

ELEVATOR SYSTEM HAVING A FORCE-ESTIMATION OR
POSITION-SCHEDULED CURRENT COMMAND CONTROLLER

COPYRIGHT PROTECTION NOTICE

A portion of the disclosure of this patent application contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

TECHNICAL FIELD

This invention relates to a system having active suspension control, and more particularly relates to an elevator system having active horizontal suspension control for controlling horizontal movement of an elevator car with respect to guide rails in an elevator hoistway.

BACKGROUND OF THE INVENTION

As elevators are required to move faster, there is a greater need for improved lateral suspension to provide smooth, quiet rides for passengers. Current practice is to employ a passive suspension comprised of rollers mounted with springs and dampers to the corners of the elevator car. Advanced elevator suspension concepts are being proposed and developed that employ electromagnets as active force generators in the suspension to

counteract external forces acting on the elevator system. For example, an active roller guide (ARG) elevator system and an active magnetic guidance (AMG) elevator system are two such advanced suspension systems. Force generating mechanisms used for both the ARG and AMG elevator systems are electromagnet pairs that are required to generate controlled magnetic forces in the presence of relatively large air gap variations in a range of 2 to 10 millimeters.

Figure 3 shows a known flux and current feedback compensation controller for a magnet driver. In general, in Figure 3, the known flux and current feedback compensation controller feeds back magnetic flux and current to regulate electromagnet currents. The flux and current feedback system utilizes an outer force control feedback loop (flux feedback using Hall effect sensors), to regulate command current for the magnets in the presence of unknown gaps. (Note that Figure 3 is simplified by showing only one magnet).

Specifically, an elevator suspension controller (not shown in Figure 3) of the elevator system provides a force command signal F_d to the flux and current feedback compensation controller 50. The force command signal F_d is indicative of a desired force to move the elevator car with respect to guide rails in an elevator hoistway. A Hall effect sensor 52 measures a flux density FD , and provides a sensed flux density signal F_{FD} indicative of a

magnetic force exerted by a magnet 54 on a guide rail (not shown). A squaring circuit 56 squares the sensed flux density signal F_{FD} , and provides a squared sensed flux density signal F_{FD}^2 proportional to the sensed flux density signal F_{FD} . A first comparator 58 subtracts the squared sensed flux density signal F_{FD}^2 from the force command signal F_d , and provides a force error signal F_{FE} . A force feedback compensation processor 60 responds to the force error signal F_{FE} , and provides a desired force compensated current command signal I_c .

A current measurement circuit 62 measures a magnet driver current I_{MDC} , and provides a measured current signal I_{mcu} indicative of a measured current from a driver 64 to the magnet 54. A second comparator 66 subtracts the measured current signal I_{mcu} from the desired force compensated current command signal I_c , and provides a current error signal I_e . A current feedback compensation processor 68 responds to the current error signal I_e and provides a driver feedback current signal I_{DFC} to the driver 64. The driver 64 responds to the driver feedback current signal I_{DFC} , and provides the driver current signal I_{MDC} to the magnet 54.

The flux and current feedback compensation controller 50 for the AMG elevator system, as well as the ARG elevator system, both use the Hall effect sensor 52 as one of the primary feedback elements to control the magnetic force (See United States Patent Nos. 5,294,757,

5,304,751 and 5,308,938). The Hall effect sensor 52 is mounted in the magnetic field at a pole face of the magnet 54 and measures the local magnet flux density. In the AMG and ARG elevator systems, the force produced by the magnet 54 is roughly proportional to the square of the flux density.

However, there are four major disadvantages of using the Hall effect sensor 52 in the AMG elevator systems: First, the Hall effect sensor 52 must be located between the pole face of the magnet 54 and a reaction surface of the elevator guide rail (not shown). The location creates a severe reliability and durability concern, especially on the AMG elevator system in which the reaction surface is the elevator guide rail. The reaction surface can be moving at speeds greater than 10 meters per second (m/s) relative to the Hall effect sensor 52. Second, when using the Hall effect sensor 52 a spacer having thickness greater than the thickness of the Hall effect sensor 52 must be installed in the gap to protect the Hall effect sensor 52 from inadvertent impact. The thicker spacer reduces the useful travel of the suspension and/or requires the magnet 54 to produce magnetic force across a larger air gap. This is especially disadvantageous because the magnet force decreases as a function of the square of the air gap length. Third, the Hall effect sensor 52 requires additional wiring and connections, increasing the cost

and reducing the overall reliability of the suspension control system. Fourth, the existence of the Hall effect sensor 52 on the pole face of the magnet 54 limits the achievable force generation capability of the electromagnet. The Hall effect sensor 52 must be mounted in close proximity to the active electromagnet 54 - on the reaction plate (armature) in the case of the ARG elevator system, and directly on the magnet pole face in the case of the AMG elevator system. The limited temperature range of the Hall effect sensor 52 limits the allowable temperature rise of the magnet, and therefore limits the maximum magnet excitation. The ARG elevator systems have similar problems.

The disadvantages discussed above are difficulties in the ARG elevator systems, but prove to be serious limitations to the implementation of AMG elevator systems.

DISCLOSURE OF INVENTION

The invention provides a new and unique method for controlling forces produced by electromagnets in large gap applications without using a Hall effect sensor and its associated feedback loop.

In particular, the invention features an elevator system for controlling horizontal movement of an elevator car with respect to guide rails in an elevator hoistway, having a force-estimation or position-scheduled current

command controller and a magnet driver circuit. The force-estimation or position-scheduled current command controller responds to a force command signal, and further responds to a sensed gap signal, for providing a force-estimation or position-scheduled current command controller signal. The magnet driver circuit responds to the force-estimation or position-scheduled current command controller signal, for providing a magnet driver circuit signal to control the horizontal movement of the elevator car with respect to the guide rails in the elevator hoistway.

In one embodiment, the invention includes a force-estimation processor that responds to the sensed gap signal and a sensed magnet driver current signal, for providing a force-estimation processor signal indicative of an estimated force used to control the horizontal movement of the elevator car with respect to the guide rails in the elevator hoistway. The force-estimation processor may be implemented in hardware or software.

In another embodiment, the invention includes a position-scheduled current command controller that responds to the force command signal and the sensed gap signal, for providing a position-scheduled current command controller signal as a current command to the magnetic driver circuit. The position-scheduled current command controller may be implemented in hardware or software to control the movement of the elevator car with

respect to the guide rails in the elevator hoistway.

One significant advantage of the invention is that it eliminates the need for the Hall effect sensor.

Accordingly, the invention comprises the features of construction, combination of elements, and arrangement of parts which will be exemplified in the construction hereinafter set forth, and the scope of the invention will be indicated in the claims.

BRIEF DESCRIPTION OF THE DRAWING

For a fuller understanding of the nature of the invention, reference should be made to the following detailed descriptions taken in connection with the accompanying drawings, not in scale, in which:

Figure 1 is a schematic diagram of an elevator car that moves with respect to guide rails in an AMG elevator system.

Figure 2 is a block diagram of a circuit for controlling the movement of the elevator car in Figure 1 with respect to the guide rails.

Figure 3 is a block diagram of a known flux and current feedback compensation controller in a known magnet driver.

Figure 4 is a block diagram of a current command controller 200 of the present invention.

Figure 5a is a block diagram of one embodiment of a force-estimation current command controller 210/300 which

is one embodiment of the current command controller 200 in Figure 4.

Figure 5b is a block diagram of another embodiment of a force-estimation current command controller 210/300 which is one embodiment of the current command controller 200 in Figure 4.

Figure 6 is a block diagram of a microprocessor architecture for the force-estimation current command controller 210/300 of Figure 5.

Figure 7 is a diagram of a side-to-side configuration for the AMG elevator system.

Figure 8 is a diagram of a front-to-back configuration for the AMG elevator system.

Figure 9 shows how Figures 9A and 9B fit together to illustrate a hardware embodiment of a force estimation processor 310 in Figure 5.

Figure 10 is a block diagram of a position-scheduled current command controller 210/400 which is another embodiment of the current command controller 200 in Figure 4.

Figure 11 is a block diagram of the position-scheduled current command controller 210/400 shown in Figure 10 and the coordinated controller 100 shown in Figure 2.

Figure 12 shows a graph of side-to-side AMG magnet performance.

BEST MODE FOR CARRYING OUT THE INVENTION

An AMG Elevator System in General:

Figure 1 shows an elevator car 12 for an AMG elevator system shown and is described in a U.S. Patent application having Serial No. 08/292,660, filed August 18, 1994, and hereby incorporated by reference. The elevator car 12 has a car frame 13 with four guide heads 10, 20, 30, 40, which are shown in this example as magnetic guide heads. In general, the elevator car 12 includes a car frame 13 that operates on vertical guide rails (not shown) mounted at opposite sides of an elevator shaft or hoistway (not shown) of a building (not shown). The four guide heads 10, 20, 30, and 40 are mounted on the car frame 13 to guide the car along the rails and to reduce the vibrations transmitted to the car.

As shown, the rigid body motion of the elevator car 12 is kinematically defined in five degrees of freedom of a global coordination system (GCS) having X, Y, Z axes by side-to-side translation along the X axis, front-to-back translation along the Y axis, a pitch rotation about the X axis, a roll rotation about the Y axis, and a yaw rotation about the Z axis. The GCS may have its origin at the geometric (or mass) center of the elevator car 12 or at any other convenient origin. It should be realized that the choice of a coordinate system, though not arbitrary, need not be rectangular, although it is the

most logical and convenient. The side-to-side linear translation X_c is measured along the X axis in the GCS and a force F_x is defined along the X axis. The front-to-back linear translation Y_c is measured along the Y axis in the GCS, and a force F_y is defined along the Y axis. The pitch rotation θ_x is rotationally measured about the X axis in the GCS, and a moment M_x is defined about the X axis. The roll rotation θ_y is rotationally measured about the Y axis in the GCS, and a moment M_y is defined about the Y axis. The yaw rotation θ_z is measured about the Z axis in the GCS, and a moment M_z is defined about the Z axis. Each of the three arrows shown in Figure 2 indicates a positive moment about the respective axes. (Note that for the purpose of this discussion the measurement and motion of the elevator car 12 are not controlled by the AMG system with respect to translations along the Z axis.)

In addition, each guide head 10, 20, 30, 40 has a respective local coordinate system LCS_{10} , LCS_{20} , LCS_{30} , LCS_{40} , having x_i , y_i , z_i axes. For example, the guide head 10 has a local coordinate system LCS_{10} having an x_1 axis and a y_1 axis with forces F_{x1} and F_{y1} respectively defined along these axes, as shown. The guide head 20 has a local coordinate system LCS_{20} having an x_2 axis and a y_2 axis with forces F_{x2} and F_{y2} respectively defined along these axes, as shown. The guide head 30 has a local coordinate system LCS_{30} having an x_3 axis and a y_3 axis

with forces F_{x1} and F_{y1} , respectively defined along these axes, as shown. The guide head 40 has a local coordinate system LCS_{40} having an x_4 axis and a y_4 axis with forces F_{x4} and F_{y4} , respectively defined along these axes, as shown.

For each of the four guide heads 10, 20, 30, 40, its three respective electromagnets produce forces F_{x1} , F_{y1} , F_{z1} , F_{x2} , F_{y2} , F_{z2} , F_{x3} , F_{y3} , F_{z3} and F_{x4} , F_{y4} , F_{z4} along the respective local x_i and y_i axes. It is assumed that the local forces along the x_i and y_i axes act through the origin of its respective local coordinate system LCS_i . It should be realized that one could easily account for any offset in the local z_i axis between the local forces due to magnet positioning by adding additional length parameters in this kinematic characterization.

The local coordinate systems LCS_{10} , LCS_{20} , LCS_{30} , LCS_{40} are related to the GCS based on five lengths a , b , c , d and e , as shown in Figure 1. The lengths a and b define the lever arms for the pitch rotation θ_x about the X axis and the roll rotation θ_y about the Y axis. The lengths c , d and e define the lever arms for the yaw rotation θ_z about the Z axis. For the typical case, one assumes $a=b$, $d=e$ and $c=0$.

In one embodiment described in U.S. Serial No. 08/292,660, the position of the elevator car 12 is measured in three of the four local coordinate systems LCS_{10} , LCS_{20} , LCS_{30} , and coordinating local forces F_{x1} , F_{x2} , F_{y1} , F_{y2} , F_{y3} are applied in the same three local coordinate

systems LCS_{10} , LCS_{20} , LCS_{30} . The measurements are used to determine the deviation of the elevator car 12 from a desired position in the GCS, and the forces necessary to move the elevator car 12 back to the desired position in the GCS.

Figure 2 shows an example of a coordinated controller 100 described in U.S. Serial No. 08/292,660 for controlling the movement of the elevator car 12 shown in Figure 1 in the AMG elevator system. In Figure 2, the coordinated controller 100 includes position sensors 110 and acceleration sensors 120, a position and acceleration feedback controller 130, a high voltage power supply 140, magnet drivers 150, and electromagnets 160. The position sensors 110 are known in the art, and an example thereof is shown and described in U.S. Patent No. 5,294,757, hereby incorporated by reference. The position and acceleration feedback controller 130 keeps the car frame 13 centered between the guide rails and minimizes the lateral vibrations, and is shown and described in U.S. Serial No. 08/292,660. Other position and acceleration feedback controllers are also shown and described in U.S. Patent Nos. 5,294,757; 5,304,751 and 5,308,938, which are also hereby incorporated by reference.

As discussed above in detail, Figure 3 shows a known flux and current feedback compensation controller for the magnet drivers 150 shown in Figure 2.

The Force-Estimation or Position-Scheduled
Current Command Controller

Figure 4 shows a block diagram of the present invention in its broadest context. As shown, the present invention provides a current command controller 200 to regulate currents to an electromagnet 160 that serves as a force generator in the AMG elevator system.

As shown in Figure 4, the current command controller 200 has a force-estimation or position-scheduled current command controller 210 and a magnet driver circuit 220. The force-estimation or position-scheduled current command controller 210 responds to a force command signal F_d , and further responds to a sensed gap signal G_m , for providing a current command signal I_c . As described below, the current command signal I_c takes the form of either a force-estimation current command controller signal or a position-scheduled current command controller signal, and is indicative of a current command to the magnet driver circuit for driving electromagnets to control the elevator car 12 (Figure 1) with respect to the guide rail 205 in the elevator hoistway (not shown). The position and acceleration feedback controller 130 in Figure 2 provides the force command signal F_d , which is indicative of a horizontal force to move the elevator car (Figure 1) with respect to the guide rail 205 in the elevator hoistway (not shown). One of the position sensors 110 in Figure 2 provides the measured gap signal

G_m of Figure 4, which is indicative of an actual gap of the elevator car 12 (Figure 1) with respect to the guide rail 205 in the elevator hoistway (not shown).

The magnet driver circuit 220 responds to the current command signal I_c , for providing a magnet driver circuit signal to the magnet 160 in Figure 4 to control movement of the elevator car 12 with respect to the guide rails (not shown) in the elevator hoistway (not shown).

In effect, the present invention uniquely provides two different control systems to regulate the bi-directional force generated by the electromagnets 160 operating over relatively large air gaps without the need for Hall effect sensors. Figures 5(a), 5(b) and 6-9 show embodiments in which the current command controller 200 has a magnetic-force estimation current command controller 210/300 to regulate currents to electromagnets 160 that serve as force generators. Figures 6-8 and 10-11 show an embodiment in which the current command controller 200 has a position-scheduled current command controller 210/400. These embodiments use an idealized relationship between the magnetic force produced given the air gap between the magnet and a reaction surface (e.g., the elevator guide rail) and command currents for the magnets. The force estimation approach may use existing current and position sensing elements in the ARG and AMG elevator systems. However, it should be realized that the present invention is also applicable to elevator

systems having a plurality of active guides of any type, including active roller guides, active slide guides, etc. Moreover, the scope of the invention is also not intended to be limited to only current commands, because embodiments are envisioned in which voltage commands may be used for controlling the elevator car 12 (Figure 1) with respect to the guide rails 205 in the elevator hoistway (not shown).

