NorthTek -- Application Note 1003

Implementing arbitrary mounting configurations with the GEDC-6/DC-4/AHRS-8 navigation sensors.

Introduction

This application note describes a method of programming the GEDC-6/DC-4/AHRS-8 navigation sensors to allow any arbitrary mounting configuration. A standard set of orthogonal orientations is provided in the baseline configuration items for a GEDC-6/DC-4/AHRS-8. This application note demonstrates a way to mount the navigation sensor in any orientation and have software adjust the heading, pitch and roll outputs relative to the new orientation.

Background

The GEDC-6/DC-4/AHRS-8 navigation sensors have internal configuration that is stored in non-volatile memory. One of the internal configuration items is the boresight "matrix". The matrix holds the 3 dimensional rotation of the navigation sensor's mounting position with respect to a reference orientation with pitch and roll at 0 degrees and a heading of 0 degrees magnetic North. The preset orientation choices allow this matrix to be selected from a standard set of orthogonal rotations. However for unique circumstances or where the user desires to correct any slight misalignment in mounting it is possible to have the navigation sensor compute the boresight matrix based on the current mounting orientation. This application note describes the NorthTek program that accomplishes this task.

Mathematics of the Tare Process

The Tare function is fairly simple. The navigation sensor orientation, or platform, matrix is built-up from estimated X,Y,Z vectors computed within the AdaptNav™ algorithm. The X-axis (North) of the navigation sensor is represented by the vector CP2. The Y-axis (East) of the navigation sensor is represented by CP1. The Z-axis (Down) is represented by accelEst. These vectors are used to build the 3x3 orientation matrix called Matrix2. The new boresight matrix is just the inverse of Matrix2. Essentially, the boresight matrix multiplied to Matrix2 will produce the identity matrix which represents a north facing, level navigation sensor.

$$\text{Matrix2} = \begin{bmatrix} CP2x & CP1x & accelEstx \\ CP2y & CP1y & accelEsty \\ CP2z & CP1z & accelEstz \end{bmatrix}$$

$$\text{BoresightMatrix} = [\text{Matrix2}]^{-1}$$

sparton
NAVIGATION AND EXPLORATION
5612 Johnson Lake Rd • Deleon Springs, FL • 32130
800.824.0682 • 386.985.4631
www.sparton.com/ne • www.spartonnavex.com

Usage of the Tare program

The Tare program is simple to use. The program must be downloaded to an operating navigation sensor that is positioned in the orientation that is desired. The user performs the following steps:

1) Connect the navigation sensor to a PC that has a terminal emulator such as Hyperterminal or TeraTerm.
2) Power up the navigation sensor and start the terminal emulator.
3) Position the user device in the desired position with the device oriented at 0 degrees and at the desired roll and pitch orientation. The tare script will align the navigation sensor to 0 degrees heading and 0 degrees pitch and roll in this device orientation.
4) Verify communications by hitting return a few times and verify that the "OK" prompt is returned.
5) Configure the terminal emulator to add 5 milliseconds of line delay for each transmitted line.[1]
6) Use the "send file" option to send the program to the navigation sensor. Use only the plain text send functions, the use of a specialized protocol such as xmodem is not required.
7) Wait until the adjustment complete message is printed.
8) Reset or power cycle the navigation sensor.

Description of the Tare NorthTek Program

In this section the Tare program will described line by line. The user may wish to have the full program listing available as a reference. The full listing is provided at the end of this document. In Forth, the native language of NorthTek, a comment is started with the // operator. Note that a space must follow all operators in Forth, thus the reader will note that all comments (in fact all operators) are surrounded by whitespace[2]. Thus in the program listing all lines that start with // are comments. Note that the comment operator may appear on a line of code and the comment extends to the end of the line.[3] The reader is encouraged to read the program comments. For the discussion that follows all comments are ignored.

