DEVELOPING A MOBILE REDUCED GRAVITY SIMULATOR

by

TIMOTHY JOHN MOURLAM

B.S., Kansas State University, 2011

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Mechanical & Nuclear Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas
2012

Approved by:

Major Professor
Dr. Dale Schinstock

Abstract

This thesis describes the design, development, and initial testing of the Mobile Reduced Gravity Simulator (MoRGS). MoRGS is a hoist with active force control, to be used in terrestrial environments with human test subjects for the simulation of partial gravity or zero gravity environments. It is to be used with the subject performing activities while being harnessed to the hoist. The following work here describes the mechanical design, structural and dynamic analyses, simulations used to aid in the control design and component selection, the development of unique control algorithms tailored to the objectives and uncommon dynamics of MoRGS, and initial testing performed without the use of human subjects.

Major components of the MoRGS system include: AC servo motor, gearbox, custom-designed drum, pneumatic muscle, load cell, and a microprocessor. The system is designed to track the motion of the test subject over several meters of vertical travel at speeds of up to 2 Gs of acceleration. This allows for high performance during subject's physical tests, including running on a treadmill and a climbing ladder. It is capable of offloading 50 lb. to 600 lb. and the level of desired reduced gravity is programmable.

Results from testing of the system demonstrate that MoRGS system achieves its goals. It performs well, and the sensitivity of the force controller enables it to compensate for the most minute human motion disturbance.

Table of Contents

List of Figures

List of Tables

Nomenclature

Acronyms

AC	- Alternating current
ADC	- Analog to digital converter
AEVA	- Advanced extravehicular activity
AGMA	- American Gear Manufacturer Association
AMI	- Kansas State University Advanced Manufacturing Institute
ARGOS	- Active Response Gravity Offload System
ARM	- Architecture for computer processors
AVS Lab	- Autonomous Vehicle Systems Laboratory
CAD	- Computer-aided design
CG	- Center of gravity
CMAA	- Crane Manufacturers Association of America
CNC	- Computer numerical control
DC	- Direct current
DMA	- Direct memory allocation
DOF	- Degree of freedom
ECE Dept.	- Kansas State University Electrical and Computer Engineering Department
ECP	- Exercises Countermeasures Project
EMF	- Electromotive force
EVA	- Extra-vehicular activity
EWARM	- Embedded Workbench for ARM
FEA	- Finite element analysis
FOS	- Factor of safety

IAD	- Intelligent assist device
IAR	- Embedded system technology company
IDE	- Integrated development environment
ISS	- International Space Station
KIN Dept.	- Kansas State University Kinesiology Department
KSU	- Kansas State University
LCD	- Liquid crystal display
MATLAB	- MathWorks' numerical computing environment & model-based design
MNE Dept.	- Kansas State University Mechanical & Nuclear Engineering Department
MoRGS	- Mobile Reduced Gravity Simulator
NASA	- National Aeronautics and Space Administration
PI	- Proportional-integral
PMA	- Pneumatic Muscle Actuator
POGO	- Partial Gravity Simulator
PWM	- Pulse-width modulated
RPM	- Revolution per minute
SISO-TOOL	- Single input-single output design interface for compensator design
USART	- Universal asynchronous receiver / transmitter
VSE	- Vision for Space Exploration

Dedication

This thesis is dedicated to my fiancée, Katie. I cannot thank you enough for your continued love, support, and guidance.

Chapter 1

Introduction and Background Information

1.1 Motivation for Project

The AVS Lab has partnered with the ECE Dept. and the KIN Dept. in a cross-discipline astronaut physiology research project for NASA. The ECE Dept. is contributing biometric sensors, the KIN Dept. is performing the physiological tests, and the AVS Lab is responsible for the design and construction of MoRGS for the project. In order to perform simple simulation tests on human subjects and to identify key physiological measurements of astronaut fitness in space, the KIN Dept. has a need for a MoRGS system. The work described in this thesis has been performed in an effort to fulfill that need.

The KIN Dept. is developing tests and measures of astronaut fitness that can be used in space. The existing understanding of the performance of an astronaut in reduced gravity environments is limited because of the lack of opportunities to perform tests while in space or on the moon. Therefore, some testing must be done in terrestrial environments. No widely accepted physical fitness markers exist that deem an astronaut to be physically successful when sent on a space mission in a reduced gravity environment. The majority of the fitness tests performed by future astronauts originates in NASA's Apollo era and has become outdated. Continued success of mission-critical tasks relies on the health and fitness of the crew members especially since space missions continue to increase in duration. Space suit design and extended space flights are two of the most prominent areas of interest of NASA in regard to human physiology. The new tests by the KIN Dept. will address these concerns and MoRGS will help in their development.

1.1.1 Space Suit Design

In 2005, the VSE and the NASA Space Authorization Act of 2005 were passed and redirected the long term goals of NASA [15]. The existing American and Russian space suits were inadequate compared to the new required capabilities of space suits in the VSE. Space suits from the space shuttle era are nearly 30 years old and are large, expensive, and can weigh over 500 lb. New materials and new technologies promise a much more effective space suit.

NASA formed the AEVA Project which generated a comprehensive list of 24 criteria for the future "Über suit." The criteria directly related to astronaut physical performance were intravehicular mobility, microgravity mobility, comfort (unpressurized), comfort (pressurized), and surface mobility. New suit designs require thorough testing in reduced gravity simulators to evaluate how well they accommodate the mobility requirements of the astronaut before being sent into space.

1.1.2 Extended Space Flight

Programs at NASA such as the ECP are working to understand reduced gravity environment's impact on an astronaut's health during and after a space mission [20]. Astronauts returning from space have suffered bone, muscle, cardiovascular, sensorimotor/neurovistibular, and psychological losses, sometimes severe. The ECP believes exercise during the spaceflight is the best means of minimizing the erosion of the astronaut's health.

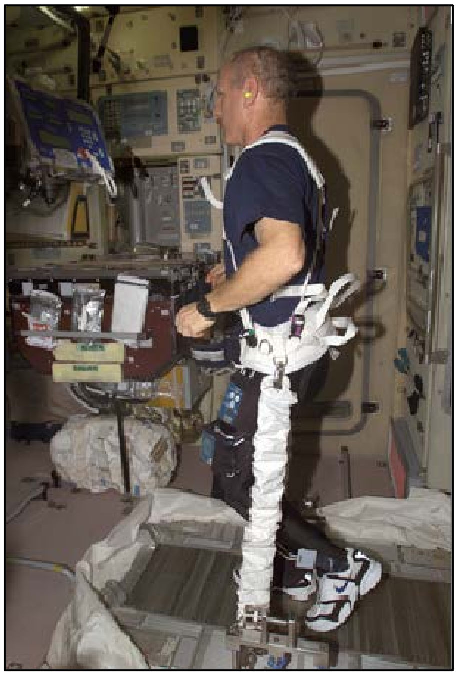


Figure 1-1: *Astronaut walking on a treadmill on the ISS [19]*

These health losses are becoming more prominent because astronauts are spending longer periods of time on the ISS, as compared to past missions. It is also the intention of NASA to have astronauts on the moon and asteroids for longer durations in the future.

Studies have been performed by the ECP to formulate a series of exercises similar to what is shown in **Figure 1-1** to be performed in the space craft to maintain a level of fitness for the astronaut. Further work is necessary to consider the impact that ordinary mission-critical tasks have on the fitness of the astronaut, in addition to the prescribed exercises. The KIN Dept.'s research explores the relationship between specific body performances and astronaut-type tasks. With their results, the ECP will further understand the exertion of energy required of an astronaut while performing mission-critical tasks on the ISS, the moon, or an asteroid.

1.2 MoRGS System Requirements

The KIN Dept. set the preliminary system requirements for MoRGS using existing documents for similar systems at NASA, as well as the requirements for their research program. These system requirements include criteria such as cost, mobility, design/build cycle, maximum velocity, and load capacity, shown in **Table 1-1**.

Table 1-1: *System requirements*

Performance			**System**	
Load Capacity (lbs)	200 - 750		Power Supply	Portable
Velocity (m/s)	0.75		Structure	Easy Disassembly
Acceleration (m/s^2)	8.8		Footprint Size	20'x20'x20'
Lift Stroke (m)	3		Budget	$30,000
Gravity Offload (G)	0 - 1		Safety	Double Fault
Tracking Accuracy	Ideal		Design/Build Cycle	1 year
Change Parameters	Yes		Custom Fabrication	Limited

1.3 Existing Systems

The KIN Dept. requested a MoRGS from the AVS lab that was inexpensive, portable, ready for use in one year, and interchangeable to a variety of loads. Specific requirements of the project will be discussed in Section 1.5. The initial research of this project focused on finding what sort of human offloading systems were already available and understanding if any of these systems were suitable for MoRGS. This investigation proved to be challenging because several of these systems were industry proprietary or NASA restricted.

1.3.1 Intelligent Assist Devices

Industrial lift devices like cranes, hoists, and winches have been commonplace in factories and plants for decades. In order to advance technology and reduce employee workload, industrial facilities have adopted IADs. IADs started as a research venture between Northwestern University and Stanley Assembly Technologies to improve efficiency in automotive factories [5]. They combined a haptic (touch) device with an industrial hoist in order to reduce the physical strain of workers moving and manipulating large loads. In this system, the steel wire rope is wound about a drum connected to an AC servo motor by a high gear ratio gearbox. The research project was successful, and Stanley introduced the *Cobotics* series of IADs in 1998.

A load cell integrated in the hoist handle senses changes in force applied by the user to raise and lower the hoist. A "float" mode allows the user to apply the force to the object after the IAD has a reading for the initial weight of the load [22]. IADs were well received by the material handling industry for its breakthrough in removing the operator from physical exertion. The Stanley *iLift* and the Gorbel *G-Force* IADs both offer great features for an industrial environment, such as virtual limits, gentle setdown, and dump mode.

Table 1-2: *The performance of off-the-shelf IADs was inadequate (Adapted from [12][22])*

Device	Load Capacity (lbs)	Velocity (m/s)	Acceleration (m/s^2)	Lift Stroke (m)	Voltage
MoRGS Require	750	0.75	8.8	3	48 DC
Stanley *iLift*	500	0.7	4.9	3	220 VAC
Gorbel *G-Force*	660	0.19	--*	2.44	220 VAC

* Unknown performance

The performance goals of the Stanley and Gorbel products were designed for manipulating loads in industrial applications [12][22]. **Table 1-2** compares the performance requirements of the MoRGS to the available off-the-shelf systems. The Stanley *iLift* and the Gorbel *G-Force* offer an adequate wire rope stroke but are unsatisfactory in all other areas. The Gorbel *G-Force* offers a higher load capacity unit, 1320 lb. [12], but the speed and acceleration performance is less than the 660 lb.

capacity unit. Another downside to these IADs is that they are not configurable to different offloading scenarios, for example the moon versus an asteroid. Finally, when approached with the possibility of using these devices with humans, both companies stated they would not sell the device for MoRGS if they knew that was its purpose. For these reasons, the Stanley *iLift* and the Gorbel *G-Force* were not selected as the hoist for the MoRGS.

1.3.2 ARGOS System

NASA's Johnson Space Center is home to ARGOS [8]. The Dynamic Systems Test Branch developed this system using many of the components from the Stanley *iLift* system. ARGOS can handle loads up to 250 lb. after modifying the original Stanley gearbox to increase the speed of the system. One useful piece of the ARGOS design is the use of a PMA which acts as a spring and damper. It is an off the shelf product that acts as a buffer between the human and the motor. The spring and damper coefficients are modified by varying the pressure in the sealed rubber membrane. More discussion on the PMA is in Section 2.4.2.

The engineers with NASA were not satisfied with the completed ARGOS [9]. They claimed the ARGOS lift was best for heavy, slow lifting and were unhappy with its performance with a "shirt-sleeved" person, meaning an individual without the space suit on, at a load less than 200 lb. The ARGOS structure was designed for a natural frequency of 12-14 Hz, and NASA resolved that this was not adequate. It needed to be stiffer in order to increase the natural frequency. NASA has plans of decommissioning the ARGOS simulator and replacing it with a highly redesigned system. **Figure 1-2** shows how large the ARGOS is.

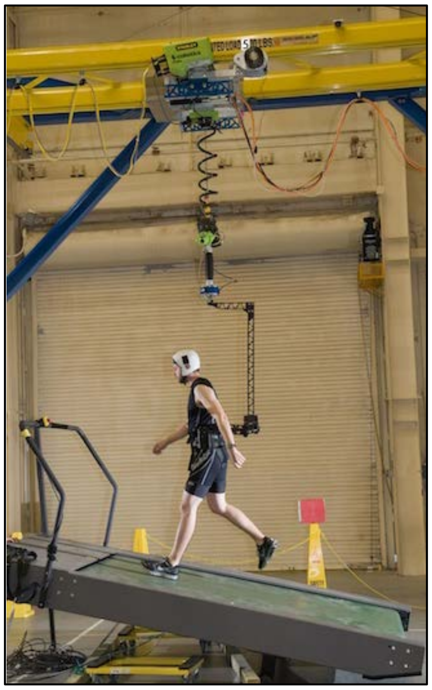


Figure 1-2: *NASA's ARGOS simulator [3]*

In addition to these faults cited by NASA, the ARGOS system did not meet several other fundamental requirements of MoRGS. First, ARGOS was powered by 480 VAC and since MoRGS needed to be portable, this was not an option. Second, Stanley refused to sell a modified *iLift* (like they did for NASA) in the future that would be used for any form of human offloading. Third, the structure's rail support system manufacturer also refused to sell parts for the same reason as Stanley. Lastly, the support structure of ARGOS is 41 ft. by 24 ft. by 18.5 ft. KSU does not have available space to house such a large structure. Therefore, a copy of the ARGOS v1 design was not a viable option, evidenced by **Table 1-3**. The resources invested in the ARGOS program were unreasonable to recreate at KSU. The budget for ARGOS was 3-4 times greater than MoRGS. Also, a team of 4-5 engineers developed ARGOS, whereas MoRGS only had one full-time engineering student.

Table 1-3: *Comparison of performance: MoRGS to ARGOS (Adapted from [8][9])*

Device	Load Capacity (lbs)	Velocity (m/s)	Acceleration (m/s^2)	Lift Stroke (m)	Voltage	Able to Copy Design
MoRGS Require	750	0.75	8.8	3	xx DC	-
ARGOS v1	250	1	4.9	3	480 VAC	No
ARGOS v2	750	1.25	9	3	480 VAC	No

The ARGOS engineers are developing a second generation of the system rectifying all of the shortcomings listed above in **Table 1-3**. Unfortunately, these designs are not available to the general public until the system has been completed and the documents are approved for release [9].

1.3.3 POGO System

Before ARGOS was built, NASA used a Partial Gravity Simulator that was also known as POGO. POGO was first used in the Apollo era of NASA and received its name because the astronauts bounced in a manner similar to a pogo stick. POGO is unlike ARGOS because it is a passive offload device, meaning there are no motors and no sensors. The human is offloaded by a large piston reaching 30 ft. when fully extended. This piston has an air supply that balances the column of air. These are common in industrial devices, called air balancers, and products like Stanley's *iLift* was intended to replace such devices in industry. NASA built ARGOS to replace the POGO because of several dynamic limitations in POGO.

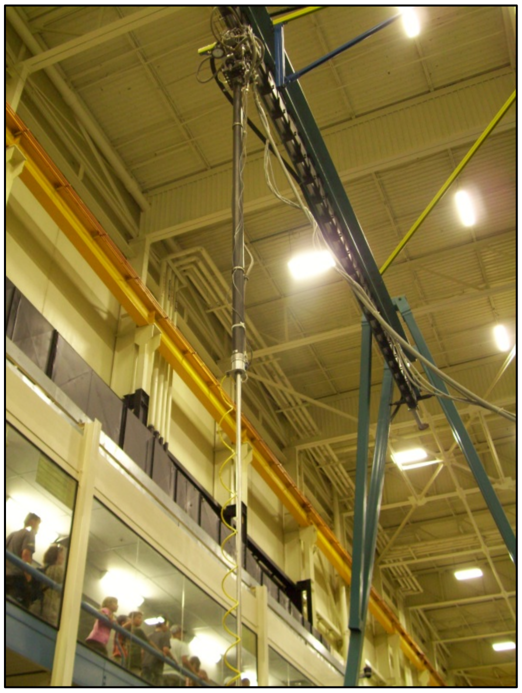


Figure 1-3: *The piston on the POGO simulator [1]*

The POGO system, shown in **Figure 1-3**, is a crude device. The air balancer does not have very good tracking characteristics to human inputs. In addition, NASA reported that the system impeded astronaut motion because of the inertia of the simulator [19]. The piston provided unwanted resistance to movement which gave inaccurate test results. Since the piston is rigidly mounted to the overhead beam (no rotational freedom), the astronaut was able to support themselves artificially from POGO, creating an unrealistic situation. In some instances, the astronaut modified their posture and running gait significantly to avoid this uncomfortable impedance. This system did allow horizontal travel in one direction thanks to an air-bearing system. However, this did not eliminate all friction because NASA also reported astronauts "pulling" POGO along the beam in tests.

Table 1-4: *Comparison of MoRGS requirements to POGO [19]*

Device	Load Capacity (lbs)	Velocity (m/s)	Acceleration (m/s^2)	Lift Stroke (m)	Voltage	Able to Copy Design
MoRGS Require	750	0.75	8.8	3	xx DC	-
POGO	450	-- *	-- *	5	N/A	No

* Unknown performance

POGO unfortunately does not have any data acquisition capabilities and therefore NASA does not know the actual velocity and acceleration of the system. Also, since the design was from the Apollo era, NASA does not have suitable design documentation to replicate the system. Therefore, the POGO system was not selected for MoRGS.

1.3.4 Spring Balance

The spring balance was considered as a means to minimize the size of the motor and power requirements significantly. The concept was based on a research project at the University of Colorado developing a reduced gravity simulator for small tools and mockups [21]. The hoist cable's drum was attached to a torsion spring that had a set preload for the static gravity offload. As the object moved, a small motor rotated the torsion spring to increase or decrease the torque applied to the cable's drum, therefore raising or lowering the hoist. This is shown in **Figure 1-4**.

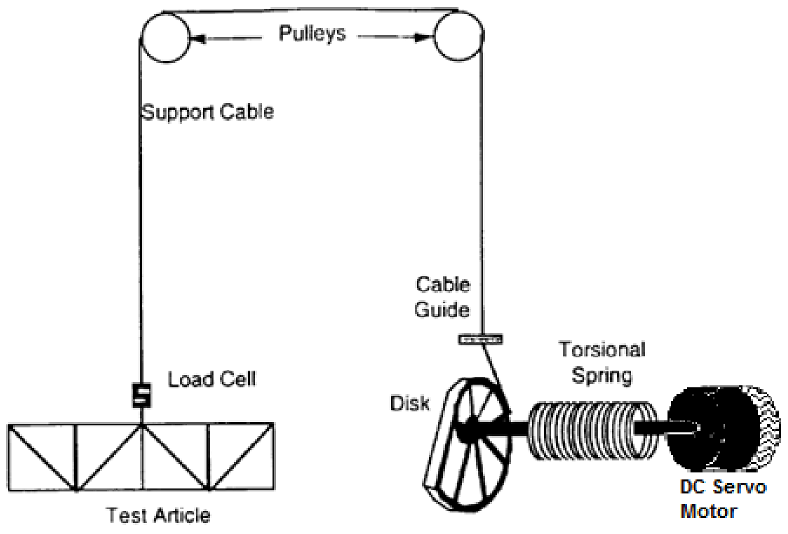


Figure 1-4: *The motor was the active control and the spring was the passive control [21]*

The combination of the torsion spring and a small servo motor seemed to be a viable option lending to further investigation into a torsion spring. The spring and motor setup at the University of Colorado had a maximum capacity of 100 lb. and a stroke of 1 m. The spring manufacturers that were contacted insisted there was no commercially available spring for the load and stroke desired. A custom spring would have to be designed to handle these extreme parameters, therefore the system could not be entirely copied.

The spring calculations were based on the equation from Shigley's Machine Design text for finding the torsion spring rate [4],

$$K_{per\ turn} = \frac{\Delta T}{\Delta \theta} = \frac{Ed^4}{10.8 D_m N_a} \tag{1.1}$$

where

$$N_a = N_b + N_e = N_b + \frac{L_1 + L_2}{3\pi D_m} \tag{1.2}$$

and

$$D_m = \frac{D_1 N_b}{N_b + \Delta\theta} \tag{1.3}$$

and the stress in the torsion spring due to bending,

$$S = \frac{32T}{\pi d^3} K_B \tag{1.4}$$

where

$$K_B = \frac{4C^2 - C - 1}{4C(C-1)} \tag{1.5}$$

The iterative calculation process converged on a compromise between the amount of load offloaded and the amount of travel possible.

Table 1-5: *The spring parameters became unrealistic*

Spring Parameters						
Material =	OQ&T wire			S_{ut} =	167.34	kpsi
Mean Coil Diam. D =	2.6	in		S_y =	145.59	kpsi
Wire Diam. d =	**0.5**	in		C =	5.20	
E =	30	psi*10^6		K_i =	1.17	
Spring Rate k' =	**494.80**	lb*in/turn		M_{max} =	1530.81	lbf*in
Pin Size D_p =	1	in		ϴ' =	**3.09**	turns
M_{max} =	1260	lb*in		ϴ'$_s$ =	1113.77	degrees
M_{min} =	0	lb*in		N_b =	134.95	turns
Drum Diam. =	12	in		D' =	2.54	in
Load =	**210**	lb		n_f =	**1.21**	FOS

The best combination of wire diameter and number of turns can be found in **Table 1-5**. In order to support the 750 lb. load, the torsion spring would have to increase in diameter over three times. With the increased cross-sectional area, the stiffness of the spring increases to match the spring rate k'. As a result, the number of turns decreases significantly and the spring becomes a stiff coiled rod that offers no rotation, ϴ'. One solution to this problem is to decrease the size of the spring and increase the size of the motor. However, as **Table 1-5** shows, a fraction of the desired load still requires a massive spring that is not available for manufacture.

1.4 Alternative Means of Reduced Gravity

In addition to the hoist-type concepts, several alternative methods of providing partial or zero gravity simulations have existed previously and/or still exist at NASA. These include neutral buoyancy, parabolic flight, and supine support. It is difficult to compare these to the hoist type systems directly. It was determined these systems simply do not meet the needs of the goals of the testing in the KIN Dept. However, they are described here for completeness.

Neutral buoyancy has become a common method of recreating zero-gravity for NASA testing. NASA's Neutral Buoyancy Lab is the largest pool in the world, holding 6.2 million gallons of water and offers a work area of 202 ft. in length, 102 ft. in width, and 40 ft. in depth [6]. Typically the tests run are practice sessions for mission-critical tasks that require access to full-scale mockups of space station components for future EVAs. The neutral buoyancy effect is generated by a combination of weights and flotation devices within the astronaut's suit to maintain a hover effect. Unfortunately, the astronaut does not feel true "weightlessness" [19].

Parabolic flight offers martian, lunar, and zero gravities and is the closest simulation to actual space conditions. *G-Force One* is NASA's Boeing 727 aircraft used for parabolic flights for tests in reduced gravity [23]. For safe flight and manageable Gs transition for humans, the peak of the parabola only lasts 20-25 seconds. After each parabola, all the occupants in the plane must lie on the floor of the gutted plane for the 1.8 G section of the flight path. Occupants have noted that they must keep their heads fixed or else they suddenly suffer from nausea and must make use of a vomit bag. This effect is why the *G-Force One* is nicknamed the "Vomit Comet".

Supine support is the process of rotating the test subject so the gravity vector is perpendicular to the relative Z-axis in the coordinate system of the person. The limbs are supported in a series of pulleys, shown in **Figure 1-5**.

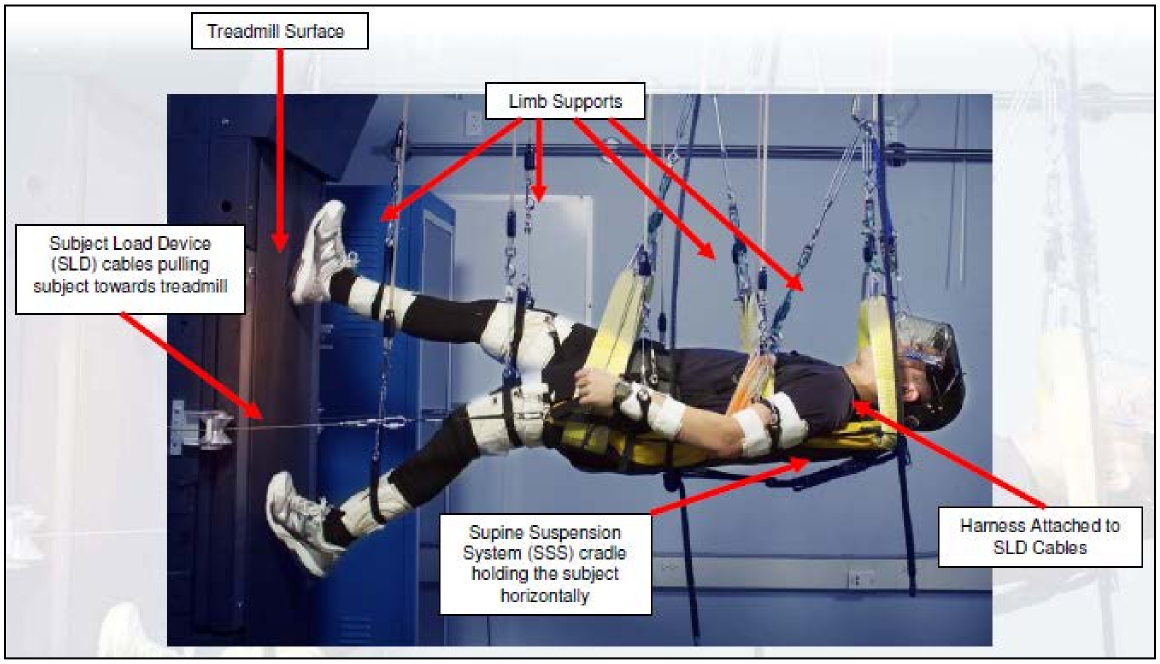

Treadmill Surface
Limb Supports
Subject Load Device (SLD) cables pulling subject towards treadmill
Supine Suspension System (SSS) cradle holding the subject horizontally
Harness Attached to SLD Cables

Figure 1-5: *A zero-G supine support system [20]*

For martian or moon gravity simulation, the person is rotated from the horizontal in order to generate a fraction of their terrestrial gravity, as realized by the person. This is shown in **Figure 1-6**.

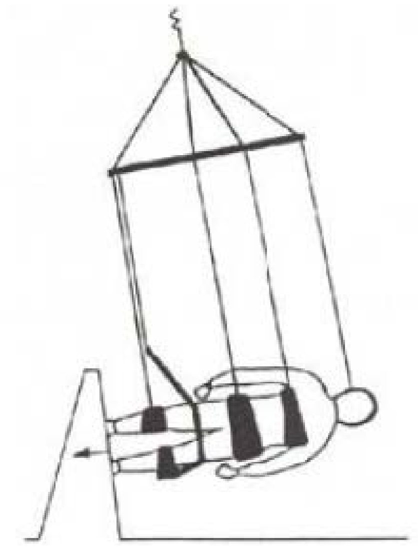


Figure 1-6: *A lunar gravity orientation [20]*

The supine support solution only allows linear locomotion on a treadmill-like device and does not permit the option of tests with mission-critical tasks like tool manipulation [20].

1.5 Conceptual Design

The performance and system requirements described in Section 1.2 make MoRGS a unique simulator that has neither an off-the-shelf product with equivalent performance nor any NASA test equipment with adequate system requirements. Therefore, a custom designed and built simulator was the only viable option for MoRGS. The final MoRGS concept combines features of IADs, ARGOS, and the Spring Balance while still considering the limited resources of the project. This concept, which currently only includes the hoisting portion of the MoRGS system, consists of four major components: a motor, a drum, a carriage, and a control computer.

A brushless DC (AC Servo) motor was selected because it offered the most flexibility compared to the other alternatives. The DC motor is capable of running on battery power, making it highly portable. The combinations of frame size and winding configuration create endless possibilities of RPM and torque tailored to MoRGS's requirements. A decision to design the system so that it could be powered by batteries allowed the use of larger motors. With a larger electrical current from the batteries, compared to many AC supply circuits, the brushless motor can supply the constant torque needed to offload the test subject. This eliminated the passive torsion spring. To meet the wide range of loads (200 – 750 lb.), the motor is mated with gearboxes of different gear ratios to achieve the torque for each load. By using off-the-shelf gear-heads for the gearbox, complicated drive lines, like on ARGOS, are eliminated. Lastly, the brushless motor provides the best control bandwidth and offers the best tracking of the test subject's motion as compared to the other solutions. Motor and gear-head sizing and selection can be found in Section 2.1.

One of the most complicated parts of the ARGOS design is the drum and its winding of the cable. To simplify this on MoRGS, the worm-drive cable guide mechanism on ARGOS was discarded and replaced with a larger diameter drum. The increase in diameter of the drum decreases the number of windings on the drum which minimizes the horizontal translation of the cable on the drum. Therefore, MoRGS has minimal horizontal translation, increasing the simplicity of angle position sensing, shown in **Figure 1-7**.

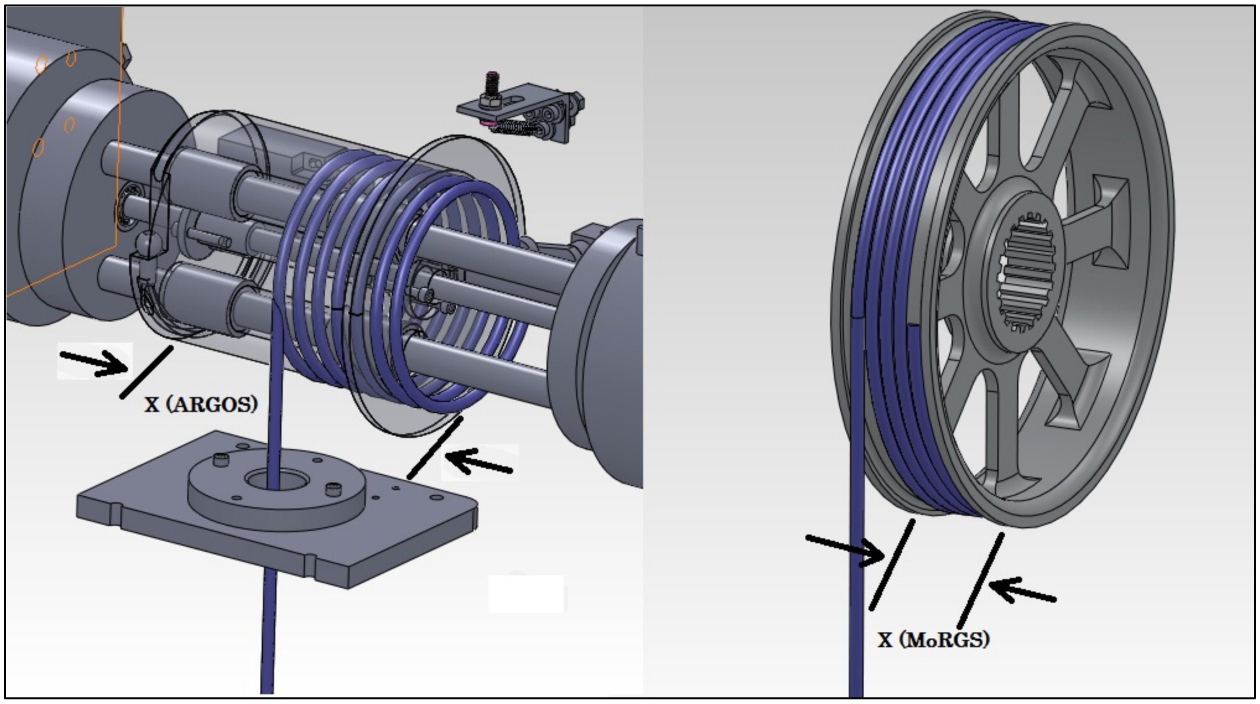


Figure 1-7: *The size of the ARGOS drum compared to the simple MoRGS concept*

The drum has a helical groove machined into its circumference that effectively guides the cable without overlap or binding, similar to the spool on a garage door cable system. The complete design analysis is in Section 2.2.

A simple box-like carriage concept was selected for its flexibility of mounting and its ease of manufacturing. The carriage connects the drum, gearbox, and the motor to the hoist support structure. The hoist structure is currently an I-beam and the carriage clamps to the lower flange with bolts. This carriage will be easily integrated with translation axes for the later phases of MoRGS. Mounting to other structures is possible with only a change in clamps. The carrier includes a series of rollers that provides uniform cable winding. To simplify, the carriage is made from five 2-D profile cutouts of aluminum, shown in **Figure 1-8**.

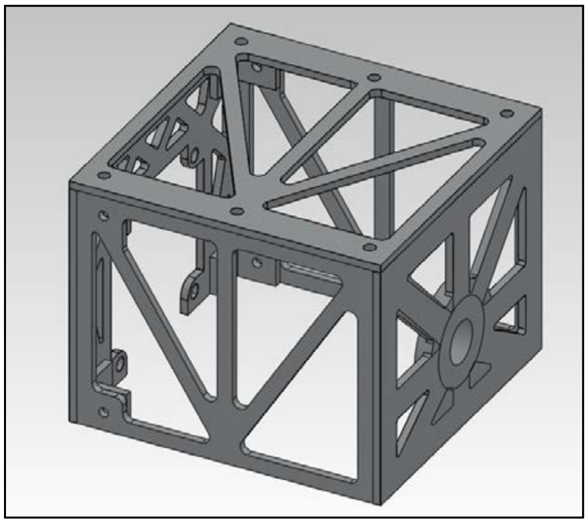


Figure 1-8: *The 2-D profile carriage*

The interface for the KIN Dept. operator needed to be interactive and intuitive. To minimize the number of digital devices, the operator interface also was combined with the force controller in the hoist. As a result, the processor needed several input and output capabilities in order to interact with the motor, load cell, and safety equipment. The ST Microelectronics STM32 F-4 microprocessor on an evaluation board was selected. This device offers an LCD display and push buttons to interact with the operator.

Serial communication sends commands to the motor drive and ADC reads the load cell. The F-4 chip has more than adequate processing power and speed to control MoRGS.

Chapter 2

Design

2.1 Motor

The motor sizing and selection was an iterative process with several variables involved. The flexibility of MoRGS' clean-slate design meant any combination of the following list of variables would be possible.

- Drum Size
- Voltage Supply
- Back EMF
- Voltage Constant
- Motor Velocity
- Constant Torque
- Cost
- Weight
- Availability
- Gear Ratios
- Load Inertia

2.1.1 Motor Selection

In order to compute the motor requirements for each of the different combination of parameters, a MATLAB m-file script was generated. This m-file script can be found in Appendix A. The motor requirements set by the KIN Dept. are shown in **Table 2-1**.

Table 2-1: *Motor performance requirements*

Performance	
Load Capacity (lbs)	200 - 750
Velocity (m/s)	0.75
Acceleration (m/s^2)	8.8
Power Supply	Portable

The drum diameter directly influenced the dynamic parameters of the motor and gear head. The final drum diameter selected was 9 in. and the resulting calculated dynamic parameters for each level of loading are in **Table 2-2**.

