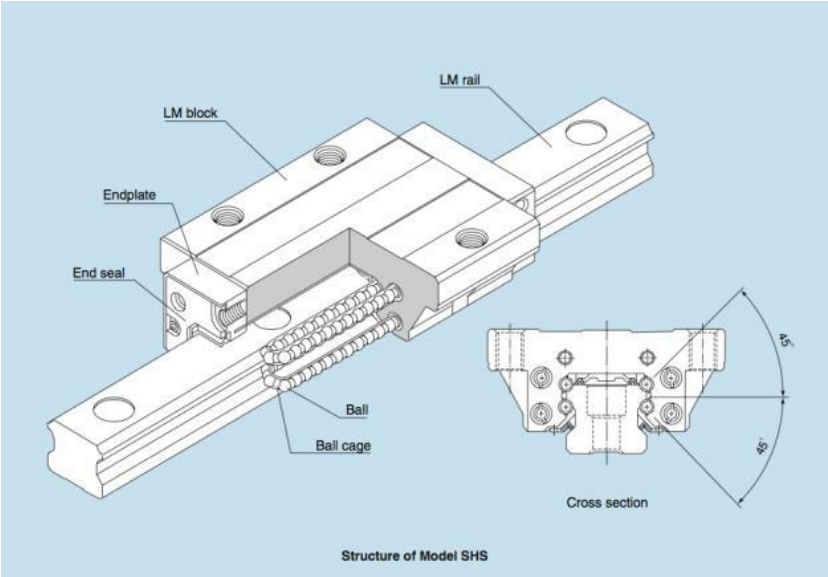


Figure 3-22: Schematic drawing of the caged ball LM guides [1]

This revealed a different aspect to the problem: Consider a rigid circular body free-rolling over a rigid wavy surface, as shown in Figure 3-23. If both bodies are perfectly rigid, then the center of mass of the circle has a wavy trajectory similar to the surface profile.

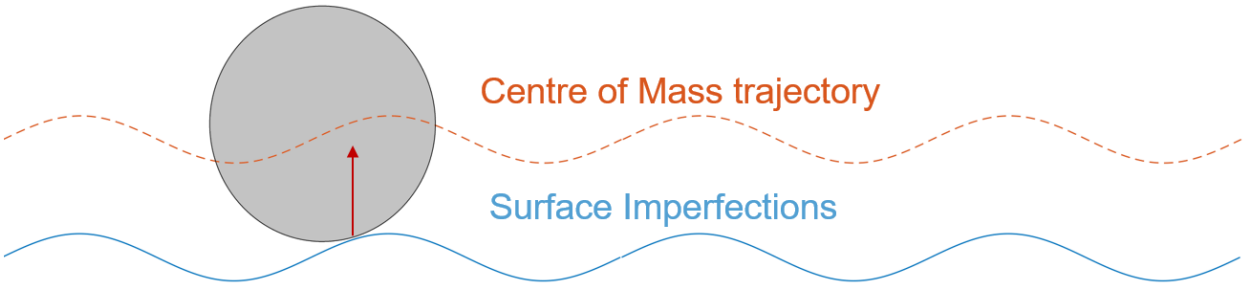


Figure 3-23 Free rolling of a rigid circle over rigid wavy surface

A wavy trajectory can be represented by a function like $y = f(x)$ where $\dot{y} \neq constant \implies \ddot{y} \neq 0$. For acceleration of the center of mass, there should be a force acting on the body. For a free-rolling body, this force comes from the contact. Therefore, the contact forces will have a pattern that is similar to the surface profile.

If this wavy vertical force profile is supplied externally, i.e. by the actuators, then the rigid body contact should have a constant reaction force. If this analogy is extended to non-rigid practical bodies, then if a wavy force profile is supplied to our wheel, a constant compression can be maintained at the contact which will result in a flat vertical reaction force. So even if the guideways are frictionless and the stiffness is driven to zero, the force fluctuations would exist. So passive methods cannot be used to regulate force. An active control algorithm would be required to eliminate or reduce the force oscillations.

3.9 Single Loop PID Force Control Analysis

PID control is the first approach to tracking and disturbance rejection problems. Figure 3-24 shows the loop that was set up for vertical force control. The force feedback would be compared against the reference force command to generate force error. This force error would be the input to the PID controller, which will supply motor torque commands to the servo actuators. Torque from motors would get converted into linear vertical force compensation by means of the ballscrew drive mechanism. A motor regulates torque by changing the current to the armature. Therefore, when a motor is operated in torque demand mode, the controller output is input to the motor's current loop.

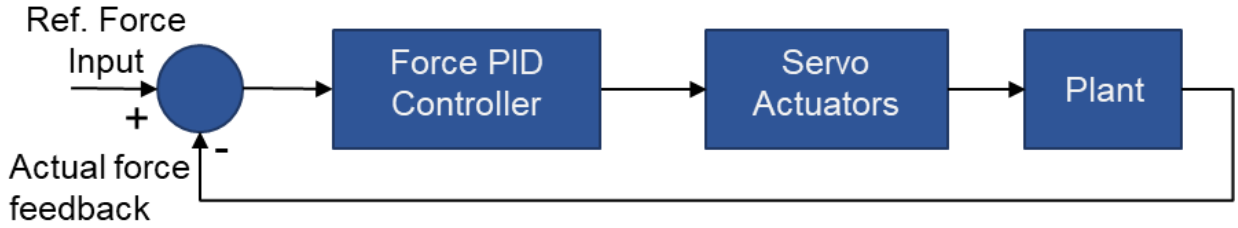


Figure 3-24: Single loop PID control loop for active vertical force control

Before tuning the controller, it is necessary to measure the plant dynamics and, if possible, estimate them with an analytical transfer function. This process helps expedite and also make the tuning process safer by first simulating the control system virtually, and only upon satisfaction, implementing the controller on the actual hardware. There are several methods to estimate a plant model and they all come under the domain of estimation theory. To measure the plant dynamics, the frequency domain approach is chosen. Pseudo Random Binary Noise (PRBN) or sinusoidal chirp noise is injected into the system and measurements are made at any two points in the system. One of these measurements becomes the input and the other becomes the output.

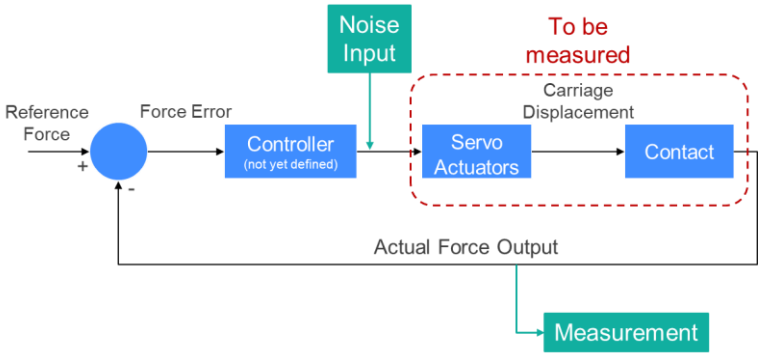


Figure 3-25: Measuring the plant dynamics for single loop vertical force control

It was advised by Kollmorgen engineers that the current loop inside the actuators should not be modified as that leads to a variety of issues. There is usually no need for resetting those loops as they are factory tuned to a very high bandwidth. Therefore, the current loops were included inside the plant to be measured by supplying the disturbance noise as the motor torque demand. Figure 3-25 shows the inputs and outputs for the system measurement process.

The sine-sweep plant measurement looks like Figure 3-26. The plant input is the same as the noise excitation, which is a sinusoidal torque command to servo actuators with a constantly changing frequency from 0.5 Hz to 700 Hz. The output is the vertical force measurement. Initially, both wheels were in contact. Then the actuators were enabled. A constant torque command of 1400N in the vertically upwards direction (hence the negative sign) was supplied to ensure that the actuators are pushing upwards and the carriage does not fall down. The vertical force reading was allowed to stabilize and then the excitation was turned on from the Bode Tool. MechaWare™ implementation is as shown in Figure 3-27.

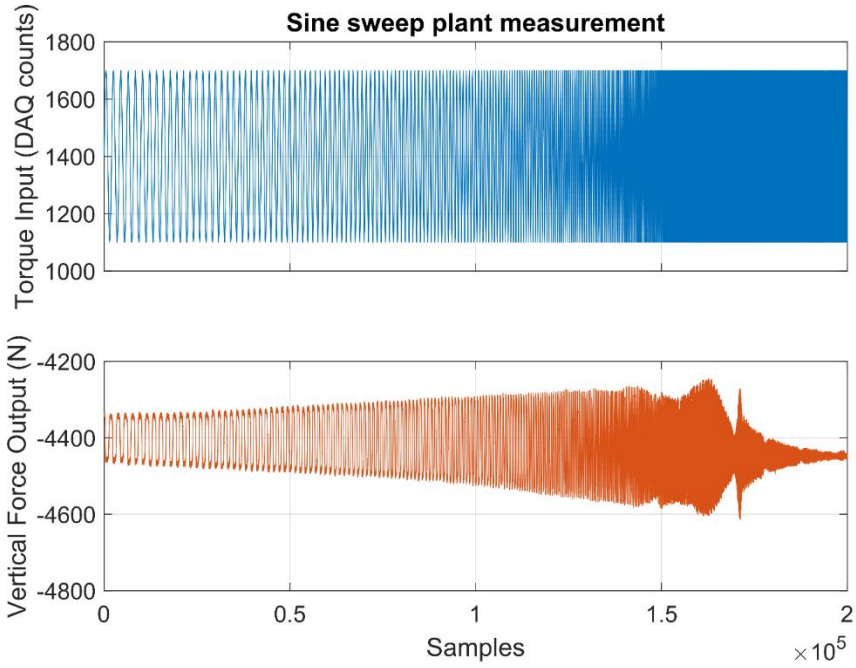


Figure 3-26: Input-Output time series plot for plant measurement in Figure 3-25

Figure 3-28 shows the frequency response plot of the plant. In the low frequency region, the gain is lower than -5 dB with the phase being around 50^{o} . There is a lightly damped complex pole pair, present at about 7 Hz which impacts the gain response in the low frequency region from DC to 10 Hz. The resonant peak at 10 Hz has a magnitude of +3.7 dB. There is also a pair of lightly damped complex poles and zeros at around 20 Hz which are separated in frequency, giving rise to a notch and a secondary peak in gain plot. It should be noted that if a loop is closed around this open loop with a gain of 1, then the closed loop would be very close to instability as the margins are virtually zero. The gain at the secondary peak (after 20 Hz) is roughly equal to zero while the phase is very close to -180^{o} .

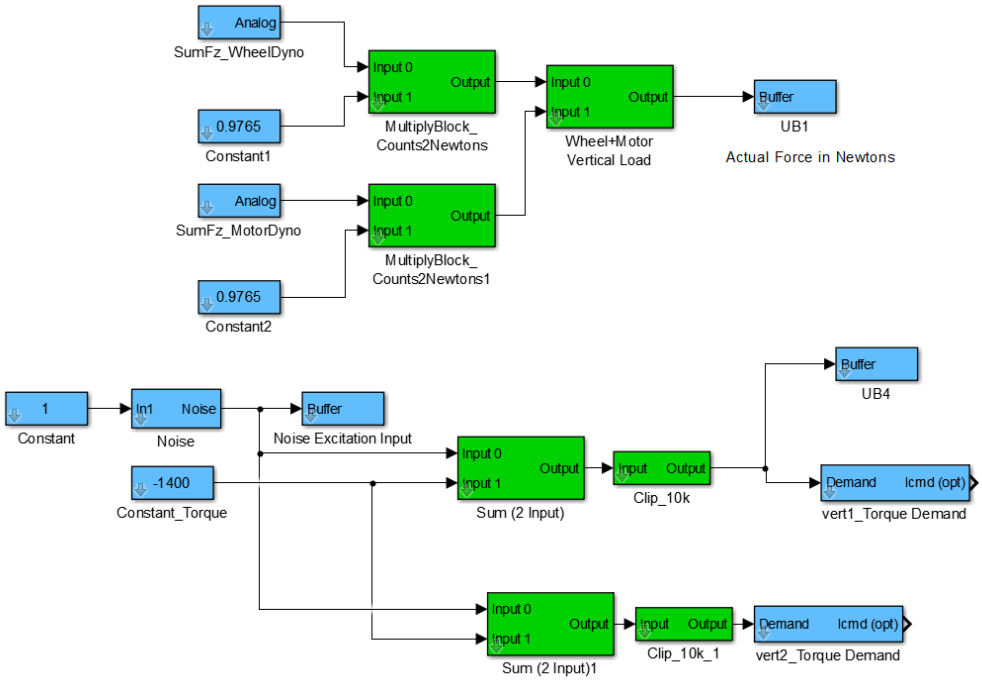


Figure 3-27: MechaWare™ model schematic for plant measurement in Figure 3-25

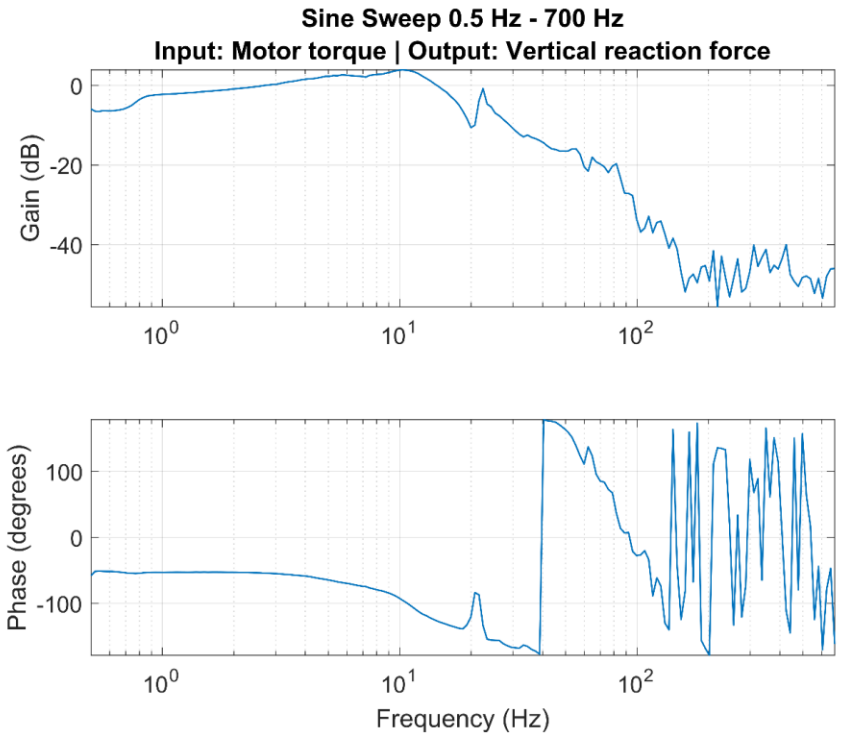


Figure 3-28 Bode plot for plant measurement experiment in Figure 3-25

Numerous attempts were made to conduct a system identification on the plant by conducting a frequency domain curve fit as well as an H1 estimation based on input-output time

series data. The results were unsatisfactory with a very low percentage of fit. Therefore, experimental loop shaping was attempted to try and obtain a controller with desired characteristics. Traditional PID control, as well as high order transfer functions with custom poles and zeros were attempted, but a satisfactory controller was good stability margins was not achieved. Thus, the single loop feedback controller design was abandoned and a different active controller design was desired.

3.10 Feedforward Force Controller

The motivation behind adopting the active methods for force control was discussed in Section 3.8, where the center of mass of the wheel had to move in a non-linear trajectory in order to "glide over" the surface irregularities. A non-linear sinusoidal motion trajectory means a sinusoidal acceleration profile, giving rise to a sinusoidal force profile, and this is what led to the single loop control architecture.

If the surface profile of the wheel and roller are known perfectly, then a control loop can be set up as shown in Figure 3-29, where a single position loop moves the servo actuators according to the position profile commanded.

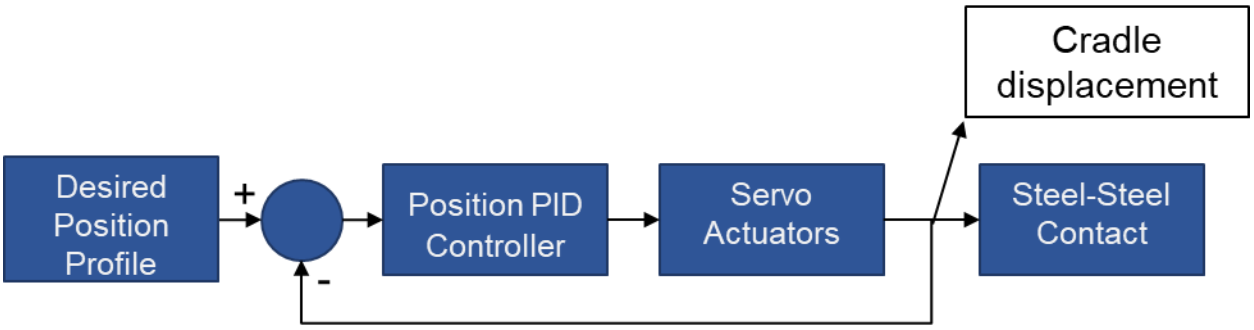


Figure 3-29: Force control system when surface irregularities are precisely known

This profile can be computed in a feed-forward fashion where the command position changes according to the angles of the wheel and the roller. To ensure the proof of concept, multiple experiments were conducted to identify the dependence of force values on the rotational angle of the wheel. One such plot was shown in Figure 3-10 where a functional dependence of vertical force fluctuations was seen on the wheel rotational angle across different experiments having different starting points, multiple rotations of wheel and no relative slippage between the wheel and the roller. The highly repetitive nature of the plot points towards adopting a feedforward-based approach.

To test the feedforward approach, a MechaWare™ model (Appendix A) was set up as shown in Figure 3-30. The value in UB10 (User buffer 10) is computed as shown in Figure 3-31. The lookup table block contains reference to a text file which as lookup table entries that are delimited by a tab. Details about the LOOKUP block can be obtained in Appendix A. The first few rows of the lookup table that was created are shown in Table 3-2. UB2 contains the rotational angle of the wheel calculated in radians. This is calculated by obtaining the encoder feedback value in counts and determining the remainder of the division operation between the encoder feedback and the counts per rotation value, which is 5242880. If we divide this remainder by counts per rotation, then we get the fraction of one full rotation that has taken place. For example, if the encoder value is 75732870, then the modulo of this when the divisor is 5242880 (counts per single

revolution) is 2332550. This remainder value will always be less than 1. If this value is divided by 5242880, then the fractional rotation is obtained: 0.4449. Note that if the encoder value is directly divided by the counts per revolution value, then 14.4449 is obtained, which means 14 full revolutions and 0.4449^{th} fraction of one revolution. If this fraction is multiplied by 6π, then the rotational angle in radians is obtained: 8.3862. The MechaWare™ operations are as shown in Figure 3-31.

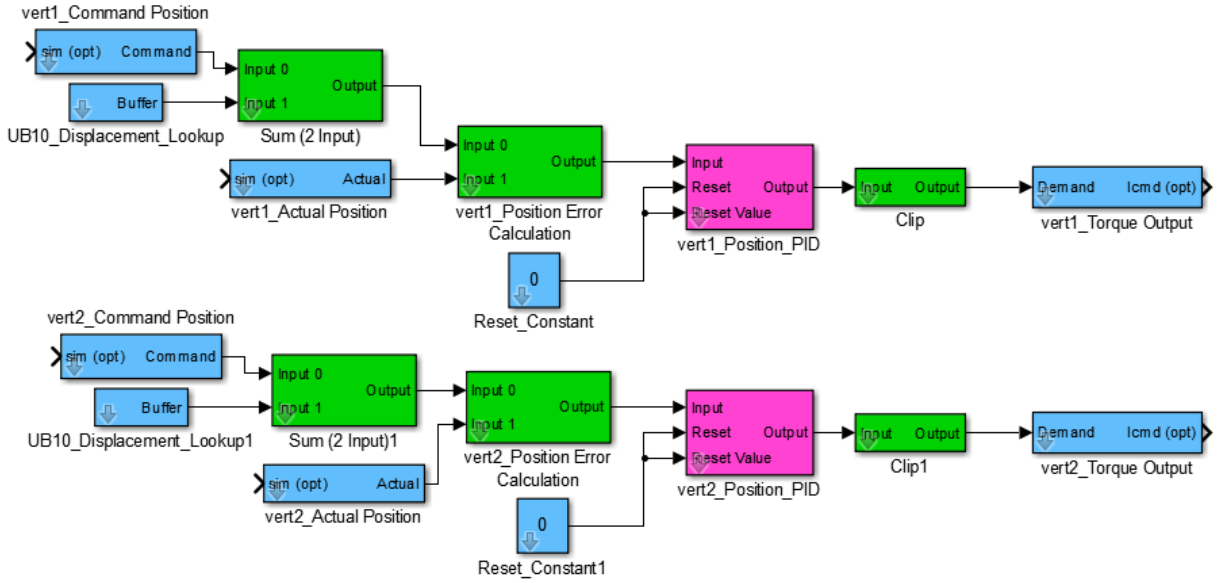


Figure 3-30: MechaWare™ model for the feedforward force control scheme

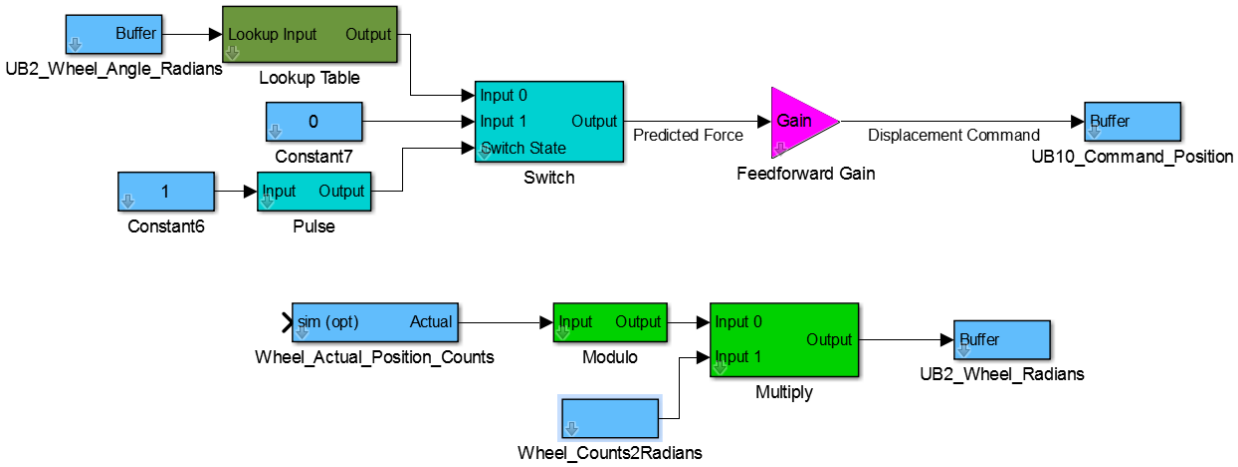


Figure 3-31: Computing the feedforward command displacement

This wheel rotation value in radians is fed into the lookup table, which interpolates and outputs a value of predicted force error based on the first column of the lookup table text file, such as Table 3-2. The switch block introduces a timer based delay for writing feedforward displacement command in User Buffer 10 and is not necessary.

Table 3-2: First few rows of a representative lookup table

Wheel rotational angle (rad)	Predicted force error (N)
0.029301	-213.854
0.060907	-207.737
0.091426	-198.49
0.122883	-195.574
0.15375	-193.853
0.18481	-184.429
0.215771	-171.123

The lookup tables were generated after conducting multiple experiments and obtaining a mean of the force error values encountered. The values output by the lookup table block are written in User Buffer 10, which is used to add a displacement value on top of the existing command value of the two actuators. The position feedback loop of the vertical actuators would respond to this displacement command and hence the force error will reduce. The results were very promising, as shown in Figure 3-32.

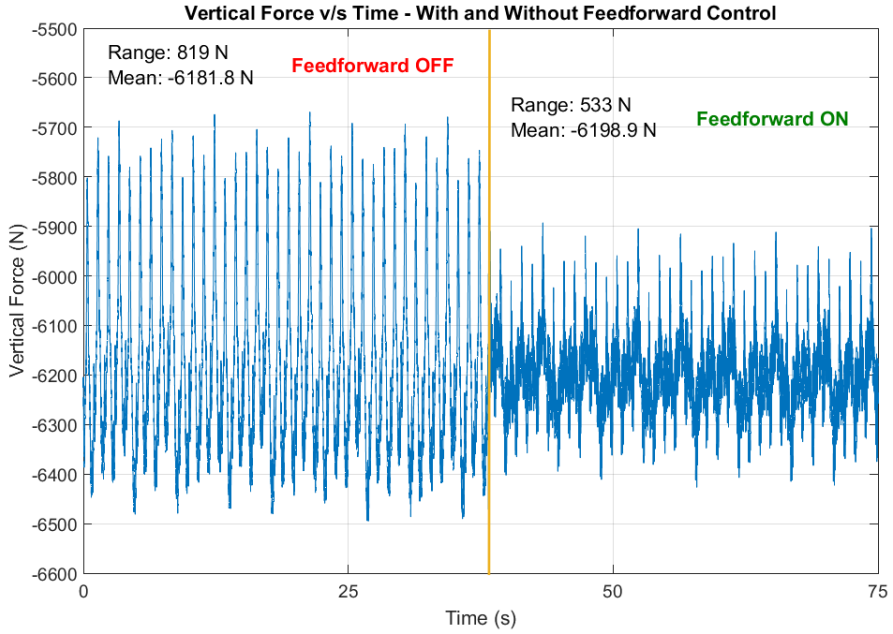


Figure 3-32: Results of the feedforward force control experiment

Feedforward control has the advantage of being very fast while being less prone to instabilities caused by delay in feedback loop. The disadvantage of feedforward control is that it cannot track the output variable since the command displacement is not a function of the output variable, which in this case is the actual force value. For example, in this case the displacement command value is entirely dependent on the wheel rotation angle and the relative slip between wheel and roller is kept to be zero. If the operating conditions change even slightly, like say the relative starting positions of the wheel and the roller angles shift due to relative slip, then the results shown in Figure 3-32 will not be reproduced.

The reader should note that the force fluctuations are a function of both wheel and roller surface imperfections, as shown in Figure 3-8. The ideal feed-forward control mechanism should

take into account both the rotational angles of wheel and roller, i.e. the displacement command output should be a function of both wheel and roller rotational angles. A linear superposition scheme can be followed, as shown in Figure 3-33. The FF (feedforward) gains can be tuned individually to obtain optimal contributions from wheel and roller geometries.

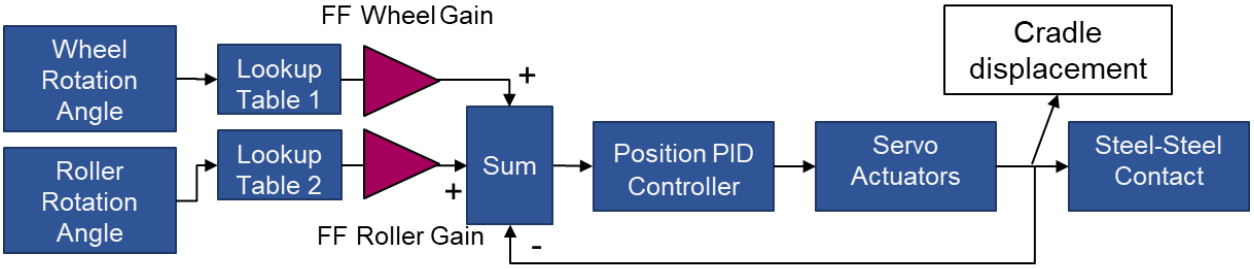


Figure 3-33: Proposed linearly superposed feedforward displacement-based force control

This approach, however, was not implemented because the feedforward lookup table had to be changed whenever something in the Rig changed. The wheel and roller surfaces change with every experiment, thus requiring the lookup table to be changed very frequently. The data collection and subsequent feedforward tuning was a very time-consuming task and was deemed impractical. This difficulty can be overcome by using adaptive signal processing techniques like the Filtered-X LMS algorithm, which can be used to change the feedforward gains and/or the lookup table. Such a method may yield better force compensation than the existing force control results mentioned in Section 3.11, and the reader is advised to refer to a standard adaptive signal processing textbook.

