Compensator Objects

Introduction

A **Compensator** object manages a single compensation table. Its primary function is to provide an interface to configure both the compensating axes and the compensated axis. It also provides an interface for loading the on-controller compensation tables. The Compensator object is a host-based object that has a corresponding compensator object embedded on the controller. The embedded compensator handles the real-time issues associated with axis position compensation.

Before creating the MPI Compensator object, the corresponding embedded compensator object on the controller must be enabled. Also, before configuring the MPI Compensator object, the controller's compensation table must be allocated with a sufficient size to hold all required compensation values (or points). Both of these items can be configured using mpiControlConfigGet/Set(.) methods.

NOTE: Configuring the compensator table size using mpiControlConfigSet(...) will reallocate the controller's dynamic memory. Reallocating dynamic memory on the controller affects multiple objects and should only be done at the very beginning of your application.

For more information on determining compensation table size please see Determining Required Compensator Table Size.

See Also:
Configuring the Compensator Objects for Operation
Determining Required Compensator Table Size
Loading the Compensation Table
Setting up an area for 2D Position Compensation

Methods

Create, Delete, Validate Methods

mpiCompensator**Create**	Create Compensator object
mpiCompensatorl**Delete**	Delete Compensator object
mpiCompensator**Validate**	Validate Compensator object

Configuration and Information Methods

mpiCompensator**ConfigGet**	Get Compensator configuration

mpiCompensator**ConfigSet**	Set Compensator configuration
meiCompensator**Info**	Get Compensator information
meiCompensator**TableGet**	Get Compensator table
meiCompensator**TableSet**	Set Compensator table

Memory Methods

meiCompensator**Memory**	Set address to be used to access Compensator memory
meiCompensator**MemoryGet**	Get bytes of Compensator memory and place it into application memory
meiCompensator**MemorySet**	Put (set) bytes of application memory into Compensator memory

Relational Methods

meiCompensator**Control**	Return handle of Control object associated with Compensator
meiCompensator**Number**	Get number of Compensator

Data Types

MPICompensator**Config**

MPICompensator**Dimension**

MPICompensator**Info**

MPICompensator**InputAxis**

MPICompensator**Message**

MPICompensator**Range**

Constants

MPICompensator**DimensionsMAX**

mpiCompensatorCreate

Declaration

```
MPICompensator  mpiCompensatorCreate(MPIControl  control,
                                     long        number);
```

Required Header: stdmpi.h

Description

mpiCompensatorCreate creates a Host Compensator object associated with the compensation object identified by ***number*** located on motion controller ***control***. CompensatorCreate is the equivalent of a C++ constructor.

Valid compensator numbers are zero (0) to MPIControlMAX_COMPENSATORS.

Before creating a Compensator object, the controller compensation objects must be enabled using MPIControlConfig.compensatorCount, or the host object will be invalid.

Return Values	
handle	handle to a Compensator object
MPIHandleVOID	if the object could not be created

See Also

mpiCompensatorDelete | mpiCompensatorValidate | MPIControlConfig

mpiCompensatorDelete

Declaration

```
long  mpiCompensatorDelete(MPICompensator  compensator);
```

Required Header: stdmpi.h

Description

mpiCompensatorDelete deletes a host Compensator object (***compensator***) and invalidates its handle.

CompensatorDelete is the equivalent of a C++ destructor.

Return Values	
MPIMessageOK	if *CompensatorDelete* successfully deletes a Compensator object and invalidates its handle

See Also

mpiCompensatorCreate | mpiCompensatorValidate

mpiCompensatorValidate

Declaration

```
long  mpiCompensatorValidate(MPICompensator  compensator);
```

Required Header: stdmpi.h

Description

mpiCompensatorValidate validates the Compensator object (***compensator***) and its handle. Always call mpiCompensatorValidate after creating a new Compensator object.

Return Values	
MPIMessageOK	if Compensator is a handle to a valid object.
MPICompensatorMessageCOMPENSATOR_INVALID	If the *number* used in mpiCompensatorCreate() was not a valid number. Valid numbers range from 0 to MPIControlMAX_COMPENSATORS
MPICompensatorMessageNOT_ENABLED	If the corresponding controller compensation object number is valid but is disabled on the controller.

