How to obtain angular velocity and angular acceleration values directly from 3D accelerometer array data using simple matrix operations in Propeller/SPIN

*...the merit of service is seldom attributed
to the true and exact performer.*
William Shakespeare (1564-1616)
All's Well That Ends Well, Act III, scene vi

In the followings we shall reproduce the algorithm described in

K. Parsa, J. Angeles and A. K. Misra
Rigid-body pose and twist estimation using an accelerometer array
In **Applied Mechanics**, 74 (2004) pp. 223-236.

without the agonizing pain of abstract tensor and matrix algebra. The method will be demonstrated with many numeric examples. To calculate these examples I used only Propeller/SPIN and the **FPU_Matrix_Driver.SPIN** object from OBEX (**http://obex.parallax.com/objects/317/**). First the arrangement of four 3-axis acceleration sensors will be described, then the algorithm will be introduced and exercised via numeric examples.

The arrangement of the sensors
Let us put two H48C 3D accelerometers at the opposite corners of a square plate as shown in Fig. 1.

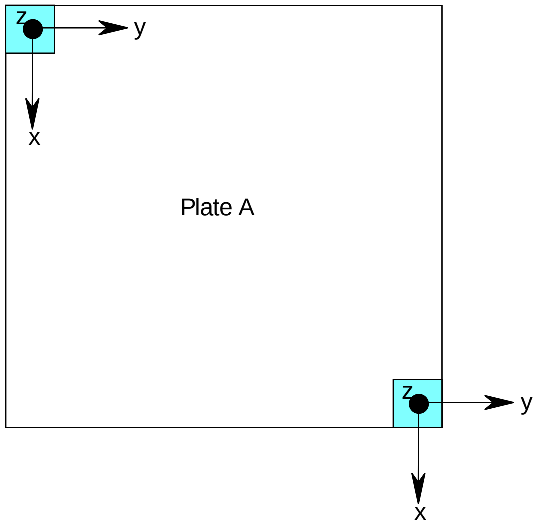

Figure 1. The z-axis of the H48C accelerometers are pointing towards the reader.

Let us make another square plate, equipped with two other sensors, like in Fig. 2.

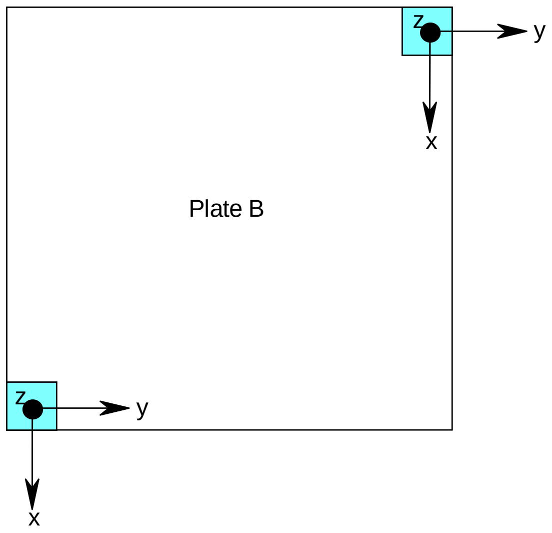

Figure 2. The z-axis of the H48C accelerometers are pointing towards the reader.

Now let us mount Plate A on top of Plate B to form a regular cube as shown in Fig. 3.

