University of Pennsylvania
Center for Sensor Technologies

SUNFEST

NSF REU Program
Summer 2004

EXPERIMENTAL DEVELOPMENT OF THE MOBILE VESTIBULAR PLATFORM

NSF Summer Undergraduate Fellowship in Sensor Technologies
Alexander H. Chang (Comp. and Telecom. Engineering) – University of Pennsylvania
Microsoft Sunfest Fellow
Advisor: Dr. Daniel D. Lee

ABSTRACT

Cable robotics is an emerging research field in robotics which has the potential to be applied to a variety of practical purposes and tasks. Applications dealing with short distance transportation of hazardous materials as well as the handling of and interaction around these materials are several of the major tasks that can be handled efficiently and safely by cable robots, reducing the risk to employee lives as well as the need for more complex methods to accomplish the same task. Environmental monitoring of deep mines, hostile environments, and other locations inaccessible or uninhabitable by humans is another very realistic and viable application of cable robotics, allowing for the current sensory information about a remote location to be known at any time. Because of the simplistic nature of the design of cable robots, entire networks can also be implemented in order to cooperate with one another with the purpose of completing a particular task and reacting to sudden changes in sensory data extracted from the surrounding environment.

The particular robot being developed in this project is a versatile, cable driven robotic platform capable of movement in a three-dimensional space, and is named the mobile vestibular platform (MVP). It is designed to be versatile in that the system can easily accommodate several different sensory functions, while also being low budget and light weight. The particular implementation used allows for high precision motor control and thus higher precision in the movement of the robotic platform through space, while a Matlab program handles calculation tasks and remote operation of the platform from a central processing PC.

Table of Contents

SECTION 1: INTRODUCTION **pg. 3**
SECTION 2: A General Overview of the Cable Robot System **pg. 6**
2.1 Command Terminal Overview **pg. 6**
2.2 Mobile Platform (End-effector) Overview **pg. 7**
SECTION 3: Mech. Design of the MVP System **pg. 7**
3.1 Suspension Cables **pg. 7**
3.2 Cable Mount Locations **pg. 9**
3.3 Platform Design **pg. 11**
SECTION 4: Elec. Design of the MVP System **pg. 12**
4.1 Components **pg. 12**
4.2 PCB Design **pg. 16**
4.3 Flowchart of the Electrical System **pg. 16**
4.4 Problems/Issues **pg. 17**
4.5 Completed MVP End-Effector **pg. 19**
SECTION 5: Soft. Design of the MVP System **pg. 19**
5.1 Mobile Platform End-Effector System Software **pg. 20**
5.2 Command Terminal Software **pg. 20**
SECTION 6: DISCUSSIONS AND CONCLUSIONS **pg. 22**
6.1 Foreseeable System Issues **pg. 22**
6.2 Future Work **pg. 22**
SECTION 7: RECOMMENDATIONS **pg. 24**
SECTION 8: ACKNOWLEDGEMENTS **pg. 24**
SECTON 9: REFERENCES **pg. 25**
APPENDIX A: MVP - End-Effector System Software **pg. 26**
APPENDIX B: MVP - Command Terminal System Software **pg. 37**

1. INTRODUCTION

The field of research in cable robotics has been gathering increasing attention due to the growing number of applications to which this field can be applied. Cable robots present several advantages in different applications due to their fundamental methods of operation, making them more suitable than other types of robots for several applications. This emerging field deals with the development of a form of robotics in which a central platform, or end-effector, is suspended by several cables, each of which is anchored to a stationary point in the surrounding environment. [3] The end-effector is so-named because it is the component performing the primary task of the robotic system and therefore is the only module of the system supporting the equipment needed for the particular primary task. It is moved from one point to another in three dimensional space by changing the lengths of the cable by which it is supported. This particular fundamental operation of the system is performed by actuators, or motors, that rotate the spools to which the suspension cables are attached. [2] The mobile vestibular platform (MVP) that was developed in this project is a particular example of a cable robot design. Its physical and mechanical aspects are shown in figure 1.

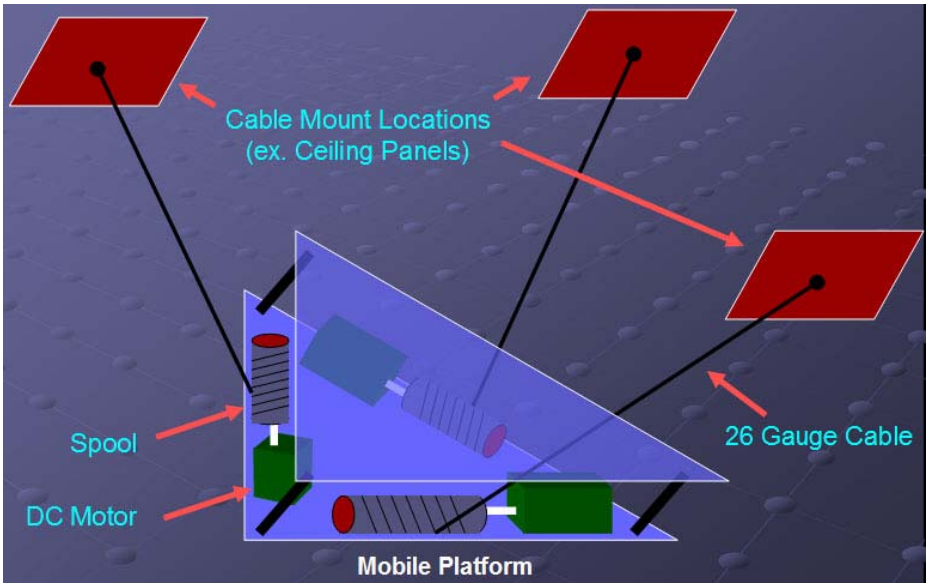

Fig. 1: Diagram of mechanical aspects of mobile vestibular platform end-effector

Several key properties of cable robots allow them significantly greater capabilities than other types of robotics. Because of their suspension by cables to stable points in the surrounding environment, cable robots have the capability to move in three dimensions, thus allowing them to move in efficient trajectories that other robots, constrained to movement in a plane, cannot employ. In this way, complex terrain and ground obstacles do not play a significant factor in the problem of moving from any point A to another point B, especially since obstructions and obstacles in the volume above a ground surface are much less common that those that may occur on the ground surface. In addition, the capability of movement in a third dimension allows for a much larger volume in which the robotic system may perform its tasks or work. Thus, this space or volume, constituting all of the points in the three-dimensional space to which the end-effector may travel, is defined as the *workspace* [4]. In addition to this, the robotic system can be

designed to be modular as well as extremely versatile, in many ways. Modularity allows for easy and quick repair and maintenance of the robotic system. The versatility of the cable robot is evident in designs where the end-effector is the only component performing the primary function within the robot's environment. Different end-effectors may be installed, or the equipment on the end-effectors changed, to allow for different tasks to be performed, such as sensory data collection, object transportation, etc. Also, cable robot design need not be tailored to the environment at hand, since the suspension cable mounts may be installed to any stationary location, after which initial setup and calibration of the system will "customize" the system to its particular environment.

This type of robotic system is appropriate for situations in which simple tasks or routines must be performed with precision in environments that are not particularly suitable or safe for human interaction. For instance, routine measurements of environmental properties, such as gaseous air content or temperature, in deep mines is one application to which cable robots are well-suited. Another realistic and practical application of cable robots is the manipulation and short distance transportation of hazardous materials, whether in a disposal or research environment, which would requires a system that is extremely robust as well as reliable and precise. Another reason for the increasing interest in cable robotics is its suitability to future applications and problems. For instance, one possible future use of the cable robot would be in the monitoring and surveillance of the exterior region of a space shuttle or station, as well as the execution of simple operations such as object retrieval and data collection.

Several models of cable robots are currently in existence and are used in industries for which they were designed. One well-known example is the SkyCam (figure 2, left), which is a camera suspend by three very thin, but extremely robust, Kevlar-braided cables. The .25 mm diameter and reliability of the Kevlar-braided suspension cables allow the end-effector, the camera, to be transported at speeds up to 28 mph [1]. Due to the fact that this system transports a known, constant load, as opposed to a varying one such as in the application of waste disposal, its capabilities are easily determined and relatively stable. This property of SkyCam, however, is not always the case, as is evident in another well-known cable robot model, RoboCrane (figure 2, right), which is an industrial cable robot developed by the National Institute of Standards and Technology (NIST). Due to its extreme robustness, it can perform tasks dealing with variably heavy loads, such as warehouse item management and aircraft maintenance. [1]

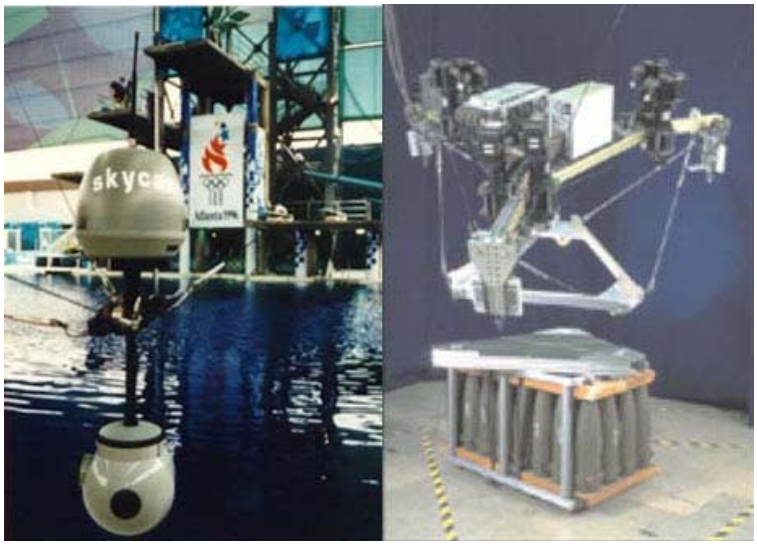

Fig. 2: SkyCam (left) and NIST RoboCrane (right)

The cable robot developed in this project is geared largely toward providing sensor mobility in a variety of environments, independent of the local or surrounding terrain, with the exception of the necessary stable cable mount locations. It was envisioned to be both light weight, to allow for operation in a wider gamut of environments, and low cost, so that entire sensory networks can be developed with these mobile platforms as sensory nodes for a variety of different sensory data and equipment. The intended purpose of the mobile vestibular platform necessitates the incorporation of several essential properties in order for the system to be practical and widely used. Versatility is one important aspect; the system should accommodate several different sensory functions at once and also be readily and quickly modified to support other functions and equipment. Another property needed for this particular system is precision mobility. The mobile platform is the means by which a general variety of light weight sensory equipment will be made mobile. As a result, the system's operations will be required to be precise enough in its movements so that that functions such as locating sources of sensory data (for example sound, light, chemicals sources, etc.) can be accomplished efficiently. This was done by incorporating precision motor controller IC's, which maintain precise tracking of motor position and movement related quantities, into the electrical system of the platform. In addition to the mobile platform and its own software/hardware control system, another remote terminal control system was implemented in a Matlab GUI application, allowing users to remotely control most higher-level functions of the platform as well as some basic functions for system debugging purposes.

A general overview describing the hardware and software systems that are implemented on the mobile vestibular platform system is first presented, in section 2. Section 3 provides a brief description of the mechanical system of the mobile platform end-effector, in order to give the reader an idea of the physical aspects of the system. Also, a description of the electrical system is presented in detail in section 4, summarizing the function of each component as well as revealing some of the problems encountered in the overall electrical system. In addition, several of the approaches taken to attempt to solve the different obstacles encountered are discussed. Finally, the programming/software aspect of this system is illustrated in two parts, one for the

microcontroller program controlling the end-effector, and one for the Matlab GUI program controlling the command terminal, both in section 5. The justification for the functions implemented in each of these two modules of code is also provided. In the end, some apparent issues of the mobile vestibular platform system as well as recommended solutions and future work are discussed.