Table 2-2: *Drum influenced performance requirements*

Motion Profile	Level 1	Level 2	Level 3
Offload (lb.)	250	440	625
Angular Velocity (rad/s)	6.56	6.56	6.56
Angular Acceleration (rad/s^2)	77.33	77.33	77.33
Gear Ratio	3	3	3
Constant Torque (N-m)	42.4	74.58	105.77

The equations used for analysis of potential motor configurations started with a simple voltage loop where

$$V = IR + L\frac{dI}{dt} + K_b\omega \qquad (2.1)$$

For MoRGS, the goal for the peak voltage was to be no more than 85% of the supply voltage. The 15% difference between the peak back EMF voltage and the supply voltage allows the motor the capacity to control the torque and velocity. In order to evaluate if the motor meets the requirements during the maximum velocity and acceleration while maintaining within 85% of the supply voltage, the expression in (2.1) was modified to

$$0.85 \cdot V_{supply} > I_{max}R + K_b\omega_{max} \tag{2.2}$$

after including $L\frac{dI}{dt}$ in the supply voltage buffer. Next, the maximum current from (2.2) is substituted with the maximum torque required by the motor by the expression

$$I_{max} = \frac{T_{max}}{K_T} \tag{2.3}$$

where

$$T_{max} = T_{constant} + J_{total}A_{max} \tag{2.4}$$

and where $T_{constant}$ is the continuous torque necessary to offload the human and J_{total} is the sum of the inertias of the motor, gearbox, and the drum. Since K_T and K_b are interchangeable when converted to the appropriate units, K_b was replaced with K_T, thus turning (2.2) into

$$0.85 \cdot V_{supply} > \frac{T_{max}}{K_T}R + K_T\omega_{max} \tag{2.5}$$

Next, (2.5) was set equal to zero

$$0 > \frac{T_{max}}{K_T}R + K_T\omega_{max} - 0.85 \cdot V_{supply} \tag{2.6}$$

To verify that the motor configuration met the velocity and torque requirements, the values of K_T satisfying (2.6) were found by finding the roots of

$$0 > [\omega_{max}]{K_T}^2 - [0.85 \cdot V_{supply}]K_T + [T_{max}R] \tag{2.7}$$

The two real roots of (2.7) were a lower and upper limit of possible values for the motor's torque constant that would meet the velocity and torque requirements. A motor with a torque constant above the upper limit would be capable of the maximum torque but not the maximum velocity. Meanwhile, a torque constant below the lower limit would be capable of the maximum velocity but not the maximum torque.

The results of the final motor configuration and its calculated performance are shown in **Table 2-3**. The motor selected was the Baldor BSM100N-3150-AB with the specifications listed in **Table 2-4** [3].

Table 2-3: *The selected motor meets acceleration requirements for all loads*

Performance Results	**Theoretical Check**			**Motor Check**
Load	250	440	625	
Torque Constant K_T (N-m/A)	0.34 - 1.84	0.66 - 1.81	1.1-1.82	1.79
Max. Torque (N-m)	47	79	110	136
Gear Ratio	3	3	3	
Peak Current (A)	26.3	44.3	61.1	75.87
Peak Acceleration (m/s^2)	8.8	8.8	6.6	
Peak Velocity (m/s)	0.71	0.63	0.54	

Table 2-4: *Specifications for Baldor BSM100N-3150-AB (Adapted from [3])*

General Parameters		**Electrical Parameters**	
Cont. Stall Torque (N-m)	34	Torque Constant (N-m/A)	1.79
Cont. Stall Current (A)	21.08	Voltage Constant (V_{pk}/kRPM)	153.2
Peak Torque (N-m)	136	Resistance (Ohms)	0.25
Peak Current (A)	75.87	Inductance (mHy)	2.74
Mechanical Parameters		**Other**	
Inertia (kg-cm^2)	30.8217	Resolver Feedback	
Max Speed (RPM)	4000	Brake	
No. of Motor Poles	8		

To maintain the simple design of MoRGS, the gearbox was selected to be an off the shelf gear head compatible with the Baldor motor selected. The required gear ratio of the gearbox was selected by the motor sizing equations above in Section 2.1.1 to be 3:1. The Baldor line of GBSM100 gearheads was selected because they are designed for the motor's frame size. Choosing the particular model was based on the maximum torque seen by the gearhead. The MRP155 model was chosen for all the load segments because its maximum torque is 139 N-m and the calculated maximum torque is 110 N-m [3].

2.2 Drum

The drum design had 3 goals: able to be manufactured at K-State, minimal rotational inertia, and effectively spool the wire rope. The drum is a multi-spoke wheel with a helical groove for the wire rope. Based on the motor performance requirements, the drum diameter is 9 in. The wire rope is 0.25 in. in diameter, as discussed in Section 2.4.1. The groove depth and the pitch of the wire rope on the drum follows the CMAA Specification #70 2004, same as ARGOS.

> The hoist drum grooves shall have a depth greater than or equal to .375 x rope diameter. The drum grooves shall have a pitch greater than or equal to either 1.14 x rope diameter or rope diameter + 1/8 inch, whichever is smaller. [9]

The groove depth on MoRGS' drum is 0.125 in. and the wire rope pitch is 0.285 in. conforming to CMAA Specification #70 2004. **Figure 2-1** shows the entire drum assembly.

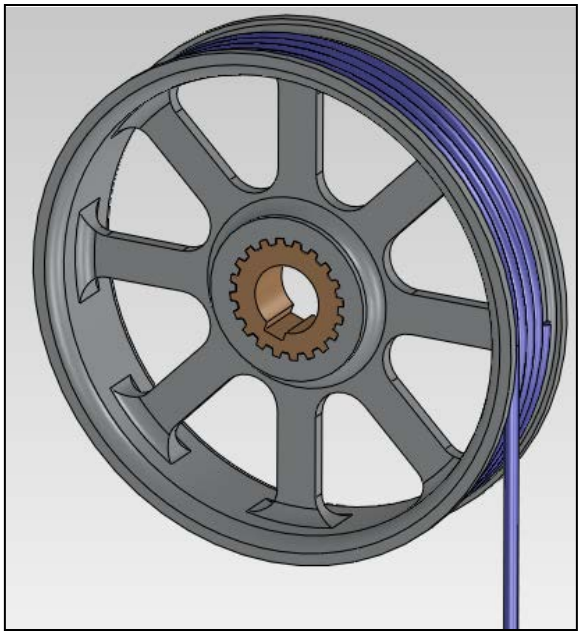
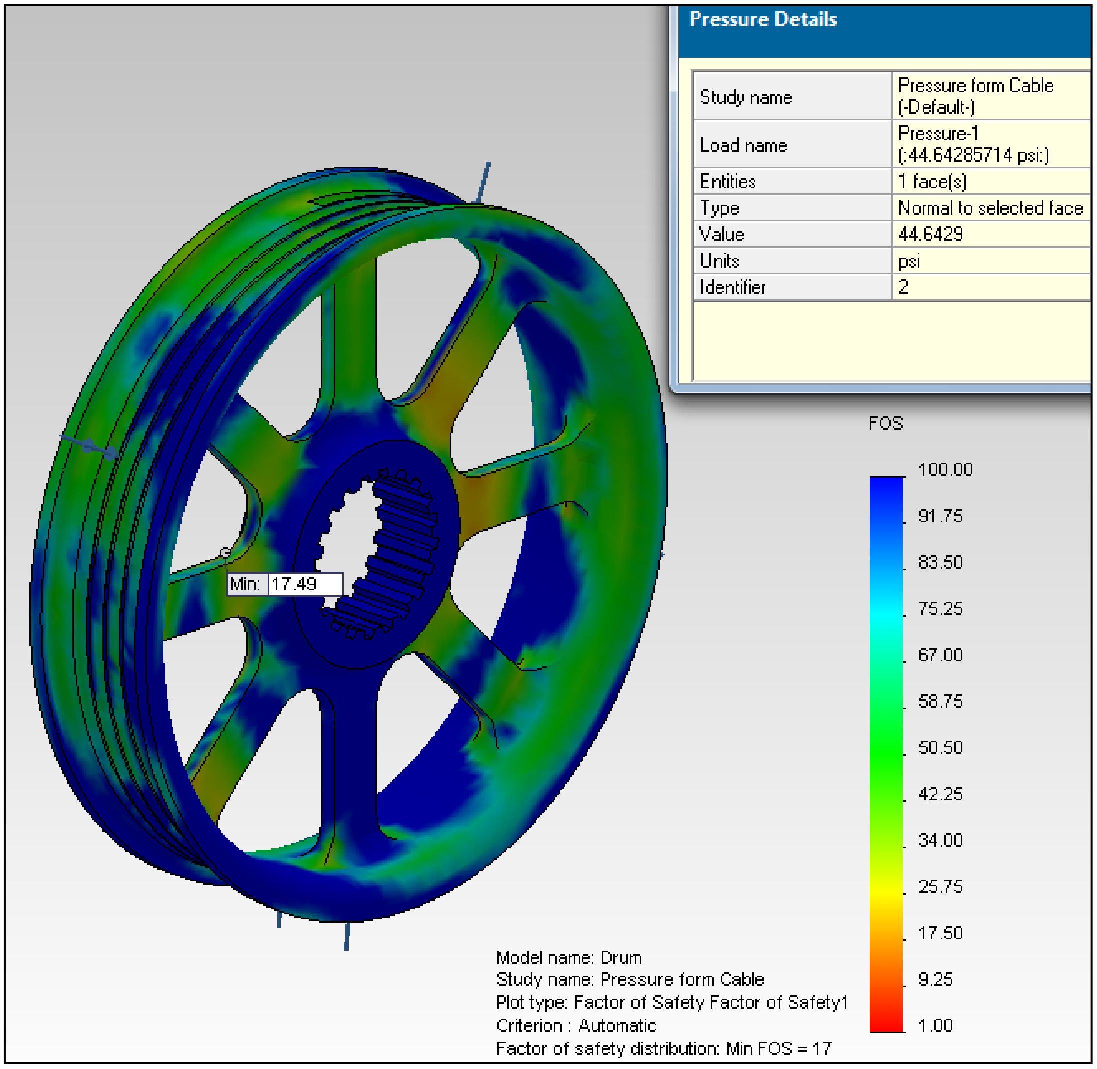


Figure 2-1: *Drum, splined gear insert, and wire rope model*

2.2.1 Cable Winding

The drum, along with the other custom built features of MoRGS, was modeled in SolidWorks CAD software and analyzed in SolidWorks Simulation FEA software. The material selected for the drum was 7075 Aluminum for its high stiffness to weight ratio and its ease of machining. The maximum load of 625 lb. was applied in all scenarios. First, the pressure exerted onto the circumference of the drum was analyzed. The pressure calculated and applied was 44.6 psi. The FOS of the drum was 17, which shows this is not the weak loading scenario for the drum, shown in **Figure 2-2**.



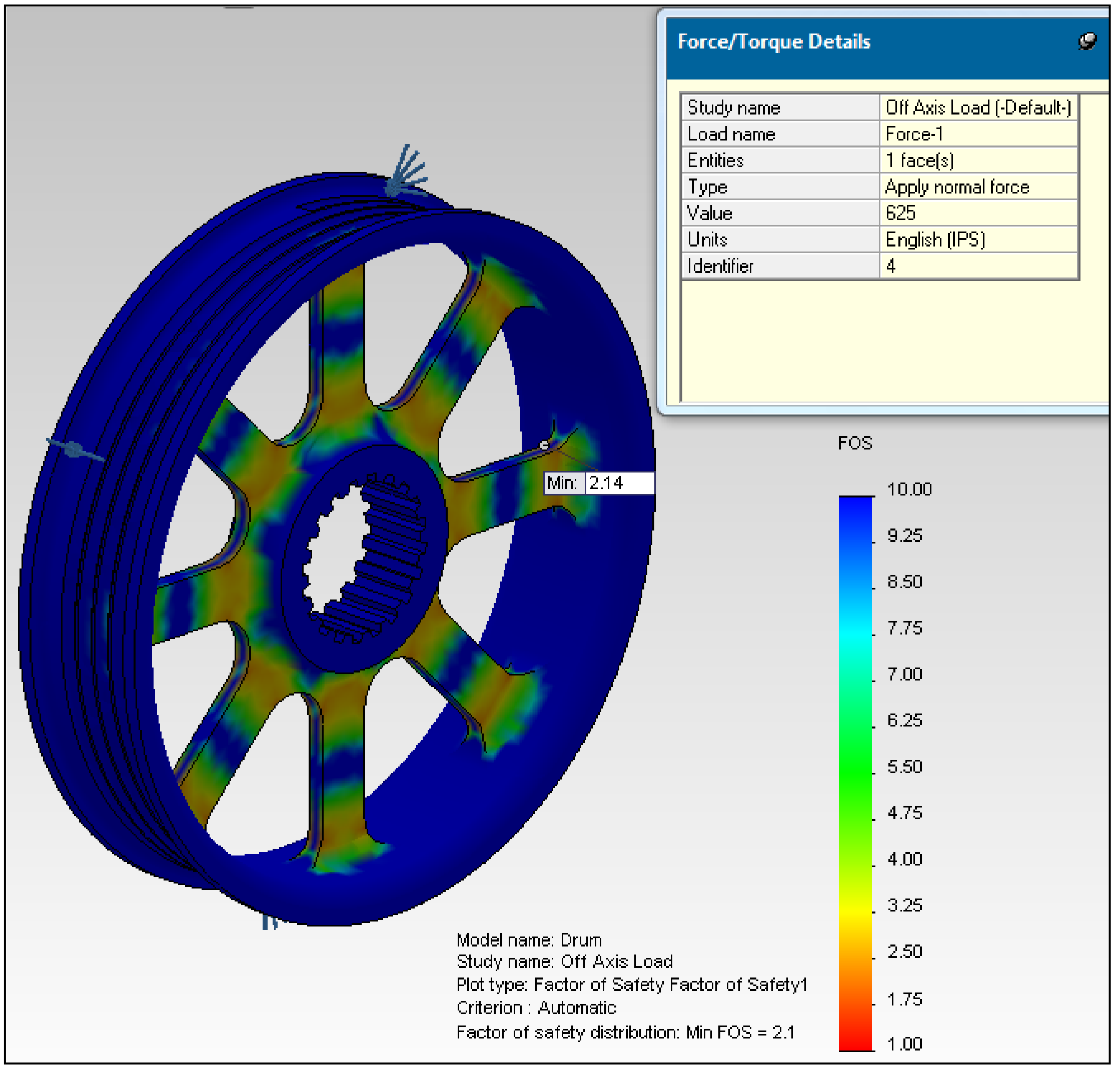
Pressure Details

Study name	Pressure form Cable (-Default-)
Load name	Pressure-1 (:44.64285714 psi:)
Entities	1 face(s)
Type	Normal to selected face
Value	44.6429
Units	psi
Identifier	2

Model name: Drum
Study name: Pressure form Cable
Plot type: Factor of Safety Factor of Safety1
Criterion : Automatic
Factor of safety distribution: Min FOS = 17

Figure 2-2: *Pressure from the wrapped wire rope analysis*

The pressure applied scenario was not a limiting factor to the strength of the drum. The next loading situation was the off-axis load where the wire cable is nearly unspooled completely and the 625 lb. load is applied on the edge of the drum. To investigate the lateral stiffness, this loading also was applied laterally along the axis of rotation. This extreme loading simulated a worst case side loading and it survived with a FOS of 2.1. **Figure 2-3** shows the flow of stress from the rigid drum center to the perimeter through the spokes. The fillets along the spokes were added to better transition this flow of stress and reduce the number of stress concentration factors.



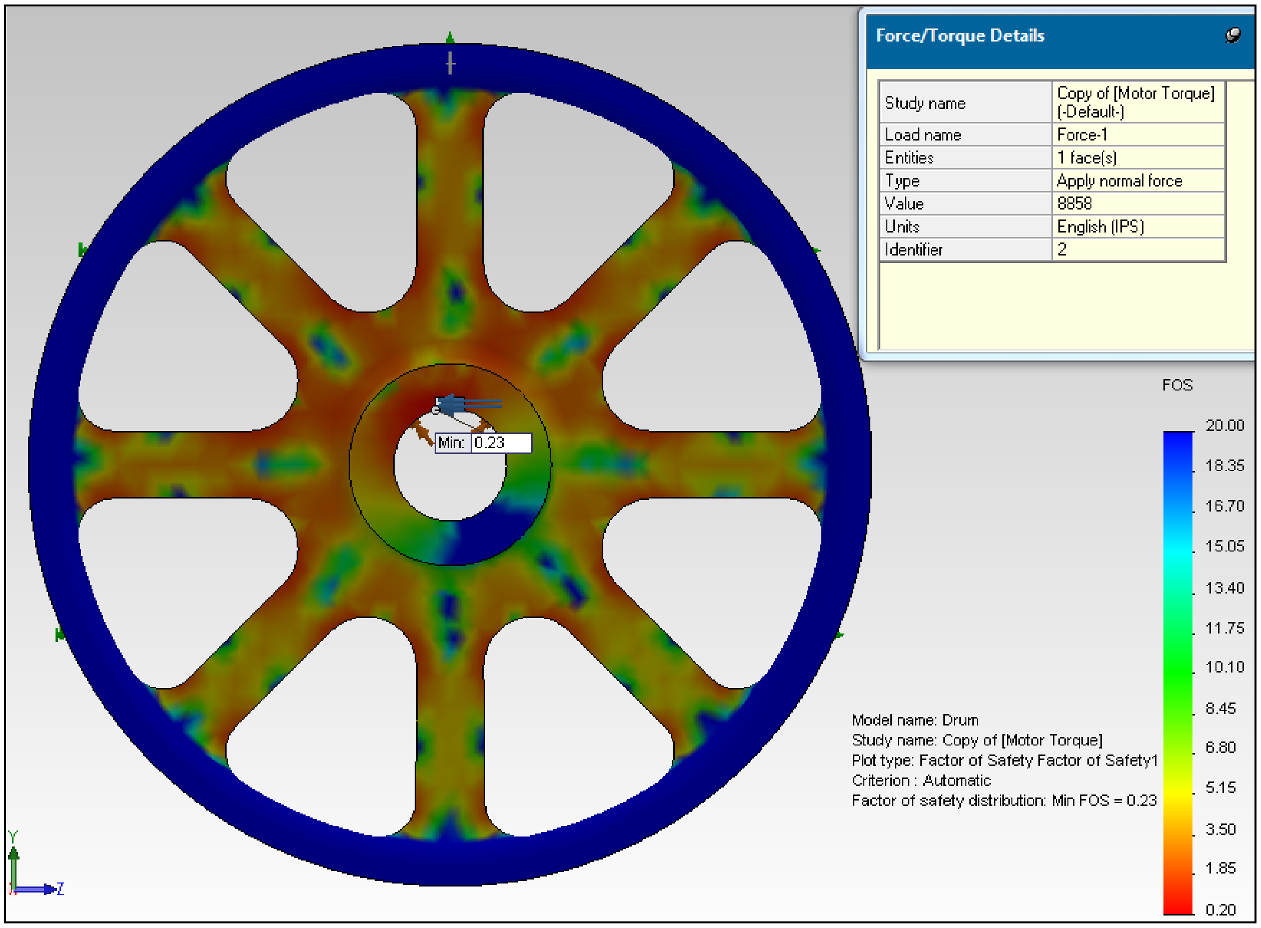
Force/Torque Details	
Study name	Off Axis Load (-Default-)
Load name	Force-1
Entities	1 face(s)
Type	Apply normal force
Value	625
Units	English (IPS)
Identifier	4

Model name: Drum
Study name: Off Axis Load
Plot type: Factor of Safety Factor of Safety1
Criterion : Automatic
Factor of safety distribution: Min FOS = 2.1

Figure 2-3: *Complete off-axis loading of drum*

2.2.2 Mating with Gearbox

The largest magnitude of force applied to the drum is when the motor accelerates to raise the hoist at maximum torque. At the instant of applied torque, the perimeter of the drum is fixed by the wire rope. This simulation was ran with a keyway machined into the aluminum center of the drum. Unfortunately, the drum did not adequately distribute the motor torque and caused extreme stress concentration at the keyway resulting in a FOS of 0.23, shown in **Figure 2-4**.



Force/Torque Details

Study name	Copy of [Motor Torque] (-Default-)
Load name	Force-1
Entities	1 face(s)
Type	Apply normal force
Value	8858
Units	English (IPS)
Identifier	2

Model name: Drum
Study name: Copy of [Motor Torque]
Plot type: Factor of Safety Factor of Safety1
Criterion : Automatic
Factor of safety distribution: Min FOS = 0.23

Figure 2-4: *Failed keyway shaft from motor torque*

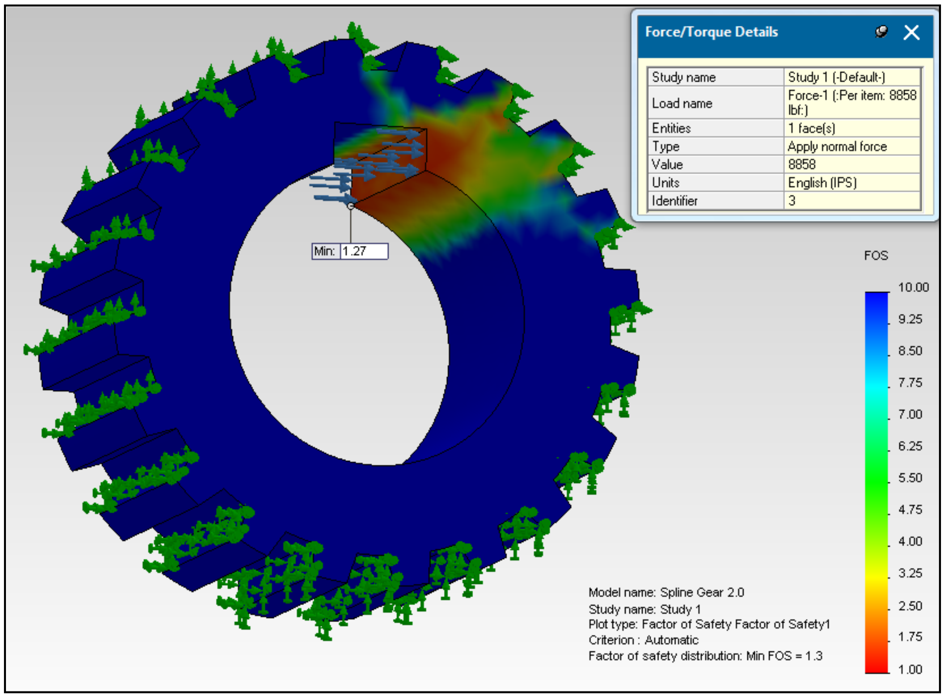
The off the shelf gearbox output shaft has a key in a steel shaft to transmit the torque and was not easily modified. Instead, the drum design had to be redesigned to withstand these higher loads from the motor. It was understood that the keyway slot created a single contact area, increasing the stress concentration factor. As a result, increasing the contact points and therefore the area would minimize the stress concentration and distribute it across the entire drum center. A steel insert with the inner diameter machined with a keyway slot for the gearbox and the outer diameter machined with a high tooth-count spline would carry the applied torque.

A purchased splined gear shaft was selected with the teeth already machined in the hardened 4130 Steel. The inner diameter keyway slot was later machined at KSU. The information on the final gear shaft is shown in **Table 2-5**.

Table 2-5: *Specification for splined gear shaft [15]*

Grob P/N 2125-20-2-4140	
Nominal Size (in.)	2.125
Number of Teeth	20
Chord Max (in.)	0.496
Chord Min (in.)	0.495
Over Teeth	2
OD Max. (in.)	2.255
OD Min (in.)	2.250
Root Diam. Max. (in.)	2.000
Root Diam. Min. (in.)	1.995
Tool Number	0320
Material Steel	4140
Length (in.)	1.000

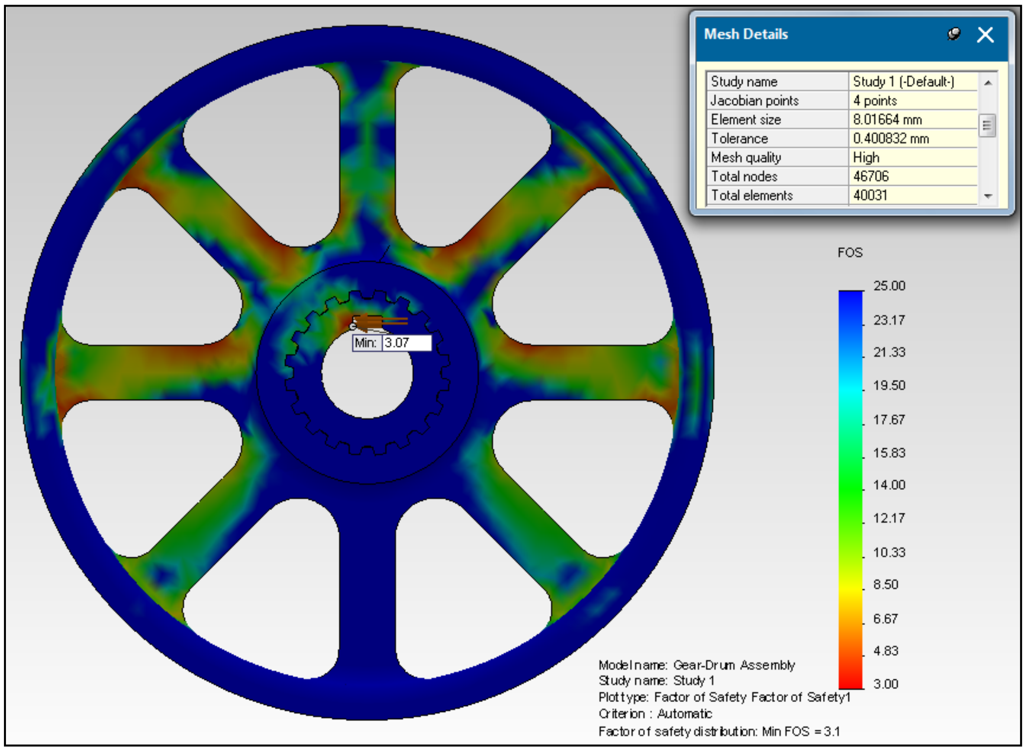
The finite element analysis reveals a FOS for the gear at 1.27 which is below the specified FOS of 3 for the MoRGS system, shown in **Figure 2-5**. But the teeth on the gear are not infinitely stiff as it is simulated, instead, the mating aluminum teeth on the drum absorb the force because they displace. Therefore, the mating of the drum and insert gear decrease the stress in the gear. **Figure 2-6** shows the stress distribution of the drum and the gear as an assembly. The FOS in the gear and the drum now meet the specified minimum FOS of 3.



Study name	Study 1 (-Default-)
Load name	Force-1 (:Per item: 8858 lbf:)
Entities	1 face(s)
Type	Apply normal force
Value	8858
Units	English (IPS)
Identifier	3

Model name: Spline Gear 2.0
Study name: Study 1
Plot type: Factor of Safety Factor of Safety1
Criterion : Automatic
Factor of safety distribution: Min FOS = 1.3

Figure 2-5: *Part analysis of spline gear insert*



Study name	Study 1 (-Default-)
Jacobian points	4 points
Element size	8.01664 mm
Tolerance	0.400832 mm
Mesh quality	High
Total nodes	46706
Total elements	40031

Model name: Gear-Drum Assembly
Study name: Study 1
Plot type: Factor of Safety Factor of Safety1
Criterion : Automatic
Factor of safety distribution: Min FOS = 3.1

Figure 2-6: *Analysis of gear and drum passes FOS of 3*

The drum was CNC machined at the KSU's AMI and then sent to the gear manufacturer where the outer mating spline was broached in the aluminum center. The inner diameter on the splined gear was machined to the outer diameter of the gearbox shaft and then broached with a key way broach on a hydraulic press.

2.3 Carriage

The carriage or housing for the MoRGS hoist has the task of bringing all the different elements together into one compact package. The carriage needed to be compact so that the moments generated by the load transferred from the drum to the gearbox and from the gearbox to the I-beam was minimized. Also the carriage needed to be simple to manufacture at K-State and easy to mount on various I-beam like structures. Lastly, weight savings was under consideration for the carriage if MoRGS added an X and Y axis. **Figure 2-7** shows the hoist layout.

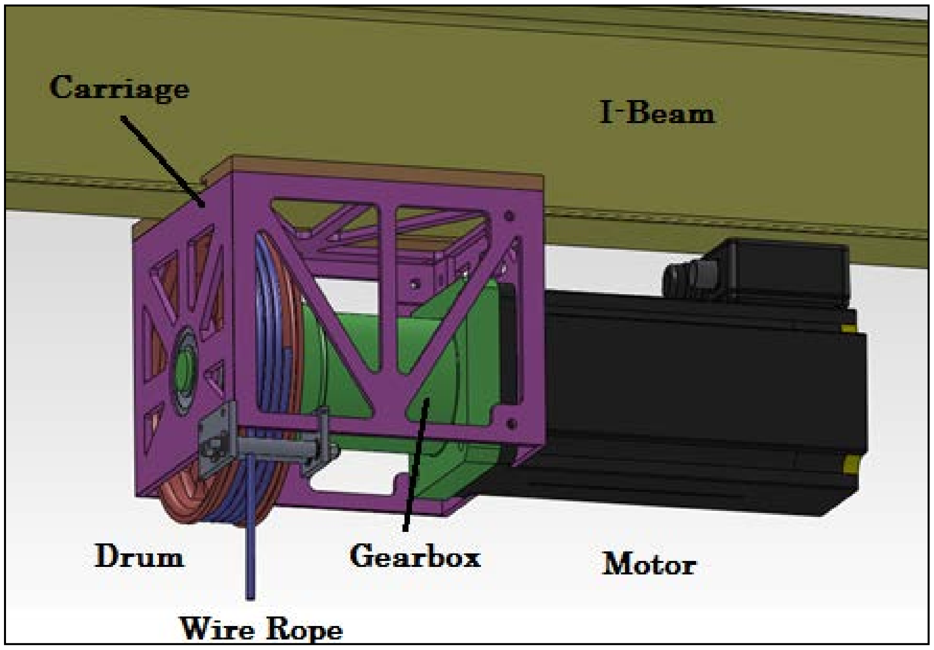
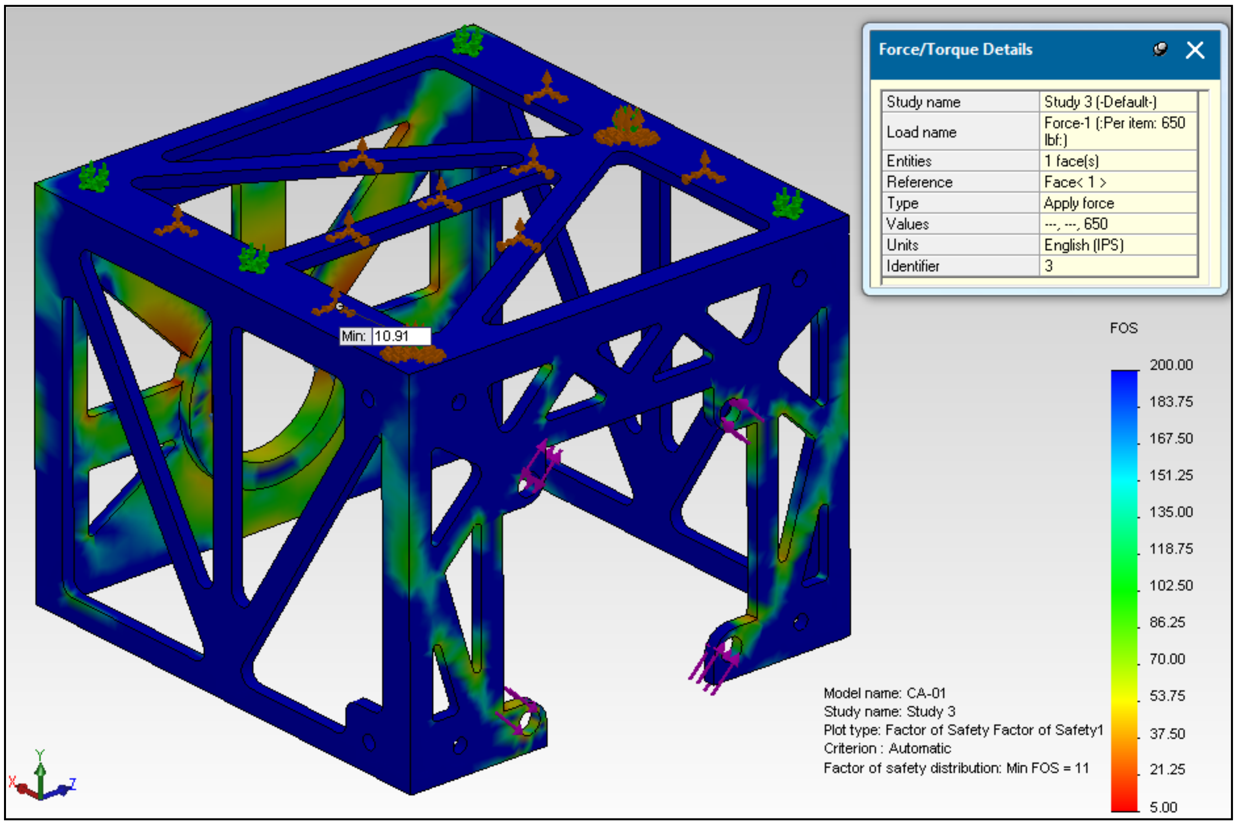


Figure 2-7: *Complete MoRGS hoist assembly*

2.3.1 Mounting of Motor, Gearbox, and Drum

The motor only has a four bolt mounting pattern perpendicular to the drive shaft axis. The gearbox bolted directly to the four bolt mounting pattern on the motor and as a result, the mounting schemes were limited. The configuration selected placed the drive shaft of the gearbox in a roller bearing opposite the drum. The carriage also bolted to the four bolt mounting pattern on the motor opposite of the gearbox putting the hoist in double shear. The carrier was water-jet cut at KSU from 0.25 in. thick 7075 Aluminum plate in 2-D profiles, simplifying the manufacturing. The four sides away from the motor were welded together and then bolted with "L" brackets to the profile attached to the motor. The roller bearing was lastly pressed into the carrier.



Force/Torque Details

Study name	Study 3 (-Default-)
Load name	Force-1 (:Per item: 650 lbf:)
Entities	1 face(s)
Reference	Face< 1 >
Type	Apply force
Values	---, ---, 650
Units	English (IPS)
Identifier	3

Model name: CA-01
Study name: Study 3
Plot type: Factor of Safety Factor of Safety1
Criterion : Automatic
Factor of safety distribution: Min FOS = 11

Figure 2-8: *FEA analysis result for the carrier*

The carrier was analyzed like the drum with finite element analysis. The part was fixed at the I-beam contact area with zero displacement. The vertical load from the test subject was applied as a force balance between the bearing race and the opposite side motor mount. In addition, the torque from the motor was distributed to the four bolts at the motor mount tangent to the motor's axis of rotation. **Figure 2-8** shows the complex loading of the carrier and its FOS of 11. The bearing on the gearbox shaft is 32 mm inner diameter and upgraded from a ball bearing to a roller bearing to withstand the 625 lb. load. In order to lower the stress in the carrier, the longest available roller bearing was selected, which was 1 in.

2.3.2 Rollers

Two hardened steel rollers were added where the wire rope extends from the bottom of the carrier. The rollers serve three purposes. First, the rollers guide the wire rope smoothly on and off the drum's groove and prevent the wire rope from skipping its groove. Second, the rollers keep the wire cable from rubbing against the carrier when the test subject is not directly below the hoist. Third, the rollers position the wire rope to exit the carrier assembly precisely along the center axis of the I-beam. Originally the wire rope was extended off-axis from the I-beam but this caused unsatisfactory torsion in the I-beam and lowered the beam's natural frequency. This lower natural frequency limited the bandwidth of the controller and therefore was modified to eliminate the torsional loading of the I-beam.

The translation of the rollers from the off-axis position to the on-axis position with respect to the I-beam dramatically increased the lateral cyclic loading on the rollers from the wire rope. A fatigue calculation was made to insure proper long term functionality of the rollers. The Goodman equation [4] was used for evaluating the number of cycles.

$$n = e^{{}^{1}/_{y} * ln {}^{K_E}/_{C_E}} \tag{2.8}$$

The endurance stress factor, K_E, was computed from

$$K_E = \frac{K_U(^{S_2}/_{S_U} - ^{S_1}/_{S_U})}{2K_U - (^{S_2}/_{S_U} + ^{S_1}/_{S_U})} \tag{2.9}$$

where K_U is equal to the ultimate strength factor, S_2 is the maximum stress, S_1 is the minimum stress, and S_U is the ultimate stress of the material. The maximum and minimum stresses were calculated under the assumptions that ¼ of the roller's circumference is in contact with the wire rope and the wire rope's contact patch is 50% of its diameter, 0.125 in. The complete stress and cycle life calculation is shown in **Table 2-6**. The product life of the rollers was also computed based on an average of 1 roller revolution per second, for 2 hours a day, for 15 days in a month, equates to 11 months of operation before the rollers should be changed.

Table 2-6: *The fatigue life of the rollers is 11 months with a 200 lb. load*

Parameter			Computation		
S_1	-4074	psi	K_E	8.46E-02	
S_2	4074	psi	n	1156706	cycles
S_U	5.60E+04	psi	Life	10.7	months
K_U	0.52122				
C_E	0.4579				
Y	-0.121				

The current rollers are rated for 200 lb. When the complete suited astronaut is applied, larger rollers will be required. The manufacturing of the rollers began as steel rod was turned down to fit the bearing races and cut to the width of the drum. The ends of the rollers were tapped and nuts were added to keep the assembly together. The rollers are mounted in sets of bearings pressed into aluminum plates. The plates are bolted to the bottom of the carrier. **Figure 2-9** shows the layout of the rollers.