3.11 Cascaded Loop Force/Position Control

It is not possible to directly measure the surface profiles of the wheel and the roller; neither is it possible to predict them somehow. However, the effect of the surface imperfections can be measured via the vertical force fluctuations. Following the promising results of the feedforward method, it was decided to adopt the same architecture but add a force feedback loop and close it around the position feedback control loop. The block diagram is shown in Figure 3-34.

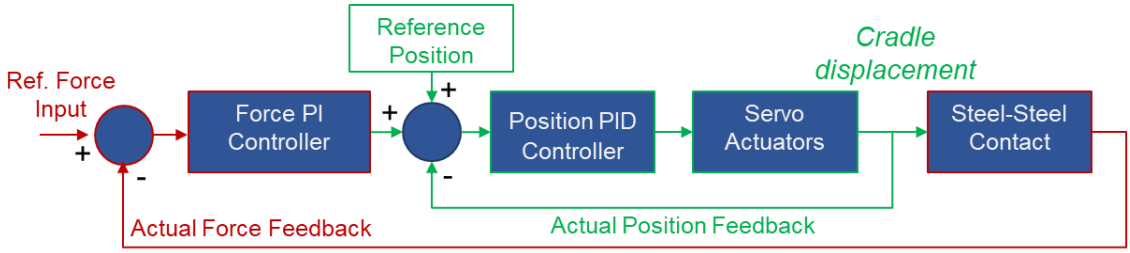


Figure 3-34: Cascaded loop force/position feedback control scheme

Such an idea was successfully demonstrated by De Schutter et. al [11] and is used for force control in industrial manipulators. For Roller Rig, this cascaded loop control system behaves like an admittance: force error input to the system results in a displacement output of the cradle. A simple way to represent the system is shown in Figure 3-34. The reference force is constantly compared against the actual force, and a force error is generated. This force error then acts as an input to the controller that behaves like a compliance, which displaces up or down depending on

the input force error. This displacement of wheel relative to the roller would provide the wheel with the desired motion profile for the center of mass, thereby reducing the amount of contact force required and/or the resulting contact deformation.

The first step in any control system design is to measure and/or estimate the open loop dynamics. Based on an estimated stiffness, a proportional gain was added to the system and an open loop was constructed as shown in Figure 3-35.

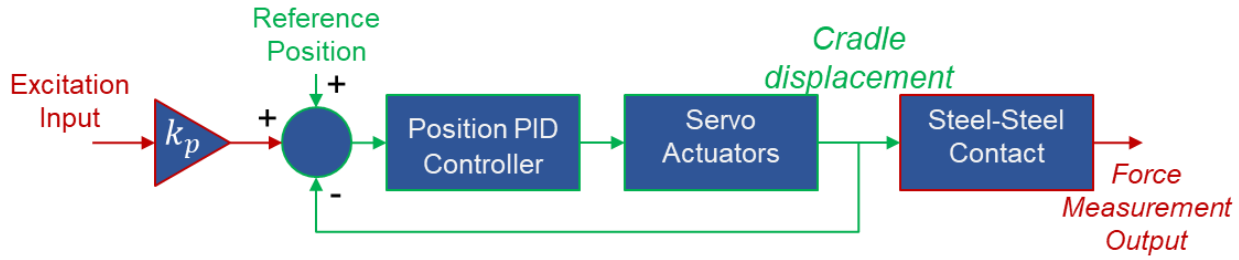


Figure 3-35: Open loop measurement scheme

The frequency response is as shown in Figure 3-36. The open loop measurement shows that if the feedback loop is closed around this controller, the control system is going to be unstable. However, this bode plot is favorable for closed loop design because of the maximally flat gain and phase response in the low frequency region. If two poles with a damping ratio greater than 0.7 are kept somewhere around 7 Hz to 10 Hz, then the undesired dynamics at the unmatched underdamped pole-zero pair at 20 Hz and the high frequency amplification can be attenuated, yielding us a maximally flat open loop gain response in the low frequency region. The same is illustrated in Figure 3-37.

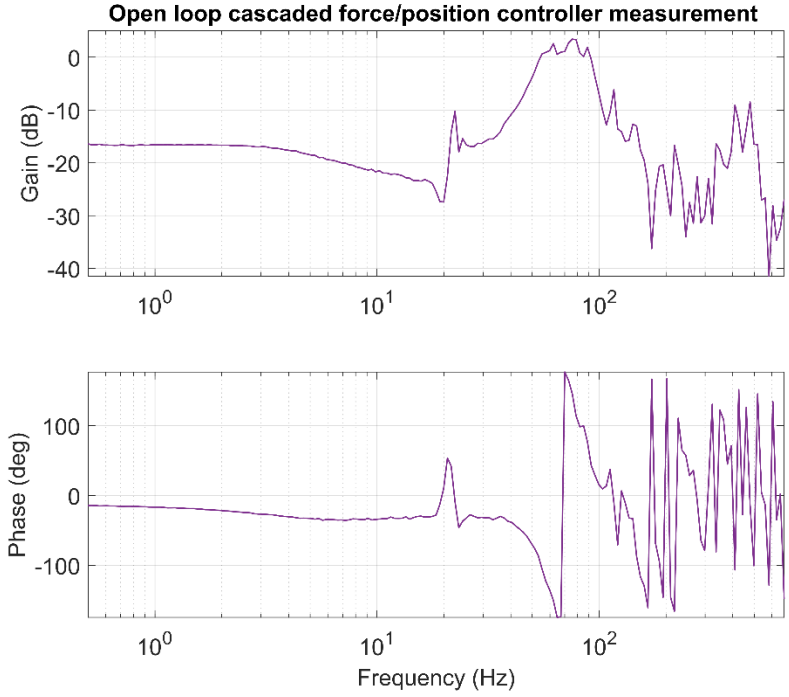


Figure 3-36: Open loop measurement for cascaded loop force/position control

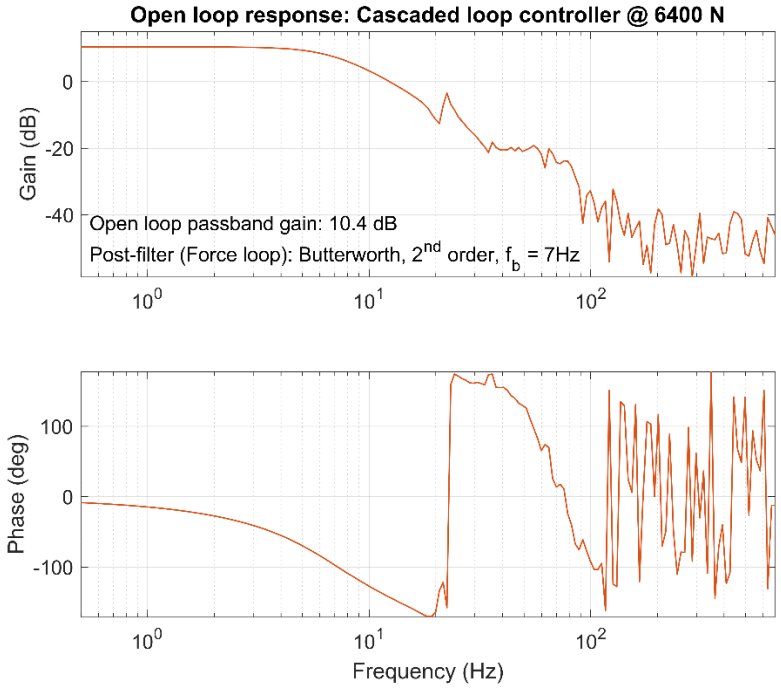


Figure 3-37: Bode plot for open force loop with P controller in series 2nd order Butterworth filter and closed PID position control loop

The open loop system shown in Figure 3-37 has a gain margin of 10.72 dB and a phase margin of 36.54**⁰** . These values were estimated from the phase/gain crossovers observed in the frequency domain plots.

There are two ways to simulate a closed loop system response based on the measured open loop response:

1. Simulations based on System Identification
2. Estimation based on open loop response data

The term "system identification" is used to represent a vast scientific field that aims to identify an analytical model for a process based on observed signal. There are multiple methods to conduct a system identification exercise; one of them is called non-parametric frequency domain system ID. The first step of the process is to obtain a relatively noise-free data estimate of the transfer function. There are two estimates for this [13]:

- H1 estimate:

$$H_1(mF) = \left(\frac{S_{uy}(mF)}{S_{uu}(mF)}\right) \tag{1}$$

- H2 estimate:

$$H_2(mF) = \left(\frac{S_{yy}(mF)}{S_{yu}(mF)}\right) \tag{2}$$

$$S_{uy} = U(mF)^{*}Y(mF): Dot\ product\ of\ complex\ conjugate\ of\ input\ with\ output$$

$$S_{uy}\ is\ usually\ not\ the\ same\ as\ S_{yu}$$

$$(U(mF) = complex\ fourier\ transform\ of\ input\ time\ series\ U)$$

Each method has some pros and cons, and the reader is advised to refer to any modal analysis or system identification textbook for more detailed explanation. For the Roller Rig, we have access to a utility (Bode Tool) that directly computes the transfer function from specified input-output data. This data can be used to estimate the transfer function in MATLAB using the system identification or the signal processing toolboxes. MATLAB functions tfest() [14] or invfreqz() [15] can be used to perform a frequency domain curve fit. An analytical expression for the open loop transfer function was obtained, and the closed loop response was simulated as:

$$CLTF = \frac{OLTF}{1 + OLTF} \tag{3}$$

Where CLTF = Closed loop transfer function and OLTF = Open loop transfer function.

Another way to simulate a closed loop response is to directly use the frequency domain transfer function estimate data in equation (3). This is a valid approach and is free of any approximation errors that the system ID inevitably will have. However, this method might show false dynamics that are a result of noisy measurement and may not be a true representation of open loop dynamics. The reader is advised to set open loop parameters in order to obtain a conservatively stable simulated closed loop performance. After that, the reader should carry out an actual closed loop measurement to verify the simulated results. This would minimize the risk of hitting instabilities and potential mechanical damage. Both of the methods mentioned above were used for closed loop simulation and the results are as shown in Figure 3-38.

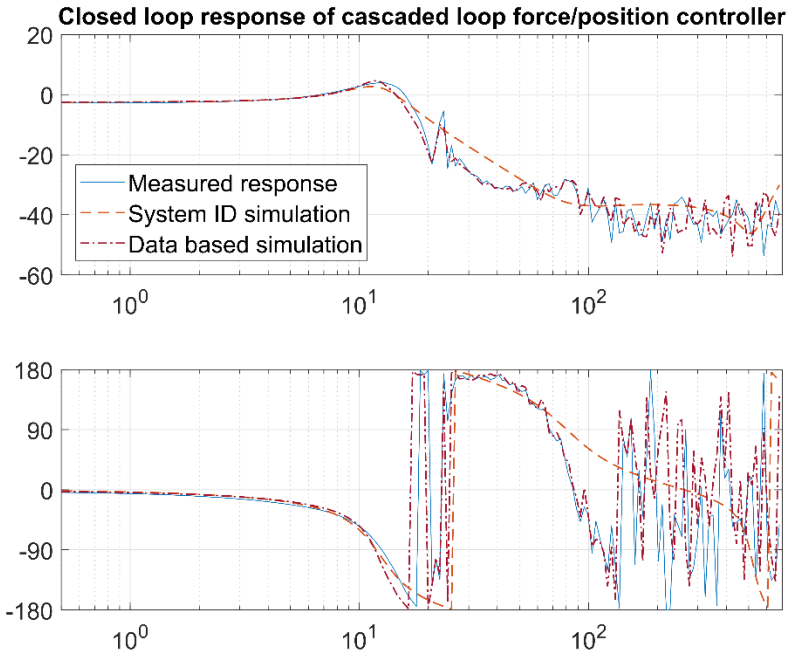


Figure 3-38: Closed loop design of the cascaded loop force/position controller

It can be seen that both of the estimation methods are in good agreement with the measurement data for low frequency range below 10 Hz. After 10 Hz, the analytical estimation method fails to capture the peak-valley pair at about 20 Hz which the data derived estimation method replicates. There are also some differences in the phase crossovers between the measured

and estimated data. The resonant peak was found to be a little higher which was reduced by changing the proportional force loop gain.

The final performance of the design was judged by the time series performance characteristics. Two experiments were set up: One in position control mode, and another in the tuned cascaded loop force/position control. The starting load was set to be approximately the same at 6400N. Both wheels were pressed together and rotated and the vertical force fluctuation data was recorded. The time series were de-trended so as to only focus on the fluctuations about the mean. The time series were subjected to the same low pass filtering with a non-causal MATLAB filtfilt() filtering so as to cause minimum phase distortion. The filter chosen was a 2nd order Butterworth with a break frequency of 10 Hz. The improvement in disturbance rejection can be clearly seen in one of the results as reported in Figure 3-39.

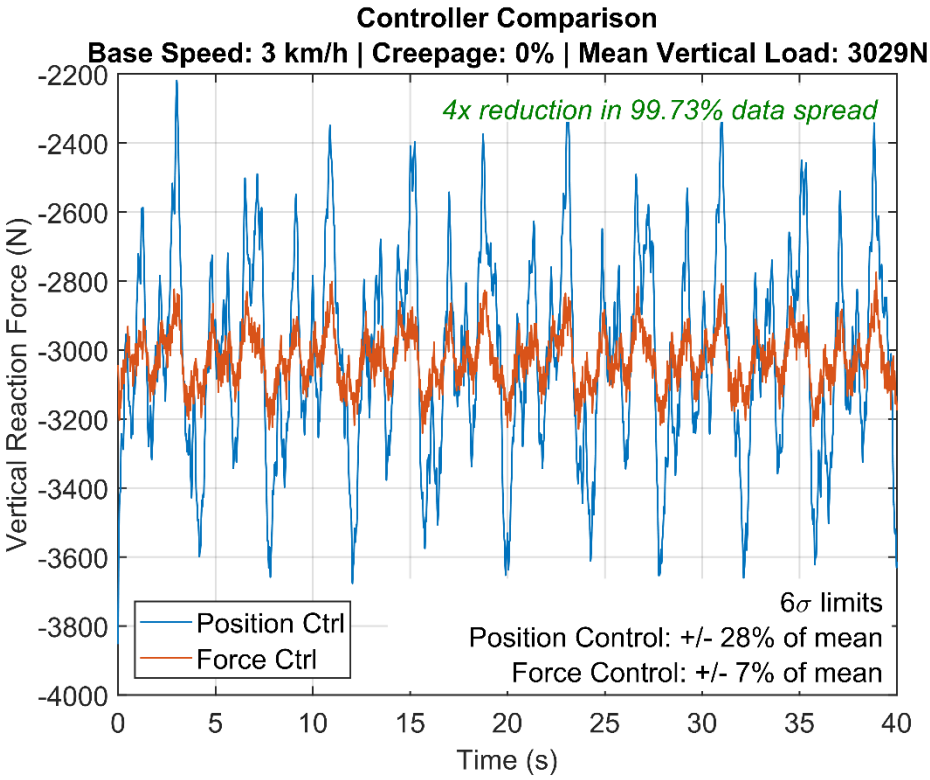


Figure 3-39: Vertical force disturbance comparison with force control and position control

There is a four-fold reduction in the standard deviation of the vertical force fluctuation with a properly tuned force controller, when compared to the position control. In terms of disturbance power, this corresponds to a 12 dB reduction. Thus, the cascaded loop force/position control method achieves a good disturbance rejection while being stable across different operating conditions. Stability and robustness is discussed in Section 3.12.

3.12 Results

The controller was tested at multiple loads and speeds to test for stability and robustness. The stability was significantly impacted by the vertical load operating point, and the force control gains have to be changed in order to obtain a stable controller. A proper tuning procedure is suggested so as to achieve a good disturbance rejection at all operating points.

3.12.1 Stability analysis for multiple loading conditions

There is no "one size fits all" solution for the force controller gain, as the proportional gain of the force controller has to be changed for every force level. This is because the steel shows a non-linear stiffness curve. The contact becomes stiffer with an increase in vertical load. As a result, to ensure stability, the proportional gain should be decreased as the vertical load operating point is increased. Consider the operating point of 3000N. The open loop measurements show the frequency response, as in Figure 3-40.

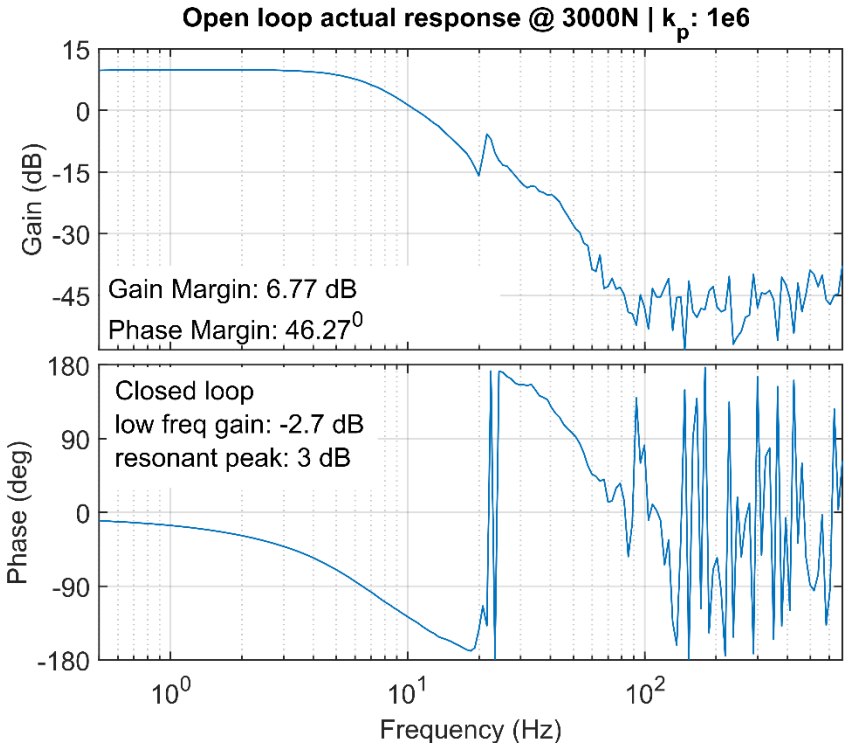


Figure 3-40: Frequency response plot and margins with suitable gain at operating point of 3000N

The closed loop is stable with gain margin greater than 2 dB and phase margin in 35^{o} -45. The controller causes a 12 dB reduction in the power of vertical force fluctuations when compared to position control. The same open loop measurement when carried out at the operating point of 6000 N gives a frequency response as shown in Figure 3-41.

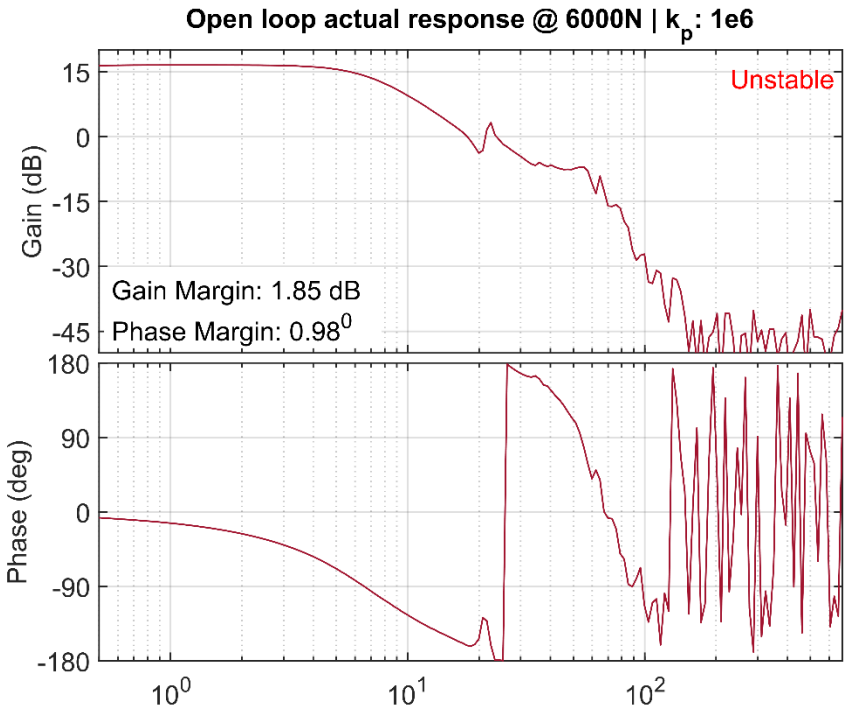


Figure 3-41: Frequency response and margins at 6000N operating point with the same gains as at 3000N

The non-linear stiffness characteristic of the steel-on-steel contact causes a rise in open loop gain, as can be seen above. This increase elevates the entire gain curve which results in an increase of the phase crossover frequency. Increase in the phase crossover frequency reduces the phase margins and in this case, the closed loop controller would be unstable. The solution is to change the force loop gain for every level and to decrease it for higher operating point of force. Table 3-3 shows the idea in a convenient form.

Table 3-3: Gain table for different force operating point

Force operating point (N)	Gain value
3000	8e5
6000	4e5
9000	2e5

These gains should first be verified by testing on the open loop system. If open loop system has good margins, like a phase margin of 35° -45° and a resonant peak less than or equal to 3dB, then the closed loop system will be stable and there will be a good disturbance rejection with the vertical force fluctuation power reduction of 12 dB. Figure 3-42 shows the disturbance rejection results at 3000N if the above tuning methodology is followed.

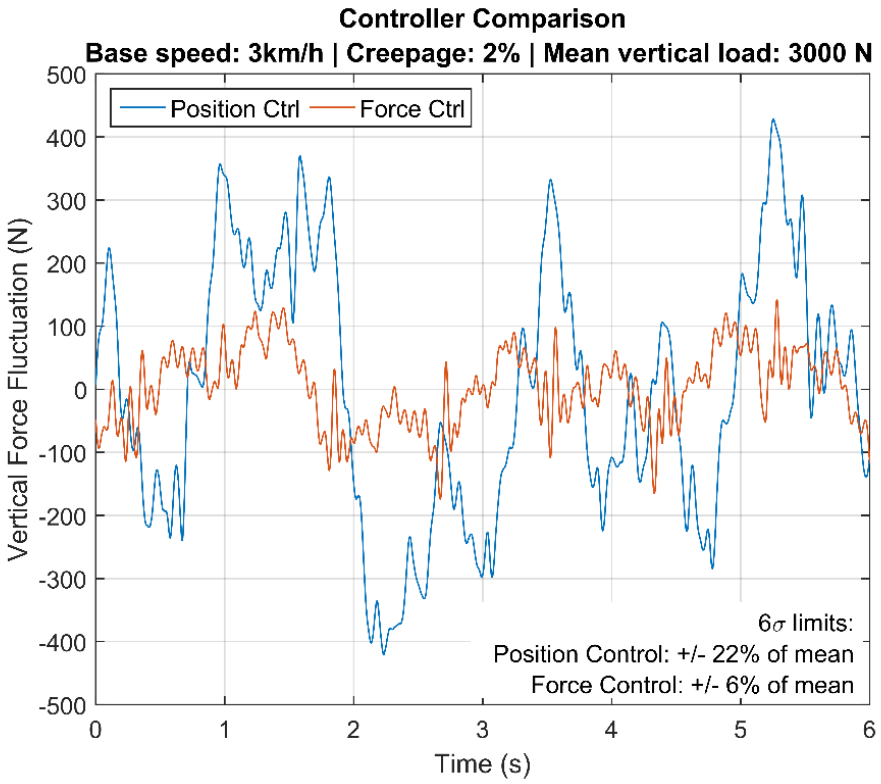


Figure 3-42: Disturbance rejection due to force control in contrast to position control at 3000 N; kp = 8e5

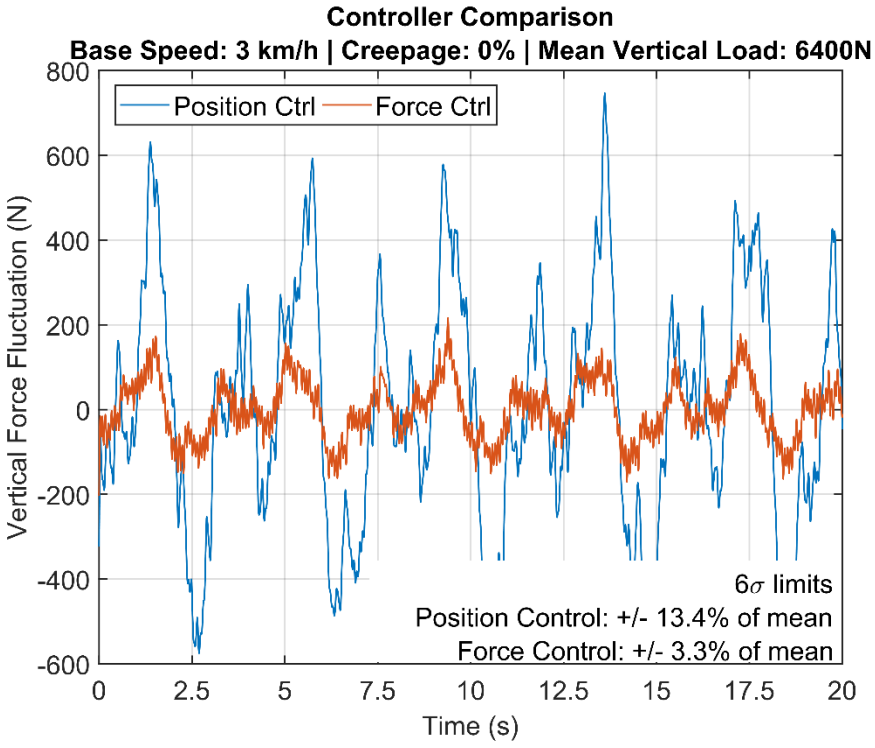


Figure 3-43: Controller comparison at 6400N. kp: 4e5

Figure 3-43 shows the disturbance rejection by the controller when the force loop gain was reduced to kp = 4e5. It is evident that by retuning the controller for different operating points, a good disturbance rejection will be achieved. It is noteworthy that the force fluctuations as a percentage of the mean vertical load value decreases for position control as the vertical load is increased. However, the force controller will achieve a reduction of 400% irrespective of the fluctuation level in force control.

An alternative to retuning would be to set up a gain table which sets gains according to the load conditions. This can be implemented using lookup blocks in MechaWare™ and actual vertical force value. This can be a good improvement for the future.

3.12.2 Stability analysis for multiple speed conditions

The desired bandwidth of the force controller depends on the frequency at which the disturbance is generated. The force fluctuation is caused by minute deviations in wheel and roller contact surface in position control. Assuming a vertical contact stiffness of 10^6 lb/in. a surface deviation of +/-0.0001 results in +/-100 lb. (+/-475 N) of force fluctuations or 200 lb (1000 N) peak-to-peak variation in vertical force, which is what we experience when the roller Rig is operated with position control. Thus, the maximum frequency at which the disturbance will take place should be a multiple of the maximum frequency at which the wheel and the roller can rotate. The Rig has a maximum speed of 10 mph [7], which corresponds to a rotational frequency of 6.74 Hz for the wheel (faster moving).. The experimental results were already shown for 3km/h speeds previously. Results for 6 km/h are shown in Figure 3-44 and 9km/h are shown in Figure 3-45.