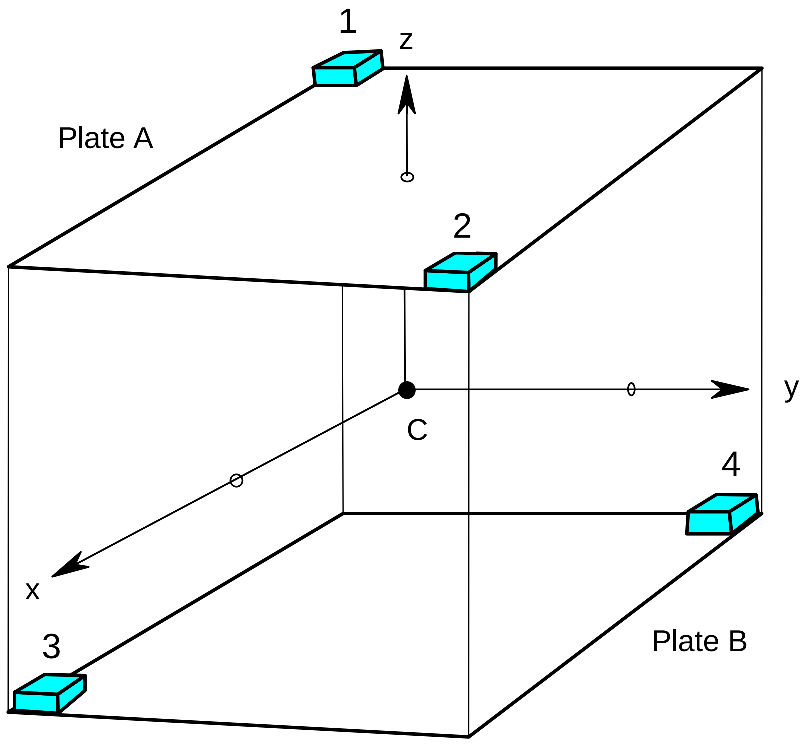

Figure 3. The four H48Cs arranged at the vertices of a tetrahedron.

The three corresponding axis of the sensors are parallel, and, by design, mutually orthogonal. The centroid of the pickup points is denoted by C and the sensors are numbered as shown.

Definition of matrices

Let us define two [3 by 4] matrices. These are matrix **R** of the relative positions and matrix **A** of the relative accelerations. The position of the sensors is related to the centroid C. Let us take the length of the side of the cube as one, then the coordinates of the sensors are

$$\begin{aligned} r_1 &= [-0.5,\ -0.5,\ \ 0.5] \\ r_2 &= [\ 0.5,\ \ 0.5,\ \ 0.5] \\ r_3 &= [\ 0.5,\ -0.5,\ -0.5] \\ r_4 &= [-0.5,\ \ 0.5,\ -0.5] \end{aligned}$$

as you can verify this in Fig. 3. The **R** matrix contains the coordinates of the sensors in its columns

$$\underline{\mathbf{R}} = \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \\ \hline -0.5 & 0.5 & 0.5 & -0.5 \\ -0.5 & 0.5 & -0.5 & 0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \end{array} \qquad \text{[3 by 4] matrix}$$

The **A** matrix is the matrix of the relative accelerations. The acceleration vectors measured by the sensors are

$$\begin{aligned} a_1 &= [\ a_{1x},\ a_{1y},\ a_{1z}] \\ a_2 &= [\ a_{2x},\ a_{2y},\ a_{2z}] \\ a_3 &= [\ a_{3x},\ a_{3y},\ a_{3z}] \\ a_4 &= [\ a_{4x},\ a_{4y},\ a_{4z}] \end{aligned}$$

How to make relative acceleration values from these? Let us first calculate the average acceleration vector a_C

$$a_C = 0.25 \cdot [a_{1x}+a_{2x}+a_{3x}+a_{4x},\ a_{1y}+a_{2y}+a_{3y}+a_{4y},\ a_{1z}+a_{2z}+a_{3z}+a_{4z}]$$

Then subtract a_C from the acceleration vectors to obtain relative accelerations

$$\begin{aligned} a_{r1} &= [\ a_{1x}-a_{Cx},\ a_{1y}-a_{Cy},\ a_{1z}-a_{Cz}] \\ a_{r2} &= [\ a_{2x}-a_{Cx},\ a_{2y}-a_{Cy},\ a_{2z}-a_{Cz}] \\ a_{r3} &= [\ a_{3x}-a_{Cx},\ a_{3y}-a_{Cy},\ a_{3z}-a_{Cz}] \\ a_{r4} &= [\ a_{4x}-a_{Cx},\ a_{4y}-a_{Cy},\ a_{4z}-a_{Cz}] \end{aligned}$$