The first lines of code in the program are:

```
forget matrix
variable matrix 9 allot
variable matrix2 9 allot
```

This line demonstrates a technique that is repeated often in NorthTek™ programs. The first actual definition in this file is of the variable "matrix". Often times NorthTek programs are downloaded again and again in the

[1] On TeraTerm this is found on the setup->serial port menu.
[2] Newlines or spaces count as white space, tabs are not used.
[3] Inline comments can be done with "(" and ")", surrounded by whitespace.

same session for debug purposes. The storage space for program is finite, so it is possible, through multiple downloads, to overflow the program space.[4] What "forget matrix" does is cause the interpreter to remove all word definitions back to the named word, in this case "matrix". The first time this program is loaded the interpreter will print "Can't find it", but this is harmless. However the second time the program is loaded, this command will find the previous instance of "matrix" and delete all items declared after it, recovering the memory space. In this way each subsequent download erases the last download to keep from increasing the amount of memory used. This technique is repeated often in NorthTek™ sample programs.

After the interpreter executes the "forget matrix" command, it encounters two standard Forth variable declarations. Two variables, called "matrix" and "matrix2" are declared. Each has 9, 32 bit words allocated to them. [5] These matrices will be used to compute the new boresight matrix.

The GEDC-6/DC-4/AHRS-8 database contains 3x3 matrices in that each row of the matrix is stored as a separate variable. The words that follow in the file are used to manipulate single rows of matrices and the vectors and copy the correct values back and forth. Additionally the "set" word that is used to write a new vector or row of a matrix requires that the starting and stopping index of the row be the first two elements of the array passed to set the variable. Thus the following words are mostly "helper" functions to manage the getting and setting of array values.

```
: row0 0 + ;
: row1 12 + ;
: row2 24 + ;
```

These words establish three helper words. As most computer science students are aware, matrices or two dimensional arrays are stored in linear memory with a mapping function. In NorthTek™ the elements of each row in a matrix are stored together. These helper words simply compute the starting address of the elements in row 0, 1 or 2 of the matrix. The words expect the base address of the matrix[6] to be passed in.

Next the actual boresight computation is performed.

```
: compute
  cp2 di@ cp1 di@ accelEst di@  // get the three desired columns
  matrix buildMatrix            // build the matrix with rows
  matrix matrix2 T(m)           // Transform the matrix so data is columns
  matrix2 matrix inv(m)         // Invert the matrix to build the boresight
                                // matrix.
;
```

[4] The word "status" will print the amount of used and available memory.
[5] Technically the size of the amount of memory for the matrices is 10, 32 bit words, since variable allocates one 32 word itself. The 9 value is left to avoid confusion on the part of the reader as we are attempting to allocate a 3x3 array. To be completely correct the "allot" word could be passed 8 and the exactly correct amount would be allocated.
[6] Typing the name of a variable returns the address of the 0th element.

This word, when executed performs the following actions (see the mathematics description above). First the cp2, cp1 and accelEst vectors are read from the database. Each element in turn is converted to the pointer to the array of values. These three pointers to arrays are passed to the "buildMatrix" function with "matrix" pushed on the top of the stack. Once built the "matrix" and "matrix2" are passed to the matrix transpose function "T(m)" that places the results in "matrix2". Once transposed, "matrix2" is inverted and the result placed in "matrix". The resultant is the desired boresight matrix. All that remains is to take each row of the resultant matrix and place them in the correct rows of the boresight matrix in the database.

```
variable copyarray 5 allot

// ************************************************************
// Shorthand to compute an array index and write to it
// in the copyarray variable.
// ************************************************************
: index! 4 * copyarray + ! ;


// ************************************************************
( ptr_to_array -- ) // this is a standard stack diagram
// See the NorthTek programming manual or any Forth reference.
// cp
// copies a 3 element array into the copyarray
// copyarray ends up with 0 2 V1 V2 V3 which is
// the right form for setting in the database
// ************************************************************
: cp
  0 0 index!              // store 0 in position 0
  2 1 index!              // 2 in position 1, therefore we set items 0..2
  dup @ 2 index!          // make copy of pointer, store 1st element.
  4 + dup @ 3 index!      // move to second element, make copy, store value
  4 + @ 4 index!          // move to third element, store it.
;
```

This piece of code defines an array that is used to copy the rows into the database. As a reminder the "set" database word requires that for arrays an array with the starting and stopping indices in the first two locations be passed. Forth is more relaxed than most languages in that an array can contain different types of elements. In this case the array "copyarray" will contain two integers, 0 and 2, in the first two locations, followed by three floating point values that are the desired new values for the array.

The word "index!" is a shorthand "word" that when passed a value and an index, stores that value at that index in the "copyarray".