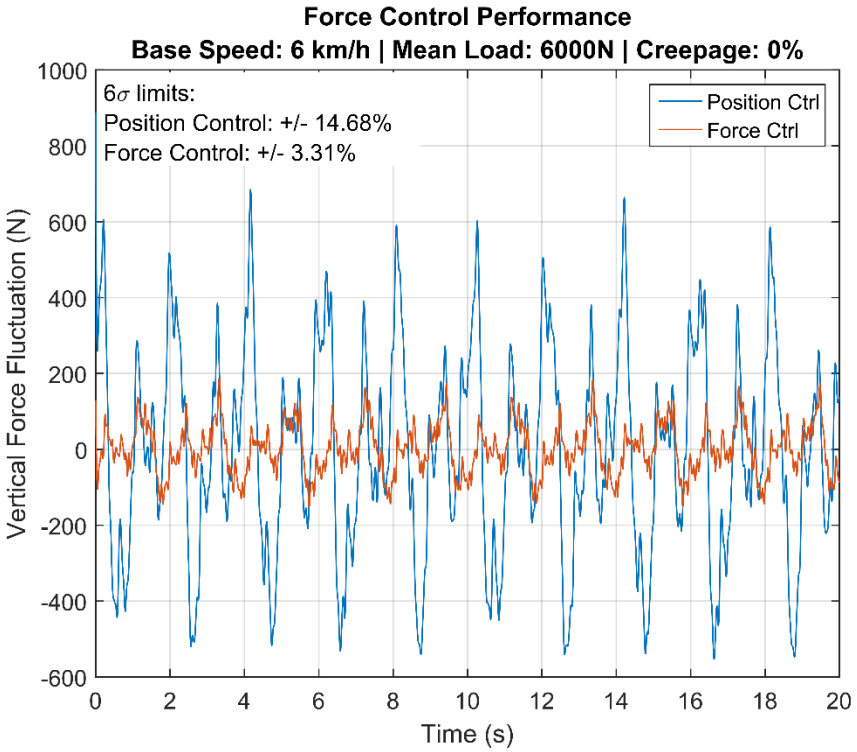


Figure 3-44: Force control performance at 6km/h base speed, 0% creepage and 6000N load

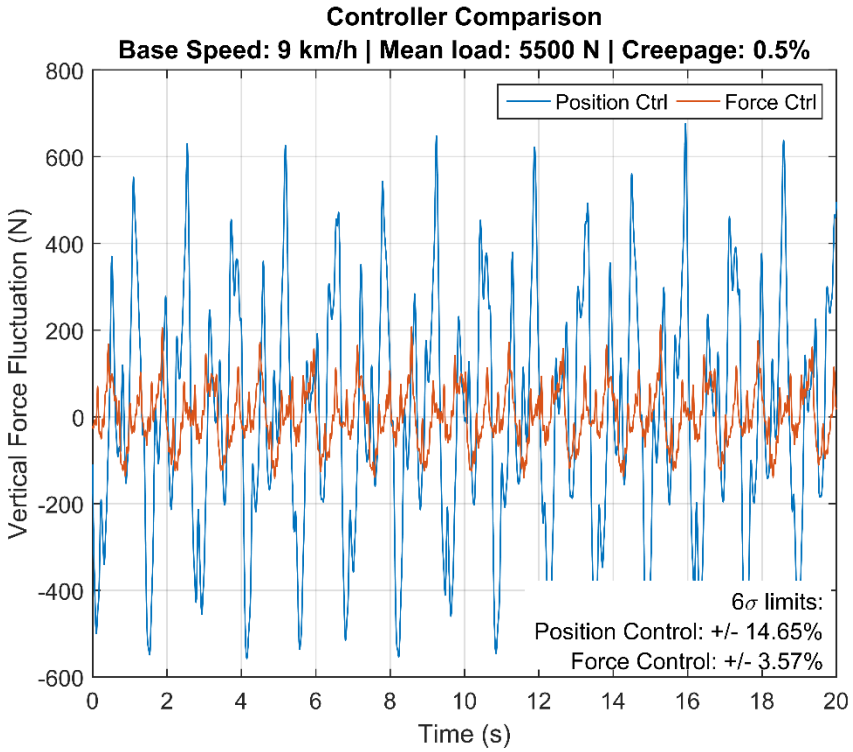


Figure 3-45: Force control performance at 9km/h, 0.5% creepage and 5500N load

3.12.3 Stability analysis with anti-alias filter in vertical force feedback

Anti-alias filters are necessary to prevent "aliasing" in measured data, which refers to the phenomena when signals of higher frequency are mistakenly interpreted as signals of lower frequency due to an insufficient digital sampling rate. An anti-alias filter ensures that the signal is "band-limited" by attenuating all high frequency components above a certain frequency. Anti-alias filters are always analog. A good anti-alias filter design would have an attenuation of at least 20dB at the Nyquist frequency.

Anti-alias filters introduce an additional delay in the sensor data. If this measurement channel is used for feedback, then the delay causes a decrease in phase margins for the feedback control system. A reduction in phase margins reduces the relative stability of the control system and a large feedback delay can also make the feedback control system unstable. The Rig uses the vertical force channels from the dynamometers as one of the feedbacks for the cascaded loop force/position controller. This Section presents a relative stability comparison for different implementations of anti-alias filters.

The charge amplifiers of the Rig's force-moment measurement system provide analog 2nd order low pass Butterworth filters with the following break frequencies as anti-alias filters:

- 100 Hz
- 300 Hz
- 500 Hz
- 1000 Hz
- 2000 Hz
- LP Off (No low pass filter)

Based on the current sampling frequency of 2000 Hz, the Nyquist frequency becomes 1000 Hz. Choosing a 100 Hz 2nd order Butterworth low pass filter gives an attenuation of 40 dB at Nyquist frequency. Choosing a 300 Hz 2nd order Butterworth low pass filter gives an attenuation of 20 dB at Nyquist frequency. Choosing other break frequencies will yield lower attenuation at Nyquist frequency and it is recommended that an attenuation of at least 20 dB at Nyquist frequency. These filters are compared with the no low pass filter option and the open loop frequency response. Easy phase margin comparison is provided in Figure 3-46.

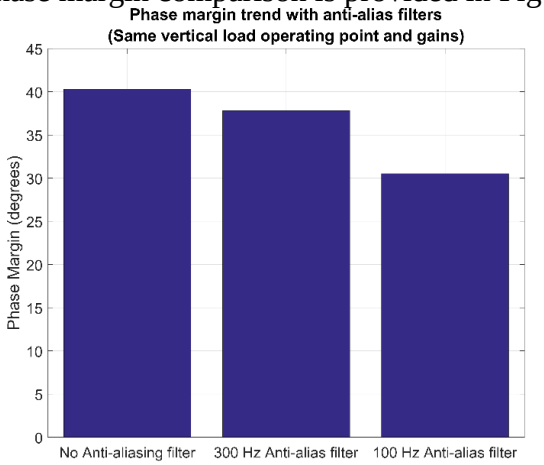


Figure 3-46: Bar graph showing the effect of anti-alias filters on phase margins of force controller

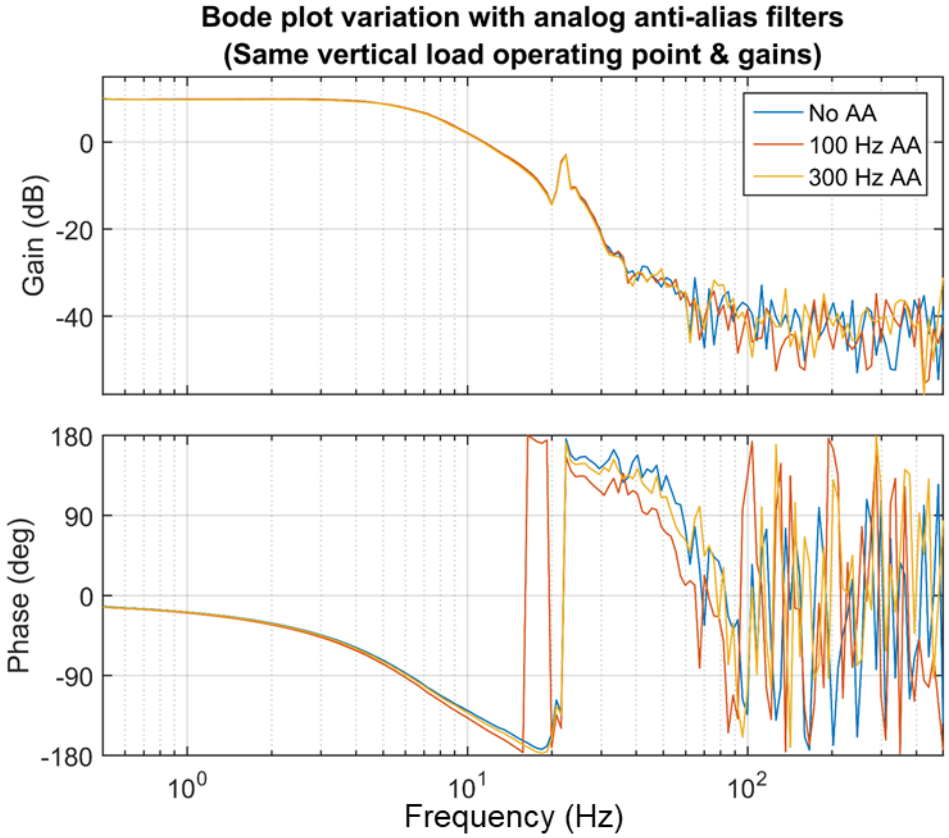


Figure 3-47: Open loop frequency response comparison with low pass filters

Figure 3-47 shows the open loop frequency response comparison with and without anti-alias filters. It can be seen that the gain response is the same for all low pass filter choices. The phase response differs for different low pass filter options. 100 Hz low pass filter shows the greatest drop in phase margins when compared with the no low pass filter option. The cascaded force/position feedback controller is tuned without any anti-alias filter but choosing a 300 Hz low pass filter provides good anti-aliasing property due to the 20 dB attenuation at Nyquist frequency while not causing severe drops in phase margins that impact stability.

3.13 Conclusions

A feasible and practically viable solution for vertical force disturbance rejection has been reached through a practical data analysis and physics based fundamental reasoning. Reasonably good compensation has been obtained for loading conditions from 2500N to 9500N and base speed conditions from 0.5 km/h to 9km/h. Exhaustive testing has verified that the controller has a good mix of robustness and performance, making it ready to be put into practical testing applications.

4. Automation Software Development

4.1 Case for Automation

4.1.1 Introduction to testing

This section presents an argument for testing automation on the Roller Rig. After the force measuring system was configured and the measurement drift issue was resolved, baseline testing for the Rig was commenced. The chosen test was to generate a creep-creepage curve. Creep-creepage curve is a traditional railway contact mechanics characteristic, where the traction forces are plotted against the relative slippage between two bodies. Creep-creepage curves can be obtained for various boundary conditions but the most widely reported is the longitudinal creep-creepage curve. The x-axis is longitudinal creepage, expressed as a percentage and computed according to the following relationship. The y-axis is the normalized creep force, which is longitudinal force normalized by the vertical load. Normalized creep value is obtained for multiple creepage points for generating the curve. This plot is influenced by a lot of conditions some are listed as follows:

- Locomotive speed
- Relative slippage (creepage)
- Angle of attack
- Cant angle
- Vertical load
- Third body layer at the contacting surfaces
- Temperature of contact
- Coefficient of friction

These conditions are uncontrolled when testing on a locomotive out on the track. The indoor test environment provides at least a limited degree of measurement and control which is significantly greater than field experiments. The scientifically scaled design allows the Rig to maintain a close resemblance with contact conditions present in the field.

To establish an accurate baseline, all the boundary conditions mentioned above have to be controlled. This can be ensured by appropriate design of experiments.

4.1.2 Experiment design

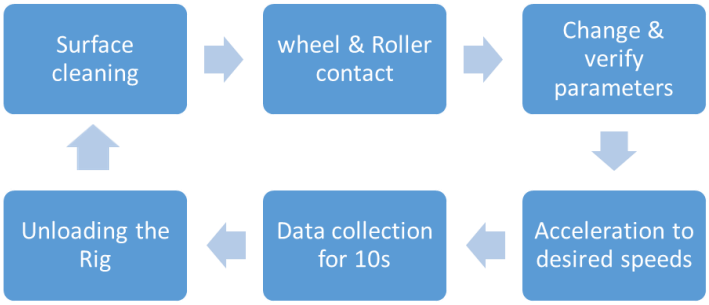


Figure 4-1: Experiment workflow with position control

Figure 4-1: Experiment workflow with position control shows the experiment workflow when the baseline tests were conducted. This experiment workflow gave the maximum possible control over the boundary conditions while producing repeatable results. A time cost calculation

of this workflow is presented in Table 4-1. This time calculation is assuming ideal testing condition and no breaks, and does not take into account the time loss due to human errors, faulty data collection, etc.

Table 4-1: Time cost calculation for creep-creepage curve calculation with one set of parameters

Task	Time
Surface cleaning	180 s
wheel-Roller Contact	30 s
Verifying vertical load condition	30 s
Steady state attainment	60 s
Data collection	600 s
Unloading	60 s
Number of creepage points	24 s
Time for one repetition curve	6.4 hours
One-time machine initialization time cost	40 minutes = 0.67 hours
Number of repetitions per creepage point	5
Total time for one C-C curve	5 days at 7 hours per day of testing
Personnel needed with manual testing	2

If a multi-parameter study were to be conducted, the testing time can easily stretch over one entire month assuming that multiple shifts of two or more personnel are working round the clock on the Rig. The tasks are very time consuming, but they do not require any special aptitude. One person would sit at the workstation, making multiple clicks to set boundary conditions, start data logging and stop the testing. Another person is required to clean the wheel and roller surfaces to ensure that no third body layer effects are seen in the measurement data. When the two wheels are pressed against each other and they rotate while slipping, the debris formation causes a significant variation in the traction forces over time which is referred to as "natural third body layer effects." Thus, the case for automation comes from testing experience with the Rig

4.2 Requirements

Before starting a big project like software design, it is important to determine the requirements of the software. Following requirements were decided:

- The software should be able to continuously poll any parameter of interest present on the SynqNet network, and display them real-time on a graphical user interface.
- The software should be able to provide the means for manual emergency stop and abort options.
- The software should be able to execute a time-based trajectory which would be input by the user as a file (tab delimited txt or csv).
- The software should continuously monitor for faults and if in case a fault occurs, the software should immediately shut down the testing and restore the Rig to a safe state, irrespective of whether the Rig is executing commands or is idle.
- The software should be able to automatically record data if some conditions are met. The data should be recorded on the hard drive in near real time. For example, if the wheel velocity reaches the commanded velocity, collect and record the force data, the encoder data and the time stamp data in a tab delimited text file.

4.3 Project Milestones

In order to streamline the development that meets the expectations, a workflow sequence was devised which would also serve as milestones that track the progress of the code. These are:

- Initialize the program and communicate with the controller card to fetch a parameter that would serve as a success flag.
- Start and stop motion on a motor. Poll motion supervisor status and determine whether a motion is underway or if there is a fault on some motors.
- Fetch and poll a parameter from the SynqNet network.
- Set a parameter on the SynqNet network and poll it on GUI.
- Stop a motion command that is already underway.
- Execute "forward motion-STOP-reverse motion" sequence based on motion supervisor status.
- Append the existing motion command on a servo motor to construct a queue of motion points (trajectory).
- Set up an event listener and poll all the events being received by controller.
- Execute a step-dwell type velocity-time trajectory motion with a variable time gap. Ensure that the motor stays at that velocity for the time gap commanded.
- Set up an event listener and monitor the trajectory execution using polled events
- Create a recorder object and implement in a separate thread where recording can be started or stopped using GUI events.
- Integrate recorder thread with events from motion supervisor to automatically record data when the actual motor velocity reaches the desired velocity.
- Implement three separate threads (motion thread, event monitor and recorder thread) with a GUI front-end to command automatic motion and poll network parameters.
- Repeat the same process for multiple motors/ degrees of freedom.

As of yet, all of the above milestones have been achieved except for the last one. Multi-axis expansion is expected to be a reimplementation of single axis automation with increased logical comparisons and should not be difficult. However, multi-axis automation requires a precisely defined use-case so that the effort invested in software development reaps tangible time and economic benefits. The progress achieved with single axis automation should set a template for future automation and probable hardware-in-the-loop Automation.

4.4 Development process

This section assumes a good understanding of C++ from the reader. The reader should also be familiar with the MPI environment and should have some experience with running the sample MPI applications. The reader should have access to the MPI wizard in the Visual Studio environment and should have successfully created the default files from the wizard. The development process will be elaborated by means of the milestones mentioned earlier. The focus is on explaining the flow of logic rather than providing a line-by-line explanation of the code.

The reader should note the use of dynamic memory allocation techniques and extreme care should be exercised to prevent and/or locate and/or address memory leaks. The code provided has been taken out of GUI v1.1 and/or v1.2 files

4.4.1 Communicate with controller and fetch a parameter

Task flow diagram

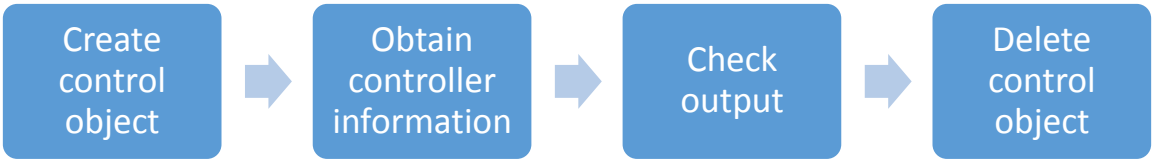


Figure 4-2: Logic flow diagram for controller communication setup and verification

Code

The code below creates a control object, fetches a controller parameter (firmware option number) and displays it as a trace on the debug window of Visual Studio.

```
MPI_RESULT returnValue; // standard return message for MPI

MPIControl control; // handle to control object
MPIControlInfo ControlInfo; // controller info object

/* Declaration and initialization of controller details ONLY for Roller Rig*/
MPIControlType      controlType;
MPIControlAddress   controlAddress;
controlType = MPIControlTypeDEFAULT;
controlAddress.number = 0;

try{ //always use try and catch functionality for exception handling and error finding

returnValue = mpiControlCreate(&control, controlType, controlAddress);
CheckMESSAGE(returnValue);
returnValue = mpiControlInfo(control,&ControlInfo);
CheckMessage(returnValue);
/* trace firmware number on debug output. ControlInfo class stores all controller information */
TRACE("Firmware number: %d\n", ControlInfo.firmware.option);

}
catch (MpiException& e)
    {
        // Handle MPI exceptions here
        CString    strData;

        strData.Format("MPI error %s", e.what());
        AfxMessageBox(strData);
    }
    catch (...)
    {
        // Handle unknown exceptions here
        throw;
    }
}
```

4.4.2 Start and stop motion on a motor

The Section provides code snippets for commanding motion, stopping motion and getting motion status from the motion supervisor.

Task Flow Diagram

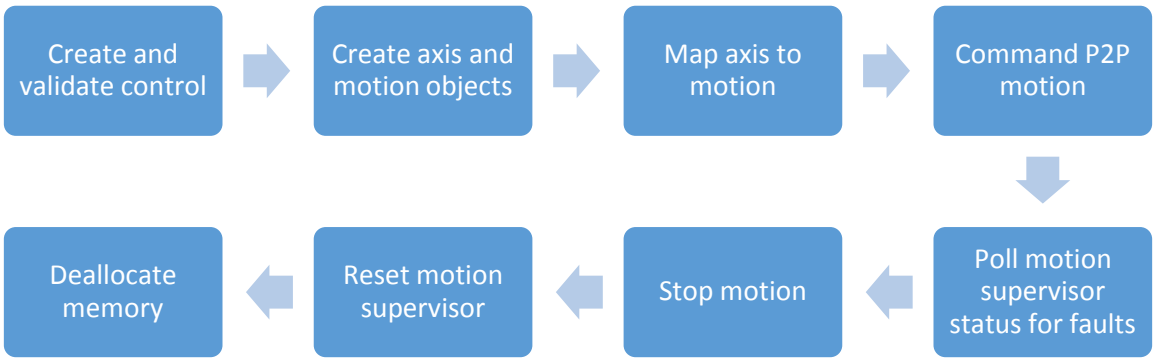


Figure 4-3: Starting and stopping motion on a motor

Code

```
1.  // definitions for easy interpretations
2.  #define AXIS_WHEEL (0) // preprocessor definition containing wheel axis number
3.  #define MS_WHEEL (0) // preprocessor definition containing wheel motion supervisor numb
    er
4.  .
5.  .
6.  .
7.  // declaration of object handles required
8.  MPIMotion msWheel; // Motion supervisor object for the motor
9.  MPIAxis axisWheel; // axis object for the motor
10. MPIMotionAxisMap    motionAxisMapWheel; //to map axis to motion supervisor
11.
12. //implement a try-
    catch routine as shown in previous Section at every Section of code where
13. // CheckMessage() is used.
14.
15. // control is carried over from previous milestone
16.
17. //create and assign axisWheel handle to axis number AXIS_WHEEL
18. returnValue = mpiAxisCreate(&axisWheel, control, AXIS_WHEEL);
19. CheckMessage(returnValue);
20.
21. //create and assign msWheel handle to motion supervisor number MS_WHEEL
22. returnValue = mpiMotionCreate(&msWheel,control, MS_WHEEL);
23. CheckMessage(returnValue);
24.
25. // mapping axis0 to MS wheel
26. // obtain existing axis map from motion supervisor
27. returnValue = mpiMotionAxisMapGet(msWheel, &motionAxisMapWheel);
28. CheckMessage(returnValue);
29.
30. //change parameters of interest in the motionAxisMapWheel object
31. motionAxisMapWheel.count = 1;
32. motionAxisMapWheel.number[0] = AXIS_WHEEL;
33. //set the modified motionAxisMapWheel object as the new axis map for motion supervisor
    msWheel
```

```
returnValue = mpiMotionAxisMapSet(msWheel, &motionAxisMapWheel);
CheckMessage(returnValue);
// the motion supervisor axis mapping is now complete. Motion can be commanded
// make sure a stable controller is in place amplifier is enabled before proceeding

// there are multiple options to command motion. Shown here is simple trapezoidal P2P move

returnValue = mpiMotionSimpleTrapezoidalMove(msWheel,GOAL_POSITION,VELOCITY,ACCELERATION,DECELERATION);
CheckMessage(returnValue)
.
.
.
// to stop a motion, use the following code
returnValue = mpiMotionAction(msWheel,MPIActionSTOP);
CheckMessage(returnValue);

// to clear fault on motion supervisor use the following code
returnValue = mpiMotionAction(msWheel,MPIActionRESET);
CheckMessage(returnValue);

//to poll for motion supervisor status, use the following code
MPIMotionStatus MoStatus; // declaration for object that will contain motion supervisor status
returnValue = mpiMotionStatus(msWheel,&MoStatus);

// MoStatus will contain an enumerated member called state. Following code explains the comparison
// and interpretation exercise
 switch(MoStatus.state){
        case MPIStateIDLE :
            printf("IDLE\n");
            break;

        case MPIStateMOVING :
            printf("Moving\n");
            break;
        case MPIStateSTOPPING :
            printf('stopping...\n");
            break;
        case MPIStateSTOPPED :
            printf('stopped\n");
            break;
        case MPIStateSTOPPING_ERROR :
            printf('stopping Error\n");
            break;
        case MPIStateERROR :
            printf("Axis in Error\n");
            break;
        default:
            printf("Invalid Msg\n");
        }

// clear memory after program completion. deletion must be in reverse order of allocation and/or initialization
returnValue = mpiMotionDelete(msWheel);
CheckMessage(returnValue);
returnValue = mpiAxisDelete(axisWheel);
CheckMessage(returnValue);
returnValue = mpiControlDelete(control);
```

```
90. CheckMessage(returnValue);
```

4.5 Software Command Structure

The automation software is designed to be capable of carrying out multiple tasks simultaneously. Such a design is an absolute necessity while controlling something as complex as the Roller Rig. The parallel tasks, with the rationale and expectations are mentioned as follows:

GUI thread: The primary interface for user control and intervention. This thread has to be the master thread and should be capable of shutting down the entire machine or parts of it by the press of a button. This GUI ideally should also be able to change the trajectories on the fly or change the behavior of the machine based on the user's whim. The data recording location should be decided by this GUI and also the option to record the data.

Poll thread: To continuously poll parameters of interest on the network and send them to a GUI that shows them distinctly and clearly. This GUI also should have E-stop capabilities. Polling should be done on a separate thread because the motion and network parameters should always be available for visual observation so that the user may terminate the process if something is not right.

- Motion thread: The thread that carries out the motion tasks. This thread has to be separate from the main thread because a motion automation task would execute in a loop. If the loop is not run on a parallel thread, then the user loses control over the Rig until the motion task is complete.

- Event thread: The thread that continuously collects the events taking place on the network. For the software, this thread communicates the status of various tasks and components on the machine such as "Velocity reached", "Motion Started", "Axis in Error" and "Recorder full". The notifications provided by this thread should be communicated to the respective threads and also to the master GUI thread.

- Recorder thread: The tread that looks out for the 'start record" event and once it is received, the recorder starts emptying the contents of the controller buffer into a text file, until the "stop recording" flag is received. This record remains in standby and autonomously triggers based on commands from the motion thread.

The control and communication flow is described in Figure 4-4.

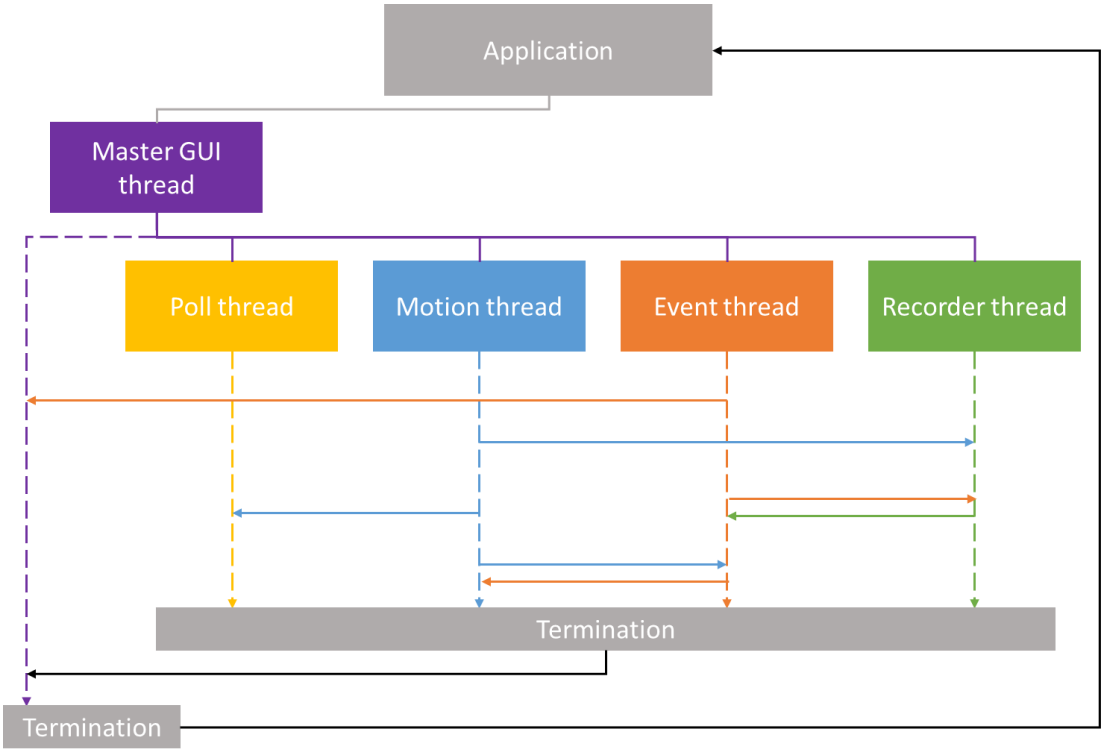


Figure 4-4: Application design: Control flow and communication

4.6 Contributions

Beyond automation: Software based co-simulation

Successful implementation of single axis automation has an impact that goes beyond automatic testing. There has been a long-standing expectation to develop the Rig into a co-simulation capable testing facility. In case of the Roller Rig, the idea can mean something like a co-simulation between Roller Rig and a simulation software like CONTACT. As is the case with any simulation, many parameters must be defined for a contact mechanics simulation. Some of these parameters are physics based while some are estimates. Verification of these simulation parameters can be done via simultaneous experimentation on the Roller Rig.