See Also

mpiCompensatorCreate | mpiCompensatorDelete

mpiCompensatorConfigGet

Declaration

```
long  mpiCompensatorConfigGet(MPICompensator         compensator,
                              MPICompensatorConfig  *config,
                              void                  *external);
```

Required Header: stdmpi.h

Description

mpiCompensatorConfigGet gets the configuration of a Compensator object (***compensator***) and puts (writes) it in the structure pointed to by ***config***, and also writes it into the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The configuration information in external is intended for future use and is not currently used. Set this value to NULL.

Return Values	
MPIMessageOK	if *CompensatorConfigGet* successfully gets the configuration of a Compensator object and writes it into the structure(s).
MPIMessagePARAM_INVALID	If **config* parameter is NULL, or if **external* parameter is NOT NULL.

See Also

mpiCompensatorConfigSet | MEICompensatorConfig

mpiCompensatorConfigSet

Declaration

```
long  mpiCompensatorConfigSet(MPICompensator        compensator,
                              MPICompensatorConfig  *config,
                              void                  *external);
```

Required Header: stdmpi.h

Description

mpiCompensatorConfigSet sets (writes) the configuration of a Compensator object (***compensator***) using data from the structure pointed to by ***config***, and also using data from the implementation-specific structure pointed to by ***external*** (if ***external*** is not NULL).

The configuration information in ***external*** is in addition to the configuration information in ***config***, i.e. the configuration information in ***config*** and in ***external*** is not the same information.

NOTE: ***config*** or ***external*** can be NULL (but both cannot be NULL).

Remarks

external either points to a structure of type **MEICompensatorConfig{}** or is NULL.

Return Values	
MPIMessageOK	if *CompensatorConfigSet* successfully writes the configuration of a Compensator object using data from the structure(s)
MPIMessagePARAM_INVALID	If **config* parameter is NULL, or if **external* parameter is NOT NULL
MPIMessageARG_INVALID	The axisOutNumber or any of the inputAxis[n].axisNumbers correspond to invalid Axis objects on the controller. See MPICompensatorConfig.

MPICompensatorMessageDIMENSION_NOT_SUPPORTED	dimensionCount is out of range. See MPICompensatorConfig.
MPICompensatorMessageAXIS_NOT_ENABLED	The axisOutNumber or any of the inputAxis[n].axisNumbers correspond to disabled Axis objects on the controller. See MPICompensatorConfig.
MPICompensatorMessagePOSITION_DELTA_INVALID	The *positionDelta* argument is either out of range or not a multiple of the range. See MPICompensatorConfig.
MPICompensatorMessageTABLE_SIZE_ERROR	The host compenstation table will not fit within the controller's configured compensation table. See Determining Required Compensator Table Size.

See Also

mpiCompensatorConfigGet | MEICompensatorConfig

mpiCompensatorInfo

Declaration

```
long  mpiCompensatorInfo(MPICompensator      compensator,
                         MPICompensatorInfo  *info);
```

Required Header: stdmpi.h

Description

mpiCompensatorInfo reads the static information about the compensator object, and writes it into the structure pointed to by ***info***.

compensator	a handle to the Compensator object.
***info**	a pointer to a compensator information structure.

Return Values	
MPIMessageOK	if *CompensatorInfo* successfully writes the Compensator object information into the *info* structure.
MPICompensatorMessageNOT_CONFIGURED	MPI Compensator object must be configured before calling mpiCompensatorTableGet/Set or mpiCompensatorInfo.

See Also

MPICompensatorInfo | MPIControlConfig | mpiCompensatorTableGet | mpiCompensatorTableSet | mpiCompensatorInfo

mpiCompensatorTableGet

Declaration

```
long  mpiCompensatorTableGet(MPICompensator   compensator,
                             long             *table);
```

Required Header: stdmpi.h

Description

mpiCompensatorTableGet reads the NxM Compensator table stored on the controller whose dimensions are defined by the values in the MPICompensatorConfig structure. These values are written into the location specified by ***table***.