Figure 5: Force-Estimation
Current Command Controller 300

Figure 5a shows an embodiment of the invention in which the current command controller 200 has a force-estimation current command controller 210/300 that includes a force-estimation processor 310, a force comparator 320 and a force feedback compensation processor 330.

The force-estimation processor 310 responds to the sensed gap signal G_m from a gap sensor 110, and further responds to a sensed or measured magnet driver current signal I_{mcs} from a magnetic current sensor 228, for providing a force-estimation processor signal indicative of an estimation of the horizontal force to control the elevator car 12 with respect to the guide rails in the elevator hoistway. The force comparator 320 is known in the art and subtracts the force-estimation processor signal F_{cs} from the force command signal F_c provided by the position and acceleration feedback controller 130 in

Figure 2, for providing a force error signal F_e indicative of comparisons of the force command signal F_d and the force-estimation processor signal F_{est} to control the elevator car 12 with respect to the guide rails in the elevator hoistway (not shown). The force feedback compensation processor 330 responds to the force error signal F_e , for providing a force feedback compensation processor signal as a current command I_c to the magnet driver circuit 220.

The force estimation processor 310 may be implemented in hardware or software. When implemented in software, a typical microprocessor architecture is used, as generally shown in Figure 6. For example, the force estimation processor 310 may include a microprocessor (CPU) 340, a Random Access Memory (RAM) 342, a Read Only Memory (ROM) 344, an input/output controller (I/O) 346, and an address, data and control bus (BUS) 348 for connecting the microprocessor (CPU) 340, the Random Access Memory (RAM) 342, the Read Only Memory (ROM) 344 and the input/output controller (I/O) 346. One advantage of implementing the force-estimation processor 310 in software is that it can easily be updated and adapted over time by simply adjusting the software program. An implementation of the force-estimation processor 310 using hardware is discussed below with respect to Figure 9.

The force feedback compensation processor 330 may

similarly be implemented in software with the same or a separate microprocessor such as shown in Figure 6. The scope of the invention is not intended to be limited to any particular embodiment of the force-estimator processor 310 or the force feedback compensation processor 330, or whether it is implemented in hardware or software, or a combination thereof.

The magnet driver circuit 220 is known in the art of elevator systems and includes a current-error comparator 222, a current feedback compensation processor 224, a driver 226 and a magnet current sensor 228. The current-error comparator 222 subtracts the measured magnet driver current signal I_{meas} indicative of the measured magnet driver current applied to electromagnets 160 from the current command signal I_c , for providing a current error signal I_e . The current feedback compensation processor 224 responds to the current error signal I_e , for providing a current feedback compensation processor signal I_{FC} . The driver 226 responds to the current feedback compensation processor signal I_{FC} , for providing a driver signal to the magnet 160 that provides a magnet force to control movement of the elevator car 12 (Figure 1) with respect to the guide rails in the elevator hoistway. The magnetic current sensor 228 responds to the magnet driver current, for providing the sensed or measured magnet driver current signal I_{meas} to the current-error comparator 222. As shown, the "not used" in Figure 5 indicates that

the force-estimation current command controller 210/300 replaces the Hall effect sensor 52 and the squaring circuit 54 shown in Figure 3, which are no longer needed.

In operation, the force-estimation processor 310 takes the sensed gap signal G_m and the measured magnet driver current signal I_{mcs} , and computes a force-estimation processor signal using a derived Equation 1.0, as follows:

$$\text{Force} = K_{mag} * (I_c / G_{mag})^2 \quad (\text{Eq. 1}),$$

where the Force represents a force scaled in Newtons, K_{mag} represents a proportionality constant (100 for the smaller Front-To-Back Axis AMG Magnet and 425 for the Larger Side-To-Side Axis AMG Magnet) for a given magnet configuration, as discussed below with respect to Figure 12, I_c represents the current command signal scaled in Amperes, and G_{mag} represents an actual magnet gap signal scaled in millimeters between a pole face of a magnet and a reaction surface of a guide rail, which is determined from the sensed gap signal G_m .

In the embodiment shown in Figure 5b, the current command signal I_c is fed to the force-estimation processor 310 instead of the sensed or measured magnet driver current signal I_{mcs} from a magnetic current sensor 228, if the bandwidth is high enough so the sensed or measured magnet driver current signal I_{mcs} follows the current

command signal I_c . In operation, the force-estimation processor 310 responds to a feedback current command signal I_c and the sensed gap signal G_m , for providing the force-estimation processor signal indicative of the estimation of the horizontal force to control the elevator car 12 with respect to the guide rails in the elevator hoistway. Otherwise, the embodiment is similar in design to that shown and described with respect to Figure 5a above.

Figures 7 and 8: Gap Sensor Processing Algorithm:

Figures 7 and 8 show a typical physical relationship between the magnet, the reaction surface and the position sensor. As discussed above, in order to estimate the force produced by the AMG and ARG electromagnets, the air gap length and magnet current for each magnet must be known. The magnet current is information readily available from the magnet driver electronics. In implementing the drivers, current feedback may be used to improve the response of the current control. In the case of a system that does not use current feedback, the current must still be monitored to insure that it stays within safe levels, and is therefore available for the force estimation technique.

Side-To-Side AMG Actuator Gaps:

The actual magnetic gap signal G_{mag} includes the

right side magnet gap G_r and the left side gap G_l .

Figure 7 shows a physical system configuration for a Side-to-Side axis. Two position sensors 110a and 110b are examples of the position sensors 110 in Figure 2 and are used since the gap for both left and right side magnets must be known even as the Distance Between the Guiderails (DBG) changes. The offset between the magnet gap and each position sensor 110a and 110b must be physically measured. This is shown and designated in Figure 7 as L_{offset} for the left side magnet and R_{offset} for the right side magnet. The measured offsets L_{offset} and R_{offset} are a result of the mounting configuration of the magnets and the position sensor and do not change with system operation. The measured left gap is G_{mls} .

The left side gap G_l is determined using Equation 2, as shown below:

$$G_l = G_{mls} - L_{offset} \quad (\text{Eq. 2}).$$

The right side magnet gap G_r is determined using Equation 3, as shown below:

$$G_r = G_{mrs} - R_{offset} \quad (\text{Eq. 3}).$$

Note for the side-to-side configuration, the gap processing algorithm is identical for either side.

Front-To-Back AMG Actuator Gaps:

The actual magnetic gap signal G_{mag} includes the front magnet gap G_f and the back gap G_b , which must be determined.

Figure 8 shows a physical system configuration for a front-to-back axis. Note that there is only one position sensor 110c in the front-to-back axis and two magnets. Force estimation using Equation 1 requires gap information for each magnet. The guide rail may be a conventional 3/4 T rail which has a highly uniform 3/4 inch (1.9 centimeters) width, although the scope of the invention is not intended to be limited to any particular type of guide rail. The spacing of the front and back magnets is determined by the guide head hardware. Hence the sum of the front magnet gap G_f and the back magnet gap G_b is a constant total G_t . Therefore, once a measured front-to-back gap G_{mag} is measured, and the front magnet gap G_f is determined, then the back magnet gap G_b can also be determined knowing the constant total gap G_t .

As with the side-to-side magnets, there is an offset between the front magnet gap and the position sensor gap. This offset must also be physically measured and recorded as "offset". The position sensor 110c measures and provides the measured front-to-back gap G_{mag} . The position sensor processing algorithm for the front magnet gap is the same as for each side magnet above.

The front magnet gap G_f is determined using Equation

4, as shown below:

$$G_f = G_{\text{min}} - \text{offset} \quad (\text{Eq. 4})$$

Knowing the front gap G_f and the constant total gap G_t , the back gap G_b is determined using Equation 5, as shown below:

$$G_b = G_t - G_f \quad (\text{Eq. 5})$$

Figure 9: Hardware Implementation
of the Force-Estimation Processor 310

Figure 9 shows a detailed circuit diagram of a hardware implementation of the force-estimation processor 310 of Figure 5 for computing the force estimate for driving a pair of magnets. The circuit is designed to be easily configured for either Front-To-Back or Side-To-Side Operation by setting a jumper shown in Figure 9A to either S/S for side-to-side or F/B for front-to-back, as shown.

Side-to-Side Gaps:

The jumper is set to S/S for side-to-side position.

A Left Side Magnet Gap G_{1s} :

The operational amplifiers U1-A and U1-B process the gap information for a left side magnet 150a in Figure 7

using Equation 3.

Gap 1 represents a measured left side gap signal G_{mls} and is a voltage output provided directly from the left side position sensor 110a in Figure 7. The measured left side gap signal G_{mls} is scaled to give a 0 to 10 volt signal proportional to a 0 to 10 millimeter gap. The operational amplifier U1-A has a potentiometer P1 associated therewith that is adjusted to calibrate the measured left side gap signal G_{mls} , so that the voltage at test point node TP1 equals a negative of the measured offset between the position sensor and the magnet gap. The output from the operational amplifier U1-A represents a negative an actual left side magnet gap $-G_L$.

The operational amplifier U1-A is configured as a summing junction computing:

$$-1 * (G_{mls} + -L_{offset})$$

where a left offset voltage L_{offset} is negative. In addition, the operational amplifier U1-A provides a single pole lag filter having a capacitor C20 and a resistor R2 with a breakpoint set at 30 Hz to reduce noise. The operational amplifier U1-B is a unity gain inverting amplifier. The output of the operational amplifier U1-B represents an actual left side magnet gap G_L .

A Right Side Magnet Gap G_r :

Operational amplifiers U1-C and U1-D process the gap information for a right side magnet 150b, shown in Figure 7, using Equation 4. Gap 2 represents a measured right side gap signal G_{mr} and is the voltage output directly from the right side position sensor 110b. Similar to the calibration discussed above, the operational amplifier U1-C has a potentiometer P2 that is adjusted to calibrate the measured right side gap signal G_{mr} , so that the voltage at a test point node TP2 equals the negative of the measured offset between the position sensor and the magnet gap. The output from the operational amplifier U1-D represents an actual right side magnet gap G_r .

Front-To-Back Gaps:

The jumper is set to F/B for front-to-back position.

A Front Magnet Gap G_f :

The operational amplifiers U1-A and U1-B process the gap information for a front magnet 150c shown in Figure 8 using Equation 5. Gap 1 represents a measured front-to-back gap signal G_{mb} and is the voltage output directly from the position sensor 110c. Calibration by adjustment of P1 is the same as for the left side magnet 150a (Figure 7), as discussed above. The output from the operational amplifier U1-B is an actual front magnet gap G_f .

A Back Magnet Gap G_b :

The operational amplifiers U1-C and U1-D process the gap information for a back magnet 150d using a combination of Equation 5 and Equation 6 shown below. The circuit in Figure 9 processes the actual back magnet gap G_b using Equations 4 and 5, as follows:

$$G_f = G_{mb} - \text{offset} \quad (\text{Eq. 4})$$

and

$$G_b = G_t - G_f \quad (\text{Eq. 5})$$

Substituting the actual front magnet gap G_f from Equation 4 into Equation 5 yields:

$$G_t - G_b = G_{mb} - \text{offset}$$

Solving for the actual back magnet gap G_b yields:

$$G_b = -G_{mb} + \text{offset} + G_t \quad (\text{Eq. 6})$$

Because there is only one sensor, the Gap 2 input is connected to the Gap 1 input, and this is the voltage output directly from the sole front-to-back position sensor 110c in Figure 8. The sensor output is scaled to give a 0 to 10 volt signal equal to a 0 to 10 millimeter gap. To calibrate the position processing, the potentiometer P2 is adjusted so that the voltage at node TP2 equals a minus of a sum of the constant total gap G_t and the measured front-to-back offset G_{mb} between the

position sensor 110c and the magnet gap. The output from the operational amplifier U1-C then becomes the back gap G_b .

The operational amplifier U1-C is configured as a summing junction computing the actual back magnet gap G_b of Equation 6.

In addition, the operational amplifier U1-C provides a single pole lag filter having a capacitor C21 and a resistor R1 with a breakpoint set at 30 Hz to reduce noise. The operational amplifier U1-D is jumpered-out (i.e. disabled) for the front-to-back configuration.

Force Estimation Network:

Once the gap has been successfully processed, the resulting signals must be processed with the corresponding sensed or command currents for the magnets to generate force estimates by using AD534 multipliers U2, U3, U4 and U5.

For the given values of the proportionality constant K_{mag} above, the current sense signal must be scaled to 1 volt per Ampere. If the available current sensed signal is scaled differently, the proportionality constant K_{mag} would be adjusted accordingly by recalculating Equation 1 for the given data.

The AD534 multipliers U2, U3, U4 and U5 with the following transfer function are used to divide the measured current I_{meas} by the actual magnet gap G_{mag} :

$$(X1-X2) * (Y1-Y2) = 10 * (Z1-Z2) \quad (\text{Eq. 7}),$$

where $X1$, $X2$, $Y1$, $Y2$, $Z1$, $Z2$ represent the inputs for the AD534 multipliers $U2$, $U3$, $U4$, $U5$.

By setting the $X2$, $Y1$ and $Z2$ inputs to zero, Equation 7 yields:

$$Y2 = -10 * Z1/X1 \quad (\text{Eq. 8})$$

As shown in Figure 9A, the measured current signal I_{mcs} is fed into the $Z1$ input and the actual magnet gap signal G_{mag} is fed into the $X1$ input, and the output signal becomes:

$$-10 * I_{\text{mcs}}/G_{\text{mag}}$$

The output signal from the AD534 multiplier $U2$ is fed into the AD534 multiplier $U4$ to perform a squaring function.

$$(X1-X2) * (Y1-Y2) = 10 * (Z1-Z2) \quad (\text{Eq. 7})$$

Setting the $X2$, $Y2$ and $Z2$ inputs to zero and sending the signal $10*I/G$ to both the $X1$ and $Y1$ inputs yields:

$$Z1 = 10 * I_{\text{mcs}}^2 / G_{\text{mag}}^2 \quad (\text{Equivalent with scaling to Eq. 1})$$

The AD534 multipliers U2 and U3 divide the current by the gap signal. The AD534 multipliers U4 and U5 are configured to square their input signals. The operational amplifier U6-A inverts the force estimate for channel 1, and the operational amplifier U6-B sums the two channels so that a positive signal represents the front (or Left Side) magnet force and a negative signal represents the back (or Right Side) magnet force. The operational amplifier U6-B is also configured as a first order lag filter having a capacitor C22 and a resistor R23 with a breakpoint set to 300 Hz to reduce noise without significant phase loss at 30 Hz. The operational amplifier U6-B also scales the output, as explained below.

Equation 1 determines the force in Newtons for the proportionality constant K_{mag} of 100 or 425 for the two different AMG magnets 160 shown in Figure 2. For a force feedback loop, the Force in Newtons must be scaled to a reasonable predetermined voltage range. Since the front-to-back magnet produces on the order of 650 Newtons, a reasonable output voltage might be 6.5 volts. In this case, the proportionality constant K_{mag} would become 1, and the operational amplifier U6-B would be set to a gain of 1. The side-to-side magnet produces 1300 Newtons. If the output is scaled to 6.5 volts, then the proportionality constant K_{mag} for the side-to-side magnet would become $425/200$ or 2.125. In this case, the

operational amplifier U6-B would be set to a gain of 2.125. The operational amplifier U6-C inverts this signal and provides it as the force-estimation processor signal from the force estimation processor 310 in Figure 5.

Figures 10-11: Position-Scheduled
Current Command Controller 400

Figure 10 shows an embodiment of the invention in which the current command controller 200 has a position-scheduled current command controller 210/400, which may be implemented in hardware or software. The scope of the invention is not intended to be limited to any particular embodiment of the position-scheduled current command controller 400, or whether it is implemented in hardware or software, or a combination thereof.

In the embodiment shown in Figure 10, the position-scheduled current command controller 400 is implemented in software using a microprocessor architecture similar to that shown in Figure 6.