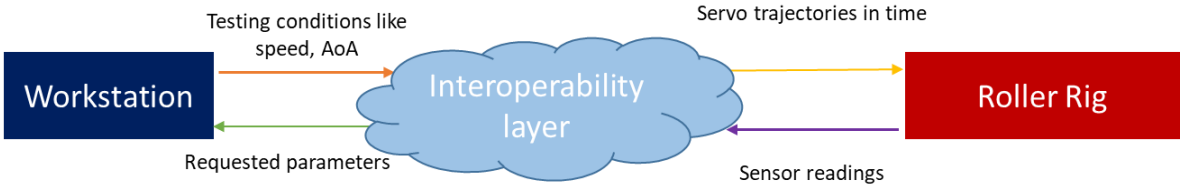


Figure 4-5: Co-simulation with the Roller Rig: A concept

The current testing protocol has a large manual involvement, that cannot provide continuous online co-simulation capabilities. The ability to control the Rig with minimal human input and purely based on an automatic software-hardware interaction program can become a game changer for railway contact testing. A suggested framework is shown in Figure 4-5.

Such a co-simulation environment would only require an interoperability layer between the program running on a workstation and the inputs that the MPI would be capable of understanding. Programming this interoperability layer would be too sophisticated and is not expected to be a simple task.

A simpler form of automation might be easily achievable while providing a high level of sophistication to the current testing process. A simulation can be run, for example in MATLAB. The code would generate a text file containing parameters for multi-axis motion depending on the testing conditions like AoA, Cant, etc. A shell script can be programmed and saved in the system, and MATLAB can call the shell script, which in turn invokes the executable file which controls the Rig and conducts the testing.

5. Conclusion

5.1 Project Summary

The work presented in this study has been an outcome of the commissioning exercises on the Virginia Tech – Federal Railroad Administration (VT-FRA) Roller Rig. There is now a deeper understanding about the components of the Rig, which has helped establish a proper testing and data analysis protocol for the Rig. Baseline testing has revealed the capabilities of the Rig, which have matched very well with the design objectives set in [1] and [7]. Some issues were also discovered and were resolved through engineering-based solutions.

The data acquisition system was thoroughly explored, both on a physical, fundamental level as well as practical application based level. Thorough investigation of the contact force-moment measurement system has obtained the ideal configuration for the measurement system settings. A systematic acquisition protocol and data conversion procedure is established and documented so that future experimenters would obtain accurate contact force-moment measurement data.

A significant part of this study was devoted to the development of the vertical force control system, which turned out to be a complicated problem. Vertical force is an important boundary condition for railway contact mechanics. It influences the contact patch dimensions and also the traction coefficients in the longitudinal and lateral directions. It was desired to reduce the force fluctuations to a level as low as possible. The solution procedure started with the root cause analysis of the phenomena. Simple physics based hypothesis was arrived at and. It was hypothesized that the force fluctuations were not due to any deficiencies in the Rig, but because of the naturally occurring minute variations that occur at the running surface of the roller and wheel. When combined with high stiffness caused by the position control, these minute variations would cause varying cycles of compressions which give rise to vertical force oscillations. Exhaustive data analysis and visualization was done to support this hypothesis.

Several engineering interventions were devised, and their merits and limitations were documented. The first idea was to use the passive mechanical springs and was based on the intuition of reducing the overall stiffness along the vertical degree of freedom. Unsatisfactory outcome of this method revealed a flaw in the understanding of the problem and it was ascertained that active methods must be used so that energy is injected into the system to make the cradle follow a highly non-linear non-deterministic motion trajectory. This led to the single loop feedback control method. However, the system identification exercise showed that the plant was too complicated to compensate. Such system characteristics were believed to be associated with the actuation of a large mass against gravity and inevitable unknown compliant elements. Another possible contribution to the undesired behavior could be an asynchronous behavior between two gantry actuators.

It was observed that the closed loop position feedback control system has a smooth and desirable frequency response with a high bandwidth, decent stability margins and a low resonant peak. This led to the idea of commanding a displacement profile to the existing position feedback control system which will cause the motion of the cradle. The displacement data was thought to be supplied in a feedforward manner, where the rotational angle of the wheel would be tied to a particular displacement value about a reference. The method was successful but the limitation was

that a feedforward based displacement command set was not practical for a dynamic testing setup like the Rig. So the cascaded loop force/position feedback control method was devised, where the displacement commands were now supplied in accordance with the force error seen be the control system. Open loop system identification revealed a very complicated plant, but it was more workable than the single loop compensation plant. With the addition of a proportional controller in series with a low pass filter, decent disturbance rejection has been achieved. The force control reduces the standard deviation of the vertical force fluctuations by a factor of four at every operating level when compared with position control. Hence, a better vertical force control has been achieved.

The study also explores the possibility of automating the tests on the Rig by means of a custom control software. The need for this was realized during the baseline tests on the Rig. The VT-FRA Roller Rig is capable of an unparalleled level of parameter control and a very high degree of experimental repeatability. To ensure that this is the case, a design of experiments was arrived at which controlled positions to within 1µm, angles to within 0.1^{o} and creepages to within 0.01%. The surface conditions were very carefully monitored and no contamination of any sort was allowed to happen. Such careful experiment design meant that a high degree of human involvement was required. The time and labour costs of testing were found to be high when the task at hand was repetitive. A high potential and desire for automation was realized.

This motivation led to the development of an automation software with an intuitive Graphical User Interface for easy communication with the Rig. The software is capable of executing multiple tasks in parallel which include continuous fault monitoring of the Rig, automatic motion trajectory execution and automatic data recording system which records the data based on motion events. The software has remained at a working, yet prototype stage because a definite and repetitive use case is required for a good automation code. However, the generic nature of the modular framework developed in this study should be capable of being scaled to multiple axes and any custom future use-case. The software code resulting from the automation study also represents the infantile developmental steps towards co-simulation or hardware in the loop functionalities. The Rig is uniquely positioned to accommodate such advanced features which, if incorporated would elevate the VT-FRA Roller Rig into a league of firsts.

The final objective of this study is to create a proper documentation for knowledge transfer. Substantial matter has been included in Appendices A and B. Appendix A provides a step-by-step case study based guide for designing custom control system for the Rig's hardware. Appendix B provides a detailed explanation of programming a software code that can issue motion commands to the Rig's motion axes, acquire data and events from the SynqNet network and automatically record data based on motion status of the actuators.

Appendix A: MechaWare™ and Controller Design

A.1 Introduction

MechaWare™ utility allows the user to design, implement and load custom control algorithms on the Roller Rig QMP-Synqnet controller card. It is a third party utility which runs on the MATLAB/Simulink environment which makes it easy for someone who is used to working in MATLAB/Simulink. This integration is also beneficial because of the rich and varied Control Systems Toolbox of MATLAB which has a lot of useful features and functions that are typically used for control systems" design. This chapter assumes a basic understanding of control system modeling in Simulink environment from the reader.

The control algorithm is designed by placing fundamental blocks and connecting them in a graphical block diagram fashion, exactly like the model building in Simulink. MechaWare™ environment provides with the following types of blocks:

Input-Output blocks: They act as placeholders to denote data inputs/outputs from/to the SynqNet network. Refer to Section A.3 for input blocks and Section A.4 for output blocks.

Computational blocks: They perform operations on the inputs and generate an output. They include filter blocks (Section A.6), Math blocks (Section A.5) and the Lookup block (Section A.6)

Logic blocks: They are used for if-else based tasks and enable case based automation. (Section A.8)

Conversion blocks: These convert data types to ensure compatibility during mathematical operations. These do not have to be always explicitly specified. For example, if a double and an integer are inputs to a sum operation block, the output will be a double. This implicit conversion is built into many programming languages by default. Mentioned in Section A.9

The block diagram models are loaded to the QMP-SynqNet card by using the model writer utility, named as "mdl2mw.exe". It is a command line program that reads a ".mdl" Simulink model, converts into a code that the controller can understand and then writes it on the controller card. The process is automatic with no additional objects required by the user. Details will be discussed in Section A.2.

However, it should be noted that although Simulink and MechaWare™ share the same working environment, the native Simulink blocks cannot interact with MechaWare™ blocks for controller implementation. When the model is loaded onto the controller, the model writer utility does not recognize native Simulink blocks. Thus the user should design the control algorithm only with MechaWare™ blocks.

Simple controller model design

This Section will provide the first look into building models with MechaWare™. Building MechaWare™ models is as easy as building Simulink models: drag and drop the components from the library browser and connect them.

Open the Simulink environment from MATLAB by typing in Simulink at command line and pressing enter.

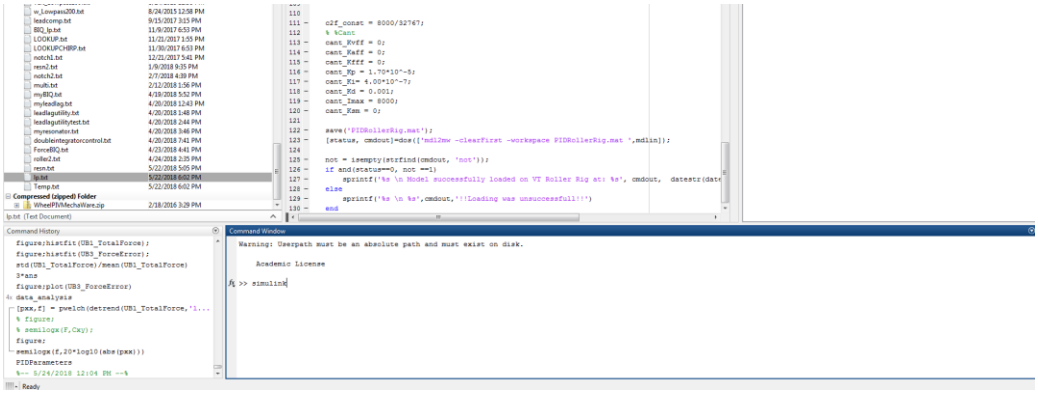


Figure 5-1 Opening Simulink

Open the Simulink library browser and expand the MechaWare™ blocks tree menu. Open a new model by clicking on the new model button.

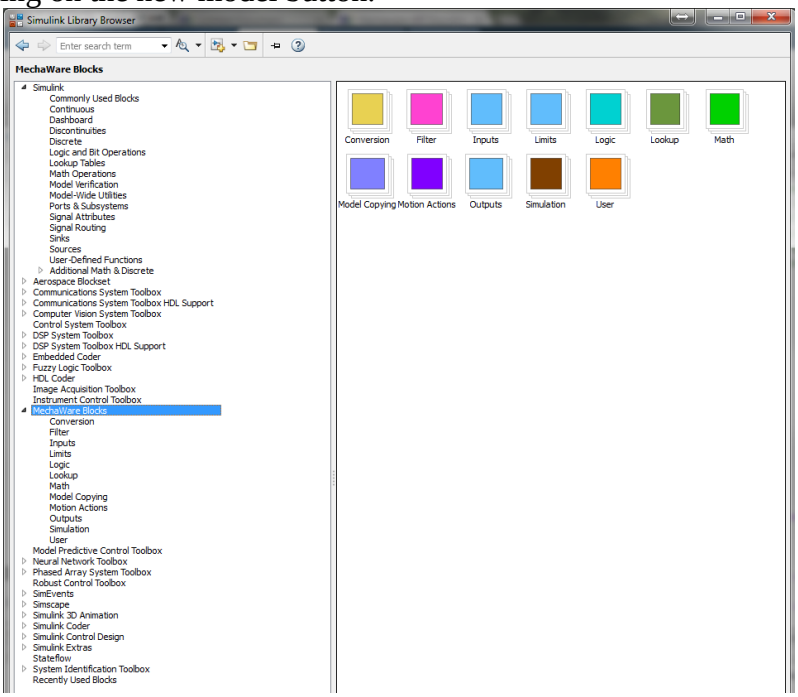


Figure 5-2: MechaWare™ blocks in Simulink

When designing a control system in MechaWare™, it is important to first decide what the variable of interest is and what the controller output is. This is because the controller in MechaWare™ does not look like a close loop as far as appearances go. At this step, decide the variable that has to be regulated, the *demand mode* in which the motor will be operated and the control algorithm that will be used.

Specify the control algorithm. We are trying out the PID error compensation algorithm so expand the MechaWare™ blocks tree item and then choose the filter sub-item as shown in the image. Drag and drop the PID with reset block from the library browser to the new model window which was just opened.

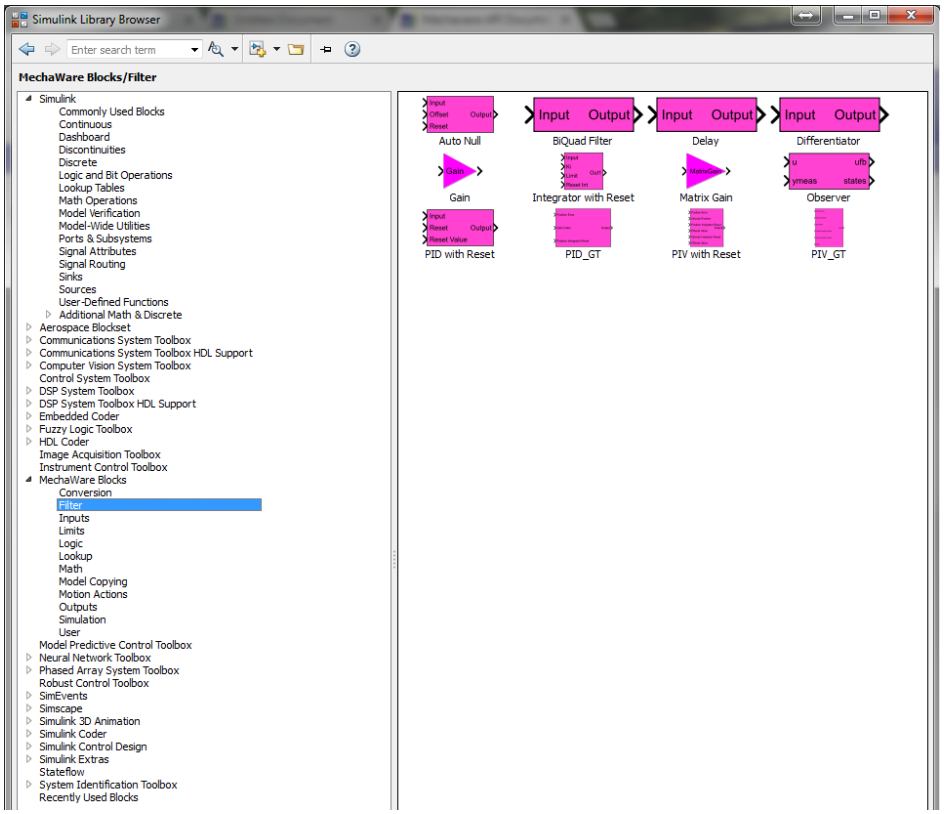


Figure 5-3: Filter blocks in MechaWare™

Specify inputs and output. Here inputs and outputs mean inputs from network and outputs to network. This may or may not be the inputs and outputs of the system but these definitely are important for control computation. We are building a position control loop. So we need a command position input, an actual position input and a motor output *demand block*, which for now we will take as the torque output block.

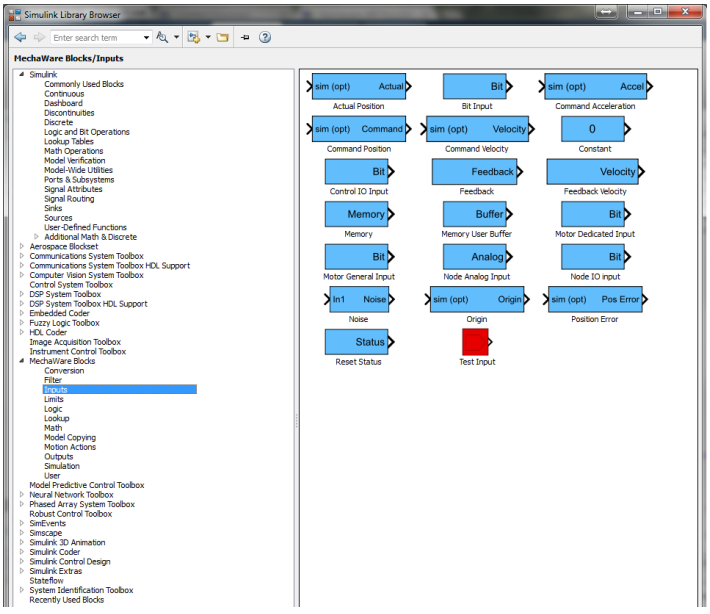


Figure 5-4: Input blocks in Simulink

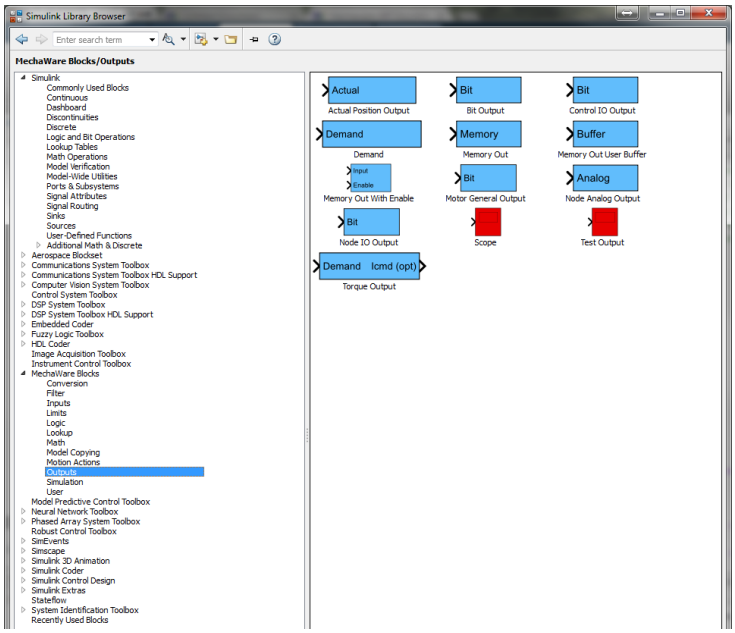


Figure 5-5: Output blocks in Simulink

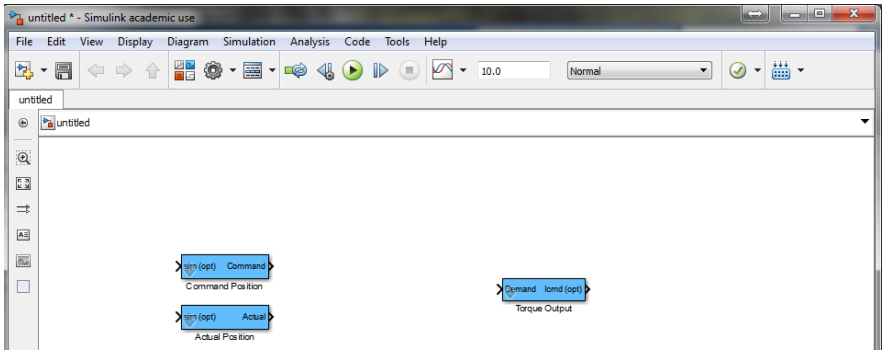


Figure 5-6: Inputs and outputs blocks arranged in the model

Double click on the command position block and specify the motor number. Do the same with the actual position block and the torque demand block.

PID servo position control works by comparing reference position input with actual position output, computing the error signal and feeding it to the PID block to evaluate the controller output. To calculate the error, we need a block which performs subtraction.

Go to the Simulink browser library and in the tree menu on the left hand side, go to MechaWare™ Blocks -> Math and drag-drop the 2 input sum block.

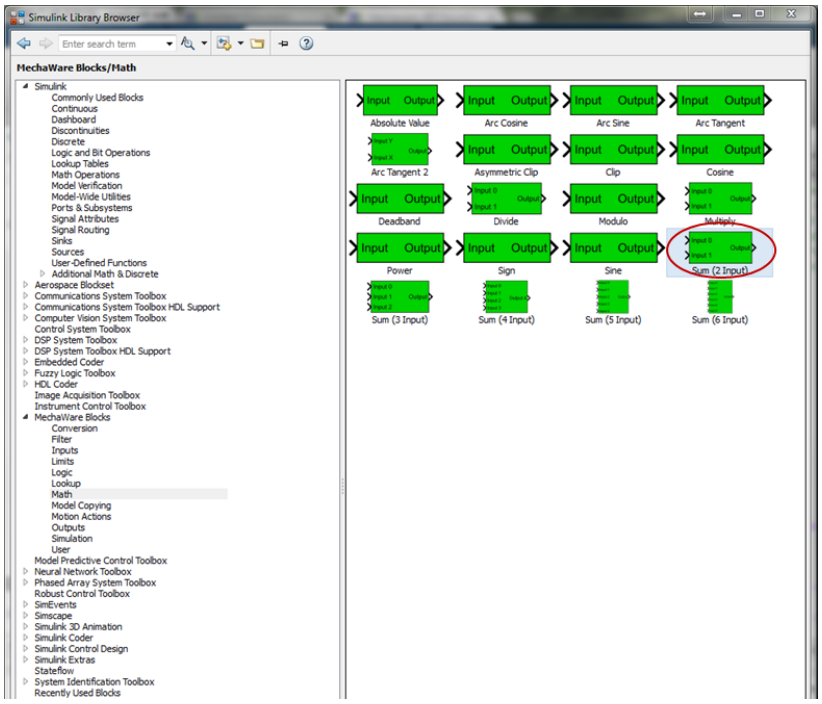


Figure 5-7: Math blocks in MechaWare™ with Sum block highlighted

When the sum block is dropped on the model window, double click the sum block and change the coefficient k2 to -1.

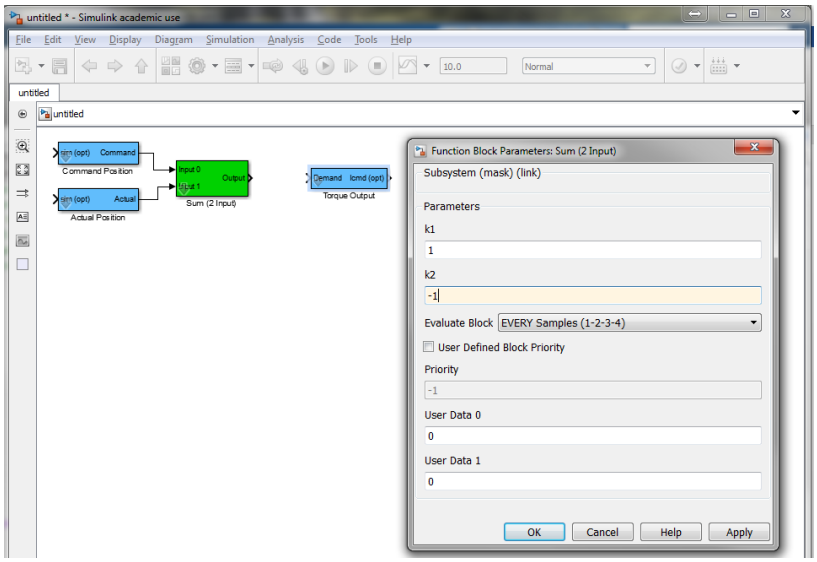


Figure 5-8: Sum block configuration to compute error

Connect the output of the command position block with the first input of the sum block modified above. Connect the output of the actual position block with the second input of the sum block modified above.

Connect the output of the sum block to the input of the PID block. The output is computed as $k1 * Command\ Position + k2 * Actual\ Position = Output$. If k1 = 1 and k2 = -1, we are basically computing the error signal. This error signal is fed into the PID block.

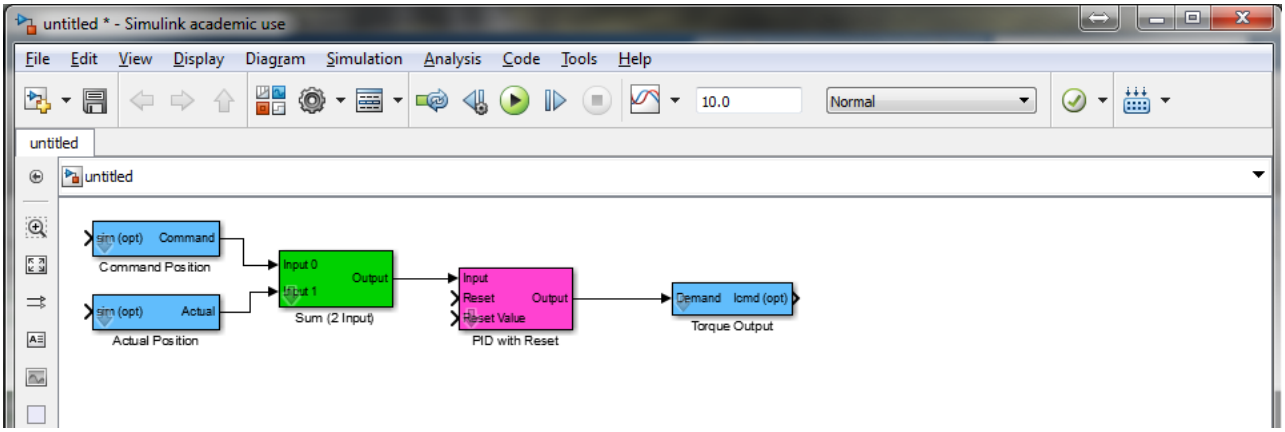


Figure 5-9: Computing position error and feeding to PID block for generating torque output compensation

Connect the output of the PID block to the input of the Torque output block.

There are two inputs that are not connected to anything. They are "Reset" and "Reset Value" block. For now, go to the library browser and then MechaWare™ blocks -> Inputs and choose the "Constant" block. (Remember, the blocks in the input are not just system inputs, they can be anything. But they definitely serve as an input to any general block).