2. A GENERAL OVERVIEW OF THE CABLE ROBOT SYSTEM

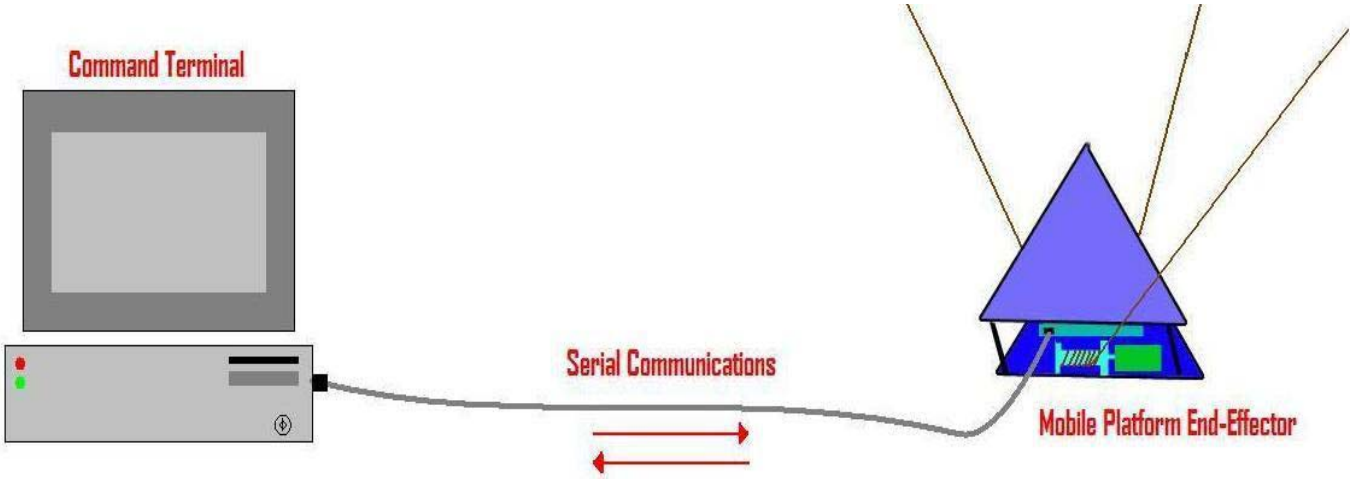

Fig. 3: Image of serial connection between command terminal and moving mobile platform

The MVP system being developed in this program is essentially a low-cost cable robot that is intended for use in tasks involving sensory data collection. The design was such that the cable robot system could be applied to a variety of environments, given the existence of the necessary number of stable cable mount locations. As a result, the environment in which this system could be used to collect sensory data did not play a large role as a variable in determining the design of the robot. In addition, the type of sensory equipment intended for use on this particular robot was limited to a relatively light weight, which also influenced both the mechanical and electrical design. Together, all of constraints as well as the intended purpose of the system led to a unique design compared to existing cable robots. The particular cable robot system that was developed in this program consists primarily of two physical entities. One was the mobile platform itself, serving as the end-effector of the system, and the other, less obvious, component was the PC running the Matlab GUI application from which the user can issue commands to the platform. The PC is referred to as the command terminal of the cable robot system. Both of these components of the mobile vestibular platform system are shown in figure 3, illustrating the mutal communication between the two.

2.1 Command Terminal Overview

Any PC, with Mathworks' Matlab Release 13 installed on it and at least one serial port, can be turned into the command terminal, with command terminal software installed. The command terminal PC runs a Matlab GUI application that was created with Matlab's Graphical User Interface Design Environment (GUIDE) to allow a user easy manipulation of the mobile vestibular platform system, while hiding the complexities of the calculations and communication that are required to carry-out those operations. The Matlab command terminal GUI application

allows operation of the system on different levels in that the user may command the mobile platform to move as a whole unit to different locations, or it may issue commands to perform more fundamental operations, such as the movement of one or more motors individually. From this command terminal, the user may also check the status of different components of the system as well as monitor overall system status. A very long serial cable connected the command terminal and the mobile platform to enable serial communication. The main objectives of creating the command terminal was, not only allow to the user easy control over the operations of the mobile platform, but to also provide a remote method of control of the platform. This is essential in situations where the platform cannot easily be reached for maintenance or manual control, which may be a large majority of the time since the mobile platform is most suitable for implementation in environments that are hostile or that make frequent access impractical.

2.2 Mobile Platform (End-Effector) Overview

The bulk of the design of the system was concentrated on the mobile platform. Mechanical, electrical, and software design were needed in order to effectively implement the system. Later sections will elaborate, in much more detail, the mechanical, electrical, and software aspects of the design of the mobile platform. The mobile platform, in this system, is the end-effector of the cable robot where the sensory equipment is to be attached. The mobile platform provides mobility to the attached sensory equipment. In essence, it 'houses' the sensory equipment, while also providing mobility, and thus justifies the term "vestibular" in its name since it acts much like a car or train vestibule, transporting the equipment onboard. The platform has the ability to move in three dimensional space, within a volume limited only by the placement of its cable mounts. In addition, because of its mechanical design, it has the ability to change its relative orientation. For example, instead of being constantly horizontal with the ground surface, it can operate at variable angles, such as forty-five degrees relative to the surface, and thus has some limited rotational capability. The mobile platform is very modular, with the onboard microcontroller controlling all fundamental operations of the various components. It only requires commands, in a pre-determined format, to be issued to it serially in order to execute operations or sequences of operations to perform a task.

3. MECHANICAL DESIGN OF THE MOBILE VESTIBULAR PLATFORM SYSTEM

One of the main objectives was to make the mobile platform design light-weight. With the mobile platform supported by a given number of cables, each with equivalent and constant maximum torque capabilities, each amount less that the mobile platform end-effector can weigh is an additional amount of weight more that can be allocated to sensory equipment. Using sensory equipment with greater weight generally means allowing more complex equipment with greater capabilities, which increases the effectiveness and applicability of the cable robot system as a whole. This section discusses several other mechanical properties of the mobile vestibular platform that may seem relatively insignificant at first, but allow the mobile platform several unique properties.

3.1 Suspension Cables

There are several design issues that affect the operational capabilities of a cable robot system. An obvious consideration is the number of cables from which to suspend the mobile platform end-effector. The minimum number of cables needed to suspend an end-effector is directly related to the number of degrees of freedom desired. For example, if the platform is restricted to only moving along one line in space, then a minimum of two cables are needed; one cable to move the end-effector in one direction along the line and another cable to move it in the opposite direction along the same line. In general, ***n+1*** cables are required in order allow the system to operate with ***n*** degrees of freedom. More generally, for ***n*** degrees of freedom, a system requires ***n+1*** magnitude-varying, yet directionally-constant forces acting upon it [1]. This means that the force of gravity itself can serve as a sufficient surrogate for an additional cable, along the same direction as the force of gravity. In the particular cable robot design outlined in this paper, for example, only three physical cables are used to suspend and move the platform, yet the system can still perform movements through three dimensional space. With three degrees of freedom, the system requires 3+1 = 4 forces acting upon it. They are provided by the three physical cables plus the force of gravity, pulling the platform toward the center of mass of the Earth.

This particular design, where fewer physical cables are used, than are required for a system operating with a particular number of degrees of freedom, is defined as an *under-constrained* system (shown in figure 4, right). In other applications, such as the futuristic space station application proposed, in which the cable robot is used to survey the exterior of a space shuttle or station, the negligible overall force of gravity in any particular direction necessitates a fourth force to allow three dimensional movement in the region outside a space craft. [1] This can be accomplished, perhaps, by some type of onboard propulsion system. This type of system, in which all of the forces required for the particular number of degrees of freedom are implemented by system-controlled forces (ie. cables, constant propulsion devices) is defined as a *fully-constrained* system (shown in figure 4, left). [1] The mobile vestibular platform system could have been designed as a fully-constrained system. However, adding another suspension cable by which to manipulate movement of the mobile platform increases the complexity of the program and also requires more equipment on the platform, increasing the weight and cost. Since the force or gravity serves as a sufficient fourth force, to allow three degrees of freedom, the inclusion of a fourth cable in the direction of gravity was justifiably neglected in the design.

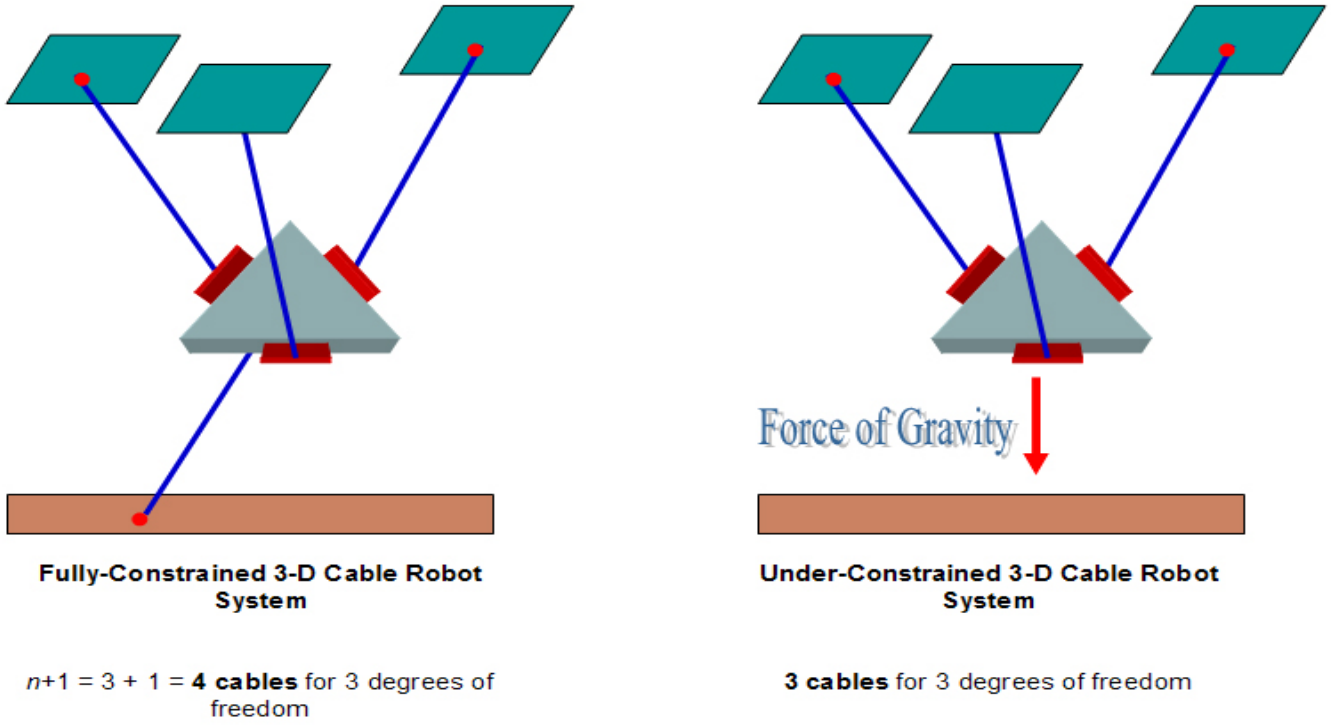

Fig. 4: Right: Under-constrained MVP system versus, Left: full-constrained MVP system

3.2 Cable Mount Locations

Related to the number of cables is the placement or the mounting locations of the cables on the mobile platform itself. Different on-platform, cable-mounting orientations allow different capabilities for the robotic system. For example, mounting all of the suspension cables on the periphery of the end-effector allows more stability against external disturbances, such as strong winds, collisions with other moving objects, during system operations. This is illustrated in figure 5 (right), where a strong wind can cause a point-mass cable mounting configured end-effector to sway from side to side severely. The image on the left in figure 5 displays the relatively little effect the same wind disturbance can have on a peripherally cable mounted end-effector. Also, peripheral mounting of the suspension cables on the end-effector allows for control over its orientation, allowing a mobile platform to orient itself parallel to different relative angular surfaces as needed. An alternative to peripheral orientation of the on-platform cable mounts is mounting the suspension cables at the center of mass of the platform, called point-mass mounting of the cables [4]. The advantage of this method is that there is no 'wobbling' of the platform as it is moved from one location to another. Wobbling is an issue using the peripheral orientation. [4]

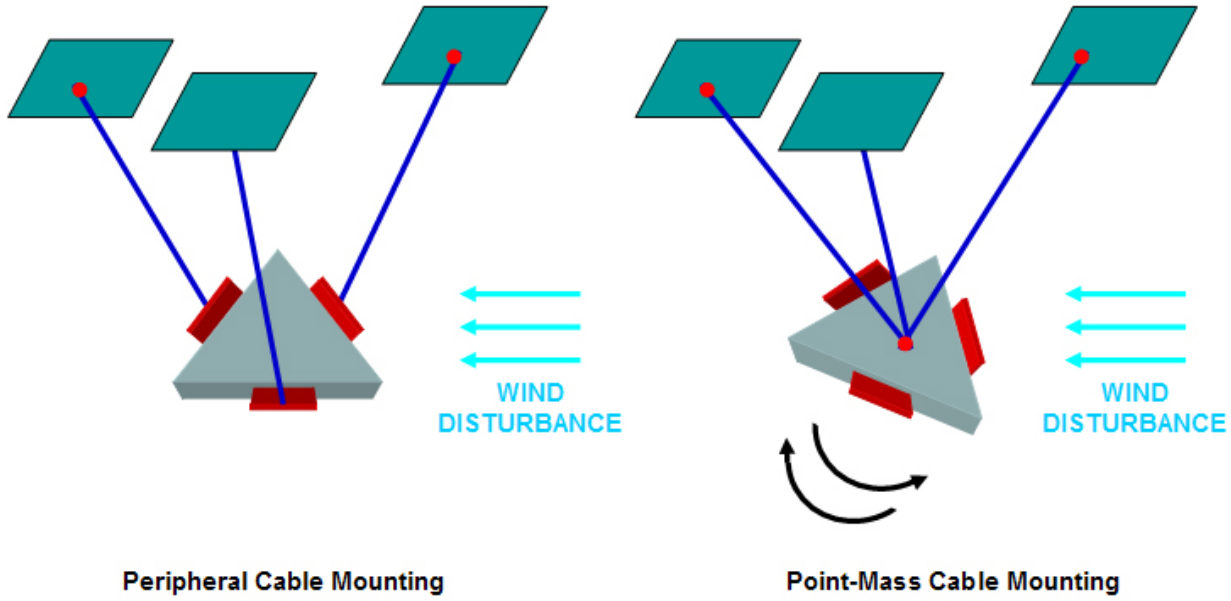

Fig. 5: Comparison of cable mount locations (Left: Peripheral Mounting, Right: Point-Mass Mounting)

'Wobbling' occurs when changing cable lengths, cause the platform to change its angular orientation continuously during movement. This occurs because with any given trajectory, the cable lengths may be required to change nonlinearly with time, in order to move the platform from one point to another. The 'wobbling' effect occurs when the system is operated or programmed assuming that the cables lengths should change linearly, as opposed to nonlinearly, with time. Figure 6 displays this effect with trajectory 1, where when moving the platform from point A to point B, the cable length, L, does not change. However, because of this, the part of the end-effector connected to this cable, will dip lower than other cable connections to the end-effector. A desired trajectory would be trajectory 2, in figure 6, where a straight path is followed from point A to point B. However, this path requires non-linear change in the cable length, L, as the end-effector travels from point A to point B. With multiple cables being used to control a platform, this effect can occur for each cable, each with different degrees of severity, depending upon the trajectory specified. With a center of mass configuration, however, a platform can have only one orientation due to the force of gravity, and thus it is not subject to 'wobbling' (assuming the center of mass remains constant). Thus with the center of mass configuration, no complexity has to be taken into account to minimize the 'wobbling' effect.

