Integrated GPS/INS/Magneto Navigation System Built from Parallax Components

In the spirit of ***Learning by Doing*** for the memory of some of my friends, who flew blind bravely with only instruments of the fifties.

...Grau, teurer Freund, ist alle Theorie,
Und grün des Lebens goldner Baum.
Johann Wolfgang Goethe (1749-1832)
Faust, erster Teil, Studierzimmer

The first Inertial navigation System (INS) was designed in 1948. The traditional Inertial Measurement Unit (IMU) of INS uses the linear accelerometer to sense the linear acceleration and the gyroscope to measure angular velocity. It is an inherently stealthy, self contained system that is very difficult to jam. A precise INS does not need other information, satellite signal or any additional instrumentation to determine continuously the position, attitude and speed of the vehicle. Inertial Navigation Systems are widely used in many applications including civilian and military aviation, spatial and nautical segments, automobiles, automated vehicles and robotics.

This living (i.e. periodically enhanced and updated) document follows embedded application projects, based upon the Parallax Propeller microcontroller. First a construction of a gyro-free 6DOF IMU then the build of a 3-axis magnetometer/inclinometer using WMM2005 (or EGRF-2010) magnetic field model will be described. Then their integration with a GPS will be presented.

On the route only SPIN and PASM programming are used and no external fancy PC programs, except the IDE of Propeller and the Propeller Serial Terminal are needed. They are necessary and sufficient to assist the building and testing of those devices. The finished sensor units will be self-contained and independent devices. They will not need any more PC "assistance" to make self-tests, calibrations or to initiate themselves. The operation of these devices will be again fully autonomous.

No Matlab simulations of performances will be done, instead all units will be actually built and the real performances will be tested with real measurements. I take care to publish all software and technical details necessary to reproduce the devices with the announced results by anyone interested.

Why from Parallax components?

Instead of using high dollar sensors from other sources, I decided to build only with sensor modules of Parallax that are aimed primarily for the hobby electronic market. These modules are well designed, robust and easy to use or to communicate with. This makes them an attractive choice for IMU/Magneto projects. However, they apply low-cost and low-accuracy sensor chips. The performance parameters of these sensors are appropriate for hobby electronic projects, but they create quite a challenge to build sophisticated IMU/Magneto units from them. Fortunately, the computing capabilities of the Parallax Propeller microcontroller made it possible to construct surprisingly precise and robust embedded devices from these sensors. I hope that in this way more parallaxians or propeller fans will be interested in the project.

Parallax Inc. has excellent user support for its products and a lot of educational material to assist the newcomers. Parallax manages active Discussion

Forums, where talented programmers, enthusiasts and experts from many fields will help you.

Those lucky ones having more capable sensors, also may benefit from the physics and mathematics learned from these projects, especially from the methods to optimize performance. The neat big boys did not allow the young and very spoiled Eddy Merckx onto the road, so he had to bycicle in the knee-deep sand beside it. When the others first let him to ride on "their" paved road, nobody could overtake the kid, and the carrier of an Amateur World Champion and "The Cannibal" professional started.

Gyro-free IMU

A gyro-free IMU is a specific array of accelerometers where location and orientation are chosen so that angular and linear motions can be decoupled and computed separately. Recent advantages in Micro Electro Mechanical Systems (MEMS) technology have made inertial and magnetic sensors more affordable. The cost of micro-machined accelerometers and gyroscopes are decreasing while their performance are being improved. Micro-machined accelerometers are now in large volume production, cost a few dollars and have been showing reliability. MEMS gyroscopes, besides their higher costs, are less robust than MEMS accelerometers, due to their more complicated inherent structure. The temperature sensitivity of the MEMS accelerometers is usually less then those of the MEMS gyroscopes. The advantages that accelerometer-only IMUs could cause come from the relative simplicity of accelerometers. In general, accelerometers are more reliable, less expensive and require less power than angular rate sensors.