In Figure 10, the position-scheduled current controller 210/400 responds to the force command signal F_d from the highest level control (i.e., outer position and/or acceleration loop) and the sensed gap signal G_m fed back from a gap sensor 110 of the magnet pair of interest, and uses Equation 1 to determine and provide a position-scheduled current controller signal as a current command I_c to the magnet driver circuit 220. The

position-scheduled current controller 210/400 must determine the actual magnet gap signal G_{mag} from the sensed gap signal G_m , as discussed above.

As also discussed above, the magnet driver circuit 220 is known in the art of elevator systems and similar to that discussed above with respect to Figure 5.

In operation, the position-scheduled current command controller 210/400 receives the force command signal F_d , and the sensed gap signal G_m , and generates a position-scheduled current command controller as the current command I_c to the magnet driver circuit 200, using a technique that requires the current output of the magnet driver circuit 220 to follow accurately the current command signal I_c . The method for controlling may include the following steps:

Step 1: Determine the magnet to energize

Based on the sign of the desired force request, one can determine the appropriate electromagnet to energize. For example, a positive force indicates to energize the front or right side magnet, while a negative force indicates to energize the back or left side magnet.

Step 2: Determine the actual magnet gap G_{mag}

Determine from the measured gap sensors what the actual magnet gap G_{mag} for the specific magnet identified in Step 1. This position sensor processing follows the

same procedure as outlined in the prior discussion with respect to Figures 6 and 7 of the gap sensor processing algorithm.

Step 3: Determine the command currents for the magnets

Based on steps (1) and (2), determine the command currents for all magnets using the idealized model of Equation 1 and the empirically derived magnet proportionality constant K_{mag} , that is,

$$I_c = \text{square root } (F_d/K_{mag}) * G_{mag}$$

Attached as Table 1 at the end of the specification is the software code for the position-scheduled current command controller 210/400.

Figure 11 shows the position-scheduled current command controller 210/400 in combination with the position and acceleration feedback control 300 corresponding to reference numeral 130 in Figure 2, magnets 150c, 150d in Figure 8, position sensor 110c in Figure 8 and a magnet driver 500 similar to the magnet driver circuit 200 in Figure 10. As shown, the magnet driver circuit 500 responds to the current command signal I_c , which represents a position-scheduled current command, and provides a magnet driver signal to magnets 150c, 150d for front-to-back control of an elevator car (not shown). The magnet driver circuit 500 includes a pair of diodes

502, 504, an inverter 506, two comparators 508, 510, which are simultaneously biased, lag filters 512, 514, amplifiers 516, 518, and nodes 520, 522 for feeding back left front-to-back signals l_n to the lag filters 512, 514.

Figure 12:

Figure 12 shows a graph of side-to-side AMG magnet performance used to derive the Equation 1.

In general, the two different magnet configurations of Figures 7 and 8 were tested in a testing rig that holds the magnets a fixed distance from an elevator guide rail (reaction surface) in a static relationship. A fixture was equipped to measure a force produced using precision load cells and a magnetic field using Hall effect sensors while the magnet was energized with a DC current. The experiment was conducted for gaps from 2 to 10 millimeters in 2 millimeter increments and currents from 0 to 20 Amperes in 2 Ampere increments. The data was analyzed and an empirical model of Force versus Current and Gap was derived by curve fitting. Figure 12 provides the graph of a set of data tested using the model of the derived Equation 1:

$$\text{Force} = K_{\text{mag}} * (I_c / G_{\text{mag}})^2,$$

where the Force represents a force signal scaled in

Newtons, K_{mag} represents a proportionality constant for a given magnet configuration, (100 for the smaller Front-To-Back Axis AMG Magnet and 425 for the Larger Side-To-Side Axis AMG Magnet), I_c represents a desired force compensated current command signal scaled in Amperes, and G_{mag} (also referred to as G_{meas} above) represents the actual magnet gap between a pole face and a reaction surface scaled in millimeters.

For a range of gaps between 2 to 10 millimeters and a range of magnet currents between 0 to 20 Amperes, within the rated force limits of the electromagnets, Equation 1 models the force generated within less than 10% from the load cell measured force. The plot in Figure 12 is for the 1300 Newton Side-To-Side AMG magnet operating at a 6 millimeter air gap. The asterisk "*" in Figure 12 indicates where the current measurements were taken. Both of these cases are outside the normal operating range of the system. This estimation error using current and gap is comparable to the error associated with the Hall effect sensor derived force estimation. These force errors are insignificant in active elevator suspension systems that typically have outer feedback loops (based on gap sensing and/or acceleration signals) closed around this inner force loop.

Scope of the Invention

It will thus be seen that the objects set forth above, and those made apparent from the preceding description, are efficiently attained and, since certain changes may be made in the above construction without departing from the scope of the invention, it is intended that all matter contained in the above description or shown in the accompanying drawings shall be interpreted as illustrative and not in a limiting sense.

It should be realized that the position and feedback controllers 130, 300 of Figures 2 and 11, respectively, may take forms other than indicated, e.g., according to a "learned-rail" approach shown in U.S. Serial No. 08/292,660, as well as that disclosed in U.S. Patent Nos. 5,294,757; 5,304,751 and 5,308,938.

Other applications include any system that uses an electromagnet to produce a force across a large air gap. The electromagnet force may be either attractive or repulsive. Specific applications are envisioned in the field of magnetic levitation trains or shuttles, and magnetic bearings.

It is also to be understood that the following claims are intended to cover all of the generic and specific features of the invention herein described and all statements of the scope of the invention which, as a matter of language, might be said to fall therebetween.

TABLE 1

** FOR USE WITH BORLANDC C++ or TC++
**

** COMPUTER: PC
**

** MODULE PURPOSE: Program for data acquisition of "16" single ended
** analog channels of data from a MetroByte DAS-20 board and
** drive "6" channels of magnets using the DDA06 Analog output
** card. Second DDA06 is used for diagnostic purposes.

- UNPUBLISHED WORK -

© 1996 Otis Elevator Company

```

**      Data to be acquired at user defined rate using the
**      DAS-20 programmable timer. This version acquires data
**      under program control. The first channel is synced to
**      the programmable timer and all following channels are taken
**      immediately.

```

```

** using MEDIUM memory model
** Uses project file that includes TC_ALLOC.C and DAS20CM.LIB

```

```

** Copyright - United Technologies Research Center

```

```

...../

```

```

#include <conio.h>      // clrscr() getch() kbhit()
#include <stdio.h>      // printf() fopen() fclose() sscanf() FILE
#include <stdlib.h>     // exit()
#include <alloc.h>      // farcorleft() farcalloc() farfree()
#include <math.h>       // sin() M_PI fabs()
#include <dos.h>        // delay

```

```

extern int das20(int, int *); // declare CALL structure

```

```

#define BUFFERL      101
#define BUFFER       10
#define DAS_ADDR     0X310      // DAS-20 base address
#define DDA06_ADDR   0X330      // DDA06 base address
#define DDA06_ADDR1  0X340      // DDA06 -1 base address
#define NUM_CHANS     16
#define CONFIG_COEFS  16
#define CAL_COEFS      7
#define SS_STATE_COEFS 40
#define SS_PIDM_COEFS  20
#define SS_LAG_COEFS   40
#define SS_PLAG_COEFS  40
#define FB_STATE_COEFS 80
#define FB_PIDM_COEFS  80
#define FB_LAG_COEFS   80
#define FB_PLAG_COEFS  40
#define GLAG_COEFS     80

```

```

/*
.....
GLOBAL VARIABLES
.....

```

```

*/
int Params[16];
int Flag;
int Mode;

```

```

/*
.....
DEFINE FUNCTIONS
.....

```

```

*/
void get_chans_gains();
void read_config_fil();
void read_cal_fil();
void read_pos_fil();
void read_acc_fil();
void read_ss_state_fil();
void read_ss_pidm_fil();
void read_ss_lag_fil();
void read_ss_plag_fil();
void read_fb_state_fil();
void read_fb_pidm_fil();
void read_fb_lag_fil();
void read_fb_plag_fil();
void read_glag_fil();
void init_das20();
void init_dda06();
void define_queue();
void set_block_timer_freq();
void ss_control();
void fb_control();
void ss_gap_filter();
void fb_gap_filter();
void analog_out();
void analog_out1();

```

```

/*

```

MAIN PROGRAM

```

/*

```

```

main(void)
{

```

```

    register i;

```

```

    int samples_per_sec;
    int toggle;
    int base;
    int temppos;

```

```

    int chans[NUM_CHANS];
    int gains[NUM_CHANS];
    int temp_data[NUM_CHANS];
    int temp_chan_data[NUM_CHANS];
    int map1[BUFFER];
    int blfb_magnet_cmd[BUFFER];
    int bss_magnet_cmd[BUFFER];
    int tss_magnet_cmd[BUFFER];
    int brfb_magnet_cmd[BUFFER];
    int tlf_magnet_cmd[BUFFER];
    int trfb_magnet_cmd[BUFFER];

```

```

unsigned int dma_actual_seg;
unsigned int dma_dest_seg;

float c_to_v = 10./2047;
float v_to_c = 2047./10;
float kss = 8.37; // [425 * 1.6^2(A/V)] / [130(N/V)]
float knb = 3.94; // [100 * 1.6^2(A/V)] / [65 (N/V)]

float ch0_v, blf_gap_mm, blf_gap_err;
float ch1_v, bls_gap_mm, bls_gap_err;
float ch2_v, brf_gap_mm, brf_gap_err;
float ch3_v, brs_gap_mm;
float ch4_v, tlf_gap_mm, tlf_gap_err;
float ch5_v, tls_gap_mm, tls_gap_err;
float ch6_v, trf_gap_mm, trf_gap_err;
float ch7_v, trs_gap_mm;
float ch8_v;
float ch9_v;
float ch10_v;
float ch11_v;
float ch12_v;
float ch13_v;
float ch14_v;
float ch15_v;

float config[CONFIG_COEFS];

float blfb_cal_coef[CAL_COEFS];
float bls_cal_coef[CAL_COEFS];
float brfb_cal_coef[CAL_COEFS];
float brs_cal_coef[CAL_COEFS];
float tlfb_cal_coef[CAL_COEFS];
float tls_cal_coef[CAL_COEFS];
float trfb_cal_coef[CAL_COEFS];
float trs_cal_coef[CAL_COEFS];

float ss_state_coefs[SS_STATE_COEFS];
float ss_pidm_coefs[SS_PIDM_COEFS];
float ss_lag_coefs[SS_LAG_COEFS];
float ss_plag_coefs[SS_PLAG_COEFS];
float fb_state_coefs[FB_STATE_COEFS];
float fb_pidm_coefs[FB_PIDM_COEFS];
float fb_lag_coefs[FB_LAG_COEFS];
float fb_plag_coefs[FB_PLAG_COEFS];
float glag_coefs[GLAG_COEFS];

int *ch0;
int *ch1;
int *ch2;
int *ch3;
int *ch4;
int *ch5;
int *ch6;
int *ch7;

```

```

int *ch8;
int *ch9;
int *ch10;
int *ch11;
int *ch12;
int *ch13;
int *ch14;
int *ch15;

float *ss_cmd_v;
float *fb_cmd_v;
float *filtered_gaps;
init_dda06();

clrscr();

// READ CONTROL COEFFICIENTS DATA FROM DISK

read_config_fil(&config);

read_cal_fil(&blfb_cal_coef,&bls_cal_coef,&brfb_cal_coef,&brs_cal_coef,
            &tlfb_cal_coef,&tls_cal_coef,&trfb_cal_coef,&trs_cal_coef);

read_ss_state_fil(&ss_state_coefs);
read_ss_pidm_fil(&ss_pidm_coefs);
read_ss_lag_fil(&ss_lag_coefs);
read_ss_plag_fil(&ss_plag_coefs);
read_fb_state_fil(&fb_state_coefs);
read_fb_pidm_fil(&fb_pidm_coefs);
read_fb_lag_fil(&fb_lag_coefs);
read_fb_plag_fil(&fb_plag_coefs);
read_glag_fil(&glag_coefs);

samples_per_sec = (int) config[0];

init_das20();
get_chans_gains(chans,gains);
define_queue(chans,gains,NUM_CHANS);
set_block_timer_freq(samples_per_sec);

if (((int *) ch0 = calloc(BUFFERL,sizeof(int))) == NULL)
{
    printf("ERROR Allocating memory for CHANNEL 0");
    exit(-1);
}
if (((int *) ch1 = calloc(BUFFERL,sizeof(int))) == NULL)
{
    printf("ERROR Allocating memory for CHANNEL 1");
    exit(-1);
}
if (((int *) ch2 = calloc(BUFFERL,sizeof(int))) == NULL)
{
    printf("ERROR Allocating memory for CHANNEL 2");
    exit(-1);
}

```

```
}  
if (((int *) ch3 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 3");  
    exit(-1);  
}  
if (((int *) ch4 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 4");  
    exit(-1);  
}  
if (((int *) ch5 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 5");  
    exit(-1);  
}  
if (((int *) ch6 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 6");  
    exit(-1);  
}  
if (((int *) ch7 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 7");  
    exit(-1);  
}  
if (((int *) ch8 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 8");  
    exit(-1);  
}  
if (((int *) ch9 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 9");  
    exit(-1);  
}  
if (((int *) ch10 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 10");  
    exit(-1);  
}  
if (((int *) ch11 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 11");  
    exit(-1);  
}  
if (((int *) ch12 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 12");  
    exit(-1);  
}  
if (((int *) ch13 = calloc(BUFFERL, sizeof(int))) == NULL)  
{  
    printf("ERROR Allocating memory for CHANNEL 13");  
}
```

```

    exit(-1);
}
if (((int *) ch14 = calloc(BUFFERL, sizeof(int))) == NULL)
{
    printf("ERROR Allocating memory for CHANNEL 14");
    exit(-1);
}
if (((int *) ch15 = calloc(BUFFERL, sizeof(int))) == NULL)
{
    printf("ERROR Allocating memory for CHANNEL 15");
    exit(-1);
}
if ((dma_actual_seg = TC_alloc(2*NUM_CHANS, &dma_dest_seg)) == 65535)
{
    printf("\n\t Cannot Allocate Buffer");
    exit(-1);
}
if (((int *) ss_cmd_v = calloc(2, sizeof(float))) == NULL)
{
    printf("ERROR Allocating memory for SS_CMD_V");
    exit(-1);
}
if (((int *) fb_cmd_v = calloc(4, sizeof(float))) == NULL)
{
    printf("ERROR Allocating memory for FB_CMD_V");
    exit(-1);
}
if (((int *) filtered_gaps = calloc(3, sizeof(float))) == NULL)
{
    printf("ERROR Allocating memory for FILTERED GAPS");
    exit(-1);
}
}

// BLOCK SCAN FOR DATA

Params[0]=2;           // Number of times thru queue
Params[1]=dma_actual_seg; // Pointer to address for data
Params[2]=2;           // Internal trigger no gate
Params[3]=0;           // Recycle
Flag = das20(27, Params);
    if (Flag != 0)           // Error if flag not 0
    {
        printf("\nMode.27: Error in call to block scan. Flag=%d\n", Flag);
        exit(-1);
    }

toggle=1;
base=0;
i=0;

do
{
    //WAIT FOR COMPLETION OF SINGLE QUEUE PASS