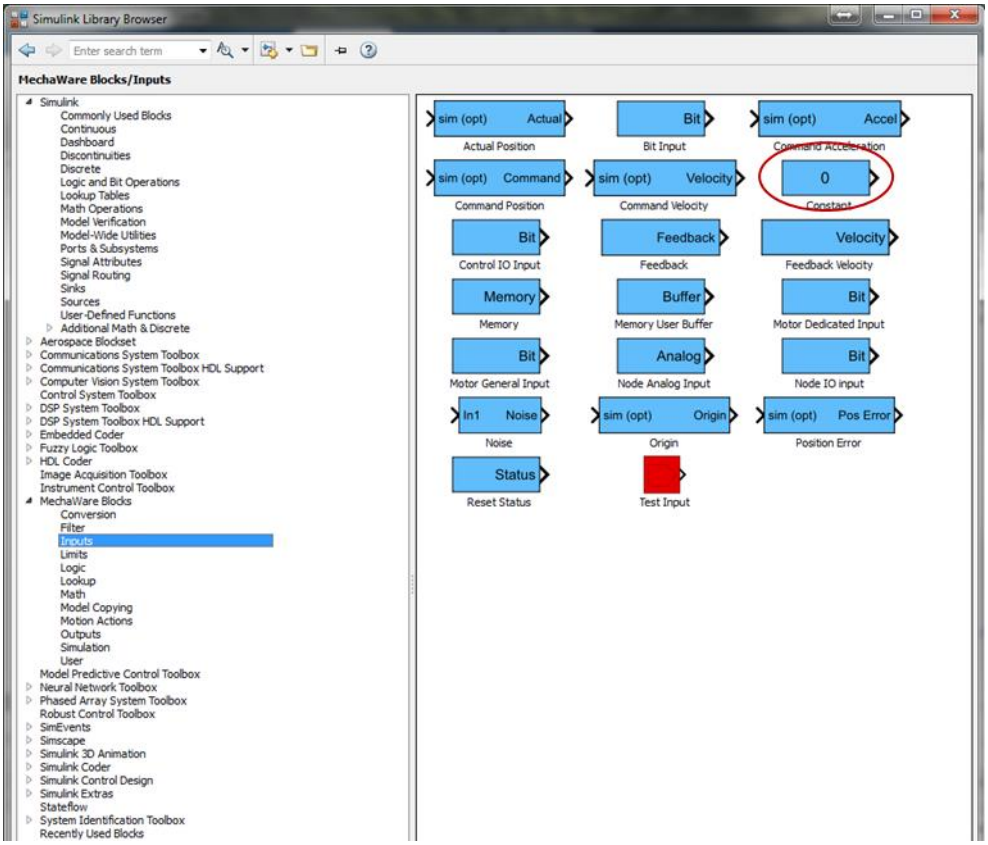


Figure 5-10: Constant block highlighted among MechaWare™ input blocks

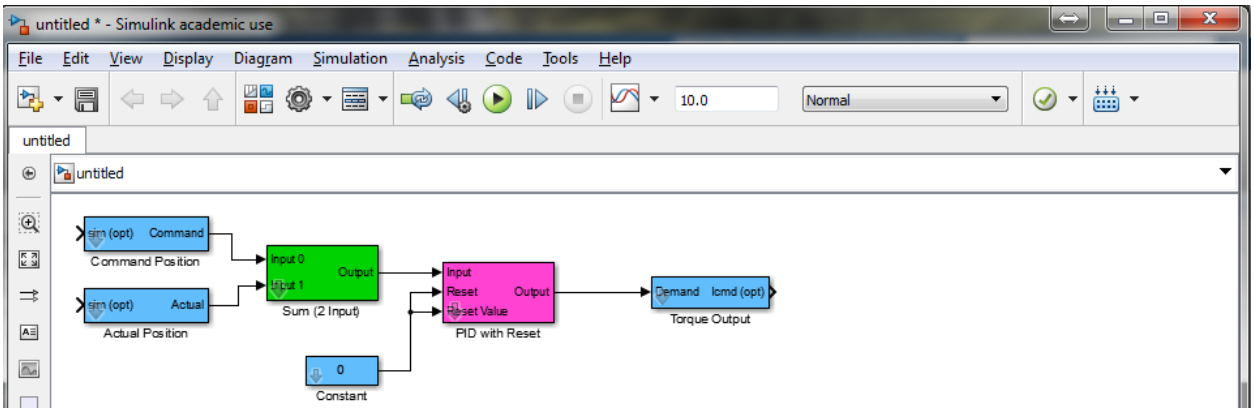
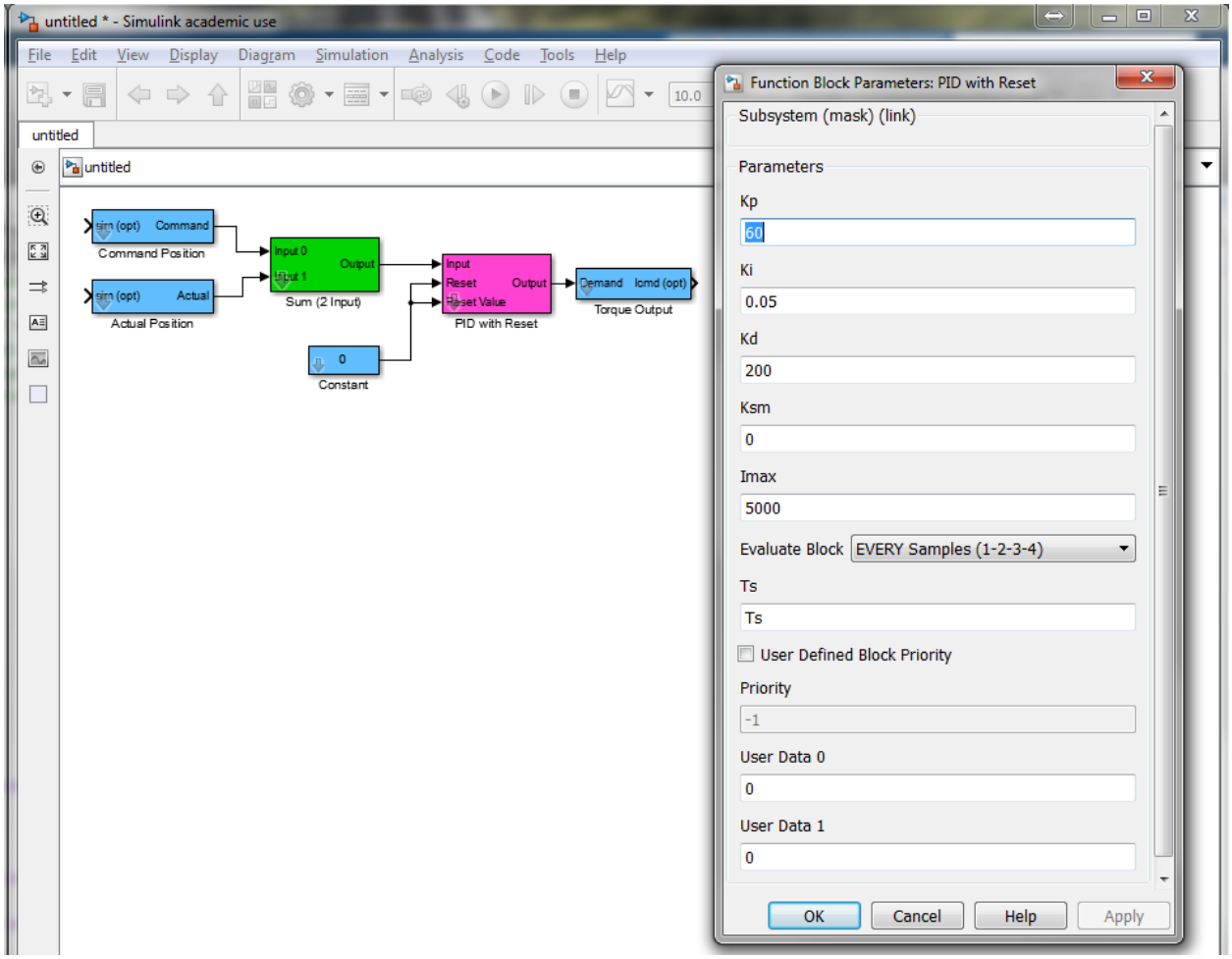


Figure 5-11: Completed position PID control loop

i) Double click the constant block and set the constant value to numeric zero for now. Connect the output of this block to the Reset input and Reset Value input of the PID block. These will be explained in the filter Section.

ii) You have set up the PID position control for the motor number which corresponds to the torque demand block. Now double click on the PID block and set the proportional gain, integral gain and derivative gains, along with sample time, Max. integrator value (Imax) and other details about the controller. These cannot be ad hoc and have to be set after careful analysis and tuning procedures.

iii) 

iv) This concludes the setup in Simulink/MechaWare™.

This model now has to be written on the controller card and has to be tuned/analyzed by measuring time or frequency domain based methods. The following sections will address that followed by a detailed block design documentation.

A.2 Model writer utility (mdl2mw.exe)

Model writer utility is provided by Kollmorgen to load MechaWare™ models onto the QMP-SynqNet controller card. It is an executable named mdl2mw.exe and can only be launched in the command prompt environment. Note that the MEI folder is already added as a PATH environment variable in the system so Windows knows where to look for the utility file mdl2mw.exe. As a result, the utility can be called from any directory. The utility is command line based and to execute it, the user has to pass command line arguments [16]. The following table indicates the usage for the utility. The arguments pertinent to the Rig are shaded in light red.

A.2.1 mdl2mw.exe usage

mdl2mw [-control #] [-server #] [-port #] [-trace #] [-save #] [-model #] [-help] [-workspace #] [-suppressModelError] [-clearFirst] [-quiet] model_name	
-control	Controller number (default = 0)

-server #	Name or IP address of the host running server.exe (Not important for Rig)
-port #	TCP/IP port on the host computer (default = 3300)
-trace #	Bit mask to specify trace information outputs
-save #	Store the output as an intermediate (.bin) file instead of loading the controller
-model	Set the model number to be loaded
-help	Use only this argument to load the usage help in command line.
-workspace	Specify MATLAB workspace file (.mat file) to read variables from
-suppressModelError	Do not generate error if a non-existent model is specified in the .mdl file
-clearFirst	Clear target's (controller card's) model buffer before loading model
-quiet	Don't print informational messages on console (NOT recommended)
Model_name	The full name of the MechaWare™ model file, including the .mdl extension and with no "-" prefix.

A.2.2 Usage example with hard-coded values

If all the values like gains, constants, etc. are numerically specified in the MechaWare™ model, then follow these steps:

i) Once the model is built in Simulink/MechaWare™, save it as a .mdl file. This is important. The utility does not recognize the newer .slx Simulink file format. Let's call it "example.mdl"

ii) Open command line and change the current directory to the folder where the file "example.mdl" is saved

iii) Enter `mdl2mw.exe -clearFirst example.mdl` and hit enter. This clears the existing model on the controller card and then uploads the example.mdl model.

iv) Observe the output. It should be a description of all the blocks present in the model with their name and type mentioned. Scroll over the entire output to make sure that all the blocks in the model are mentioned and there aren't any fail messages.

A.2.3 Usage example with MATLAB script [Recommended]

The user can choose to not hard code the values in MechaWare™ model parameters but instead can specify MATLAB workspace variables. This feature is very convenient for parameters like gains which have to be iteratively changed until satisfactory combination is achieved. Changing hard-coded parameters is time consuming compared to changing parameters in a script. For example in our "example.mdl" file, specify the gains in the PID block as Kp, Ki, Kd, sample time as Ts, Integrator max value as Imax. Write the following script in MATLAB.

When this script is run, the model will be uploaded on the controller card. The first five lines define these variables in workspace. The names of these variable should be the same as the names specified in the respective parameter fields in Simulink/MechaWare™ model. Line 6 saves these parameters in example.mat file. The next line calls the dos function which is used to execute dos commands from MATLAB script. The -workspace flag specifies that the utility should reference the .mat file which succeeds the flag for any non-numeric parameter values in the model. The dos command returns two outputs: 'status" which has a value of 0 if the dos command executed successfully and a non-zero value if the command entered was invalid. The "cmdout" variable contains the output returned by the mdl2mw utility. Specifying the –quiet flag while execution will not populate cmdout and is not recommended. The "cmdout" variable actually would tell us if the model was successfully uploaded or not. To check that programmatically, line 8 is used. Strfind command finds the occurrence of a substring ("not" in this case) in a bigger string ("cmdout" in this case). If cmdout variable contains "not" anywhere in it, the isempty function would return FALSE (0), making not = 0. IF not == 0, then model upload was not successful and the MATLAB command window will say "!!Loading was unsuccessful!!".

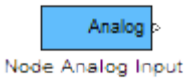
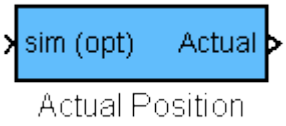
```
Kp = 1; % proportional gain
Ki = 0.01; %integral gain
Kd = 0; %derivative gain
Ts = 1/2000; %sample time
Imax = 5000; %max absolute output of integrator

save('example.mat');
[status, cmdout]=dos('mdl2mw -clearFirst -workspace example.mat
example.mdl');

not = isempty(strfind(cmdout, 'not'));
if and(status==0, not ==1)
    sprintf('%s \n Model successfully loaded on VT Roller Rig at: %s',
cmdout,  datestr(datetime))
else
    sprintf('%s \n %s',cmdout,'!!Loading was unsuccessfull!!')
end
```

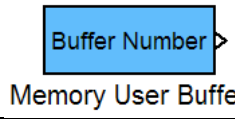
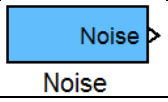
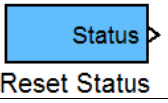
Using the command line utility alone does not specify an easy to comprehend message like this. These MATLAB commands are also useful in indicating whether any port of any block is left unconnected. The model writer utility will NOT throw an exception and post a failure message when there are unconnected ports or blocks in it. But in the utility output (cmdout variable), during the block description, the utility will indicate that the input or output of a particular block is "not connected". This not will be picked up by the strfind function and the user will see an unsuccessful load message in the end.

So, when the user sees the message "!!Loading was unsuccessful!!" in MATLAB command window, the user is advised to revisit the model and make sure every input/output port of every block is connected to something and there are no errors in modeling. It is however a good practice to check the entire output and not just the final success message.

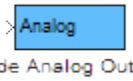
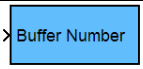
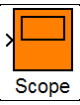
A.3 Input blocks

Input blocks are placeholders that instruct the controller card to fetch values from the SynqNet network to provide inputs to other MechaWare™ blocks. For example, consider an "Actual Position" block. This block takes in no inputs and outputs the actual position feedback of a servo motor. This output can be connected to the input of another block to make that block receive the actual position feedback of a servo.

Block Name	Image
Analog Input	
Provides the digitally converted value for the analog voltage measured on the input pin specified. Roller Rig SQIO can measure analog input voltages of +/- 10 V and it undergoes a 16-bit ADC converstion. Thus +/- 10V is converted to a digital value between -32767 to +32766 counts. This input is important for reading values from sensors interfaced with the SynqNet. Parameters: • Node Number: The SqNode number where the analog input ius located • Channel: The channel number which is being measured	
Actual Position	
Actual Position block gives the actual position feedback data. The actual position is accessed from the axis module of MPI. Parameters: Axis number which is being referred to.	
Command Position	
Command position block outputs the position commanded to the axis module. The commanded position value is generated by the motion supervisor and passed on to the axis module. Parameters: Axis number which is being referred to.	
Command Acceleration	
Provides command acceleration data. The command acceleration to a motor is generated by the motion supervisor which acts as the trajectory generator for the axis under it. The input is optional and is not considered when the model is downloaded to the controller Parameters: • Axis Number: The axis whose command acceleration is required.	

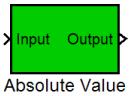
Command Position	
Provides command position data. The command position is generated by the motion supervisor which acts as the trajectory generator for the axis under it. The input is optional and is not considered when the model is downloaded to the controller. Parameters: • Axis Number: The axis whose command position is required.	
Command Velocity	
Provides command velocity data. The command velocity to a motor is generated by the motion supervisor which acts as the trajectory generator for the axis under it. The input is optional and is not considered when the model is downloaded to the controller Parameters: Axis Number: The axis whose command velocity is required.	
Memory User Buffer	
Provides data from one of the memory user buffers. User buffers are memory locations provided for custom usage by the programmer. These user buffers provide a great deal of convenience to communicate data and signals between different blocks or to the application. There are a total of 1024 user buffers available. Parameters: • Memory user buffer number: Defines the user buffer register which will be accessed • Data Type: The type of data which is contained by the above register	
Noise Generator	
Provies access to the QMP-SynqNet controller's internally generated random noise source. This block is a placeholder used by the bode tool to insert noise into the system.	
Reset Status	
Used to reset the value of an integrator based on whether the axis is enabled or disabled. Provides a convenient method to prevent integrator saturation when the axis is disabled. Parameter: • Axis Number: The axis which is being monitored for the "DISABLED" flag • Enable: If enable is 0, the output is always zero. If enable is 1, then the output is zero when the axis is ENABLED. And the output is non-zero when axis is DISABLED.	

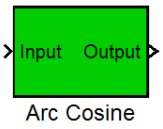
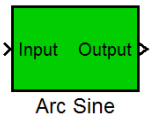
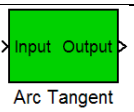
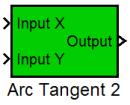
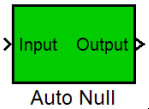
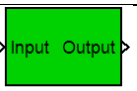
A.4 Output blocks

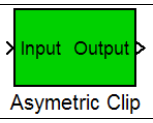
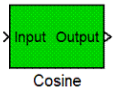
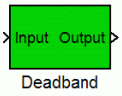
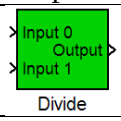
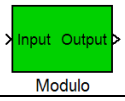
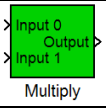
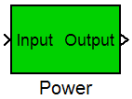
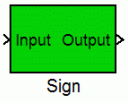
Name	Image
Analog Output	
Can be used to write analog values to outputs on the analog SqNode I/O. Roller Rig SQIO can write analog input voltages of +/- 10 V after it undergoes a 16-bit DAC converstion. Thus to write +/- 10V, this block should write a value between -32767 to +32766 counts. Parameters: • Node number: The SqNode number where the analog output is being written • Channel: The analog channel number on the sqnode where the output is being written	
Memory Out User Buffer	
Provides a way to write custom data into a memory location on the SynqNet. User buffers are memory locations provided for custom usage by the programmer. These user buffers provide a great deal of convenience to communicate data and signals between different blocks or to the application. There are a total of 1024 user buffers available. Parameters: • Memory user buffer number: Defines the user buffer register where the data will be written • Data Type: The type of data which is contained by the above register	
Demand	
Scope	
An alternative way to write data to a user buffer. A scope block automatically obtains the buffer where the data is going to be written. For example the first scope block in model will write data to User Buffer 0. The second scope block will write to User Buffer 1 and so on. The data type by default is set to double.	
Torque Output	
Defines the motor which will receive torque commands. Roller Rig MechaWare™ provides two demand modes: Velocity and Torque Demand Modes. Parameters: Motor Number, Sample Time	

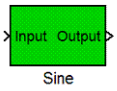
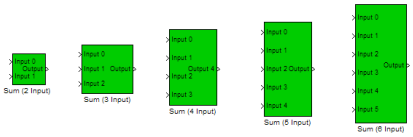
A.5 Math Blocks

It is important to understand the types and functions of the math blocks in order to harness the full potential of MechaWare™. The library provides with a plethora of math functions which cover almost every use-case imaginable.

Block Name	Image
Absolute Value	
Outputs the absolute value of the input. Works with all data types, fractional parts are preserved	

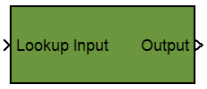
Arc Cosine	
Computes inverse cosine of the input value in Radians. Input should be a double precision floating point value and the output is a double precision floating point value in radians.	
Arc Sine	
Computes the inverse sine of the input value in Radians. Input should be a double precision floating point value and the output is a double precision floating point value in radians.	
Arc Tangent	
Computes the inverse tangent of the input value in Radians. Input should be a double precision floating point value and the output is a double precision floating point value in radians.	
Arc Tangent 2	
Computes inverse tangent of the input values as $\theta = tan^{-1}(\frac{x}{y})$. This block is useful for handling the divide be zero cases ($\theta = 90°$) where the standard block mentioned above cannot work.	
Auto Null	
Auto Null block increments the output value whenever the input > 0.0 When the input < 0.0 then the output is decremented by the specified value. Physically the auto null block acts as an integrator with a constant derivative (linear relationship between output and input). This block can be useful to counter the drift present in analog sensors. Parameters: • Increment: The specified increment which will be added to or subtracted from input	
Clip (Symmetric)	
Keeps the output within +/- limit as specified in the limit parameter. For example, if the limit is specified as 5000, then the output will not exceed +5000 or go lower than -5000. This block is useful for limiting values in systems with a possibility of windup. For example, the actuators in the Rig cannot be commanded a torque that is more than 32767 counts. In presence of large errors, a PID controller can easily output values greater than 32767 or less than -32768, which could be potentially damaging to the actuators. Its recommended to add a clip block before every demand block in the model. Parameters: • Limit: Max of absolute(output). Example: Limit = 5000 means output within +/- 5000 • Offset: the constant DC value which can be added to the input signal. Useful for scenarios like the vertical degree of freedom where a constant torque value has to be supplied in order to prevent the carriage from falling when the input to clip block is zero.	

Clip 2 (Asymmetric)	
Asymmetric clip also clips the output between limits, but the lower and upper limits can be individually set. Parameters: • Upper limit: Max possible output value • Lower limit: Minimum possible output vale • Offset: Constant value that is added to the input	
Cosine	
Outputs cosine value of the input. The input should be specified in radians.	
Deadband	
Generates a zero output when input is within a certain range. This is important for representing phenomena where the output is zero for a range of inputs, like the Roller Rig vertical degree of freedom, where the force output is zero for a range of vertical position coordinates. Parameters: • Start of dead zone: Inputs below this level generate non zero output • End of dead zone: Inputs above this level generate non zero output	
Divide	
Performs $Output = \frac{Input0}{Input1}$. Both inputs should be double precision values	
Modulo	
Computes whole number remainder of the division of the input by the divisor which is the input parameter.	
Multiply	
Performs $Output = Input0 * Input\ 1$. Inputs should be double precision values	
Power	
Output is equal to the input raised by the exponent which is the input parameter of this block	
Sign	
Signum function: Output = -1 for input < 0, output = 0 for input =0, output = 1 for input > 0. Important block, can be used in places like feedforward compensation where the feedforward push should act in the same direction as the direction of motion.	

Sine	
Computes sine value of the input. The input should be a double precision value in radians	
Sum	
Computes the summation between the input values multiplied by their gains which are parameter inputs.	

A.6 Lookup blocks

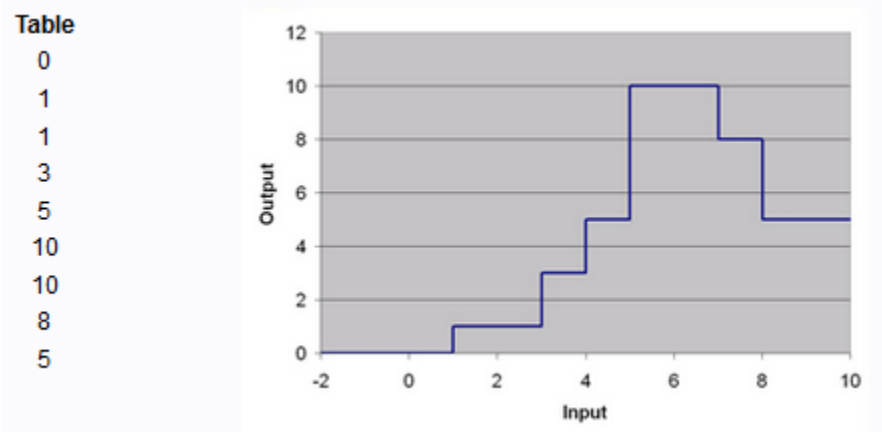
Lookup blocks provide a means to implement lookup tables. Lookup tables are an important feature and might be required for applications such as gain scheduling where the gains of the feedback or feedforward controller depend on the input or some other property of the system. Lookup tables can also serve as a much faster alternative to computationally expensive tasks like sine or cosine computation. MechaWare™ provides with two types of lookup blocks: one dimensional (1-D) and two dimensional (2-D).

Block name	Image
Lookup Table [17]	

The output can be connected to any block except the block "floating to integer conversion" as the output is of double precision. Performs a step or piecewise linear mapping of the input. There are three types of operations in the lookup block:

- Direct (Type 0)
- Fixed form (Type 1)
- Vector (Type 2)

Direct form uses the value of input to produce the output value. For example, consider the following table:



If input value is 0, the output is 0. If input value is 1, output is 1. If input value is 4, the output is 5. The value of the input serves as the row number and the value of the number contained in that row becomes the output. When inputs go out of bounds, the end values are

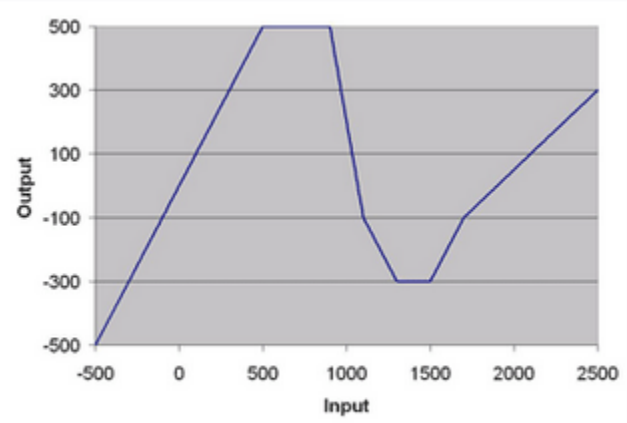
retained. For example, for all inputs < 0, output would be 0. For all inputs > 8, the output would be 5.

Fixed form (Type 1) requires two parameters to be defined: Input minimum value and the input step size. The output value is determined by the "index value" of the input.

$$Index\ value = floor(\frac{Input\ value - minimum\ value}{step\ size})$$

Floor means to round towards the lower integer. The output value is the value contained in the index-th row value. Out of bounds inputs are linearly interpolated. For example:

Table	Comment
100	// Input Minimum Value
200	// Input Step Size
100	// Table Entry 0
300	
500	
500	
500	
-100	
-300	
-300	
-100	
0	



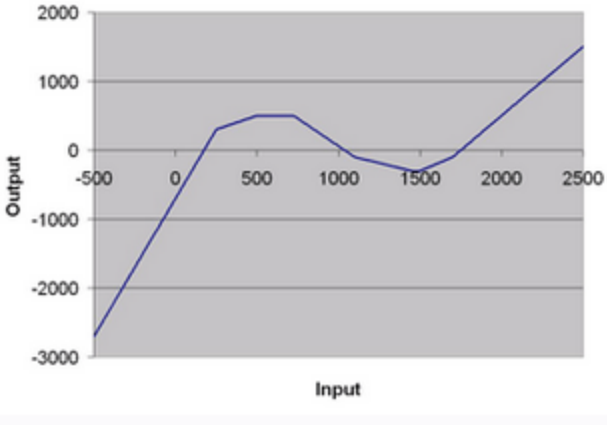
If input = 750, then the index value is: 3 ⇒ Output = 500, as seen in the figure. If input = -500 (less than input minimum value) then index value = -3. Output would be the same as if the table extended to negative infinity with 3 steps away, i.e. Output = (100 – 3*200 = -500).

Vector form (Type 2) is perhaps the most important type of lookup table. Output is determined by two tab separated columns. The input vector is specified as the first column in the table and the second column is the output value. The input vector must be in the ascending order. Here the closest two indices in the lookup table are determined as:

$$vector[n] \leq input < vector[n+1], then\ n^{th}\ and\ (n+1)^{th}\ indices\ are\ closest$$

The output value is linearly interpolated from these two values. For example:

Input Vector	Table
200	100
250	300
500	500
700	500
725	500
1100	-100
1450	-300
1500	-300
1700	-100
1750	0

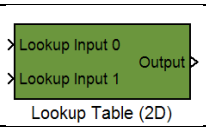


If the input = 275, then the nearest two indices are 1 and 2 (indices start from 0). The output is linearly interpolated and calculated to be 320. If the input is out of bounds, then output is

extrapolated. If input = 2000, then previous two indices are 8 and 9. The output is extrapolated linearly based on these two points and computed to be 500.

All lookup tables must be saved in a text file. The file name is one of the parameters. The other parameter is the Lookup type which have been described above.