The idea of making gyro-free IMU only from robust accelerometers is not new. Angular velocity measurement without using gyros was first mentioned by DiNapoli in 1965. In 1967 Shuler proposed the gyro-free strap-down scheme. He presumed a vehicle motion analyses requiring at least nine single-axis accelerometers. In theory, a minimum of six accelerometers is required for a complete description of a rigid body motion. In 1994, Chen and Lee was succesfull to present the six-accelerometer scheme. They ignored in the math, however, the effect of gravity and attitude so their original six-accelerometer math formalism could not apply on navigation. The idea was expanded to use 3-axis accelerometers in arbitrary arrangements and the general mathematical framework, including the effect of gravity and attitude, was developed soon. Nowadays, the boiled down mathematics and the advanced calibration methods make the gyro-free IMU designs more and more attractive when compared with the customary triad of mutually orthogonal gyroscopes, which provide direct estimates of angular velocity.

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

...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** 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.

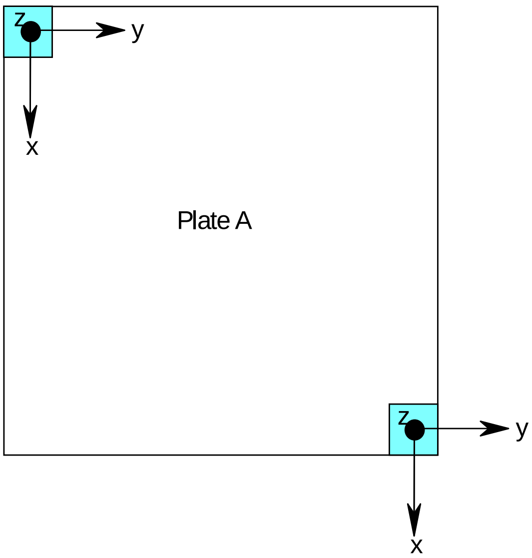

Figure 1. The z-axis of the H48C accelerometers are pointing towards the reader.

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

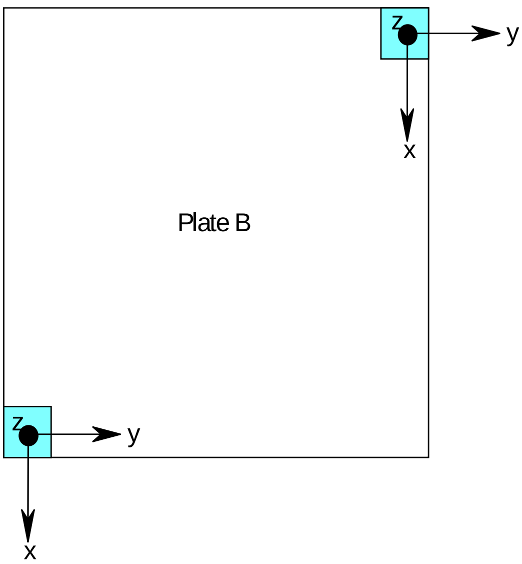

Figure 2. The z-axis of the H48C accelerometers are pointing towards the reader.