```

```

do
{
    if ((Flag = das20(12,Params)) !=0)
    {
        printf("\n\t Mode 12: Error monitoring conversions. Flag=%d",
            Flag);
        printf("\n\t Conversion %u",Params[2]);
        exit(-1);
    }
}while (Params[2] != NUM_CHANS * toggle); // Wait for A/D conversions

// READ CONVERSIONS INTO TEMPORARY ARRAY

Params[0]=NUM_CHANS;           // Number of channels
Params[1]=dma_actual_seg;      // Pointer to address for data
Params[2]=base;                // Offset address for source data
Params[3]=offadr(temp_data);   // Where to put data
Params[4]=offadr(temp_chan_data); // Where to put channel data
Params[5]=1;                   // Bipolar data
Params[6]=0;                   // No SSH-4

Flag = das20(13, Params);
if (Flag != 0)                 // Error if flag not 0
{
    printf("\n\t Mode 13: Error Retrieving data form memory. Flag=%d\n",
        Flag);
    exit(-1);
}

//READ TEMPORARY ARRAY INTO FINAL ARRAYS

*(ch0+i) = temp_data[0];
*(ch1+i) = temp_data[1];
*(ch2+i) = temp_data[2];
*(ch3+i) = temp_data[3];
*(ch4+i) = temp_data[4];
*(ch5+i) = temp_data[5];
*(ch6+i) = temp_data[6];
*(ch7+i) = temp_data[7];
*(ch8+i) = temp_data[8];
*(ch9+i) = temp_data[9];
*(ch10+i) = temp_data[10];
*(ch11+i) = temp_data[11];
*(ch12+i) = temp_data[12];
*(ch13+i) = temp_data[13];
*(ch14+i) = temp_data[14];
*(ch15+i) = temp_data[15];

/* Generate Square Wave to check Timing. Wave will be HIGH (5V)
   for 1/samples_per_sec and LOW for 1/samples_per_sec. I.E.
   Square Wave will be 1/2 Sample Rate Frequency.
*/

```

```
map1[i] = 1024 * toggle;
```

```
//FIRST LINEARIZE AND NORMALIZE THE GAP SENSORS.
```

```
ch0_v = c_to_v * *(ch0+i);
blf_gap_mm = ch0_v * ch0_v * ch0_v * ch0_v * blfb_cal_coef[0] +
             ch0_v * ch0_v * ch0_v *      * blfb_cal_coef[1] +
             ch0_v * ch0_v *      * blfb_cal_coef[2] +
             ch0_v *      * blfb_cal_coef[3] +
             0.5 * blfb_cal_coef[4] +
             blfb_cal_coef[5];
```

```
blf_gap_err = blf_gap_mm - 0.5 * blfb_cal_coef[5];
```

```
ch1_v = c_to_v * *(ch1+i);
bls_gap_mm = ch1_v * ch1_v * ch1_v * ch1_v * bls_cal_coef[0] +
             ch1_v * ch1_v * ch1_v *      * bls_cal_coef[1] +
             ch1_v * ch1_v *      * bls_cal_coef[2] +
             ch1_v *      * bls_cal_coef[3] +
             bls_cal_coef[4];
```

```
ch2_v = c_to_v * *(ch2+i);
brf_gap_mm = ch2_v * ch2_v * ch2_v * ch2_v * brfb_cal_coef[0] +
             ch2_v * ch2_v * ch2_v *      * brfb_cal_coef[1] +
             ch2_v * ch2_v *      * brfb_cal_coef[2] +
             ch2_v *      * brfb_cal_coef[3] +
             0.5 * brfb_cal_coef[4] +
             brfb_cal_coef[5];
```

```
brf_gap_err = brf_gap_mm - 0.5 * brfb_cal_coef[5];
```

```
ch3_v = c_to_v * *(ch3+i);
brs_gap_mm = ch3_v * ch3_v * ch3_v * ch3_v * brs_cal_coef[0] +
             ch3_v * ch3_v * ch3_v *      * brs_cal_coef[1] +
             ch3_v * ch3_v *      * brs_cal_coef[2] +
             ch3_v *      * brs_cal_coef[3] +
             brs_cal_coef[4];
```

```
ch4_v = c_to_v * *(ch4+i);
tlf_gap_mm = ch4_v * ch4_v * ch4_v * ch4_v * tlfb_cal_coef[0] +
             ch4_v * ch4_v * ch4_v *      * tlfb_cal_coef[1] +
             ch4_v * ch4_v *      * tlfb_cal_coef[2] +
             ch4_v *      * tlfb_cal_coef[3] +
             0.5 * tlfb_cal_coef[4] +
             tlfb_cal_coef[5];
```

```
tlf_gap_err = tlf_gap_mm - 0.5 * tlfb_cal_coef[5];
```

45

```

ch5_v = c_to_v * (*(ch5+i));
ils_gap_mm = ch5_v * ch5_v * ch5_v * ch5_v * ils_cal_coef[0] +
              ch5_v * ch5_v * ch5_v *      * ils_cal_coef[1] +
              ch5_v * ch5_v *      * ils_cal_coef[2] +
              ch5_v *      * ils_cal_coef[3] +
              ils_cal_coef[4];

```

```

ch6_v = c_to_v * (*(ch6+i));
trf_gap_mm = ch6_v * ch6_v * ch6_v * ch6_v * trfb_cal_coef[0] +
              ch6_v * ch6_v * ch6_v *      * trfb_cal_coef[1] +
              ch6_v * ch6_v *      * trfb_cal_coef[2] +
              ch6_v *      * trfb_cal_coef[3] +
              trfb_cal_coef[4] +
              0.5 * trfb_cal_coef[5];

```

```

trf_gap_err = trf_gap_mm - 0.5 * trfb_cal_coef[5];

```

```

ch7_v = c_to_v * (*(ch7+i));
trs_gap_mm = ch7_v * ch7_v * ch7_v * ch7_v * trs_cal_coef[0] +
              ch7_v * ch7_v * ch7_v *      * trs_cal_coef[1] +
              ch7_v * ch7_v *      * trs_cal_coef[2] +
              ch7_v *      * trs_cal_coef[3] +
              trs_cal_coef[4];

```

```

// Send Out Linearized Gaps for Diagnostic Purposes

```

```

temppos = (int) v_to_c * blf_gap_err;
analog_outl(1,temppos);
temppos = (int) v_to_c * brf_gap_err;
analog_outl(2,temppos);
temppos = (int) v_to_c * tlf_gap_err;
analog_outl(4,temppos);
temppos = (int) v_to_c * trf_gap_err;
analog_outl(0,temppos);

```

```

// Zero Offset on ACCELEROMETERS

```

```

if (i==0)
{
    blfb_cal_coef[6] = c_to_v * (*(ch8+i));
    bls_cal_coef[6] = c_to_v * (*(ch9+i));
    brfb_cal_coef[6] = c_to_v * (*(ch10+i));
    tlfb_cal_coef[6] = c_to_v * (*(ch11+i));
    tls_cal_coef[6] = c_to_v * (*(ch12+i));
    trfb_cal_coef[6] = c_to_v * (*(ch13+i));
}
ch8_v = (c_to_v * (*(ch8+i))) - blfb_cal_coef[6];
ch9_v = (c_to_v * (*(ch9+i))) - bls_cal_coef[6];
ch10_v = (c_to_v * (*(ch10+i))) - brfb_cal_coef[6];
ch11_v = (c_to_v * (*(ch11+i))) - tlfb_cal_coef[6];

```

```

ch12_v = (c_to_v * (ch12-mm) + fb_cal_coef[6],
ch13_v = (c_to_v * (ch13-mm) + tlb_cal_coef[6],

// Left Side Minus Right Side is Insensitive to Changes in DBG
bls_gap_err = (bls_gap_mm - brs_gap_mm)/2 - bls_gap_mm,
tls_gap_err = (tls_gap_mm - trs_gap_mm)/2 - tls_gap_mm,

//CALL SIDE TO SIDE PID CONTROLLER
ss_control(ss_cmd_v,bls_gap_err,tls_gap_err,
           ch9_v,ch12_v,
           ss_state_coefs,ss_pidm_coefs,
           ss_lag_coefs,ss_plag_coefs,i);

//CALL FRONT TO BACK PID CONTROLLER
fb_control(fb_cmd_v,blf_gap_err,brf_gap_err,
           tlf_gap_err,trf_gap_err,
           ch8_v,ch10_v,ch11_v,ch13_v,
           fb_state_coefs,fb_pidm_coefs,
           fb_lag_coefs,fb_plag_coefs,i);

//FILTER GAP READINGS TO COMPUTE CURRENT CONTROL
ss_gap_filter(filtered_gaps,bls_gap_mm,brs_gap_mm,
              tls_gap_mm,trs_gap_mm,
              glag_coefs,i);

fb_gap_filter(filtered_gaps,blf_gap_mm,brf_gap_mm,
              tlf_gap_mm,trf_gap_mm,
              glag_coefs,i);

bls_gap_mm = *(filtered_gaps+0);
brs_gap_mm = *(filtered_gaps+1);
tls_gap_mm = *(filtered_gaps+2);
trs_gap_mm = *(filtered_gaps+3);
// blf_gap_mm = *(filtered_gaps+4);
// brf_gap_mm = *(filtered_gaps+5);
// tlf_gap_mm = *(filtered_gaps+6);
// trf_gap_mm = *(filtered_gaps+7);

//ADD INJECTED FORCE SIGNAL TO CLOSED LOOP FORCE COMMAND
ch14_v = c_to_v * (ch14+i); // diagnostic for force injection

*(ss_cmd_v ) = *(ss_cmd_v ) + ch14_v * config[11];
*(ss_cmd_v+1) = *(ss_cmd_v+1) + ch14_v * config[14];
// *(ss_cmd_v+1) = ch14_v * config[14];

*(fb_cmd_v ) = *(fb_cmd_v ) + ch14_v * config[10];
*(fb_cmd_v+1) = *(fb_cmd_v+1) + ch14_v * config[12];
*(fb_cmd_v+2) = *(fb_cmd_v+2) + ch14_v * config[13];
*(fb_cmd_v+3) = *(fb_cmd_v+3) + ch14_v * config[15];

```

```
/*SOFT START
```

```
/*
    if (t < 5)
    {
        start_gain = t/5;
        *(ss_cmd_v) = *(ss_cmd_v) * start_gain;
        *(ss_cmd_v+1) = *(ss_cmd_v+1) * start_gain;
        *(fb_cmd_v) = *(fb_cmd_v) * start_gain;
        *(fb_cmd_v+1) = *(fb_cmd_v+1) * start_gain;
        *(fb_cmd_v+2) = *(fb_cmd_v+2) * start_gain;
        *(fb_cmd_v+3) = *(fb_cmd_v+3) * start_gain;
    }
*/
```

```
/* SIDE TO SIDE CURRENT CONTROL
```

ss_cmd_v is the volts equivalent for Force in newtons.
The conversion is 130 N/V.

```
/*
// Bottom
if ( *(ss_cmd_v) < 0 ) // Use right hand side magnet
{
    *(ss_cmd_v) = -1 * (*(ss_cmd_v));
    *(ss_cmd_v) = -1.2 * fabs(brs_gap_mm) * sqrt( *(ss_cmd_v)/kss);
}
else // Use left hand side magnet
{
    *(ss_cmd_v) = 1.2 * fabs(bls_gap_mm) * sqrt( *(ss_cmd_v)/kss);
}

// Top
if ( *(ss_cmd_v+1) < 0 ) // Use right hand side magnet
{
    *(ss_cmd_v+1) = -1 * (*(ss_cmd_v+1));
    *(ss_cmd_v+1) = -1.27 * fabs(trs_gap_mm) * sqrt( *(ss_cmd_v+1)/kss);
}
else // Use left hand side magnet
{
    *(ss_cmd_v+1) = 1.27 * fabs(tls_gap_mm) * sqrt( *(ss_cmd_v+1)/kss);
}
*/
```

```
/* FRONT TO BACK CURRENT CONTROL
```

fb_cmd_v is the volts equivalent for Force in newtons.
The conversion is 65 N/V.

```
/*
// Bottom Left
if ( *(fb_cmd_v) < 0 ) // Use Back Magnet
{
    *(fb_cmd_v) = -1 * (*(fb_cmd_v));
    *(fb_cmd_v) = -1 * (blfb_cal_coef[5] - blf_gap_mm - config[1]) *
        sqrt( *(fb_cmd_v)/kfb);
}
else // Use Front Magnet
{
    *(fb_cmd_v) = 1 * (blf_gap_mm - config[2]) *
*/
```

```

        sqrt( *(fb_cmd_v)/kfb);
    }

    // Bottom Right
    if ( *(fb_cmd_v+1) < 0 ) // Use Back Magnet
    {
        *(fb_cmd_v+1) = -1 * *(fb_cmd_v+1);
        *(fb_cmd_v+1) = -1 * (brfb_cal_coef[5] - brf_gap_mm - config[3]) *
            sqrt( *(fb_cmd_v+1)/kfb);
    }
    else // Use Front Magnet
    {
        *(fb_cmd_v+1) = 1 * (brf_gap_mm - config[4]) *
            sqrt( *(fb_cmd_v+1)/kfb);
    }

    // Top Left
    if ( *(fb_cmd_v+2) < 0 ) // Use Back Magnet
    {
        *(fb_cmd_v+2) = -1 * *(fb_cmd_v+2);
        *(fb_cmd_v+2) = -1 * (tlfb_cal_coef[5] - tlf_gap_mm - config[5]) *
            sqrt( *(fb_cmd_v+2)/kfb);
    }
    else // Use Front Magnet
    {
        *(fb_cmd_v+2) = 1 * (tlf_gap_mm - config[6]) *
            sqrt( *(fb_cmd_v+2)/kfb);
    }

    // Top Right
    if ( *(fb_cmd_v+3) < 0 ) // Use Back Magnet
    {
        *(fb_cmd_v+3) = -1 * *(fb_cmd_v+3);
        *(fb_cmd_v+3) = -1 * (trfb_cal_coef[5] - trf_gap_mm - config[7]) *
            sqrt( *(fb_cmd_v+3)/kfb);
    }
    else // Use Front Magnet
    {
        *(fb_cmd_v+3) = 1 * (trf_gap_mm - config[8]) *
            sqrt( *(fb_cmd_v+3)/kfb);
    }

    blfb_magnet_cmd[i] = (int)v_to_c * *(fb_cmd_v);
    bss_magnet_cmd[i] = (int)v_to_c * *(ss_cmd_v);
    brfb_magnet_cmd[i] = (int)v_to_c * *(fb_cmd_v+1);
    tlfb_magnet_cmd[i] = (int)v_to_c * *(fb_cmd_v+2);
    tss_magnet_cmd[i] = (int)v_to_c * *(ss_cmd_v+1);
    trfb_magnet_cmd[i] = (int)v_to_c * *(fb_cmd_v+3);

    //Send Control Outputs to Magnet Drivers

    analog_out(0,blfb_magnet_cmd[i]),

```

```

analog_out(1,bss_magnet_cmd[i]);
analog_out(2,brfb_magnet_cmd[i]);
analog_out(3,tlfb_magnet_cmd[i]);
analog_out(4,lss_magnet_cmd[i]);
analog_out(5,trfb_magnet_cmd[i]);

//Send these signals to the D/A for test purposes

//  analog_out(4,map1[i]); // timing

//WAIT FOR NEXT TRIGGER
do
{
    if ((Flag = das20(12,Params)) !=0)
    {
        printf ("\n\t Mode 12: Error monitoring conversions. Flag=%d",
                Flag);
        printf ("\n\t Conversion %u",Params[2]);
        exit(-1);
    }
}while (Params[2] == NUM_CHANS*toggle);    // Wait for trigger

if (toggle==1)
{
    toggle=0;
    base=NUM_CHANS;
}
else
{
    toggle=1;
    base=0;
}

i=1;
}while(!kbhit());

// STOP DMA OPERATION

if ((Flag = das20(11,Params)) !=0)
{
    printf ("\n\t Mode 11: Error stopping DMA. Flag=%d",Flag);
    exit(-1);
}

// ZERO ALL ANALOG OUTPUTS

init_dda06();

TC_free(dma_actual_seg);

free (ch0);
free (ch1);
free (ch2);

```

```

free (ch3);
free (ch4);
free (ch5);
free (ch6);
free (ch7);
free (ch8);
free (ch9);
free (ch10);
free (ch11);
free (ch12);
free (ch13);

```

```

return;
}

```

```

/*

```

```

.....
GET_CHANS_GAINS: GETS CHANNELS AND GAIN DEFINITION FOR SCAN QUEUE

```

```

.....
*/

```

```

void get_chans_gains(chans,gains)
int chans[];
int gains[];
{
    int i;

    for (i=0; i<NUM_CHANS; i++)
    {
        chans[i] = i;
        gains[i] = 1;
    }
    return;
}