NOTE: The array pointed to ***table*** must have enough memory allocated to hold the entire size of the configured compensation table.

Return Values	
MPIMessageOK	
MPICompensatorMessageNOT_CONFIGURED	MPI Compensator object must be configured before calling mpiCompensatorTableGet/Set or mpiCompensatorInfo.
MPIMessagePARAM_INVALID	If **table* is NULL.

See Also

mpiCompensatorTableSet | MPICompensatorConfig

mpiCompensatorTableSet

Declaration

```
long  mpiCompensatorTableSet(MPICompensator   compensator,
                             long             *table);
```

Required Header: stdmpi.h

Description

mpiCompensatorTableSet writes the values stored in the location specified by **table* to the Compensator table stored on the controller.

NOTE: The array pointed to **table* must have a size large enough to fill the configured compensation table size (as defined by the MPICompensatorConfig structure) or memory access violations may occur.

Return Values	
MPIMessageOK	
MPICompensatorMessageNOT_CONFIGURED	MPI Compensator object must be configured before calling mpiCompensatorTableGet/Set or mpiCompensatorInfo.
MPIMessagePARAM_INVALID	If **table* is NULL.

See Also

mpiCompensatorTableGet | MPICompensatorConfig

mpiCompensatorMemory

Declaration

```
long  mpiCompensatorMemory(MPICompensator   compensator,
                           void             **memory);
```

Required Header: stdmpi.h

Description

mpiCompensatorMemory sets (writes) an address (used to access a Compensator object's memory) to the contents of ***memory***.

Return Values	
MPIMessageOK	if *CompensatorMemory* successfully writes the Compensator's address to the contents of *memory*.

See Also

mpiCompensatorMemoryGet | mpiCompensatorMemorySet

mpiCompensatorMemoryGet

Declaration

```
long  mpiCompensatorMemoryGet(MPICompensator    compensator,
                              void              *dst,
                              void              *src,
                              long              count);
```

Required Header: stdmpi.h

Description

mpiCompensatorMemoryGet copies ***count*** bytes of a Compensator's (***compensator***) memory (starting at address ***src***) to application memory (starting at address ***dst***).

Return Values	
MPIMessageOK	if *CompensatorMemoryGet* successfully copies data from Compensator memory to application memory.

See Also

mpiCompensatorMemorySet | mpiCompensatorMemory

mpiCompensatorMemorySet

Declaration

```
long  mpiCompensatorMemorySet(MPICompensator    compensator,
                              void              *dst,
                              void              *src,
                              long              count);
```

Required Header: stdmpi.h

Description

mpiCompensatorMemorySet copies ***count*** bytes of application memory (starting at address ***src***) to a Compensator's (***compensator***) memory (starting at address ***dst***).

Return Values	
MPIMessageOK	if *CompensatorMemorySet* successfully copies data from application memory to Compensator memory.

See Also

mpiCompensatorMemoryGet | mpiCompensatorMemory

mpiCompensatorControl

Declaration

```
MPIControl  mpiCompensatorControl(MPICompensator   compensator);
```

Required Header: stdmpi.h

Description

mpiCompensatorControl returns a handle to the Control object with which the compensator is associated.

compensator	a handle to the Compensator object

Return Values	
MPIControl	handle to a Control object
MPIHandleVOID	if *compensator* is invalid

See Also

mpiCompensatorCreate | mpiControlCreate

mpiCompensatorNumber

Declaration

```
long  mpiCompensatorNumber(MPICompensator   compensator,
                           long             *number);
```

Required Header: stdmpi.h

Description

mpiCompensatorNumber writes the index of a compensation object (object on the motion controller that the Compensator object is associated with) to the contents of ***number***.

Return Values	
MPIMessageOK	if *CompensatorNumber* successfully writes the index of a compensation object to the contents of *number*.