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

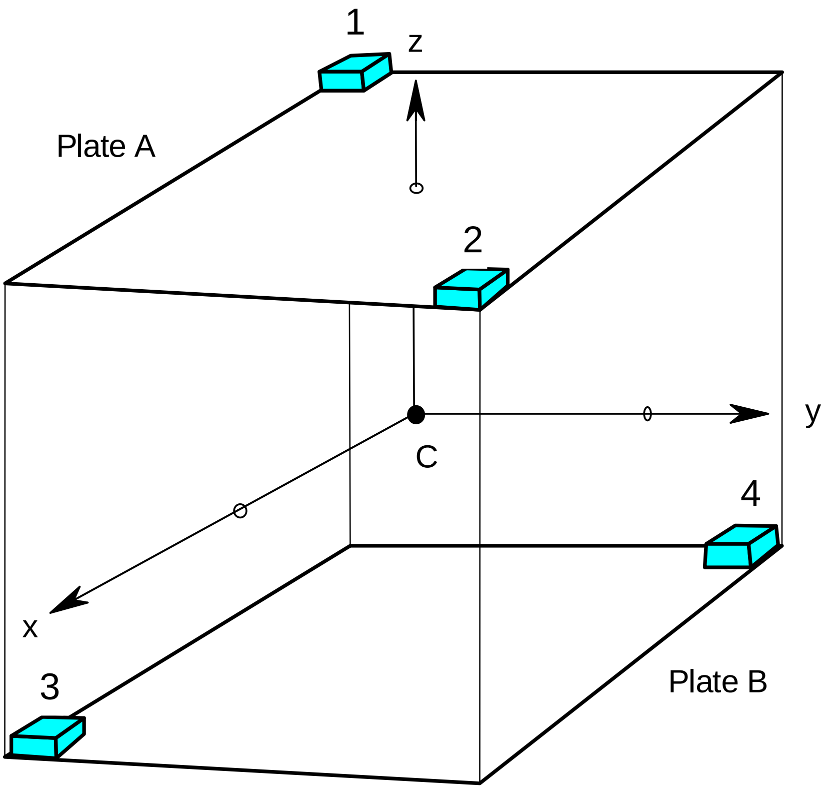

Figure 3. The four H48Cs arranged at the vertices of a tetrahedron.

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.

The next Figure shows the actual arrangement of the sensors at an intermediate stage of mechanical assembly. The estimated distance between the center of each sensors is about 100 mm (±0.5 mm) and the measured side length of the frame cube is 100(±0.2)mm. The brass spacers are accurately machined ones with the length of 67.75(±0.01)mm. The thickness of the carbon composit plates is 3(±0.02)mm. Vertically fitted L shaped Al profiles (not shown) to the corners will give further strength to the design and PCBs will be fixed onto the brass spacers inside.