```

```

/*

```

```

.....
READ_CONFIG_FILE: READS CONFIGURATION DATA FROM DISK

```

```

.....
*/

```

```

void read_config_fil(map1)
float *map1;

{
    int i;
    int j=20;
    float temp_data1;
    char in_line[18];
    FILE *data;

```

```

if ((data = fopen("cnfgcoef.dat", "r")) == NULL)
{
    printf("\n\n ERROR !! Could not open data file CNFGCOEF.DAT\n\n");
    exit(-1);
}

for (i=0; i<CONFIG_COEFS; ++i)
{
    if (fgets(in_line, j, data) == NULL)
    {
        printf("\n\n ERROR READING CNFGCOEF.DAT !!");
        printf("\n\n Could not read line %d . \n", i+1);
        exit(-1);
    }

    if (sscanf(in_line, " %e ",
               &temp_data1) != 1)
    {
        printf("\n\n ERROR READING CNFGCOEF.DAT !!");
        printf("\n\n Wrong format for line %d. \n", i+1);
        exit(-1);
    }
    map1[i] = temp_data1;
}

fclose(data);
return;
}

/*
.....

READ_CAL_FILE: READS SENSOR CALIBRATION DATA FROM DISK

.....
*/

void read_cal_fil(map1, map2, map3, map4, map5, map6, map7, map8)
float *map1;
float *map2;
float *map3;
float *map4;
float *map5;
float *map6;
float *map7;
float *map8;

{
    int i;
    int j=150;
    float temp_data1;
    float temp_data2;
    float temp_data3;
    float temp_data4;
    float temp_data5;

```

```

float temp_data0,
float temp_data7,
float temp_data8,
char in_line[140],
FILE *data;

if ((data = fopen("calcoef4.dat", "r")) == NULL)
{
    printf("\n\n ERROR !! Could not open data file CALCOEF4.DAT\n");
    exit (-1);
}

for (i=0; i<CAL_COEFS; ++i)
{
    if (fgets (in_line, j, data) == NULL)
    {
        printf("\n\n ERROR READING CALCOEF4.DAT !!");
        printf("\n\n Could not read line %d . \n", i+1);
        exit (-1);
    }

    if (sscanf (in_line, " %c %c %c %c %c %c %c %c %c ",
        &temp_data1,
        &temp_data2,
        &temp_data3,
        &temp_data4,
        &temp_data5,
        &temp_data6,
        &temp_data7,
        &temp_data8) != 8)
    {
        printf("\n\n ERROR READING CALCOEF4.DAT !!");
        printf("\n\n Wrong format for line %d. \n", i+1);
        exit (-1);
    }
    map1[i]= temp_data1;
    map2[i]= temp_data2;
    map3[i]= temp_data3;
    map4[i]= temp_data4;
    map5[i]= temp_data5;
    map6[i]= temp_data6;
    map7[i]= temp_data7;
    map8[i]= temp_data8;
}
fclose(data);
return;
}

/*
.....

READ_SS_STATE_FILE: READS Side To Side STATE COEFFICIENTS FROM DISK
.....

```

```

*/
void read_ss_state_fil(map1)
float *map1;

{
    int i;
    int j=40;
    int offset;
    float temp_data1;
    float temp_data2;
    char in_line[49];
    FILE *data;

    if ((data = fopen ("ssstatcf.dat","r")) == NULL )
    {
        printf ("\n\n ERROR !! Could not open data file SSSTATCF.DAT\n");
        exit (-1);
    }

    offset = SS_STATE_COEFS/2;

    for (i=0; i<offset; ++i)
    {
        if (fgets (in_line, j, data) == NULL)
        {
            printf ("\n\n ERROR READING SSSTATCF.DAT !!");
            printf ("\n\n Could not read line %d . \n",i+1);
            exit (-1);
        }

        if (sscanf (in_line, " %c %c "
                    &temp_data1,
                    &temp_data2) != 2)
        {
            printf ("\n\n ERROR READING SSSTATCF.DAT !!");
            printf ("\n\n ERROR !! Wrong format for line %d. \n",i+1);
            exit (-1);
        }
        map1[i] = temp_data1;
        map1[i+( offset)] = temp_data2;
    }
    fclose(data);
    return;
}

/*
.....

READ_SS_PIDM_FILE: READS Side To Side PIDM COEFFICIENTS FROM DISK

.....
*/
void read_ss_pidm_fil(map1)
float *map1;

```

```

{
    int i;
    int j=40;
    int offset;
    float temp_data1;
    float temp_data2;
    char in_line[49];
    FILE *data;

    if (( data = fopen ("sspidmcf.dat","r")) == NULL )
    {
        printf ("\n\t ERROR !! Could not open data file SSPIDMCF.DAT\n");
        exit (-1);
    }

    offset = SS_PIDM_COEFS/2;

    for (i=0; i<offset; ++i)
    {
        if (fgets (in_line, j, data) == NULL)
        {
            printf ("\n\t ERROR READING SSPIDMCF.DAT !!");
            printf ("\n\t Could not read line %d . \n",i+1);
            exit (-1);
        }

        if (sscanf (in_line, " %c %c ",
                    &temp_data1,
                    &temp_data2) != 2)
        {
            printf ("\n\t ERROR READING SSPIDMCF.DAT !!");
            printf ("\n\t ERROR !! Wrong format for line %d. \n",i+1);
            exit (-1);
        }
        map1[i] = temp_data1;
        map1[i+( offset)] = temp_data2;
    }
    fclose(data);
    return;
}

/*
.....

READ_SS_LAG_FILE: READS Side To Side LAG COEFFICIENTS FROM DISK
.....
*/
void read_ss_lag_fil(map1)
float *map1;

{

```

```

int i;
int j=40;
int offset;
float temp_data1;
float temp_data2;
char in_line[49];
FILE *data;

if ((data = fopen("sslagcf.dat", "r")) == NULL)
{
    printf("\n\n ERROR !! Could not open data file SSLAGCF.DAT\n");
    exit(-1);
}

offset = SS_LAG_COEFS/2;

for (i=0; i<offset; ++i)
{
    if (fgets(in_line, j, data) == NULL)
    {
        printf("\n\n ERROR READING SSLAGCF.DAT !!");
        printf("\n\n Could not read line %d . \n", i+1);
        exit(-1);
    }

    if (sscanf(in_line, " %c %c ",
               &temp_data1,
               &temp_data2) != 2)
    {
        printf("\n\n ERROR READING SSLAGCF.DAT !!");
        printf("\n\n ERROR !! Wrong format for line %d. \n", i+1);
        exit(-1);
    }
    map1[i] = temp_data1;
    map1[i+offset] = temp_data2;
}
fclose(data);
return;
}

```

```

/*

```

```

.....

READ_SS_PLAG_FILE: READS Side To Side Position
LAG COEFICIEFNTS FROM Disk

```

```

.....
*/

```

```

void read_ss_plag_fil(map1)
float *map1;

```

```

{
    int i;
    int j=40;

```

```

int offset;
float temp_data1;
float temp_data2;
char in_line[49];
FILE *data;

if ((data = fopen("ssplagcf.dat", "r")) == NULL)
{
    printf("\n\t ERROR !! Could not open data file SSPLAGCF.DAT\n");
    exit(-1);
}

offset = SS_PLAG_COEFS/2;

for (i=0; i<offset; ++i)
{
    if (fgets(in_line, j, data) == NULL)
    {
        printf("\n\t ERROR READING SSPLAGCF.DAT !!");
        printf("\n\t Could not read line %d . \n", i+1);
        exit(-1);
    }

    if (sscanf(in_line, " %e %e ",
               &temp_data1,
               &temp_data2) != 2)
    {
        printf("\n\t ERROR READING SSPLAGCF.DAT !!");
        printf("\n\t ERROR !! Wrong format for line %d. \n", i+1);
        exit(-1);
    }
    map1[i] = temp_data1;
    map1[i+ offset] = temp_data2;
}
fclose(data);
return;
}

```

```

/*

```

.....

READ_FB_STATE_FILE: READS Front to Back STATE COEFICIEFNTS FROM DISK

.....

```

*/

```

```

void read_fb_state_fil(map1)
float *map1;

{
    int i;
    int j=80;
    int offset;
    float temp_data1;

```

```

float temp_data2;
float temp_data3;
float temp_data4;
char in_line[99];
FILE *data;

if ((data = fopen ("fbstatcf.dat", "r")) == NULL )
{
    printf ("\n\n ERROR !! Could not open data file FBSTATCF.DAT\n");
    exit (-1);
}

offset = FB_STATE_COEFS/4;

for (i=0; i<offset; ++i)
{
    if (fgets (in_line, j, data) == NULL)
    {
        printf ("\n\n ERROR READING FBSTATCF.DAT !!");
        printf ("\n\n Could not read line %d . \n", i+1);
        exit (-1);
    }

    if (sscanf (in_line, " %e %e %e %e ",
        &temp_data1,
        &temp_data2,
        &temp_data3,
        &temp_data4) != 4)
    {
        printf ("\n\n ERROR READING FBSTATCF.DAT !!");
        printf ("\n\n ERROR !! Wrong format for line %d. \n", i+1);
        exit (-1);
    }
    map1[i] = temp_data1;
    map1[i+( offset)] = temp_data2;
    map1[i+(2*offset)] = temp_data3;
    map1[i+(3*offset)] = temp_data4;
}
fclose(data);
return;
}

/*
.....

READ_FB_PIDM_FILE: READS Front To Back PIDM COEFFICIENTS FROM DISK

.....
*/
void read_fb_pidm_fil(map1)
float *map1;

{
    int i;

```

```

int j=80;
int offset;
float temp_data1;
float temp_data2;
float temp_data3;
float temp_data4;
char in_line[99];
FILE *data;

if (( data = fopen ("fbpidmcf.dat","r")) == NULL )
{
    printf ("\n\t ERROR !! Could not open data file FBPIDMCF.DAT\n");
    exit (-1);
}

offset = FB_PIDM_COEFS/4;

for (i=0; i<offset; ++i)
{
    if (fgets (in_line, j, data) == NULL)
    {
        printf ("\n\t ERROR READING FBPIDMCF.DAT !!");
        printf ("\n\t Could not read line %d . \n",i+1);
        exit (-1);
    }

    if (sscanf (in_line, " %c %c %c %c ",
        &temp_data1,
        &temp_data2,
        &temp_data3,
        &temp_data4) != 4)
    {
        printf ("\n\t ERROR READING FBPIDMCF.DAT !!");
        printf ("\n\t ERROR !! Wrong format for line %d . \n",i+1);
        exit (-1);
    }

    map1[i] = temp_data1;
    map1[i+( offset)] = temp_data2;
    map1[i+(2*offset)] = temp_data3;
    map1[i+(3*offset)] = temp_data4;
}
fclose(data);
return;
}

/*
.....

READ_FB_LAG_FILE: READS Front To Back LAG COEFICIEFNTS FROM DISK
.....
*/
void read_fb_lag_fil(map1)

```

```

float *map1;

{
    int i;
    int j=80;
    int offset;
    float temp_data1;
    float temp_data2;
    float temp_data3;
    float temp_data4;
    char in_line[99];
    FILE *data;

    if ((data = fopen ("fblagcf.dat", "r")) == NULL )
    {
        printf ("\n\t ERROR !! Could not open data file FBLAGCF.DAT\n");
        exit (-1);
    }

    offset = FB_LAG_COEFS/4;

    for (i=0; i<offset; ++i)
    {
        if (fgetc (in_line, j, data) == NULL)
        {
            printf ("\n\t ERROR READING FBLAGCF.DAT !!");
            printf ("\n\t Could not read line %d . \n", i+1);
            exit (-1);
        }

        if (sscanf (in_line, " %c %c %c %c ",
            &temp_data1,
            &temp_data2,
            &temp_data3,
            &temp_data4) != 4)
        {
            printf ("\n\t ERROR READING FBLAGCF.DAT !!");
            printf ("\n\t ERROR !! Wrong format for line %d. \n", i+1);
            exit (-1);
        }
        map1[i] = temp_data1;
        map1[i+( offset)] = temp_data2;
        map1[i+(2*offset)] = temp_data3;
        map1[i+(3*offset)] = temp_data4;
    }
    fclose(data);
    return;
}

/-----
READ_FB_PLAG_FILE: READS Front To Back Position LAG COEFFICIENTS FROM
DISK

```

```

.....
*/
void read_fb_plag_fil(map1)
float,*map1;

{
    int i;
    int j=80;
    int offset;
    float temp_data1;
    float temp_data2;
    float temp_data3;
    float temp_data4;
    char in_line[99];
    FILE *data;

    if (( data = fopen ("fbplagcf.dat","r")) == NULL )
    {
        printf ("\n\n ERROR !! Could not open data file FBPLAGCF.DAT\n");
        exit (-1);
    }

    offset = FB_PLAG_COEFS/4;

    for (i=0; i<offset; ++i)
    {
        if (fgets (in_line, j, data) == NULL)
        {
            printf ("\n\n ERROR READING FBPLAGCF.DAT !!");
            printf ("\n\n Could not read line %d . \n",i+1);
            exit (-1);
        }

        if (sscanf (in_line, " %c %c %c %c ",
            &temp_data1,
            &temp_data2,
            &temp_data3,
            &temp_data4) != 4)
        {
            printf ("\n\n ERROR READING FBPLAGCF.DAT !!");
            printf ("\n\n ERROR !! Wrong format for line %d. \n",i+1);
            exit (-1);
        }
        map1[i] = temp_data1;
        map1[i+( offset)] = temp_data2;
        map1[i+(2 *offset)] = temp_data3;
        map1[i+(3 *offset)] = temp_data4;
    }
    fclose(data);
    return;
}

/*

```

```

.....
      READ_GLAG_FILE: READS Gap Filter COEFFICIENTS FROM DISK
.....
*/
void read_glag_fil(map1)
float *map1;

{
    int i;
    int j=80;
    int offset;
    float temp_data1;
    float temp_data2;
    float temp_data3;
    float temp_data4;
    char in_line[99];
    FILE *data;

    if ((data = fopen ("glagcf.dat","r")) == NULL )
    {
        printf ("\n\t ERROR !! Could not open data file GLAGCF.DAT\n");
        exit (-1);
    }

    offset = GLAG_COEFS/4;

    for (i=0; i<offset; ++i)
    {
        if (fgets (in_line, j, data) == NULL)
        {
            printf ("\n\t ERROR READING GLAGCF.DAT !!");
            printf ("\n\t Could not read line %d . \n", i+1);
            exit (-1);
        }

        if (sscanf (in_line, " %e %e %e %e ",
                    &temp_data1,
                    &temp_data2,
                    &temp_data3,
                    &temp_data4) != 4)
        {
            printf ("\n\t ERROR READING GLAGCF.DAT !!");
            printf ("\n\t ERROR !! Wrong format for line %d. \n", i+1);
            exit (-1);
        }
        map1[i] = temp_data1;
        map1[i+( offset)] = temp_data2;
        map1[i+(2*offset)] = temp_data3;
        map1[i+(3*offset)] = temp_data4;
    }
    fclose(data);
    return;
}

```

```

}

/*
.....

INIT_DAS20: Initializes the A to D card for interrupt
and DMA level.

.....
*/
void init_das20()
{
    Params[0] = DAS_ADDR;           // DAS-20 base address
    Params[1] = 5;                  // interrupt level
    Params[2] = 1;                  // DMA level switch
                                   // selectable on board

    Flag = das20(0, Params);
    if (Flag != 0)                  // Error if Flag not 0
    {
        printf("\nMode 0: Error in DAS20 initialization. FLAG=%d\n", Flag);
        exit(-1);
    }
    return;
}

/*
.....

INIT_DDA06: Intitalizes all analog outputs to 0 volts.

.....
*/
void init_dda06()
{
    int i;

    for (i=0; i<=5; i++)
    {
        outportb (DDA06_ADDR + 2*i, 0);
        outportb (DDA06_ADDR + 2*i + 1, 8);

        outportb (DDA06_ADDR1 + 2*i, 0);
        outportb (DDA06_ADDR1 + 2*i + 1, 8);
    }
    return;
}

/*
.....

DEFINE_QUEUE: Defines the A/D queue for Triggered samples