And the **A** matrix is

$$\underline{\mathbf{A}} = \begin{array}{cccc} a_{r1} & a_{r2} & a_{r3} & a_{r4} \\ \hline a_{r1x} & a_{r2x} & a_{r3x} & a_{r4x} \\ a_{r1y} & a_{r2y} & a_{r3y} & a_{r4y} \\ a_{r1z} & a_{r2z} & a_{r3z} & a_{r4z} \end{array} \qquad \text{[3 by 4] matrix}$$

By the way, a_C is the linear acceleration vector measured by the sensor array. So half of the 6DOF IMU job done. Now, we have to calculate the angular acceleration and the angular velocity values. In other words we will get 9DOF data, won't we? Up till now the operations were reading the sensors, adding, subtracting dividing values, some housekeeping to arrange values in arrays. So, we encountered not too many complications.

An offline task to be solved only once

Before we proceed, we have to calculate the Moore-penrose inverse **P** of matrix **R**. This is easy and has to be done only once for a given sensor arrangement. You can do it with the **FPU_Matrix_Driver** object. Some of the comments of the Matrix_SVD (Singular Value Decomposition) procedure will guide you. Or, you can use some simple matrix algebra as follows

$$\underline{\mathbf{P}} = \underline{\mathbf{R}}^T \cdot (\underline{\mathbf{R}} \cdot \underline{\mathbf{R}}^T)^{-1}$$

Again, every step can be done with the **FPU_Matrix_Driver**, like for example

```
Matrix_Transpose(@RT,@R,3,4)            'This calculates R^T
Matrix_Multiply(@RRT,@R,@RT,3,4,4,3)    'This calculates R·R^T
Matrix_Inverse(@RRTI,@RRT,3)            'This calculates inverse of (R·R^T)
Matrix_Multiply(@P,@RT,@RRTI,4,3,3,3)   'This calculates P = R^T·(R·R^T)^-1
```

For our sensor arrangement the result is

```
       [[-0.5 -0.5  0.5 ]
   P = [ 0.5  0.5  0.5 ]        [4 by 3] matrix
        [ 0.5 -0.5 -0.5 ]
        [-0.5  0.5 -0.5 ]]
```

Verify that the **R**·**P** matrix product gives a [3 by 3] identity matrix. O.K. We have **P**, we have to store it somewhere as we shall use it frequently.

A little bit of physics shouldn't hurt

From rigid body kinematics, a very compact formula can be derived for the a_i accelerations. This formula contains the acceleration a_C of the centroid C, the angular velocity ω of the body's rotation around an axis containing the centroid and the time derivative α of the angular velocity, the so called angular acceleration. Note that these are just the IMU quantities we would like to measure. Before I write down the formula, I emphasize again, that our sensor array **estimates directly** all these three basic kinematic vectors. In other words, neither we have to derivate ω to obtain α, or, nor we have to integrate α to obtain ω. Beware of the following formula, because it is so simple that you can even remember it, if you are not careful enough. The formula is

$$a_i = a_C + \alpha \times r_i + \omega \times (\omega \times r_i)$$

where x denotes the vector product. In SPIN using the **FPU_Matrix_Driver**, e.g. for a_1, it goes as

```
Vector_CrossProduct(@wr1,@omega,@r1,3,1)       'This calculates ωxr1
Vector_CrossProduct(@wwr1,@omega,@wr1,3,1)     'This calculates ωx(ωxr1)
Vector_CrossProduct(@alphar1,@alpha,@r1,3,1)   'This calculates αxr1
Matrix_Add(@alphar1wwr1,@alphar1,@wwr1,3,1)    'This calculates αxr1+ωx(ωxr1)
Matrix_Add(@a1,@ac,@alphar1wwr1,3,1)           'This calculates a1
```

Of course, here we use this formula only to calculate correct a_i values for our sensor array for different types of motion of the body to numerically check the decoding algorithm. Now, we have prepared the tests, let's get back to the decoding algorithm.