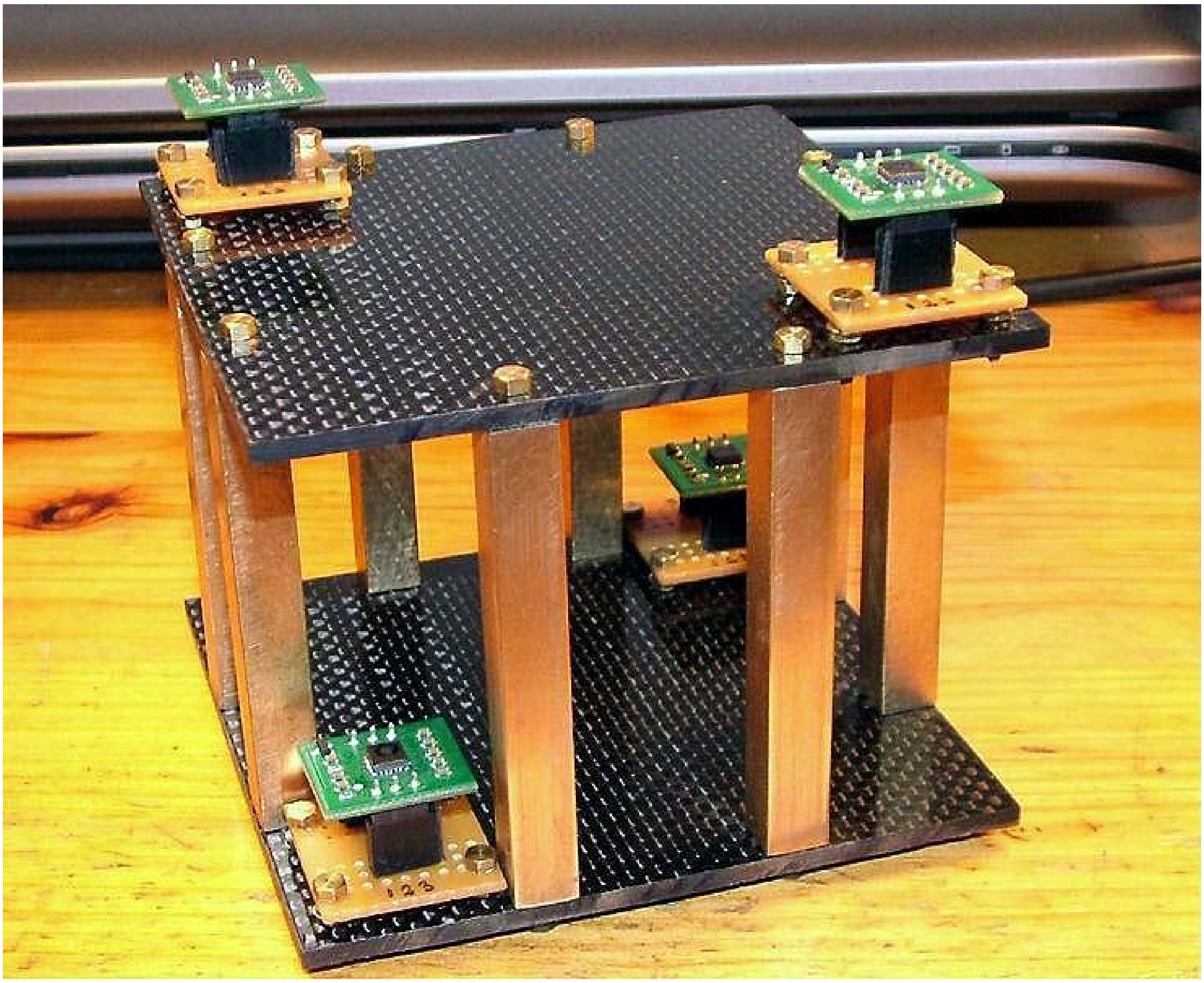

Figure 4. The four H48Cs arranged at the vertices of a tetrahedron.

Definition of matrices

Now we 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

$$
\mathbf{R} = \begin{array}{cccc} r_1 & r_2 & r_3 & r_4 \\ \end{array}
$$

$$
\mathbf{R} = \begin{bmatrix} -0.5 & 0.5 & 0.5 & -0.5 \\ -0.5 & 0.5 & -0.5 & 0.5 \\ 0.5 & 0.5 & -0.5 & -0.5 \end{bmatrix} \quad \text{[3 by 4] matrix}
$$

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

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

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

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

Then subtract a_C from the acceleration vectors to obtain relative accelerations

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

And the **A** matrix is

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

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

$$\mathbf{\underline{P}} = \begin{bmatrix} -0.5 & -0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 \\ 0.5 & -0.5 & -0.5 \\ -0.5 & 0.5 & -0.5 \end{bmatrix} \qquad \text{[4 by 3] matrix}$$

Verify that the **R**·**P** matrix product gives a [3 by 3] identity matrix. OK. 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 $\mathbf{a_C}$ 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

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

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.

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

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 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

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

where SQRT 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

We 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 at the pickup points are as follows

$$a_1 = [\ 0.0,\ 0.0,\ 9.81]$$
$$a_2 = [\ 0.0,\ 0.0,\ 9.81]$$
$$a_3 = [\ 0.0,\ 0.0,\ 9.81]$$
$$a_4 = [\ 0.0,\ 0.0,\ 9.81]$$

First we calculate a_C

$$\mathbf{a_C} = [\ 0.0,\ 0.0,\ 9.81]$$

Then the **A** matrix is

$$\mathbf{A} = \begin{bmatrix} 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}$$

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.

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 $\mathbf{a_C}$ acceleration of the whole sensor in the x direction. According to our formula

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

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

$$a_1 = [\ 1.0,\ 0.0,\ 9.81]$$
$$a_2 = [\ 1.0,\ 0.0,\ 9.81]$$
$$a_3 = [\ 1.0,\ 0.0,\ 9.81]$$
$$a_4 = [\ 1.0,\ 0.0,\ 9.81]$$

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

$$\mathbf{a_C} = [\ 1.0,\ 0.0,\ 9.81]$$

Then the $\underline{\mathbf{A}}$ matrix is

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

The $\underline{\mathbf{W}}$ 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.

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 α x r_i vectors

$$\alpha \times r_1 = [\ 0.25,\ -0.25,\ 0.00\ ]$$
$$\alpha \times r_2 = [\ -0.25,\ 0.25,\ 0.00\ ]$$
$$\alpha \times r_3 = [\ 0.25,\ 0.25,\ 0.00\ ]$$
$$\alpha \times r_4 = [\ -0.25,\ -0.25,\ 0.00\ ]$$

then the a_i vectors

$$a_1 = [\ 1.25,\ -0.25,\ 9.81\ ]$$
$$a_2 = [\ 0.75,\ 0.25,\ 9.81\ ]$$
$$a_3 = [\ 1.25,\ 0.25,\ 9.81\ ]$$
$$a_4 = [\ 0.75,\ -0.25,\ 9.81\ ]$$