```

```

.....
*/
void define_queue(chan,gain,num_chans)
int chan[];
int gain[];
int num_chans;
{
    int i=0;

    if (num_chans<2 || num_chans>2048)
    {
        printf("\n Error in DEFINE QUEUE. Wrong number of channels");
        exit(-1);
    }

    Params[0] = chan[0];
    Params[1] = gain[0];
    Params[2] = 2;           // First Queue entry

    Flag = das20(1, Params);
    if (Flag != 0)           // Error if flag not 0
    {
        printf("\nMode 1: Error in setting First queue element. Flag=%d\n",
                Flag);
        exit(-1);
    }
    i++;

    while (i+1<num_chans)
    {
        Params[0] = chan[i];
        Params[1] = gain[i];
        Params[2] = 0;       // Next Queue entry

        Flag = das20(1, Params);
        if (Flag != 0)       // Error if flag not 0
        {
            printf("\nMode 1: Error in setting %d queue element. Flag=%d\n",i,
                    Flag);
            exit(-1);
        }
        i++;
    }

    Params[0] = chan[i];
    Params[1] = gain[i];
    Params[2] = 1;          // Last Queue entry

    Flag = das20(1, Params);
    if (Flag != 0)          // Error if flag not 0
    {
        printf("\nMode 1: Error in setting Last queue element. Flag=%d\n",
                Flag);
    }
}

```

```

        exit(-1);
    }

    return ..
}

/*
.....

        SET_BLOCK_TIMER_FREQ: set programmable timer for mode 27 scan
        .....
*/
void set_block_timer_freq(freq)
int freq;
{
    Params[0] = 1000;           // counter 1
    Params[1] = 5000/freq;      // counter 2
    Flag = das20(25, Params);
    if (Flag != 0)               // Error if Flag not 0
    {
        printf("\nMode 25: Error setting up block timer. FLAG=%d\n", Flag);
        exit(-1);
    }
    return;
}

/*
.....

        ANALOG_OUT: Sends analog commands to the electromagnet
                    amplifier from the DDA06 card.
        .....
*/
void analog_out(ch, data)
int ch;
int data;
{
    int zero_offset = 2048;
    int max_analog_cmd=4095;
    int min_analog_cmd=0;
    int high_byte;
    int low_byte;

    data = data + zero_offset;
    if (data>=max_analog_cmd)
    {
        data = max_analog_cmd;
    }
    if (data <= min_analog_cmd)
    {
        data = min_analog_cmd;
    }
    high_byte = data/256;

```

```

    low_byte = data - high_byte * 256;

    outportb (DDA06_ADDR + 2 * ch, low_byte);
    outportb (DDA06_ADDR + 2 * ch + 1, high_byte);

    return;
}

```

```

/*
.....

```

ANALOG_OUT1: Sends analog outputs to BNCs for diagnostic purposes.

```

.....
*/
void analog_out1(ch, data)
int ch;
int data;
{
    int zero_offset = 2048;
    int max_analog_cmd=4095;
    int min_analog_cmd=0;
    int high_byte;
    int low_byte;

    data = data + zero_offset;
    if (data>max_analog_cmd)
    {
        data = max_analog_cmd;
    }
    if (data <= min_analog_cmd)
    {
        data = min_analog_cmd;
    };
    high_byte = data/256;
    low_byte = data - high_byte * 256;

    outportb (DDA06_ADDR1 + 2 * ch, low_byte);
    outportb (DDA06_ADDR1 + 2 * ch + 1, high_byte);

    return;
}

```

```

/*
.....

```

SS_CONTROL: PID control for Side To Side axes.

```

.....
*/
void ss_control(out,bss_gap_err,iss_gap_err,
               bss_accel,iss_accel,
               state_coefs,pidm_coefs,

```

```

lag_coefs,plag_coefs,
i)
float *out;
float bss_gap_err;
float lss_gap_err;
float bss_accel;
float lss_accel;
float state_coefs[];
float pidm_coefs[];
float lag_coefs[];
float plag_coefs[];
int i;
{
    static double x1b ,x1t;
    static double x1bo ,x1to;
    static double x2b ,x2t;
    static double x2bo ,x2to;
    // static double x3b ,x3t;
    // static double x3bo ,x3to;
    static double x3ab ,x3at;
    static double x3abo,x3ato;
    static double x3bb ,x3bt;
    static double x3bbo,x3bto;
    static double x3cb ,x3ct;
    static double x3cbo,x3cto;
    static double x3db ,x3dt;
    static double x3dbo,x3dto;
    static double x4b ,x4t;
    static double x4bo ,x4to;
    static double x5b ,x5t;
    static double x5bo ,x5to;
    static double x6b ,x6t;
    static double x6bo ,x6to;
    static double x7ab ,x7at;
    static double x7abo,x7ato;
    static double x7bb ,x7bt;
    static double x7bbo,x7bto;
    static double x7cb ,x7ct;
    static double x7cbo,x7cto;
    static double x7db ,x7dt;
    static double x7dbo,x7dto;
    static double x8b ,x8t;
    static double x8bo ,x8to;
    static double x9b ,x9t;
    static double x9bo ,x9to;

    static float bss_gap_err_old;
    static float lss_gap_err_old;

    static float bss_accel_old;
    static float lss_accel_old;

    static float u_bot_des_old;
    static float u_top_des_old;

```

```

float u_bot_des;
float u_top_des;

static float bss_gap_err_f;
static float tss_gap_err_f;

if (i == 0) // Initialize States
{
    x1b = 0.0;
    x1t = 0.0;
    x2b = 0.0;
    x2t = 0.0;
    // x3b = 0.0;
    // x3t = 0.0;
    x3ab = 0.0;
    x3at = 0.0;
    x3bb = 0.0;
    x3bt = 0.0;
    x3cb = 0.0;
    x3ct = 0.0;
    x3db = 0.0;
    x3dt = 0.0;
    x4b = 0.0;
    x4t = 0.0;
    x5b = 0.0;
    x5t = 0.0;
    x6b = 0.0;
    x6t = 0.0;
    x7ab = 0.0;
    x7at = 0.0;
    x7bb = 0.0;
    x7bt = 0.0;
    x7cb = 0.0;
    x7ct = 0.0;
    x7db = 0.0;
    x7dt = 0.0;
    x8b = 0.0;
    x8t = 0.0;
    x9b = 0.0;
    x9t = 0.0;
    bss_accel_old = bss_accel;
    tss_accel_old = tss_accel;
    bss_gap_err_old = bss_gap_err;
    tss_gap_err_old = tss_gap_err;
    u_bot_des_old = 0;
    u_top_des_old = 0;
    *(out) = 0.0;
    *(out+1) = 0.0;

    return;
}
else

```

```

{
    x1bo = x1b;
    x1to = x1t;
    x2bo = x2b;
    x2to = x2t;
    // x3bo = x3b;
    // x3to = x3t;
    x3abo = x3ab;
    x3ato = x3at;
    x3bbo = x3bb;
    x3bto = x3bt;
    x3cbo = x3cb;
    x3cto = x3ct;
    x3dbo = x3db;
    x3dto = x3dt;
    x4bo = x4b;
    x4to = x4t;
    x5bo = x5b;
    x5to = x5t;
    x6bo = x6b;
    x6to = x6t;
    x7abo = x7ab;
    x7ato = x7at;
    x7bbo = x7bb;
    x7bto = x7bt;
    x7cbo = x7cb;
    x7cto = x7ct;
    x7dbo = x7db;
    x7dto = x7dt;
    x8bo = x8b;
    x8to = x8t;
    x9bo = x9b;
    x9to = x9t;

    // Integrate Position signal for high static stiffness
    x1b = state_coefs[0] * x1bo + state_coefs[1] * bss_gap_err;
    x1t = state_coefs[20] * x1to + state_coefs[21] * tss_gap_err;

    // Integrate accelerometer signal for damping
    x2b = state_coefs[2] * x2bo + state_coefs[3] * bss_accel;
    x2t = state_coefs[22] * x2to + state_coefs[23] * tss_accel;

    // Add lags to integrated accelerometer signal
    x3ab = lag_coefs[0] * x3abo + lag_coefs[1] * x3bbo +
        lag_coefs[2] * x3cbo + lag_coefs[3] * x2bo;
    x3bb = lag_coefs[4] * x3abo + lag_coefs[5] * x3bbo +
        lag_coefs[6] * x3cbo + lag_coefs[7] * x2bo;
    x3cb = lag_coefs[8] * x3abo + lag_coefs[9] * x3bbo +
        lag_coefs[10] * x3cbo + lag_coefs[11] * x2bo;

```

```
x3db = lag_coefs[12] * x3ab + lag_coefs[13] * x3bb +
      lag_coefs[14] * x3cb + lag_coefs[15] * x2b;
```

```
x3at = lag_coefs[20] * x3ao + lag_coefs[21] * x3bo +
      lag_coefs[22] * x3co + lag_coefs[23] * x2o;
```

```
x3bt = lag_coefs[24] * x3ao + lag_coefs[25] * x3bo +
      lag_coefs[26] * x3co + lag_coefs[27] * x2o;
```

```
x3ct = lag_coefs[28] * x3ao + lag_coefs[29] * x3bo +
      lag_coefs[30] * x3co + lag_coefs[31] * x2o;
```

```
x3dt = lag_coefs[32] * x3at + lag_coefs[33] * x3bt +
      lag_coefs[34] * x3ct + lag_coefs[35] * x2t;
```

```
// High pass filter integrated accelerometer to remove dc offset
```

```
x4b = state_coefs[4] * x4bo + state_coefs[5] * x5bo +
      state_coefs[6] * x3dbo;
```

```
x5b = state_coefs[7] * x4bo + state_coefs[8] * x5bo +
      state_coefs[9] * x3dbo;
```

```
x6b = state_coefs[10] * x4b + state_coefs[11] * x5b +
      state_coefs[12] * x3db;
```

```
x4t = state_coefs[24] * x4to + state_coefs[25] * x5to +
      state_coefs[26] * x3dto;
```

```
x5t = state_coefs[27] * x4to + state_coefs[28] * x5to +
      state_coefs[29] * x3dto;
```

```
x6t = state_coefs[30] * x4t + state_coefs[31] * x5t +
      state_coefs[32] * x3dt;
```

```
// Add lags to position signal
```

```
x7ab = plag_coefs[0] * x7abo + plag_coefs[1] * x7bbo +
      plag_coefs[2] * x7cbo + plag_coefs[3] * bss_gap_err_old;
```

```
x7bb = plag_coefs[4] * x7abo + plag_coefs[5] * x7bbo +
      plag_coefs[6] * x7cbo + plag_coefs[7] * bss_gap_err_old;
```

```
x7cb = plag_coefs[8] * x7abo + plag_coefs[9] * x7bbo +
      plag_coefs[10] * x7cbo + plag_coefs[11] * bss_gap_err_old;
```

```
x7db = plag_coefs[12] * x7ab + plag_coefs[13] * x7bb +
      plag_coefs[14] * x7cb + plag_coefs[15] * bss_gap_err;
```

```
x7at = plag_coefs[20] * x7ato + plag_coefs[21] * x7bto +
      plag_coefs[22] * x7cto + plag_coefs[23] * tss_gap_err_old;
```

```
x7bt = plag_coefs[24] * x7ato + plag_coefs[25] * x7bto +
      plag_coefs[26] * x7cto + plag_coefs[27] * tss_gap_err_old;
```

```

x7ct = plag_coefs[28] * x7at + plag_coefs[29] * x7bt +
      plag_coefs[30] * x7ct + plag_coefs[31] * tss_gap_err_old;

x7dt = plag_coefs[32] * x7at - plag_coefs[33] * x7bt +
      plag_coefs[34] * x7ct + plag_coefs[35] * tss_gap_err;

bss_gap_err_f = x7db;
tss_gap_err_f = x7dt;

// Compute Output
// Bottom Side To Side Actuator
// Integrated position for static offset loads
// plus integrated accelerometer for damping

u_bot_des = pidm_coefs[0] * x1b + pidm_coefs[1] * x1t +
            pidm_coefs[2] * x6b + pidm_coefs[3] * x6t +

// Proportional gain on position for spring rate
            pidm_coefs[4] * bss_gap_err_f +
            pidm_coefs[5] * tss_gap_err_f;

u_bot_des = u_bot_des * pidm_coefs[6];

// Top Side To Side Actuator
// Integrated position for static offset loads
// plus integrated accelerometer for damping

u_top_des = pidm_coefs[10] * x1b + pidm_coefs[11] * x1t +
            pidm_coefs[12] * x6b + pidm_coefs[13] * x6t +

// Proportional gain on position for spring rate
            pidm_coefs[14] * bss_gap_err_f +
            pidm_coefs[15] * tss_gap_err_f;

u_top_des = u_top_des * pidm_coefs[16];

*(out) = u_bot_des;
*(out+1) = u_top_des;

bss_accel_old = bss_accel;
tss_accel_old = tss_accel;
bss_gap_err_old = bss_gap_err;
tss_gap_err_old = tss_gap_err;
u_bot_des_old = u_bot_des;
u_top_des_old = u_top_des;

return;

}
}

```

.....

FU_CONTROL: PID control for Front To Back axes.

.....

```

*/
void fb_control(out,blfb_gap_err,brfb_gap_err,
               tlfb_gap_err,trfb_gap_err,
               blfb_accel,brfb_accel,
               tlfb_accel,trfb_accel,
               state_coefs,pidm_coefs,lag_coefs,plag_coefs,i)
float *out;
float blfb_gap_err;
float brfb_gap_err;
float tlfb_gap_err;
float trfb_gap_err;
float blfb_accel;
float brfb_accel;
float tlfb_accel;
float trfb_accel;
float state_coefs[];
float pidm_coefs[];
float lag_coefs[];
float plag_coefs[];
int i;
{
    static double x1bl ,x1br ,x1tl ,x1tr ;
    static double x1blo ,x1bro ,x1tlo ,x1tro ;
    static double x2bl ,x2br ,x2tl ,x2tr ;
    static double x2blo ,x2bro ,x2tlo ,x2tro ;
    // static double x3bl ,x3br ,x3tl ,x3tr ;
    // static double x3blo ,x3bro ,x3tlo ,x3tro ;
    static double x3bla ,x3bra ,x3tla ,x3tra ;
    static double x3bloa ,x3broa ,x3tloa ,x3troa ;
    static double x3blb ,x3brb ,x3tlb ,x3trb ;
    static double x3blbo ,x3brbo ,x3tlbo ,x3trbo ;
    static double x3blc ,x3brc ,x3tlc ,x3trc ;
    static double x3blco ,x3brco ,x3tlco ,x3trco ;
    static double x3bld ,x3brd ,x3tld ,x3trd ;
    static double x3bldo ,x3brdo ,x3tldo ,x3trdo ;
    static double x4bl ,x4br ,x4tl ,x4tr ;
    static double x4blo ,x4bro ,x4tlo ,x4tro ;
    static double x5bl ,x5br ,x5tl ,x5tr ;
    static double x5blo ,x5bro ,x5tlo ,x5tro ;
    static double x6bl ,x6br ,x6tl ,x6tr ;
    static double x6blo ,x6bro ,x6tlo ,x6tro ;
    static double x7bl ,x7br ,x7tl ,x7tr ;
    static double x7blo ,x7bro ,x7tlo ,x7tro ;
    static double x8bl ,x8br ,x8tl ,x8tr ;
    static double x8blo ,x8bro ,x8tlo ,x8tro ;
    static double x9bl ,x9br ,x9tl ,x9tr ;
    static double x9blo ,x9bro ,x9tlo ,x9tro ;
    static float blfb_gap_err_old;

```

```

static float blfb_gap_err_old;
static float tlfb_gap_err_old;
static float trfb_gap_err_old;
static float blfb_accel_old;
static float brfb_accel_old;
static float tlfb_accel_old;
static float trfb_accel_old;