How to decode the angular acceleration?
Well, we have decoded the linear acceleration of the sensor array. That is simply a_C. To get the angular acceleration, we have first to multiply the **A** matrix of the relative accelerations with **P**. The matrix **A** was calculated from the measured a_i values before and **P** was stored somewhere.

$$\underline{\mathbf{W}} = \underline{\mathbf{A}} \cdot \underline{\mathbf{P}}$$

W has a name, it is called the angular acceleration tensor. But it doesn't matter. We got it. **W** is a small [3 by 3] matrix, nine nicely arranged float values, nothing else from now on. The angular acceleration vector is simply

$$\alpha = 0.5 \cdot [\, W_{32}-W_{23},\ W_{13}-W_{31},\ W_{21}-W_{12} \,]$$

where the double subscript of W denotes the corresponding element of the W matrix. For example W_{32} is the second element of the third row.

Yes, yes, but what about the angular velocity?
We'll get it quickly. Angular velocity components are calculated from the diagonal elements of the matrix **W**. In preparation of the final result we calculate the quantity

$$sp = 0.5 \cdot (W_{S11} + W_{S22} + W_{S33})$$

and finally

$$\omega = [\mathrm{SQR}(W_{S11}-sp),\ \mathrm{SQR}(W_{S22}-sp),\ \mathrm{SQR}(W_{S33}-sp)]$$

Where SQR denotes the square root operation. These were two additions, a multiplication, three subtractions and three square roots. The correct sign of the components can be obtained easily as described, for example, in the original paper. We shall discuss the sign determination later. We can see, that the nine numbers of the **W** matrix contain all information about angular acceleration and angular velocity. So it deserves its name. Now we continue with some practical considerations and than with the numerical tests.

What to do if I arranged the sensors in a different way?
You have to compute the **R** matrix, then the **P** matrix for your arrangement. That's all. **R**, of course, has not to be singular in order to obtain a Moore-penrose inverse. In a planar arrangement, which seems to be a practical idea to place the sensors, the third row of R contains only zeroes. And **R** is singular, then. In other words, all sensors should not line up, or should not lay in the same plane.

O.K. But how long does this decoding take?
Well, in PASM this decoding takes 4-5 msec. In the FPU it takes less than 2 msec. Whichever you choose, you can handle 100 Hz (10 msec period) acceleration data. In SPIN you can cope with 20 Hz data easily.

First example: Sensor resting on a table
Let us assume that the table is horizontal and is resting on the ground. We have gravity of course (9.81 m/sec^2), but we don't have angular rotation and angular acceleration of the sensor array. The α and ω vectors are zero and the sensed a_i vectors are as follows

$$
\begin{aligned}
a_1 &= [\ 0.0,\ 0.0,\ 9.81] \\
a_2 &= [\ 0.0,\ 0.0,\ 9.81] \\
a_3 &= [\ 0.0,\ 0.0,\ 9.81] \\
a_4 &= [\ 0.0,\ 0.0,\ 9.81]
\end{aligned}
$$

First we calculate a_C

$$a_C = [\ 0.0,\ 0.0,\ 9.81]$$

Then the **A** matrix is

$$\underline{\mathbf{A}} = \begin{matrix} [[\ 0.0 & 0.0 & 0.0 & 0.0\ ] \\ [\ 0.0 & 0.0 & 0.0 & 0.0\ ] \\ [\ 0.0 & 0.0 & 0.0 & 0.0\ ]] \end{matrix}$$

The **W** matrix is

$$\mathbf{W} = \mathbf{A} \cdot \mathbf{P} = \begin{matrix} [[\ 0.0 & 0.0 & 0.0\ ] \\ [\ 0.0 & 0.0 & 0.0\ ] \\ [\ 0.0 & 0.0 & 0.0\ ]] \end{matrix}$$

So, both the measured α and ω are null vectors.

Why do our sensors measure a positive (upwards) 9.81 when we all know that gravity points downwards?
The proof-mass, or whatever, that measures the acceleration is pulled down by the gravity. The sensor feels that it is accelerating upwards, because the proof-mass is displaced downwards inside the sensor.