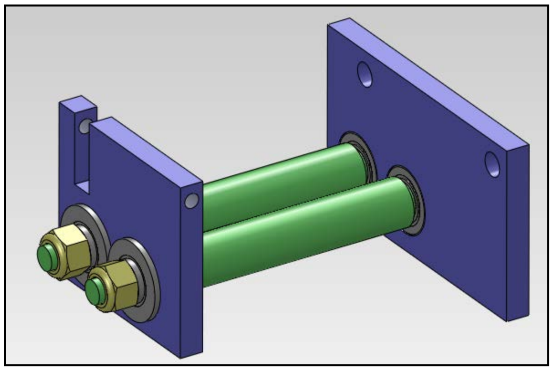


Figure 2-9: *Custom built rollers*

2.3.3 I-Beam

Hoist style I-beams are typically rated by the number of tons it can accommodate. The 625 lb. maximum load, with a FOS of 3, equates to a 2-ton I-beam. The other concern is the natural frequency of the hoist and its support structure. MoRGS has been modeled to have a natural frequency of 3 Hz based on the assumptions made about the dynamics of the test subject. For controllability of the system, an order of magnitude for the structure's natural frequency of 30 Hz was a design goal. The equation used for evaluating the I-beams assumes the beam is simply supported on both ends [5]. Solving for f_n

$$f_n = \left(\frac{1}{2\pi}\right)\left(\frac{n\pi}{L}\right)^2 \sqrt{\frac{EI}{\rho}} \tag{2.10}$$

where n = 1,2,3..., L = length of beam, E = modulus of elasticity, I = moment of inertia, and ρ = linear density. The beam selected was a W15x42 beam, meaning wide flange style, 15 in. height and 42 lb. per foot. The natural frequency result was 33 Hz, within the range of the design goal, at 625 lb. **Table 2-7** shows the parameters used for the calculation of (2.10) and the comparison of alternatives. The W15x42 was selected because it offered the highest stiffness to weight ratio.

Table 2-7: *Comparison of I-beams*

I-beam Style	ρ (lb/ft)	Ixx (ft^4)	E (lb/ft^2)	L (ft)	f_n (Hz)
W10x21	25.4	0.0053	4.25E+10	15	20.79441012
W12x31	31	0.0105	4.25E+10	15	26.48155087
W15x42	42	0.0216	4.25E+10	15	32.6311097

To support the W15x42 I-beam, a fixed height gantry structure has been selected. This off the shelf structure is bolted to the floor to increase its stiffness. The structure height was determined by the building ceiling in which it is housed. The height of the structure is 17 ft. [13].

2.4 Mechanical Components

MoRGS has several smaller components that connect the test subject to the torque at the drum from the motor.

2.4.1 Hoist Cable

The wire rope (hoist cable) was sized and installed according to NASA-STD-8719.9 Change 1 Para 4.2.6 [5]. This NASA specification states that

> 3.6.3.1.3 The hoist system shall have a factor of safety 3 in reference to yield stress for bending and contact stresses on all gears in the critical lift path as calculated by AGMA 2001-D04.
> 3.6.5.2 The rope ends shall be anchored securely by a clamp or a swaged terminal in a keyhole slot.

To insure reliability of the wire rope selected, Mil-W-83140 (for Aircraft Rescue Hoist and Cargo Handling, Type 1) standard was used. The wire rope style chosen was 19X7 Nonrotating Preformed Stainless Steel Wire Rope T302/304. The ¼ in. nominal diameter was selected to provide adequate margin of safety for the 625 lb. loading scenario. The minimum breaking strength of the wire rope is 5,760 lb., therefore equaling a FOS of 4.6 when a 2 G load is applied. The loop termination was completed by the manufacturer. The opposite termination at the drum was performed by an oval F-6 zinc plated copper stop sleeve using a 3V-F6:X:M Nicopress tool. This sleeve is specifically made for ¼ in. 7x19 steel wire rope to insure reliability.

2.4.2 Pneumatic Muscle

The PMA is a vital part of MoRGS because it acts as a low pass filter of the test subject's motions, reducing high frequency coupling (spring) and reducing the oscillations between the spring and the load (damper). The PMA selection was derived from the ARGOS system and NASA's extensive testing of the Festo style of PMAs [5]. NASA sent out five PMAs for professional testing in order to evaluate the energy absorption and breaking strength. The PMAs all failed by a load in excess of 5000 lb. which equates to a FOS of over 5. NASA's energy calculations prove that even with an 8 G stop (when the motor brake is applied in an emergency) it will not fail.

Festo's design software, MuscleSIM, was utilized in selecting the particular PMA model. **Table 2-8** shows the parameters used in the simulator, resulting in selecting a Festo MAS-20-300N-AA model.

Table 2-8: *Simulation parameters of PMA*

Parameter	Value	Units
Nominal Length	300	mm
Stroke	60	mm
Force in State 1	5751	N
Force in State 2	181	N
Muscle Diameter	20	mm
Degree of Contraction	20	%
Constant Pressure	3.5	bar

Next, the particular spring constant and damping constant were evaluated, since both are dependent on pressure in the PMA. The PMA has a sealed pressure chamber set at 50 psi, the optimal pressure based on the MuscleSIM simulation. The PMA's K$_{PMA}$, spring constant, and B$_{PMA}$, damping constant, are characterized using a three-element phenomenological model based on the following equations [9]

$$K_{PMA} = \begin{cases} 32.7 - 0.0321P & if\ 150 < P < 314\ kPa \\ 17.0 + 0.0179P & if\ 314 < P < 550\ kPa \end{cases} \tag{2.11}$$

and

$$B_{PMA} = \begin{cases} 2.90 & contraction \\ 1.57 \quad if\ 150 < P < 372 & relaxation \\ 0.311 + 0.00338P \quad if\ 372 < P < 550 & relaxation \end{cases} \tag{2.12}$$

where K$_{PMA}$ is in units of $\frac{N}{mm}$ and B$_{PMA}$ is in units of $\frac{Ns}{mm}$. Therefore, at 50 psi (345 kPa), K$_{PMA}$ equals 23175.5 N/m and B$_{PMA}$ equals 2900 Ns/m in contraction and 1570 Ns/m in relaxation mode. These parameters are used in the modeling of the system.

2.4.3 Harness Interface

The KIN Dept. is responsible for the harness that connects the test subject to MoRGS. An industrial spring-loaded hook offers the most flexibility with the harness selected by the KIN Dept. This hook is welded to a Grade 8 high-strength bolt that threads into the load cell.

2.5 Electrical Components

The electrical system has four main components: power supply, motor controller, load cell, and microprocessor. **Figure 2-10** shows the entire electrical system layout.

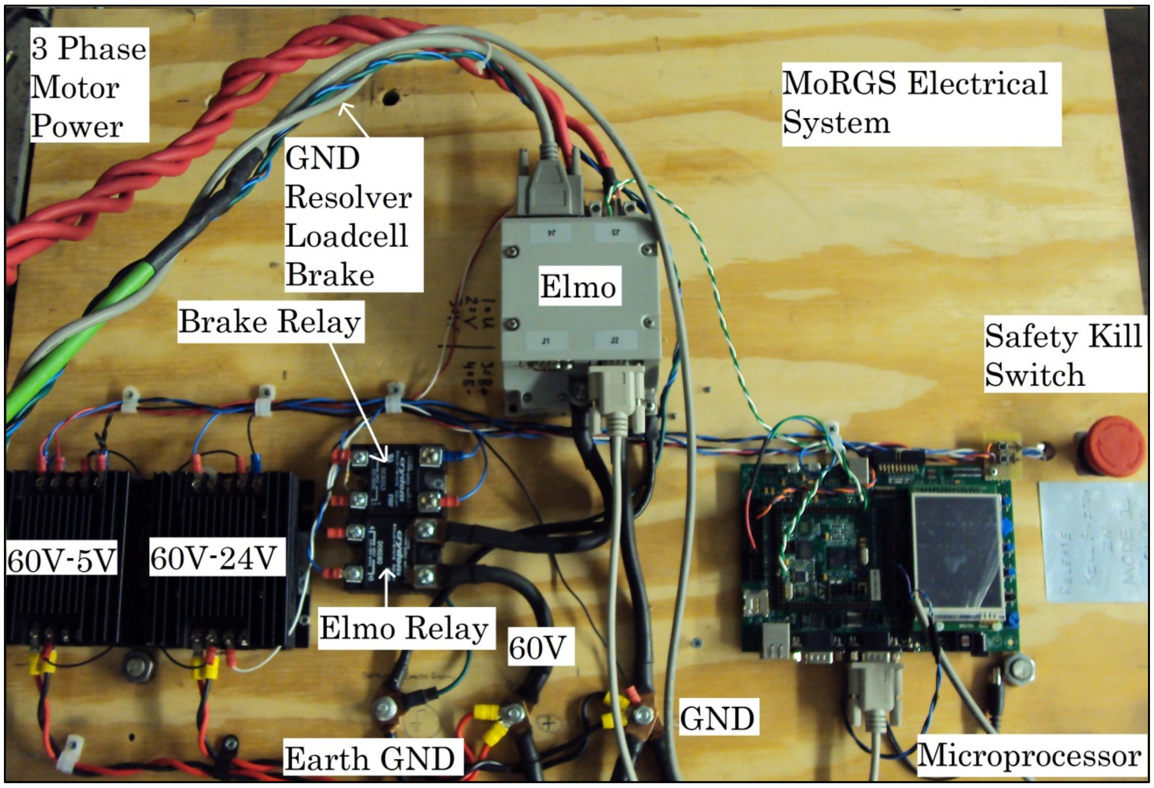


Figure 2-10: *Compact electrical layout for MoRGS*

2.5.1 Batteries

For the testing period of MoRGS, a Warner Power 9kW DC power supply was used. This large power supply has an input of 230 V, 3-phase at 60 Hz, and an output of 0-150 V DC, 3-phase, and 0-60 A. Since testing was for prolonged periods of time, the batteries were not used until the prototype was completed. The final battery pack was a reconditioned forklift pack with 765 A/h and weighs 3,070 lb. This battery pack offers 15 hours of continuous testing.

2.5.2 Motor Controller

The Elmo SimplIQ Drum digital servo drive was selected because of MoRGS' unique high current and DC voltage requirements. In addition, the Elmo motor controller includes built-in auto-tuning facilities, serial communication, and digital /

analog inputs and outputs [10]. Also, the Elmo was capable of integrating with the resolver sensor on the motor for position feedback, also why it was selected. The RS232 serial communication is used for sending commands from the microprocessor, such as turning the motor on, setting the jogging velocity, and engaging the brake. The autonomous offload mode uses an analog differential voltage input from the microprocessor to set the velocity of the motor.

2.5.3 Voltage Converters and Power Relays

The MoRGS source voltage of 60V requires a pair of DC to DC voltage converters to convert to 5V and 24V. The 5V supplies voltage to the microprocessor and the load cell. The 24V converter powers the brake disengagement circuit. Both converters were sized to handle the current within its capacity. Solid state relays were selected to switch the power to the Elmo motor controller, the motor, and the motor brake. They were also chosen so that the microprocessor's output voltage of 3.3V could switch the 60V to the Elmo as well as the 24V to the brake. In addition, these devices were sized to withstand the current based on its rated capacity.

2.5.4 Load Cell

A resistive load cell was selected for its low cost and availability. The RAS1 S beam load cell was selected for its accompanying digital load cell interface (DI-100U). To minimize the noise in the analog signal from the load cell, the analog signal is converted to a digital signal in the DI-100U. The microprocessor then read the input signal through a USART and parsed the message into a load reading in lb.

The highest possible speed from the analog to digital device was 7 Hz, which was found with an oscilloscope. 7 Hz was inadequate because the human test subject is operating at 3-4 Hz. In preliminary testing, the lag in response to load changes was noticeable and an upgrade was necessary. The analog to digital device concept was replaced with a strain gauge amplifier, the Futek CSG110 Strain Gauge Amplifier. This output is then sent by a twisted pair set of wires over 45 ft. to a differential amplifier. The differential amplifier has unity gain and filters for the long-distance noise of the signal, then the ADC in the microprocessor reads the voltage. The filtering of the noisy analog signal is also processed digitally with a moving average and a low-pass filter at 10 kHz.

2.5.5 Microprocessor

The STM32F4 Evaluation Board was selected as the microprocessor for MoRGS for its functionality, flexibility, and processing power. The pin out for the microprocessor is listed in Appendix B. A diagram showing the inputs and the outputs for the microprocessor can be found in **Figure 2-11**.

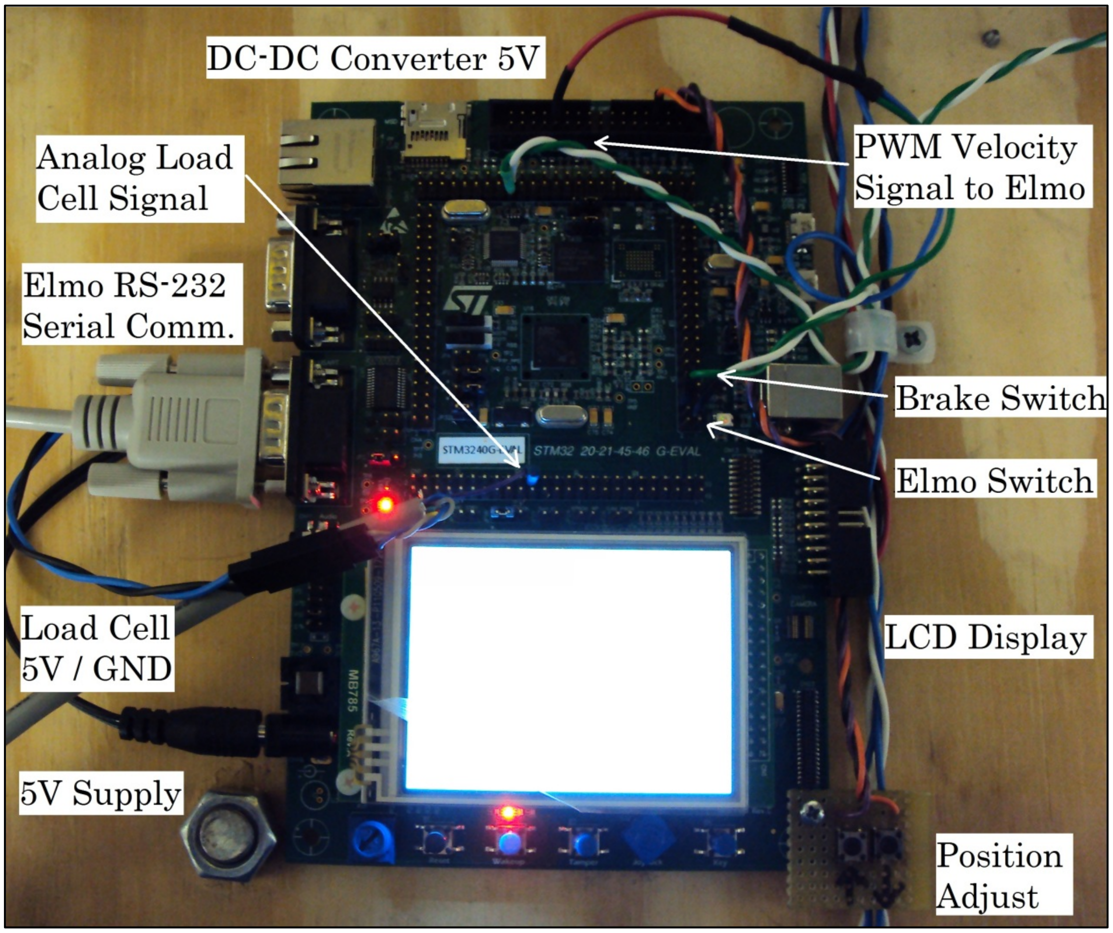


Figure 2-11: *Inputs and outputs on the microprocessor*

The processor was programmed using ST-Link through a USB connection with the EWARM IDE. The board is powered externally with a 5V supply from the main 60V power supply. Two momentary switches were added to the board to allow the user to adjust the position of MoRGS before the offloading mode is started. The LCD displays the mode the operator is in, the current force from the load cell, the direction of position adjustment, and the initial offloaded force.

2.5.6 Shunt Regulator

During the testing period of MoRGS it was realized that the motor regenerated energy much more than anticipated when MoRGS was lowering the load. This additional voltage to the system exceeded the capacity of the internal capacitors in the Elmo and the power supply and increased the overall bus voltage. As a result, the supply voltage to the Elmo exceeded its rating causing it to shut off the motor and drop the load. A shunt regulator was added to the electrical system to regulate the voltage by consuming the excess energy in a 5Ω resistor. When MoRGS exceeds 50V, the shunt regulator switches on and feeds current through the resistor. The shunt regulator switches off once the voltage drops below 50V. The 95 W dissipation capacity shunt regulator from Advanced Motion Controls was calculated to consume the energy from the falling load in the form of a power equation:

$$P_{regen\,power} = V_{speed}F_{load} \tag{2.13}$$

2.6 Software

The IAR EWARM was selected as the IDE for MoRGS. The features and availability make this free IDE an ideal software choice. The microprocessor code was written in C.

2.6.1 Communication to Load Cell

When the load cell was coupled with the DI-100 analog to digital device, the serial output was read into the microprocessor with a USART run on DMA. This included an input and output stream because the DI-100 required an initial command to stream the force values continuously. Another commonly used command was to tare the load cell through the DI-100. After the switch to the Futek amplifier, the 0-5V signal is read by an ADC on the evaluation board. This signal also is controlled through a DMA controller. The 14 kHz signal from the DMA controller needed further digital processing in order to increase its precision. The development of the digital filtering process is described in Section 4.3.1. The force value is computed from a linear function based on the voltage signal. The linear function was derived from measuring a series of known weights and creating a linear best-fit curve like in (2.14).

$$F = -(0.1001 * V - 63.887) \tag{2.14}$$

2.6.2 Communication to Elmo

The RS-232 communication port on the Elmo motor controller is being used for all commands to the motor that do not need high bandwidth. Originally all commands including the velocity command during offloading was sent through serial to the Elmo. Unfortunately the low bandwidth of the data transmission coupled with the low bandwidth of the load cell data transmission resulted in a poor performing controller that lacked fast enough disturbance response. The commands sent to the Elmo through serial are shown in **Table 2-9**.

Table 2-9: *Typical motor commands for MoRGS*

Command	Description
MO=1;	Turn motor on
MO=0;	Turn motor off
BG;	Begin motion
ST;	Stop motion
RM=1;	Differential velocity reference mode on
RM=0;	Differential velocity reference mode off
JV=xx;	Set jogging velocity in counts/sec
EO=0;	Turn message echo off

The high bandwidth velocity command during offload mode is achieved by two PWM signals from the microprocessor. Each signal outputs a peak voltage of 3.3 V. The differential voltage input pins on the Elmo have positive and negative sense, meaning the "positive" PWM signal is at 100% duty cycle, the Elmo reads 3.3V and if the "negative" PWM signal is at 100% duty cycle, the Elmo reads -3.3V. A proportional gain set in the Elmo with the Composer Software with the AG[2] parameter set to 20,755 counts/sec/volt. Counts are a distance unit used by the motor because the motor has 4,096 counts per revolution from the resolver.

2.7 Safety

Safety for MoRGS is high priority because a human subject would be attached to a very fast and very powerful motor. The system has *double fault* in its electrical system as described in Section 2.5. The *double fault* is a redundancy in safety features in case one of them fails during operation. In order for the operator to disengage the brake at any time during the operation of MoRGS, all of the criteria in **Table 2-10** must be met and if any of them are not, the brake is not disengaged and the motor does not rotate. The complete implementation in the microprocessor can be found in the source code in Appendix G.

Table 2-10: *Criteria for disengaging the motor-integrated brake*

Criteria	Explanation
Power	Power must be applied to disengage the brake
Microprocessor & Elmo command	Microprocessor sends command to the Elmo to release the brake and the Elmo activates the relay
Microprocessor enables brake power	The microprocessor controls the power supply to the brake
Safety button released	Switches on the safety button on the brake power and turns on the Elmo

The effectiveness of the brake was evaluated by two parameters: first, how long it takes to engage and stop the load, and second, the amount of slip, if any, when a load is applied. The first parameter was tested with a load of 90 lb. using two 45 lb. exercise plates. The load was lifted 10 ft. from the floor. Next the motor was shut off (while the brake was disengaged). After approximately a 5 ft. free fall, the button on the microprocessor was pushed to activate the brake. The position and the velocity in the motor were all measured using the Elmo software.

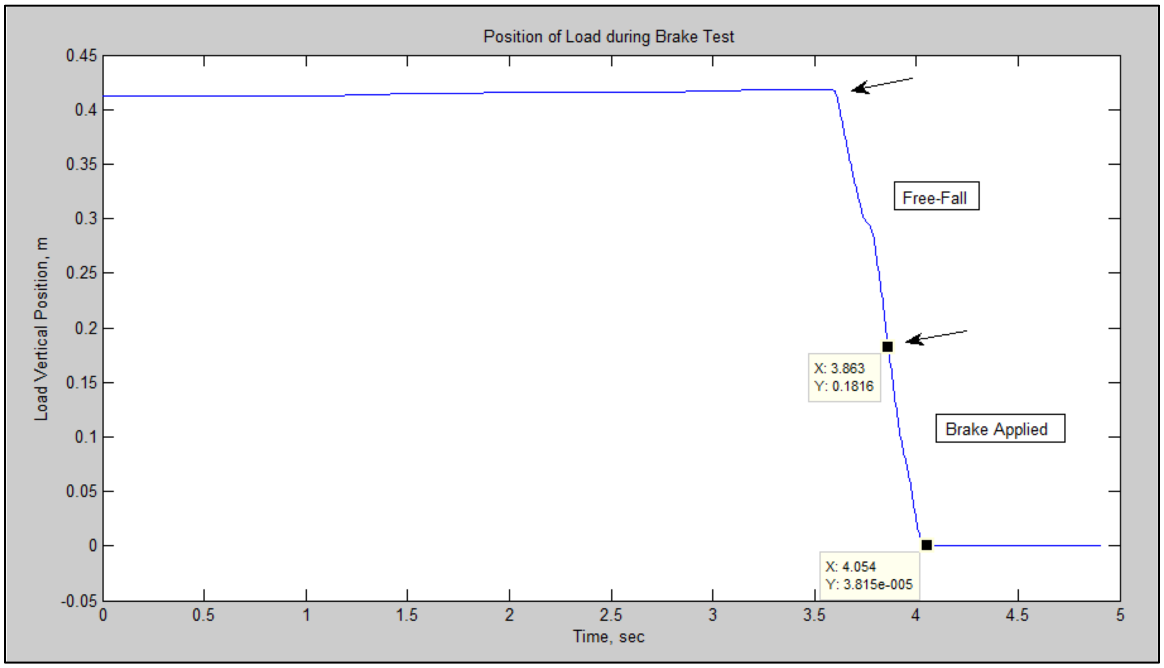


Figure 2-12: *The integrated brake provides adequate stopping power in case of an emergency*

For this test, the brake stopped the load in 0.191 seconds and the load traveled 0.1816 m, as shown in **Figure 2-12**. The specification of the brake from the motor manufacturer is 22 msec. The time delay from the MoRGS' brake is the capacitance in the brake's power supply. Despite this delay, the stopping time is adequate for safe use. The rated stopping torque for the brake is 39.5 N-m which is beyond the torque the motor is designed to supply and adequate for the loads prescribed for MoRGS. The brake maintains the performance required of the motor for torque and offers adequate set and release brake times as shown in **Table 2-11** [2].

Table 2-11: *Brake performance parameters*

Parameter	Value	Units
Brake Holding Torque	39.5	N-m
Power	33.7	W
Voltage	24	V
Current	1.4	A
Brake Set Time	22	msec
Brake Release Time	195	msec
Brake Inertia	0.723	kg-cm^2

A double throw safety switch is installed for the operator to activate in any unsafe conditions. The switch disengages the 5V signal from the microprocessor to the Elmo power relay and the control signal from the microcontroller to the motor brake's power supply, stopping the motor and engaging the brake with double redundancy. The limit stops for speed and acceleration were set in the Elmo motor controller using the Elmo Composer software. The limits were set to the performance requirements for MoRGS.

Chapter 3

Modeling & Control Design

In this chapter, the process of building a model of MoRGS is described from a schematic to a complete MATLAB Simulink simulation. In addition, the development of a unique double free integrator control system is explained. Also, the Simulink simulation shows how the human controller creates equilibrium for the motor velocity controller and the force controller.

3.1 Dynamic Equations

A schematic model of the mechanical components in MoRGS is shown in **Figure 3-1**. To simplify the model, the rotation of the motor, gearbox, and drum are replaced by an equivalent linear displacement, x_1. The inertia of the motor, the gearbox, and the drum were modeled as masses, found by reflecting their inertias through the gear ratio of the gearbox and the drum radius. This equivalent model is shown in **Figure 3-2**.

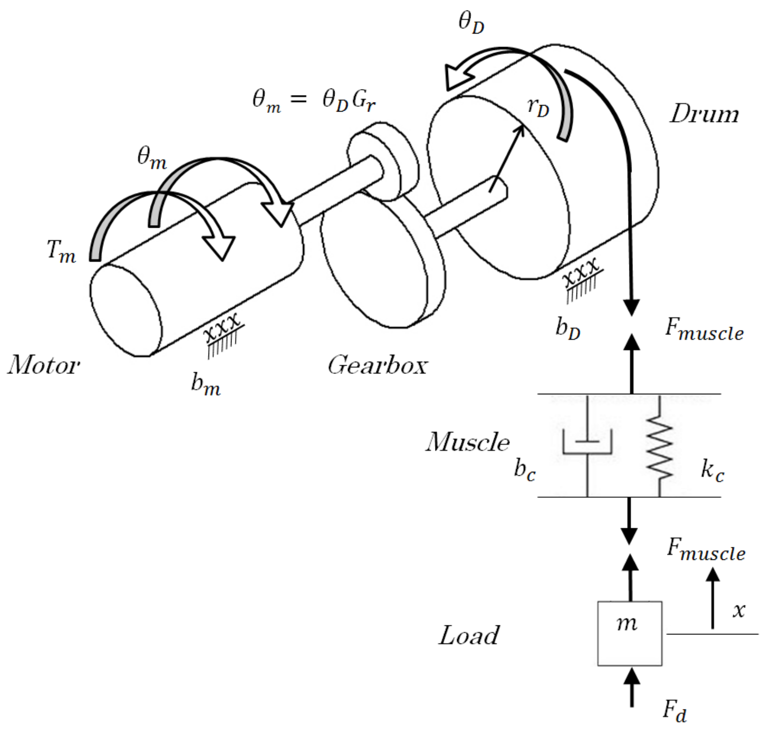


Figure 3-1: *Schematic model of MoRGS where F_d is the sum of the human disturbance and the gravitational force*

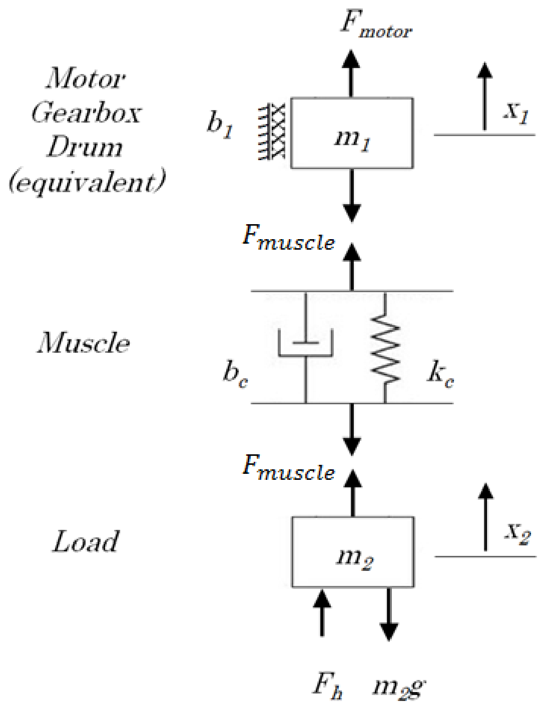


Figure 3-2: *Simplified schematic model of MoRGS*

3.1.1 Basic Model Development

The diagram in **Figure 3-2** is decomposed into two smaller systems by a boundary layer. The first system encircles the motor, gearbox, drum, and the PMA, shown in **Figure 3-3 a)**. The second system includes the load, gravity, and the PMA, shown in **Figure 3-3 b)**.

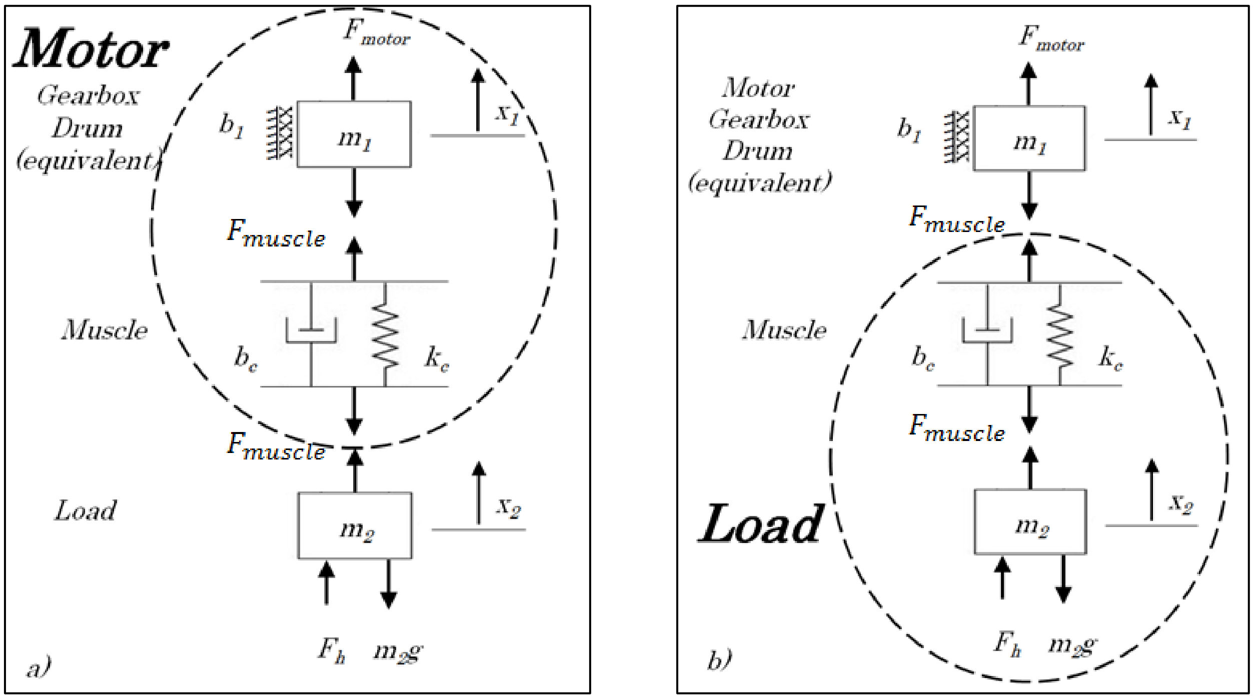


Figure 3-3: *a) The force model of the PMA and equivalent motor b) The force model of the PMA and load*

For the purposes of controller design, the basic model ignores the gravitational force, m_2g, and the human disturbance, F_h from **Figure 3-3**. Using both the diagrams (a) and (b) in **Figure 3-3**, three dynamic equations in the time domain can be generated for the motor, the muscle, and the human, respectively,

$$F_{motor}(t) = F_{muscle}(t) + m_1\ddot{x}_1(t) + b_1\dot{x}_1(t), \tag{3.1}$$

$$F_{muscle}(t) = k_c(x_1(t) - x_2(t)) + b_c(\dot{x}_1(t) - \dot{x}_2(t)), \tag{3.2}$$

$$F_{muscle}(t) = m_2\ddot{x}_2(t). \tag{3.3}$$

Then using these three equations and the Laplace transform, the following three transfer functions can be derived for the motor, the muscle, and the human, respectively,

$$G_{motor} \equiv \frac{\mathrm{v}_1(s)}{F_{motor}(s) - F_{muscle}(s)} = \frac{1}{m_1 s + b_1}, \tag{3.4}$$

$$G_{muscle} \equiv \frac{F_{muscle}(s)}{\dot{x}_1(s) - \dot{x}_2(s)} = \frac{b_C s + k_C}{s} \tag{3.5}$$

$$G_{human} \equiv \frac{\mathrm{v}_2(s)}{F_{motor}(s) - F_{muscle}(s)} = \frac{1}{m_2 s} \tag{3.6}$$

Here, $v_1(s) = sx_1(s)$ and $v_2(s) = sx_2(s)$. A block diagram model of the closed loop system with G_{muscle}, G_{motor}, and G_{human}, in addition to the velocity controller, $G_{C,velocity}$, and the force controller, $G_{C,force}$ is shown in **Figure 3-4**.

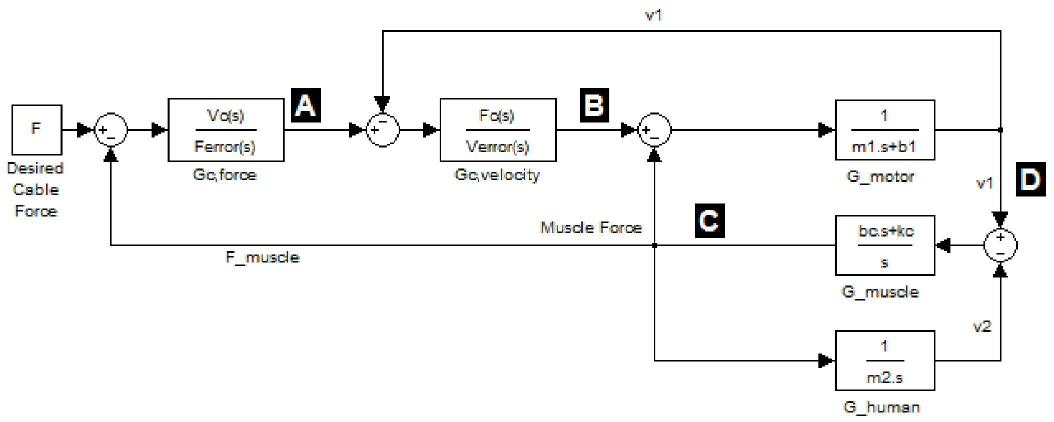


Figure 3-4: *Block diagram of MoRGS showing the velocity loop and the force loop*

3.1.2 Inner and Outer Loop

A series of block diagram algebra manipulations were performed in order to solve for the transfer function $\frac{D(s)}{B(s)}$ to be used in the velocity controller design and the transfer function $\frac{C(s)}{A(s)}$ to be used in the force controller design. The final result for the velocity loop is the transfer function

$$\frac{D(s)}{B(s)} = \frac{G_{motor}(1+G_{muscle}G_{human})}{1+G_{motor}G_{muscle}+G_{muscle}G_{human}} \tag{3.7}$$

and the final result for the force loop is the transfer function

$$\frac{C(s)}{A(s)} = \frac{G_{motor}G_{human}G_{c,velocity}}{1+G_{muscle}G_{human}+G_{motor}G_{human}+G_{motor}G_{C,velocity}(1+G_{muscle}G_{human})} \tag{3.8}$$

3.2 Controller Design

The controller design for the velocity loop on the motor and the force loop on the entire system were completed in MATLAB using the SISO-TOOL interface. Within SISO-TOOL the performance of the controller was analyzed using Bode Plot Frequency analysis of the open loop system. The goals for both controllers were robust disturbance rejection and maximum bandwidth. Disturbance rejection is very important for MoRGS since the offloading mode is, at its very basic level, rejecting disturbances from the human's up or down motion.