Lookup Table (2D) [18]



This block takes two parameters, the lookup type (method) which can be either Direct, Fixed form or Vector form. These are the same as Lookup (1-D). The difference here is that the index is two dimensional, i.e., (m, n). So, the output value is the value in the m^{th} row and n^{th} column of the matrix table. The index value is defined as:

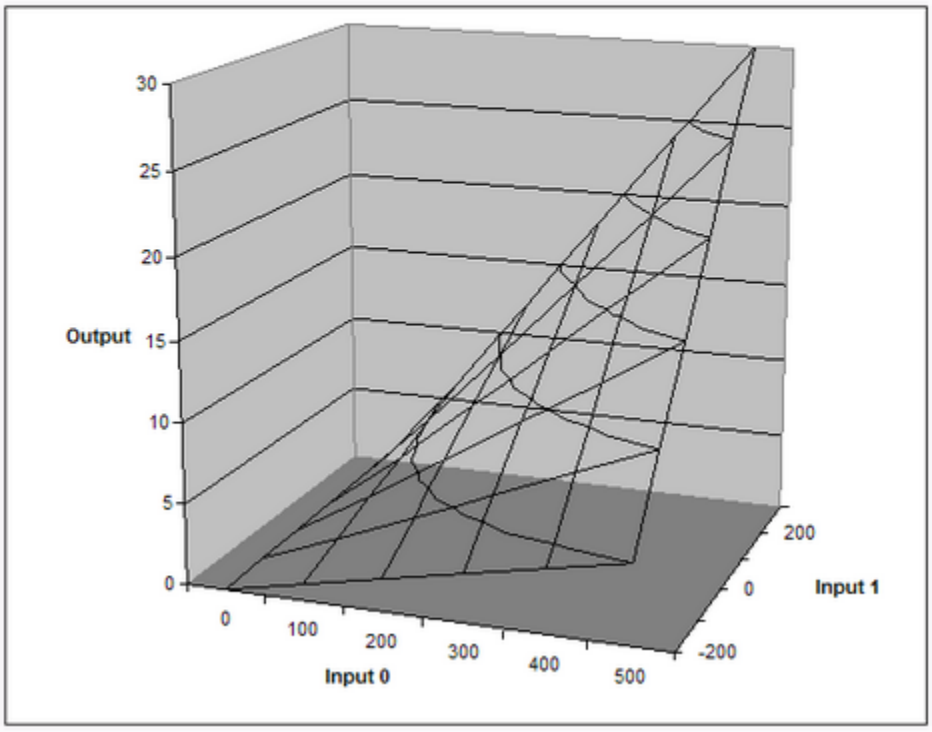
$$Input0[m] \leq Lookup\ input\ 0\ value < Input0[m+1]$$
$$Input1[n] \leq Lookup\ input\ 1\ value < Input1[n+1]$$

For example, consider the Vector form 2-D lookup. The table is written in the text file with the first space delimited line being input vector 0 and the second space delimited file being input vector 1. Then subsequent rows are the rows of the matrix with dimensions m, n.

```
0 100 200 300 400 500 Input Vector 0

-200 -100 0 100 200 300 Input Vector 1

0.0 1 2 3 4 5   Comment may start here

0.1 2 4 6 8 10  Comment must not begin with a number

0.2 3 6 9 12 15  This table has 36 entries

0.3 4 8 12 16 20

0.4 5 10 15 20 25

0.5 6 12 18 24 30
```

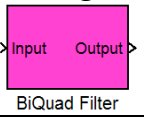
This can be visualized as:

Input Vector0	Input Vector1	Table					
0	-200	0	1	2	3	4	5
100	-100	0.1	2	4	6	8	10
200	0	0.2	3	6	9	12	15
300	100	0.3	4	8	12	16	20
400	200	0.4	5	10	15	20	25
500	300	0.5	6	12	18	24	30



If the lookup input 0 = 300 and lookup input 1 = -200, then m = 3, n = 0 ⇒ Output = 3. For intermediate inputs, the values are linearly interpolated in two dimensions.

A.7 Filter blocks

Block name	Image
Biquad Filter	

Provides the means to implement a custom transfer function. Biquad is short for biquadratic function which means a rational fraction having a second order polynomial as the numerator and the denominator. Mathematically, a biquadratic filter transfer function is represented as:

$$BIQ = \frac{B0 + B1 * z^{-1} + B2 * z^{-2}}{1 + A1 * z^{-1} + A2 * z^{-2}}$$

MechaWare™ implements as the following block diagram:

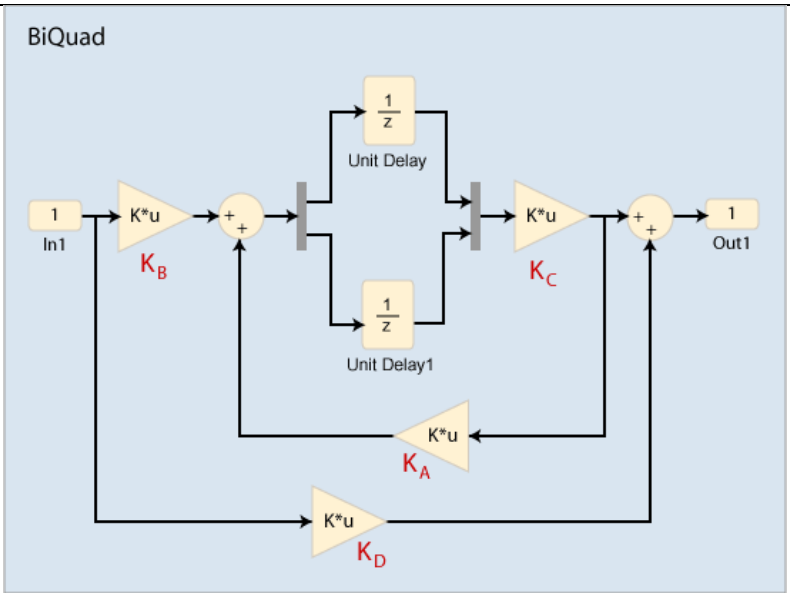


Figure 5-12: MechaWare™ implementation of Biquad filter

Parameters:

- Coefficient file: The coefficients of the Biquad filter are specified in a text file. Enter that file name without the .txt extension.

  The coefficient text file looks like this:

```
Coeff[0]  =  9.98E-01
Coeff[1]  =  2.00E+00
Coeff[2]  =  9.98E-01
Coeff[3]  =  1.9954991
Coeff[4]  =  -0.995509
```

The way to read these coefficients is as follows:

Coeff[0]	B0
Coeff[1]	B1
Coeff[2]	B2
Coeff[3]	-A1 (enter negated value)
Coeff[4]	-A2 (enter negated value)

For transfer functions with an order higher than Biquad, this Biquad block provides a way to implement them. The higher order transfer function is factorized into second order sections or biquadratic transfer functions. These second order sections are implemented as:

Figure 5-13: Cascaded Biquad filter implementation in Biquad block

The coefficients are written as:

Data Format for Multistage biquad			
Section 1	Coeff[0]	=	B0
	Coeff[1]	=	B1
	Coeff[2]	=	B2
	Coeff[3]	=	-A1
	Coeff[4]	=	-A2
Section 2	Coeff[5]	=	B0
	Coeff[6]	=	B1
	Coeff[7]	=	B2
	Coeff[8]	=	-A1
	Coeff[9]	=	-A2
Section 3	Coeff[10]	=	B0
	Coeff[11]	=	B1
	Coeff[12]	=	B2
	Coeff[13]	=	-A1
	Coeff[14]	=	-A2
Section 4	Coeff[15]	=	B0
	Coeff[16]	=	B1
	Coeff[17]	=	B2
	Coeff[18]	=	-A1
	Coeff[19]	=	-A2

These coefficients can be generated manually or by using provided MATLAB utilities.

Using the in-built MATLAB utilities

MEI/Kollmorgen provides in-built utilities for generating Biquad filters [19]. These are:

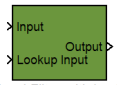
- Lowpass : Generates second order Butterworth lowpass filter
- MultiStageBIQ: Implements a higher order multi-stage cascaded Biquad filter
- MWOBS: Implements an observer block
- Notch: Implements a 2^{nd} order notch filter to filter out a range of frequencies
- PID2BIQ: Implements a PID block as a biquadratic transfer function
- Resonator: 2^{nd} order transfer function that has a resonant peak/valley at center frequency
- Unity gain: Implements unity gain filter in a Biquad block
- ZP2BIQ: Implements a second order filter with 2 poles, 2 zeros and a DC gain
- Leadlag: Implements a lead lag compensator

The user is encouraged to look up the associated reference for self-explanatory documentation.

Manual entry method is not recommended since higher order compensators can be implemented by the MultiStageBIQ utility function

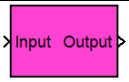
Do NOT implement higher order compensators by stacking more than one Biquad filters on top of each other. Use only a single Biquad block and implement the MultiStageBIQ function.

BiQuad filter with lookup coefficients



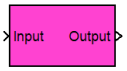
Implements a Biquad filter with coefficients that change based on the lookup input.

Delay

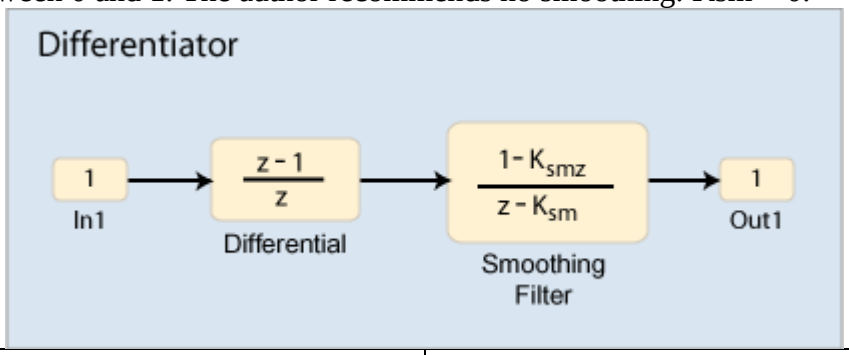


Implements a time delay (z^{-n}) of n samples where n is the input parameter delay in samples. The delay can be up to Maximum delay samples.

Differentiator



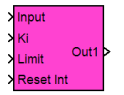
Outputs the derivative of the input signal. The input parameter is Ksm: Smoothing parameter. Ksm stays between 0 and 1. The author recommends no smoothing: Ksm = 0.



Constant gain

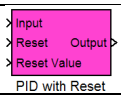
Outputs the input value multiplied by a constant gain K which is the input parameter.

Integrator with Reset



Outputs a time series integration of the input, multiplied by the value provided in Ki. Limit value keeps the value of output within +/- Limit. Reset int port is responsible for resetting the integrator. If a non-zero value is observed at this port, then the output will be reset to zero.

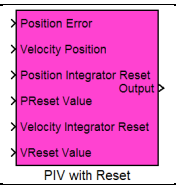
PID with Reset



A combined PID block which takes an error input (position error, force error, etc) and outputs compensation commands to the actuator based on the demand mode (velocity or torque).
A non-zero input to the Reset port resets the integrator output to a value provided to the reset value port. Parameters:

- Kp: Proportional Gain
- Ki: Integrator Gain
- Kd: Derivative Gain
- Ksm: Smoothing factor for derivative (recommended value: 0)
- Imax: The clipping limit of the integrator output, i.e. integrator saturation value.

- Ts: Sample time

PIV with Reset	

PIV is a cascaded loop architecture which has a position loop as the outer loop and the velocity loop as the inner loop. A simple block diagram representation is shown below:

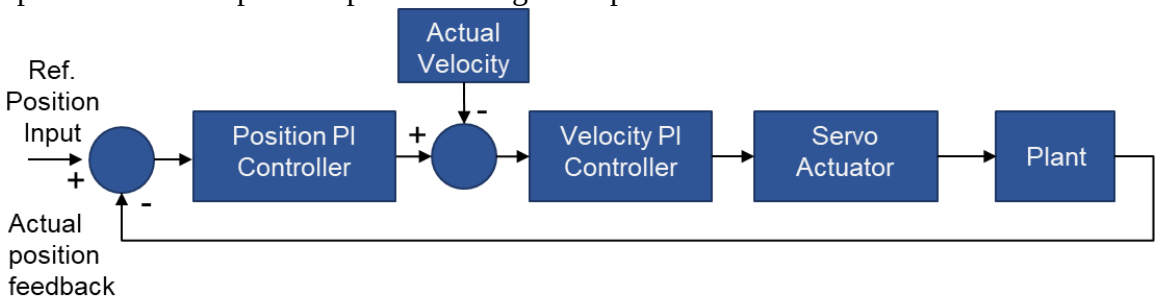


Figure 5-14: PIV control explained

The output must be connected to a torque demand mode since the velocity loop is already included in the block.

Inputs:

- Position Error: Connect position error to this input
- Velocity Position: Connect Actual Position Feedback block to this input
- Position Integrator reset: Reset the position integrator. Non-zero value = reset
- P-Reset Value: the value to which the position loop integrator will be reset to.
- Velocity Integrator reset: Reset the velocity loop integrator. Non-zero value = reset
- V-Reset value: The value to which the velocity loop integrator will be reset to.

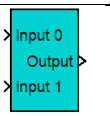
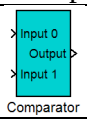
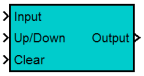
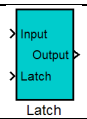
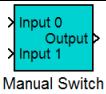
Parameters:

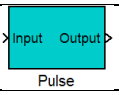
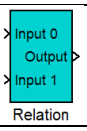
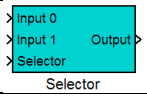
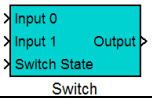
- Kpp: Position loop proportional gain
- Kip: Position loop integrator gain
- Kpv: Velocity loop proportional gain
- Kiv: Velocity loop integrator gain
- Ksm: derivative smoothing factor (A differentiator acts on actual position value to obtain actual velocity value)
- Imaxp: Position integrator saturation value
- Imaxv: Velocity integrator saturation value

A.8 Logic blocks

Perform comparison, logical operations for implementing digital control and automation tasks.

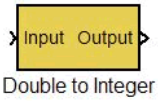
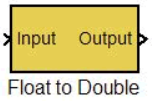
Block name	Image

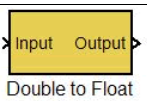
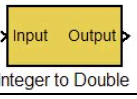
Binary Relation [20]	
Performs binary logical operations like AND, OR, NOT, NAND, XOR, NOR on the inputs. The inputs are cast into 32-bit unsigned integers and then the logical operations are performed. It is recommended that the inputs be explicitly kept as 32-bit unsigned integers to avoid typecasting errors. Parameters: • Operation: the type of operation that has to be performed on the inputs	
Comparator [21]	
Compares input 0 versus input 1. If Input0 > Input1 then the output is 1 (TRUE) else if input0 <= input1, then the output is 0 (FALSE). Both inputs should be double precision floating point numbers.	
Counter [22]	
This block can be used to count the number of state transitions for a digital input. A digital signal has two levels: 0 (Low/False) and 1 (High/True). Counter block monitors each state transition and adds/subtracts it from the previous output value. Example: A simple encoder can be attached to a digital input of our SQIO mixed-module and this block can be used to count the number of pulses that the encoder has put out. Of course modern day encoders do the counting internally and such a use-case is unlikely. Input ports: • Input: The digital signal which is being monitored for state transitions • Up/Down: An input of 0 to this port increments the output value for each state transition. A non-zero input to this port would decrement the output value • Clear: A non-zero value to this port will reset and hold the counter at zero.	
Latch [23]	
An important block for implementing automation features, latch in plain English means a device that holds on to something (a value in this case). Depending on the input provided at the Latch port, this block can either act as a pass through where the output is the same as input at every sample or a memory block where the output is equal to the input value when the input state of the latch port changed. Parameters: • Polarity: When this parameter is 0, then the output equals input when the value of latch input is non zero and the output is held constant when value of latch input is zero. When this parameter is 1, then the behavior is reversed.	
Manual Switch [24]	
A simple switch block. When the state parameter = 0, then Output = Input 0. When the state parameter = 1. Then Output = Input 1. Since the state is hardcoded in this case, this block is much less useful than the switch block and the author recommends using that.	

Pulse [25]	
Important block for time based switching, Pulse block generates a pulse when the input makes a transition from 0 to 1 (or non-zero). The pulse has a high value of 1 and a low value of zero. The delay and width characteristics which are defined in the block parameters: • Delay: The time gap between the transition of input and the start of pulse. • Width: The width in samples for which the output remains high (1) after the start of pulse. Once width number of samples have passed, the output returns to zero.	
Relation [26]	
A generalized version of the comparator block, this block can check for multiple relationships between input and output like >, >=, <, <=, ==, ~= between input0 and input1. The output is 1 (TRUE) when the relationship is satisfied or else the output is 0 (FALSE) Parameters: Operator: The type of binary comparison that has to be performed	
Proportional Selector [27]	
This block allows to mix two signals such that $Output = Selector * Input0 + (1 - Selector) * Input1$. This block can also be used as a crossfader by implementing a low pass filtered input on the selector.	
Switch (on input) [28]	
An important function for implementing signal or timer based switching. When the switch state input is 0, then output = input0. When the switch state input is 1, then output = input1.	

A.9 Conversion blocks

Any digital arithmetic can only be done between two variables having the same data type. For example, an unsigned integer can be added only with an unsigned integer, and the output also is an unsigned integer. Failure to keep a track of data types can cause serious problems which are usually hard to track. MechaWare™ does follow an implicit typecasting scheme, like C/C++ in cases such as the addition of an integer with a double precision floating point number. The integer input will be typecast to double and then added. The output will be a double too. This ensures minimum loss of data. Normally the user will NOT be required to do an explicit data type conversion but these blocks have been explained in details for completeness of the information.

Block Name	Image
Double to Integer	
Input: double precision (64-bit) floating point value. Output: 32-bit integer value	
Float to Double	
Input: 32-bit single precision floating point. Output: 64-bit double precision floating point	

Double to Float	
Input: 64-bit double precision floating point. Output: 32-bit single precision floating point	
Integer to Double	
Input: 32-bit integer. Output: 64-bit double precision floating point	

A.10 Review: Controller design process

Designing any control system is a complex process because there can never be a "one size fits all" solution. Every system is different and has a lot of parameters that have to be specified/ They can either be modeled using the laws of physics, estimated using measurement data or maybe sometimes, they have to be guessed from intuition. The design process becomes even more complex while implementing custom controllers like force control on a system as large and complicated as the roller Rig. This sort of sandbox environment, while imposing difficulties, also provides the designer to pursue solutions limited only by their creativity.

It is important to make this remark here: A good control system for a nicely designed mechanical system is definitely a match made in heaven. An ingenious system solution can also maybe salvage a less than optimal mechanical design and make it perform satisfactorily. However, a control system design cannot be expected to achieve everything – it is entirely possible that there is no feasible control system solution and the mechanical setup has to be tweaked with or overhauled entirely. The designer as well as all other stakeholders should set realistic goals with the control system design process.

Figure 5-15 shows a flowchart that may help the designer with the process. Any control system design should begin by identifying the problem and target performance specifications. This step should clearly define the variables which have to be tracked and the disturbances which have to be rejected. The performance criteria (steady state error, acceptable fluctuations etc.) should be decided for the outputs. This process is important because the plant that has to be controlled is not generic. Much like the analysis of a dynamic mechanical system, the system boundaries depend on what's being considered by the designer.

Once the plant is decided, the plant has to be measured or modeled. A control system cannot be designed without knowing the system (plant) that has to be controlled. The dynamics of the plant very much affect the control algorithm and its gains. The first step in identifying plant is to list all the states that affect the variables of our interest, which we defined in step 1. Any dynamic system can be mathematically stated as:

$$\dot{x} = Ax + Bu \ (1)$$
$$y = Cx + Du \ (2)$$

Here x is the state vector, which affects y, our output vector. We measure y because y is the variable of our interest so we have sensors for it. The output depends on the state of the plant. The states also evolve dynamically according to the equation 1. u is our control input. The fundamental idea of a control system is to control the time evolution of states using our control inputs and hence manipulate the output to our liking. The manner in which the input u is chosen is exactly what the control algorithm is. The A, B, C & D matrices have different properties

depending on whether the system is linear or non-linear and time-varying or time invariant. However, these matrices have to be determined in order to define the control algorithm.

When A, B, C, D are determined by using physical models, then the process is known as plant modeling. When the matrices are determined from experimental data, then the process is known as plant measurement/estimation. Physics based modeling is a very good approach and yields the engineer a solid idea about the process they are trying to control. But any modeling exercise has its limitations which are inherently human. Physics based equations are often simplified representations of the actual dynamics and have errors. These modeling errors sometimes are small and can be ignored, but on other occasions such equations fail to capture all the pertinent dynamics. This is especially the case when the plant is large and complex.

In such scenarios, plant estimation becomes the go-to method. It is well documented and yields results in a small amount of time. However, it is also an error prone exercise which requires expertise because not all data that is recorded is important. Inevitable sensor noise and errors resulting from discretization of analog signals corrupt the data and the designer should be able to distinguish the signal from noise.

Estimation is not an excuse to ignore the physics of the plant. Even for estimating the plant, the designer should have a good idea of the underlying physics. There are pros and cons to both modeling and estimation. The reader is advised to pursue a method they feel most comfortable in.

For Roller Rig, plant and system measurements can be done by the Bode Tool. A detailed discussion is included in the Section A.11.

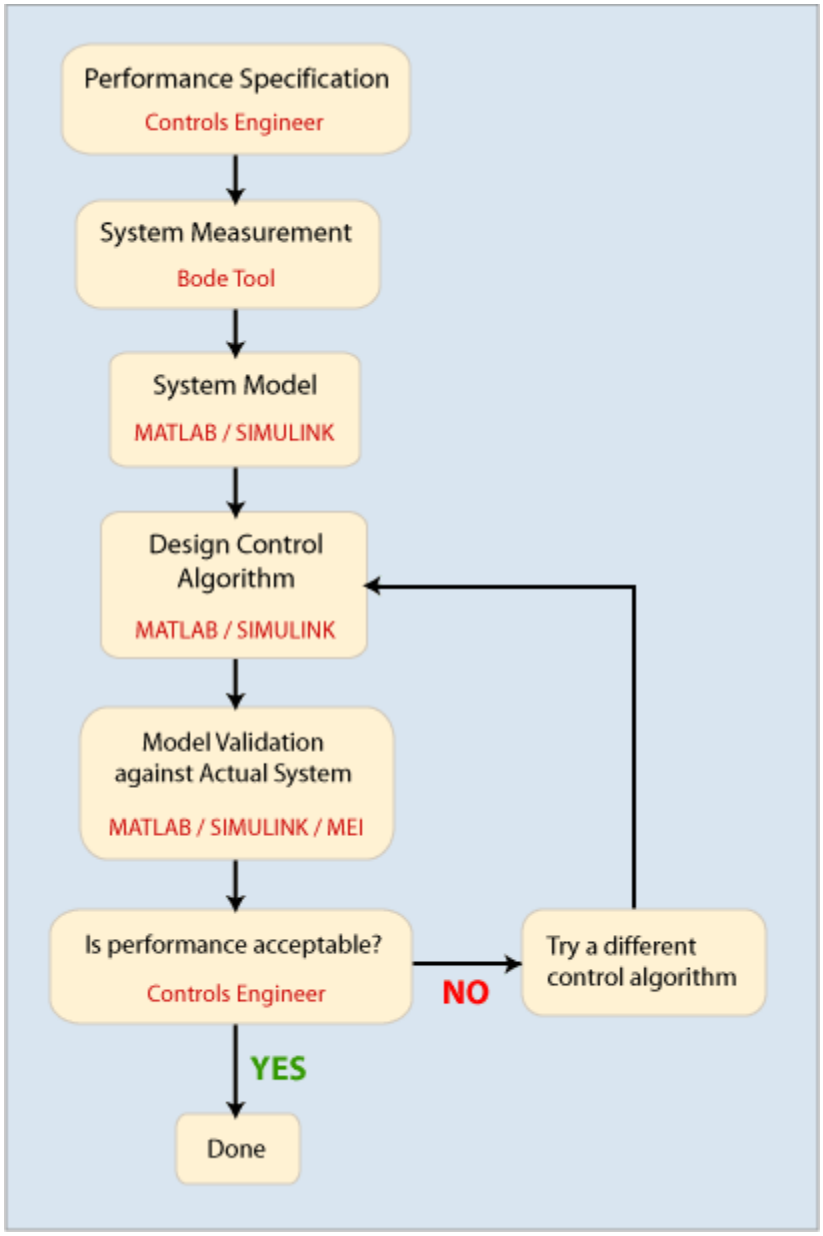


Figure 5-15: Control system design flow

Based upon the plant model or estimation, a control algorithm is decided and the design is carried out. This is the fun part. There are no set rules and no textbook methods. The reader is advised to explore the literature and get an idea of different design methods and case studies. Once the controller is designed, it has to be tuned. This starts an iterative exercise as shown in Figure 5-15.

A.11 Bode Tool

Bode tool is a software that allows the user to conduct frequency domain measurements on the Roller Rig systems. This is a powerful and important tool, but requires a good understanding of frequency domain design from the user. Please refer to any standard controls textbook for a

refresher on frequency domain methods. A background in loop shaping controls design is recommended.

Roller Rig is equipped with two different kinds of firmware: Standard firmware and MechaWare™ firmware. Bode tool has slightly different appearance and functionalities for both of these firmwares. This Section will highlight them as they come up.

A.11.1 Connecting to the controller

Normally Bode Tool would open using the previously loaded settings and this step would be unnecessary. If for some reason, the controller is not found using current settings, the change controller dialog would show as follows:

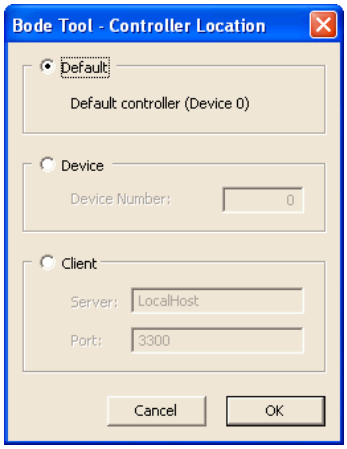


Figure 5-16: Change controller dialog in Bode Tool

Choose the "Default" option to load the Roller Rig controller card.

A.11.2 Main Dialog

The appearances are similar for both standard and MechaWare™ versions of the Bode Tool utility. The top right corner of the dialog box shows the firmware details. For Roller Rig, the firmware version is 915, revision is B3 and option 0 means standard firmware and option 20 means MechaWare™ firmware. When the Bode Tool is launched in standard firmware, then the dialog title just says "Bode Tool." When the software is launched with MechaWare™ firmware, then the dialog title says "Bode Tool (MechaWare™)". If the firmware is changed while the bode tool is open, then the user will have to close and re-launch the software.

A.11.3 Mode (Type of excitation signal)

In the mode Section, it can be seen that there are two types of excitation noise available: Random and Sine. Random refers to PRBN: Pseudo Random Binary Noise. This is often considered a practical equivalent of the theoretical white noise. In signal processing, white noise refers to a theoretical signal which has equal power at all frequencies, just as white is the color that is obtained when all the colors in the visible spectrum are mixed in equal amounts. If a power spectrum plot of PRBN is taken, it turns out to be almost flat across frequencies from 0 Hz to Nyquist. Frequency domain measurements with PRBN excitation has become the industry standard and is considered the most reliable way for measuring the system dynamics.

The next noise type is Sine. This refers to the sinusoidal chirp signal which consists of consecutive sine waves from a starting frequency to the ending frequency. The system is excited

with consecutive sine waves of continuously increasing frequency. The resulting sine wave is measured and its amplitude and phase are plot on a bode plot.

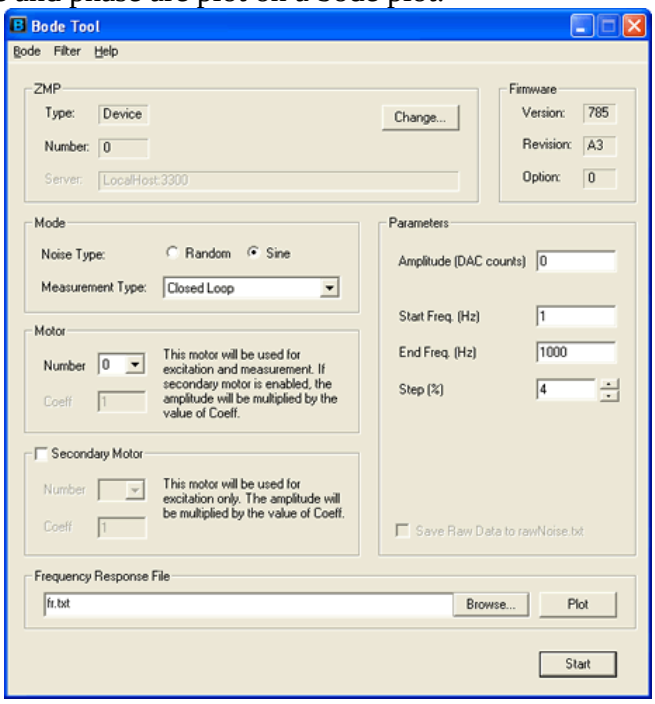


Figure 5-17: Bode Tool on standard firmware

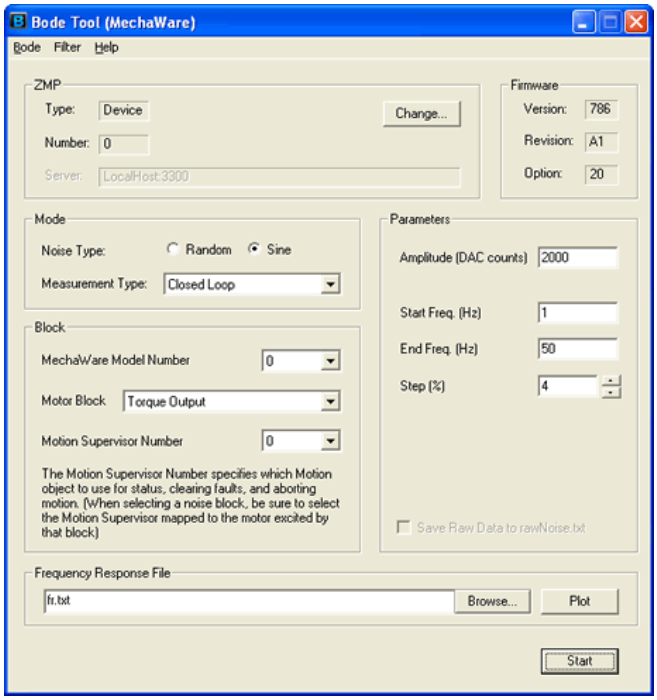


Figure 5-18: Bode Tool on MechaWare™ firmware. Note the option 20 and dialog title

A.11.4 Measurement Type

Measurement type defines the location where the excitation signal is inserted and where the measurement is taken. Available measurement types are Closed Loop, Open Loop, Controller,

Plant, Custom, User Buffer and None. These measurement types can be best understood by looking at the Table 5-1 and Figure 5-19 together. The system in every case is being excited in the location which is marked by the placeholder "Noise" block. The scopes indicate the points where the measurements are being taken.