See Also

mpiCompensatorCreate

MPICompensatorConfig

Definition

```
typedef struct MPICompensatorConfig {
    long                          dimensionCount,
    MPICompensatorInputAxis       inputAxis[MPICompensatorDimensionMAX],
    long                          outputAxisNumber,
} MPICompensatorConfig;
```

Description

dimensionCount	The input dimension count of the compensation table. Valid values are from zero (0) to MPICompensatorDimensionsMAX. A value of zero (0) effectively disables the compensation object
inputAxis	The substructure used to configure each input dimension of the Compensator object.
outputAxisNumber	This specifies the axis number of the Axis to be compensated by the Compensator object. This number must correspond to a valid (existing) and enabled Axis on the controller.

See Also

MPIControlConfig | mpiCompensatorConfigGet | mpiCompensatorConfigSet | MPICompensatorDimension | MPICompensatorDimensionsMAX

MPICompensatorDimension

Definition

```
typedef struct MPICompensatorDimension {
    MPICompensatorDimensionX,
    MPICompensatorDimensionY,
} MPICompensatorDimension;
```

Description

MPICompensatorDimension an enumeration of valid Compensator dimensions.

MPICompensatorDimensionX	First Compensating Dimension
MPICompensatorDimensionY	Second Compensating Dimension

See Also

MPICompensatorConfig | MPICompensatorInfo

MPICompensatorInfo

Definition

```
typedef struct MPICompensatorInfo {
    long     tableDimensions[MPICompensatorDimensionMAX],
    long     tableSizeBytes,
} MPICompensatorInfo;
```

Description

tableDimensions	The dimensions along each axis of the table. This value is affected by the values set in the MPICompensatorConfig structure.
tableSizeBytes	The size (in Byte) required to store the entire compensation table in a host resident structure or array. This value is affected by the values set in the MPICompensatorConfig structure. This is NOT the amount of memory allocated on the controller by setting the MPIControlConfig.compensatorPointCount value.

See Also

MPICompensatorConfig | MPIControlConfig | MPICompensatorDimension | MPICompensatorDimensionMAX

MPICompensatorInputAxis

Definition

```
typedef struct MPICompensatorInputAxis {
    long                      axisNumber,
    MPICompensatorRange       range,
    long                      positionDelta,
} MPICompensatorInputAxis;
```

Description

axisNumber	This specifies the axis number of the compensating Axis object. The position from this Axis will be used to index a single dimension of the compensation table. This number must correspond to a valid (existing) and enabled Axis on the controller.
range	Used to configure the feedback positions along the compensation axis where compensation will start and end.
positionDelete	Spacing between compensation positions on the compensating axis. **NOTE**: This value must be an exact multiple of the range (i.e. the difference between range.positionMax – range.positionMin)

See Also

MPICompensatorConfig | mpiCompensatorConfigGet | mpiCompensatorConfigSet

MPICompensatorMessage

Definition

```
typedef enum {
   MPICompensatorMessageCOMPENSATOR_INVALID,
   MPICompensatorMessageNOT_CONFIGURED,
   MPICompensatorMessageNOT_ENABLED,
   MPICompensatorMessageAXIS_NOT_ENABLED,
   MPICompensatorMessageTABLE_SIZE_ERROR,
   MPICompensatorMessagePOSITION_DELTA_INVALID,
   MPICompensatorMessageDIMENSION_NOT_SUPPORTED,
} MPICompensatorMessage;
```

Description

MPICompensatorMessageCOMPENSATOR_INVALID

If the number used to in mpiCompensatorCreate(...) was not a valid number. Valid numbers range from 0 to MPIControlMAX_COMPENSATORS.

MPICompensatorMessageNOT_CONFIGURED

MPI Compensator object must be configured before calling mpiCompensatorTableGet/Set or mpiCompensatorInfo.

MPICompensatorMessageNOT_ENABLED

If the corresponding controller compensation object number is valid, but is disabled on the controller.

MPICompensatorMessageAXIS_NOT_ENABLED

The axisOutNumber or any of the inputAxis[n].axisNumbers correspond to disabled Axis objects on the controller. See MPICompensatorConfig{}.

MPICompensatorMessageTABLE_SIZE_ERROR

The host compenstation table will not fit within the controller's configured compensation table. See Determining Required Compensation Table Size.

MPICompensatorMessagePOSITION_DELTA_INVALID

The positionDelta argument is either out of range or is not a multiple of the range. See MPICompensatorConfig{}.