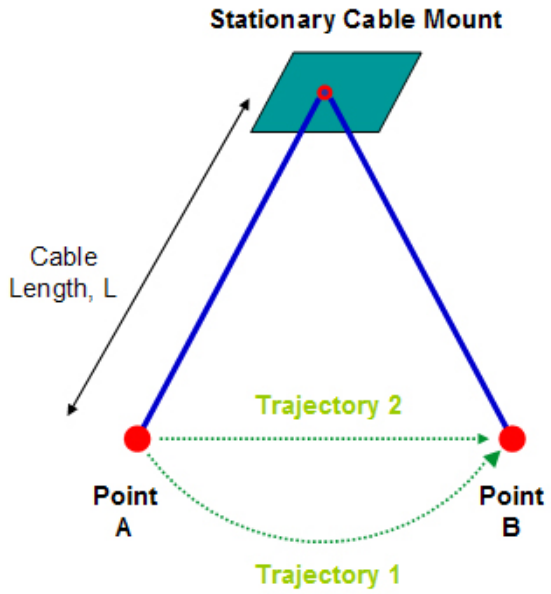

Fig. 6: Illustration of non-linear (Trajectory 2) versus linear (Trajectory 1) cable length actuation along one cable during platform movement

In the mobile vestibular platform, the suspension cables were mounted on the periphery of the platform, near the motors. As a result, the wobbling effect will play a factor in system performance, however, it can be approximately compensated for in the software of the command terminal running the Matlab GUI, the system which can handle the processing of this task most efficiently. Assigning this processing task to the command terminal PC significantly reduces the additional negative complexity that would otherwise be thrown onto the on-platform microcontroller, which is much less efficient in handling such operations.

3.3 Platform Design

The mobile platform that was designed for use as the end-effector in this system is a two-tiered, light-weight plastic structure. The two-tiered structure allows for protection of the equipment housed inside. This is useful as the environment in which the system may be operating is unknown. As a result, the components providing functionality to the system are hidden as well as protected from potential collisions. Also, the two-tiered structure allows for more flexibility in placing sensory and other communications-related equipment. Basically, there's more surface area on which to mount components. Both tiers of the mobile platform are equilateral triangles, which allows sensory equipment to be mounted symmetrically, aiding future platform operations that are dependent upon the comparison of data collected at different locations, such as in the task of locating the source of a type of data.

The three motors, which actuate the lengths of each of the three suspension cables, are mounted on the peripheral edges of the triangular platform, in between the two tiers. This particular placement of the cable length actuators, or motors, is different from many other conventional or past developed cable robot systems. The motors are not mounted to the stable cable mounting locations in the environment, but rather, they are mounted on the end-effector. Also, the suspension cable mounts in the environment (as opposed to on the end-effector) are

simply permanent attachments of the cable to the stable mount location, such as a ceiling or pole. There are several advantages to the placement of all of the motors on the mobile platform. First, the microcontroller on the platform has full control of the motors and thus its movements, making it very modular. In addition, when the system is used in difficult-to-reach locations, the motors can easily be replaced or maintained by having the mobile platform move to the most convenient location. When the motors are mounted to remote stable locations within the environment, repairing or replacing the motors can be very difficult. In addition, for remotely-mounted motors, a separate method of communication from a command terminal is needed. In the mobile vestibular platform, communication is accomplished through a single serial cable from the command terminal to the mobile platform.

4. ELECTRICAL DESIGN OF THE MOBILE VESTIBULAR PLATFORM SYSTEM

The electrical system that controls the mobile platform and its components is a relatively simple, modular system, consisting of components (IC's) that each perform a unique task in the operation of the mobile platform. The on-board microcontroller acts as the processing center for the mobile platform, directly receiving serial communications as well as issuing the commands to operate the rest of the components of the board. In addition to this, the program to control the microcontroller, and thus the entire mobile platform electrical system, was permanently installed in the system, reducing the end-effector system's reliance on other external systems by allowing it the capability to perform all of the basic operations independently. The following sections give a detailed description of the role of the various components of the mobile platform's electrical system and explain the relationships among them. Also discussed are the problems and obstacles encountered in developing the electrical aspects of the mobile vestibular platform system, and how they were solved.

4.1 Components

Adapt11C24DX Microcontroller Board with Embedded Motorola M68HC11 Microcontoller:

The Adapt11C24DX microcontroller board, with its embedded M68HC11 microcontroller, is the microcontroller unit (MCU) used on-board the mobile platform to control all the fundamental operations of the platform. It is also the board to which the serial cable is connected to receive communication from the command terminal. The mobile platform control program, to be run by the MCU, is compiled and the resulting machine code loaded into the on-board memory of the Adapt11C24DX microcontroller board. This control program is then executed by the MCU, allowing it to process the commands received serially and to execute the operations needed to carry out those commands received. The MCU controls the LM629 IC's, or precision DC motor controller IC's, to be explained momentarily, instructing them to execute the desired procedures to correctly operate the DC motors. The microcontroller essentially acts as the mobile platform's personal processing unit, which is what gives the mobile platform its modularity. Much like a computer with many peripheral modules being controlled by a central processing unit (CPU), the mobile platform also consists of many peripheral components or IC's which are controlled, directly or indirectly, by the MCU. The microcontroller relies only on commands issued from an external source in order to perform its basic operations of controlling the motion of the three motors. After receiving commands from the command terminal, it

communicates with the LM629 IC's through several I/O pins to issue the proper instructions. A modular diagram of the on-board peripheral components of the microcontroller board is shown in figure 7, below.

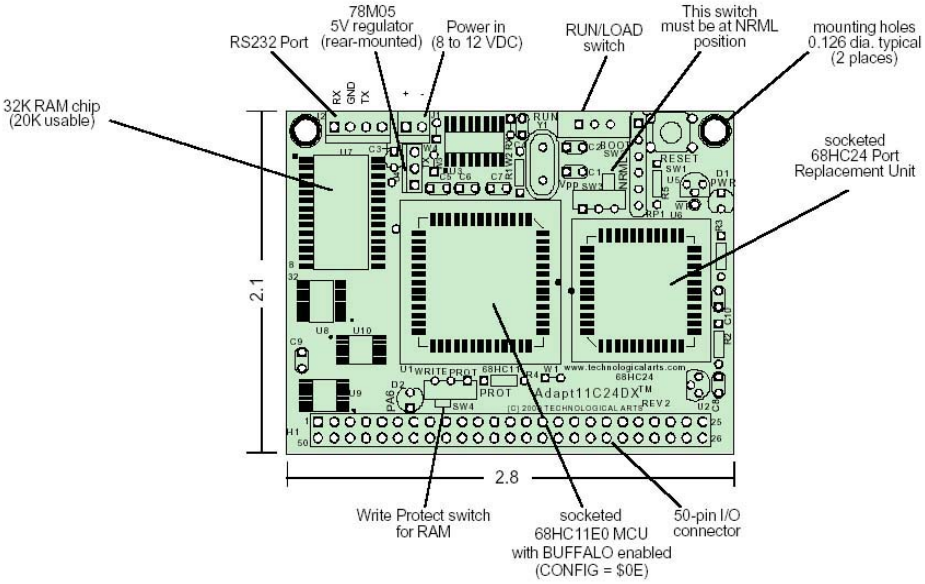

Fig. 7: Adapt11C24DX microcontroller board with embedded Motorola M68HC11 MCU

An advantageous feature of the Adapt11C24DX microcontroller board is that it contains 32K of electrically erasable programmable memory (EEPROM). This is a type of ROM that can be cleared, modified, and written through special procedures. Anything stored in EEPROM is not erased when the power supply is disrupted, as opposed to random access memory (RAM), where any type of power disruption wipes out all contents of the memory. This capability of the microcontroller board is a major contributor to making the mobile platform more modular, since the program to be run by the microcontroller does not need to be reloaded every time the power to the mobile platform system is reconnected or upon every start-up of the system.

National Semiconductor LM629N-6 Precision DC Motor Controller:

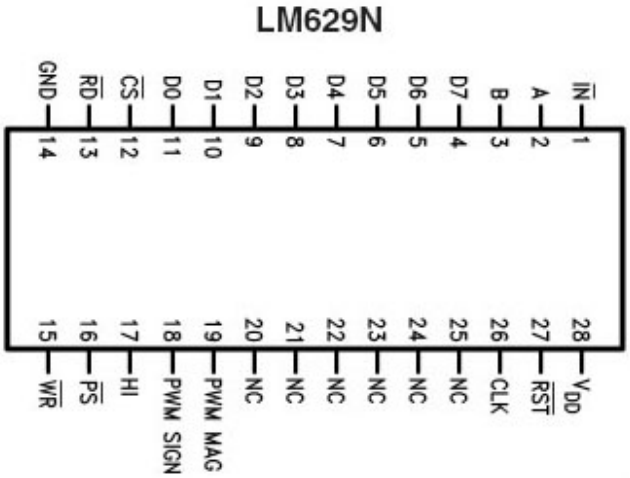

Fig. 8: LM629N-6 Precision DC Motor Controller IC

The LM629N-6 (figure 8) is, as its name implies, a high precision DC motor controller. This precision DC motor controller IC plays a large role by carrying out an entire aspect of the mobile platform electrical system, which is the precise control of the motor system. It can control the motion of a DC motor with significant precision in revolutions per second. The LM629 IC itself acts as a limited microprocessor, receiving and executing commands, although in reality it doesn't perform any processing. It is capable of receiving commands from its data lines. To put it simply, the fundamental commands it can perform allow it to receive an angular position to which to move the motor as well as an angular velocity and acceleration at which to move the motor. When the IC has received this information from the data lines, and is then commanded to execute the position change of the motor, it produces a pulse width modulated (PWM) signal to drive the motor to the intended position. In addition to this, the LM629 has two pins to receive optical encoder feedback from the motor, which provides a precise indication of the current position of the motor through the use of a quadrature amplitude encoded signal. With this two-line signal, the LM629 can make adjustments to the PWM signal being sent to the DC motor in order to speed it up, slow it down, or stop it, so that the desired position, velocity, and acceleration parameters are met as closely as possible. In this way, much of the processing needed to produce and adjust the necessary PWM signals, as well as the calculations to process the optical encoder feedback, are lifted off of the shoulders of the M68HC11 microcontroller on the platform, and automated in the LM629 IC instead. In addition, a large portion of the code and complexity that would otherwise have to be programmed and loaded into the microcontroller board's memory can be cut out.

National Semiconductor LMD18201T P+ H-Bridge:

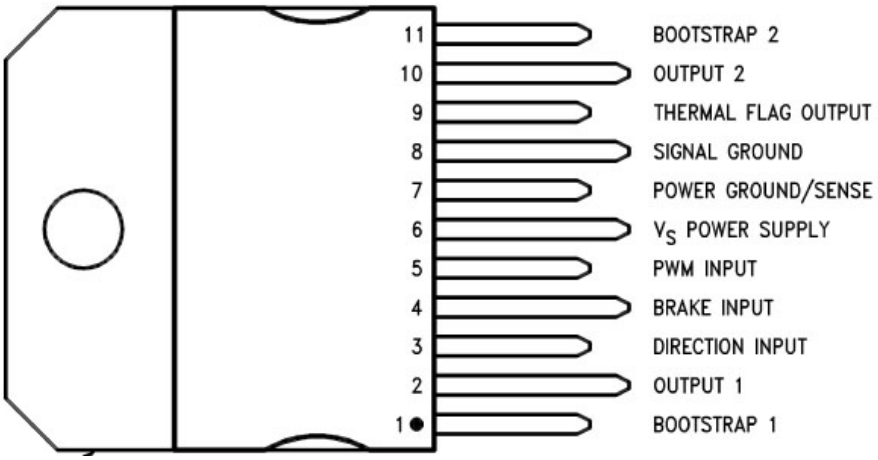

Fig. 9: LMD18201T P+ H-Bridge DC Motor Driver

Although the LM629 precision DC motor controller produces the PWM signal to operate the DC motors at the desired angular velocity and acceleration to reach the desired angular position, the signal power is relatively low since the IC's themselves cannot draw too much current without burning out. As a result, the PWM signals produced by the precision motor

controllers must be current-amplified in order to drive the DC motors. This is accomplished by the LMD 18201T P+ H-Bridge DC motor drivers. Given an input PWM signal, the H-bridge IC takes that PWM signal and produces the same PWM signal but at a higher voltage and current. For example, in this system the LM629 produces a PWM signal with a peak voltage of 5 V. The H-bridge takes that signal and amplifies it to a peak voltage of 12 V, while still maintaining the time characteristics of the signal and allowing the current drawn by the DC motor, from the H-bridge, to be increased significantly, giving the DC motors the necessary power to effectively actuate the mobile platform. The LMD18201T IC also has several other features such as a brake pin, which causes a signal of 0 V DC to be sent to the DC motor, immediately stopping any signal driving the DC motor. Another pin of the H-bridge, directly used by the LM629 precision DC motor controller, is the direction pin. This pin toggles the polarity of the PWM signal produced by the H-bridge and driving the DC motor. As a result, when the LM629 needs to do so, the DC motor will be driven in the opposite direction and thus will move with a negative, or reverse angular velocity. The DC motor controls the lengths of the cables by which the mobile platform is suspended and thus, the cables must be able to be reeled in and out in order to allow free movement of the mobile platform anywhere in a three dimensional space. In a more simplified model, the LMD18201T H-bridge can be thought of as a PWM signal amplifier, which receives the low voltage-current PWM signal from the LM629 precision DC motor controller, and amplifies this signal to the necessary characteristics in order to directly drive the DC motors.