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

$$\mathbf{a_C} = [\ 1.0,\ 0.0,\ 9.81]$$

But the **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 **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

$$\mathbf{a_C} = [\ 1.0,\ 0.0,\ 9.81\ ]$$

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

The constituents to the a_i accelerations at the pickup points are

$$\mathbf{a_C} = [\ 1.0,\ 0.0,\ 9.81\ ]$$

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

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

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

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

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 $\mathbf{a_C}$, α and ω vectors. $\mathbf{a_C}$ is simply the average of the four a_i vectors

$$\mathbf{a_C} = [\ 1.0,\ 0.0,\ 9.81\ ]$$

The $\underline{\mathbf{A}}$ 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] $\underline{\mathbf{W}}$ matrix with a simple matrix multiplication

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

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

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

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 precompiled one.

The [3 by 3] product matrix is used to estimate the α *and the* ω *vectors.*

α *is obtained directly from this matrix with a simple formula.*

The absolute value of the components of ω *is calculated again with simple formulas.*

The sign of the components of the ω *vector is obtained from the sign of the sum of the* α*'s components.*

This last method is something like an integration, but the difference between using the sign of a value, or using the value itself, is huge. This can be make numerically more robust with using the previous value of ω for the "integral" before adding the new $\alpha\cdot\Delta t$ increment, because ω is the correct, measured value for that integral of α. Sign uncertainty of the angular velocity will appear only when the size of the ω components shrinks below the noise level. In other words when ω and α components will be very small, and these uncertainties will be random with zero mean. I am tempted to say, that the sign of the noise should not bother us too much. In fact, the physical cross-dependency of the measured α and ω will help us in minimizing the drift of both by software.

If there remains any more drift in ω, that would mean a rotation of the whole sensor array in space. However, such whole body rotations would be sensed by the accelerometers during an arbitrary motion by noticing the rotating gravitation vector. But there will be no rotation of the gravitation vector for the bias of ω. So that bias can be corrected back to close to zero. In more mathematical terms, the drift of ω can be bounded to be less than the highest frequency component of the realistic whole-body movement of the vehicle. This bias elimination method does not work for a steady vehicle with a perfectly leveled 6DOF IMU. This situation does not exist in the real life. Or, if so, then a 3-axis Magnetometer/Inclinometer sensor would resolve that uncertainty, as well.

The General MEMS Sensor model

The MEMS sensors are usually robust and reliable, but contain some imperfections originating from the present incapabilities of the micro machining technology. They are, however, robust and reliable in reproducing those imperfections, as well. So we can count on their systematic behavior to remove those imperfections in a similarly systematic manner.

Static calibration of the individual H48C sensors

Neither all H48C sensors are created equal, nor they are perfect. So it is necessary to do static calibration of them before using their readings in the previously described measuring algorithm. The H48C sensors are calibrated to about 10% accuracy at the factory. They are temperature compensated by design, but there remains some noticable temperature dependence, that should be accounted for during operation. So, there is a lot of place for improvement.

Physics behind static calibration of accelerometers

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 satisfy the equation

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

in every steady 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 calibration parameters will be based upon of many steady measurements and on the previous equation.

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 (1st eccentricity)^2 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 and Alt is the altitude above (or below) see level.

The General Sensor Model object

This object is available at OBEC (**http://obex.parallax.com/objects/464/**). It contains Bias Compensation, Axis Scale Factors, Axis Misalignment and Axis Crosstalk compensation and run-time Temperature Correction in a compact and simple form. This boiled down form is especially well suited for embedded applications with usually limited resources. First, Bias is compensated as

Value = Raw_Reading - Bias

for each axis, then an Ellipsoid to Sphere backtransformation is applied on the vector of the three values. This linear transformation is a simple [3-by-3] matrix [[A]], by which the vector is multiplied. In 3-axis magnetometers this **General Sensor Model** covers Hard- and Soft Iron Compensations, too.

Finally, temperature correction is done with a simple scalar multiplication of the spherified data vector, from which the distortions have been already removed. In summary

[Calibrated vector] = [[A]] · ([Raw Vector] - [Bias Vector])

and the temperature correction factor is calculated on the fly, then applied as

[Calibrated vector] = Temp_Corr · [Calibrated vector]

The mathematical background of this simplicity (y=A·(x-b)) is based on the observation that the many aforementioned distortion effects to be compensated, can be expressed one by one in matrix form. In the **General Sensor Model** we directly measure with the calibration the final product matrix of those many matrices. In this way we do not have to bother with the separate details of formalism beyond those different imperfections.