```

```

static float u_bl_des_old;
static float u_br_des_old;
static float u_tl_des_old;
static float u_tr_des_old;
float u_bl_des;
float u_br_des;
float u_tl_des;
float u_tr_des;
float blfb_gap_err_f;
float brfb_gap_err_f;
float tlfb_gap_err_f;
float trfb_gap_err_f;

```

```

if (i == 0) // Initialize States
{

```

```

    x1bl = 0.0;
    x1br = 0.0;
    x1tl = 0.0;
    x1tr = 0.0;
    x2bl = 0.0;
    x2br = 0.0;
    x2tl = 0.0;
    x2tr = 0.0;
//    x3bl = 0.0;
//    x3br = 0.0;
//    x3tl = 0.0;
//    x3tr = 0.0;
    x3bla = 0.0;
    x3blb = 0.0;
    x3blc = 0.0;
    x3bld = 0.0;
    x3bra = 0.0;
    x3brb = 0.0;
    x3brc = 0.0;
    x3brd = 0.0;
    x3tla = 0.0;
    x3tlb = 0.0;
    x3tlc = 0.0;
    x3tld = 0.0;
    x3tra = 0.0;
    x3trb = 0.0;
    x3trc = 0.0;
    x3trd = 0.0;
    x4bl = 0.0;

```

```

x4br  = 0.0;
x4tl  = 0.0;
x4tr  = 0.0;
x5bl  = 0.0;
x5br  = 0.0;
x5tl  = 0.0;
x5tr  = 0.0;
x6bl  = 0.0;
x6br  = 0.0;
x6tl  = 0.0;
x6tr  = 0.0;
x7bl  = 0.0;
x7br  = 0.0;
x7tl  = 0.0;
x7tr  = 0.0;
x8bl  = 0.0;
x8br  = 0.0;
x8tl  = 0.0;
x8tr  = 0.0;
x9bl  = 0.0;
x9br  = 0.0;
x9tl  = 0.0;
x9tr  = 0.0;
blfb_accel_old = blfb_accel;
brfb_accel_old = brfb_accel;
tlfb_accel_old = tlfb_accel;
trfb_accel_old = trfb_accel;
blfb_gap_err_old = blfb_gap_err;
brfb_gap_err_old = brfb_gap_err;
tlfb_gap_err_old = tlfb_gap_err;
trfb_gap_err_old = trfb_gap_err;
u_bl_des_old = 0.0;
u_br_des_old = 0.0;
u_tl_des_old = 0.0;
u_tr_des_old = 0.0;
*(out) = 0.0;
*(out+1) = 0.0;
*(out+2) = 0.0;
*(out+3) = 0.0;

return;
}
else
{
x1blo  = x1bl;
x1bro  = x1br;
x1tlo  = x1tl;
x1tro  = x1tr;
x2blo  = x2bl;
x2bro  = x2br;
x2tlo  = x2tl;
x2tro  = x2tr;
// x3blo  = x3bl;
x3blo  = x3bla;

```

```

x3blo = x3blb;
x3ble = x3ble;
x3bld = x3bld;
// x3bro = x3br;
x3bra = x3bra;
x3brb = x3brb;
x3bre = x3bre;
x3brd = x3brd;
// x3tlo = x3tl;
x3tla = x3tla;
x3tlb = x3tlb;
x3tle = x3tle;
x3tld = x3tld;
// x3tro = x3tr;
x3tra = x3tra;
x3trb = x3trb;
x3tre = x3tre;
x3trd = x3trd;
x4blo = x4bl;
x4bro = x4br;
x4tlo = x4tl;
x4tro = x4tr;
x5blo = x5bl;
x5bro = x5br;
x5tlo = x5tl;
x5tro = x5tr;
x6blo = x6bl;
x6bro = x6br;
x6tlo = x6tl;
x6tro = x6tr;
x7blo = x7bl;
x7bro = x7br;
x7tlo = x7tl;
x7tro = x7tr;
x8blo = x8bl;
x8bro = x8br;
x8tlo = x8tl;
x8tro = x8tr;
x9blo = x9bl;
x9bro = x9br;
x9tlo = x9tl;
x9tro = x9tr;

// Integrate Position signal for high static stiffness

x1bl = state_coefs[0] * x1blo + state_coefs[1] * blfb_gap_err;
x1br = state_coefs[20] * x1bro + state_coefs[21] * brfb_gap_err;
x1tl = state_coefs[40] * x1tlo + state_coefs[41] * tlfb_gap_err;
x1tr = state_coefs[60] * x1tro + state_coefs[61] * trfb_gap_err;

// Integrate accelerometer signal for damping

x2bl = state_coefs[2] * x2blo + state_coefs[3] * blfb_accel;
x2br = state_coefs[22] * x2bro + state_coefs[23] * brfb_accel;

```

```

x2tl = state_coefs[42] * x2tlo + state_coefs[43] * tllb_accel;
x2tr = state_coefs[62] * x2tro + state_coefs[63] * trfb_accel;

```

```
// Add lags to integrated accelerometer signal
```

```

x3bla = lag_coefs[0] * x3blo + lag_coefs[1] * x3blbo +
        lag_coefs[2] * x3blco + lag_coefs[3] * x2blo;

```

```

x3blb = lag_coefs[4] * x3blo + lag_coefs[5] * x3blbo +
        lag_coefs[6] * x3blco + lag_coefs[7] * x2blo;

```

```

x3blc = lag_coefs[8] * x3blo + lag_coefs[9] * x3blbo +
        lag_coefs[10] * x3blco + lag_coefs[11] * x2blo;

```

```

x3bld = lag_coefs[12] * x3bla + lag_coefs[13] * x3blb +
        lag_coefs[14] * x3blc + lag_coefs[15] * x2bl;

```

```

x3bra = lag_coefs[20] * x3brao + lag_coefs[21] * x3brbo +
        lag_coefs[22] * x3brco + lag_coefs[23] * x2bro;

```

```

x3brb = lag_coefs[24] * x3brao + lag_coefs[25] * x3brbo +
        lag_coefs[26] * x3brco + lag_coefs[27] * x2bro;

```

```

x3brc = lag_coefs[28] * x3brao + lag_coefs[29] * x3brbo +
        lag_coefs[30] * x3brco + lag_coefs[31] * x2bro;

```

```

x3brd = lag_coefs[32] * x3bra + lag_coefs[33] * x3brb +
        lag_coefs[34] * x3brc + lag_coefs[35] * x2br;

```

```

x3tla = lag_coefs[40] * x3tlo + lag_coefs[41] * x3tlbo +
        lag_coefs[42] * x3tlco + lag_coefs[43] * x2tlo;

```

```

x3tlb = lag_coefs[44] * x3tlo + lag_coefs[45] * x3tlbo +
        lag_coefs[46] * x3tlco + lag_coefs[47] * x2tlo;

```

```

x3tlc = lag_coefs[48] * x3tlo + lag_coefs[49] * x3tlbo +
        lag_coefs[50] * x3tlco + lag_coefs[51] * x2tlo;

```

```

x3tld = lag_coefs[52] * x3tla + lag_coefs[53] * x3tlb +
        lag_coefs[54] * x3tlc + lag_coefs[55] * x2tl;

```

```

x3tra = lag_coefs[60] * x3trao + lag_coefs[61] * x3trbo +
        lag_coefs[62] * x3trco + lag_coefs[63] * x2tro;

```

```

x3trb = lag_coefs[64] * x3trao + lag_coefs[65] * x3trbo +
        lag_coefs[66] * x3trco + lag_coefs[67] * x2tro;

```

```

x3trc = lag_coefs[68] * x3trao + lag_coefs[69] * x3trbo +
        lag_coefs[70] * x3trco + lag_coefs[71] * x2tro;

```

```

x3trd = lag_coefs[72] * x3tra + lag_coefs[73] * x3trb +

```

```
lag_coefs[74] * x3trc + lag_coefs[75] * x2tr;
```

```
//High pass filter integrated accelerometer to remove dc offset
```

```
x4bl = state_coefs[4] * x4blo + state_coefs[5] * x5blo +  
state_coefs[6] * x3blo;
```

```
x5bl = state_coefs[7] * x4blo + state_coefs[8] * x5blo +  
state_coefs[9] * x3blo;
```

```
x6bl = state_coefs[10] * x4bl + state_coefs[11] * x5bl +  
state_coefs[12] * x3bl;
```

```
x4br = state_coefs[24] * x4bro + state_coefs[25] * x5bro +  
state_coefs[26] * x3bro;
```

```
x5br = state_coefs[27] * x4bro + state_coefs[28] * x5bro +  
state_coefs[29] * x3bro;
```

```
x6br = state_coefs[30] * x4br + state_coefs[31] * x5br +  
state_coefs[32] * x3br;
```

```
x4tl = state_coefs[44] * x4tlo + state_coefs[45] * x5tlo +  
state_coefs[46] * x3tlo;
```

```
x5tl = state_coefs[47] * x4tlo + state_coefs[48] * x5tlo +  
state_coefs[49] * x3tlo;
```

```
x6tl = state_coefs[50] * x4tl + state_coefs[51] * x5tl +  
state_coefs[52] * x3tl;
```

```
x4tr = state_coefs[64] * x4tro + state_coefs[65] * x5tro +  
state_coefs[66] * x3tro;
```

```
x5tr = state_coefs[67] * x4tro + state_coefs[68] * x5tro +  
state_coefs[69] * x3tro;
```

```
x6tr = state_coefs[70] * x4tr + state_coefs[71] * x5tr +  
state_coefs[72] * x3tr;
```

```
x7bl = plag_coefs[0] * x7blo + plag_coefs[1] * x8blo +  
plag_coefs[2] * blfb_gap_err_old;
```

```
x8bl = plag_coefs[3] * x7blo + plag_coefs[4] * x8blo +  
plag_coefs[5] * blfb_gap_err_old;
```

```
x9bl = plag_coefs[6] * x7bl + plag_coefs[7] * x8bl +  
plag_coefs[8] * blfb_gap_err;
```

```
x7br = plag_coefs[10] * x7bro + plag_coefs[11] * x8bro +  
plag_coefs[12] * brfb_gap_err_old;
```

```
x8br = plag_coefs[13] * x7br + plag_coefs[14] * x8br +
      plag_coefs[15] * brfb_gap_err_old;
```

```
x9br = plag_coefs[16] * x7br + plag_coefs[17] * x8br +
      plag_coefs[18] * brfb_gap_err;
```

```
x7tl = plag_coefs[20] * x7tl + plag_coefs[21] * x8tl +
      plag_coefs[22] * tlfb_gap_err_old;
```

```
x8tl = plag_coefs[23] * x7tl + plag_coefs[24] * x8tl +
      plag_coefs[25] * tlfb_gap_err_old;
```

```
x9tl = plag_coefs[26] * x7tl + plag_coefs[27] * x8tl +
      plag_coefs[28] * tlfb_gap_err;
```

```
x7tr = plag_coefs[30] * x7tr + plag_coefs[31] * x8tr +
      plag_coefs[32] * trfb_gap_err_old;
```

```
x8tr = plag_coefs[33] * x7tr + plag_coefs[34] * x8tr +
      plag_coefs[35] * trfb_gap_err_old;
```

```
x9tr = plag_coefs[36] * x7tr + plag_coefs[37] * x8tr +
      plag_coefs[38] * trfb_gap_err;
```

```
blfb_gap_err_f = x9bl;
```

```
brfb_gap_err_f = x9br;
```

```
tlfb_gap_err_f = x9tl;
```

```
trfb_gap_err_f = x9tr;
```

```
// Compute output
```

```
// BOTTOM LEFT FRONT TO BACK ACTUATOR
```

```
// Integrated Position for Static Offset Loads plus
```

```
// Integrated accelerometer for damping
```

```
u_bl_des = pidm_coefs[0] * x1bl + pidm_coefs[1] * x1br +
           pidm_coefs[2] * x1tl + pidm_coefs[3] * x1tr +
```

```
// Lagged accelerometer for roll off
```

```
pidm_coefs[4] * x6bl + pidm_coefs[5] * x6br +
pidm_coefs[6] * x6tl + pidm_coefs[7] * x6tr +
```

```
// Proportional gain on position for spring rate
```

```
pidm_coefs[8] * blfb_gap_err_f +
pidm_coefs[9] * brfb_gap_err_f +
pidm_coefs[10] * tlfb_gap_err_f +
pidm_coefs[11] * trfb_gap_err_f;
```

```
u_bl_des = pidm_coefs[12] * u_bl_des;
```

// BOTTOM RIGHT FRONT TO BACK ACTUATOR

// Integrated Position for Static Offset Loads plus

// Integrated accelerometer for damping

$$u_{br_des} = pidm_coefs[20] * x1bl + pidm_coefs[21] * x1br + \\ pidm_coefs[22] * x1tl + pidm_coefs[23] * x1tr +$$

// Lagged accelerometer for roll off

$$pidm_coefs[24] * x6bl + pidm_coefs[25] * x6br + \\ pidm_coefs[26] * x6tl + pidm_coefs[27] * x6tr +$$

// Proportional gain on position for spring rate

$$pidm_coefs[28] * blfb_gap_err_f + \\ pidm_coefs[29] * brfb_gap_err_f + \\ pidm_coefs[30] * tlfb_gap_err_f + \\ pidm_coefs[31] * trfb_gap_err_f;$$

$$u_{br_des} = pidm_coefs[32] * u_{br_des};$$

// TOP LEFT FRONT TO BACK ACTUATOR

// Integrated Position for Static Offset Loads plus

// Integrated accelerometer for damping

$$u_{tl_des} = pidm_coefs[40] * x1bl + pidm_coefs[41] * x1br + \\ pidm_coefs[42] * x1tl + pidm_coefs[43] * x1tr +$$

// Lagged accelerometer for roll off

$$pidm_coefs[44] * x6bl + pidm_coefs[45] * x6br + \\ pidm_coefs[46] * x6tl + pidm_coefs[47] * x6tr +$$

// Proportional gain on position for spring rate

$$pidm_coefs[48] * blfb_gap_err_f + \\ pidm_coefs[49] * brfb_gap_err_f + \\ pidm_coefs[50] * tlfb_gap_err_f + \\ pidm_coefs[51] * trfb_gap_err_f;$$

$$u_{tl_des} = pidm_coefs[52] * u_{tl_des};$$

// TOP RIGHT FRONT TO BACK ACTUATOR

// Integrated Position for Static Offset Loads plus

// Integrated accelerometer for damping

$$u_{tr_des} = pidm_coefs[60] * x1bl + pidm_coefs[61] * x1br + \\ pidm_coefs[62] * x1tl + pidm_coefs[63] * x1tr +$$

// Lagged accelerometer for roll off

$$pidm_coefs[64] * x6bl + pidm_coefs[65] * x6br + \\ pidm_coefs[66] * x6tl + pidm_coefs[67] * x6tr +$$

// Proportional gain on position for spring rate

$$pidm_coefs[68] * blfb_gap_err_f + \\ pidm_coefs[69] * brfb_gap_err_f + \\ pidm_coefs[70] * tlfb_gap_err_f +$$

```

        pidm_coefs[71] * trfb_gap_err_f;
    u_tr_des = pidm_coefs[72] * u_tr_des;

    *(out) = u_bl_des;
    *(out+1) = u_br_des;
    *(out+2) = u_tl_des;
    *(out+3) = u_tr_des;
}

blfb_accel_old = blfb_accel;
brfb_accel_old = brfb_accel;
tlfb_accel_old = tlfb_accel;
trfb_accel_old = trfb_accel;
brfb_gap_err_old = brfb_gap_err;
blfb_gap_err_old = blfb_gap_err;
trfb_gap_err_old = trfb_gap_err;
tlfb_gap_err_old = tlfb_gap_err;
u_bl_des_old = u_bl_des;
u_br_des_old = u_br_des;
u_tl_des_old = u_tl_des;
u_tr_des_old = u_tr_des;

return;
}

/*
*****

SS GAP FILTER: THIRD ORDER POSITION FILTER FOR CURRENT CONTROL

*****
*/
void ss_gap_filter(out,bl_gap_mm,br_gap_mm,
                  tl_gap_mm,tr_gap_mm,
                  lag_coefs,i)
float *out;
float bl_gap_mm;
float br_gap_mm;
float tl_gap_mm;
float tr_gap_mm;
float lag_coefs[];
int i;
{
    static double xbla,xbra,xlla,xtra;
    static double xblao,xbrao,xllo,xtrao;
    static double xblb,xbrb,xllb,xtrb;
    static double xblbo,xbrbo,xllbo,xtrbo;
    static double xblc,xbrc,xllc,xtrc;
    static double xblco,xbrco,xllco,xtrco;
    static double xblld,xbrld,xllld,xtrld;
    static double xblldo,xbrldo,xllldo,xtrldo;
    static float bl_gap_mm_old;
    static float br_gap_mm_old;