Maxon DC Motor with Quadrature Amplitude Optical Encoder Feedback:

Fig. 10: Maxon DC Motor with Optical Encoder Positioning Feedback

While the LM629 precision motor controllers send PWM signals to the H-bridges, the H-bridges in turn send their output PWM signals directly to the Maxon DC motors, shown in figure 10. These motors wind and unwind the spools to which they are attached to lengthen or shorten suspension cables, allowing the mobile platform to change its position in three dimensional space. These DC motors, manufactured by Maxon, are also equipped with optical encoder feedback, allowing precise determination of the DC motor's operation or movement. Put simply, the optical encoder system produces two signals or channels that are sent back to the LM629 precision motor controller. The optical encoder feedback signals are quadrature amplitude encoded signals from which increments in the smallest rotational unit (which is determined by the precision of the optical encoder system) can be determined as well as the direction of rotation. The optical encoder system measures these increments in rotational units by optically

monitoring the rotational component of the DC motor. As a result, the data sent to the LM629 from the optical encoder of the DC motor allows for a measure of the real physical change in angular position of the DC motor, allowing the LM629 IC to make the necessary adjustments to the PWM signal driving the DC motor. In this way, there is a feedback loop from the DC motor output to the LM629 precision motor controller, so that adjustments can be made to follow the specified trajectory as closely as possible. In addition to this, this feedback from the DC motors allows the LM629 to make adjustments so that the DC motor stays in the same position once it has reached the desired angular position. When the platform has moved to its final position and must stay stationary in that position, the DC motor optical encoder feedback will provide the necessary data to the LM629 so that adjustments may be made in the PWM signal sent to the DC motor, to maintain the motor in the same position despite opposing torque from the load, which may be the platform itself along with other equipment.

4.2 PCB Design

The electrical system of the mobile platform was first implemented and tested on a series of proto-boards, with all of the motors connected. The circuit that was assembled on the proto-boards was exactly equivalent to the electrical system to be implemented on the two-tiered mobile platform. However, because the proto-board circuit required the use of almost three proto-boards, each being rather heavy, the equivalent circuit was implemented and laid out on a printed circuit board (PCB), on which all of the components needed for the circuit could easily and robustly be soldered. This greatly reduced the weight of the circuit and also allowed for it to be made much more compact in order to better fit on the mobile platform. In addition, all of the lines that were used to connect the proto-board components were readily capable of being accidentally removed, making the idea of using a PCB much more favorable. Two PCB designs were laid out, the first having dimensions of approximately 3.8 x 4.5 inches, and the second (shown in figure 11) much more compact with dimensions of approximately 3.6 x 3.9 inches. The second PCB layout significantly cut down on excess area and therefore seemed like the obvious choice to have manufactured. Problems caused by the particular PCB design used are discussed in section 4.4.

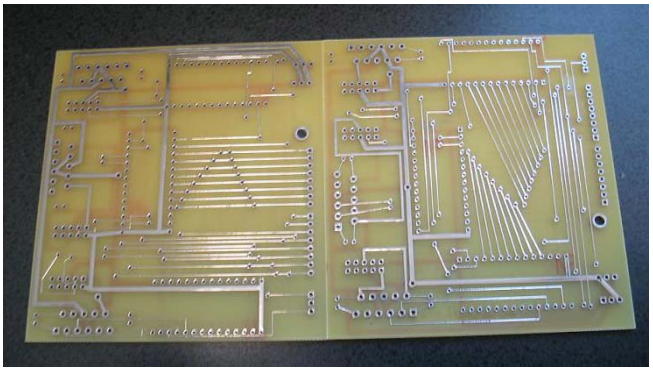

Fig. 11: Mobile Platform End-Effector System PCB

4.3 Flowchart of the Electrical System

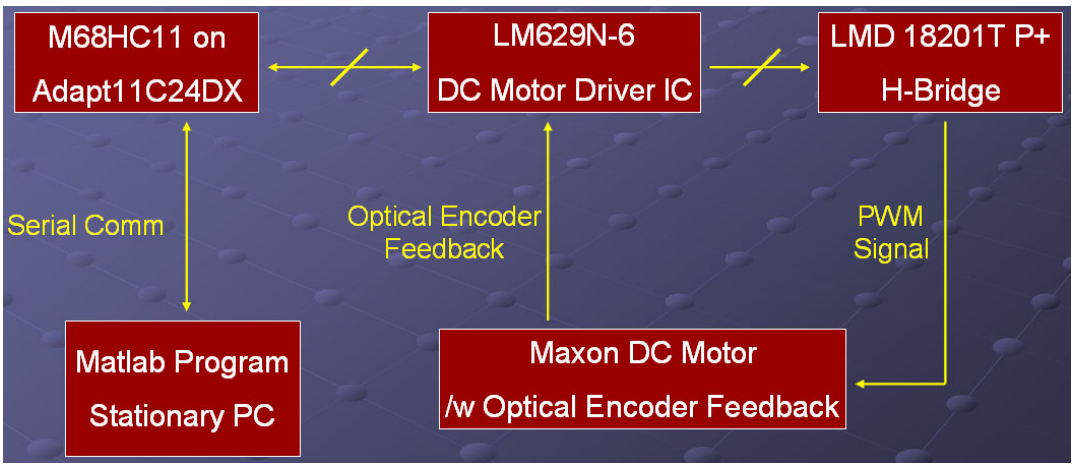

Fig. 12: Mobile Vestibular Platform Electrical System Flowchart

Figure 12 is the flowchart that corresponds to the software/hardware system developed for this cable robot system. Beginning with the Adapt11C24DX microcontroller board, we can see that this component communicates with the LM629 precision DC motor controller IC through several lines (data and command). The microcontroller follows the protocols required in order to issue byte commands to the LM629. The LM629 then communicates with the H-bridge by sending it a PWM signal, with which the DC motor should be driven, and a direction signal. The H-bridge, in turn, takes these two signals and produces a two channel PWM signal equivalent to the one received from the LM629, but voltage and current amplified sufficiently enough to drive the DC motor. These two PWM output channels directly drive the Maxon DC motor. From here, the optical encoder feedback system produces a two channel quadrature amplitude encoded signal that is sent back to the LM629, which then uses this signal to keep track of the current real angular position, velocity, and acceleration of the motor. With this information, the LM629 then makes the necessary adjustments to the PWM signal in order that the trajectory programmed into the LM629, by the microcontroller, is followed as closely as possible.

4.4 Problems/Issues

Of all of the problems and obstacles that have arisen during the development of the mobile vestibular platform system, the most persistent and difficult to debug problems, by far, were related to the electrical circuit for the mobile platform. Many of these problems seemed to be related to interference with the LM629 precision DC motor controller IC's. The circuit first developed, on a series of proto-boards, was found to consistently operate correctly, as opposed to the PCB circuit. The proto-board circuit was very well organized and easy to debug. In addition to this, all of the lines were well isolated simply due to the nature of proto-board. Therefore, this circuit was compared to the PCB circuit in an effort to determine the sources of various problems. Of all of the problems encountered during the development stages, the most interesting two are discussed below.

Floating LM629 Data Lines:

The first interesting issue encountered was a problem that occurred on the proto-board circuit. One motor could run by itself, however, when multiple motors were run, they operated in a very sporadic and random way, clearly not following the intended trajectory. Because this only occurred with multiple motors being run, or multiple LM629's being operated at the same time, it was deduced that the problem lay with the LM629's somehow causing interference with one another. However, the LM629's were connected to one another only through the control and data lines. Only the data lines were occasionally left to "float" (a situation in which no particular voltage is applied over the lines and thus random signals may travel). It was hypothesized that operating LM629's created unintentional signals over these 'floating' data lines, and those signals were interfering with the operation of other LM629 IC's. The microcontroller program was modified so that control over the control and data lines to the LM629's was 'tighter'; whenever control or data lines were not used by the microcontroller, they were set to a definite value (0 V or 5 V). This seemed to solve this particular problem on the proto-board circuit.

Electro-magnetic Interference (EMI):

Electro-magnetic interference was probably the most time consuming issue encountered. EMI occurred in several ways, requiring major modifications to be made to the PCB circuit. In the mobile platform electrical system, there are two power lines and a ground line. The first power line is the 5 V line, which powers the LM629's, the optical encoders on each DC motor, the microcontroller, and any other component powered through the microcontroller. The second power line is the 12 V line, which is used only to power the H-bridges, driving the DC motors. The issue arises with the H-bridges, which produce a high voltage, high current PWM signal. This signal varies from 0 V, and thus 0 A current, to 12 V with a correspondingly high current capable of reaching 1 A. Because a PWM signal is produced by the H-bridge, the current drawn from the 12 V line must vary rapidly over a large range of 0 A to 3 A in the worst case, since three H-bridges are driving three motors simultaneously and are powered by the same 12 V power line. Thus, we can assume that a large amount of EMI is produced along the 12 V power line. The negative effects of the EMI, produced along the 12 V line, were seen in several places on the PCB, determined through experimentation. The first place where this was a problem was where the 12 V and 5 V lines ran a long length of the board in very close proximity. The result was that the 5 V line became affected by the EMI produced, causing the LM629's to reset and behave erratically in driving the motor. Another place where this problem occurred was in the close proximity of the H-bridges and LM629's. It seemed that the PWM produced at the output of the H-bridges was located very close to the LM629's creating internal disturbances within the LM629 IC's themselves, and causing internal registers to store incorrect values. Furthermore, placing a 12 V line wire directly over an LM629 also caused it to operate incorrectly, making the DC motors operate with seemingly random trajectories, further supporting the hypothesis of EMI being the source of several problems. On the complete prototype of the mobile platform, these issues were carefully dealt with. For example, the 12 V line was disconnected from the PCB and all components requiring the 12 V power line were also removed and isolated to another region, away from any 5 V power lines and LM629 IC's. This required extensive manual modifications to the PCB as well as a new bread-board circuit to be soldered together with the sole purpose of isolating the 12 V line and its components from the rest of the circuit. The EMI problem also played a role in the physical placement of the components on the final prototype of the mobile

platform, since now some components were restricted in how close they could physically be to other specific components.

4.5 Completed Mobile Platform End-Effector

The completed mobile vestibular platform system's end-effector is displayed in figure 13, below. The Adapt11C24DX microcontroller can be seen on the top tier of the platform with the PCB, holding the LM629's behind it. A separate soldered bread board can be seen on the right side of the top tier of the platform, to which the motor's connect and on which the 5 V and 12 V lines can be isolated. The bottom tier contains the three motors with their spools and cables attached to each. The H-bridges, on their own PCB's can be seen here, each under the spool belonging to motor which the H-bridge controls.

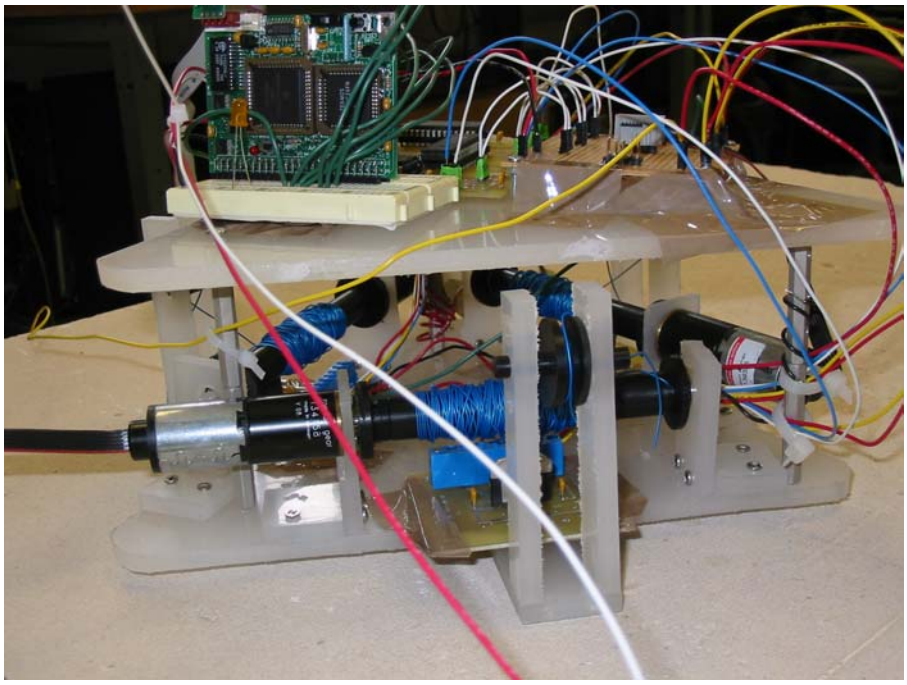

Fig. 13: Complete End-Effector of the Mobile Vestibular Platform System

5. SOFTWARE SYSTEM OF THE MOBILE VESTIBULAR PLATFORM SYSTEM

All electrical components of the mobile vestibular platform system are controlled by two software modules. One module is the mobile platform system program that controls all operations of the mobile platform system through the Motorola M68HC11 microcontroller. This program was coded using Technological Arts' ICC11 software, in C, and was compiled and downloaded to the EEPROM module piggy-backed on the Adapt11C24DX microcontroller board, using Technological Arts' MicroLoad program. This particular program is modular in the sense that it does not rely on external systems in order to perform the basic functions to operate the mobile platform, such as operating the DC motors to run specific trajectories. The other program module is the Matlab GUI application which acts as the command terminal from which a user can remotely operate several different functions of the mobile platform. This module was programmed using Matlab's Graphical User Interface Development Environment (GUIDE) and

is run by Matlab to communicate instructions and data to and from the mobile platform through the PC's serial port.