3.2.1 Simulation Parameters

In addition to the calculations for the motor sizing and the PMA, simulation was used to couple the different components in MoRGS together and check that they worked well in terms of their performance. **Table 3-1** lists the parameters obtained from the physical components. The complete list of parameters used in the MATLAB simulation is in Appendix D. As discussed previously in Section 3.1, the rotational inertia and damping of the motor and the drum were translated into a linear mass and damping named m_1 and b_1, respectively, and are defined as

$$m_1 = \frac{J_m \cdot {G_r}^2 + J_D}{{r_D}^2} \tag{3.9}$$

$$b_1 = \frac{b_m \cdot {G_r}^2 + b_D}{{r_D}^2} \tag{3.10}$$

The motor's input current was also translated into a linear force by equation (3.11).

$$F_{motor} = \frac{K_T \cdot G_r}{r_D} \cdot I \tag{3.11}$$

The definitions for the variables used in (3.9), (3.10), and (3.11) can be found in **Table 3-1**.

Table 3-1: *These parameters were used in the Simulink simulation*

Motor			
Torque Constant	K_T	1.79	N-m/A
Damping	b_m	0.5979	N-s/m
Inertia	J_m	0.0031	kg-m^2
Gearbox			
Gear Ratio	G_r	5	
Drum			
Radius	r_D	0.1143	m
Damping	b_D	0.0005	N-m-s
Inertia	J_D	0.0096	kg-m^2
Muscle			
Pressure	P	344	kPa
Spring Constant	k_c	23,157.60	N/m
Damping - Tension	b_c	2,900	N-s/m
Damping - Compression	b_c	1,570	N-s/m
Load			
Mass	m_2	83	kg

3.2.2 Motor Speed Controller

The motor's velocity controller is handled by the Elmo motor controller and not the microprocessor. The automatic tuning program within the Elmo was utilized and the bandwidth from that tuning process peaked at 90 Hz. The evaluation process for the Elmo can be found in Section 4.1.1. For the MATLAB Simulink simulation, the velocity controller was modeled following the characteristics of the actual Elmo tuning. Therefore the target bandwidth for the motor speed controller was 90 Hz. The transfer function $\frac{D(s)}{B(s)}$ was used as the plant, for the entire transfer function, see equation (3.7). The Elmo uses a PI controller. **Figure 3-5** shows the frequency response of the plant for the velocity control system. **Figure 3-6** shows the frequency response with the following controller applied where the zero is placed at 9 Hz.

$$G_{c,velocity} = 5000\frac{s+56.5}{s} \tag{3.12}$$

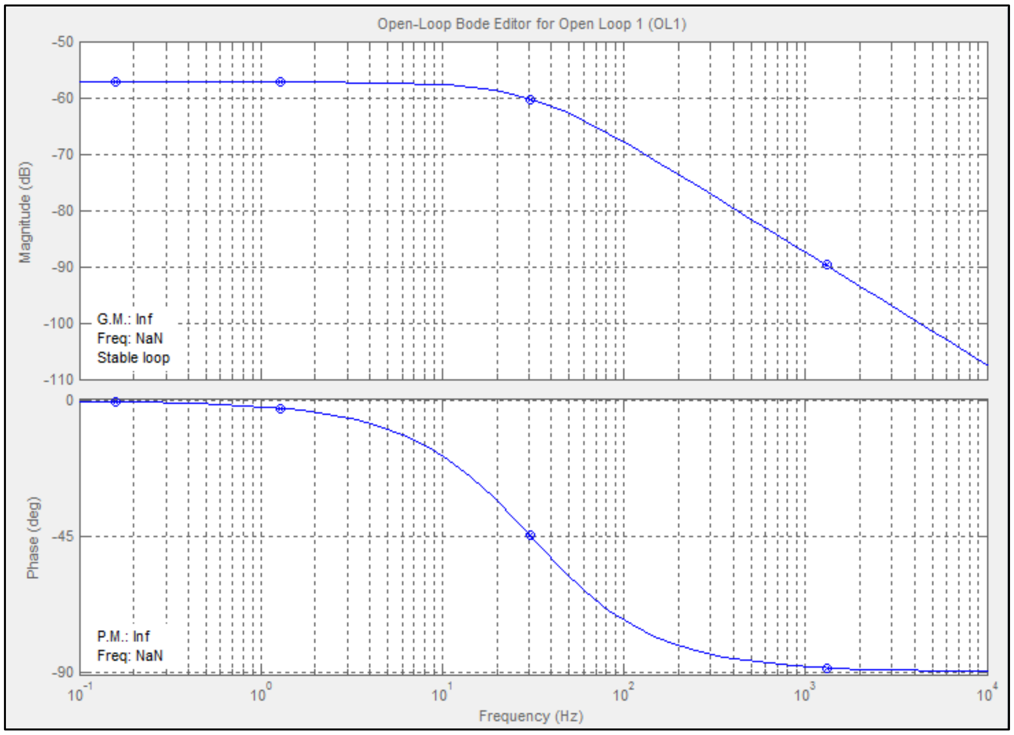


Figure 3-5: *The initial motor plant without the controller*

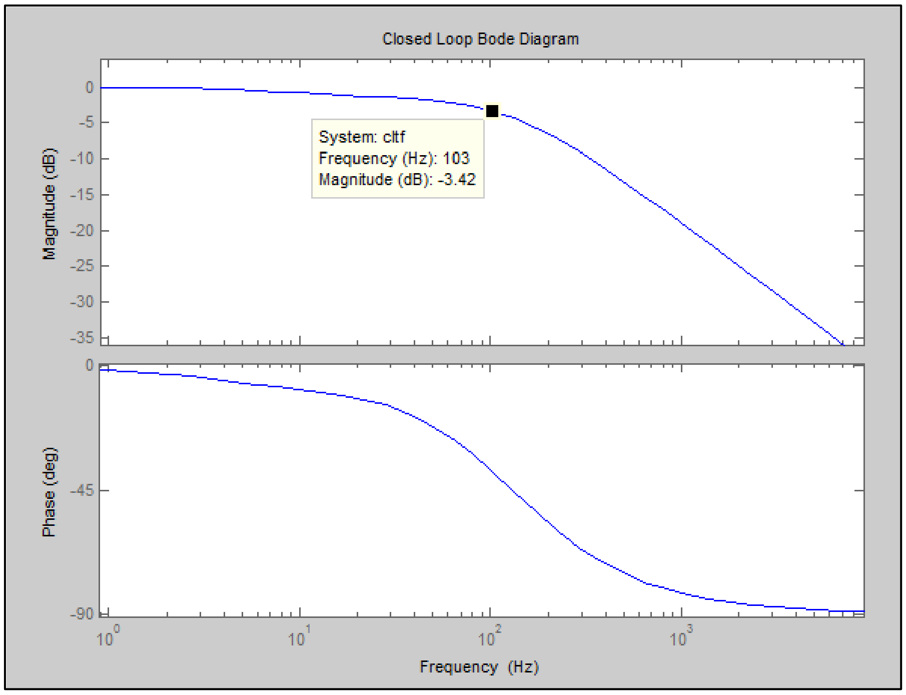


Figure 3-6: *Closed loop Bode plot of the motor velocity controller with a target bandwidth of approximately 90 Hz*

3.2.3 Force Controller

Identifying what sort of system the plant, $\frac{C(s)}{A(s)}$ for the force loop was the first step in the force controller design. The computation of $\frac{C(s)}{A(s)}$ gives the transfer function

$$\frac{C}{A} = \frac{5.448\text{e}010s^9+2.91\text{e}013s^8+5.176\text{e}015s^7+5.966e16s^{10}+3.065\text{e}017s^6}{1.407\text{e}010s^{11}+1.226\text{e}013s^{10}+4.074\text{e}015s^9+6.319\text{e}017s^8+4.396\text{e}019s^7+1.028\text{e}021s^6+7.097\text{e}021s^5} \quad (3.13)$$

To understand the nature of equation (3.13), the poles and zeros were examined. From the higher order nature of $\frac{C(s)}{A(s)}$, theMATLAB function “minreal” was used to account for any pole-zero cancellation and the results are shown in **Table 3-2**. To gain a visual grasp of the system, an open loop Bode plot is shown in **Figure 3-7**.

Table 3-2: *The poles and zeros of C / A using MATLAB's pzmap and minreal functions*

C / A Plant	
Poles (15)	Zeros (13)
-246.5 ± 24.9 *i*	-191
-171.6	-171.6
-171.6	-171.6
-22.6	6 Free Differentiators
-12.4	
5 Free Integrators	
C / A Plant with MinReal	
Poles (5)	Zeros (3)
-246.5 ± 24.9 *i*	-191
-22.6	1 Free Differentiator
-12.4	

The force controller is designed based on the shape of the Bode plot for the plant shown in **Figure 3-7**. A realistic absolute maximum crossover frequency for the open loop magnitude with the controller might be 30 Hz, 1/3 of the velocity loop bandwidth. The low frequency slope of the plant's magnitude is 20 dB/decade; therefore two free integrators are included in the controller to change this to -20 dB/decade. Two zeros are used to keep the slope near -20 dB/decade after the two low frequency poles in the plant. The proposed controller is

$$G_{c,force} = \frac{K}{s^2} \cdot \frac{(s+z)^2}{(s+p)^2} \cdot \left(\frac{p}{z}\right)^2 \tag{3.14}$$

The additional two poles in this controller are added to keep the portion of the controller without the integrators proper, and to reduce the effects of high frequency noise. It was desirable for controller implementation to keep the two integrators separate to aid in the implementation of integral anti-windup protection.

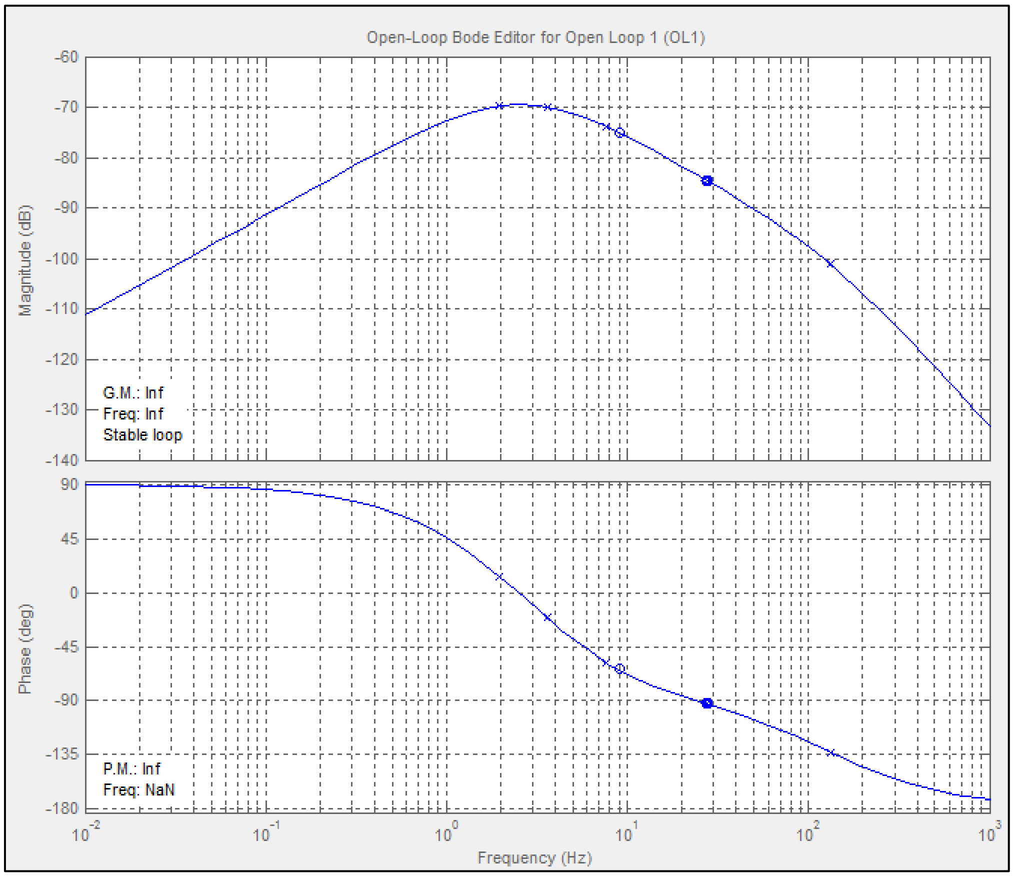


Figure 3-7: *The Bode plot of the C/A transfer function*

Similar to the velocity controller design, the force controller was modeled in MATLAB's SISO-TOOL in the *Compensator Editor*. The zero and pole locations for the force controller were variable. Several iterations between the Simulink model and the physical MoRGS system developed the pole and zero location. The final pole location was selected at several orders of magnitude greater than the desired bandwidth of the force controller, therefore *p = 3770 rad/s*, which is equivalent to 600 Hz. The zeros' location was selected at the peak of the open-loop Bode plot of the plant, which was at 3 Hz, shown in **Figure 3-7**. Therefore, the zero location is located at *z = 18.85 rad/s*. This location at the peak of the plant's frequency response allowed 3-4 Hz of additional bandwidth from its previous zero position which was beyond the low frequency poles. Further discussion on the placement of the poles and zeros is given in Section 4.2.2. **Figure 3-8** shows the shape of the open-loop frequency response with the *Force Controller* applied. The resulting controller with the coefficients inserted is

$$G_{c,force} = \frac{2.5e11}{s^2} \cdot \frac{(s+18.85)(s+18.85)}{(s+3770)(s+3770)} \tag{3.15}$$

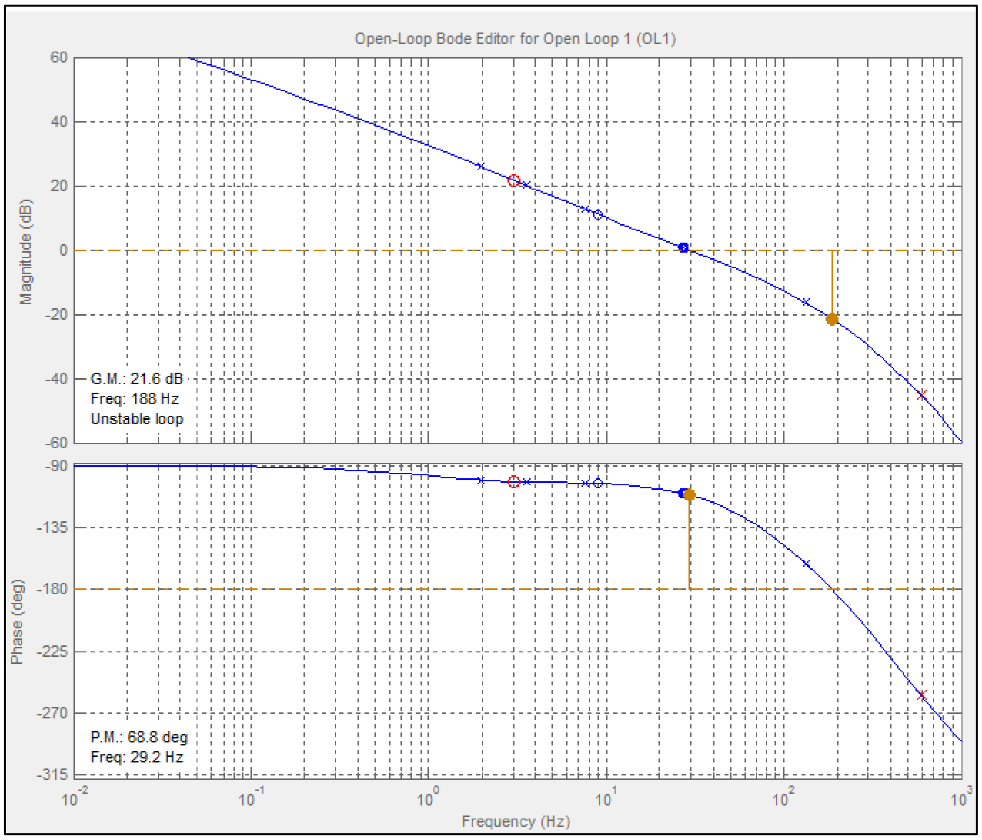


Figure 3-8: *Open-loop frequency response with the Force Controller applied*

The frequency response in **Figure 3-8** has a -20 dB/decade slope before the crossover translating to improved tracking for the closed loop system at low frequencies. The response has resistance to high-frequency noise with its continued decrease in magnitude after the crossover. This controller is simulated in MATLAB Simulink and the results are in Section 3.4.2. In addition, this controller also was tested on MoRGS and those results are found in Section 4.2.

3.2.4 Controller Implementation

In both the simulation and the physical tests, the controller needed to be translated into discrete time. The portion of the controller without the integrators

$$\frac{C(s)}{R(s)} = \frac{K(s+z)^2}{(s+p)^2} \cdot \left(\frac{p}{z}\right)^2 \tag{3.16}$$

is converted to discrete time using Tustin's approximation

$$s \cong \frac{2}{T} \cdot \left(\frac{z-1}{z+1}\right) \tag{3.17}$$

The result is a finite difference equation of the form

$$C_k = \frac{a_0 R_k + a_1 R_{k-1} + a_2 R_{k-2} - d_1 C_{k-1} - d_2 C_{k-2}}{d_0} \tag{3.18}$$

The integrators then operate on this output, C_k, to give an acceleration and velocity command using the following approximations, respectively,

$$A_k = A_{k-1} + T \cdot C_k, \tag{3.19}$$

$$V_k = V_{k-1} + T \cdot A_k. \tag{3.20}$$

Through experimentation it was found that this form of the controller could be used to conveniently provide anti-windup protection for the integrators. The integrators were simply limited to maximum velocity and acceleration of the system which are known from the properties of the motor and motor controller. The complete MATLAB algorithm for generating C_k is in Appendix C.

3.3 Robustness

3.3.1 Natural Frequency

MoRGS was designed to handle offloads from 50 lb. to 625 lb. and there was a concern how this varying load would impact the force controller. The shape of the frequency response in the Bode plot was used to compare the transfer function $\frac{C(s)}{A(s)}$

with multiple loads applied to the system. The open loop Body plot of $\frac{C(s)}{A(s)}$ is shown in **Figure 3-9** with loads applied varying from 50 lb. to 625 lb. The Bode plot shows that the four different loads follow the same shape until after 3 Hz when the four responses begin to separate.

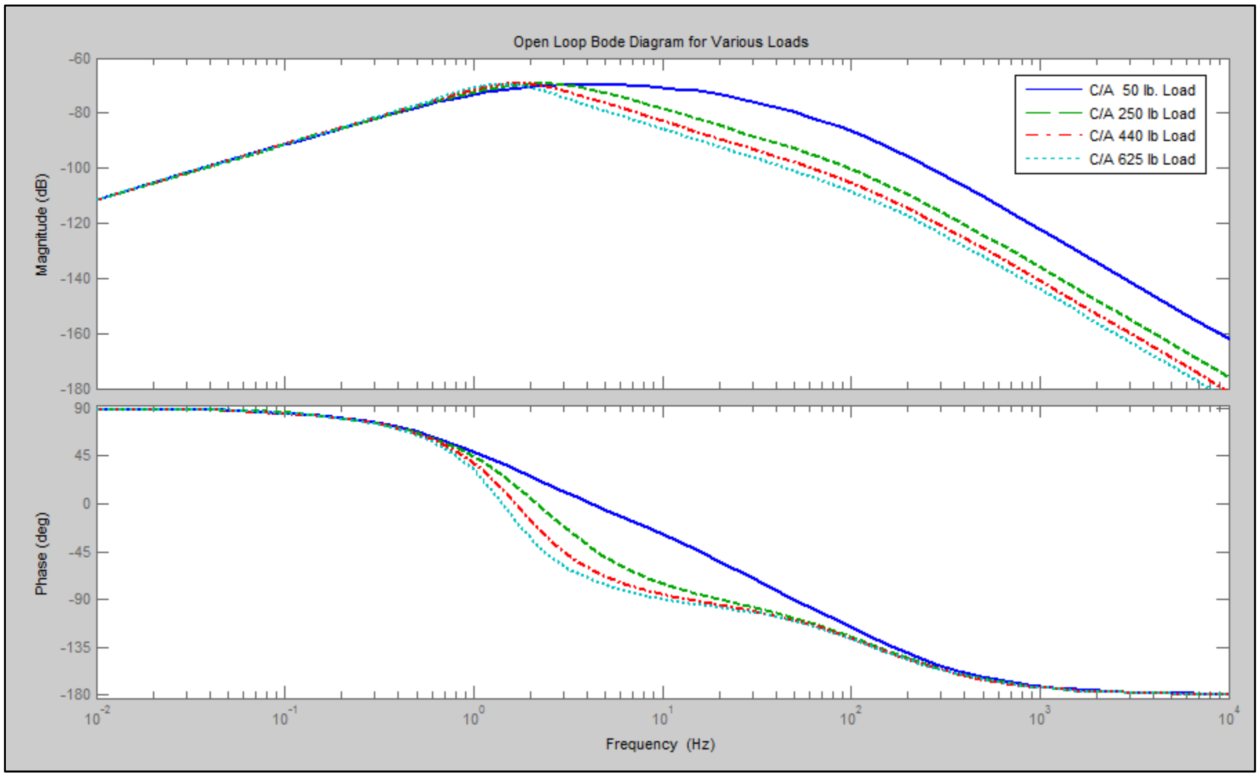


Figure 3-9: *Comparison of the dynamic model for a 50 lb. load up to a 625 lb. load*

The question then arose as to what made the natural frequency of the system with varying loads stay fairly constant. The natural frequency of the system is dependent on the two masses and the spring that couples them together. The matching of the system responses in **Figure 3-9** can be explained by the ratio of the two masses. The 2-DOF semi-definite system [13] has the equation for the natural frequency in the form

$$\omega = \sqrt{\frac{k}{m} \cdot \frac{1+M/m}{M/m}} \tag{3.21}$$

when the system has one spring coupling two masses, as is the case for MoRGS. If the ratio of the larger mass, *M*, and the smaller mass, *m*, in (3.21) approaches ∞ then the natural frequency of the system approaches

$$\omega = \sqrt{\frac{k}{m}} \tag{3.22}$$

The graphical representation of the cancelling of the larger mass's effect on the natural frequency is shown in **Figure 3-10**.

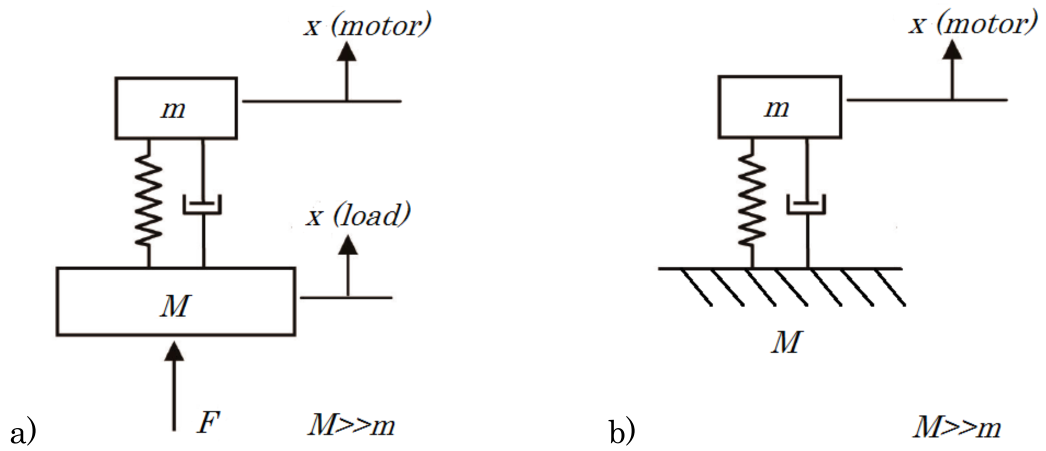


Figure 3-10: *The system represented in a) becomes system b) when M>>m*

As a result of equations (3.21) and (3.22), the impact of increasing the load on MoRGS should have little effect if the ratio of $^{M}/_{m}$ is large for the entire range of loads. **Table 3-3** shows the ratio and natural frequency results for the smallest load of 50 lb. (for testing) and the largest anticipated load of 625 lb.

Table 3-3: *The exact and approximate natural frequencies are nearly the same*

Natural Frequency Analysis			
Load Force (lb)	Load Mass (kg)	M/m Ratio	Natural Frequency - Exact (Hz)
50	22.7	3.39	10.65
250	114	17.01	9.63
440	199.6	29.79	9.51
625	283.5	42.31	9.47
Motor Mass (kg)		6.7	
Spring Constant (N/m)		23157	
Natural Frequency - Approximation (Hz)		**9.36**	

As a result of this analysis, the force controller should be compatible with all four loading scenarios. Some minor tuning adjustments to the gain may improve the response but the overall controller is robust enough to accommodate the small and large loads. The deviation of the responses from **Figure 3-9** all occur after 50 Hz, beyond the range of the force controller and therefore are ignored.

3.3.2 Human Contribution

The block diagram model in **Figure 3-4** was inserted into a MATLAB Simulink simulation. The model was not capable of stability and proved the necessity of the human's input. MoRGS was unable to run properly without an additional outside contribution of stability, a human controller. With the human only applying a constant force, i.e. the load from partial gravity, once motion was started in any direction that motion continued. A PI-controller was designed for the human contribution of stability. MATLAB SISO-TOOL was used in the modeling of the frequency response using Bode plot analysis. The open-loop Bode plot of the human's mass is shown in **Figure 3-11** without the controller.

The targeted bandwidth of the human controller was 3 Hz, based on assumed performance of a human's reaction time. **Figure 3-12** demonstrates the improved performance and increased bandwidth of the human plant. The human controller transfer function is

$$G_{c,human} = 2{,}517 \cdot \frac{s+1}{s} \tag{3.23}$$

The added zero provides the -40dB/decade slope for disturbance rejection and the zero shifts the slope to -20dB/decade around the crossover frequency.

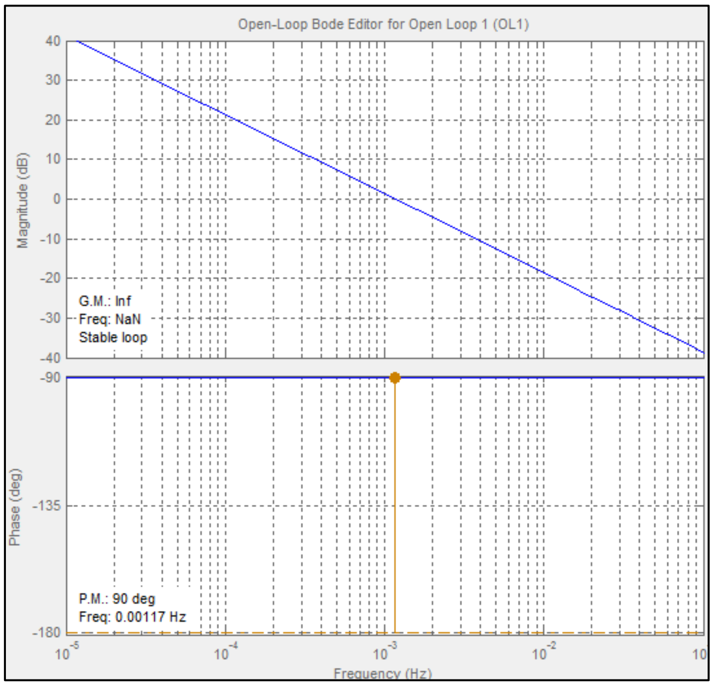


Figure 3-11: *The frequency response of the human with 0.00117 Hz bandwidth*

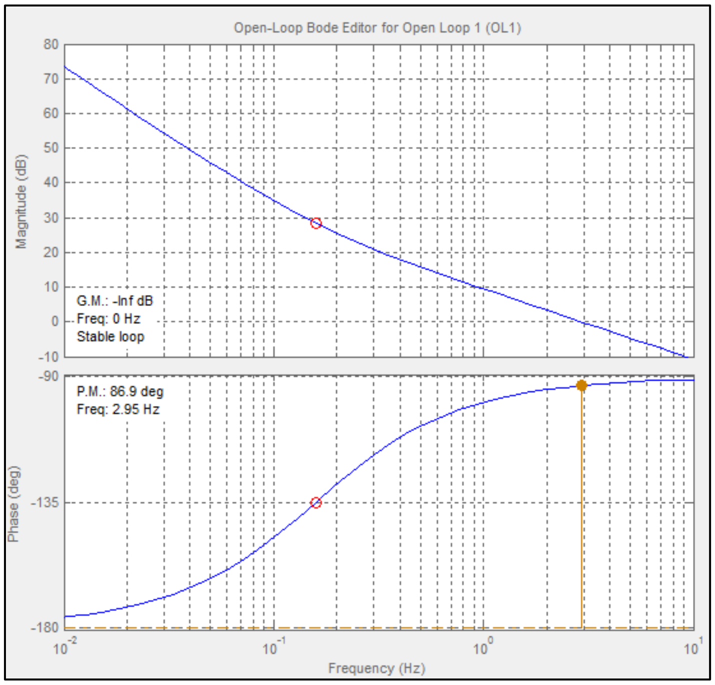


Figure 3-12: *The human controller applied to the human plant with a bandwidth of 3 Hz*

3.4 MATLAB Simulink

MATLAB's Simulink software was utilized in order to better understand the dynamics of the system and to implement the discrete control algorithm that would eventually go into the code for the microprocessor. MATLAB's *S-Function* was used in order to model the discrete behavior of the microcontroller. With this function, the model generated a discrete time step between function calls and included the different update rates of the load cell and the motor. The motor loop has the higher update rate than the load cell. This effect was achieved by a "For" loop with a flag set at multiples of the ratio of the two update rates. The *S-Function* has two inputs (the difference of the desired and actual cable force and the motor velocity) and one output (desired motor current). The discrete time step was set to 1.0e-4 seconds. This parameter was derived from the system clock in the microprocessor which was programmed to 10 kHz. The entire MATLAB m-file script can be found in Appendix F.

3.4.1 Simulink Diagram

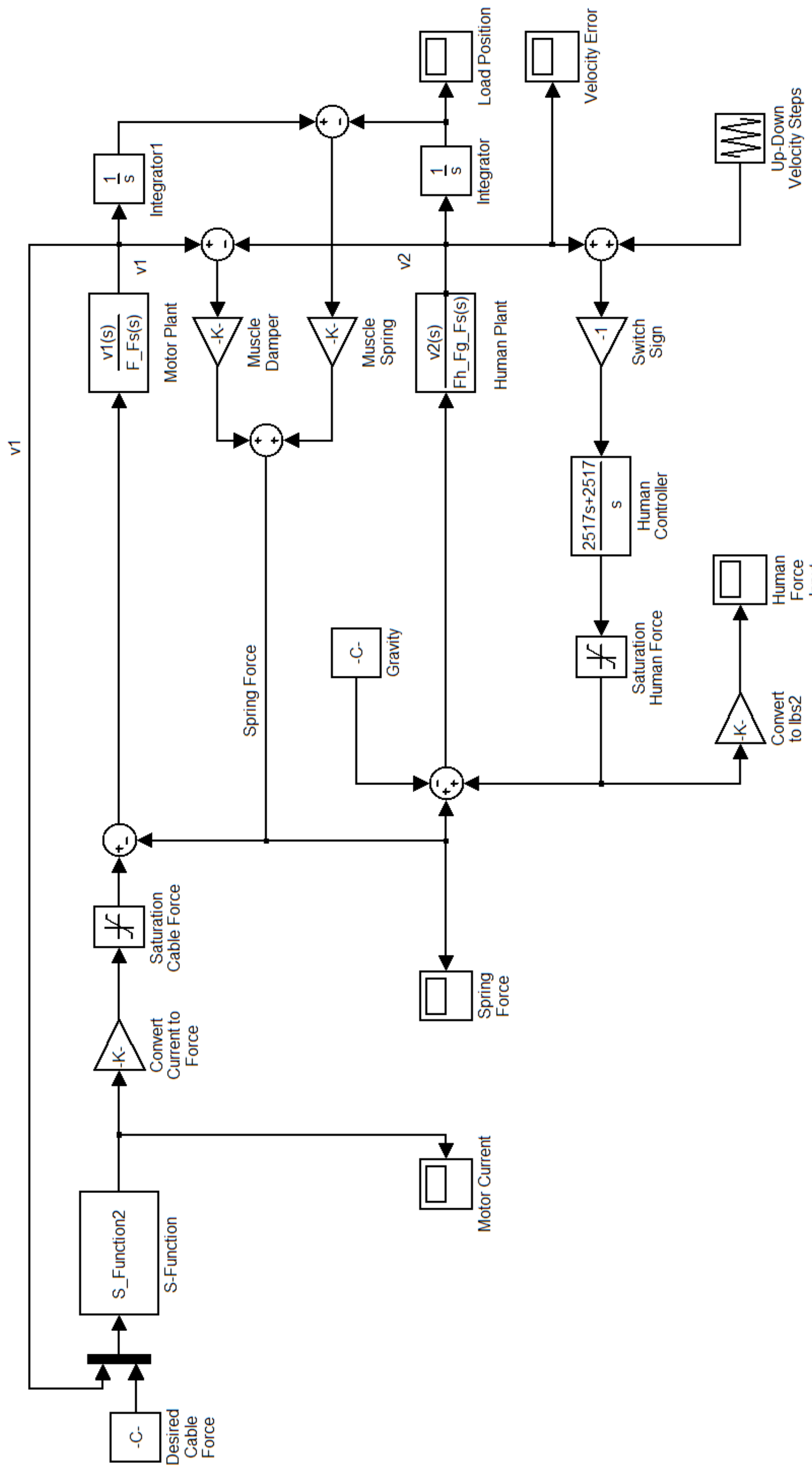


Figure 3-13: *The complete Simulink diagram*

3.4.2 Initial & Equilibrium Conditions

The Simulink diagram was useful in understanding the role gravity plays in MoRGS. The force from gravity generates the initial force in the system and remains constant. The motor and the human each contribute force to MoRGS and equilibrium was a matter of balancing these two controllers in the model.

In order to set the motor's applied force to the spring, the PMA needed an initial displacement at time equal to zero, but with zero velocity. To create this initial stretching of the PMA, the spring force and the damping force were separated into two function blocks in Simulink. The difference in motor and human velocity was input to the damping force block. Meanwhile, the motor and human velocity signals are integrated into position for the spring force block. This integration allowed an initial condition to be set. The human position was set to zero and the motor position was set to a value that, when multiplied by the PMA's spring constant, generated the offloading force necessary. **Figure 3-14** shows the added blocks and their location relative to the rest of the diagram.

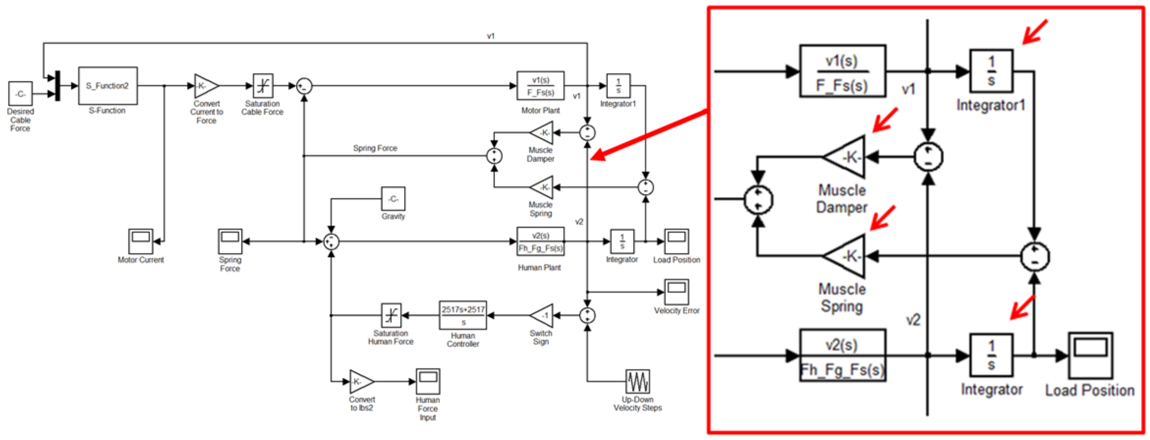


Figure 3-14: *The red arrows show the added Simulink blocks to generate initial conditions in the PMA*

A saturation block was added in order for the simulation to stay within the reasonable capabilities of the human to balance one's self. **Figure 3-15** shows the blocks added to the diagram and their location.