The word "cp" is used to take one three element array as an argument and put it into the "copyarray" with the proper format for the database "set" function. It uses "index!" function to aid in this process. First a 0 is stored in index 0 position of the "copyarray", followed by a 2 in the index 1 position. At this point the top of the stack contains the address of the source array. The word "dup" is used to make a copy of the array

pointer[7], and then it is de-referenced with "@" bringing the first element value to the top of the stack. This value is then written to the array in position 2. The array pointer is still on the stack at this point. It is incremented to the second source value and again duplicated. The 2nd value is brought to the stack and stored in the 3rd index position of the array. Finally the pointer that was pointing to the 2nd element of the source is incremented to point to the 3rd element of the source, de-referenced and stored in index position 4 of the copy array.

The next word definition uses many of the previous words to copy the computed boresight matrix into the database.

```
: copyit
  matrix row0 cp                    // Copy row to the copy array
  boresightMatrixX copyarray        // Set the database with the row
  set drop                          // Drop the result
  matrix row1 cp                    // same as before, Y row
  boresightMatrixY copyarray
  set drop
  matrix row2 cp                    // same as before Z row
  boresightMatrixZ copyarray
  set drop
;
```

The "copyit" word uses the "row0" operators and the "cp" word to first copy the 0th row of the matrix into the "copyarray" then gets the database identifier of the X boresight matrix. This and the "copyarray" are passed to the "set" database word to update the variable. The return value of "set" is dropped.[8] This is repeated for all three rows of the boresight matrix.

Finally comes the words that assemble all the pieces into a complete program.

```
// ********************************************************
: printit
  matrix cr m.                  // Use the matrix print function
  boresightMatrixX  di.         // Use the database print function
  boresightMatrixY  di.
  boresightMatrixZ  di.
;

// ********************************************************
// Create a small program to perform the Tare function
// ********************************************************
: tare
  matrix clear(m)       // clear the matrix we declared
  orientation 0 set     // setup to default orientation, required.
  5000 delay            // Wait 5 seconds for this to settle in computation
  compute copyit        // compute the matrix and copy to the boresight matrix.
  printit               // Print it for verification.
```

[7] Since "@" is going to consume the pointer on the top of the stack.
[8] The word "set" returns true or false if the values are within limits, including the array indices. The user may add to the program to print a message after each "set" rather than dropping the result.

```
;
```

The “printit” word uses the native “m.” command to print the computed matrix. The three following lines instruct the database to print the current values of the boresight matrix elements. This allows the operator to determine if the computed matrix is correctly stored in the database.

Finally the program is executed in-line as it is being downloaded.

```
: bye
." Boresight adjustment complete!\r\n"
;
// ************************************************************
// Run the program
// After it runs, unload this macro.
// ************************************************************
tare bye forget matrix
```

The word “bye” just prints a message when the program is done. As the program is loaded the last line of the file is interpreted and the “tare” program runs, performing the new boresight alignment. After that is complete the “bye” word is executed, printing the completion message. Finally the program is unloaded by calling “forget matrix”. Note that because the interpreter processes one token at a time, the “tare” word gets executed before either “bye” or “forget matrix” are even parse. Thus both the “tare” and “bye” programs run to completion before the entire program is “forgotten”.

To clear the new boresight matrix the user should enter “orientation 0 set”. This will reset the orientation to the nominal, horizontal orientation.

In summary the Tare program will compute the required orientation matrix “in-situ” in the GEDC-6/DC-4/AHRS-8 navigation sensor for any arbitrary mounting configuration.