Sensor array accelerating but not rotating
Let us push the sensor array in the x direction with 1 m/sec^2 linear acceleration. The α and ω vectors are zero again , but we have a linear a_C acceleration of the whole sensor in the x direction. According to our formula

$$a_i = a_C + \alpha \times r_i + \omega \times (\omega \times r_i)$$

we calculate a_i values, taking into account the sensed g, of course

$$
\begin{aligned}
a_1 &= [\ 1.0,\ 0.0,\ 9.81] \\
a_2 &= [\ 1.0,\ 0.0,\ 9.81] \\
a_3 &= [\ 1.0,\ 0.0,\ 9.81] \\
a_4 &= [\ 1.0,\ 0.0,\ 9.81]
\end{aligned}
$$

The measured a_C, the average of the four a_i vectors, is

$$a_C = [\ 1.0,\ 0.0,\ 9.81]$$

Then the **A** matrix is

$$\underline{\mathbf{A}} = \begin{matrix} [[\ 0.0 & 0.0 & 0.0 & 0.0\ ] \\ [\ 0.0 & 0.0 & 0.0 & 0.0\ ] \\ [\ 0.0 & 0.0 & 0.0 & 0.0\ ]] \end{matrix}$$

The **W** matrix is

$$\underline{\mathbf{W}} = \underline{\mathbf{A}} \cdot \underline{\mathbf{P}} = \begin{bmatrix} 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

So, both the measured α and ω are null vectors, again.

Let us take some increasing spin

Now, we accelerate the sensor array as in the previous example, but this time we start to rotate it with

$$\alpha = [\ 0.0,\ 0.0,\ 0.5]$$

[rad/sec^2] angular acceleration around the z axis. According to our formula

$$a_i = a_C + \alpha \times r_i + \omega \times (\omega \times r_i)$$

we calculate a_i values again. First let us calculate the $\alpha \times r_i$ vectors

$$\begin{aligned} \alpha \times r_1 &= [\ \ 0.25,\ -0.25,\ 0.00\ ] \\ \alpha \times r_2 &= [\ -0.25,\ \ 0.25,\ 0.00\ ] \\ \alpha \times r_3 &= [\ \ 0.25,\ \ 0.25,\ 0.00\ ] \\ \alpha \times r_4 &= [\ -0.25,\ -0.25,\ 0.00\ ] \end{aligned}$$

then the a_i vectors

$$\begin{aligned} a_1 &= [\ 1.25,\ -0.25,\ 9.81\ ] \\ a_2 &= [\ 0.75,\ \ 0.25,\ 9.81\ ] \\ a_3 &= [\ 1.25,\ \ 0.25,\ 9.81\ ] \\ a_4 &= [\ 0.75,\ -0.25,\ 9.81\ ] \end{aligned}$$

The a_C vector, the average of a_i is the same as before

$$a_C = [\ 1.0,\ 0.0,\ 9.81]$$

But the $\underline{\mathbf{A}}$ matrix of the relative accelerations is filled not only with zeroes now

$$\underline{\mathbf{A}} = \begin{bmatrix} 0.25 & -0.25 & 0.25 & -0.25 \\ -0.25 & 0.25 & 0.25 & -0.25 \\ 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}$$

The $\underline{\mathbf{W}}$ matrix is

$$\underline{\mathbf{W}} = \underline{\mathbf{A}} \cdot \underline{\mathbf{P}} = \begin{bmatrix} 0.0 & -0.5 & 0.0 \\ 0.5 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

From this, using the formula for the decoded, measured α

$$\alpha = 0.5 \cdot [\ W_{32}-W_{23},\ W_{13}-W_{31},\ W_{21}-W_{12}\ ]$$

we obtain

$$\alpha = [\ 0.0,\ 0.0,\ 0.5\ ]$$

and the decoded, measured angular velocity vector is

$$\omega = [\ 0.0,\ 0.0,\ 0.0\ ]$$

Well, so far, so good.