The object contains the necessary procedure templates that you will need to make your own application with any 3-axis MEMS sensors. Each following step is well separated and commented in the code. With a minimum effort of commenting and decommenting, you can taylor the program for the particular task with your H48C. For other type of sensors, you have to use the appropriate drivers and timings.

Step 1. Acquire steady data for calibration

Here you can collect acceleration data in several steady poses of the H48C modul. You do not have at all to strive for exact alignments, but I recommend to expand a sphere almost uniformly with the measured vectors. Some systematic method of axis and pose changes will help a lot to achieve that. I attached picture of the tools I used in the calibrations. In my method the sensors were fixed with X, Y and Z axis Up and Down into a vertically standing vise (3x2 poses). Then the vise was pitched (0), +45, -45 and -90 degrees (6x4 poses). This was repeated, starting with vertical vise head, but with rotating it (0), 90, 180, 270 degrees horizontally (24x4=96 poses).

Handwriting the calibration data is boring and prone to errors

A full automated version of the **General Sensor Model** object is in preparation, where the collected steady calibration data is written into EEPROM, so you will not have to write them down. This data is automatically read by the Least-Squares Parameter Optimizer, and the optimized parameters are stored in EEPROM, too. The verification of the calibration is automatic, again. This feature will be implemented in the firmware of the finished sensor units, where an external "SELF_CALIBRATE" command will trigger the process. In this way periodic recalibration will be very convenient to cope with the wear and tear of the units, or to self-adjust the magnetometer to some changed magnetic environment.