5.1 Mobile Platform End-Effector System Software

This program was written largely in order to allow the Adapt11C24DX microcontroller board to program and operate the LM629 precision DC motor controller IC. The program for this module can be found in Appendix A. It contains several functions, which perform simple but frequently used operations such as writing command bytes to and reading data bytes from the LM629 IC. Several hierarchically higher functions use the basic command and data writing and reading functions in order to perform specific tasks such as programming a specific trajectory into the LM629 or reading the many registers that provide information bytes indicating the status of the precision DC motor controller IC. For example, there are functions that issue commands to the LM629 in order to read the various status registers such as chip status, current real position of the motor, temporal desired position of the motor, current real velocity of the motor, temporal desired velocity of the motor, etc. The LM629 is a very complex IC with many different functions and capabilities and thus, it makes sense that the mobile platform system program would require many functions in order to cover all of the operations the LM629 is capable of. In addition to this, the program is also responsible for the control of all three motors of the mobile platform.

The mobile platform system, however, will not perform any functions without the commands to do so. The commands needed to carry out operations are received through the serial port on the microcontroller board. A baud rate of 19200 bps is used to transfer commands and data between the remote user command terminal and mobile platform system, with a specific format that is hard-coded in the mobile platform system software. Because the commands are received through the serial port using the RS-232 protocol and use a specific format and set of text commands, the commands can be issued from any source or program with access to a serial port and the correct format. This makes the mobile platform system itself largely modular and capable of being used as a component in other, perhaps larger or upgraded, systems.

5.2 Command Terminal Software

The other software module used in the mobile vestibular platform system, as mentioned above, is the remote user command terminal program. Whereas the mobile platform system software handles the processing and execution of instructions to perform basic functions on the mobile platform, the command terminal program processes and executes direct commands from the user, requiring more complex computation as well as more complex sequences of instruction to be issued to the mobile platform system. For instance, there are fields in the command terminal GUI, shown in figure 14, that allow the user to move the mobile platform from its current position in three dimensional space to a new position by entering the new coordinates into the proper fields of the GUI. The GUI then takes all of the data from the proper fields and performs the calculations to determine the change in each cable length that must occur and then the trajectory that should be taken by each motor to change the cable lengths. Thus it must calculate the number of basic rotational units for each motor in order to change the cable lengths to the desired lengths. These calculations depend upon several initial factors including the

dimensions of the platform, the positions of the motors, and the positions of the stable external cable mounts. The center of the mobile platform at its initial position upon powering up the system is considered to be the origin and all initial positions of the cable mounts must be measured relative to that origin. The cable mount locations relative to the origin must be entered into the GUI upon powering up the system, before any platform movements commands are issued from the GUI. All other initial data required, such as the platform dimensions and motor positions are already hard-coded into the GUI application calculations. Upon completing the required calculations, the Matlab GUI application sends the appropriate text commands to the mobile platform system which instructs the motors to operate properly in order to move the platform to the desired location. The software source code for the command terminal module can be found in Appendix B. It was coded using Matlab's own programming syntax.

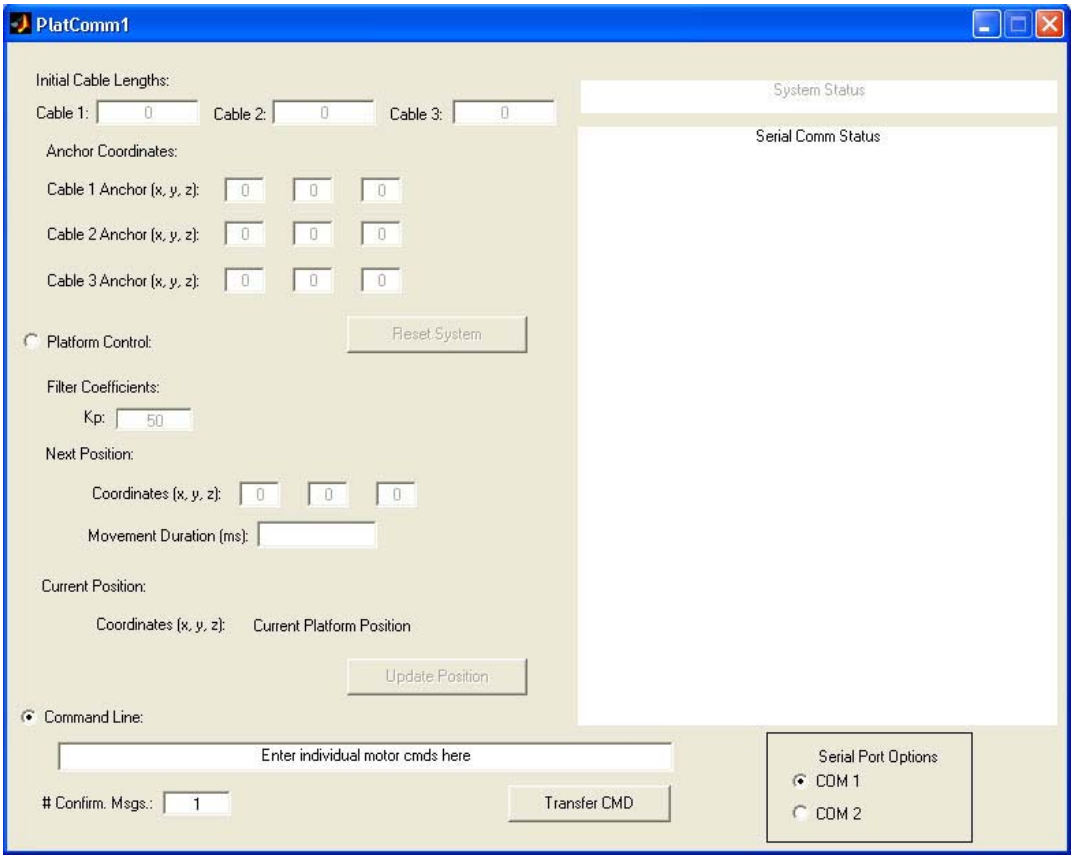

Fig. 14: Remote User Command Terminal GUI, developed in Matlab using GUIDE

The major advantage obtained by using this command terminal Matlab GUI is the reduction in calculations that the M68HC11 microcontroller would otherwise have to perform. In addition, the calculations needed to be performed are floating point (or decimal) calculations. These calculations require significantly more processing power to complete. Since the calculations performed by the Matlab GUI are largely floating point calculations, putting this extra work on the shoulders of the microcontroller would undoubtedly cause the system to run more slowly, likely resulting in a lag between the time the user enters a command and the time the mobile platform actually begins to move. This severity of this problem would likely be amplified in the future, when more complex algorithms are implemented to reduce other unintended behaviors of

the platform. By having the Matlab GUI perform all of the required floating point calculations for moving the mobile platform between coordinates, the calculations can be done much faster, since the PC has much greater processing power than the microcontroller.

One other aspect of the Matlab GUI command terminal application is that it allows the user to enter text commands to be sent through the serial port directly to the mobile platform. Although the user must be familiar with the command format and commands that can be sent to the mobile platform to execute these commands, this capability of the GUI allows the user an easy method to control hierarchically lower functions of the mobile platform such as operating each motor individually as well as changing different parameters used by the LM629's and checking the status of the system through the LM629's status registers. This has been very useful for debugging and minor adjustment purposes on the mobile platform.

6. DISCUSSION AND CONCLUSIONS

A large portion of the development of the mobile vestibular platform system was spent debugging and determining the source of problems in the circuit. However, a good deal of work was accomplished in the development of this system, with various modifications in the original design being made in the electrical system of the mobile platform. Testing of various operations of the mobile platform raises some issues that may be potential problems in the future. There are also many more features that could be implemented in the mobile vestibular platform system, in both hardware as well as software, to make it more applicable to different tasks and convenient to operate.

6.1 Foreseeable System Issues

In surveying and evaluating the operation of the mobile vestibular platform system, a couple of potential sources of future problems arise. These problems occur in both the mechanical as well as electrical realm of the mobile platform design. First, in the mechanical design, the suspension cables currently used are very thin wires, which are relatively easy to break or stretch. With stretching occurring frequently in the suspension cables, commands issued to move the mobile platform to a particular position in three dimensional space will become less accurate since the stretching will gradually cause the mobile platform to deviate from its desired 'stationary' position. In the electrical design of the system, the mobile platform currently uses two external power supplies to power itself, one for the 5 V supply and one for the 12 V supply. This is due to the fact that even when the 5 V supply is drawn from the output of a voltage regulator supplied with the 12 V line, EMI still causes significant problems and thus, two separate power supplies are used instead. Eventually, the mobile platform will have to operate with an on-board power supply such as a battery. This may become troublesome if the same problem with interference arises. Having two separate power supplies for these two voltages, on-board the mobile platform, is impractical since it would significantly increase the load being operated by the motors.

6.2 Future Work

There is an endless supply of ideas that can be applied to the mobile vestibular platform system to enhance as well as to apply it. These ideas apply both to the hardware and the software aspects of this cable robot system.

Of these, several were for this research program but were not completed. The first is infrared (IR) remote control of the mobile platform. With an IR detector in the mobile platform electrical system, the on-board microcontroller could detect and measure pulse widths from an IR remote control, and decode the pulse sequences received to determine which button on the IR remote control was pressed. Based on the button pressed, the microcontroller can then execute a function associated with that particular remote control button. This would be useful for manual adjustment of individual motors while surveying the operation of the motor from a location other than the remote user command terminal. This method of adjustment could be a useful, temporary solution to re-align the mobile platform to a horizontal position parallel to the ground, when stretching cable lengths alter this. In addition, other buttons on the remote can be applied to operate future components aboard the mobile platform.

Another enhancement to the system is a software solution to the 'wobbling' effect, explained in section 3.2, where cable mount locations on the mobile platform end-effector are discussed. A software solution to this problem can be implemented in the Matlab GUI command terminal application. When a user enters a new coordinate for the mobile platform, the Matlab program can take the trajectory from the mobile platform's current position to the new position and divide it into several unit divisions along the trajectory. Then, the Matlab GUI can issue a separate trajectory command to the mobile platform for each of these unit divisions along the overall intended platform path until the desired position is reached. As a result, the 'wobbling' effect will be greatly reduced because it will only occur in the continuous transition for each of these small unit path divisions, which will barely be noticeable. The downside is that the same set of floating point calculations will have be made for each of these small platform movements along the path to the destination position, whereas only one set of calculations would otherwise have to be made for one continuous movement of the platform from the current position to the destination position. However, this does not pose a significant problem since these calculations are performed on a PC with significant processing power. Each set of calculations will be computed before the mobile platform is even nearly completed executing the trajectory for the previous path division.

One final additional feature that would have been rather exciting to implement on the mobile vestibular platform system would be some type of sensory equipment, such as an IR detector. Several IR detectors could be installed on the bottom of the mobile platform, with each connected to an A/D converter on the microcontroller board. As a result, the microcontroller would collect and process the data from these IR sensors and act according to a pre-programmed algorithm. For instance, it would have been very interesting to program the mobile platform system to follow an IR emitter within its reachable space. A simple algorithm to implement and test would be to instruct the mobile platform to move in the direction of the IR detector which detected the strongest IR signal from the emitter, allowing the mobile platform to approximately follow an IR emitting source.

There are an endless number of possibilities for enhancements and applications of the mobile vestibular platform. It will be interesting to see how this field of robotics develops in the future and how readily cable robot systems will be applied to a variety of environments including industrial, military, and domestic households. The unique capabilities that are made possible by the fundamental mechanical design of the cable robot make it a very suitable alternative to conventional wheeled robots, for use in a variety of applications and tasks in the real world and probably even more in the future.

7. RECOMMENDATIONS

I have a few recommendations which I believe may be of use for future development of the mobile vestibular platform system. Many of the recommendations relate to the issue protecting the electrical system of the mobile platform from the unwanted effects resulting from electro-magnetic interference (EMI). One possible solution to this problem is to isolate each LM629, H-bridge, and DC motor connector on a different PCB. From past experimentation, it seemed that operating one LM629 and H-bridge set on a PCB did not cause problems. This is a solution that currently being experimentally implemented for the mobile platform end-effector system. Another benefit from this is added modularity of the electrical circuit of the system. With this design, if more suspension cables, and thus LM629's and their H-bridges, are to be added to the cable robot system, it can be done simply by using another one of these PCB's, containing only a set of components to operate a single motor.