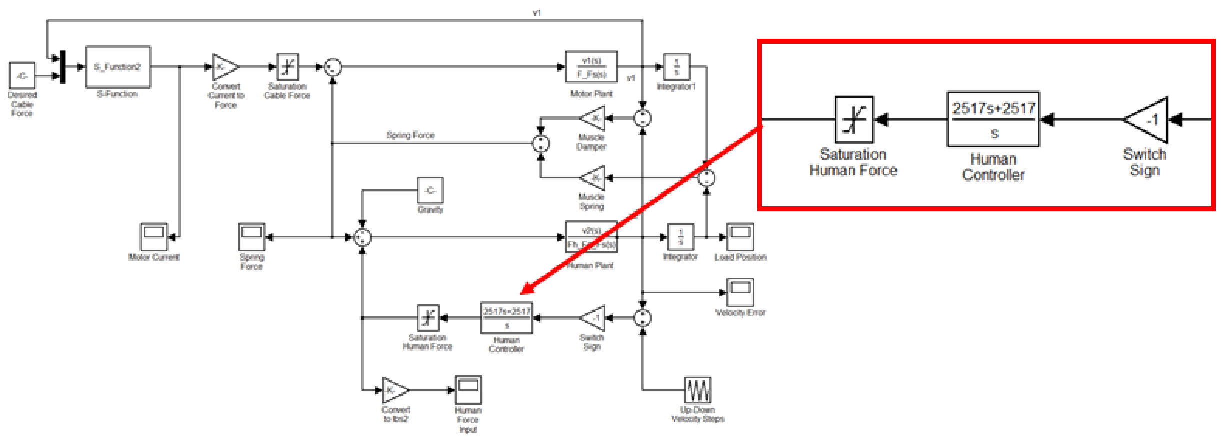


Figure 3-15: *The human itself is a PI force controller for MoRGS*

The equilibrium condition was tested in Simulink to verify the stability of the velocity and force controllers in controlling the entire system. The simulation had a duration of 30 seconds. Several scopes were added to monitor the effectiveness of the system. All plots of performance are in SI units (m, m/s). As desired, the human position found equilibrium at an initial PMA displacement of 5 mm. An initial condition imbalance on the human's free integrator in the simulation was set to -10 N to excite the system and balance itself, shown in **Figure 3-16**. The velocity also balances well with a ±0.002 m/s oscillation and does not drift away as it did in the initial iterations of the simulation. Two important lessons were learned in the equilibrium condition of the simulation, first, the load must be at complete rest before autonomous mode and second, the initial estimation of the load must be accurate in the control algorithm. The actual MoRGS system behaved much better in a steady state condition when these two criteria were met.

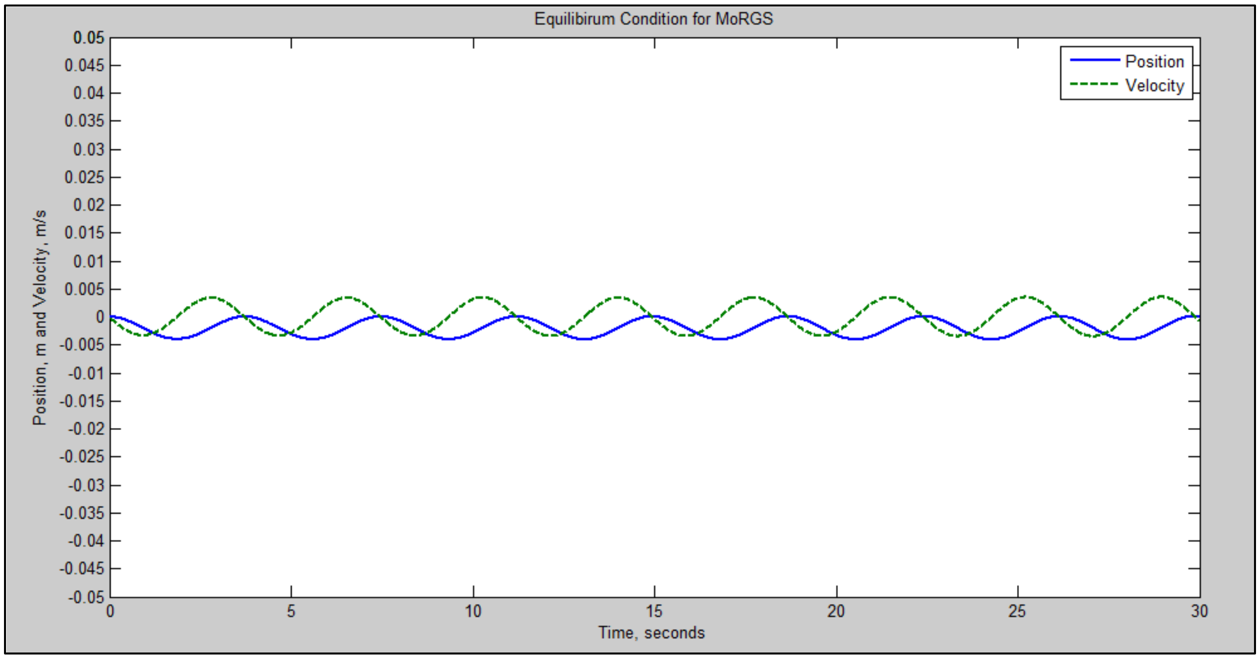


Figure 3-16: *Equilibrium for the human's position (m) and velocity (m/s)*

Since the human will always be contributing some form of disturbance, the equilibrium is only a gage of stability. Stability for this system is the constant force in the cable, with minimal walk from the human's position. The system is stable because the force finds equilibrium with no oscillatory behavior, shown in **Figure 3-17**. A concern for MoRGS was whether the motor would saturate at the maximum current throughout this equilibrium simulation. **Figure 3-17** shows no saturation at the initial stabilization of the system and once the force settles to equilibrium, the motor current is not saturated.

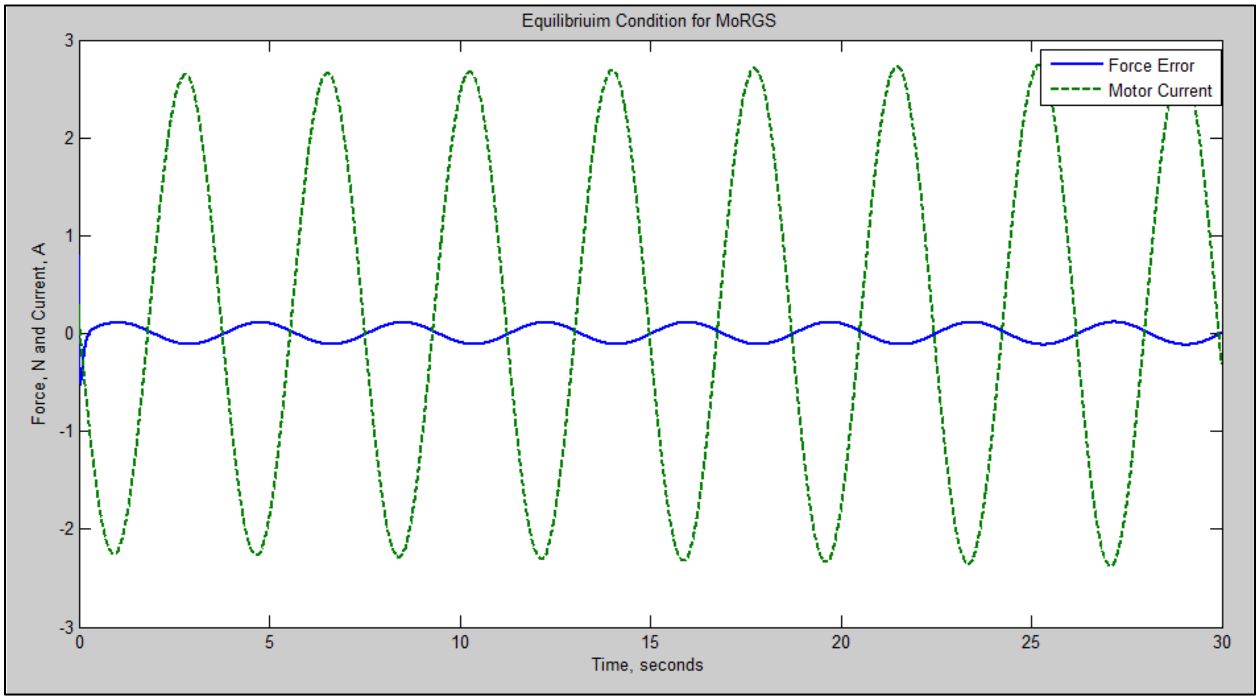


Figure 3-17: *Force Error (N) times 10 and motor current (A)*

The simulation helped with the understanding of how force was propagated within MoRGS. Small oscillations of position are possible, like in **Figure 3-16**, because force disturbances require some form of acceleration, provided by the human, but if the velocity is approximately constant, then force equals zero. There must be some change in force for there to exist an acceleration to create the oscillation in the human's velocity. Testing with MoRGS showed that finding perfect initial conditions was not perfect and the model above reflects that small initial disturbance, caused by the human.

For actual use of MoRGS, the human will be contributing a consistent disturbance and so overall performance of the two force controllers was based on a human running on a treadmill. The estimated profile of a human running was based on information from the KIN Dept. dictating the human to have a vertical displacement of 0.125m at 2 Hz. **Figure 3-18** shows how the human disturbance was integrated into the simulation. The tracking of the force controller was an emphasis in design because MoRGS could not disrupt or alter the motion of the human while keeping an offload of the desired force. To gage the tracking, the velocity of the human in the system was analyzed. This velocity is a response of the force controller to the human's disturbance. The motion profile of the human with the disturbance is shown in **Figure 3-19**. The saw-tooth profile is the human's disturbance, or desired velocity, representing the human running on a treadmill. The other profile is the actual human velocity, based on MoRGS' contribution.

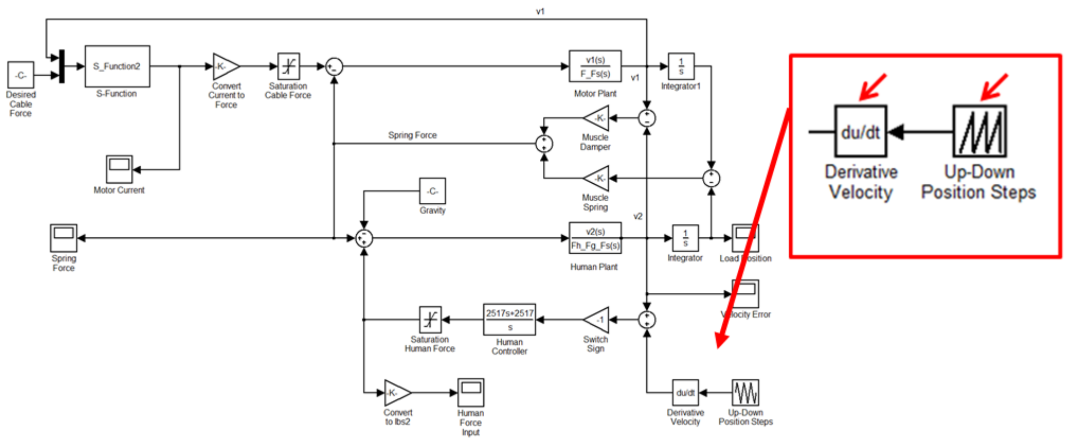


Figure 3-18: *Human disturbance of 0.125m at 2 Hz*

The system maintains stability throughout the 10 seconds of human disturbance. This stability was measured at the human's overall position from the start of the simulation to insure that the human was not being thrown or dropped. **Figure 3-20** shows the human's position system response along with the force error.

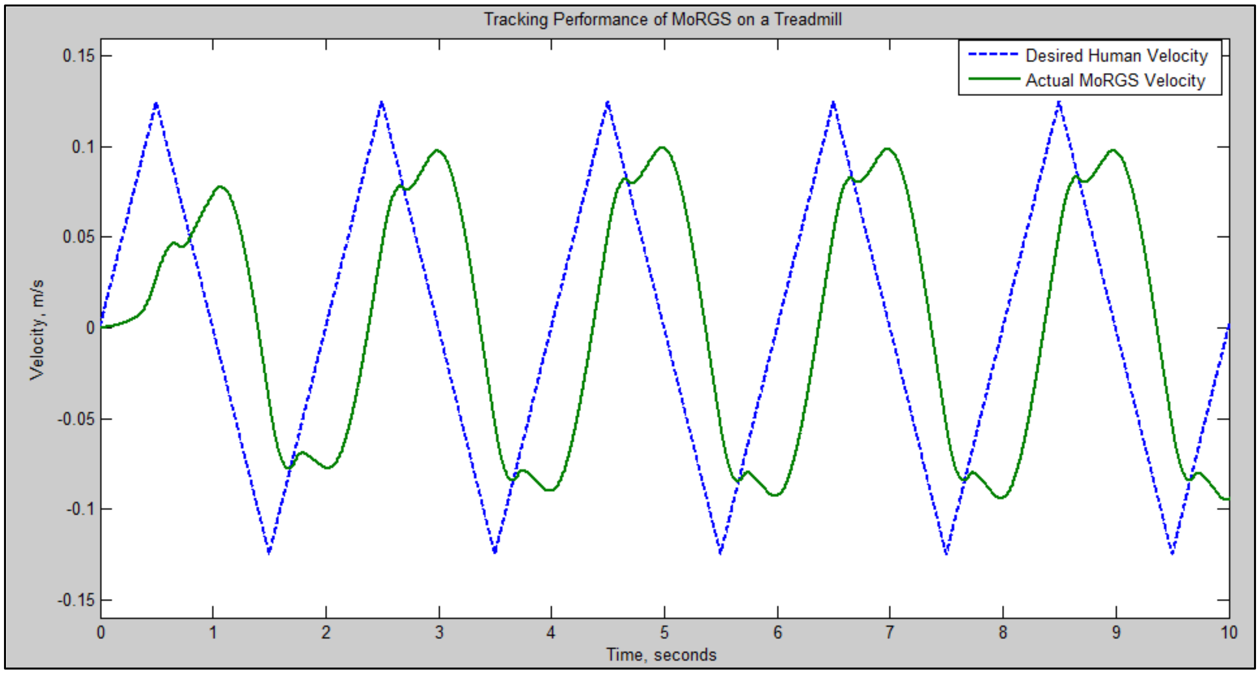


Figure 3-19: *The 2 Hz velocity profile of the disturbance and system response*

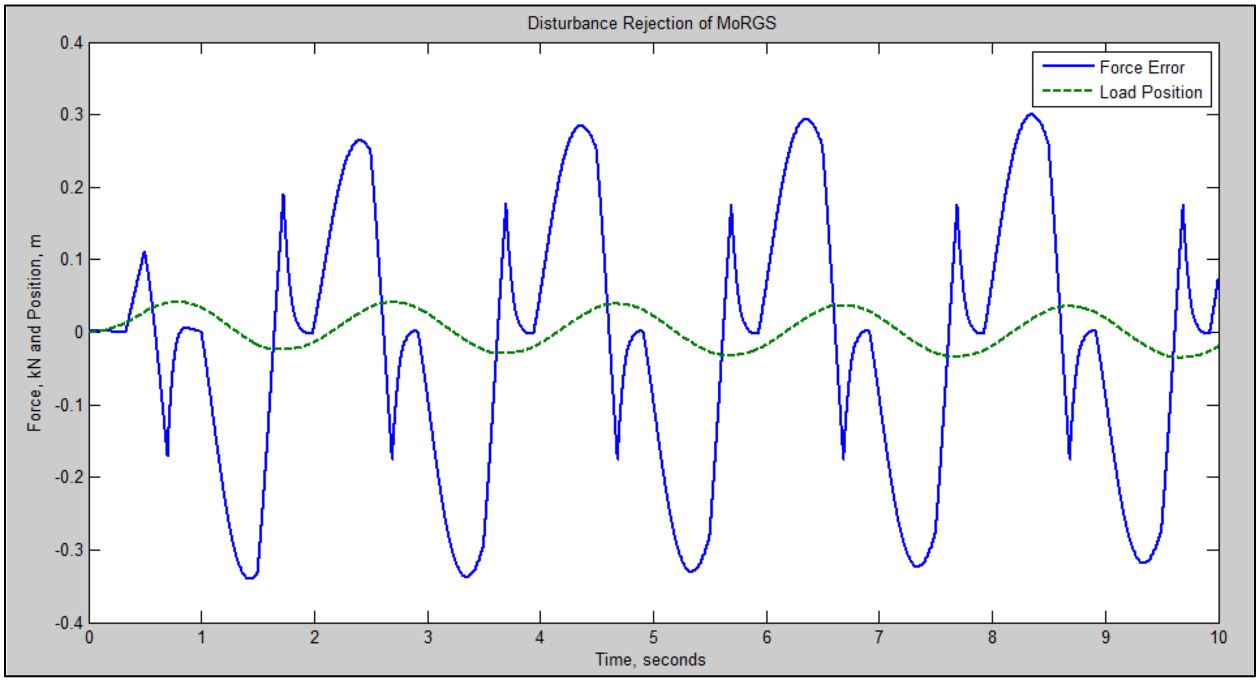


Figure 3-20: *The human's position and cable force is stable based on the 2 Hz human disturbance*

The response to the human disturbance was a major focus in the physical tuning of MoRGS. The human disturbance in the testing phase of MoRGS was a force disturbance and not a velocity disturbance, but the end result is similar, shown by the similar saw-tooth shaped force error in **Figure 3-20**. The phase shift in the desired force and the response of MoRGS is approximately 45 degrees lagging, described in Section 4.3.2. The Simulink model mirrored the phase shift found in the actual system and validated the model for further performance evaluations and understanding.

Chapter 4

Experimental Results

In this chapter, the tuning of the motor velocity controller and the force controller is discussed. In addition, the process of improving the force sensor by a series of filters is also explained. Finally, the robustness of MoRGS is tested with weights from 45 lb. to 180 lb. all while maintaining the same force controller, including the gains.

4.1 Motor Velocity Control

The Elmo motor controller has an integrated velocity controller for the motor. This feature was utilized because it eliminated the need for the microprocessor to have high speed access to the motor resolver data. With the Elmo, the signal processing for the motor all occurs within its processor, but all the settings are programmable for different performance goals. For MoRGS, the Elmo receives a velocity command from the microprocessor and the Elmo implements both a velocity and a current controller. The complete implementation in the microprocessor can be found in the source code in Appendix G.

4.1.1 Elmo Motor Controller Tuning

The Elmo has its own tuning wizard for the velocity controller. This wizard was used in order to tune the current and velocity control loops and to obtain data used in the estimation of the controller gains used for the velocity loop in the dynamic model. The functionality of the wizard in the Elmo software is limited. It went through an automatic current loop and commutation loop without any user input. The velocity and position loop did require some input from the user as to how fast or how accurate the motor needed to be, based on a sliding scale of slow to fast and accurate to noisy. The goal for MoRGS was to achieve an active force offload so that a human would not notice its dynamics. Therefore, maximizing bandwidth for the systems upstream from the human was a priority. The *Fast Tuning* setting was selected for MoRGS to maximize disturbance rejection and bandwidth.

The step response of the Elmo velocity controller showed some higher frequency noise in the response. This noise, though, has been ignored in the simulation because its frequency is much higher than MoRGS' system dynamics. The behavior of the velocity controller is shown in the step response in **Figure 4-1**.

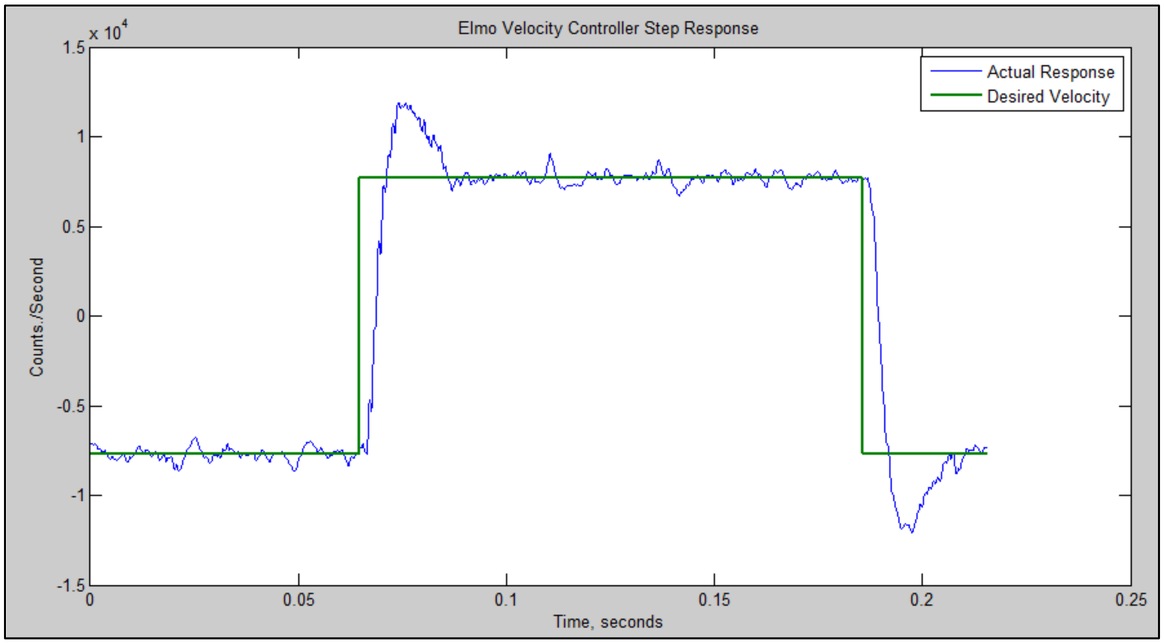


Figure 4-1: *The stable step response of the Elmo velocity controller*

The wizard tuner was run on the actual motor with slow, medium, and fast tuning parameters. The frequency response was recorded and the closed loop Bode plot of the three controllers is shown in **Figure 4-2**. As expected, the *Fast Tuning* achieved the maximum bandwidth of 92 Hz, followed by the *Medium Tuning* at 47 Hz, finished by the *Slow Tuning* with 9 Hz. There is a decrease in the tracking ability as the controllers increase in speed. Since the bandwidth of the velocity controller must be maximized in order to provide the human with a consistent offload, the *Fast Tuning* controller was implemented in MoRGS. The MATLAB Simulink simulation was based around a velocity controller with a bandwidth of 90 Hz, and the Elmo wizard created a controller for the actual system with a bandwidth of 92 Hz. Its effect on the *Force Controller* is shown in Section 4.2.

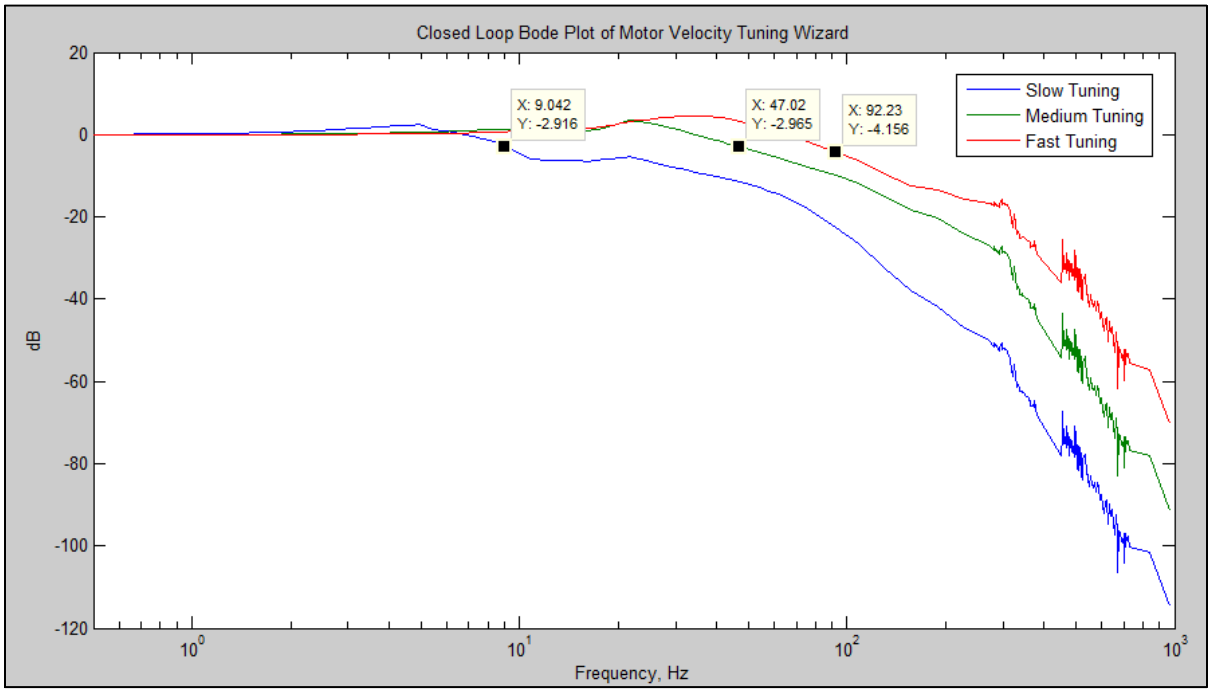


Figure 4-2: *The bandwidth of the motor velocity at 3 different settings measured at -3 dB*

4.2 Force Control

The force controller ties together the entire MoRGS system by connecting the human to the motor. When developing the force controller, the MATLAB simulation provided a good basis for understanding the dynamics of MoRGS. But the true test for the controller was only possible with the actual MoRGS.

4.2.1 Force Signal Conditioning

As mentioned in Section 2.5.4, the digital load cell interface did not have the adequate bandwidth (6 Hz) required for a high performance system like MoRGS. Instead, the analog signal was wired to the analog to digital converter on the microprocessor and then read as a force for the control algorithm. This signal travels 43 ft. up the wire rope, along the gantry beam, and back down to the microprocessor. As a result, the signal is susceptible to noise from the distance traveled and its proximity to the motor cables.

Several measures are taken to improve the quality of the load cell signal. **Figure 4-3** shows a functional diagram of the signal processing involved in MoRGS. A Futek *CSG110 Strain Gauge Amplifier* is connected to the load cell to amplify the millivolt signal to volts. Also a twisted pair is used for signal transmission along with a differential amplifier installed at the microprocessor side of the cable. This amplifier eliminates a majority of the noise generated by electromagnetic interference, as shown in **Figure 4-4**. In addition, the differential amplifier reduces the effects of voltage drop in the ground wire to the amplifier. The signal is over sampled and 10 values from the ADC are averaged, for each sample. Finally, a 100 Hz low-pass filter is applied inside the force calculation loop, which runs at 10 kHz. The generation of the discrete low-pass filter is referenced in Appendix C and the complete implementation in the microprocessor can be found in the source code in Appendix G. The effectiveness of the *Mean of n Values* and the *Low-Pass Filter* is shown in **Figure 4-5** based on a 200 lb. load applied.

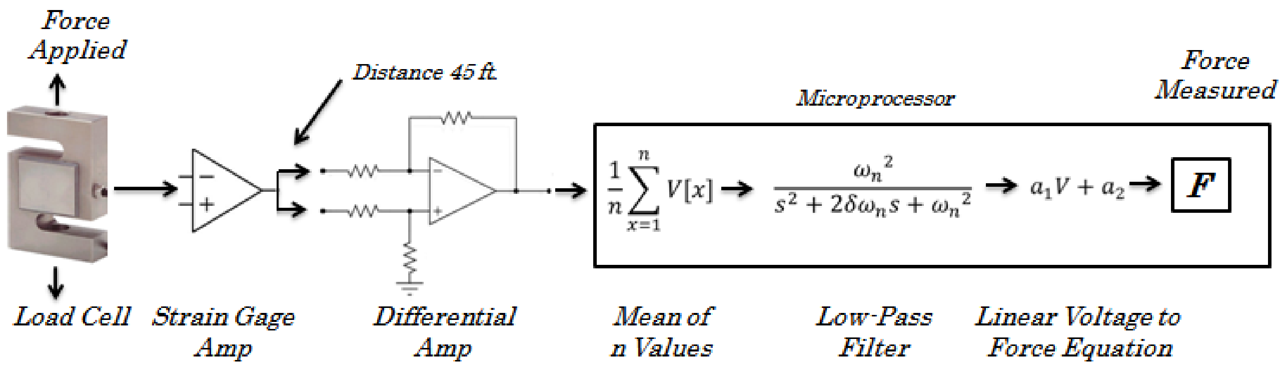


Figure 4-3: *The load cell signal goes through several levels of signal processing (Adapted from [18]*

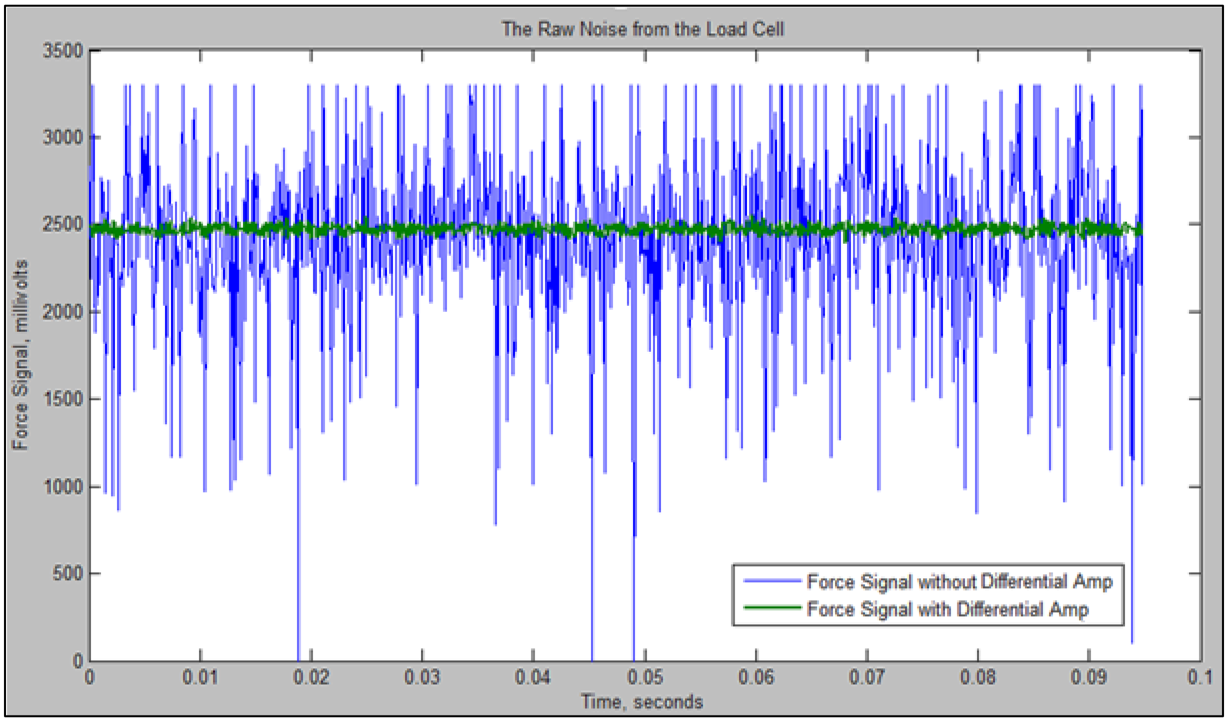


Figure 4-4: *The impact from the differential amplifier was tremendous*

In order to further understand the improvement of the signal processing, **Table 4-1** was made and it shows the standard deviation of the signals converted into force.

Table 4-1: *The additions of the load cell signal processing improves the standard deviation from 50.06 to 0.13*

Signal State	Standard Deviation (mV)	Standard Deviation (lb.)
No Processing	500.11	50.06
Differential Amplifer (DA)	29.00	2.90
Mean Value (MV) & DA	6.09	0.61
Low-Pass Filter & DA, MV	1.28	0.13

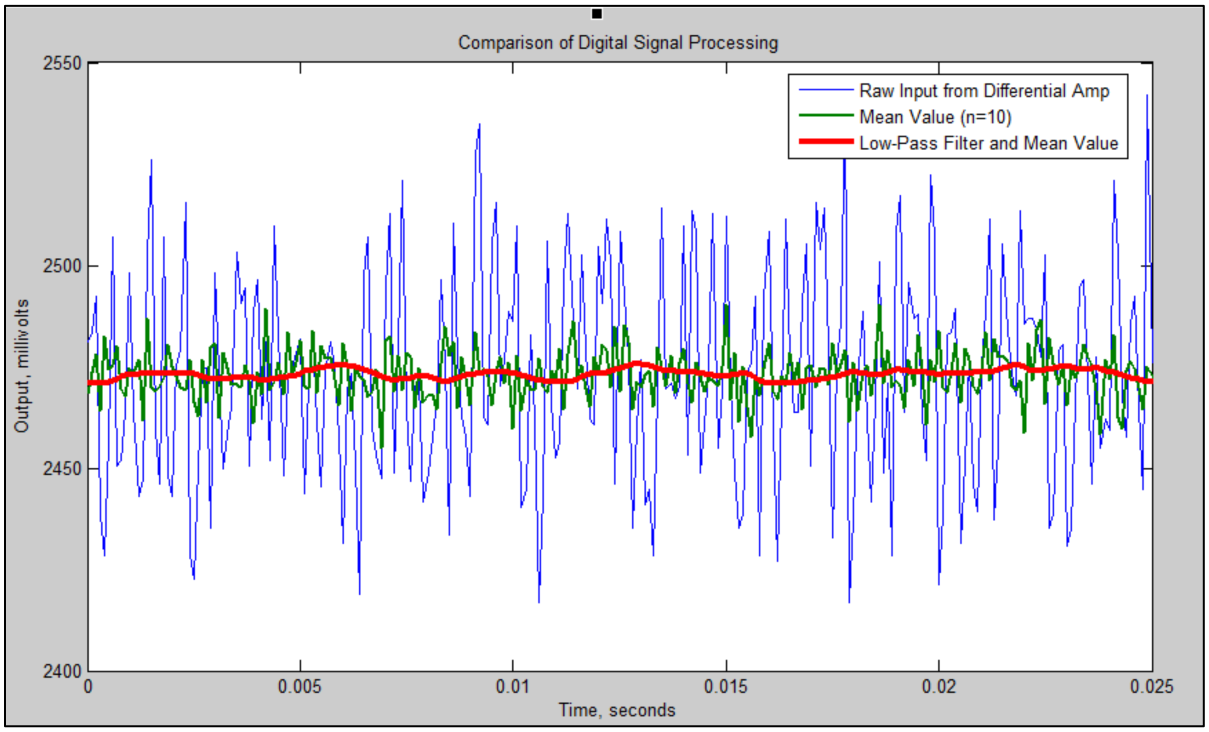


Figure 4-5: *The digital signal processing enhanced the quality of the force signal*

4.2.2 Tuning the Force Controller

As presented in Section 3.2.3, the force controller was presented and modeled in MATLAB Simulink. The results of the simulation were used iteratively with the physical system in order to improve both the model and MoRGS. The metrics of performance were force stability and velocity tracking equating to an emphasis on disturbance rejection. The controller zeros are placed at the frequency where the peak of the open-loop bode plot of the plant is located. They were originally at 25 rad/s but the improved model showed it instead needed to be at 18.85 rad/s. This change in zero location raised the crossover frequency, and so a factor was multiplied to the original gain to maintain a consistent understanding of the gain applied to the acceleration algorithm. Moving the zero to the natural frequency peak of the plant lowered the magnitude of the oscillations in the force controller until not visible by the naked eye. The effect on the stability of MoRGS is shown in **Figure 4-6**.