Measurement Type	Input scope location	Output scope location
Closed loop	0	1
Open loop	2	1
Plant	2	3
Controller	4	2

Table 5-1: Scope locations shown in Figure 5-19 for common measurements in Bode Tool [29]

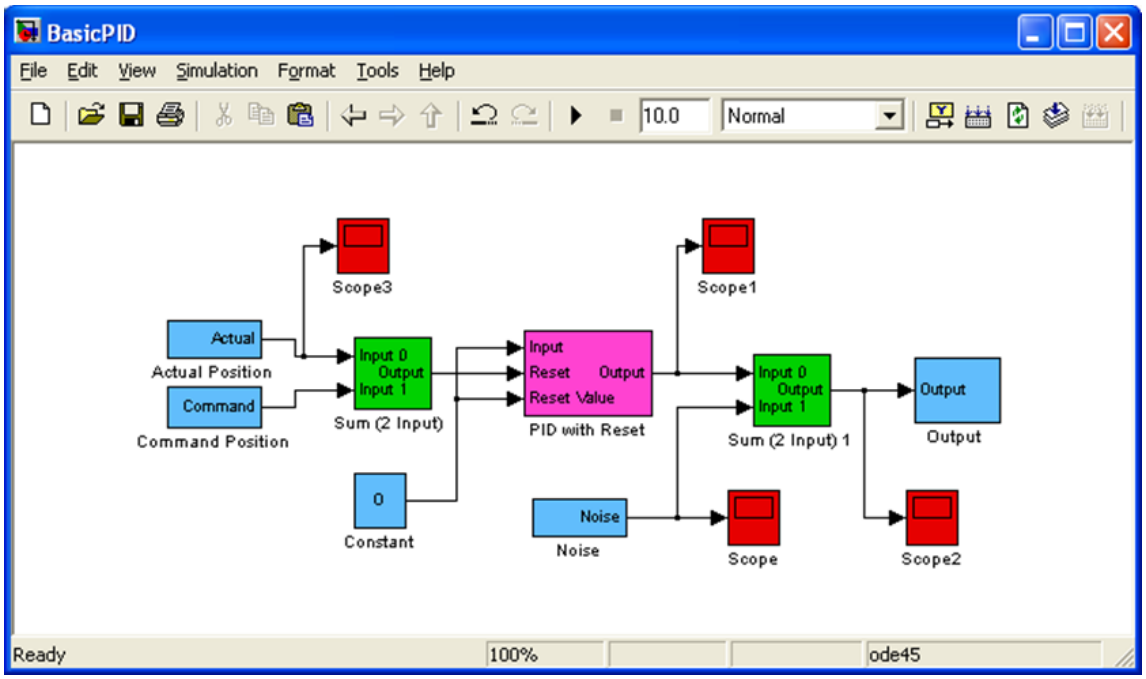


Figure 5-19: Typical closed loop in MechaWare™ and scope blocks to show noise excitation input, measurement input and output ports [29]

The important thing to note here is that the measurement options in Table 5-1 are guaranteed to work only for simple position control PID loops which have a predictable architecture as shown in Figure 5-19. The author maintains that the best way to ensure a proper measurement is to make a MechaWare™ model with the desired measurements being made by the user buffers and then use the user buffer measurement mode in Bode Tool. The true power of the Bode Tool can be harnessed by using the User Buffer or the Custom measurement mode.

A.11.5 MechaWare™ model number

This should default to 0 unless in future multiple MechaWare™ model control architecture is ever utilized. Note that a single ".mdl" file is just one model, even if it contains control algorithms for multiple axes.

A.11.6 Motor Block

This parameter is active only under measurement types: closed loop, open loop, plant, controller. Choosing this parameter tells the Bode Tool to automatically deduce the closed loop that has to be tested. When control loops for multiple motors and axes are in the model, this parameter has to be carefully chosen. The name of the motor block is the same as the name of the demand block the user chooses in the MechaWare™ Model. A distinct naming scheme is recommended to avoid confusion.

A.11.7 Motion Supervisor Number

This parameter is only important when working with the Bode Tool in standard firmware. This parameter defaults to zero when Bode Tool is used with MechaWare™ and is a known bug in the software with no known available patch.

Motion supervisor number is required by the Bode Tool to monitor for any faults that may happen during testing, like position limit error or overcurrent fault. When such things happen during the tuning process, the motion supervisor throws a "Motion Supervisor fault" and this is picked up by the Bode Tool, causing it to stop excitation of the system. It is an important safety feature, only in standard firmware. The user should specify the number of the motion supervisor under which the motor/axis being tested falls. Bode Tool also monitors the "Amp Enable" flag of all the motors that fall under the said Motion Supervisor. For example, if the vertical degree of freedom is being tuned and tested, then both the amplifiers should be enabled otherwise the testing cannot proceed.

In MechaWare™ firmware, the defaulting of motion supervisor to 0 means that no matter which axis is being tested, the amplifier for Motor 0 (wheel motor) has to be enabled in order for the software to commence excitation and measurement. This also means that safety monitoring by the software is non-existent while testing with MechaWare™ models and a heightened level of alertness is required by the user in that case.

A.11.8 Parameters

The parameters depend on the type of excitation noise that is being inserted into the system.

Random Noise

Random noise is generated by the random number generator in the QMP-controller card. Random noise generation is done by a deterministic algorithm, so in reality random noise is referred to as PRBN (Pseudo Random Binary Noise). The spectrum of the PRBN noise is virtually flat across all frequencies from DC to Nyquist, with the phase being random and meaningless. The following parameters are required for a PRBN excitation

- Amplitude (counts): Amplitude refers to the maximum absolute value that the PRBN sequence will have. For example, an amplitude of 500 counts means that the noise will not have any sample that has absolute value greater than 500 counts. When default measurement types are chosen, the amplitude translates into the maximum torque command that will be supplied (in case of a torque demand block). A maximum torque demand of +/- 32767 can be made before the actuators are driven to saturation/overcurrent fault so user should be careful with that. *When the noise input is to a motor torque demand, it is advised to not exceed 500 counts for noise amplitude.*

- Sample Count: Sample count refers to the number of samples that the PRBN sequence will have. Greater number of samples means greater amount of testing time. Greater number of samples also means more measurement data which can lead to a less noisy frequency response plot due to greater averaging. *Recommend sample count: Greater than 10 times to the number of FFT points and always an exponent of 2.*
- Number of FFT Points: The number of FFT points directly influences the lowest frequency that can be measured in the frequency response and the resolution of frequency response. $Frequency\ Resolution = \frac{Sampling\ Frequency}{Num.\ FFT\ points}$. For example, if the frequency of interest is 0.24 Hz, the FFT should have a resolution lower than that in order to get sufficient resolution in data. At the same time, more FFT points will yield a noisier frequency response plot. The number of FFT points cannot be greater than Sample Count. *NFFT should always be a power of 2.*
- Overlap (%): Overlap is a technique in Welch's periodogram method which is related to the windowing of data. The reader is advised to refer to a standard signal processing textbook for more details on the Welch's periodogram method (the bode plot frequency response that Bode Tool calculates). *Recommended Value: 50%. Never more than 50%, can be less than 50% but will mean lesser averaging and a noisier bode plot.*
- Checkbox: Save Raw Data to rawNoise.txt: Allows the user to save the injected PRBN sequence in a .txt file. This noise sequence is valuable for analysis in MATLAB. However, the location of the .txt file is NOT the same as the "frequency response file" location. Instead the rawNoise.txt file is *sometimes* saved in C:\Windows\SysWOW64. In order to retrieve the noise excitation file, the user is advised to do a computer search with the exact name match and the user should look out for the most recent timestamp among the search results.

Sine Noise

Sine noise measurement refers to a sinusoidal sweep signal that is injected into the system. The following parameters can be set:

- Amplitude: Amplitude refers to the peak/valley value of the sine wave. *Recommended value: Less than 500 counts when noise block outputs to a torque demand block*
- Start Freq. (Hz): Refers to the starting frequency of the sine sweep. If this value = 1 Hz, then the first sine wave will have a frequency of 1 Hz. *0.5 Hz is a decent starting value to estimate DC gain and phase measurements.*
- End Freq. (Hz): Refers to the final frequency of the sine sweep. *This value should be less than Nyquist frequency, i.e. 1000 Hz for current sampling rate of 2000 Hz.*
- Step (%): Refers to the increment in frequency between two sine waves. For example, when starting frequency is 1 Hz and a step of 4% is defined, then the next sine waves will have frequencies of 1.04, 1.04, 1.0816, 1.124864, etc. More steps mean a longer test but a smoother curve and greater detail. *Recommended step: start with a 4% step and go lower when fine-tuning system.*

PRBN excitation and FFT method is the most rigorous and trustworthy method for measuring the system's frequency response and should always be preferred. To attain a higher

resolution and smoother curve at lower frequencies, the frequency response from sine sweep measurements can be added to bode plot attained from FFT.

Frequency Response File

Probably the most important parameter, this field should always be checked before starting a test. Click on Browse to choose a folder and type in a file name to name that test file. Use the Plot button to plot the frequency response measurement from a previous test file chosen using the Browse button. If a previous test file is chosen and the measurement is commenced by pressing 'start," then the previous file will be overwritten and the data will be lost.

Appendix B:Motion Programming Interface

B.1 Introduction

(This chapter assumes a good knowledge of development in C/C++ along with advanced concepts like multi-threading and code debugging from the reader.)

Command and data acquisition in the VT-FRA Roller Rig is handled by the Motion Programming Interface (MPI). MPI was developed by Kollmorgen (formerly Motion Engineering) and is a C/C++ based object oriented interface provided for developing motion control applications. The interface abstracts the platform-specific and firmware implementation details via a set of functions which can be used for the high-level design of Roller Rig control application. MPI also includes methods to access Synqnet memory locations, thus also enabling a more fundamental level design for more specialized applications.

When the Roller Rig SynqNet is initialized, the QMP controller stores a map of all components installed on the network on the host computer. In Roller Rig, this file is named QMP915.map. This map is then utilized by the host software running MPI libraries to access the desired memory location and obtain or modify values contained at those locations. MPI provides configuration and information methods for all the objects to *get* and *set* the values. Direct access by outside programs like the host software is prohibited due to the memory locations being defined in a protected or private access specification.

For convenient debugging, MPI also provides global and per-object trace methods which can be used to track function calls, memory allocation, access and modification. The trace feature enables you to follow the progress of an executing program by observing the stream of messages produced whenever a library function is entered (displaying calling parameters) and whenever a library function is exited (displaying the return value) [30].

You also have the ability to validate all library function parameters, and to stop execution whenever an error occurs, displaying the source file name and line number where the error occurred. You can configure the MPI for optimal trace and debug support during the design and test phase, and then reconfigure the MPI for optimal performance (without the debugging support) [30] .

User defined debugging in Visual Studio can be carried out using TRACE(…) function provided by MFC which takes argument in the same way as a printf(..) statement. Such methods help visualize the program flow especially when the application involves numerous function calls, multiple threads and memory access and modification.

B.2 General Definitions

Before we venture into details about the library, it would be better if we defined some terminology which will be extensively used in the coming sections.

B.2.1 MPI

The **M**otion **P**rogramming **I**nterface (MPI) defines a set of object-oriented, C-language functions and data types that you can use to develop motion control applications. The MPI is designed to be independent of the motion controller hardware used, as well as the platform (operating system & compiler) on which the motion control application is built. To support the MPI, a platform must be 32-bit (e.g. Windows NT). Most 32-bit platforms are capable of multi-

threading and multi-tasking, and the MPI has built-in features that simplify the design and development of these types of applications [31] .

B.2.2 Platform

A combination of the host computer, operating system, and C compiler is referred to as the platform [31] .

B.2.3 MPI Installation Directory

The MPI installation directory is the location into which the MPI software has been installed. For the Rig, it is: C:\Program Files(x86)\MEI\MDK\04.03.00\ . The contents of the directories are as shown:

Folder	**Contents**
\controller	Firmware files & firmware address map
\Win32	Libraries, utility programs, device drives and .exe programs for Windows
\MPI	Contains everything pertaining to MPI programming: MPI header files, apputil source files, examples and utility program sources.

Table 5-2: Description of the MPI installation directory [31]

B.2.4 Object

An object is an instance of a data structure. An object contains all the properties of the underlying data structure that it represents. As far as MPI/C++ is concerned, the data structure would be a class. Think of the class as a template and the object as the document based on that template. A class is a container that contains variables as well as functions (code) which are all related by some property. All of those properties are manifested in an object. There is no uniqueness attached to an object, there can be multiple copies of the same object.

When an application calls a **function** (also called a **method**) to create an object, the function returns a handle to the object. The object handle is then passed as the first argument to all other functions (methods) that are associated with that object. These functions provide indirect access to the object's data structure. When it is finished using an object, an application simply calls another function to delete it [31].

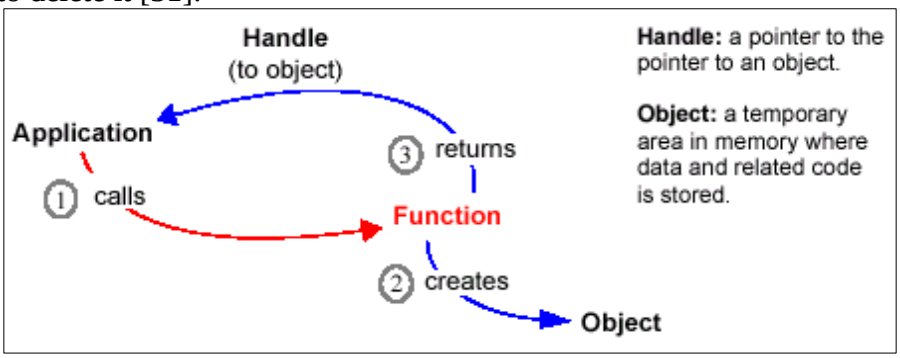


Figure 5-20: Object creation process in MPI [31]

To make this concept absolutely clear, let's go through an actual object creation exercise: An axis object is created by the mpiAxisCreate(...) function, which returns a handle of type MPIAxis. This handle can be used by the host application to interface with the hardware axis

through the methods that the axis object contains. This concept is known as abstraction and encapsulation in programming.

The mpiAxisStatus(...) function takes an Axis handle as its first argument and returns the status of the hardware axis with which the Axis object is associated. An Axis object is deleted by calling the mpiAxisDelete(...) function .

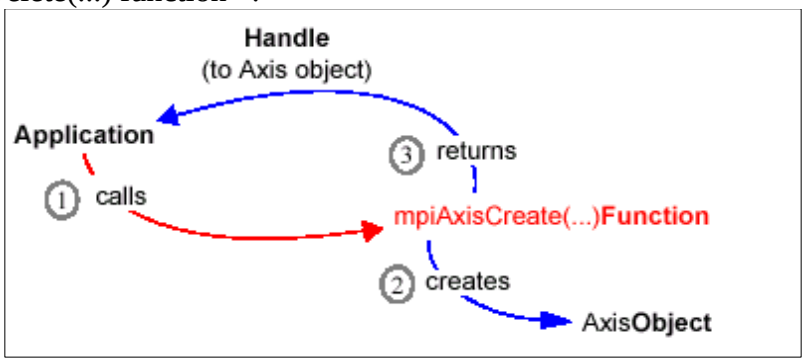


Figure 5-21: Axis creation in MPI [31]

B.2.5 Handle

A handle is a pointer to a pointer to an object. When an object is created, a handle is returned by the method which has created the object. A handle uniquely identifies an object[3] . For example, MPIAxis is a class that can be used to store a handle. Similarly, MPIHandle is a class that MPI uses to define a generic handle. A handle is invalid if it is not passed as an argument to a mpi____Create(…) function. When a function like, say mpiMotorCreate(…) is called, the function allocates memory from the heap to create a Motor object – a software representation that contains all the properties of the physical hardware motor object.

There is only one physical motor, however it is perfectly okay for the application to create multiple representations of the same physical hardware. All these representations will point to the same physical object and will always contain the latest updated configuration of the physical object. However it is absolutely essential to call mpi____Delete(…) functions to deallocate the memory pointed to by the handle. The importance of this cannot be overstressed. Failure to do so can result in memory leaks that will be hard to track and jeopardize the entire application.

The symbol MPIHandleVOID is used to represent an invalid handle . An MPI implementation could define a handle to be a pointer into a static array of data structures, or an index into such an array. Such an implementation arises for example when working with firmware methods [31].

Regardless, an application MUST NOT do anything with a handle other than to pass it as an argument to other functions. In cases where a handle is a pointer, the pointer cannot be dereferenced, because the object data structure to which the handle points is not directly available to the application [31].

B.2.6 Method

A function associated with an object is called a method. The first argument of a method is always the handle of the object with which it is associated. A function that is not associated with an object is called a function.

For example, the MpiAxisCreate(&axis,…); function is considered to be a method. Notice that the first argument is the handle of the object [31].

B.2.7 Module

A **module** is a source file (module.c) whose interface is declared in a header file (module.h). A module contains code symbols (methods, functions, macros) and data symbols (data, data types and constants) that together implement an object. However, a module may also contain functions that have a logical connection without being associated with an object [31] . Following are some important modules in MPI:

MPI Module	Header
Axis	axis.h
Capture	capture.h
Control	control.h
Filter	filter.h
Firmware	firmware.h
Motion	motion.h
Motor	motor.h
Notify	notify.h
Recorder	recorder.h
SqNode	sqnode.h
SynqNet	synqnet.h
Trace	trace.h

Table 5-3: Important modules in MPI [31]

Each module file has a header and a source file. For example, axis module has a header file called axis.h that contains all the relevant declarations. There is also a supporting file called axis.c which contains the definitions of the declarations made in the header file .

B.3 Overview of MPI Objects

B.3.1 Relationship of objects and hierarchy

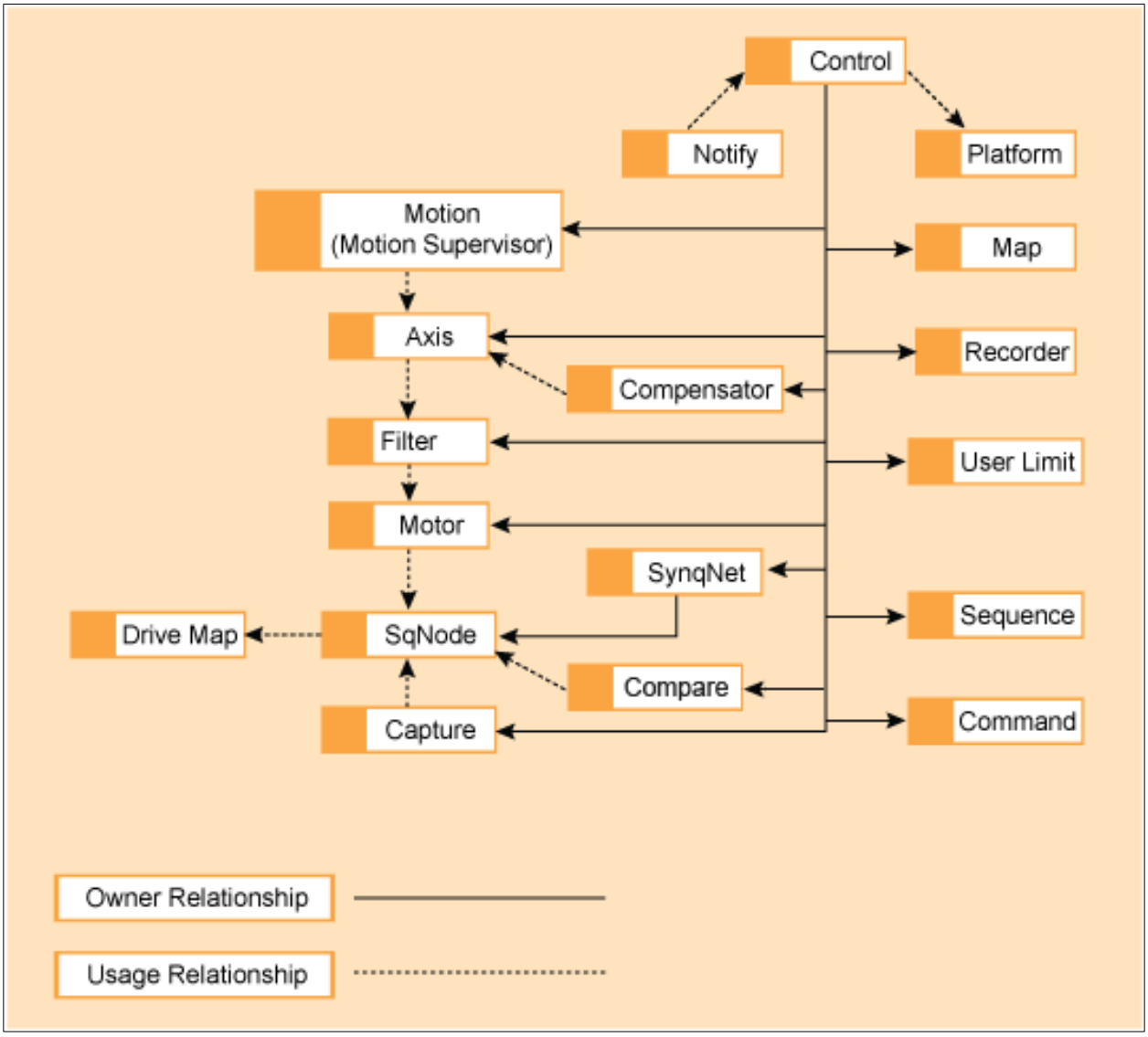


Figure 5-22: Hierarchy of objects in MPI

Figure 5-22 shows the relationship between different objects in MPI. These objects will be described in the coming sections.

Owner relationship means that the top object is required to define and access the bottom object. For example, a Control object is parent to all the other objects on the network and is at the top of the hierarchy tree. Usage relationship means that the object at the end of the arrow requires the object at the root of the arrow to function. For example, a Motion object is required to be mapped with one or more axes objects so as to complete the functionality. Note that the Motion object does not own the axis object and the usage relationship can be changed via the host

application with no impact. However, to create a Motion object, the programmer has to specify the parent, which is the control object.

B.3.2 Control Object

Think of a Control object as a "Controller" object, which for the Roller Rig is the QMP-controller card which resides in the Roller Rig workstation. A Control object manages the motion controller.

Every application creates a single Control object per controller.

A Control object can read and write device memory using I/O port, memory-mapped or device driver methods. All communication with motion controller firmware is handled by a Control object. A Control object can also load a memory snap-shot of a controller stored in a file [32].

B.3.3 Axis Objects

An Axis object is associated with a single physical axis on a motion controller, and corresponds to a geometric axis used for calculation of a path of motion. An Axis may be controlled by one or more Motion objects.

The concept of an Axis is a "geometric" idea, but the main purpose of an Axis object is to generate the desired path (trajectory calculations, i.e., to generate command positions) on every sample, using the path-planning data provided by a Motion Supervisor. An Axis object is mostly a computational block.

The Filter and Motor objects ensure that the command path (calculated by Axis) is followed, and that the signals get to the right motors [32].

B.3.4 Motion Objects

Think of a Motion object as really a "Motion Supervisor" object. The Motion (Supervisor) object corresponds to a coordinate system or collection of axes. The primary function of the Motion Supervisor is to provide data in a synchronized manner to the Axis objects for use in path creation.

A second important function of the Motion Supervisor is to monitor the status of all of the Axes under its control (and all of the Motors, and Filters associated with these Axes), so that motion can be stopped or resumed in a controlled manner, especially in the event of errors. The Motion Supervisor is the primary interface for a your MPI application with respect to motion. A Motion object maintains an ordered list of Axis objects, which specify the coordinate system for all motions to be performed with that Motion object. When a motion is started, the type of motion is specified (trapezoidal, S-curve, parabolic, etc.) along with type-specific motion parameters. A Motion can be started directly (by calling a Motion method), or started when the Motion is associated with a Command that is called by a Sequence (that is executing). An Axis object may be controlled by more than one Motion object, but only one of those Motion objects may be active at a time [32].

B.3.5 Filter Objects

The Filter object is concerned with the controller's closed-loop servo control loop, i.e., what should the controlled output be based on the position error? The Filter is primarily a computational block, taking command positions and actual positions and computing errors.

The Filter object calculates the demand output (representing a torque or velocity command)) that controls a physical motor or motors, using data (command positions) calculated by the Axis object. PID, PIV and Biquad filter calculations are all parts of the Filter object [32].

B.3.6 Event Objects

An Event object contains information about an asynchronous event that has occurred. Event messages are generated and stored in the controller's memory buffer. When an event occurs, an interrupt is generated to the host and the control event service retrieves the event messages and distributes them to the application threads waiting for a notification [32].

B.3.7 Notify Objects

A Notify object is used by a thread to wait for event notification. You can configure a Notify object to wait and look for specific events and for specific event sources [32].

B.3.8 Control Event Service

The Control Event Service thread does the following [32]:

1. Obtains asynchronous events from the Controller that the Control object is associated with
2. Generates event messages for enabled event sources
3. Awakens any threads that are waiting for events

B.3.9 Recorder Objects

The Recorder objects allow you to record any data from the controller's memory. The data is stored in a controller memory buffer and retrieved by the host application. Multiple recorder objects can be enabled and can be started/stopped from the application or by using trigger conditions in the controller memory [32].

B.3.10 Sequence Objects[1]

A motion application can issue individual motion commands, or can create a series of motion commands (that are executed in sequence by the controller). Essentially, you can use the Sequence object to download commands that are executed by the XMP controller, and not executed by the host.

Using the MPI, you can create a sequence of commands (a Sequence object), using high-level motion (e.g., trapezoidal motion profile on axes 2 & 4), using low level work (e.g., write 0x00043433 into memory address 0x2000).

A typical Sequence might be:

1. Start a motion
2. Wait 60 msec.
3. Turn on a specific I/O bit

[1] *It is strongly discouraged to implement Sequence Objects, unless there is a compelling need for them. The Roller Rig Automation Console DOES NOT use Sequence Objects. One needs to consult Kollmorgen engineers before attempting any sequencing based on this.*

4. Wait for motion to finish
5. Wait for another I/O bit
6. Start a new motion

A Sequence object is always executed starting with the first command. After completing the first command, subsequent commands can initiate a new motion, set values in firmware memory, execute a delay for a specified period, branch to a different command in the sequence, wait for a condition to be met, and so on [32].

B.3.11 Command Objects

A Command object specifies a single action that is executed by a Sequence, such as motion, conditional branch, computation, time delay, wait for condition, etc. Any Command object that specifies motion must have a Motion object associated with it [32].