Another simpler recommendation, relating to the PCB design, would be to have the PCB's manufactured with an epoxy coating, in order to prevent accidentally applying voltages from hanging wires to incorrect lines. This is a simple recommendation in order to help safe guard against burning out the LM629's which occurred quite often and became somewhat expensive, due to the 12 V line being so close to the 5 V line on the PCB, with both lines exposed and unprotected by any type of non-conducting coating.

Finally, it may be useful to use fishing wire, which can sustain significantly more tension than the cable currently used, as the suspension cables. The cables used now were originally intended to supply the voltage-current needed by the system. However, since this method is no longer used, the suspension cables can be replaced with fishing wire which is less susceptible to stretching and can support significantly heavier loads.

8. ACKNOWLEDGMENTS

This research program has offered me the opportunity to experience ten weeks of intense research and development in a field I previously had little knowledge about. I feel I have grown greatly and have also gained invaluable experience in the research and development process. An academic knowledge in electric circuits and systems as well as programming is largely the only thing I brought with me into the program. However, I have come out having learned many skills and concepts through from research, lab experience, and my own mistakes due to lack of

experience. I would like to thank the National Science Foundation's Research Experience for Undergraduates program for their support of the SUNFEST Undergraduate Research Program at the University of Pennsylvania. I would also like to recognize the financial support of the Microsoft Corporation.

Much appreciation and many thanks also go to my advisor, Dr. Daniel Lee, for his invaluable support, time, patience, and advice in helping a relatively un-experienced undergraduate student learn the ropes of the research and development process.

Furthermore, I'd also like to thank the graduate students Chintan Cadala and Mubeen Bhatti, working under Dr. Daniel Lee, for their support and patience in helping me become acquainted with the project as well as for their suggestions and advice.

9. REFERENCES

1. Bosscher, P., Ebert-Uphoff, I., Lipkin, H., Riechel, A. (2003) Concept Paper: Cable-Driven Robots for Use in Hazardous Environments. http://www.google.com/url?sa=U&start=1&q=http://robot.me.gatech.edu/~paul/papers/RRSHE2004_concept_paper.pdf&e=747

2. Ebert-Uphoff, I., Voglewede, P. (2004) On the Connections Between Cable-Driven Robots, Parallel Manipulators and Grasping. http://robot.me.gatech.edu/~phil/downloads/ICRA2004_connections.pdf

3. Oh, S., Mankala, K., Agrawal, S., Albus, J., Dynamic Modelling and Robust Controller Design of a Two-Stage Parallel Cable Robot, Proc. 2004 IEEE ICRA Conf., New Orleans, LA, USA, April 2004, pp. 3678-3683. http://ieeexplore.ieee.org/xpl/abs_free.jsp?arNumber=1308830

4. Riechel, A., Ebert-Uphoff, I. (2003) Force-Feasible Workspace Analysis for Underconstrained, Point-Mass Cable Robots. http://www.google.com/url?sa=U&start=1&q=http://robot.me.gatech.edu/~andrew/ffw_paper3.pdf&e=747

APPENDIX A: Mobile Vestibular Platform - End-Effector System Software

PLTFRM2.C
(Source Code in C, to be compiled in ImageCraft's ICC11 for download to Technological Arts' Adapt11C24DX Microcontroller Board with Embedded Motorola M68HC11 Microcontroller)

```
#include <hc11.h>
#include <ctype.h>
#include <string.h>

#define BAUD19200 0x03
#pragma interrupt_handler TOIISR
/*
#pragma interrupt_handler IC1ISR
#pragma interrupt_handler IC2ISR
#pragma interrupt_handler IC3ISR
*/
unsigned int byte = 0;
unsigned int STATUS = 0;
unsigned int choice = 0;
unsigned int cur_Motor = 0;

unsigned int input = 0;
char msg_rcv[100];
unsigned int msg_length = 0;
unsigned int loop = 0;

unsigned long int n_position = 0;
unsigned long int n_velocity = 0;
unsigned long int n_acceleration = 0;
unsigned int n_filter_kp = 0;

unsigned int count = 0;

void TOIISR() {

        TFLG2 &= 0x80;                                  //clearing TOF to enable further interrupts

        if (count % 32 == 0)
                if (PORTA & 0x10)
                        PORTA &= 0xEF;
                else
                        PORTA |= 0x10;

        count++;
}

void main() {

        int x = 0;
        setbaud(BAUD19200);

        TMSK2 |= 0x80;                          //locally enable TOI
```

```
	asm("CLI");

	PORTB = 0xFE;				//set all control lines high and select motor #1
	run_HDW_RSTSEQ();

	while(1) {
		msg_length = 0;
		x = 0;
		n_position = 0;
		n_velocity = 0;
		n_acceleration = 0;
		n_filter_kp = 0;

		DDRC = 0xFF;

		input = getchar();
	while (input != 10 && input != 13) {			//get serial comm msg
	msg_rcv[msg_length] = input;
	msg_length++;
	input = getchar();
	}

putline("Received CMD: ", 14);					//echo/confirm serial comm msg
while (x < msg_length) {
   putchar(msg_rcv[x]);
   x++;
}
putchar(10);					//line feed delimiter

		if (msg_rcv[0]=='c') {
			cur_Motor = (msg_rcv[1]-'0');
		}
		else if (msg_rcv[0] == 'r') {
			putline("RESETTING ALL MOTOR LM629's ...", 30);
			putchar(10);
			if (run_HDW_RSTSEQ()) {
				putline("ALL MOTORS RST SUCCESSFUL", 25);
				putchar(10);
			}
			else {
				putline("ALL MOTORS RST FAILED", 21);
				putchar(10);
			}
			continue;
		}

		loop = 3;

		if (msg_rcv[loop] == 'p') {
			loop++;
			while(isdigit(msg_rcv[loop])) {
				n_position = n_position*10 + (msg_rcv[loop]-'0');
				loop++;
			}
			loop++;
```

```
		if (msg_rcv[loop] == 'v') {
			loop++;
			while(isdigit(msg_rcv[loop])) {
				n_velocity = n_velocity*10 + (msg_rcv[loop]-'0');
				loop++;
			}
			loop++;
		}

		if (msg_rcv[loop] == 'a') {
			loop++;
			while(isdigit(msg_rcv[loop])) {
				n_acceleration = n_acceleration*10 + (msg_rcv[loop]-'0');
				loop++;
			}
			loop++;
		}
		LOAD_TRAJ(n_position, n_velocity, n_acceleration);
}

else if (msg_rcv[loop] == 'r' && msg_rcv[loop+1] == 'e') {
		putline("RESETTING CURRENT MOTOR LM629 #: ", 33);
		putchar(cur_Motor + '0');
		putchar(10);
		if (run_SFTW_RSTSEQ()) {
			putline("MOTOR RST SUCCESSFUL", 25);
			putchar(10);
		}
		else {
			putline("MOTOR RST FAILED", 21);
			putchar(10);
		}
}

else if (msg_rcv[loop] == 'l' && msg_rcv[loop+1] == 'f') {
		loop += 2;
		while(isdigit(msg_rcv[loop])) {
			n_filter_kp = n_filter_kp*10 + (msg_rcv[loop]-'0');
			loop++;
		}
		loop++;
		LOAD_FILTER(n_filter_kp);
}

else if (msg_rcv[loop] == 'r' && msg_rcv[loop+1] == 's') {
		RD_SIGS();
}

else if (msg_rcv[loop] == 'r' && msg_rcv[loop+1] == 'p') {
		RD_RPOS();
}

else if (msg_rcv[loop] == 'd' && msg_rcv[loop+1] == 'p') {
		RD_DPOS();
}
```

```
                else if (msg_rcv[loop] == 'r' && msg_rcv[loop+1] == 'v') {
                        RD_RVEL();
                }

                else if (msg_rcv[loop] == 'd' && msg_rcv[loop+1] == 'v') {
                        RD_DVEL();
                }

                else if (msg_rcv[loop] == 's' && msg_rcv[loop+1] == 'a') {
                        run_SampleTraj();
                }

                else if (msg_rcv[loop] == 's' && msg_rcv[loop+1] == 'm') {
                        STT_MOTOR();
                }

                for (x=0; x < msg_length; x++)      //clear msg_rcv array so prev msg doesn't interfere
                        msg_rcv[x] = ' ' ;
        }
}

void putline(char* msg_str, int length) {
        int i = 0;

        for (i = 0; i < length; i++)
                putchar(msg_str[i]);
}

void putNum(unsigned int number) {                  //puts numbers in character format to serial

        putchar(number/10000 + '0');
        number %= 10000;
        putchar(number/1000 + '0');
        number %= 1000;
        putchar(number/100 + '0');
        number %= 100;
        putchar(number/10 + '0');
        number %= 10;
        putchar(number + '0');
}

void wait_BUSY() {                                  //loops until LM629 NOT busy

        STATUS = 0x01;
        DDRC = 0x00;

        while (STATUS & 0x01) {
                PORTB &= 0xDF;                              //PB5 = PS low
                PORTB &= 0x7F;                              //PB7 = RD low

                STATUS = PORTC;                             //read STATUS byte

                PORTB |= 0x80;                          //PB7, RD high
                PORTB |= 0x20;                          //PB5, PS high
        }
```

```
	DDRC = 0xFF;				//give data line control back to HC11
}

void chk_STATUS() {			//immediate update of STATUS value from LM629

	DDRC = 0x00;

	chng_Motor(cur_Motor);		//CS low
	PORTB &= 0xDF;				//PB5 = PS low
	PORTB &= 0x7F;				//PB7 = RD low

	STATUS = PORTC;

	PORTB |= 0x80;				//PB7, RD high
	PORTB |= 0x20;				//PB5, PS high
	chng_Motor(0);				//CS high

	DDRC = 0xFF;				//give data line control back to HC11
}

void write_CMD(int CMD) {

	DDRC = 0xFF;
	PORTC = CMD;

	chng_Motor(cur_Motor);		//CS low
	PORTB &= 0xDF;				//PB5 = PS low
	PORTB &= 0xEF;				//PB4 = WR low
	PORTB |= 0x10;				//PB4 = WR high
	PORTB |= 0x20;				//PB5 = PS high
	wait_BUSY();
	chng_Motor(0);				//CS high
}

void write_DATA(unsigned int DATA_MS, unsigned int DATA_LS) {

	DDRC = 0xFF;

	chng_Motor(cur_Motor);		//CS low
	PORTB |= 0x20;				//PB5, PS high
	PORTC = DATA_MS;
	PORTB &= 0xEF;				//PB4 = WR low
	PORTB |= 0x10;				//PB4 = WR high

	PORTC = DATA_LS;
	PORTB &= 0xEF;				//PB4 = WR low
	PORTB |= 0x10;				//PB4 = WR high
	wait_BUSY();
	chng_Motor(0);				//CS high
}

unsigned int read_DATA() {
	unsigned int DATA = 0;

	DDRC = 0x00;
```

```
	chng_Motor(cur_Motor);		//CS low