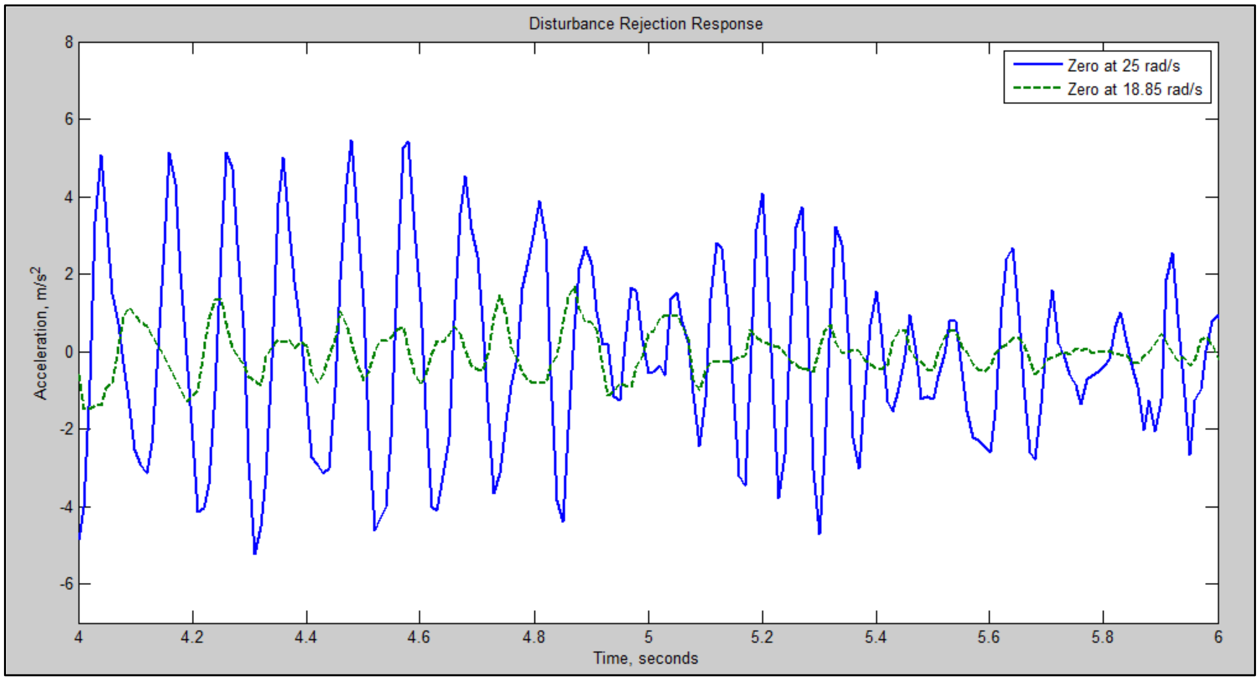


Figure 4-6: *Moving the zeros to the peak of the plant's OL Bode plot decreased noise in the system*

After being satisfied with the new stability in MoRGS by moving the zeros, further testing continued. The next component tuned was the controller's poles. The poles originally were placed at 60 Hz. More understanding of the Simulink model and the motor dynamics showed that this location was not large enough. Several pole locations from an order of magnitude to several orders of magnitude were tested. An upper limit was the force controller's frequency of 10 kHz and did not want to go near this frequency. The pole locations tested were 100 Hz, 500 Hz, and 1,000 Hz. **Figure**

4-7 shows the acceleration command from the force controller for all three pole locations used.

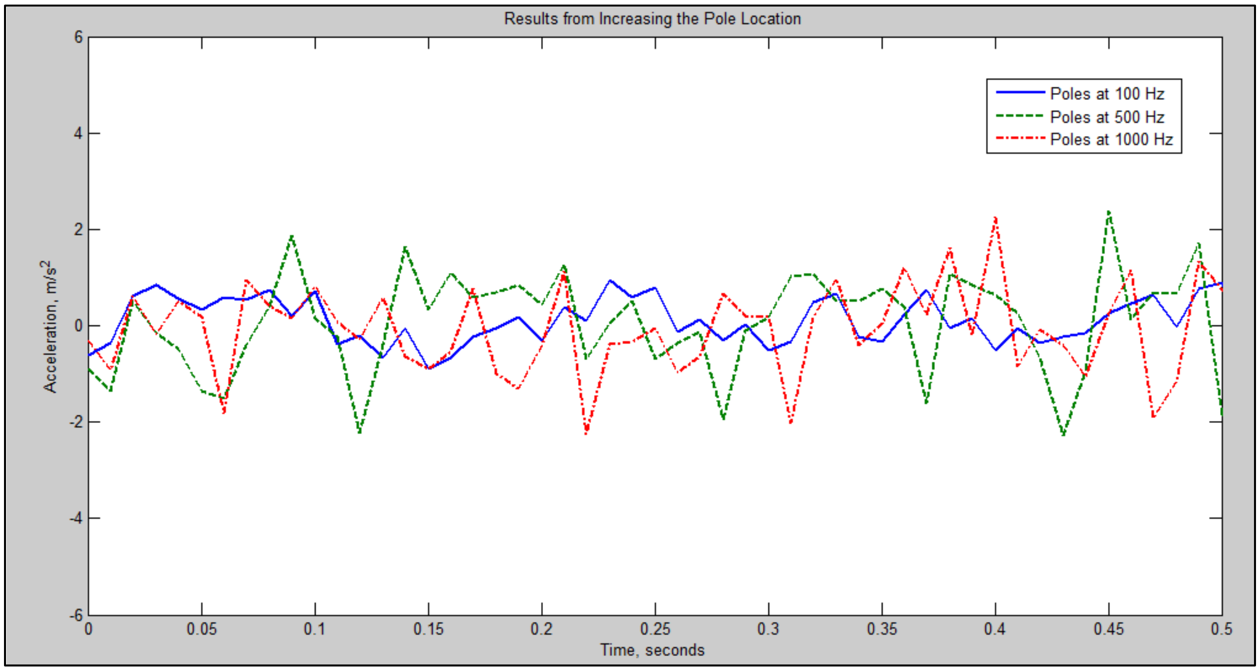


Figure 4-7: *The increase of the pole frequency had small effects on the stability of the acceleration*

To gain a better understanding of the three different pole locations, the standard deviation of each was evaluated from time equal to zero to time equal to 1 second. The result is in **Table 4-2**. The difference between 500 Hz and 1,000 Hz is approximately the same. Their difference accounted for the noise in the sensor.

Table 4-2: *The increase in pole placement increased the standard deviation of the acceleration*

Poles Location	Standard Deviation (m/s^2)
100 Hz	0.4743
500 Hz	1.0382
1000 Hz	0.9554

As the pole locations were moved further out, the acceleration increased in noise. This noise was attributed to an increase in noise in the force controller. The force controller was approaching the same order of magnitude as the force signal's low-pass filter. It was believed that this was the cause of the detrimental performance. To increase the performance of the force controller, the filter frequency was maximized. Its frequency was increased to 300 Hz, 600 Hz, and 1,000 Hz and tested. **Figure 4-8** shows the force signal after filtering.

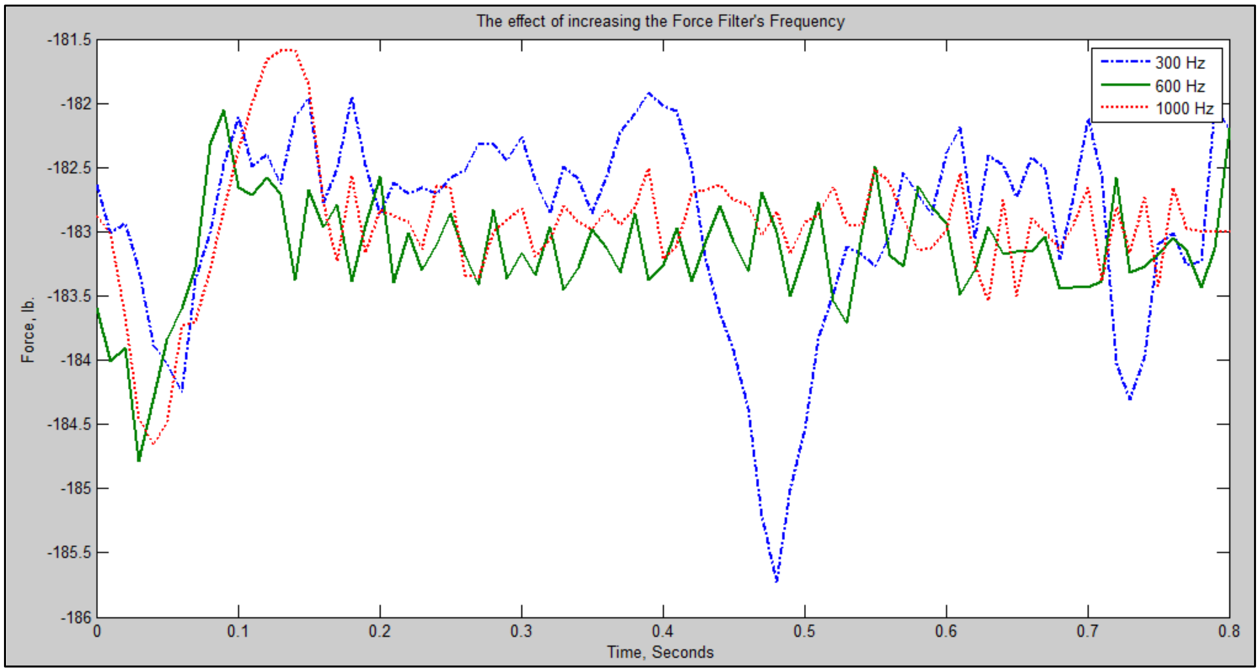


Figure 4-8: *The filter at 600 Hz provided the best average force result*

The interaction between the force controller frequency and the force low-pass filter is shown in the 1,000 Hz plot in **Figure 4-9**. During operation, MoRGS opposed the motion of the operator and had unintended and unsafe motions. Therefore, the 1,000 Hz filter was not viable for further use. To compare the 300 Hz and the 600 Hz filter frequencies, the standard deviation was computed. **Table 4-3** shows that the best solution is a force filter at 600 Hz.

Table 4-3: *The 600 Hz filter provided the lowest standard deviation*

Force Filter Location	Standard Deviation (lb.)
300 Hz	0.7995
600 Hz	0.4291
1000 Hz	0.5286

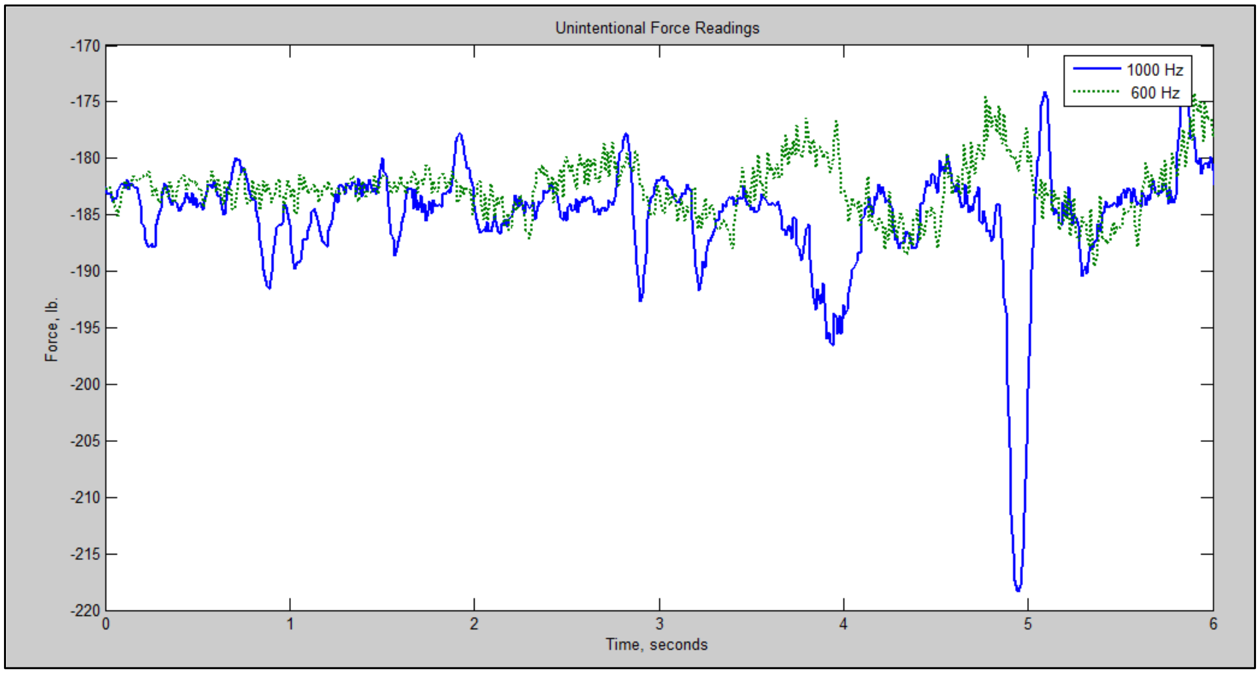


Figure 4-9: *The force filter did not function properly at 1000 Hz*

The next step in tuning MoRGS was maximizing the gain. It was understood that this system was not going to have tremendously high bandwidth (less than 30 Hz) but all efforts were made to maximize how much the gain could be increased without driving the system unstable. All of the previous steps in this tuning process, the zero placement, the pole placement, and the filter placement were means for taking advantage of all possible ways to squeeze out any additional bandwidth for MoRGS. The three gains tested were 1.0, 1.25, and 1.5. The gain is applied before the integration of the acceleration step of the control algorithm. The complete implementation in the microprocessor can be found in the source code in Appendix G. To compare the three gains, the velocity response is plotted along with the force error.

Since the velocity and the force are neither in the same units nor magnitude, the force is scaled to match the magnitude of the velocity, and in this case 10 times.

In the experimental data shown, the velocity is generated by a human operator moving the hanging mass manually. Ideally the force error will be zero. The elapsed time between the peaks of the force and velocity sinusoids was measured and the gain with the least time delay was selected. **Figure 4-10** shows the three gains and their velocity performance responding to an extremely aggressive force profile. An estimation of the time measured is shown in **Figure 4-10** by the data labels and the actual elapsed time is computed in **Table 4-4**. Stability was still kept in consideration because when the gain exceeded 1.5, the system's force controller became unsafe for the operator as evidenced by the instability in the force in **Figure 4-11**. Therefore, the gain selected for MoRGS was 1.50.

Table 4-4: *The time delay for velocity response to a force disturbance for each gain tested*

Gain	Elapsed Time (s)
1.00	0.58
1.25	0.51
1.50	0.40
1.75	0.3
2.00	0.21
2.25	0.17

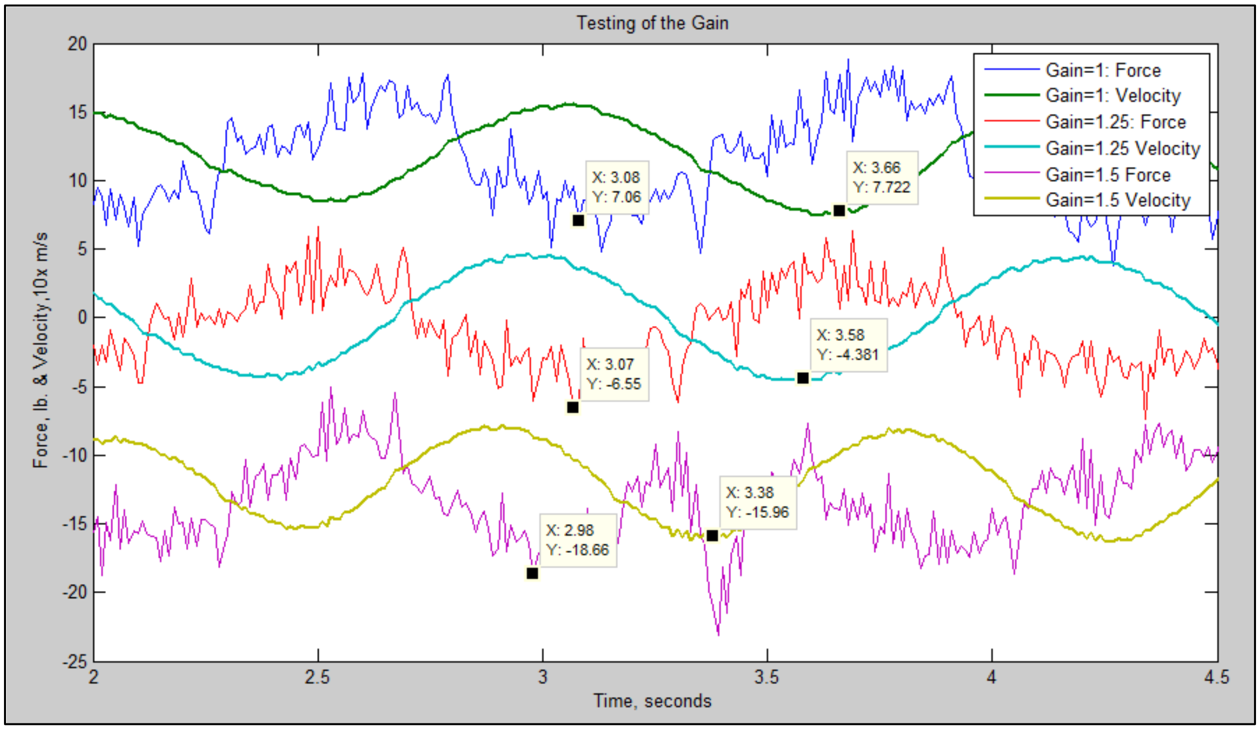


Figure 4-10: *Three different gains were tested with the three force disturbances and velocity responses plotted together. Force error and velocity values are offset for plot clarity.*

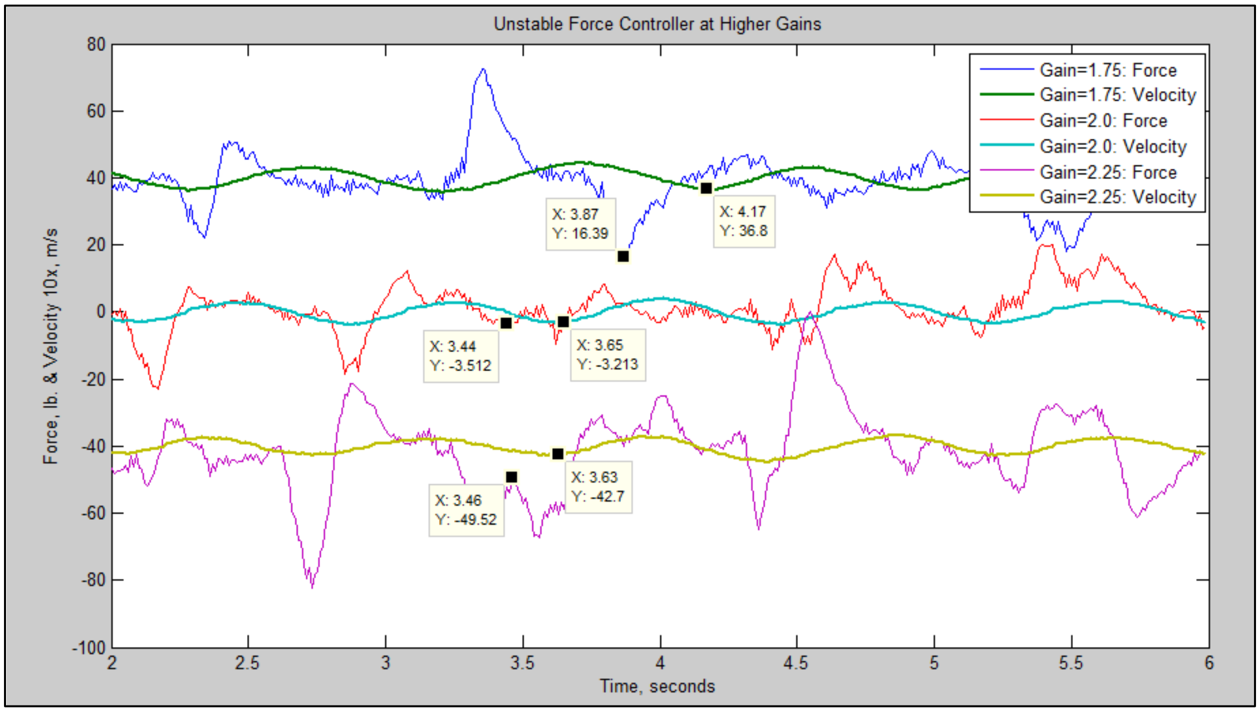


Figure 4-11: *Three higher gains were tested but had unfavorable results. Force error and velocity values are offset for plot clarity.*

4.2.3 Verifying the Model

The velocity response to a force disturbance for both the MATLAB model and MoRGS was compared. A similar force input was used and the two velocity responses, one from the model and the other from actual physical testing, are plotted in **Figure 4-12**. As shown by this figure, the two responses match very well for their frequency, and their phase shift delay from the force disturbance. The velocity of MoRGS is approximately twice in magnitude to that of the model but this difference has been attributed to the operator of MoRGS difficulty in maintaining smooth force applied. The model was verified to match the physical system and offers the opportunity for future investigation into adding additional features to MoRGS.

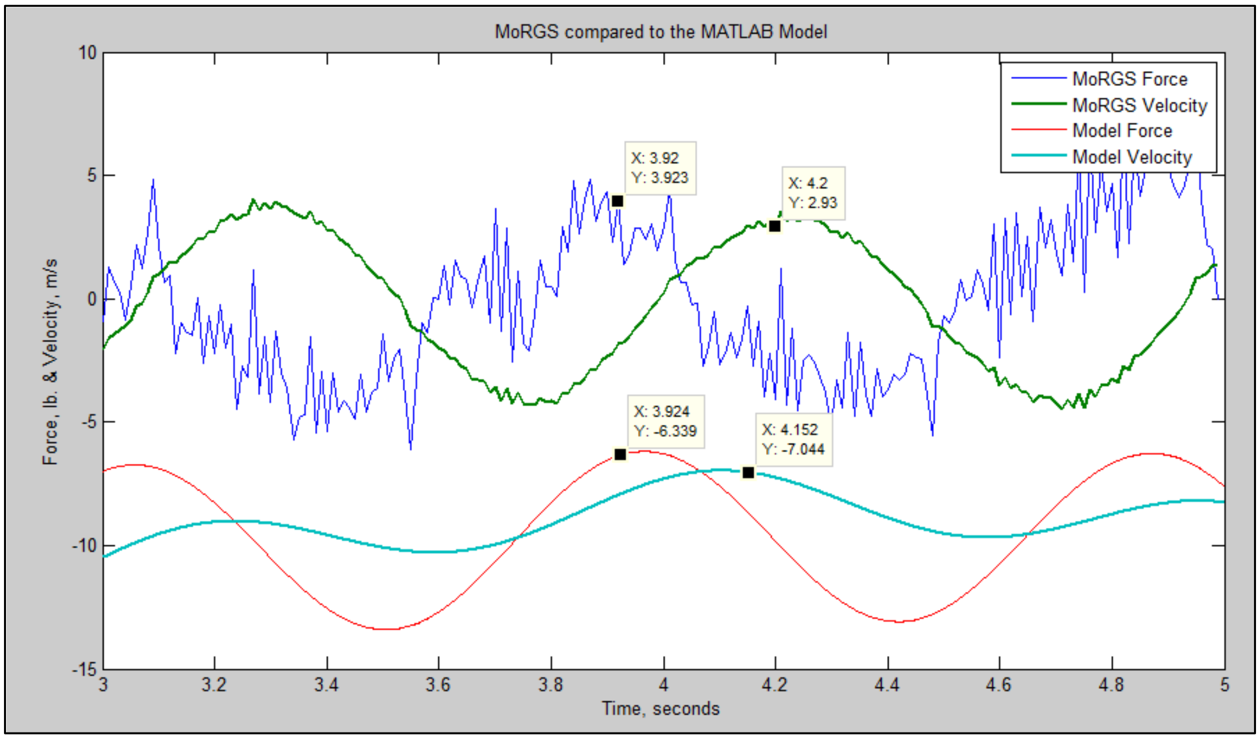


Figure 4-12: *The model is a close match to the actual MoRGS system*

4.2.4 Spectrum of Loads

In the simulation, the force controller was capable of controlling a very wide range of load values. This was explained by the assumption that a two mass system separated by a spring can be assumed to be one mass attached to a spring with one end fixed if the natural frequency ratio of the two masses is at least 10 times. Section 3.3.1 shows the complete analysis. The loads selected for this test were 45 lb., 90 lb., 140 lb. and 180 lb. based on the available weights. **Figure 4-13** shows the complete results of the tracking of the force error for all four loads. In addition the responses to disturbances generated by a human are shown in **Figure 4-14** to **Figure 4-17**. They show both the velocity and force as discussed for **Figure 4-10** to **Figure 4-12**. There seemed to be no appreciable differences for the different loads and maintained stability throughout.

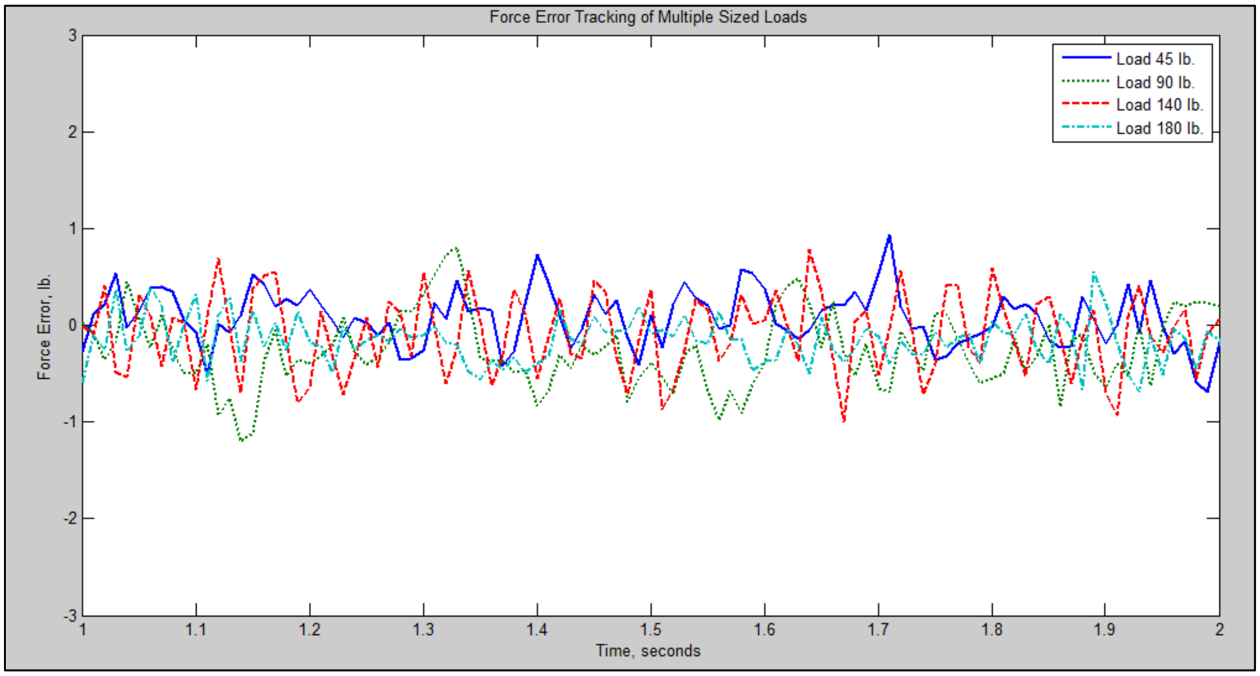


Figure 4-13: *The single force controller maintains the cable force for all loads applied*

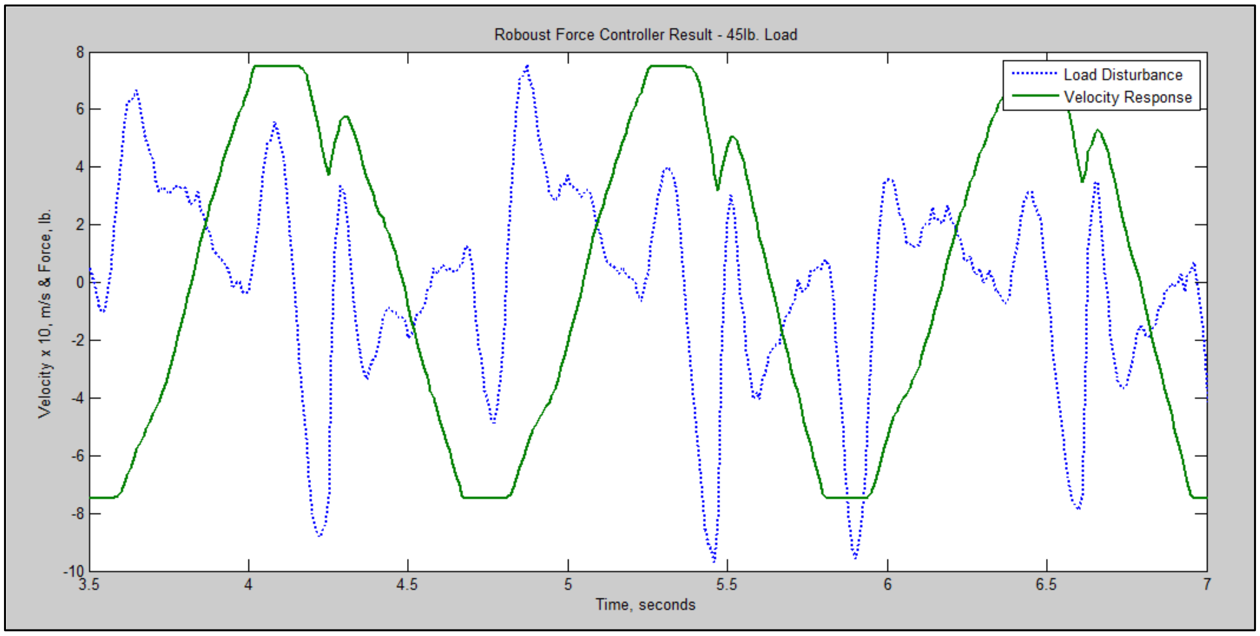


Figure 4-14: *The force controller performance with a 45 lb. load and a cyclic human disturbance*

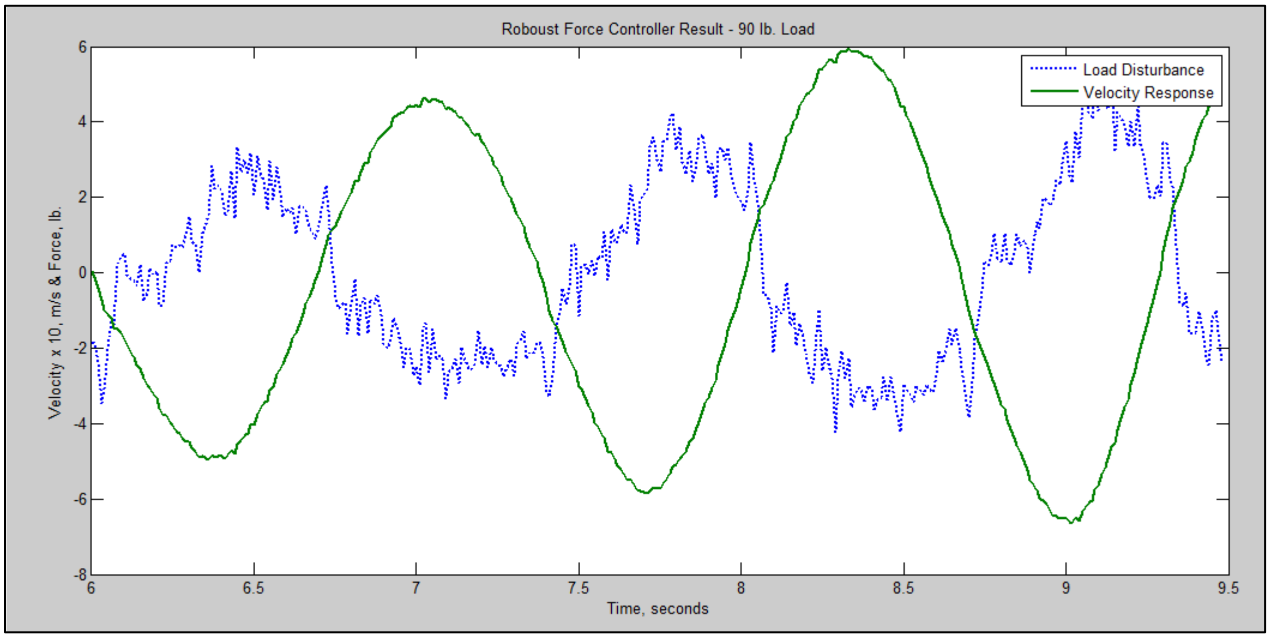


Figure 4-15: *The force controller performance with a 90 lb. load and a cyclic human disturbance*

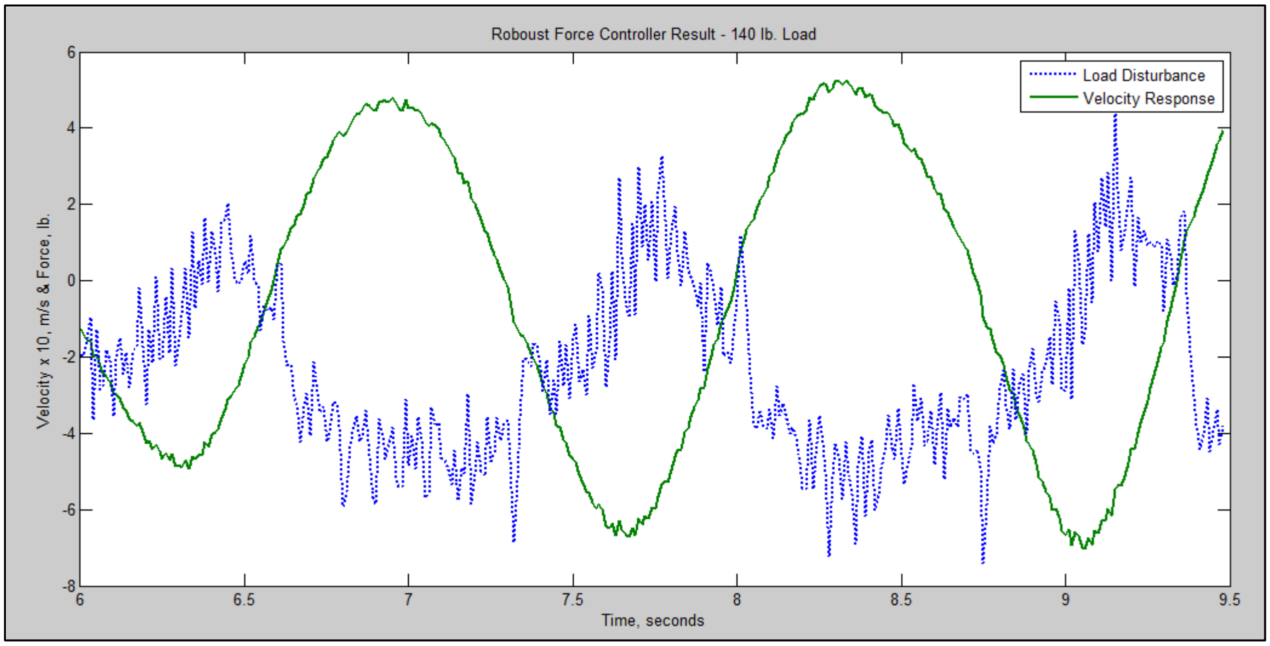


Figure 4-16: *The force controller performance with a 140 lb. load and a cyclic human disturbance*

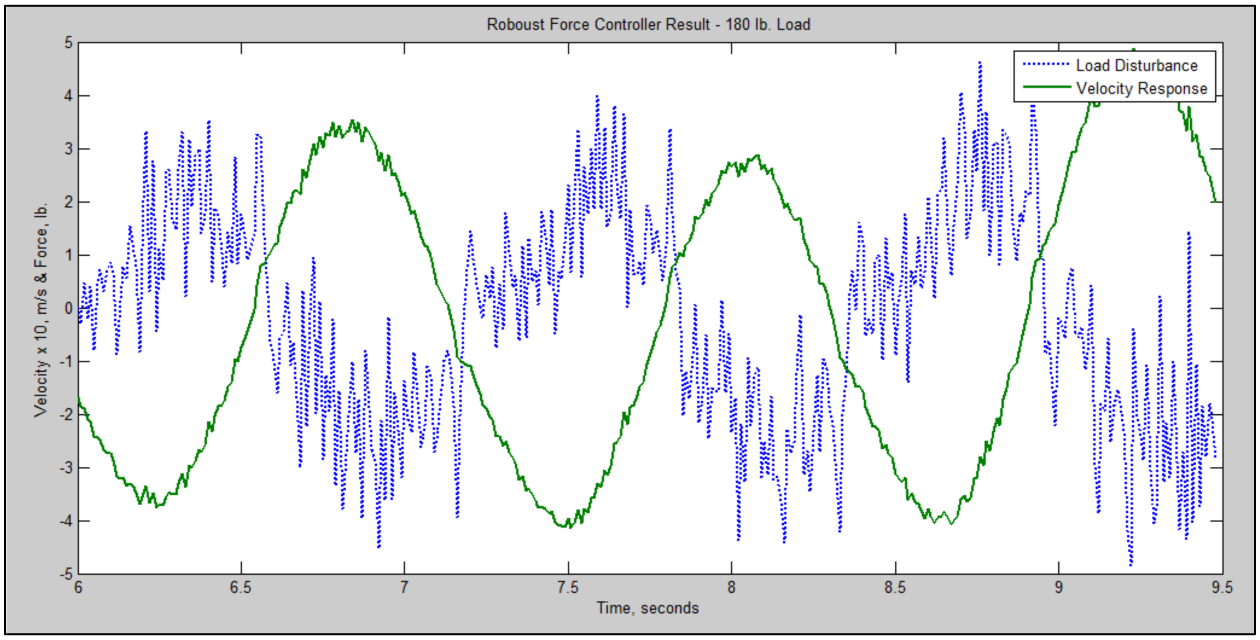


Figure 4-17: *The force controller performance with a 180 lb. load and a cyclic human disturbance*

Chapter 5

Conclusions and Future Work

Although several reduced gravity simulators already exist either at universities or at NASA, no simulators were suited for the unique needs of the KIN Dept. The necessity of a portable, interchangeable, and low-cost simulator lent itself to a custom designed and built simulator named MoRGS. A thorough study of reduced gravity simulators generated a hybrid concept. Many of the lessons NASA learned from its previous simulator, ARGOS, were implemented in MoRGS.