Accelerating and rotating sensor array
Four seconds have passed and the sensor array is rotating now with

$$\omega = [\ 0.0,\ 0.0,\ 2.0\ ]$$

[rad/sec] angular velocity, while accelerating linearly and angularly as before

$$a_C = [\ 1.0,\ 0.0,\ 9.81\ ]$$

$$\alpha = [\ 0.0,\ 0.0,\ 0.5\ ]$$

The constituents to the a_i accelerations at the pickup points are

$$a_C = [\ 1.0,\ 0.0,\ 9.81\ ]$$

$$\begin{aligned} \alpha \times r_1 &= [\ \ 0.25,\ -0.25,\ 0.00\ ] \\ \alpha \times r_2 &= [\ -0.25,\ \ 0.25,\ 0.00\ ] \\ \alpha \times r_3 &= [\ \ 0.25,\ \ 0.25,\ 0.00\ ] \\ \alpha \times r_4 &= [\ -0.25,\ -0.25,\ 0.00\ ] \end{aligned}$$

$$\begin{aligned} \omega \times (\omega \times r_1) &= [\ \ 2.00,\ \ 2.00,\ 0.00\ ] \\ \omega \times (\omega \times r_2) &= [\ -2.00,\ -2.00,\ 0.00\ ] \\ \omega \times (\omega \times r_3) &= [\ -2.00,\ \ 2.00,\ 0.00\ ] \\ \omega \times (\omega \times r_4) &= [\ \ 2.00,\ -2.00,\ 0.00\ ] \end{aligned}$$

These are sensed accelerations at the pickup points and according to our formula

$$a_i = a_C + \alpha \times r_i + \omega \times (\omega \times r_i)$$

they are added (scrambled) in the sensors

$$\begin{aligned} a_1 &= [\ \ 3.25,\ \ 1.75,\ 9.81\ ] \\ a_2 &= [\ -1.25,\ -1.75,\ 9.81\ ] \\ a_3 &= [\ -0.75,\ \ 2.25,\ 9.81\ ] \\ a_4 &= [\ \ 2.75,\ -2.25,\ 9.81\ ] \end{aligned}$$

Now let us see, how the algorithm unscrambles the a_C, α and ω vectors. The a_C is simply the average of the four a_i vectors

$$a_C = [\ 1.0,\ 0.0,\ 9.81\ ]$$

The **A** matrix of the relative accelerations is

$$\underline{\mathbf{A}} = \begin{bmatrix} 2.25 & -2.25 & -1.75 & 1.75 \\ 1.75 & -1.75 & 2.25 & -2.25 \\ 0.00 & 0.00 & 0.00 & 0.00 \end{bmatrix}$$

We obtain the [3 by 3] **W** matrix with a simple matrix multiplication

$$\mathbf{\underline{W}} = \mathbf{\underline{A}} \cdot \mathbf{\underline{P}} = \begin{bmatrix} -4.0 & -0.5 & 0.0 \\ 0.5 & -4.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}$$

And we unscramble the angular acceleration vector immediately, using the formula

$$\alpha = 0.5 \cdot [\ W_{32}-W_{23},\ W_{13}-W_{31},\ W_{21}-W_{12}\ ]$$

resulting

$$\alpha = [\ 0.0,\ 0.0,\ 0.5\ ]$$

Then we calculate the quantity

$$sp = 0.5 \cdot (W_{S11} + W_{S22} + W_{S33}) = -4.0$$

And finally, from the formula

$$\omega = [SQR(W_{S11}-sp),\ SQR(W_{S22}-sp),\ SQR(W_{S33}-sp)]$$

we get the unscrambled ω vector

$$\omega = [\ 0.0,\ 0.0,\ 2.0\ ]$$

Right, again.

Summarizing the steps of the algorithm

To check and to follow the steps of the numeric examples one can use the Propeller/SPIN language and the **FPU_Matrix_Driver**. Some practice will ensure the user how simple it is and how easy to program the algorithm in Propeller/SPIN. Now I summarize briefly the main steps of the process.

The four sensor readings (a_i vectors) are stored in a [3 by 4] matrix, column wise.

The average of the acceleration vectors gives the linear acceleration $\boldsymbol{a_C}$ *of the sensor array.*

This average vector is subtracted from each column of the matrix, and the resulting matrix is multiplied with a precomputed one.

The [3 by 3] product matrix is used to estimate the $\boldsymbol{\alpha}$ *and the* $\boldsymbol{\omega}$ *vectors.*

$\boldsymbol{\alpha}$ *is obtained directly from this matrix with a simple formula.*

$\boldsymbol{\omega}$ *is obtained directly again with simple formulas.*

One can obtain the sign of the components of the ω *vector, for example, for* ω*_x easily as*

$$Sign\_of\_\omega_x = SIGN[(\omega_x\_Now - \omega_x\_Previous) \cdot \alpha_x\_Now]$$

based upon the fact that the components of α are measured with sign and they are directly related to the change of the consecutive components of ω.