B.3.12 Coordinate Systems

To create a coordinate system for a motion application, you map the Axis numbers to the Motion Supervisor in the controller's memory. The axis count and list of axis numbers define the coordinate system [32].

B.4 Naming Convention

B.4.1 Uniqueness

The MPI naming convention enables you to look at a symbol and know whether the symbol refers to code or data. The MPI uses "MPI" as the initial token of data symbols, while code symbols use an initial token of mpi [33].

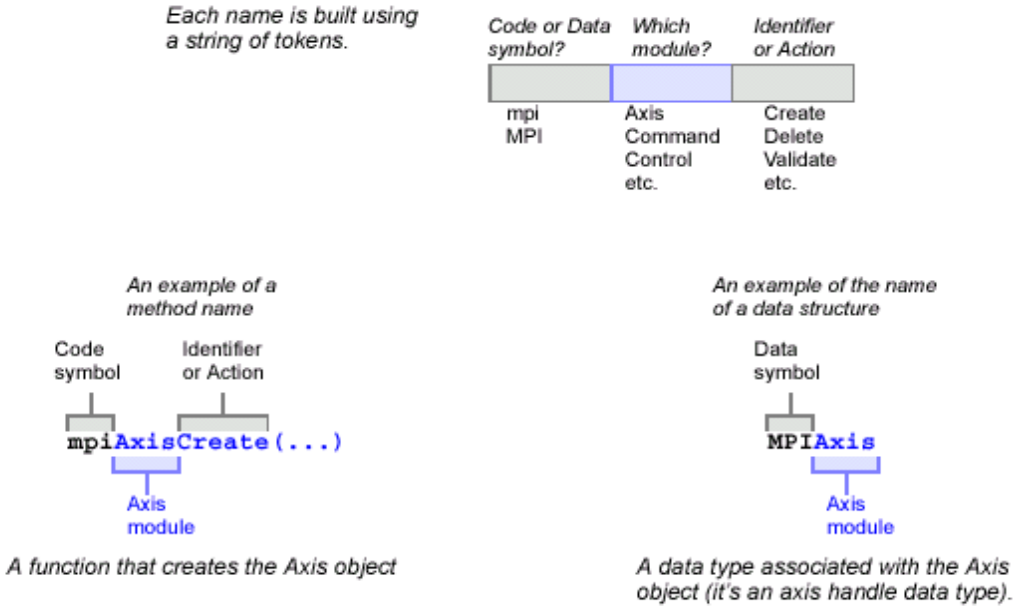


Figure 5-23: Breakdown of MPI naming convention [33]

For example, the axis handle data type is called MPIAxis. The method that creates an axis object is called mpiAxisCreate(...).

B.4.2 Symbol Declaration & Definition

MPI symbols are declared in header files and defined in source files. To determine the file in which an MPI symbol is declared and defined, look at the second token of the symbol name: the second token indicates the module where the symbol is declared and defined. Also, to maximize the portability of the code, the MPI uses a DOS-like 8.3 file naming convention. Therefore, the second token of any symbol name will not exceed 8 characters, nor will the names of modules and objects [33].

For example, consider a symbol named MPIObjectXyz. The symbol is a data symbol (MPI) declared in object.h and defined in object.c. Similarly, a symbol named mpiObjectAbc is a code symbol (mpi) declared in object.h and defined in object.c.

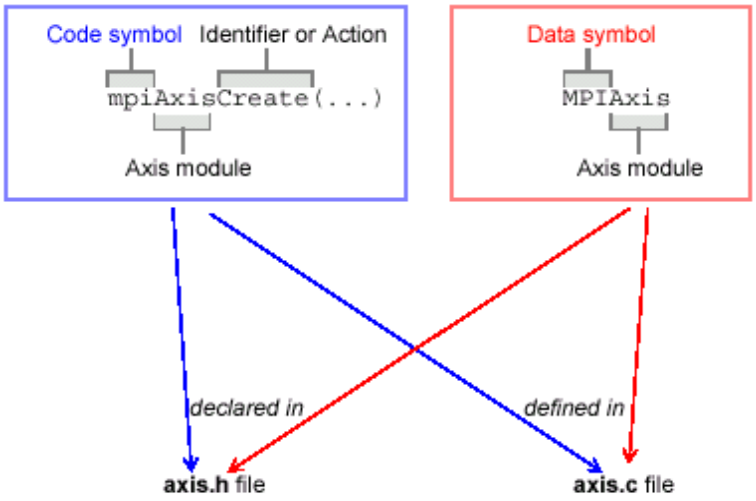


Figure 5-24: Declaration and definition of symbols in MPI [33]

B.4.3 Symbol Naming

The previous sections use the concept of a token when discussing symbol names, where a token is a word or abbreviation contained in a symbol name. The name of an MPI symbol consists of a left-to-right sequence of tokens (with no spaces in between). An uppercase letter indicates the beginning of a token, except the first token of a symbol, which may be either upper (MPI) or lower case (mpi). The MPI uses underscores in symbol names only when it is necessary to separate tokens that are all uppercase letters.

The first token of a symbol (MPI or mpi) guarantees uniqueness and indicates whether the symbol is a data symbol (MPI) or a code symbol (mpi) . The second token indicates the module that declares and defines the symbol. The remaining tokens of the symbol name vary, depending on whether the symbol is code or data [33].

B.4.4 Data Symbols

Tokens in data symbols are arranged from left-to-right in a hierarchical fashion, with the leftmost token being general and the rightmost token being specific. If the data symbol names are listed alphabetically, this scheme results in a listing that groups related data symbols together.

If the rightmost token of a data symbol is all uppercase, then that data symbol is a constant. If the rightmost token of a data symbol is not all uppercase, then that data symbol is either data or a data type. Because the MPI contains virtually no global data, there is little need to distinguish between data and data type [33].

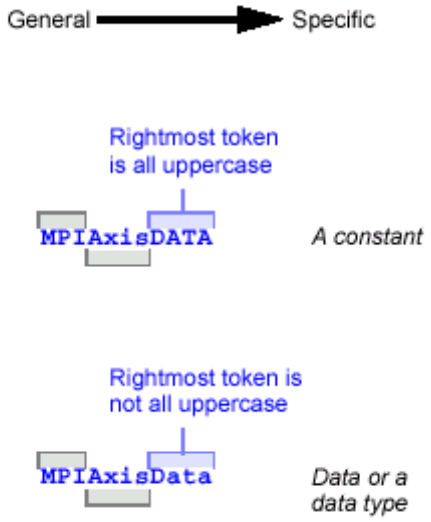


Figure 5-25: Naming convention for data symbols [33]

Consider the MPIMotorLimitType enumeration:

```
      typedef enum MPIMotorLimitType {
              MPIMotorLimitTypeINVALID = -1,

              MPIMotorLimitTypeAMP_FAULT,
              MPIMotorLimitTypeAMP_WARNING,
              MPIMotorLimitTypeFEEDBACK_FAULT,
              MPIMotorLimitTypeERROR,
              MPIMotorLimitTypeTORQUE,
              MPIMotorLimitTypeHW_NEG,
              MPIMotorLimitTypeHW_POS,
              MPIMotorLimitTypeSW_NEG,
              MPIMotorLimitTypeSW_POS,

              MPIMotorLimitTypeEND,
              MPIMotorLimitTypeFIRST = MPIMotorLimitTypeINVALID + 1,
              MPIMotorLimitTypeCOUNT       =       MPIMotorLimitTypeEND       -
MPIMotorLimitTypeFIRST
      } MPIMotorLimitType;
```

The enum name (MPIMotorLimitType) is incorporated in the name of all of the enum members, whose last token is all uppercase (AMP_FAULT, AMP_WARNING, FEEDBACK_FAULT, etc.) to indicate that they are constants (instead of a data type). It is standard coding practice to use enums rather than #defines to declare constants, which allows for greater control of the name space and better type-checking. The use of FIRST and END members in an enum is also standard coding practice, so that functions can bounds-check enum values [33].

B.4.5 Code Symbols

Tokens in code symbols indicate the data type and order of the arguments to the function. The last token in a code symbol is often the action that the function performs, or the data type that the function returns. If the code symbol names are listed alphabetically, this scheme results in a listing that groups related code symbols together [33].
If the rightmost token of a code symbol is all uppercase, then that code symbol is a macro.
If the rightmost token of a code symbol is not all uppercase, then that code symbol is either a function or a method.

Consider mpiAxisConfigGet(...) and mpiAxisConfigSet(...), which are 2 methods used to configure an Axis. Because these functions are methods, the first argument to them is an Axis handle. The second argument is a pointer to a structure of type MPIAxisConfig{}. The last token of the method name indicates the action to take; Get fills the supplied structure with Axis configuration and Set configures the Axis from the supplied structure.

The mpiAxisControl(...) method returns the MPIControl handle with which the Axis was created, while mpiElementNEXT(...) is a macro that returns the MPIElement following the specified Element [33].

B.5 Object Methods

B.5.1 Note

Where used in the rest of this Section, the word ***Object*** may be replaced with the name of an MPI object in order to obtain the name of the methods for that object. mpi*Object*ConfigGet can mean mpiAxisConfigGet, mpiMotorConfigGet, etc. MPI*Object*Config can mean MPIAxisConfig, MPIMotorConfig, etc. For example, for the term mpi*Object*Create(...), replace *Object* with Axis to obtain the name of the method that you would use to create Axis objects, such as mpiAxisCreate(...). For the term MPIObject, replace *Object* with Axis to obtain the data type of the Axis handle (MPIAxis) returned by the Axis method mpiAxisCreate(...), and also used by other Axis methods, such as mpiAxisValidate(...).

B.5.2 Introduction

Object-oriented languages such as C++ and Java provide capabilities such as inheritance and polymorphism. The MPI is written in the C-language, and so these more exotic object-oriented features are not available. However, the basic concepts of object-oriented design are in the MPI: modularity, data hiding and no global or static data.
It is similar to C++.

If you are already familiar with C++, you will quickly realize that the MPI methods that create and delete objects are patterned after C++ constructors and destructors. An MPI method takes an object as its first argument, much as a C++ method has a "this" pointer. Object data is private and available only by calling object methods. MPI source code is organized into modules, with one object per module. Library state is distributed across application-created objects; there is virtually no state information maintained by the library.

All object methods and all functions return a value indicating whether the call to them succeeded or not. In general, the return value is of type long, with a value of 0 indicating success. The return value can be treated as an MPIMessage, so that *you can call the mpiMessage(...) function to obtain a text string that describes the return value. The text string that indicates a successful return value of MPIMessageOK (0) is empty ("").*

All MPI objects must implement certain required methods. Note that most MPI objects implement standard configuration and memory methods; many MPI objects implement standard status and list manipulation methods and a few objects implement standard event notification methods. For multi-threaded environments, all objects must implement standard resource allocation timeout methods.

In general, there are few methods that are unique to an object, and those unique methods tend to be those that perform object-specific actions. The following sections describe the standard methods that are implemented by many objects [34].

B.5.3 Common Methods for MPI Objects

All MPI objects have four types of common methods:

- methods which create an object
- methods which delete an object
- methods which validate an object
- methods which identify an object

Warning: The mpiControlReset(), mpiControlResetToDefault(), and mpiControlConfigSet() methods will all erase previously configured object settings. For this reason these methods must be called before creating any dependent objects, (motors, axes, recorders, etc.). If these methods are called after the creation of dependent objects, then error (MPIControlMessageOBJECTS_CREATED) will be returned. Extra care must be taken for multi-threaded applications. If objects are created in one thread, these control methods may not return an OBJECTS_CREATED error if called from a different thread. The same problems associated with objects created before calling these methods applies however even if the objects are in different threads [34].

B.5.4 mpiObjectCreate(...)

Use the Create method to create an object and return a handle that uniquely identifies that object. *The MPI does not support declaring an object directly, so you must always use the Create method*. The Create method arguments depend on the type of object being created.

If an object cannot be created, mpiObjectCreate(…) returns an error and set the handle to MPIHandleVOID. In general, if you call a method using an object handle that is invalid, the method will fail and return a value indicating the cause of failure.

The MPI supports multiple motion controllers. Typically, an MPI application first creates a Control object that corresponds to the desired motion controller. If the application will use more than one motion controller, it must create a Control object for each motion controller. The arguments to mpiControlCreate(...) associate a Control object with a specific motion controller.

Most MPI objects are associated with a specific instance of a resource on a specific motion controller. An Axis object, for example, is associated with a single axis on a motion controller. When creating an MPI object that corresponds to a specific motion controller resource, the Create method takes an MPIControl handle as its first argument and a number as the second argument. The number argument identifies the specific motion controller resource to be used.

Example: The number argument of mpiAxisCreate(...) identifies an axis on the motion controller specified by the MPIControl handle argument.

Note: Number arguments, for those MPI objects that take them, start with 0 (not 1). For example, on an 8-axis XMP motion controller, valid axis numbers are 0 – 7 [34].

B.5.5 mpiObjectDelete(...)

Use the Delete method to delete an object created by the Create method. After an object has been deleted, the handle to that object can no longer be used.

If a method attempts to use a handle to a deleted object, it returns MPIMessageObject_FREED, but not always. If the memory originally allocated for the deleted object is subsequently allocated for a different purpose, then the method will not return MPIMessageObject_FREED.

Your application must ensure that handles to deleted objects are no longer used. **After an object is deleted, your application should ensure that the handle to that object is set to MPIHandleVOID.** After that, any calls to methods using this handle will return MPIMessageHANDLE_INVALID.

When your application exits, it should call Delete methods for all objects that have been created by your application. It is recommended that your application delete objects in the reverse order in which they were created. Regardless, the Control object should be the last object deleted. **The MPI does not provide the ability to automatically delete all of the objects that have been created** [34]**.**

Alternatively, to delete objects, your application may use the C library function atexit(...), which guarantees that the objects will be deleted even if the application terminates abnormally.

B.5.6 mpiObjectValidate(...)

Use the Validate method to validate an object handle. The Create methods will automatically validate the object handle. The Validate method can be called at any time in an application.

If the object is valid, the Validate method will return MPIMessageOK. If the handle passed to the Validate method has value MPIHandleVOID, the Validate method returns MPIMessageHANDLE_INVALID. If a method attempts to use a handle to a deleted object, the method returns MPIMessageObject_FREED. (Also see the previous discussion in mpi*Object*Delete(...)).

Other possible return values are declared in the message header file(MPI\include\message.h) and in the object's header file (MPI\include\object.h) [34].

B.6 Configuration Methods

B.6.1 Introduction

An object's configuration is static information that may be read and written, and an object retains its configuration until an application changes it. To configure an MPI object, you use a Config structure that is specific to each object.

To configure an MPI object, get the current Config from the motion controller by calling the ConfigGet method. Modify the Config structure as desired, and then send it back to the motion controller by calling the ConfigSet method.

Warning: The mpiControlReset(), mpiControlResetToDefault(), and mpiControlConfigSet() methods will all erase previously configured object settings. For this reason these methods must be called before creating any dependent objects, (motors, axes, recorders, etc.). If these methods are called after the creation of dependent objects then error (MPIControlMessageOBJECTS_CREATED) will be returned. Extra care must be taken for multi-threaded applications. If objects are created in one thread, these control methods may not return

an OBJECTS_CREATED error if called from a different thread. The same problems associated with objects created before calling these methods applies however even if the objects are in different threads [35].

B.6.2 MPIObjectConfig{} Structure

An object's configuration structure contains all of the configurable items for that object. To configure an object, your application must declare a variable of type MPIObjectConfig{}. Note that some Config structures are very large, so if you declare a Config structure on a small stack, your application might encounter problems. A pointer to a Config structure is passed to the Config methods [35].

Root	Method	Description
ConfigGet	mpiObjectConfigGet(…)	Fill configuration structures with current motion controller configuration
ConfigSet	mpiObjectConfigSet(…)	Change current motion controller configuration

Table 5-4: General object configuration methods in MPI

B.6.3 mpiObjectConfigGet(...) / mpiObjectConfigSet(...)

The mpiObjectConfig( ) structure is meant to illustrate how objects are configured. Please swap the object you are interested in for the object to configure. For example, mpiObjectConfigSet( ) shows the naming convention that mpiAxisConfigGet( ) follows.

Use the **Get** method to fill the configuration structures with the current motion controller configuration for the object.

Use the **Set** method to change the current motion controller configuration for the object [35].

B.7 Memory Methods

B.7.1 Introduction

You use Memory methods to access a motion controller's RAM. The MPI imposes no structure on motion controller RAM, so the use of memory methods requires implementation-specific knowledge of the motion controller's memory map.

The controller's memory map is version specific. If you use a hard-coded address with a memory method, it may not be compatible with other controller firmware versions. The firmware.h header file contains the definitions for the direct memory access methods. The mfw.h header file defines the controller's memory map and the interface to the MPI. It is intended for internal purposes only. The defines, structures, and ordering will change with each revision [36].

B.7.2 Access Memory Methods

Memory access via these methods is thread-safe; only one thread at a time can read or write the portion of memory associated with the object. *The Get/Set methods will bounds-check the memory address to be accessed, and return an error if the address is not associated with the object* [36].

B.7.3 Control Object Memory Methods

The Control object memory methods are an exception. Using the ControlMemory methods, an application may access all memory on the motion controller at any time, without constraint. *The ControlMemory methods are **not** thread-safe.* Memory methods for the other objects are generally implemented by locking the Section of memory associated with the object, and then calling a ControlMemory method.

It is possible to write a motion application using only a Control object. After creating and initializing the Control object, the address of motion controller memory can be obtained, and you can use the ControlMemoryGet/Set methods to access that memory.

Depending on the type of Control object created, an application can directly access the motion controller memory without using the ControlMemoryGet/Set methods. Such an application would bypass the rest of the MPI library and must implement its own thread safety, as well as deal with how the motion controller firmware operates [36].

Method	Description
mpiObjectMemory(…)	Return host address of object in memory
mpiObjectMemoryGet(…)	Read motion controller memory
mpiObjectMemorySet(…)	Write motion controller memory

Table 5-5: Generic object memory methods in MPI

B.7.4 mpiObjectMemory(...)

The mpiObjectMemory(...) method returns a host address that maps to the Section of motion controller memory associated with the object. mpiObjectMemory(...) uses 2 arguments: the **object handle**, and an **output argument** of type void **. If a call to mpiObjectMemory(...) succeeds, the location pointed to by the output argument is set to the host address [36].

B.7.5 mpiObjectMemoryGet(...) / mpiObjectMemorySet(...)

These methods read (get) and write (set) motion controller memory. They take 4 arguments:

- The object handle
- a destination address of type void*
- a source address of type void*
- length of memory written or accessed in bytes

The Get method's destination address points to host memory, while the source address points to motion controller memory (based on the address returned by the Memory method for the object). The Set method is the opposite. The Set method's destination address points to motion controller memory and the source address points to host memory.

Figure 5-26: Working of MPI memory methods [37]

B.8 Event Notification Methods

B.8.1 Introduction

The MPI Control Event Service is responsible for the collection and distribution of host notifications of firmware events. An application often needs to be notified of events that take place on the motion controller. Events include normal motion completion, motion limits being reached, hardware failure, etc. The Event methods enable your application to request host notification of certain types of events, while ignoring other types of events. Some events are latched, in which case your application must reset the event before the event can be triggered again [37].

Method	Description
mpiObjectEventNotifyGet(...)	Get event mask used for host notification
mpiObjectEventNotifySet(...)	Set event mask to request host notification of events
mpiObjectEventNotifyReset(...)	Reset event notification

Table 5-6: Generic event notification method nomenclature in MPI

B.8.2 mpiObjectEventNotifyGet(...)

Use EventNotifyGet to return an MPIEventMask, which has a bit set for each type of event that host notification has been requested for, by the object. The event mask determines which events will be reported as notifications/interrupts to control event service and which ones will not be [37].

B.8.3 mpiObjectEventNotifySet(...)

Use EventNotifySet to request host notification for each type of event specified in the MPIEventMask argument.

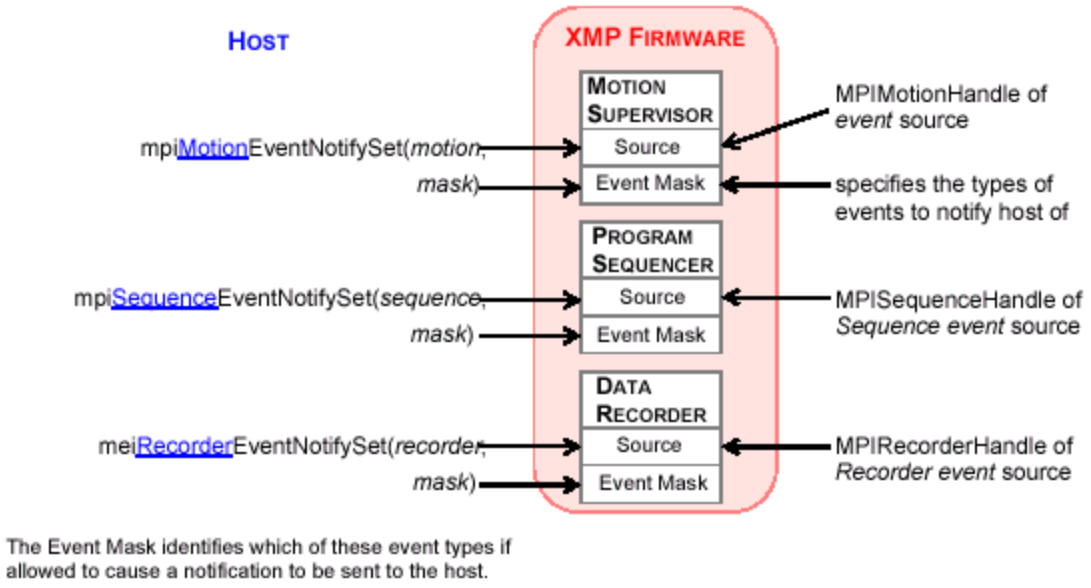


Figure 5-27: Examples of MPI notify set functions [37]

B.8.4 mpiObjectEventNotifyReset(...)

Use EventNotifyReset to reset each type of event specified in the MPIEventMask argument [37].

B.9 Conclusion

A brief yet intuitive description of MPI programming methods was discussed. The reader is encouraged to look up at the respective manuals for a more involved understanding. These programming concepts have been widely used for developing the automation software as discussed in the next chapter.

Bibliography

[1] S. Meymand, “State of the Art Roller Rig for Precise Evaluation of wheel-Rail Contact Mechanics and Dynamics,” Virginia Polytechnic Institute and State University, 2015.

[2] Kistler, “Instruction manual 3-Component Force Sensor 9027C, 9028C,” no. 949. Kistler Group, pp. 0–64, 2015.

[3] J. Karki, “Signal Conditioning Piezoelectric Sensors,” 2000.

[4] Kistler, “Instruction Manual Kistler Multichannel Charge Amplifier Type 5070A.” pp. 1–183, 2015.

[5] Kollmorgen, “SQIO-MIXEDMODULE1 | Kollmorgen | SynqNet I/O 64 Isolated Digital Inputs 64 Isolated Digital Outputs 16 Differential Analog Inputs 8 Single Ended Analog.” [Online]. Available: https://www.kollmorgen.com/en-us/products/machine-controls/i-o/synqnet/sqio/sqio-mixedmodule1/. [Accessed: 07-Jun-2018].

[6] S. Iwnicki and T. Dahlberg, *Handbook of Railway Vehicle Dynamics*. 2006.

[7] M. Hosseinipour, “Electromechanical Design and Development of the Virginia Tech Roller Rig Testing Facility for wheel-Rail Contact Mechanics and Dynamics,” Virginia Tech, 2016.

[8] Kollmorgen, “XMP Gantry Configurations.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_00/Topics/gantry_config.htm. [Accessed: 11-May-2018].

[9] L. Villani and J. De Schutter, “Force Control,” in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Cham: Springer International Publishing, 2016, pp. 195–220.

[10] N. Hogan, “Impedance Control: An Approach to Manipulation: Part I—Theory,” *J. Dyn. Syst. Meas. Control*, vol. 107, no. 1, pp. 1–7, Mar. 1985.

[11] J. De Schutter and H. Van Brussel, “Compliant Robot Motion II. A Control Approach Based on External Control Loops,” *Int. J. Rob. Res.*, vol. 7, no. 4, pp. 18–33, 1988.

[12] “PID_Loop.gif (GIF Image, 600 × 392 pixels).” [Online]. Available: http://tikalon.com/blog/PID_Loop.gif. [Accessed: 18-May-2018].

[13] S. Southward, “Lecture 9, ME 5564.” 2017.

[14] Mathworks, “Transfer function estimation - MATLAB tfest.” [Online]. Available: https://www.mathworks.com/help/ident/ref/tfest.html. [Accessed: 05-Jun-2018].

[15] Mathworks, “Identify discrete-time filter parameters from frequency response data - MATLAB invfreqz.” [Online]. Available: https://www.mathworks.com/help/signal/ref/invfreqz.html. [Accessed: 05-Jun-2018].

[16] Kollmorgen, “Mechaware model writer documentation.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/model_dwnldr.htm. [Accessed: 23-May-2018].

[17] Kollmorgen, “Mechaware API Documentation > Lookup > Lookup (1D).” [Online].

Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/lookup.asp. [Accessed: 05-Jun-2018].

[18] Kollmorgen, “Mechaware API Documentation > Lookup > Lookup (2D).” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/lookup2d.asp. [Accessed: 05-Jun-2018].

[19] Kollmorgen, “Matlab Utilities for Biquad filters.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/matlab_utilities.htm. [Accessed: 27-May-2018].

[20] Kollmogen, “Mechaware API Documentation > Logic > Binary Relation.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/binary_relation.asp. [Accessed: 27-May-2018].

[21] Kollmorgen, “Mechaware API Documentation > Logic > Compare.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/compare.asp. [Accessed: 27-May-2018].

[22] Kollmorgen, “Mechaware API Documentation > Logic > Counter.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/counter.asp. [Accessed: 27-May-2018].

[23] Kollmorgen, “Mechaware API Documentation > Logic > Latch.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/latch.asp. [Accessed: 27-May-2018].

[24] Kollmorgen, “Mechaware API Documentation > Logic > Manual Switch.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/man_switch.asp. [Accessed: 27-May-2018].

[25] Kollmorgen, “Mechaware API Documentation > Logic > Pulse.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/pulse.asp. [Accessed: 27-May-2018].

[26] Kollmorgen, “Mechaware API Documentation > Logic > Relation.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/relation.asp. [Accessed: 27-May-2018].

[27] Kollmorgen, “Mechaware API Documentation > Logic > Proportional Selector.” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/selector.asp. [Accessed: 27-May-2018].

[28] Kollmorgen, “Mechaware API Documentation > Logic > Switch (on Input).” [Online]. Available: http://support.motioneng.com/Software-Mechaware_04_02/API-Documentation/switch.asp. [Accessed: 27-May-2018].

[29] Kollmorgen, “Mechaware Training Presentation.” 2018.

[30] Kollmorgen, “MPI Trace Objects.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_04/docs/Trace/trc_out.htm. [Accessed: 02-May-2018].

[31] Kollmorgen, “MPI General Definitions.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_02/basics/basics/gen_def.htm. [Accessed: 02-May-2018].

[32] Kollmorgen, “MPI Object Descriptions.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_02/basics/basics/obj_des.htm. [Accessed: 02-May-2018].

[33] Kollmorgen, “MPI Naming Conventions.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_02/basics/basics/naming_conventions.htm. [Accessed: 02-May-2018].

[34] Kollmorgen, “MPI Object Methods.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_02/basics/methods/obj_methods.htm. [Accessed: 09-Jun-2018].

[35] Kollmorgen, “MPI Configuration Methods.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_02/basics/methods/cf_methods.htm. [Accessed: 09-Jun-2018].

[36] Kollmorgen, “MPI Memory Methods.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_02/basics/methods/mem_methods.htm. [Accessed: 09-Jun-2018].

[37] Kollmorgen, “MPI Event Notification Methods.” [Online]. Available: http://support.motioneng.com/Software-MPI_04_02/basics/methods/evtnfn_methods.htm. [Accessed: 09-Jun-2018].