MoRGS was designed with a 48v DC power supply coupled with a high current motor controller. The brushless DC motor was selected after rigorous analysis of torque and speed with respect to the gearbox size and the custom drum. A strain gauge load cell provides feedback to the controller inside the microprocessor. A velocity loop for the motor and a force loop for the entire system were designed for maximum disturbance rejection at a system bandwidth of 30 Hz.

The initial selection of a digital load cell converter did not provide an adequate update rate. Instead an analog signal is sent to the microprocessor where it is read through an analog to digital converter in addition to significant signal processing. A shunt regulator was added to the power circuitry to regulate the system’s voltage during regeneration situations for example when the load is lowered at high speeds. For the future, the next product series should be used for the motor controller that tolerates 75 V. In addition, the power supply voltage can be increased to 60v to further meet NASA's velocity requirements. The MoRGS system can reach a speed of 2.01 m/s but is regulated to 0.75 m/s because it was an adequate speed for testing while maintaining a sense of safety.

Analysis of the natural frequency of the structure that MoRGS is secured to is a priority, especially if the system is added to an X and Y simulator. The current MoRGS design uses a 6-ton I-beam, although the maximum load is only ½ ton because of the negative effect the smaller I-beam's natural frequency has on the system, proven in testing.

MoRGS proved to be a robust reduced gravity simulator in terms of its controllability over a range of loads. The unique feature of the smallest of loads being significantly higher compared to the motor in terms of its natural frequency allows the system to have one type of controller that is capable of controlling a load from 50 lb. to 700 lb. In addition, the 4 step signal processing of the load cell allows the analog signal to travel 45 ft. around noisy electrical components and still maintain a standard deviation of 0.13 lb.

Future work will need a thorough analysis of safety in the mechanical components and the software. A refined analysis of the dynamic performance using frequency response will be necessary. Further study will revolve around the safe and low-cost implementation of MoRGS in an X-Y gantry system while maintaining a high natural frequency and disturbance rejection. Sensing the X-Y position of the human will be challenging, possibly requiring complex and expensive sensors.

Bibliography

[1] Ade, Carl (2011). POGO at Johnson Space Center. Houston, TX.

[2] Baldor (2009). AC servo motors and servo rated gearheads for the automation industry. (BR1202-E). Fort Smith, AR. Retrieved 6/25, 2012, from http://www.baldor.com/support/Literature/Load.ashx/BR1202-E?LitNumber=BR1202-E

[3] Bibby, Joe (2011). NASA software robotics and simulation division-ER5 (Dynamic systems test branch – ARGOS). Retrieved 6/25, 2012, from http://er.jsc.nasa.gov/ER5/

[4] Budynas, R., & Nisbett, K. (2010). Shigley's mechanical engineering design (9th ed.) McGraw-Hill.

[5] Colgate, J. E., Peshkin, M., & Klostermeyer, S. H. (2003). Intelligent assist devices in industrial applications: A review. Intelligent robots and systems, 2003. (IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference, 2516-2521 vol.3.

[6] Collins, J. W. (2012). Sonny carter training facility - neutral buoyancy laboratory. Retrieved 6/25, 2012, from http://dx12.jsc.nasa.gov/site/index.shtml

[7] Cunningham, T. (2009). Festo muscle testing. Houston, TX: NASA.

[8] Cunningham, T. (2010). System requirements document for the active response gravity offload system (ARGOS). (Engineering directorate No. AR&SD-08-007). Houston, TX: NASA.

[9] Dungan, L. (2011). NASA ARGOS conference call meeting

[10] Elmo Motion Control (2011). SimlIQ software manual. (MAN-SIMSW). Nashua, NH.

[11] Gorbel (2012). Fixed height gantry cranes. Retrieved 6/25, 2012, from https://www.gorbel.com/Products/Gantry/fixedgantry.aspx

[12] Gorbel (2008). G-Force Q and iQ Series installation & operation manual. Fishers, NY. Retrieved 6/25, 2012, from https://gorbel.com/resources/products/intelligentassistdevices/gforce/documents/Q-iQManual.pdf

[13] Grob Gear (2012). Grob stand spline CAD technology. Retrieved 6/25, 2012, from http://oem.cadregister.com/asp/PPOW_Entry.asp?company=445809&elementID=28186005/GSS/XM0001/3&orderedResponses=21|0|0|0|0|0|0|0|0|0|0|1|

[14] Hall, K. L. (2011). Dynamic control for a pneumatic muscle actuator to achieve isokinetic muscle strengthening. Ph.D., Wright State University, Dayton, OH.

[15] Hill, T. R., & Johnson, B. J. (2010). EVA space suit architecture: Low earth orbit vs. moon vs. mars. Aerospace Conference, 2010 IEEE, 1-28.

[16] Kim, Y., Jhung, M. J. (2010). Mathematical analysis using two modeling techniques for dynamic responses of a structure subjected to a ground acceleration time history. Nuclear Engineering and Technology, Vol. 43 No. 4.

[17] Lichtenberg, B. K. (2008). Interface control document: Boeing 727-200. (No. ZGC-ICD). Vienna, VA: Zero Gravity Corporation.

[18] Loadstar Sensors (2011). RAS1 S beam load cell. Fremont, CA.

[19] Norcross, J. R., Chappell, S. P., & Gernhardt, M. L. (2010). Lessons learned from performance testing of humans in spacesuits in simulated reduced gravity. HRP Investigators' Workshop, 1.

[20] Perusek, G. P., DeWitt, J. K., Cavanagh, P. R., Grodsinsky, C. M., & Gilkey, K. M. (2007). Zero-gravity locomotion simulators: New ground-based analogs for microgravity exercise simulation. ESA Symposium: Technology for artificial gravity and microgravity simulation, Noordwijk; Netherlands.

[21] Straube, T. M. (1993). A large motion zero-gravity suspension system for experimental simulation of orbital construction and deployment. M.S., University of Colorado, Boulder, CO.

[22] The Stanley Works (2005). Stanley Assembly Technologies. Cleveland, Ohio. Retrived 6/25, 2012, from http://www.emhartamericas.com/sites/www.emhartamericas.com/files/downloads/stanley-assembly-cobotics-brochure.pdf

[23] Van Dijk, A. (2012). Flight opportunities program: G-force one. Retrieved 6/25, 2012, from https://flightopportunities.nasa.gov/platforms/parabolic/gforce-one/

Appendix A

Motor Sizing MATLAB

```
% Tim Mourlam
% MoRGS Motor Sizing


drum_inertia = .0568; %lb-in-s^2

%Angular Velocity of Motor Required
w625 = 10.6*.78; %rad/sec .6
w440 = 10.6*.85;%8.6*.9; %rad/sec .2
w250 = 10.6*.93;%26.6*.28; %rad/sec .7

%Angular Acceleration of Motor Required
max_accel_625 = 77.3; %rad/s^2
max_accel_440 = 77.3;%133.3*.9; %rad/s^2
max_accel_250 = 77.3;%234.6*1; %rad/s^2

motor_inertia = .0273; %lb-in-s^2
R = .25; %ohms
Kt = 1.79; %N-m/amp
V = 60; %volts
Vcpeak = 153.2; %Vpk/KRPM

%-----------------------------------------------------------
% 625lb load
max_accel = max_accel_625;
w = w625;
Gr = 3;
constant_torque625 = 234*12/Gr; %in-lbs load drum diam
constant_torque = constant_torque625;

total_inertia = drum_inertia*Gr^2+motor_inertia;
accel_torque = total_inertia*max_accel;
max_torque(1,1) =(constant_torque+accel_torque)*.113; %N-m

Kb=1/1000*60/(2*pi)*Vcpeak; %V-s/rad
k625 = roots([w*Gr -V max_torque(1,1)*R]);

Vpeak(1,1) = Kb*w*Gr + max_torque(1,1)/Kt*R;

%-----------------------------------------------------------
% 440lb load
max_accel = max_accel_440;
w = w440;
```

```
Gr = 3;
constant_torque440 = 165*12/Gr; %in-lbs load drum diam
constant_torque = constant_torque440;

total_inertia = drum_inertia*Gr^2+motor_inertia;
accel_torque = total_inertia*max_accel;
max_torque(1,2) =(constant_torque+accel_torque)*.113; %N-m

Kb=1/1000*60/(2*pi)*Vcpeak; %V-s/rad
k440 = roots([w*Gr -V max_torque(1,2)*R]);

Vpeak(1,2) = Kb*w*Gr + max_torque(1,2)/Kt*R;

%-------------------------------------------------------------
% 250lb load
max_accel = max_accel_250;
w = w250;
Gr = 3;
constant_torque250 = 93.75*12/Gr; %in-lbs load drum diam
constant_torque = constant_torque250;

total_inertia = drum_inertia*Gr^2+motor_inertia;
accel_torque = total_inertia*max_accel;
max_torque(1,3) =(constant_torque+accel_torque)*.113; %N-m

Kb=1/1000*60/(2*pi)*Vcpeak; %V-s/rad
k250 = roots([w*Gr -V max_torque(1,3)*R]);

Vpeak(1,3) = Kb*w*Gr + max_torque(1,3)/Kt*R;

Vpeak
max_torque
k=[k625 k440 k250]
constant_torque = [constant_torque625 constant_torque440 constant_torque250]
```

Appendix B

Microprocessor Pinout

Load Cell	***Digital***		
Color	**Description**	**Pin on Board**	**Pin on Software**
Red	5v	JP18-HS-2	
White	Rx	CN4-12	
Green	Tx	CN4-13	
Black	Gnd	CN4-1	
Velocity Command			
Color	**Description**	**Pin on Board**	**Pin on Software**
White	Analog (+)	CN3-18	PC8
Green	Analog (-)	CN3-17	PC9
Brake Power			
Color	**Description**	**Pin on Board**	**Pin on Software**
Red	5v Ref	CN5-25	Power +5
Green	Pwr Signal	CN2-6	PA6
White	ELMO Signal	CN2-3	PA3
Position Adjust			
Color	**Description**	**Pin on Board**	**Pin on Software**
Orange	Down	CN5-7	PI6
Purple	Up	CN5-3	PI5
Black	Ground	CN5-2	GND
Load Cell	***Analog***		**DB9 Load Cell**
Color	**Description**		**Color**
Green	Signal +		Brown
White	Signal -		Red
Red	Excitation +		Black
Black	Excitation -		Orange
Bare	Shield		
Amplifier Module			
Color	**Description**	**Pin DB9**	
Black	Signal Current	1	
Purple	Signal Voltage	8	
Gray	Power Supply (+)	9	
Blue	Ground (-)	7	

Appendix C

Discrete Filter Generation Algorithm

```
% Controller Filter Algorithm 7/9/2012

syms T p z b s wn zeta real

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 1 (2 zeros, 2 poles, 2 free integrators)
filt = (s+z)^2/(s+p)^2*(p/z)^2;

s = 2/T*(1-b)/(1+b);

filt = collect(expand(subs(filt)), b);

a2=(T^2*p^2*z^2 - 4*T*p^2*z + 4*p^2);
a1=(2*T^2*p^2*z^2 - 8*p^2);
a0=( T^2*p^2*z^2 + 4*T*p^2*z + 4*p^2);

b2=(T^2*p^2*z^2 - 4*T*p*z^2 + 4*z^2);
b1=(2*T^2*p^2*z^2 - 8*z^2);
b0= T^2*p^2*z^2 + 4*T*p*z^2 + 4*z^2;

C = (a0*R+a1*R_1+a2*R_2-b1*C_1-b2*C_2)/b0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Force Filter - Low Pass

filter = (wn)^2/(s^2+2*zeta*wn*s+(wn)^2);
s = 2/T*(1-b)/(1+b);
filter = collect(expand(subs(filter)), b);

c2=T^2*wn^2;
c1=(2*T^2*wn^2);
c0=  T^2*wn^2;

d2=(T^2*wn^2 - 4*zeta*T*wn + 4);
d1= (2*T^2*wn^2 - 8);
d0=T^2*wn^2 + 4*zeta*T*wn + 4;

F = (c0*I+c1*I1-d1*F1-d2*F2)/d0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 2 (PID with Free integrator)

filt = (s+z)^2;
```

```
s = 2/T*(1-b)/(1+b);

filt = collect(expand(subs(filt)), b);

e2=(T^2*z^2 - 4*T*z + 4);
e1=(2*T^2*z^2 - 8);
e0= T^2*z^2 + 4*T*z + 4;

f2= T^2;
f1=(2*T^2);
f0= T^2;

C = (e0*R+e1*R_1+e2*R_2-f1*C_1-f2*C_2)/f0;
```

Appendix D

Simulation Parameters

```
% NASA Mobile ARGOS Project
% Tim Mourlam
% 7/29/11
% Dynamic Simulation
% Variable Definitions and Declarations

%Structure of Parameters

%  Festo muscle
pressure = 344;                     % kPa
param.Kc = 17000 + 17.9*pressure;   % N/m :  Spring Constant for Cable "Muscle"
param.bc = 2900;  param.bc_compression = 1570;             % N-s/m

% Load mass
param.m  = 200;              % lbf      :  Load on System
param.m = param.m*4.448221615;  % kg
param.L = 5/6*param.m;
param.l = param.L/param.Kc;
%param.m = 6.80388555;       % Demo/Practice

% Drum
r = 0.1143;                  % m (9 in drum diameter)
Jd = 0.0096;                 % Based from Mass Analysis from SolidWorks
param.Jd = Jd;               % kg-m^2   :  Inertia of drum
param.Rd = r;                % m        :  Radius of drum
param.Bd  = 0.0005;          % N-m-s    :  Damping Coefficient - drum

% Baldor G BSM100-MRP120-5
param.Gr = 250;                             % Gear Ratio of Gearhead
param.Kb = 28;                            % N-m/arcm  : stiffness of gear box
param.Kb = param.Kb/pi*180*60;            % N-m/rad
param.bb = 1e2;                           % N-m-s/rad : ?? backlash damping of
gear box
param.bl = 10;%15;                        % arcmin    : backlash of gear box (at
load?)
param.bl = param.bl*param.Gr*pi/180/60; % rad at motor

%  Baldor BSM100N-3150AB
param.Kt = 1.79;                 % N-m/A  : motor torque constant
param.bm = 0.409;                % N-m/krpm : motor damping
param.Bm = param.bm/1000/pi*60/param.Rd^2; % N-s/m
param.Jm = 0.0031;               % kg-m^2 : motor inertia
param.Je = param.Jm*param.Gr^2 + param.Jd;
param.Be = param.Bm*param.Gr^2 + param.Bd;

% Controller
param.T = 1/10000;    % Discrete Time Step

param.CableForce = 250;   % Desired Cable Force


% References

% x(1)= x
% x(2)= xdot
% x(3)= theta_d
% x(4)= theta_ddot


% u(1)= Tm lbf-in   :  Motor Torque
% u(2)= Fd lbf      :  Disturbance Force

%Definitions for Masses_Velocity_Simulation
v1 = [1];
b1 = param.Be/param.Rd^2;
F_Fs = [param.Je/param.Rd^2 b1];
v2 = 1;
Fh_Fg_Fs = [param.m 0];
Fs = [param.bc param.Kc];
v1_v2 = [1 0];
```

Appendix E

Simulation Transfer Functions

```
% Transfer Functions for Velocity_Masses Simulation

% Reference

% G1 = Motor Plant
% G2 = Muscle Plant
% G3 = Load Plant
% G4 = Free Integrator for Velocity to Position
% Gcp = Motor Controller
% Gcs = Load Cell Controller

G1 = tf(v1,F_Fs);
G2 = tf(Fs,v1_v2);
G3 = tf(v2,Fh_Fg_Fs)*tf([2517 2517],[1 0]);
G4_p = tf(1,[1 0]);
G4_v = 1;
G4_a = tf([1 0],1);

[G4_pNum, G4_pDen] = tfdata(G4_p,'v');
 G4_vNum = 1; G4_vDen = 1;
[G4_aNum, G4_aDen] = tfdata(G4_a,'v');

% Inner Loop
% Cable Velocity (Motor Side) / Current Command --- Converted to Force
A_R = G1*(1+G2*G3)/(1+G1*G2+G2*G3)*(param.Kt*param.Gr/param.Rd);
%A_R = minreal(A_R);

param.KpC = 2143; param.KiC = 2143*191;

% Motor Controller
Gcp = tf([param.KpC param.KiC],[1 0]);
[GcpNum, GcpDen] = tfdata(Gcp,'v');

% Outer Loop
% Motor Velocity Command / Force Feedback
C_Qv = (G1*G3*Gcp)/(1+G2*G3+G1*G3+G1*G4_v*Gcp*(1+G2*G3));
C_QvMin = minreal(C_Qv);

param.KpV1 = 4.2e10;
param.KpV2 = 3.8e7;


% Outer Loop
% Motor Position Command / Force Feedback
C_Qp = (G1*G3*Gcp)/(1+G2*G3+G1*G3+G1*G4_p*Gcp*(1+G2*G3));
C_Qp = minreal(C_Qp);

% Force Feedback Controller - Velocity Controller
%GcsV = tf([param.KpV param.KiV],[1 0]);
%[GcsVNum, GcsVDen] = tfdata(GcsV,'v');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Outer Loop
% Motor Acceleration Command / Force Feedback
C_Qa = (G1*G3*Gcp)/(1+G2*G3+G1*G3+G1*G4_a*Gcp*(1+G2*G3));
C_Qa = minreal(C_Qa);

% Force Feedback Controller - Acceleration Controller
GcsA = tf([22743 6.889e5 1.801e6],[1 0]);
[GcsANum, GcsADen] = tfdata(GcsA,'v');

% Force Feedback Controller - Acceleration Controller
GcsApi = tf([5101 5101*24.2],[1 0]);
[GcsApiNum, GcsApiDen] = tfdata(GcsApi,'v');

% Human PD Controller at 3 Hz
Ghnum = [2511 2511*1];
Ghden = [1 0];
```

Appendix F

Simulation S-Function

```
function [sys,x0,str,ts]=S_Function2(t,x,u,flag,param)
% PIDfpSfun: s-function for a discrete time implentation of a PID controller
with
%  a fast pole

switch flag

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0
    [sys,x0,str,ts] = mdlInitializeSizes(param);

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1
    sys = mdlDerivatives(t,x,u,param);

  %%%%%%%%%%%%%%%%%%%%%%%%
  % Update and Terminate %
  %%%%%%%%%%%%%%%%%%%%%%%%
  case {2,9}
    sys = []; % do nothing

  %%%%%%%%%%
  % Output %
  %%%%%%%%%%
  case 3
    sys = mdlOutputs(t,x,u,param);

  otherwise
    error(['unhandled flag = ',num2str(flag)]);
end

% end limintm

%
%=============================================================================
% mdlInitializeSizes
% Return the sizes, initial conditions, and sample times for the S-function.
%=============================================================================
%
function [sys,x0,str,ts] = mdlInitializeSizes(param,xi)

sizes = simsizes;
```

```
sizes.NumContStates  = 0;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 2;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;

sys = simsizes(sizes);
str = [];
x0  =[];
ts  = [param.T 0];   % sample time: [period, offset]

% end mdlInitializeSizes

%
%=============================================================================
% mdlDerivatives
% Compute derivatives for continuous states.
%=============================================================================
%
function sys = mdlDerivatives(t,x,u,param)

% do nothing

% end mdlDerivatives

%
%=============================================================================
% mdlOutputs
% Return the output vector for the S-function
%=============================================================================
%
function sys = mdlOutputs(t,x,u,param)

persistent eC R C p z a2 a1 a0 b2 b1 b0 c2 c1 c0 d2 d1 d0 wn zeta Output e2 e1
e0 f2 f1 f0 Amax Vmax A_1 A R_1 R_2 C_1 C_2 F_2 F_1
persistent T uIkC KpC KiC i V_1 max_current% variables that need to be
remembered between calls

% 1 -> Inner Loop, 2 -> Outer Loop
if t==0 %|| t==param.T                          % initialize the memory
variables to zero (zero I.C.'s)
    i=0; uIkC=0; max_current = 56.0; % A
    T=param.T; KpC=param.KpC; KiC=param.KiC; uIC=0; Output=0;
    p= 3770; z= 18.85; Amax=15; Vmax=0.75; wn=6283; zeta=.707; V_1=0; A_1=0;
R_1=0; R_2=0; C_1=0; C_2=0;F_2=0; F_1=0;
    a2=(T*T*p*p*z*z - 4*T*p*p*z + 4*p*p); a1=(2*T*T*p*p*z*z - 8*p*p);
a0=T*T*p*p*z*z + 4*T*p*p*z + 4*p*p;
    b2=(T*T*p*p*z*z - 4*T*p*z*z + 4*z*z); b1=(2*T*T*p*p*z*z - 8*z*z);
b0=T*T*p*p*z*z + 4*T*p*z*z + 4*z*z;
    c2=(T*T*wn*wn); c1=(2*T*T*wn*wn); c0=T*T*wn*wn;
    d2=(T*T*wn*wn - 4*zeta*T*wn + 4); d1=(2*T*T*wn*wn - 8); d0=T*T*wn*wn +
4*zeta*T*wn + 4;
    z= 12; e2=(T*T*z*z - 4*T*z + 4); e1=(2*T*T*z*z - 8); e0= T*T*z*z + 4*T*z +
4;
```

```
    f2= T*T; f1=(2*T*T); f0= T*T; sys=14.2;
end
    R = u(2);
    F =  (c0*R+c1*R_1+c2*R_2-d1*F_1-d2*F_2)/d0;
    C = (a0*F+a1*F_1+a2*F_2-b1*C_1-b2*C_2)/b0; %
    %C = (e0*R+e1*R_1+e2*R_2-f1*C_1-f2*C_2)/f0;
    A = A_1+1.5*0.569*C*T; %// 2.5 for small load, 2.0 for large load
  if(A > Amax) %// 0.5 of Max Peak Velocity (RPM) converted to counts/sec
    A = Amax;
  end
  if(A < -Amax)
     A = -Amax;
  end
  V = V_1+A*T;
  if(V > Vmax) %// old:  2.356 0.5 of Max Peak Velocity (RPM) converted to
counts/sec
     V = Vmax;
  end
  if(V < -Vmax)
    V = -Vmax;
  end
  A_1 = A;
  V_1 = V;
  F_2=F_1;
  F_1=F;
  R_2 = R_1;
  R_1 = R;
  C_2 = C_1;
  C_1 = C;

    eC = eC+V-u(1);

  if i==2     % 220 Hz loadcell, 10000 Hz Motor
      eC=eC/3;
    uIC = uIkC + KiC*T*eC;

    if uIC>0.25*max_current %1/2 Saturation Current from Motor
        uIC=0.25*max_current;
    end
    if uIC<-0.25*max_current %1/2 Saturation Current from Motor
        uIC=-0.25*max_current;
    end

    uoutC = KpC*eC + uIC;

    if uoutC>max_current  % Saturation Torque from Motor
        uoutC=max_current;
    end
    if uoutC<-max_current  % Saturation Torque from Motor
        uoutC=-max_current;
    end
      i=0;
      eC=0;
      uIkC=uIC;
      Output = uoutC;
  end
```

```
    i=i+1;
    sys=Output;


% end mdlOutputs
```