Complete Listing of the Tare.4th program.

```
// ************************************************************
// If the macro gets reloaded in the same session
// Forget the previous version
// Needs  revision 2.1.1 or later.
// ************************************************************
// This forgets the matrix from a previous load of the macro
// this is standard practice with NorthTek for debugging
// so as to not overflow the wordlist space.
// When this file is reloaded with a terminal a second time
// this removes the previous program.
forget matrix

// ************************************************************
// Declare a few working matrices.
// ************************************************************
// Matrix is the working matrix for the new boresight matrix
```

```
// matrix2 is a working matrix since some matrix operators
// output the results in a second matrix.
variable matrix 9 allot
variable matrix2 9 allot

// ************************************************************
// Functions (aka NorthTek words): row0, row1, row2
// some convenient shorthand for matrices
// Inputs:
//  TOS: Ptr to start of a matrix
// Outputs:
//  TOS: Ptr. to start of a row within the given matrix
// ************************************************************
// Since in Forth you must do matrix/array index calculations
// explicitly, these operators just make it easy to
// get to rows 1 and 2 of a 3x3 matrix.
: row0 0 + ;
: row1 12 + ;
: row2 24 + ;

// ************************************************************
// Function (aka NorthTek word): compute
// The actual computation
( -- )
// This stack diagram indicates that there are no params
// and no explicit results.
// This function uses the global variables matrix and matrix2
// ************************************************************
: compute
  cp2 di@ cp1 di@ accelEst di@  // get the three desired columns
  matrix buildMatrix            // build the matrix with rows
  matrix matrix2 T(m)           // Transform the matrix so data is columns
  matrix2 matrix inv(m)         // Invert the matrix to build the boresight
                                // matrix.
;

// ************************************************************
// This is a temporary array used
// to constuct an array that the database will
// accept in the "set" command.
// ************************************************************
variable copyarray 5 allot

// ************************************************************
// Function (aka NorthTek word): index!
// Shorthand to compute an array index and write to it
// This word takes a value and an index and stores that element
// in that position in the copyarray.
// ************************************************************
( value index -- )
: index! 4 * copyarray + ! ;


// ************************************************************
// Function (aka NorthTek word): cp
( array_ptr -- )
// copies a 3 element array into the copyarray
// copyarray ends up with 0 2 V1 V2 V3 which is
// the right form for setting in the database
// ************************************************************
: cp
  0 0 index!              // store 0 in position 0 of copyarray
  2 1 index!              // 2 in position 1, therefore we set items 0..2
  dup @ 2 index!          // make copy of pointer,for next index, store 1st element.
  4 + dup @ 3 index!      // move to second element, make copy, store value
```

```
  4 + @ 4 index!         // move to third element, store it.
;

// ************************************************************
// Function (aka NorthTek word): copyit
// Copy each row of the matrix into the
// The corresponding row of the boresight matrix.
// ************************************************************
: copyit
  matrix row0 cp                    // Copy row 0 of matrix to
                                    //   the copyarray
  boresightMatrixX copyarray        // Set the database with the row
  set drop                          //   and then drop the result
                                    //   from the stack
  matrix row1 cp                    // same as before, Y row
  boresightMatrixY copyarray
  set drop
  matrix row2 cp                    // same as before Z row
  boresightMatrixZ copyarray
  set drop
;

// ************************************************************
// Function (aka NorthTek word): printit
// Printout the computed matrix and the current boresight
// matrix.
// ************************************************************
: printit
  matrix cr m.                  // Use the matrix print function
  boresightMatrixX  di.         // Use the database print function
  boresightMatrixY  di.
  boresightMatrixZ  di.
;

// ************************************************************
// Function (aka NorthTek word): tare
// Create a small program to perform the Tare function
// Inputs:
//   Database variables (used by compute): cp2, cp1, accelEst
// Modifies:
//   NorthTek variable: matrix
//   Database variable: orientation
// Outputs:
//    Database variables (set by copyit):
//         boresightMatrixX
//         boresightMatrixY
//         boresightMatrixZ
// ************************************************************
: tare
  matrix clear(m)       // clear the matrix we declared
  orientation 0 set     // setup to default orientation, required.
  5000 delay            // Wait 5 seconds for this to settle in computation
  compute copyit        // compute the matrix and copy to the boresight matrix.
  printit               // Print it for verification.
;

// ************************************************************
// Function (aka NorthTek word): untare
// Set the orientation back to "Horizontal" which in turn sets
// the boresightMatrix back to the identity matrix.
// Inputs:
//   None
// Outputs:
//    Database variables:
//         orientation
```

```
//          boresightMatrixX
//          boresightMatrixY
//          boresightMatrixZ
// ************************************************************
: untare
  orientation 0 set
  printit
;


// Function (aka NorthTek word): bye
: bye
." Boresight adjustment complete!\r\n"
;
// ************************************************************
// Run the program
// After it runs, unload this macro.
// ************************************************************
tare bye forget matrix
```

Want to know more?

- Check it out here: www.spartonnavex.com