Another, and numerically more robust way to get the sign of the components of the ω vector is to store the sum of the α components. The sign of the stored sums will yield the sign of the measured ω components at any moment. Yes, I know that this is something like an integration. But the difference between using the sign of a value, or using the value itself, is huge. The combination of the two methods of sign determination is especially robust and stable and that will be used in the final assembly. Sign uncertainty will appear only when the size of the ω components shrink below the noise level, in other words when ω components will be very small, and these uncertainties will be random with zero drift.

Static calibration of the individual sensors

Not all H48C sensors are created equal, so it is necessary to perform static calibration of them before using their readings in the previously described measuring algorithm. The H48C sensors are temperature compensated at the factory but there remains some temperature dependence, as well, that should be accounted for in the calibration procedure. There will be additional static and dynamic self-calibrations *en-route* of the craft that carries the sensor array. These *en-route* calibrations will compensate most of the remaining temperature dependence and mounting orientation inaccuracies of the 6DOF sensor array. These procedures enhance the precision of the device by several order of magnitudes and they will be done automatically by the Propeller of the sensor array. These *en-route* calibrations will be described later.

Physics behind static calibration

When the H48C is held steadily in relation to the Earth, for example is laying on a table, it senses and measures only the gravity. When we know our position (Lat, Lon, Alt) on the Earth, we can calculate precise approximation for the size of the g gravity vector there. That g vector is an extremely stable, almost ideal reference with no additional costs. The endpoints of the measured acceleration vectors are on a sphere with radius g in every pose of the resting sensor. So the yet unknown calibrated components a_x, a_y, a_z of the measured accelerations with the steady sensor satisfy the equation

$$(a_x)^2 + (a_y)^2 + (a_z)^2 = g^2$$

in every orientation of the H48C. We shall use scale factors and biases to transform the raw ADC counter readings into the calibrated a_x, a_y, a_z values. The determination of these factors and biases will be based upon of twelve steady measurements and on the previous equation.

A simple but accurate approximation of g

When you want to be more precise than just to apply 9.81 [m/s^2] for the size of g everywhere, than you can use the next mathematical form that describes the magnitude of gravity at the surface of the WGS84 ellipsoid

$$g_{WGS84} = g_0 \cdot (1 + g_1 \cdot SIN^2(Lat))/(1 - \varepsilon^2 \cdot SIN^2(Lat))$$

where g_0 = 9.7803267714 [m/s^2] is the size of gravity at the equator,
g_1 = 1.93185139E-3 is a gravity formula constant for the WGS84 Earth and
ε^2 = 6.694378E-3 is geometry constant (first eccentricity squared) of the WGS84 ellipsoid. If your H48C sensor is far away from see level, you can apply a simple altitude correction in the form

$$g(Alt) = g_{WGS84} \cdot (1 - 2 \cdot Alt / r)$$

where r = 6371 [km] is the mean radius of the Earth.

Scale factors and biases

Individual cnt_i sensor readings for each of the axis are transformed to the calibrated a_i accelerations according to the formula

$$a_i = f_i \cdot cnt_i + b_i$$

where f_i is the scale factor for axis i (= x, y, z) and b_i is the offset bias for that axis. The overall temperature dependence of the measured acceleration values are treated for each axis with the same cure f_o as

$$a_i = f_o \cdot (f_i \cdot cnt_i + b_i)$$

Transforming all of these into the equation of a origin centered sphere

$$(f_o)^2 \cdot ((f_x \cdot cnt_x + b_x)^2 + (f_y \cdot cnt_y + b_y)^2 + (f_z \cdot cnt_z + b_z)^2) = g^2$$

and we can count seven calibration constants, f_o, f_x, b_x, f_y, b_y, f_z, b_z, there. Let us take f_o as one but we have then to collect the static calibration data precisely at the same temperature. f_o will be obtained later using *en-route* measured accelerations. This calibration can happen several times during a mission. We have now left six parameters to determine, so we have to use at least six equations with different acceleration counts corresponding to a minimum of six different orientations.