MPICompensatorMessageDIMENSION_NOT_SUPPORTED

dimensionCount is out of range. See MPICompensatorConfig{} for more information.

See Also

MPICompensatorRange

Definition

```
typedef struct MPICompensatorRange {
    long    positionMin,
    long    positionMax,
} MPICompensatorRange;
```

Description

positionMin	The minimum feedback position (counts) along the compensation axis where compensation will occur.
positionMax	The maximum feedback position (counts) along the compensation axis where compensation will occur.

See Also

MPICompensatorConfig | mpiCompensatorConfigGet | mpiCompensatorConfigSet

MPICompensatorDimensionsMAX

Definition

```
#define MPICompensatorDimensionsMAX (MPICompensatorDimensionLAST)
```

Description

MPICompensatorDimensionMAX defines the maximum number of dimensions supported by the Compensator object's compensation tables. Currently, the maximum dimension value is 2.

See Also

MPICompensatorDimension | MPICompensatorInfo | MPICompensatorConfig

Determining Required Compensator Table Size

The compensator table size is dependent on the number of dimensions (1D or 2D), the position range, and the resolution (or granularity) of the compensation points. The compensator uses linear interpolation to calculate the compensation values between each distinct compensation point.

For each compensation axis there are three position values: Min, Max, and Delta. The compensating range for an axis is specified by the Min and Max positions along the axis. The range (Max – Min) divided by Delta, determines the number of required points for the compensator. You can calculate the number of required compensator points by using the following equations:

1D Compensation: Points = (positionMax-positionMin) / positionDelta + 1

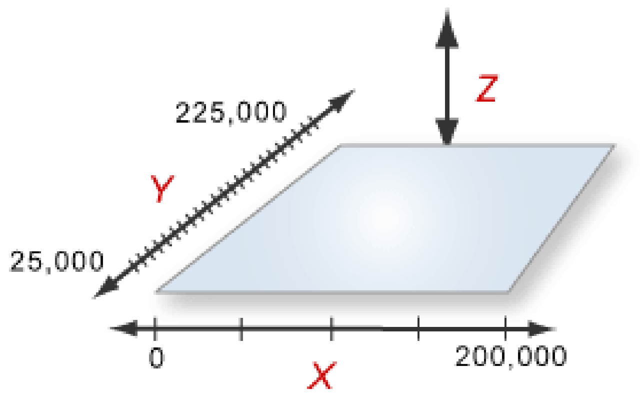
2D Compensation: Points = PointsX * PointsY

NOTE: Delta must be an exact multiple of the difference between Min and Max.

Example: (taken from comp.c sample application)

To compensate a Z (vertical) axis for X-Y surface irregularities, first define the X-Y area to be compensated (Xmin to Xmax, Ymin to Ymax). Then define the spacing of the measuring points (delta) for the X and Y axes to determine the compensation table size.

For the X-Y table diagram below we have:
Xmin = 0, Xmax = 200000, Xdelta = 50000
Ymin = 25000, Ymax = 225000, Ydelta = 10000



For this table our X & Y dimensions are:
X_DIM = (200000-0)/50000 + 1 = 5

Y_DIM = (225000-25000)/10000 +1 = 21

which requires a table point count of:

Points = X_DIM * Y_DIM = 105

With this information we can now configure the size of our compensation table on the controller using MPIControlConfig.compensatorPointCount = 105.

Configuring the Compensator Objects for Operation

After determining the required compensator table size, we need to configure both the embedded compensation tables on controller and the MPI Compensator object.

We will illustrate how to do this using the X-Y-Z system defined in the Determining Required Compensator Table Size section.