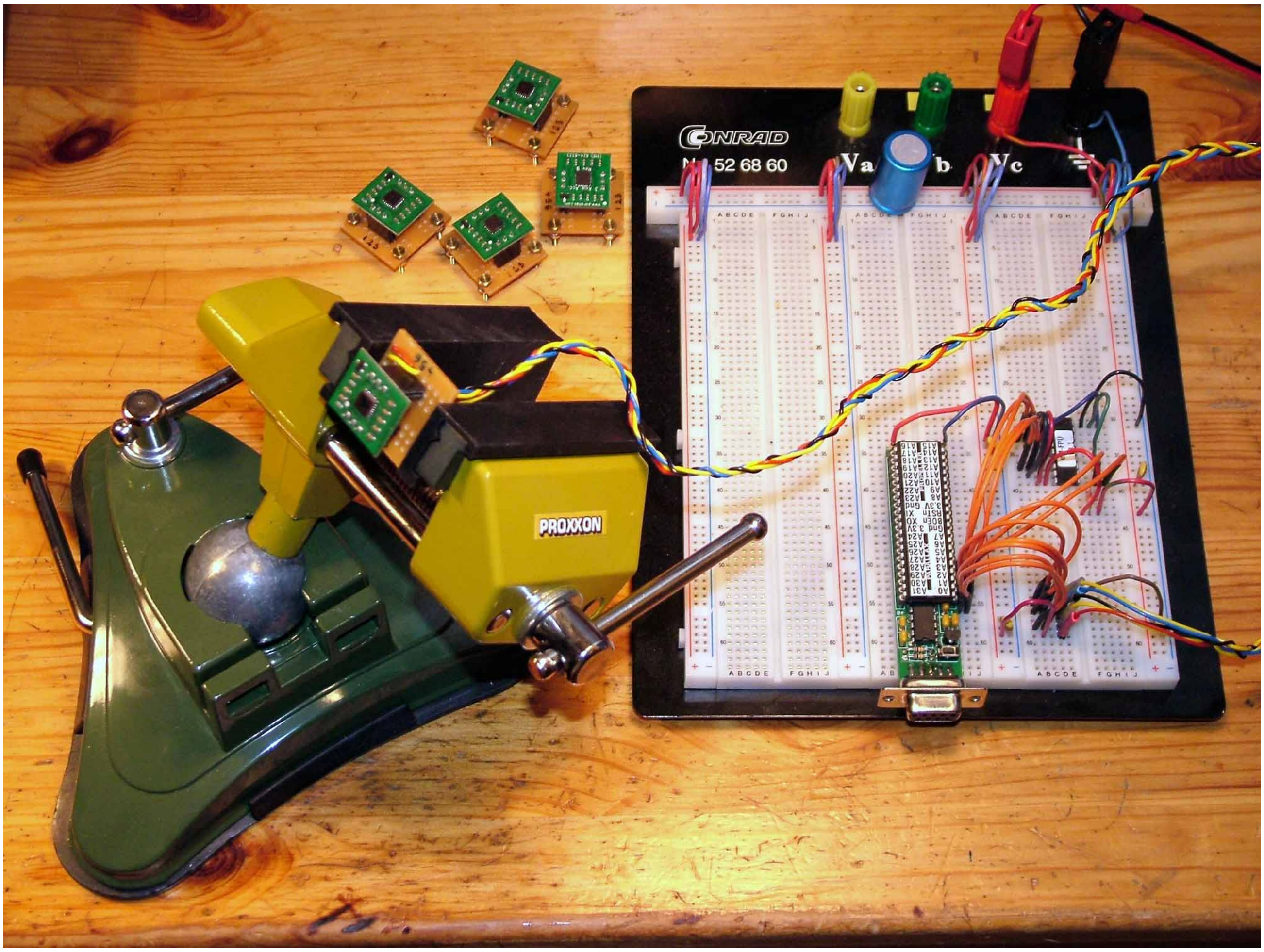

Figure 5. The tools used in the calibration.

This liberty of the alignments during calibration is especially convenient in the tuning of 3-axis magnetometers. You do not have to know where Magnetic North is for the alignments! Again, the point is to cover the sphere approximately uniformly. Here, of course, you have to know the (3D!) magnitude of the Earth's magnetic field at your place.

Temperature should be stable!

You have to collect the static calibration data with stabilized electronics and at the same sensor temperature! The "same" means here within 1 degree of celsius or better. Disobeying this requirement may seriously undermine the quality of your results. The good news are that after the uniform temperature calibration, precise and full automatic temperature correction can be done during true measurements at the unpredictable or changing temperatures of the real application. This automatic temperature correction has of vital importance for MEMS sensors that are usually more of sensors for temperature than sensors of the measured physical effect. Even the so called "temperature compensated" H48C "accelerates" heavily if you put your fingertip on it very gently. MEMS Gyros are even worst in this respect because of their more temperature sensitive internal structure then those of the MEMS accelerometers.

Step 2. Calculation of the parameters

The **General Sensor Model** contains twelve steady parameters. The Bias vector adds three ones and the [[A]] matrix contributes with nine. These parameters are constant for a particular piece of sensor. To find these parameters you are provided with a general Least-Squares Parameter optimizer. First you have to enter your correct and double checked calibration data into the DAT section. Then you have to recompile and run the code to get the those parameters automatically.

The Least-Squares Parameter Optimizer does its job in two shots. It computes preliminary Biases and Scale factors first, then calculates the twelve general parameters in the second run, initiating from the previous results.

The evident outliers from the fit (if there are some) usually identify mistyped or badly measured data. An error greater than five times the displayed standard error might indicate an outlier. Check and correct those cases, but do not remove valid data because of its somewhat larger residual error. Nothing is perfect, including your sensor, and such data carries important information, too. Many outliers or large residual errors probably indicate unstable temperature or poses during the data collection.

In preparation for Step 3. you have to write down the final parameters and then enter them into the corresponding section of the code.

Step 3. Verify the Calibration

Hard work is over. Here you have only to see and enjoy the high accuracy of the General Sensor Model calibration while noticing the beneficial effect of the real-time temperature correction. Individual H48Cs are now performing with at least ±2mg accuracy (0.4%) or usually better. The temperature correction is calculated on the fly during normal operation. The previously homogenized scales and axes by the **General Sensor Model** makes this first-order temperature correction very effective. The precision at full 200 Hz bandwidth, or, in other words, the one standard deviation of the noisy values without digital filtering is about 5 mg (0.5% at 1 g) at 24 degrees of celsius.

Next the calibration method of the ready-made and fixed sensor array will be appended along with the release of the SPIN/PASM full working v1.0 code of the unit.

cessnapilot, 26.06.2009