Appendix G

Software C Code in Microprocessor

```
/**
  ******************************************************************************
  * @author  Tim Mourlam - MoRGS System
  * @version V1.0.0
  * @date    6/21/2012
  ******************************************************************************/

/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "stm32f4xx.h"
#include "stm32_eval_legacy.h"
#include "stm324xg_eval_lcd.h"
#include <stdio.h>
#include <stdlib.h>  //for atof() function

/* Private typedef -----------------------------------------------------------*/
DMA_InitTypeDef  DMA_InitStructurex;
DMA_InitTypeDef  DMA_InitStructurey;

GPIO_InitTypeDef  GPIO_InitStructure; //TO TEST TIME NEEDED FOR INTERRUPT

/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

//typedef enum {BUFFER_NO1=0,BUFFER_NO2=1}BUF_NO;
//BUF_NO Free_Buffer_No;
uint8_t BUFFER_NO1=1;
uint8_t BUFFER_NO2=2;
uint8_t Free_Buffer_No; //To check whick buffer is free
uint8_t Buffer_Ok;    //To check if any buffer is filled with bytes

uint8_t USARTy_DMA_Buffer1[14];  //The 1st variable to receive bytes
uint8_t USARTy_DMA_Buffer2[14];  //The 2nd variable to receive bytes

uint8_t USART_DMA_Buffer1[14];  //The 1st variable to receive bytes
uint8_t USART_DMA_Buffer2[14];  //The 2nd variable to receive bytes

uint8_t StringBuffer1[12]; //For display BUFFER1
uint8_t StringBuffer2[12]; //For display BUFFER2

//There will be something wrong if the display length is 12 bytes;
uint8_t DisplayFirstLine[14]; //To save bytes displyed on the 1st line of LCD
uint8_t DisplaySecondLine[14];//To save bytes displyed on the 2nd line of LCD
uint8_t DisplayTemp[12];
uint8_t DisplayCounter;

//The following variables are to convert the bytes to float type variable.
char ByteToStringBuffer1[12];  //
char ByteToStringBuffer2[12];
extern float ForceValue=0;
float ForceValueBuffer1;
float ForceValueBuffer2;
```

```
uint8_t CmdMotorVelString;
uint8_t CmdBufferTransmit [0x04] = "WC\n\r";    // Continuously Transmit Load Cell
uint8_t CmdBufferTare     [0x06] = "TARE\n\r";  // Tare the Load Cell
uint8_t CmdMotorEcho      [0x05] = "EO=0";  //To motor controller to turn off Echo

__IO uint32_t TimeOut = 0x0;
__IO JOYState_TypeDef PressedButton = JOY_NONE;
/* Private function prototypes -----------------------------------------------*/
static void TimeOut_UserCallback(void);
static void USART_Config(void);
static __IO uint32_t TimingDelay;
RCC_ClocksTypeDef RCC_Clocks;

/* NVIC configuration */
static void NVIC_Config(void);

// globals
#define NUMSAMPS 950
extern float Data[NUMSAMPS][5];
extern __IO uint8_t StoreMode=0;
extern __IO uint16_t StoreDelay=99;
extern int StoreDelayCnt=0;
extern int DataIdx=0;

void CharToStringToFloatBuffer1(void); //For Buffer1
void CharToStringToFloatBuffer2(void); //For Buffer2
void Display_Init(void);
void GPIOm_Init(void);
void ADC3_CH4_DMA_Config(void);
void SetLoad(void);
void ModeDisplay(void);
void LoadCellDisplay(void);
void USART_InitLoadCell(void);
void MotorVelCommand(int CmdAction, int Size);
void sysTickCallback(void);
void sysTickSetup(void);
void OutputDutyCylcleOnPWM(float dutyCycle);
void PwmOutInit(void);
void SetDutyCycle(float DesiredDutyCycle);
int FindSize(int VelVal);
int SizeV;
float Duty=0.0;
//char  *VelCmdString;
//char *VelVal;
extern int Mode = 1;
//extern int i = 1;
extern char VelCmdString[13] = "0";
//extern char VelVal[6] = "0";
extern float uIkV   = 0.0;
extern float KpV    = 16115.0;
extern float KiV    = 443162.5;
extern float eV     = 0.0;
extern float VelCmd = 0.0;
extern float T      = 0.0001;
extern float uIV    = 0.0;
extern float InitLoad = 0.0;
extern float OffLoad=0.0;
extern float A_1 = 0.0;
extern float V_1 = 0.0;
extern float V   = 0.0;
extern float A   = 0.0;
extern float eV_1 = 0.0;
extern float F_1 = 0.0;
extern uint8_t Size = 0;
extern uint8_t Start [0x3] = "BG";
extern uint8_t CmdMotorOn   [0x05] = "MO=1";  //To motor controller
extern uint8_t CmdMotorOff  [0x05] = "MO=0";
extern uint8_t CmdMotorUp   [0xA]  = "JV=900";
extern uint8_t CmdMotorDown [0xB]  = "JV=-900";
extern uint8_t CmdMotorHold [0x08] = "JV=0";
extern uint8_t CmdMotorRMon [0x05] = "RM=1";
extern uint8_t CmdMotorRMoff [0x05] = "RM=0";
float a2, a1, a0, b2, b1, b0, c2, c1, c0, d2, d1, d0, e2, e1, e0, f2, f1, f0;
float p = 3770.0, z = 18.85; //z=12 for controller 2
float zeta=0.707, wn = 6283.0; // rad/s //6.2831853 rad/s / Hz
extern int i=0;
extern int J=0;
extern float mv=0, v=0;
extern float GravityRatio = 1.0;  //Gravity Ratio - Moon, Mars, Earth
uint16_t SysTickFrequency = 1.0/0.0001;  // Frequency in 10kHz
uint32_t d_ARR; // ARR clock counts to get period
__IO uint16_t ADC3ConvertedValue[10];
__IO float ADC3ConvertedVoltage = 0;
//__IO uint16_t ADC3ConvertedVoltage_noF=0;
#define ADC3_DR_ADDRESS    ((uint32_t)0x4001224C)
#define ADC1_DR_ADDRESS    ((uint32_t)0x4001204C)

int main(void)
{
  /*!< At this stage the microcontroller clock setting is already configured,
       this is done through SystemInit() function which is called from startup
       file (startup_stm32f2xx.s) before to branch to application main.
       To reconfigure the default setting of SystemInit() function, refer to
       system_stm32f2xx.c file
     */
  //Delay(40000000);

  /* ADC3 configuration *******************************************************/
  /*  - Enable peripheral clocks                                              */
  /*  - DMA2_Stream0 channel2 configuration                                   */
  /*  - Configure ADC Channel7 pin as analog input                            */
  /*  - Configure ADC3 Channel7                                               */
  Display_Init();

  a2=(T*T*p*p*z*z - 4*T*p*p*z + 4*p*p); a1=(2*T*T*p*p*z*z - 8*p*p); a0=T*T*p*p*z*z +
4*T*p*p*z + 4*p*p;
  b2=(T*T*p*p*z*z - 4*T*p*z*z + 4*z*z); b1=(2*T*T*p*p*z*z - 8*z*z); b0=T*T*p*p*z*z +
4*T*p*z*z + 4*z*z;
  c2=(T*T*wn*wn); c1=(2*T*T*wn*wn); c0=T*T*wn*wn;
  d2=(T*T*wn*wn - 4*zeta*T*wn + 4); d1=(2*T*T*wn*wn - 8); d0=T*T*wn*wn + 4*zeta*T*wn +
4;
  e2=(T*T*z*z - 4*T*z + 4); e1=(2*T*T*z*z - 8); e0= T*T*z*z + 4*T*z + 4;
  f2= T*T;  f1=(2*T*T); f0= T*T;

  PwmOutInit();
  ADC3_CH4_DMA_Config();
  /* Start ADC3 Software Conversion */
  ADC_SoftwareStartConv(ADC1);
  // Add the Carriage Return to Common Motor Commands
  Start        [2]    = 13;
  CmdMotorOn   [4]    = 13; CmdMotorOff [4] =13; CmdMotorRMon [4]=13; CmdMotorRMoff[4] =
13;
  CmdMotorHold [4]    = 13; CmdMotorHold[5] =66; CmdMotorHold [6]=71; CmdMotorHold[7]
=13;
```

```
  CmdMotorUp   [6]    = 13; CmdMotorUp  [7] =66; CmdMotorUp   [8]=71; CmdMotorUp  [9]
=13;
  CmdMotorDown [7]    = 13; CmdMotorDown[8] =66; CmdMotorDown [9]=71;
CmdMotorDown[10]=13;
  /* USART configuration ------------------------------------------------------*/
  USART_Config();        // Setup the USART for DMA for DMA1 and DMA2
  USART_InitLoadCell();  // Send the initial command to the Loadcell and Motor
  sysTickSetup();
  GPIOm_Init();
  GPIOH->BSRRL = GPIO_Pin_9; //Turn on Pin6 -- Power to ELMO On
  //SAFETY Turn off ELMO
  //GPIOA->BSRRH = GPIO_Pin_6;  //Turn off Pin6
  Size = sizeof(CmdMotorOff);
  MotorCommand_USART3((int)CmdMotorOff, Size);

  while(1)
  {
    SetLoad();
    ModeDisplay();

    //SetDutyCycle(.0072);-0.135
    //SetDutyCycle(0.0);
    //LoadCellDisplay();
    //while (StoreMode);
  }
}
/*************************************************************************/
/*************************************************************************/
void SetLoad(void)
{
  uint8_t text[50];
  uint8_t load[50];
  uint8_t Line[50];


  ADC3ConvertedVoltage = ADC3ConvertedValue[0] *3300.0f/0xFFF; //3300
  mv = ADC3ConvertedVoltage;
  //while(J<100000){J=J+1;}
  sprintf((char*)Line,"Load (lbs) = %d   ", (int)ForceValue);
  LCD_DisplayStringLine(LINE(6), Line);
  sprintf((char*)text,"Init Load = %d     ",(int)InitLoad);
  LCD_DisplayStringLine(LINE(7),text);
  sprintf((char*)load,"Loadcell (mV) = %d    ",(int)mv);
  LCD_DisplayStringLine(LINE(8),load);


}
/*************************************************************************/
/*************************************************************************/
void VelocityControl(float ForceValue)
{
 // GPIOA->BSRRL = GPIO_Pin_3;
  static float R, R_1=0.0, R_2=0.0, C, C_1=0.0, C_2=0.0, Amax =15, Vmax = 0.75; // m/s
  //Vmax based on maximum motor RPM (45) that avoids regenerative voltage to surpass 60V
(elmo capacity)
  R = (OffLoad+ForceValue)*4.4482216f;  // Convert to N

  C = (a0*R+a1*R_1+a2*R_2-b1*C_1-b2*C_2)/b0;
  //C = (e0*R+e1*R_1+e2*R_2-f1*C_1-f2*C_2)/f0;

  A = A_1+1.5f*0.569f*C*T; // 0.569 based on crossover shift from the zero's translation
  if(A > Amax) // 0.5 of Max Peak Velocity (RPM) converted to counts/sec
  {
    A = Amax;
  }
  else if(A < -Amax)
  {
    A = -Amax;
  }
  V = V_1+A*T;
  if(V > Vmax) // old:  2.356 0.5 of Max Peak Velocity (RPM) converted to counts/sec
  {
    V = Vmax;
  }
  else if(V < -Vmax)
  {
    V = -Vmax;
  }
  A_1 = A;
  V_1 = V;
  R_2 = R_1;
  R_1 = R;
  C_2 = C_1;
  C_1 = C;
}
/**********************************************************************/
/**********************************************************************/
void sysTickCallback(void)
{
  static float temp, VmotorMax = 2.356; // m/s  --> max speed of Motor, based at the
cable
  static float DutyCycle, I, I_1=0, I_2=0, F, F_1=0, F_2=0;
  I=0;
  for( i=0;i<10;i++)
  {
    I = (ADC3ConvertedValue[i]+I);
  }
  I = I/(i+1);
  I = (I*3300.0f/0xFFF);
  //I = ADC3ConvertedValue*2970.0f/0xFFF;//V  // PA 2
  F =  (c0*I+c1*I_1+c2*I_2-d1*F_1-d2*F_2)/d0;
  if(StoreMode)
        {
          StoreDelayCnt++;
          if (StoreDelayCnt > StoreDelay)
          {
            StoreDelayCnt=0;
            Data[DataIdx][0]=ForceValue; Data[DataIdx][1]=V;
            Data[DataIdx][2]=A; Data[DataIdx][3]=I;
            Data[DataIdx][4]=mv;
            DataIdx++;
          }
          if (DataIdx == NUMSAMPS)
          {
            DataIdx=0; StoreMode=0; Data[0][0]=T*(StoreDelay+1);
          }
        }
  if (Mode==3)
  {
    ForceValue = -(0.1001f*F - 63.887f);  // lbs
    VelocityControl(ForceValue);
    temp=V;
  }
  else
  {
    temp=0.0;
    ForceValue = -(0.1001f*F - 63.887f);  // lbs
  }
  DutyCycle =  temp/VmotorMax;
  SetDutyCycle(DutyCycle);
  F_2=F_1;
  F_1=F;
  I_2=I_1;
  I_1=I;

  //SetDutyCycle(Duty);
}
/*************************************************************************/
/*************************************************************************/
void SetDutyCycle(float DesiredDutyCycle)
{
  if(DesiredDutyCycle<0) // Negative Velocity Desired (DOWN)
  {
    DesiredDutyCycle*=-1;
    // Channel 1
    TIM3->CCR3 = (uint32_t) ((0.0f+0.029f)*d_ARR); // Set PWM of Negative differential
to zero
    // Channel 2
    TIM3->CCR4 = (uint32_t) ((DesiredDutyCycle+0.0072f)*d_ARR); // Set PWM of Positive
differential to Velocity
  }
  else // Positive Velocity Desired (UP)
  {
    // Channel 1
    TIM3->CCR3 = (uint32_t) ((DesiredDutyCycle+0.029f)*d_ARR); // Set PWM of Positive
differential to Velocity
    // Channel 2
    TIM3->CCR4 = (uint32_t) ((0.0f+0.0072f)*d_ARR); // Set PWM of Negative differential
to zero
  }
}
/*************************************************************************/
/*************************************************************************/

void PwmOutInit(void)
{
  uint32_t frequency = 200000;  //default frequency to 200 kHz

  // TIM3 clock Enable
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

  // GPIOC  clock enable
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);

  // GPIO Configuration: TIM3 to CH1(PB4), CH2(PB5)
  GPIO_InitTypeDef GPIO_InitStructure;

  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP ;

  GPIO_Init(GPIOC, &GPIO_InitStructure);

  // Connect TIM3 pin to AF2
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource8, GPIO_AF_TIM3);
  GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_TIM3);
  uint32_t counterClock;

  if(frequency < 1000) {
    counterClock = 100000;
  }
  else {
    counterClock = 28000000;
  }

  uint16_t PrescalerValue = (uint16_t) ((SystemCoreClock /2) / counterClock) - 1;
  d_ARR = counterClock/frequency - 1;

  // Time base configuration
  TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure;

  TIM_TimeBaseStructure.TIM_Period = d_ARR;
  TIM_TimeBaseStructure.TIM_Prescaler = PrescalerValue;
  TIM_TimeBaseStructure.TIM_ClockDivision = 0;
  TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;

  TIM_TimeBaseInit(TIM3, &TIM_TimeBaseStructure);

  // PWM Mode configuration
  TIM_OCInitTypeDef  TIM_OCInitStructure;

  TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
  TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
  TIM_OCInitStructure.TIM_Pulse = 0;
  TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;

  // Channel 3
  TIM_OC3Init(TIM3, &TIM_OCInitStructure);
  TIM_OC3PreloadConfig(TIM3, TIM_OCPreload_Enable);

  // Channel 4
  TIM_OC4Init(TIM3, &TIM_OCInitStructure);
  TIM_OC4PreloadConfig(TIM3, TIM_OCPreload_Enable);

  TIM_ARRPreloadConfig(TIM3, ENABLE);

  // TIM3 enable counter
  TIM_Cmd(TIM3, ENABLE);

}
/************************************************************************/
/************************************************************************/
// Initialization of the system tick interrupt
void sysTickSetup(void)
{
  RCC_ClocksTypeDef RCC_Clocks;

  // Use all bits of interrupt priority to be preemption (16 priority levels)
  SCB->AIRCR = 0x05FA0300;

  /* SysTick configuration */
  RCC_GetClocksFreq(&RCC_Clocks);
  SysTick_Config(RCC_Clocks.HCLK_Frequency / SysTickFrequency);
  SCB->SHP[11]=0x10;     // Make Systick second highest priority interrupt

}
/************************************************************************/
/************************************************************************/
}
```

```
/**********************************************************************/
/**********************************************************************/
// Initialize the GPIO's
void GPIOm_Init(void)
{
          //ELMO Power Switching GPIO - PH9 - GPIOH 9
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9;
  GPIO_Init(GPIOH, &GPIO_InitStructure);
            //Brake Power Switching GPIO - PA - GPIOA 3
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_3;
  GPIO_Init(GPIOA, &GPIO_InitStructure);
          //UP GPIO - PI5 - GPIOI 5
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_5;
  GPIO_Init(GPIOI, &GPIO_InitStructure);
           //DOWN  GPIO - PI6 - GPIOI 6
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
  GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_Init(GPIOI, &GPIO_InitStructure);
}
/**********************************************************************/
/**********************************************************************/
void LoadCellDisplay(void)
{
  if (Buffer_Ok == 1)   //For 2 lines display
    {
      Buffer_Ok = 0;
      for (DisplayCounter = 0; DisplayCounter<12; DisplayCounter++)
      {
        DisplayFirstLine[DisplayCounter] = DisplaySecondLine[DisplayCounter];
      }
      if(Free_Buffer_No == BUFFER_NO1)
      {
        CharToStringToFloatBuffer1();
        for (DisplayCounter = 0; DisplayCounter<12; DisplayCounter++)
        {
          DisplaySecondLine[DisplayCounter] = StringBuffer1[DisplayCounter];
        }
      }
      else
      {
        CharToStringToFloatBuffer2();
        for (DisplayCounter = 0; DisplayCounter<12; DisplayCounter++)
        {
          DisplaySecondLine[DisplayCounter] = StringBuffer2[DisplayCounter];
        }
      }

    }
}
/**********************************************************************/
```

```
/*************************************************************************/
void ModeDisplay(void)
{
  switch (Mode)
    {
    case 1:
      LCD_DisplayStringLine(Line4, (uint8_t *)"    MODE 1 Engaged  ");
      LCD_DisplayStringLine(Line5, (uint8_t *)"Position Locked       ");
      STM_EVAL_LEDOff(LED1);
      STM_EVAL_LEDOff(LED2);
      STM_EVAL_LEDOn(LED3);
      STM_EVAL_LEDOff(LED4);
      break;
    case 2:
      LCD_DisplayStringLine(Line4, (uint8_t *)"    MODE 2 Engaged  ");
      STM_EVAL_LEDOff(LED1);
      STM_EVAL_LEDOn(LED2);
      STM_EVAL_LEDOff(LED3);
      STM_EVAL_LEDOff(LED4);
      break;
    case 3:
      LCD_DisplayStringLine(Line4, (uint8_t *)"    MODE 3 Engaged   ");
      LCD_DisplayStringLine(Line5, (uint8_t *)"Auto Offload ON       ");
      STM_EVAL_LEDOn(LED1);
      STM_EVAL_LEDOff(LED2);
      STM_EVAL_LEDOff(LED3);
      STM_EVAL_LEDOff(LED4);
      break;
    default:
      break;
    }
    //#ifdef IOE_POLLING_MODE
    if (Mode==2)
    {
      Size = sizeof(CmdMotorRMoff);
      MotorCommand_USART3((int)CmdMotorRMoff, Size);
      LCD_DisplayStringLine(Line5, (uint8_t *)"Pos. Adjust       ");

      static JOY_State_TypeDef JoyState = JOY_NONE;
      if(GPIO_ReadInputDataBit(GPIOI, GPIO_Pin_5)==0)
      {
        JoyState = JOY_LEFT;   // Load go UP
      }
      else if(GPIO_ReadInputDataBit(GPIOI, GPIO_Pin_6)==0)
      {
        JoyState = JOY_RIGHT;  // Load go DOWN
      }
      else
      {
        JoyState= JOY_NONE;
      }
      switch (JoyState)
      {
      case JOY_NONE:
        LCD_DisplayStringLine(Line9, (uint8_t *)"                ");
        break;
      case JOY_LEFT:
        LCD_DisplayStringLine(Line9, (uint8_t *)"Motion:  UP    ");
        Size = sizeof(CmdMotorOn);
        MotorCommand_USART3((int)CmdMotorOn, Size);
        GPIOA->BSRRL = GPIO_Pin_3; //Turn on Pin9 -- Power to Brake On
        Size = sizeof(CmdMotorRMoff);
        MotorCommand_USART3((int)CmdMotorRMoff, Size);
        Size = sizeof(CmdMotorUp);
        MotorCommand_USART3((int)CmdMotorUp, Size);
        //SetDutyCycle(0.2);
        while(GPIO_ReadInputDataBit(GPIOI, GPIO_Pin_5)==0 && Mode==2){}
        LCD_DisplayStringLine(Line9, (uint8_t *)"               ");
        //SetDutyCycle(0.0);
        Size = sizeof(CmdMotorHold);
        MotorCommand_USART3((int)CmdMotorHold, Size);
        //InitLoad=ForceValue;
        //GPIOA->BSRRH = GPIO_Pin_3;  //Turn off Pin9 -- Power to Brake Off
        break;
      case JOY_RIGHT:
        LCD_DisplayStringLine(Line9, (uint8_t *)"Motion:  DOWN      ");
        Size = sizeof(CmdMotorOn);
        MotorCommand_USART3((int)CmdMotorOn, Size);
        Size = sizeof(CmdMotorRMoff);
        MotorCommand_USART3((int)CmdMotorRMoff, Size);
        GPIOA->BSRRL = GPIO_Pin_3; //Turn on Pin9 -- Power to Brake On
        Size = sizeof(CmdMotorDown);
        MotorCommand_USART3((int)CmdMotorDown, Size);
        //SetDutyCycle(-0.2);
        while(GPIO_ReadInputDataBit(GPIOI, GPIO_Pin_6)==0 && Mode==2){}
        LCD_DisplayStringLine(Line9, (uint8_t *)"               ");
        //SetDutyCycle(0.0);
        Size = sizeof(CmdMotorHold);
        MotorCommand_USART3((int)CmdMotorHold, Size);
        //GPIOA->BSRRH = GPIO_Pin_3;  //Turn off Pin9 -- Power to Brake Off
        // InitLoad=ForceValue;
        break;
      default:
        LCD_DisplayStringLine(Line8, (uint8_t *)"Joystick:  ERROR    ");
        break;
      }
    }
    //#endif /* IOE_POLLING_MODE */
}
/*************************************************************************/
/*************************************************************************/
void MotorVelCommand(int CmdAction, int Size)
{
  DMA_DeInit(USARTy_TX_DMA_STREAM);
  DMA_InitStructurey.DMA_Channel = USARTy_TX_DMA_CHANNEL;
  DMA_InitStructurey.DMA_DIR = DMA_DIR_MemoryToPeripheral;
  DMA_InitStructurey.DMA_Memory0BaseAddr = (uint32_t)CmdAction; //Must contain \n\r
  DMA_InitStructurey.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructurey.DMA_BufferSize = (uint16_t)Size;
  DMA_InitStructurey.DMA_Mode = DMA_Mode_Normal;
  DMA_Init(USARTy_TX_DMA_STREAM, &DMA_InitStructurey);

  /* Enable the USART DMA requests */
  USART_DMACmd(USARTy, USART_DMAReq_Tx, ENABLE);

  /* Clear the TC bit in the SR register by writing 0 to it */
 USART_ClearFlag(USARTy, USART_FLAG_TC);

  /* Enable the DMA TX Stream, i.e. USART will start sending the command code (2bytes)
*/
  DMA_Cmd(USARTy_TX_DMA_STREAM, ENABLE);

  /* Wait the USART DMA Tx transfer complete or time out */
  TimeOut = USER_TIMEOUT;
      }
  //Clear all DMA Streams flags
  //DMA_ClearFlag(USARTy_TX_DMA_STREAM, USARTy_TX_DMA_FLAG_FEIF |
USARTy_TX_DMA_FLAG_DMEIF |
  //                                        USARTy_TX_DMA_FLAG_TEIF |
USARTy_TX_DMA_FLAG_HTIF |
  //                                        USARTy_TX_DMA_FLAG_TCIF);

}
/*************************************************************************/
/*************************************************************************/
void MotorCommand_USART3(int CmdAction, int Size)
{
  DMA_DeInit(USARTy_TX_DMA_STREAM);
  DMA_InitStructurey.DMA_Channel = USARTy_TX_DMA_CHANNEL;
  DMA_InitStructurey.DMA_DIR = DMA_DIR_MemoryToPeripheral;
  DMA_InitStructurey.DMA_Memory0BaseAddr = (uint32_t)CmdAction; //Must contain \n\r
  DMA_InitStructurey.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructurey.DMA_BufferSize = (uint16_t)Size;
  DMA_InitStructurey.DMA_Mode = DMA_Mode_Normal;
  DMA_Init(USARTy_TX_DMA_STREAM, &DMA_InitStructurey);

  /* Enable the USART DMA requests */
  USART_DMACmd(USARTy, USART_DMAReq_Tx, ENABLE);

  /* Clear the TC bit in the SR register by writing 0 to it */
 USART_ClearFlag(USARTy, USART_FLAG_TC);

  /* Enable the DMA TX Stream, i.e. USART will start sending the command code (2bytes)
*/
  DMA_Cmd(USARTy_TX_DMA_STREAM, ENABLE);

  /* Wait the USART DMA Tx transfer complete or time out */
  TimeOut = USER_TIMEOUT;

  while ((DMA_GetFlagStatus(USARTy_TX_DMA_STREAM, USARTy_TX_DMA_FLAG_TCIF) ==
RESET)&&(TimeOut != 0x00))
  {
  }

  if(TimeOut == 0)
  {
    TimeOut_UserCallback();
  }
  // The software must wait until TC=1. The TC flag remains cleared during all data
  //transfers and it is set by hardware at the last frame's end of transmission
  TimeOut = USER_TIMEOUT;
  while ((USART_GetFlagStatus(USARTy, USART_FLAG_TC) == RESET)&&(TimeOut != 0x00))
  {
  }
  if(TimeOut == 0)
  {
    TimeOut_UserCallback();
  }

  //Clear all DMA Streams flags
  DMA_ClearFlag(USARTy_TX_DMA_STREAM, USARTy_TX_DMA_FLAG_FEIF | USARTy_TX_DMA_FLAG_DMEIF
|
                                          USARTy_TX_DMA_FLAG_TEIF |
USARTy_TX_DMA_FLAG_HTIF |
                                          USARTy_TX_DMA_FLAG_TCIF);

  /* Disable the TX DMA Streams */
  DMA_Cmd(USARTy_TX_DMA_STREAM, DISABLE);
  /* Disable the USART Tx DMA request */
  USART_DMACmd(USARTy, USART_DMAReq_Tx, DISABLE);

  //TimeOut = USER_TIMEOUT;     //Just for Debugging
}
/************************************************************************/
/************************************************************************/
void USART_InitLoadCell(void)
{
/******************************************************************************/
/*                    USART Transmit The 'TARE\n\r' Command                         */
/******************************************************************************/
  DMA_DeInit(USARTx_TX_DMA_STREAM);
  DMA_InitStructurex.DMA_Channel = USARTx_TX_DMA_CHANNEL;
  DMA_InitStructurex.DMA_DIR = DMA_DIR_MemoryToPeripheral;
  DMA_InitStructurex.DMA_Memory0BaseAddr = (uint32_t)CmdBufferTare; //Must contain \n\r
  DMA_InitStructurex.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructurex.DMA_BufferSize = (uint16_t)6;
  DMA_InitStructurex.DMA_Mode = DMA_Mode_Normal;
  DMA_Init(USARTx_TX_DMA_STREAM, &DMA_InitStructurex);

  /* Enable the USART DMA requests */
  USART_DMACmd(USARTx, USART_DMAReq_Tx, ENABLE);

  /* Clear the TC bit in the SR register by writing 0 to it */
 USART_ClearFlag(USARTx, USART_FLAG_TC);

  /* Enable the DMA TX Stream, i.e. USART will start sending the command code (2bytes)
*/
  DMA_Cmd(USARTx_TX_DMA_STREAM, ENABLE);

  /* Wait the USART DMA Tx transfer complete or time out */
  TimeOut = USER_TIMEOUT;

  while ((DMA_GetFlagStatus(USARTx_TX_DMA_STREAM, USARTx_TX_DMA_FLAG_TCIF) ==
RESET)&&(TimeOut != 0x00))
  {
  }

  if(TimeOut == 0)
  {
    TimeOut_UserCallback();
  }
  /* The software must wait until TC=1. The TC flag remains cleared during all data
  transfers and it is set by hardware at the last frame's end of transmission*/
  TimeOut = USER_TIMEOUT;
  while ((USART_GetFlagStatus(USARTx, USART_FLAG_TC) == RESET)&&(TimeOut != 0x00))
  {
  }
  if(TimeOut == 0)
  {
    TimeOut_UserCallback();
  }

  /* Clear all DMA Streams flags */
  DMA_ClearFlag(USARTx_TX_DMA_STREAM, USARTx_TX_DMA_FLAG_FEIF | USARTx_TX_DMA_FLAG_DMEIF
|
                                          USARTx_TX_DMA_FLAG_TEIF |
USARTx_TX_DMA_FLAG_HTIF |
                                          USARTx_TX_DMA_FLAG_TCIF);

  /* Disable the TX DMA Streams */
  DMA_Cmd(USARTx_TX_DMA_STREAM, DISABLE);
  /* Disable the USART Tx DMA request */
  USART_DMACmd(USARTx, USART_DMAReq_Tx, DISABLE);

  TimeOut = USER_TIMEOUT;     //Just for Debugging
  Delay(10000000);
/*******************************************************************************/
/*                    USART Transmit The 'WC\n\r' Command                         */
/*******************************************************************************/
  DMA_DeInit(USARTx_TX_DMA_STREAM);
  DMA_InitStructurex.DMA_Channel = USARTx_TX_DMA_CHANNEL;
  DMA_InitStructurex.DMA_DIR = DMA_DIR_MemoryToPeripheral;
  DMA_InitStructurex.DMA_Memory0BaseAddr = (uint32_t)CmdBufferTransmit; //Must contain
\n\r
  DMA_InitStructurex.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructurex.DMA_BufferSize = (uint16_t)4;
  DMA_InitStructurex.DMA_Mode = DMA_Mode_Normal;
  DMA_Init(USARTx_TX_DMA_STREAM, &DMA_InitStructurex);

  /* Enable the USART DMA requests */
  USART_DMACmd(USARTx, USART_DMAReq_Tx, ENABLE);

  /* Clear the TC bit in the SR register by writing 0 to it */
 USART_ClearFlag(USARTx, USART_FLAG_TC);

  /* Enable the DMA TX Stream, i.e. USART will start sending the command code (2bytes)
*/
  DMA_Cmd(USARTx_TX_DMA_STREAM, ENABLE);

  /* Wait the USART DMA Tx transfer complete or time out */
  TimeOut = USER_TIMEOUT;

  while ((DMA_GetFlagStatus(USARTx_TX_DMA_STREAM, USARTx_TX_DMA_FLAG_TCIF) ==
RESET)&&(TimeOut != 0x00))
  {
  }

  if(TimeOut == 0)
  {
    TimeOut_UserCallback();
  }
  /* The software must wait until TC=1. The TC flag remains cleared during all data
  transfers and it is set by hardware at the last frame's end of transmission*/
  TimeOut = USER_TIMEOUT;
  while ((USART_GetFlagStatus(USARTx, USART_FLAG_TC) == RESET)&&(TimeOut != 0x00))
  {
  }
  if(TimeOut == 0)
  {
    TimeOut_UserCallback();
  }

  /* Clear all DMA Streams flags */
  DMA_ClearFlag(USARTx_TX_DMA_STREAM, USARTx_TX_DMA_FLAG_FEIF | USARTx_TX_DMA_FLAG_DMEIF
|
                                          USARTx_TX_DMA_FLAG_TEIF |
USARTx_TX_DMA_FLAG_HTIF |
                                          USARTx_TX_DMA_FLAG_TCIF);

  /* Disable the TX DMA Streams */
  DMA_Cmd(USARTx_TX_DMA_STREAM, DISABLE);
  /* Disable the USART Tx DMA request */
  USART_DMACmd(USARTx, USART_DMAReq_Tx, DISABLE);
  TimeOut = USER_TIMEOUT;     //Just for Debugging

  //Send the No Echo command to the Elmo Motor controller
  CmdMotorEcho   [4]    = 13;
  Size = sizeof(CmdMotorEcho);
  MotorCommand_USART3((int)CmdMotorEcho, Size);


/******************************************************************************/
/*                     USART Begin to Receiver Data                           */
/******************************************************************************/
  /* At this step the USART will receive the the transaction data*/
  DMA_DeInit(USARTx_RX_DMA_STREAM);
  DMA_InitStructurex.DMA_Channel = USARTx_RX_DMA_CHANNEL;
  DMA_InitStructurex.DMA_DIR = DMA_DIR_PeripheralToMemory;
  DMA_InitStructurex.DMA_Memory0BaseAddr = (uint32_t)USART_DMA_Buffer1;
  DMA_InitStructurex.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructurex.DMA_BufferSize = 14; //2 double the bytes from load cell
  DMA_InitStructurex.DMA_Mode = DMA_Mode_Circular;
  DMA_Init(USARTx_RX_DMA_STREAM, &DMA_InitStructurex);

  DMA_DeInit(USARTy_RX_DMA_STREAM);
  DMA_InitStructurey.DMA_Channel = USARTy_RX_DMA_CHANNEL;
  DMA_InitStructurey.DMA_DIR = DMA_DIR_PeripheralToMemory;
  DMA_InitStructurey.DMA_Memory0BaseAddr = (uint32_t)USARTy_DMA_Buffer1;
  DMA_InitStructurey.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructurey.DMA_BufferSize = 14; //2 double the bytes from load cell
  DMA_InitStructurey.DMA_Mode = DMA_Mode_Circular;
  DMA_Init(USARTy_RX_DMA_STREAM, &DMA_InitStructurey);

  NVIC_Config();


  Free_Buffer_No = BUFFER_NO2;
  Buffer_Ok = 0;  //No data ready

  USART_DMACmd(USARTx, USART_DMAReq_Rx , ENABLE);
  USART_DMACmd(USARTy, USART_DMAReq_Rx , ENABLE);

  /* Enable the DMA Stream */
  DMA_Cmd(USARTx_RX_DMA_STREAM, ENABLE);
  DMA_Cmd(USARTy_RX_DMA_STREAM, ENABLE);

  /* SysTick end of count event each 10ms */
  RCC_GetClocksFreq(&RCC_Clocks);
  SysTick_Config(RCC_Clocks.HCLK_Frequency / 100);
}
/************************************************************************/
/************************************************************************/
static void TimeOut_UserCallback(void)
{
  while (1)     //Modify by LSW. Add your codes here
  {
    //sprintf((char*) Message,"   Error = %s  ", "TimeOut");
    //LCD_DisplayStringLine(LINE(6), Message);
  }
}
/************************************************************************/
/************************************************************************/
static void USART_Config(void)
{
  USART_InitTypeDef USART_InitStructure;
  GPIO_InitTypeDef GPIO_InitStructure;
```

```
/* Enable GPIO clock */
RCC_AHB1PeriphClockCmd(USARTx_TX_GPIO_CLK | USARTx_RX_GPIO_CLK, ENABLE);
RCC_AHB1PeriphClockCmd(USARTy_TX_GPIO_CLK | USARTy_RX_GPIO_CLK, ENABLE);

/* Enable USART clock */
USARTx_CLK_INIT(USARTx_CLK, ENABLE);
USARTy_CLK_INIT(USARTy_CLK, ENABLE);

/* Connect USART pins to AF7 */
GPIO_PinAFConfig(USARTx_TX_GPIO_PORT, USARTx_TX_SOURCE, USARTx_TX_AF);
GPIO_PinAFConfig(USARTx_RX_GPIO_PORT, USARTx_RX_SOURCE, USARTx_RX_AF);
GPIO_PinAFConfig(USARTy_TX_GPIO_PORT, USARTy_TX_SOURCE, USARTy_TX_AF);
GPIO_PinAFConfig(USARTy_RX_GPIO_PORT, USARTy_RX_SOURCE, USARTy_RX_AF);

/* Configure USART Tx and Rx as alternate function push-pull */
//Loadcell
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;
//Motor Drive
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP;

//Loadcell
GPIO_InitStructure.GPIO_Pin = USARTx_TX_PIN;
GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStructure);
//Motor Drive
GPIO_InitStructure.GPIO_Pin = USARTy_TX_PIN;
GPIO_Init(USARTy_TX_GPIO_PORT, &GPIO_InitStructure);

//Loadcell
GPIO_InitStructure.GPIO_Pin = USARTx_RX_PIN;
GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStructure);
//Motor Drive
GPIO_InitStructure.GPIO_Pin = USARTy_RX_PIN;
GPIO_Init(USARTy_RX_GPIO_PORT, &GPIO_InitStructure);

/* Enable the USART OverSampling by 8 */
USART_OverSampling8Cmd(USARTx, ENABLE);
USART_OverSampling8Cmd(USARTy, ENABLE);

//Loadcell
USART_InitStructure.USART_BaudRate = 230400;  //3750000
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
/* When using Parity the word length must be configured to 9 bits */
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
USART_Init(USARTx, &USART_InitStructure);

//Motor Controller
USART_InitStructure.USART_BaudRate = 57600;   //230400;  //3750000
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
USART_InitStructure.USART_StopBits = USART_StopBits_1;
/* When using Parity the word length must be configured to 9 bits */
USART_InitStructure.USART_Parity = USART_Parity_No;
USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
```

```
  USART_Init(USARTy, &USART_InitStructure);

  /* Configure DMA controller to manage USART TX and RX DMA request ----------*/
  /* Enable the DMA clock */
  RCC_AHB1PeriphClockCmd(USARTx_DMAx_CLK, ENABLE);

  DMA_InitStructurex.DMA_PeripheralBaseAddr = USARTx_DR_ADDRESS;
  DMA_InitStructurex.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructurex.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
  DMA_InitStructurex.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
  DMA_InitStructurex.DMA_Priority = DMA_Priority_VeryHigh;
  DMA_InitStructurex.DMA_FIFOMode = DMA_FIFOMode_Disable;//Modified
  DMA_InitStructurex.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
  DMA_InitStructurex.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructurex.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
  /* Here only the unchanged parameters of the DMA initialization structure are
     configured. During the program operation, the DMA will be configured with
     different parameters according to the operation phase */

  /* Enable USART1 */
  USART_Cmd(USARTx, ENABLE);

  /* Configure DMA controller to manage USART TX and RX DMA request ----------*/
  /* Enable the DMA clock */
  RCC_AHB1PeriphClockCmd(USARTy_DMAy_CLK, ENABLE);

  DMA_InitStructurey.DMA_PeripheralBaseAddr = USARTy_DR_ADDRESS;
  DMA_InitStructurey.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructurey.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
  DMA_InitStructurey.DMA_MemoryDataSize = DMA_MemoryDataSize_Byte;
  DMA_InitStructurey.DMA_Priority = DMA_Priority_VeryHigh;
  DMA_InitStructurey.DMA_FIFOMode = DMA_FIFOMode_Disable;//Modified
  DMA_InitStructurey.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
  DMA_InitStructurey.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructurey.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
  /* Here only the unchanged parameters of the DMA initialization structure are
     configured. During the program operation, the DMA will be configured with
     different parameters according to the operation phase */

  /* Enable USART3 */
  USART_Cmd(USARTy, ENABLE);
}
/**************************************************************************/
/**************************************************************************/
void NVIC_Config(void)
{
  NVIC_InitTypeDef NVIC_InitStructure;
  /* Configure the Priority Group to 1 bit */
  NVIC_PriorityGroupConfig(NVIC_PriorityGroup_4);

  NVIC_InitStructure.NVIC_IRQChannel = DMAn_Streamn_IRQn;//Pay attention
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

    /* Enable DMA2_Stream5 Transfer complete interrupt */
  DMA_ITConfig(DMA2_Stream5,DMA_IT_TC,ENABLE);//Pay attention

  //For DMA1
  //  NVIC_InitTypeDef NVIC_InitStructure2;
  /* Configure the Priority Group to 1 bit */
  //NVIC_PriorityGroupConfig(NVIC_PriorityGroup_2);
  NVIC_InitStructure.NVIC_IRQChannel = DMAm_Streamm_IRQm;//Pay attention
  NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x01;
  NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
  NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
  NVIC_Init(&NVIC_InitStructure);

    /* Enable DMA2_Stream5 Transfer complete interrupt */
  DMA_ITConfig(DMA1_Stream1,DMA_IT_TC,ENABLE);//Pay attention
}
/*************************************************************************/
/*************************************************************************/
void Display_Init(void)/* Display By LSW */
{
   /* Initialize the LCD */
   STM324xG_LCD_Init();

   /* Clear the LCD */
   LCD_Clear(White);

   /* Set the LCD Text size */
   LCD_SetFont(&Font8x12);

   /* Set the LCD Back Color and Text Color*/
   LCD_SetBackColor(Blue);
   LCD_SetTextColor(White);

   /* Display */
   //LCD_DisplayStringLine(LINE(0x13), "  USART Communication DMA Test Program  ");

   /* Set the LCD Text size */
   LCD_SetFont(&Font16x24);

   /* Display */
   //LCD_DisplayStringLine(LINE(0), "   USART DMA Test   ");


   /* Set the LCD Back Color and Text Color*/
   LCD_SetBackColor(White);
   LCD_SetTextColor(Blue);

   /* Display */
 //  LCD_DisplayStringLine(LINE(2),"       AVSLAB    ");

   LCD_SetFont(&Font12x12); //Set fonts to have enough place on the LCD;
     /* Initialize LEDs and push-buttons mounted on STM322xG-EVAL board */
  STM_EVAL_LEDInit(LED1);
  STM_EVAL_LEDInit(LED2);
  STM_EVAL_LEDInit(LED3);
  STM_EVAL_LEDInit(LED4);

  /* Select the Button test mode (polling or interrupt) BUTTON_MODE in main.h */
  STM_EVAL_PBInit(BUTTON_WAKEUP, BUTTON_MODE);
  STM_EVAL_PBInit(BUTTON_TAMPER, BUTTON_MODE);
  STM_EVAL_PBInit(BUTTON_KEY, BUTTON_MODE);

  /* Initialize the LCD */
  STM324xG_LCD_Init();

  /* Clear the LCD */
  LCD_Clear(White);
  /* Set the LCD Back Color */
  LCD_SetBackColor(White);
  /* Set the LCD Text Color */
  LCD_SetTextColor(Red);

  LCD_DisplayStringLine(Line0, (uint8_t *)"MoRGS MODE SELECTION");
  LCD_DisplayStringLine(Line1, (uint8_t *)"Mode 1-Locked Pos.");
  LCD_DisplayStringLine(Line2, (uint8_t *)"Mode 2-Adjust Pos.");
  LCD_DisplayStringLine(Line3, (uint8_t *)"Mode 3-Auto Offload");

  /* Configure the IO Expander */
  if (IOE_Config() == IOE_OK)
  {
    //LCD_DisplayStringLine(Line4, (uint8_t *)"   IO Expander OK   ");
  }
  else
  {
    LCD_DisplayStringLine(Line4, (uint8_t *)"IO Expander FAILED ");
    LCD_DisplayStringLine(Line5, (uint8_t *)" Please Reset the  ");
    LCD_DisplayStringLine(Line6, (uint8_t *)"   board and start ");
    LCD_DisplayStringLine(Line7, (uint8_t *)"    again          ");
    while(1);
  }
}
/*************************************************************************/
/*************************************************************************/
/*The function will first convert the bytes to a string 'Message', then convert
'Message' to float
 *Return: ForceValue*/
void CharToStringToFloatBuffer1(void)
{
  sprintf((char*)StringBuffer1,"%c%c%c%c%c%c%c%c%c%c%c%c",
USART_DMA_Buffer1[0],USART_DMA_Buffer1[1],USART_DMA_Buffer1[2],USART_DMA_Buffer1[3],USAR
T_DMA_Buffer1[4],USART_DMA_Buffer1[5],USART_DMA_Buffer1[6],USART_DMA_Buffer1[7],USART_DM
A_Buffer1[8],USART_DMA_Buffer1[9],USART_DMA_Buffer1[10],USART_DMA_Buffer1[11]);
  sprintf((char*)ByteToStringBuffer1,"%c%c%c%c%c%c%c%c%c%c%c%c",
USART_DMA_Buffer1[0],USART_DMA_Buffer1[1],USART_DMA_Buffer1[2],USART_DMA_Buffer1[3],USAR
T_DMA_Buffer1[4],USART_DMA_Buffer1[5],USART_DMA_Buffer1[6],USART_DMA_Buffer1[7],USART_DM
A_Buffer1[8],USART_DMA_Buffer1[9],USART_DMA_Buffer1[10],USART_DMA_Buffer1[11]);
  //ForceValue = atof( ByteToStringBuffer1 );
}
void CharToStringToFloatBuffer2(void)
{
  sprintf((char*)StringBuffer2,"%c%c%c%c%c%c%c%c%c%c%c%c",
USART_DMA_Buffer2[0],USART_DMA_Buffer2[1],USART_DMA_Buffer2[2],USART_DMA_Buffer2[3],USAR
T_DMA_Buffer2[4],USART_DMA_Buffer2[5],USART_DMA_Buffer2[6],USART_DMA_Buffer2[7],USART_DM
A_Buffer2[8],USART_DMA_Buffer2[9],USART_DMA_Buffer2[10],USART_DMA_Buffer2[11]);
  sprintf((char*)ByteToStringBuffer2,"%c%c%c%c%c%c%c%c%c%c%c%c",
USART_DMA_Buffer2[0],USART_DMA_Buffer2[1],USART_DMA_Buffer2[2],USART_DMA_Buffer2[3],USAR
T_DMA_Buffer2[4],USART_DMA_Buffer2[5],USART_DMA_Buffer2[6],USART_DMA_Buffer2[7],USART_DM
A_Buffer2[8],USART_DMA_Buffer2[9],USART_DMA_Buffer2[10],USART_DMA_Buffer2[11]);
  //ForceValue = atof( ByteToStringBuffer2 );
}

#ifdef  USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
  ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  /* Infinite loop */
  while (1)
  {}
}
#endif
```

```
/************************************************************************/
/************************************************************************/
void ADC3_CH4_DMA_Config(void)
{
  ADC_InitTypeDef       ADC_InitStructure;
  ADC_CommonInitTypeDef ADC_CommonInitStructure;
  DMA_InitTypeDef       DMA_InitStructure;
  GPIO_InitTypeDef      GPIO_InitStructure;

  /* Enable ADC3, DMA2 and GPIO clocks ****************************************/
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2 | RCC_AHB1Periph_GPIOA, ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

  /* DMA2 Stream0 channel2 configuration **************************************/
  DMA_InitStructure.DMA_Channel = DMA_Channel_0;
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)ADC1_DR_ADDRESS;
  DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)ADC3ConvertedValue;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
  DMA_InitStructure.DMA_BufferSize = 10;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;//Disable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
  DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
  DMA_Init(DMA2_Stream0, &DMA_InitStructure);
  DMA_Cmd(DMA2_Stream0, ENABLE);
//PA 6  In6
  /* Configure ADC3 Channel7 pin as analog input ******************************/

  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

  /* ADC Common Init **********************************************************/
  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
  ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);

  /* ADC3 Init ****************************************************************/
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = DISABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_None;
  ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_Ext_IT11;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = 1;
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC3 regular channel7 configuration *************************************/
  ADC_RegularChannelConfig(ADC1, ADC_Channel_6, 1, ADC_SampleTime_84Cycles);

 /* Enable DMA request after last transfer (Single-ADC mode) */
  ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);

  /* Enable ADC3 DMA */
  ADC_DMACmd(ADC1, ENABLE);

  /* Enable ADC3 */
  ADC_Cmd(ADC1, ENABLE);
}
/**
  * @brief  Inserts a delay time.
  * @param  nTime: specifies the delay time length, in 10 ms.
  * @retval None
  */
/*************************************************************************/
/*************************************************************************/
void Delay(uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0)
  {
    TimingDelay_Decrement();
  }
}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
/*************************************************************************/
/*************************************************************************/
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }
}

#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
/*************************************************************************/
/*************************************************************************/
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  LCD_DisplayStringLine(Line0, "assert_param error!!");

  /* Infinite loop */
  while (1)
  {
  }
}
#endif
/**
  * @brief  Basic management of the timeout situation.
  * @param  None.
  * @retval None.
  */
```