	PORTB |= 0x20;				//PB5, PS high
	PORTB &= 0x7F;					//PB7 = RD low
	PORTB |= 0x80;				//PB7, RD high
	DATA = PORTC;
	DATA = DATA << 8;

	PORTB &= 0x7F;					//PB7 = RD low
	PORTB |= 0x80;				//PB7, RD high
	DATA += PORTC;
	wait_BUSY();
	chng_Motor(0);				//CS high

	DDRC = 0xFF;

	return DATA;
}

void LOAD_FILTER(unsigned int Kp) {

	putline("*** LOADING NEW FILTER COEFF: ", 30);
	putNum(Kp);
	putchar(10);

//	wait_BUSY();
	write_CMD(0x1E);		//LFIL; load filter coeff cmd

	write_DATA(0x00, 0x08);//derivative sample rate, load Kp filter coeff

	write_DATA(0x00, Kp);

//	putline("UPDATING FILTER DATA (UDF)", 26);

	write_CMD(0x04);			//UDF

	putline("FILTER UPDATE COMPLETE!", 23);
	putchar(10);
}

void LOAD_TRAJ(unsigned long int position, unsigned long int velocity, unsigned long int acceleration) {
	unsigned int vel_MSW = 0;
	unsigned int vel_LSW = 0;
	unsigned int acc_MSW = 0;
	unsigned int acc_LSW = 0;
	unsigned int pos_MSW = 0;
	unsigned int pos_LSW = 0;

	acc_MSW = ((acceleration >> 16) & 0x0000FFFF);
	acc_LSW = (acceleration & 0x0000FFFF);

	vel_MSW = ((velocity >> 16) & 0x0000FFFF);
	vel_LSW = (velocity & 0x0000FFFF);

	pos_MSW  = ((position >> 16) & 0x0000FFFF);
	pos_LSW = (position & 0x0000FFFF);
```

```
        putline("*** LOADING NEW TRAJECTORY (format MS_WORD*2^16+LS_WORD): ", 56);
        putline("pos_MSW: ", 9);
        putNum(pos_MSW);
        putline(" pos_LSW: ", 10);
        putNum(pos_LSW);
        putline(" vel_MSW: ", 10);
        putNum(vel_MSW);
        putline(" vel_LSW: ", 10);
        putNum(vel_LSW);
        putline(" accel_MSW: ", 12);
        putNum(acc_MSW);
        putline(" accel_LSW: ", 12);
        putNum(acc_LSW);
        putchar(10);

//      wait_BUSY();
        write_CMD(0x1F);          //LTRJ; load trajectory cmd

        write_DATA(0x00, 0x2B);

        write_DATA((acc_MSW >> 8) & 0x00FF, acc_MSW & 0xFF);

        write_DATA((acc_LSW >> 8) & 0x00FF, acc_LSW & 0xFF);

        write_DATA((vel_MSW >> 8) & 0x00FF, vel_MSW & 0xFF);

        write_DATA((vel_LSW >> 8) & 0x00FF, vel_LSW & 0xFF);

        write_DATA((pos_MSW >> 8) & 0x00FF, pos_MSW & 0xFF);

        write_DATA((pos_LSW >> 8) & 0x00FF, pos_LSW & 0xFF);

        putline("NEW TRAJECTORY DATA LOADED", 26);
        putchar(10);
}

void STT_MOTOR() {
        putline("EXECUTING CMD TO STT MOTOR: ", 28);
        putNum(cur_Motor);
        putchar(10);

//      wait_BUSY();
        write_CMD(0x01);                  //STT; start motor motion cmd
}

void run_SampleTraj() {
//========================== LOAD FILTER COEFFICIENTS ===============
//      wait_BUSY();
        write_CMD(0x1E);                                  //LFIL; load filter coefficients command

        write_DATA(0x00, 0x08);                   //begin load filter control word

        write_DATA(0x00, 0x30);                   //begin load kp word

        write_CMD(0x04);                                  //UDF; update filter command
```

```
//========================== LOAD TRAJECTORY DATA ===================
        write_CMD(0x1F);                                //LTRJ; load trajectory command

        write_DATA(0x00, 0x2A);                         //begin load trajectory control word

        write_DATA(0x00, 0x00);                 //begin load acc int word

        write_DATA(0x00, 0x02);                 //begin load acc frac word

        write_DATA(0x00, 0x01);                 //begin load vel int word

        write_DATA(0x34, 0x6E);                         //begin load vel fract word

        write_DATA(0x00, 0x00);                 //begin load position MS word

        write_DATA(0xAF, 0x40);                         //begin load position LS word

        write_CMD(0x01);                                //STT; Start motion command

        putline("Starting pre-programmed sample traj ...", 39);
        putchar(10);
//==============================
}

void HDW_RSET() {                               //READ STATUS BYTE

        DDRC = 0x00;                            //set to read input status byte

        PORTB &= 0xBF;                                  //PB6 = RST low
        TOC3 = TCNT + 3500;
        TFLG1 &= 0x20;
        while (!(TFLG1 & 0x20));//hold RST low for more than 8 clk cycles
        PORTB |= 0x40;                          //PB6 = RST high

        TFLG1 &= 0x20;
        TOC3 = TCNT + 7000;
        while (!(TFLG1 & 0x20));//wait for 1.5 ms after RESET
}

int run_HDW_RSTSEQ() {

        int rst_stat = 0;
        int motor = 1;
        int rst_failed = 0;

        HDW_RSET();

        for (motor = 1; motor <= 3; motor++) {
                cur_Motor = motor;
                chk_STATUS();
                if (STATUS == 0xC4 || STATUS == 0x84)
                        rst_stat = 1;
                else {
                        rst_stat = 0;
                        }
```

```
//                      wait_BUSY();
                        write_CMD(0x1D);                        //RSTI; reset interrupts command

                        write_DATA(0x00, 0x00);                 //reset all interrupts

                        chk_STATUS();
                        if ((STATUS == 0x80 || STATUS == 0xC0) && rst_stat == 1)
                                rst_stat = 1;
                        else {
                                rst_stat = 0;
                                }
                        if (rst_stat == 0)
                                rst_failed = 1;
        }
        return !(rst_failed);
}

int run_SFTW_RSTSEQ() {

        int rst_stat = 0;

//      wait_BUSY();
        write_CMD(0x00);

        TFLG1 &= 0x20;
        TOC3 = TCNT + 7000;
        while (!(TFLG1 & 0x20));//wait for 1.5 ms after RESET

        chk_STATUS();

        if (STATUS == 0xC4 || STATUS == 0x84)
                rst_stat = 1;
        else {
                rst_stat = 0;
                }

        write_CMD(0x1D);                        //RSTI; reset interrupts command

        write_DATA(0x00, 0x00);                 //reset all interrupts

        chk_STATUS();
        if ((STATUS == 0x80 || STATUS == 0xC0) && rst_stat == 1)
                rst_stat = 1;
        else {
                rst_stat = 0;
                }
        return rst_stat;
}

void RD_SIGS() {
        unsigned int signals = 0;

//      wait_BUSY();
        chk_STATUS();
        write_CMD(0x0C);                        //RDSIGS; load read signals reg command
```

```
	signals = read_DATA();

	putline("READ SIGNALS (MSB -> LSB): ", 27);
	putNum((signals >> 8) & 0x00FF);
	putchar(' ');
	putNum(signals & 0x00FF);
	putchar(10);
}

void RD_RPOS() {
	unsigned int signals_MSW = 0;
	unsigned int signals_LSW = 0;
	int i = 0;

//	wait_BUSY();

	write_CMD(0x0A);				//RDRP; load read real position reg command

	signals_MSW = read_DATA();

	signals_LSW = read_DATA();

	putline("READ REAL POS (MSW -> LSW): ", 28);
	for (i = 0; i < 16; i++)
		putchar(((signals_MSW >> (15 - i)) & 0x0001) + '0');

	putchar(' ');

	for (i = 0; i < 16; i++)
		putchar(((signals_LSW >> (15 - i)) & 0x0001) + '0');

	putline(" = ", 3);
	putNum(signals_MSW);
	putchar(' ');
	putNum(signals_LSW);
	putchar(10);
}

void RD_DPOS() {
	unsigned int signals_MSW = 0;
	unsigned int signals_LSW = 0;
	int i = 0;

//	wait_BUSY();

	write_CMD(0x08);				//RDRP; load read real position reg command

	signals_MSW = read_DATA();

	signals_LSW = read_DATA();

	putline("READ DESIRED POS (MSW -> LSW): ", 31);
	for (i = 0; i < 16; i++)
		putchar(((signals_MSW >> (15 - i)) & 0x0001) + '0');

	putchar(' ');
```

```
	for (i = 0; i < 16; i++)
		putchar(((signals_LSW >> (15 - i)) & 0x0001) + '0');

	putline(" = ", 3);
	putNum(signals_MSW);
	putchar(' ');
	putNum(signals_LSW);
	putchar(10);
}


void RD_RVEL() {
	unsigned int signals_MSW = 0;
	int i = 0;

//	wait_BUSY();

	write_CMD(0x0B);				//RDRP; load read real position reg command

	signals_MSW = read_DATA();

	putline("READ REAL VEL: ", 15);
	for (i = 0; i < 16; i++)
		putchar(((signals_MSW >> (15 - i)) & 0x0001) + '0');

	putline(" = ", 3);
	putNum(signals_MSW);
	putchar(10);
}

void RD_DVEL() {
	unsigned int signals_MSW = 0;
	unsigned int signals_LSW = 0;
	int i = 0;

//	wait_BUSY();

	write_CMD(0x07);				//RDRP; load read real position reg command

	signals_MSW = read_DATA();

	signals_LSW = read_DATA();

	putline("READ DESIRED VEL (MSW -> LSW): ", 31);
	for (i = 0; i < 16; i++)
		putchar(((signals_MSW >> (15 - i)) & 0x0001) + '0');

	putchar(' ');

	for (i = 0; i < 16; i++)
		putchar(((signals_LSW >> (15 - i)) & 0x0001) + '0');

	putline(" = ", 3);
	putNum(signals_MSW);
	putchar(' ');
```

```
        putNum(signals_LSW);
        putchar(10);
}

void chng_Motor(int m_num) {

        switch (m_num) {
                case 0:                                         //all LM629's off
                        PORTB |= 0x01;
                        PORTB |= 0x02;
                        PORTB |= 0x04;
                case 1:                                         //LM629 #1 on, PB0
                        PORTB &= 0xFE;
                        PORTB |= 0x02;
                        PORTB |= 0x04;
                        break;
                case 2:                                         //LM629 #2 on, PB1
                        PORTB |= 0x01;
                        PORTB &= 0xFD;
                        PORTB |= 0x04;
                        break;
                case 3:                                         //LM629 #3 on, PB2
                        PORTB |= 0x01;
                        PORTB |= 0x02;
                        PORTB &= 0xFB;
                        break;
                default:
                        break;
        }
}

#include "vectors.c"
```