Configuring Controller Compensation Table
From our example in the previous section we have calculated that we need at least a point count of 105 to hold all of our measured compensation points (Acquiring and loading compensation points will be described in in the next section). First we need tell the motion controller to allocate memory space to hold the compensation table. We also need to enable a compensator since compensator objects are disabled on the controller by default. For an example, see the code below.

```
MPIControlConfig config;
long returnValue; returnValue =
   mpiControlConfigGet(control,
                   &config,
                   NULL);

if (returnValue == MPIMessageOK) {
   /* configure first compensator table size so our 2D array will fit */
   config.compensatorCount = 1;
   config.compensatorPointCount[0] = 105;

   /*
    * WARNING: this is a low-level configuration that will
    * reinitialize the controller's dynamic memory buffers!
    * Only preform this operation at system initialization.
   */
    returnValue =
       mpiControlConfigSet(control,
                       &config,
                       NULL);
}
```

The comment above reminds us that calling mpiControlConfigSet(...) will reallocate dynamic memory. Reallocation of dynamic memory affects other objects on the controller, so it should only be done during system initialization and not during the execution of a move.

Configuring the MPI Compensator Object

Continuing with our example, we will now assume that our axis numbers for axis X, Y, and Z are 0, 1, & 2 respectively. If we also assume that the MPI Compensator object has already been created, the code to configure the object would look like the following:

```
if (returnValue == MPIMessageOK) {
    MPICompensatorConfig config;

    returnValue =
        mpiCompensatorConfigGet(compensator,
                                &config,
                                NULL);
}

    if (returnValue == MPIMessageOK) {
        config.dimensionCount = 2;

        /* configure first compensating (input) axis */          config.inputAxis[0].axisNumber = 0;
        config.inputAxis[0].range.positionMin = 0;
        config.inputAxis[0].range.positionMax = 250000;
        config.inputAxis[0].positionDelta = 50000;

        /* configure second compensating (input) axis */
        config.inputAxis[1].axisNumber = 1;
        config.inputAxis[1].range.positionMin = 25000;
        config.inputAxis[1].range.positionMax = 225000;
        config.inputAxis[1].positionDelta = 10000;

        /* configure compensated (out) axis */
        config.outputAxisNumber = 2;

        returnValue =
          mpiCompensatorConfigSet(compensator,
                                  &config,
                                  NULL); }
```

Once we have the Compensation table allocated and have a configured Compensation object, the last step is to Load the Compensation Table.

Loading the Compensation Table

Next we need to somehow acquire high precision distance measurements (via interferometer, etc.) to the surface at each of the X-Y locations in the compensation area, and store the X and Y offset positions.

Once you've obtained these positions, they will need to be loaded into our previously configured compensation table. (see previous section). Continuing with our original example lets assume that our measurements are as defined by the following table:

```
long compensatorTable[21][5] =
{
     { 0,    0,    0,     0,     0, },
     { 100,  200,  -200,  -100,  0, },
     { 200,  400,  -400,  -200,  0 },
     { 300,  600,  -600,  -300,  0 },
     { 400,  800,  -800,  -400,  0 },
     { 500,  1000, -1000, -500,  0 },
     { 600,  1200, -1200, -600,  0 },
     { 700,  1400, -1400, -700,  0 },
     { 800,  1600, -1600, -800,  0 },
     { 900,  1800, -1800, -900,  0 },
     { 1000, 2000, -2000, -1000, 0 },
     { 900,  1800, -1800, -900,  0 },
     { 800,  1600, -1600, -800,  0 },
     { 700,  1400, -1400, -700,  0 },
     { 600,  1200, -1200, -600,  0 },
     { 500,  1000, -1000, -500,  0 },
     { 400,  800,  -800,  -400,  0 },
     { 300,  600,  -600,  -300,  0 },
     { 200,  400,  -400,  -200,  0 },
     { 100,  200,  -200,  -100,  0 },
     { 0,    0,    0,      0,    0 },
};
```

To load the above compensation table execute the following code:

```
returnValue = mpiCompensatorTableSet(compensator,
                                     (long*)compensatorTable);
```

Once the compensation positions are loaded, the compensation will be applied to the Z-axis' position feedback loop every servo cycle.

NOTE: No more interpolated compensation of the Z-axis will occur outside of the defined compensation range. Therefore, the compensation of the Z-axis will remain fixed outside of this range.

Setting up an area for 2D Position Compensation

The XMP has 2D compensation capabilities. The user-supplied compensation table is downloaded into XMP memory. The XMP automatically applies this compensation information to optimize motion profiles in real time.