Obtaining static calibration data

A Propeller application **H48C_AcquireSteady** is supplied to assist data collection for the static calibration. This program reads H48C 3-axis data at 5 Hz rate and applies a low-pass digital filter to suppress noise. We have to make six measurements with large +/- cnt_x, +/- cnt_y and +/- cnt_z readings by orienting each of the sensor's axis parallel and antiparallel with the plumb-bob gravity. We may leave some small readings left in the other directions, but the sensor should be kept firmly and steadily in position during each measurement. To enjoy the benefits of a least-squares optimizer I recommend to make six other measurements with 'mixed' components. After the readings stabilized within +/-1 counts, we have to record the data for all the three acceleration components. A typical dataset looks like as

i	cnt_x	cnt_y	cnt_z	Pattern		
Pos.1	-48	-36	1456	0	0	+
Pos.2	-16	5	-1281	0	0	-
Pos.3	-18	1353	-7	0	+	0
Pos.4	16	-1399	-22	0	-	0
Pos.5	1348	44	83	+	0	0
Pos.6	-1415	-18	89	-	0	0
Pos.7	693	-955	794	+	-	+
Pos.8	-774	-938	-645	-	-	-

Pos.9	-717	-970	837	-	-	+
Pos.10	503	-737	-961	+	-	-
Pos.11	-852	672	-766	-	+	-
Pos.12	997	575	783	+	+	+

where I started the data collection with the +-z axis of a H48C sensor.

Finding out the scale and bias parameters

The previous data is entered into the program **H48C_LeastSquare**. This program is a least-squares parameter optimizer for the twelve

$$(f_x \cdot cnt_{xi} + b_x)^2 + (f_y \cdot cnt_{yi} + b_y)^2 + (f_z \cdot cnt_{zi} + b_z)^2 = g^2$$

equations, where each equation contains the corresponding cnt_{xi}, cnt_{yi}, cnt_{zi} counts from the ith rows of the previous table and all of them contain the same six unknown parameters f_x, b_x, f_y, b_y, f_z and b_z. After convergence the program yields the following calibration constants for the given H48C sensor

$$f_x = 7.0922E\text{-}3$$
$$b_x = 2.4358E\text{-}1$$

$$f_y = 7.1001E\text{-}3$$
$$b_y = 1.9121E\text{-}2$$

$$f_z = 7.1628E\text{-}3$$
$$b_z = -6.3008E\text{-}2$$

from the tabulated calibration dataset. These scales and biases are to be recorded on the datasheet of the given H48C sensor and are entered into a verification application.

Verification of the static calibration

With the **H48C_VerifyCalib** program one can verify in practice the static calibration. This program is supplied with the calibration factors and it measures the static acceleration at low rate with using the same low-pass digital filter as in the raw data collection. But this time. the program calculates the calibrated a_x, a_y, a_z accelerations and calculates the magnitude of the measured/calibrated static values

$$SQR[(a_x)^2 + (a_y)^2 + (a_z)^2] = Magn$$

This magnitude should be in the close vicinity of the reference g value. With that given sensor I obtained 9.81 +/- 0.05 [m/s^2] g readings in many (>24) very different static orientations. So, the original 10% (according to datasheet, actually 13%) accuracy of the sensor was improved to 1%. In the sensor array the static calibration factors and biases for each of the individual H48C sensors are to be determined individually and to be stored in the FPU to get a_C and **A** matrix very quickly from the raw sensor readings.

Sensing a state for *en-route* static calibration

When the magnitude of the high-pass component of the measured a_C acceleration of the sensor array remains below a certain low level for several seconds meanwhile the low-pass component has an approximate 9.81 magnitude, then the craft is travelling probably with constant velocity vector. Some sort of Fuzzy Logic can find this out. This vehicle state, with strong low-pass filtering of the acceleration, can be used for en-route static calibration of the sensor array to compensate for temperature variations and/or to adapt to the magnitude of local gravity which might not be accounted exactly by the applied mathematical model of the overall gravity.

cessnapilot, 25.03.2009