APPENDIX B: Mobile Vestibular Platform – Command Terminal System Software

PLATCOMM1.M
(Programmed using Mathworks' Matlab's Graphical User Interface Development Environment)

```
function varargout = PlatComm1(varargin)
% PLATCOMM1 M-file for PlatComm1.fig
%      PLATCOMM1, by itself, creates a new PLATCOMM1 or raises the existing
%      singleton*.
%
%      H = PLATCOMM1 returns the handle to a new PLATCOMM1 or the handle to
%      the existing singleton*.
%
%      PLATCOMM1('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PLATCOMM1.M with the given input arguments.
%
%      PLATCOMM1('Property','Value',...) creates a new PLATCOMM1 or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before PlatComm1_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
```

```
%      stop.  All inputs are passed to PlatComm1_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help PlatComm1

% Last Modified by GUIDE v2.5 19-Jul-2004 11:05:31

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @PlatComm1_OpeningFcn, ...
                   'gui_OutputFcn',  @PlatComm1_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before PlatComm1 is made visible.
function PlatComm1_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to PlatComm1 (see VARARGIN)

% Choose default command line output for PlatComm1
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes PlatComm1 wait for user response (see UIRESUME)
% uiwait(handles.figure1);
global Serial_COM;
global edit_SysStatTxt;
edit_SysStatTxt = 'System Status';
Serial_COM = 'COM1';

set(handles.radio_cmd, 'Value', 1);
off = [handles.radio_platform];
mutual_exclude(off);
```

```
update_radioEnables(2, handles);

% --- Outputs from this function are returned to the command line.
function varargout = PlatComm1_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% ********************************************** Unused/Unmodified Code
**********************
% --- Executes during object creation, after setting all properties.
function edit_cmd_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cmd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit_cmd_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cmd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cmd as text
%        str2double(get(hObject,'String')) returns contents of edit_cmd as a double


% --- Executes during object creation, after setting all properties.
function edit_filter_kp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_filter_kp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_filter_kp_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit_filter_kp (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_filter_kp as text
%        str2double(get(hObject,'String')) returns contents of edit_filter_kp as a double


% --- Executes during object creation, after setting all properties.
function edit_coord_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_coord_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_coord_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_coord_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_coord_x as text
%        str2double(get(hObject,'String')) returns contents of edit_coord_x as a double


% --- Executes during object creation, after setting all properties.
function edit_coord_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_coord_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_coord_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_coord_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_coord_y as text
%        str2double(get(hObject,'String')) returns contents of edit_coord_y as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit_coord_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_coord_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_coord_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_coord_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_coord_z as text
%        str2double(get(hObject,'String')) returns contents of edit_coord_z as a double


% --- Executes during object creation, after setting all properties.
function edit_num_rcvmsgs_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_num_rcvmsgs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_num_rcvmsgs_Callback(hObject, eventdata, handles)
% hObject    handle to edit_num_rcvmsgs (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_num_rcvmsgs as text
%        str2double(get(hObject,'String')) returns contents of edit_num_rcvmsgs as a double


% --- Executes during object creation, after setting all properties.
function edit_cable1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable1_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable1 as text
%        str2double(get(hObject,'String')) returns contents of edit_cable1 as a double


% --- Executes during object creation, after setting all properties.
function edit_cable2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable2_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable2 as text
%        str2double(get(hObject,'String')) returns contents of edit_cable2 as a double


% --- Executes during object creation, after setting all properties.
function edit_cable3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
```

```
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable3_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable3 as text
%        str2double(get(hObject,'String')) returns contents of edit_cable3 as a double


% --- Executes during object creation, after setting all properties.
function edit_cable1_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable1_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable1_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable1_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable1_x as text
%        str2double(get(hObject,'String')) returns contents of edit_cable1_x as a double


% --- Executes during object creation, after setting all properties.
function edit_cable1_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable1_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit_cable1_y_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to edit_cable1_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable1_y as text
%        str2double(get(hObject,'String')) returns contents of edit_cable1_y as a double


% --- Executes during object creation, after setting all properties.
function edit_cable2_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable2_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable2_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable2_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable2_x as text
%        str2double(get(hObject,'String')) returns contents of edit_cable2_x as a double


% --- Executes during object creation, after setting all properties.
function edit_cable2_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable2_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable2_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable2_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable2_y as text
%        str2double(get(hObject,'String')) returns contents of edit_cable2_y as a double
```

```
% --- Executes during object creation, after setting all properties.
function edit_cable2_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable2_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable2_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable2_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable2_z as text
%        str2double(get(hObject,'String')) returns contents of edit_cable2_z as a double


% --- Executes during object creation, after setting all properties.
function edit_cable3_x_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable3_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable3_x_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable3_x (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable3_x as text
%        str2double(get(hObject,'String')) returns contents of edit_cable3_x as a double


% --- Executes during object creation, after setting all properties.
function edit_cable3_y_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable3_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable3_y_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable3_y (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable3_y as text
%        str2double(get(hObject,'String')) returns contents of edit_cable3_y as a double


% --- Executes during object creation, after setting all properties.
function edit_cable3_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable3_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable3_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable3_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable3_z as text
%        str2double(get(hObject,'String')) returns contents of edit_cable3_z as a double


% --- Executes during object creation, after setting all properties.
function edit_cable1_z_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_cable1_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
```

```
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_cable1_z_Callback(hObject, eventdata, handles)
% hObject    handle to edit_cable1_z (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_cable1_z as text
%        str2double(get(hObject,'String')) returns contents of edit_cable1_z as a double


% --- Executes during object creation, after setting all properties.
function edit_move_duration_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_move_duration (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit_move_duration_Callback(hObject, eventdata, handles)
% hObject    handle to edit_move_duration (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit_move_duration as text
%        str2double(get(hObject,'String')) returns contents of edit_move_duration as a double


% --- Executes during object creation, after setting all properties.
function edit_systemstat_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit_systemstat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% ********************************** End Unused/Unomodified Code ******************
```

```
% --- Disables all appropriate radio buttons
function mutual_exclude(off)
set(off,'Value',0)

% --- Enables/Disables all uicontrols appropriately to what radio buttons are enabled
function update_radioEnables(radio_choice, handles)

if (radio_choice == 2)
   cmd_enable = 'on';
   plat_enable = 'off';
elseif (radio_choice == 1)
   cmd_enable = 'off';
   plat_enable = 'on';
end

set(handles.push_transfer, 'Enable', cmd_enable);
set(handles.edit_num_rcvmsgs, 'Enable', cmd_enable);
set(handles.edit_cmd, 'Enable', cmd_enable);
set(handles.edit_filter_kp, 'Enable', plat_enable);
set(handles.edit_coord_x, 'Enable', plat_enable);
set(handles.edit_coord_y, 'Enable', plat_enable);
set(handles.edit_coord_z, 'Enable', plat_enable);
set(handles.push_rst_sys, 'Enable', plat_enable);
set(handles.push_update_pos, 'Enable', plat_enable);
set(handles.edit_cable1, 'Enable', plat_enable);
set(handles.edit_cable2, 'Enable', plat_enable);
set(handles.edit_cable3, 'Enable', plat_enable);
set(handles.edit_cable1_x, 'Enable', plat_enable);
set(handles.edit_cable1_y, 'Enable', plat_enable);
set(handles.edit_cable1_z, 'Enable', plat_enable);
set(handles.edit_cable2_x, 'Enable', plat_enable);
set(handles.edit_cable2_y, 'Enable', plat_enable);
set(handles.edit_cable2_z, 'Enable', plat_enable);
set(handles.edit_cable3_x, 'Enable', plat_enable);
set(handles.edit_cable3_y, 'Enable', plat_enable);
set(handles.edit_cable3_z, 'Enable', plat_enable);
set(handles.edit_systemstat, 'Enable', plat_enable);

% --- Executes on button press in radio_cmd.
function radio_cmd_Callback(hObject, eventdata, handles)
% hObject    handle to radio_cmd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
off = [handles.radio_platform];
mutual_exclude(off);

set(hObject, 'Value', 1);

update_radioEnables(2, handles);            %enable radio_platform related uicontrols
% Hint: get(hObject,'Value') returns toggle state of radio_cmd


% --- Executes on button press in radio_platform.
function radio_platform_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to radio_platform (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
off = [handles.radio_cmd];
mutual_exclude(off);

set(hObject, 'Value', 1);

update_radioEnables(1, handles);         %enable radio_cmd related uicontrols
% Hint: get(hObject,'Value') returns toggle state of radio_platform


% --- Executes on button press in push_transfer.
function push_transfer_Callback(hObject, eventdata, handles)
% hObject    handle to push_transfer (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%--------------------------BEGIN CODE TO BE DELETED---------------------------------
if (strcmp(get(handles.edit_cmd, 'String'), 'run test protocol'))
   send_MSG('r', 3, handles);
   send_MSG('c1 sa', 2, handles);
   set(handles.edit_systemstat, 'Enable', 'on');
   count = 0;
   tempString = '  ';
   while (~strcmp(get(handles.edit_cmd, 'String'), 'stop'))
      count = count + 1;
      send_MSG('c1 rs', 2, handles);
      for loop = 1:4
         inputString = get(handles.text_serialstat, 'String');
         tempString = [tempString, num2str(count), 13, inputString(loop, :), 13];
      end
      send_MSG('c1 rp', 2, handles);
      for loop = 1:5
         inputString = get(handles.text_serialstat, 'String');
         tempString = [tempString, inputString(loop, :), 13];
      end
      send_MSG('c1 dp', 2, handles);
      for loop = 1:5
         inputString = get(handles.text_serialstat, 'String');
         tempString = [tempString, inputString(loop, :), 13];
      end
      inputString = get(handles.edit_systemstat, 'String');
      set(handles.edit_systemstat, 'String', [inputString, 10, tempString]);
   end
   while (~strcmp(get(handles.edit_cmd, 'String'), 'done'))
   end
   set(handles.edit_systemstat, 'Enable', 'off');
else
   %--------------------------END CODE TO BE DELETED---------------------------------
   send_MSG(get(handles.edit_cmd, 'String'), str2num(get(handles.edit_num_rcvmsgs, 'String')), handles);
end


% --- Executes on button press in radio_com1.
function radio_com1_Callback(hObject, eventdata, handles)
```

```
% hObject    handle to radio_com1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
off = [handles.radio_com2];
mutual_exclude(off);
set(hObject, 'Value', 1);
Serial_COM = 'COM1';
% Hint: get(hObject,'Value') returns toggle state of radio_com1


% --- Executes on button press in radio_com2.
function radio_com2_Callback(hObject, eventdata, handles)
% hObject    handle to radio_com2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
off = [handles.radio_com1];
mutual_exclude(off);
set(hObject, 'Value', 1);
Serial_COM = 'COM2';
% Hint: get(hObject,'Value') returns toggle state of radio_com2


% --- Executes on button press in push_rst_sys.
function push_rst_sys_Callback(hObject, eventdata, handles)
% hObject    handle to push_rst_sys (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global cable1_length;
global cable2_length;
global cable3_length;
global anchor1_pos;
global anchor2_pos;
global anchor3_pos;
global cur_plat_pos;

cable1_length = str2num(get(handles.edit_cable1, 'String'));
cable2_length = str2num(get(handles.edit_cable2, 'String'));
cable3_length = str2num(get(handles.edit_cable3, 'String'));

anchor1_pos = [str2num(get(handles.edit_cable1_x, 'String')) str2num(get(handles.edit_cable1_y, 'String'))
str2num(get(handles.edit_cable3, 'String'))];
anchor2_pos = [str2num(get(handles.edit_cable2_x, 'String')) str2num(get(handles.edit_cable2_y, 'String'))
str2num(get(handles.edit_cable2_z, 'String'))];
anchor3_pos = [str2num(get(handles.edit_cable3_x, 'String')) str2num(get(handles.edit_cable3_y, 'String'))
str2num(get(handles.edit_cable3_z, 'String'))];
cur_plat_pos = [0 0 0];
set(handles.statictxt_cur_coord, 'String', num2str(cur_plat_pos));

send_MSG('r', 3, handles);       %RST all LM629's to initialize position registers -> cur pos = home pos
set(handles.text12, 'String', 'Current Cable Lengths: ');


% --- Executes on button press in push_update_pos.
function push_update_pos_Callback(hObject, eventdata, handles)
% hObject    handle to push_update_pos (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
global cur_plat_pos;
global edit_SysStatTxt;

filter_coeff_kp = str2num(get(handles.edit_filter_kp, 'String'));
move_duration = str2num(get(handles.edit_move_duration, 'String'));
next_plat_pos = [str2num(get(handles.edit_coord_x, 'String')) str2num(get(handles.edit_coord_y, 'String'))
str2num(get(handles.edit_coord_z, 'String'))];
d_plat_pos = next_plat_pos - cur_plat_pos;

d_cable1_length = num2str(sqrt((anchor1_pos(1)-d_plat_pos(1)-cur_plat_pos(1))^2 + (anchor1_pos(2)-
d_plat_pos(2)-cur_plat_pos(2)+PTFM_L/4)^2 + (anchor1_pos(3)-d_plat_pos(3)-cur_plat_pos(3))^2) -
sqrt((anchor1_pos(1)-cur_plat_pos(1))^2 + (anchor1_pos(2)-cur_plat_pos(2)+PTFM_L/4)^2 + (anchor1_pos(3)-
cur_plat_pos(z))^2));
d_cable2_length = num2str(sqrt((anchor2_pos(1)-d_plat_pos(1)-cur_plat_pos(1)-PTFM_L/(8*sqrt(3)))^2 +
(anchor2_pos(2)-d_plat_pos(2)-cur_plat_pos(2)-PTFM_L/8)^2 + (anchor2_pos(3)-d_plat_pos(3)-
cur_plat_pos(3))^2) - sqrt((anchor2_pos(1)-cur_plat_pos(1)-PTFM_L/(8*sqrt(3)))^2 + (anchor2_pos(2)-
cur_plat_pos(2)-PTFM_L/8)^2 + (anchor2_pos(3)-cur_plat_pos(3))^2));
d_cable3_length = num2str(sqrt((anchor3_pos(1)-d_plat_pos(1)-cur_plat_pos(1)-PTFM_L/(8*sqrt(3)))^2 +
(anchor3_pos(2)-d_plat_pos(2)-cur_plat_pos(2)+PTFM_L/8)^2 + (anchor3_pos(3)-d_plat_pos(3)-
cur_plat_pos(3))^2) - sqrt((anchor3_pos(1)-cur_plat_pos(1)-PTFM_L/(8*sqrt(3)))^2 + (anchor3_pos(2)-
cur_plat_pos(2)+PTFM_L/8)^2 + (anchor3_pos(3)-cur_plat_pos(3))^2));

%convert from inches to cnts
d_cable1_length = d_cable1_length*925/1.884;
d_cable2_length = d_cable2_length*925/1.884;
d_cable3_length = d_cable3_length*925/1.884;

d_cable1_vel = num2str((d_cable1_length / move_duration / (1024.0 / 1000.0))*65536);
d_cable2_vel = num2str((d_cable2_length / move_duration / (1024.0 / 1000.0))*65536);
d_cable3_vel = num2str((d_cable3_length / move_duration / (1024.0 / 1000.0))*65536);

d_cable1_accel = num2str(d_cable1_vel/(1500/1000*1024));          %accelerate to max velocity over 1.5 s = 1500
ms = 1536 samples
d_cable2_accel = num2str(d_cable2_vel/(1500/1000*1024));
d_cable3_accel = num2str(d_cable3_vel/(1500/1000*1024));

msg_SysStat1 = ['Commencing Platform Movement: Pos Changes (cnts) -> Cable1: ', d_cable_length, ' Cable2: ',
d_cable2_length, ' Cable3: ', d_cable3_length, 10];
msg_SysStat2 = ['Approximate Vel(s) (cnts/smpl) -> Cable1: ', d_cable1_vel, ' Cable2: ', d_cable2_vel, ' Cable3: ',
d_cable3_vel, 10];
msg_SysStat3 = ['Accel(s) (cnts/smpl/smpl) -> Cable1: ', d_cable1_accel, ' Cable2: ', d_cable2_accel, ' Cable3: ',
d_cable3_accel, 10];
edit_SysStatTxt = [edit_SysStatTxt, 10, msg_SysStat1, msg_SysStat2, msg_SysStat3];

send_MSG(['c1 lf', filter_coeff_kp], 3, handles);
send_MSG(['c1 p', d_cable1_length, ' v', d_cable1_vel, ' a', d_cable1_accel], 3, handles);
send_MSG(['c2 lf', filter_coeff_kp], 3, handles);
send_MSG(['c2 p', d_cable2_length, ' v', d_cable2_vel, ' a', d_cable2_accel], 3, handles);
send_MSG(['c3 lf', filter_coeff_kp], 3, handles);
send_MSG(['c3 p', d_cable3_length, ' v', d_cable3_vel, ' a', d_cable3_accel], 3, handles);

send_MSG('c1 sm', 2, handles);
send_MSG('c2 sm', 2, handles);
send_MSG('c3 sm', 2, handles);
```

```
% --- Executes procedure to send msg to HC11
function send_MSG(msg, num_rcvmsgs, handles)
global Serial_COM;

static_SerTxt = 'Serial Comm Status';

if (strcmp(Serial_COM, 'COM1'))
   HC11 = serial('COM1');
elseif (strcmp(Serial_COM, 'COM2'))
   HC11 = serial('COM2');
end

HC11.BaudRate = 19200;
fopen(HC11);
fprintf(HC11, msg);

for cur_msg_num = 1:num_rcvmsgs
   HC11_from = fscanf(HC11);
   static_SerTxt = [static_SerTxt, 10, HC11_from];
   set(handles.text_serialstat, 'String', static_SerTxt);
end
delete(HC11);
clear HC11;


% --- Code to prevent user from altering system status reporting txt
function edit_systemstat_Callback(hObject, eventdata, handles)
% hObject    handle to edit_systemstat (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global edit_SysStatTxt

set(handles.edit_systemstat, 'String', editSysStatTxt);
% Hints: get(hObject,'String') returns contents of edit_systemstat as text
%        str2double(get(hObject,'String')) returns contents of edit_systemstat as a double
```