```

```

static float tl_gap_mm_old;
static float tr_gap_mm_old;

if (i == 0) // Initialize States
{
    xbla = 0.0;
    xblb = 0.0;
    xblc = 0.0;
    xbl_d = 0.0;

    xbra = 0.0;
    xbrb = 0.0;
    xbrc = 0.0;
    xbrd = 0.0;

    xtl_a = 0.0;
    xtlb = 0.0;
    xtlc = 0.0;
    xtld = 0.0;

    xtra = 0.0;
    xtrb = 0.0;
    xtrc = 0.0;
    xtrd = 0.0;

    bl_gap_mm_old = bl_gap_mm;
    br_gap_mm_old = br_gap_mm;
    tl_gap_mm_old = tl_gap_mm;
    tr_gap_mm_old = tr_gap_mm;

    *(out) = bl_gap_mm;
    *(out+1) = br_gap_mm;
    *(out+2) = tl_gap_mm;
    *(out+3) = tr_gap_mm;

    return;
}
else
{
    xblao = xbla;
    xblbo = xblb;
    xblco = xblc;
    xbl_do = xbl_d;

    xbrao = xbra;
    xbrbo = xbrb;
    xbrco = xbrc;
    xbrdo = xbrd;

    xtlao = xtl_a;
    xtlbo = xtlb;
    xtlco = xtlc;
    xtldo = xtld;

```

```

xtrao = xtra;
xtrbo = xtrb;
xtrco = xtrc;
xtrdo = xtrd;

```

```

xbla = lag_coefs[0] * xblao + lag_coefs[1] * xblbo +
      lag_coefs[2] * xblco + lag_coefs[3] * bl_gap_mm_old;

```

```

xblb = lag_coefs[4] * xblao + lag_coefs[5] * xblbo +
      lag_coefs[6] * xblco + lag_coefs[7] * bl_gap_mm_old;

```

```

xblc = lag_coefs[8] * xblao + lag_coefs[9] * xblbo +
      lag_coefs[10] * xblco + lag_coefs[11] * bl_gap_mm_old;

```

```

xbl d = lag_coefs[12] * xbla + lag_coefs[13] * xblb +
      lag_coefs[14] * xblc + lag_coefs[15] * bl_gap_mm;

```

```

xbra = lag_coefs[20] * xbrao + lag_coefs[21] * xbrbo +
      lag_coefs[22] * xbrco + lag_coefs[23] * br_gap_mm_old;

```

```

xbrb = lag_coefs[24] * xbrao + lag_coefs[25] * xbrbo +
      lag_coefs[26] * xbrco + lag_coefs[27] * br_gap_mm_old;

```

```

xbrc = lag_coefs[28] * xbrao + lag_coefs[29] * xbrbo +
      lag_coefs[30] * xbrco + lag_coefs[31] * br_gap_mm_old;

```

```

xbrd = lag_coefs[32] * xbra + lag_coefs[33] * xbrb +
      lag_coefs[34] * xbrc + lag_coefs[35] * br_gap_mm;

```

```

xtla = lag_coefs[40] * xtlao + lag_coefs[41] * xtlibo +
      lag_coefs[42] * xtlico + lag_coefs[43] * tl_gap_mm_old;

```

```

xtlb = lag_coefs[44] * xtlao + lag_coefs[45] * xtlibo +
      lag_coefs[46] * xtlico + lag_coefs[47] * tl_gap_mm_old;

```

```

xtlc = lag_coefs[48] * xtlao + lag_coefs[49] * xtlibo +
      lag_coefs[50] * xtlico + lag_coefs[51] * tl_gap_mm_old;

```

```

xtld = lag_coefs[52] * xtla + lag_coefs[53] * xtlb +
      lag_coefs[54] * xtlc + lag_coefs[55] * tl_gap_mm;

```

```

xtra = lag_coefs[60] * xtrao + lag_coefs[61] * xtrbo +
      lag_coefs[62] * xtrco + lag_coefs[63] * tr_gap_mm_old;

```

```

xtrb = lag_coefs[64] * xtrao + lag_coefs[65] * xtrbo +
      lag_coefs[66] * xtrco + lag_coefs[67] * tr_gap_mm_old;

```

```

    xirc = lag_coefs[68] * xtrao + lag_coefs[69] * xtrbo +
           lag_coefs[70] * xirco + lag_coefs[71] * tr_gap_mm_old;

    xtrd = lag_coefs[72] * xtra + lag_coefs[73] * xtrb +
           lag_coefs[74] * xirc + lag_coefs[75] * tr_gap_mm;

    *(out) = xbld;
    *(out+1) = xbrd;
    *(out+2) = xild;
    *(out+3) = xtrd;

}

br_gap_mm_old = br_gap_mm;
bl_gap_mm_old = bl_gap_mm;
tr_gap_mm_old = tr_gap_mm;
tl_gap_mm_old = tl_gap_mm;

return;
}

/*
.....

FB GAP FILTER: THIRD ORDER POSITION FILTER FOR CURRENT CONTROL

.....
*/
void fb_gap_filter(out,bl_gap_mm,br_gap_mm,
                  tl_gap_mm,tr_gap_mm,
                  lag_coefs,i)
float *out;
float bl_gap_mm;
float br_gap_mm;
float tl_gap_mm;
float tr_gap_mm;
float lag_coefs[];
int i;
{
    static double xbla ,xbra ,xtla ,xta ;
    static double xblao,xbrao,xlao,xtrao;
    static double xblb ,xbrb ,xtlb ,xtrb ;
    static double xblbo,xbrbo,xlbo,xtrbo;
    static double xble ,xbre ,xtle ,xtre ;
    static double xblco,xbrco,xlco,xtrco;
    static double xbld ,xbrd ,xild ,xtrd ;
    static double xbldo,xbrdo,xldo,xtrdo;
    static float bl_gap_mm_old;
    static float br_gap_mm_old;
    static float tl_gap_mm_old;
    static float tr_gap_mm_old;

```

```

if (i == 0) // Initialize States
{
    xbla = 0.0;
    xblb = 0.0;
    xblc = 0.0;
    xbl d = 0.0;

    xbra = 0.0;
    xbrb = 0.0;
    xbrc = 0.0;
    xbrd = 0.0;

    xtla = 0.0;
    xtlb = 0.0;
    xtlc = 0.0;
    xtld = 0.0;

    xtra = 0.0;
    xtrb = 0.0;
    xtrc = 0.0;
    xtrd = 0.0;

    bl_gap_mm_old = bl_gap_mm;
    br_gap_mm_old = br_gap_mm;
    tl_gap_mm_old = tl_gap_mm;
    tr_gap_mm_old = tr_gap_mm;

    *(out+4) = bl_gap_mm;
    *(out+5) = br_gap_mm;
    *(out+6) = tl_gap_mm;
    *(out+7) = tr_gap_mm;

    return;
}
else
{
    xblao = xbla;
    xblbo = xblb;
    xblco = xblc;
    xbl d o = xbl d;

    xbrao = xbra;
    xbrto = xbrb;
    xbrco = xbrc;
    xbrd o = xbrd;

    xtla o = xtla;
    xtlbo = xtlb;
    xtlco = xtlc;
    xtld o = xtld;

    xtrao = xtra;
    xtrbo = xtrb;
    xtrco = xtrc;

```

xtrdo = xtrd;

xbla = lag_coefs[0] * xblao + lag_coefs[1] * xblbo +
lag_coefs[2] * xblco + lag_coefs[3] * bl_gap_mm_old;

xblb = lag_coefs[4] * xblao + lag_coefs[5] * xblbo +
lag_coefs[6] * xblco + lag_coefs[7] * bl_gap_mm_old;

xblc = lag_coefs[8] * xblao + lag_coefs[9] * xblbo +
lag_coefs[10] * xblco + lag_coefs[11] * bl_gap_mm_old;

xbl d = lag_coefs[12] * xbla + lag_coefs[13] * xblb +
lag_coefs[14] * xblc + lag_coefs[15] * bl_gap_mm;

xbra = lag_coefs[20] * xbrao + lag_coefs[21] * xbrbo +
lag_coefs[22] * xbrco + lag_coefs[23] * br_gap_mm_old;

xbrb = lag_coefs[24] * xbrao + lag_coefs[25] * xbrbo +
lag_coefs[26] * xbrco + lag_coefs[27] * br_gap_mm_old;

xbrc = lag_coefs[28] * xbrao + lag_coefs[29] * xbrbo +
lag_coefs[30] * xbrco + lag_coefs[31] * br_gap_mm_old;

xbrd = lag_coefs[32] * xbra + lag_coefs[33] * xbrb +
lag_coefs[34] * xbrc + lag_coefs[35] * br_gap_mm;

xila = lag_coefs[40] * xilao + lag_coefs[41] * xilbo +
lag_coefs[42] * xilco + lag_coefs[43] * il_gap_mm_old;

xilb = lag_coefs[44] * xilao + lag_coefs[45] * xilbo +
lag_coefs[46] * xilco + lag_coefs[47] * il_gap_mm_old;

xilc = lag_coefs[48] * xilao + lag_coefs[49] * xilbo +
lag_coefs[50] * xilco + lag_coefs[51] * il_gap_mm_old;

xild = lag_coefs[52] * xila + lag_coefs[53] * xilb +
lag_coefs[54] * xilc + lag_coefs[55] * il_gap_mm;

xtra = lag_coefs[60] * xtrao + lag_coefs[61] * xtrbo +
lag_coefs[62] * xtrco + lag_coefs[63] * tr_gap_mm_old;

xtrb = lag_coefs[64] * xtrao + lag_coefs[65] * xtrbo +
lag_coefs[66] * xtrco + lag_coefs[67] * tr_gap_mm_old;

xtrc = lag_coefs[68] * xtrao + lag_coefs[69] * xtrbo +
lag_coefs[70] * xtrco + lag_coefs[71] * tr_gap_mm_old;

xtrd = lag_coefs[72] * xtra + lag_coefs[73] * xtrb +

```
lag_coefs[74] * xtrc + lag_coefs[75] * tr_gap_mm;

*(out+4) = xbid;
*(out+5) = xbrd;
*(out+6) = xtld;
*(out+7) = xtrd;

}

br_gap_mm_old = br_gap_mm;
bl_gap_mm_old = bl_gap_mm;
tr_gap_mm_old = tr_gap_mm;
tl_gap_mm_old = tl_gap_mm;

return;
}
```

WHAT IS CLAIMED IS:

1. An elevator system for controlling horizontal movement of an elevator car with respect to guide rails in an elevator hoistway, comprising:

a force-estimation or position-scheduled current command controller, responsive to a force command signal, and further responsive to a sensed gap signal indicative of an actual gap of the elevator car with respect to the guide rail in the elevator hoistway, for providing a force-estimation or position-scheduled current command controller signal; and

a magnet driver circuit, responsive to the force-estimation or position-scheduled current command controller signal, for providing a magnet driver circuit signal to control the horizontal movement of the elevator car with respect to the guide rails in the elevator hoistway,

whereby the horizontal movement of the elevator car is controlled without sensing magnetic flux.

2. An elevator system according to claim 1, wherein the force-estimation or position-scheduled current command controller comprises:

a force-estimation processor, responsive to the sensed gap signal, and further responsive to a sensed magnet driver current signal, for providing a force-estimation processor signal;

a force comparator, responsive to the force-estimation processor signal, and further responsive to

the force command signal, for providing a force comparator signal; and

a force feedback compensation processor, responsive to the force comparator signal, for providing a force feedback compensation processor signal as a current command to the magnet driver circuit.

3. An elevator system according to claim 2, wherein the force-estimation processor is implemented with hardware and further comprises:

a first operational amplifier, responsive to the sensed gap signal, for providing an operationally-amplified gap signal;

a multiplier circuit, responsive to the operationally-amplified gap signal, and further responsive to the sensed magnet driver current signal, for providing a multiplier circuit signal; and

a second operational amplifier, responsive to the multiplier circuit signal, for providing a second operationally-amplified multiplied signal to the force comparator.

4. An elevator system according to claim 1, wherein the force-estimation or position-scheduled current command controller is a position-scheduled current command controller, responsive to the force command signal, and further responsive to the sensed gap signal,

for providing a position-scheduled current command control signal as a current command to the magnet driver circuit.

5. An elevator system according to claim 1, wherein the magnet driver circuit further comprises:

- a current-error comparator, responsive to the force-estimation or position-scheduled current control signal indicative of a current command, and further responsive to a sensed magnet driver current signal, for providing a current error signal;

- a current feedback compensation processor, responsive to the current error signal, for providing a current feedback compensation processor signal;

- a driver circuit, responsive to the current feedback compensation processor signal, for providing a driver circuit signal to a magnet that provides a magnetic force to control said horizontal movement of the elevator car with respect to the guide rail in the elevator hoistway; and

- a current sensor, responsive to the driver circuit signal, for providing the sensed magnet driver current signal to the current-error comparator.

6. An elevator system according to claim 1, wherein the force command signal is indicative of a commanded horizontal movement of the elevator car with respect to

the guide rails in the elevator hoistway.

7. An elevator system according to claim 1, wherein the force-estimation or position-scheduled current command controller signal is indicative of a force-estimation or position-scheduled current command for controlling said horizontal movement of the elevator car with respect to the guide rail in the elevator hoistway.

8. An elevator system according to claim 1, wherein the magnet driver circuit signal is indicative of a current for driving the magnet to produce a magnetic force to control the horizontal movement of the elevator car with respect to the guide rail in the elevator hoistway.

9. An elevator system according to claim 1, wherein the force-estimation or position-scheduled current command controller comprises:

a force-estimation processor, responsive to the sensed gap signal, and further responsive to the force-estimation or position-scheduled current command controller signal, for providing a force-estimation processor signal;

a force comparator, responsive to the force-estimation processor signal, and further responsive to

the force command signal, for providing a force comparator signal; and

a force feedback compensation processor, responsive to the force comparator signal, for providing a force feedback compensation processor signal as a current command to the magnet driver circuit.

10. A system for controlling movement of a first object with respect to a second object, comprising:

a force-estimation or position-scheduled current command controller, responsive to a force command signal, and further responsive to a sensed gap signal indicative of an actual gap of the elevator car with respect to the guide rail in the elevator hoistway, for providing a force-estimation or position-scheduled current command controller signal; and

a magnet driver circuit, responsive to the force-estimation or position-scheduled current command controller signal, for providing a magnet driver circuit signal to control the horizontal movement of the first object with respect to the second object,

whereby the horizontal movement of the first object is controlled without sensing magnetic flux.

11. A system according to claim 10, wherein the force-estimation or position-scheduled current command controller comprises:

a force-estimation processor, responsive to the sensed gap signal, and further responsive to a sensed magnet driver current signal, for providing a force-estimation processor signal;

a force comparator, responsive to the force-estimation processor signal, and further responsive to the force command signal, for providing a force comparator signal; and

a force feedback compensation processor, responsive to the force comparator signal, for providing a force feedback compensation processor signal as a current command to the magnet driver circuit.

12. A system according to claim 10, wherein the force-estimation or position-scheduled current command controller is a position-scheduled current command controller, responsive to the force command signal, and further responsive to the sensed gap signal, for providing a position-scheduled current command control signal as a current command to the magnet driver circuit.