Onboard Attitude Determination Software for iPhone

2012.02.06 (rev.1), 2014.01.20 (rev.2), 2017.06.26 (rev.3)
by Yosuke Fukushima, ISAS/JAXA Dept. Spacecraft Enginerring
contact e-mail address: fukushima@isas.jaxa.jp

1 Introduction

Today's lecture is aiming to provide you with an experience of programming for an actual embedded system. The target system is selected as iPhone/iPad, because it has several enviroment sensors, hi-performance CPU with large size RAM. This specification is better and faster than those of standard 50Kg - 100Kg scale of small satelltes so far. Also, as iPhone has become one of the most representative and familiar embedded system in daily life for most of people, it seems that you may have chance to work with it in the way to your research activity in the future. The reason why Android phone is not used here, is simply because this lecturer doesn't have it. You can use Android smart phones for the same purpose.

Since you have taken lectures of spacecraft control, now is a suitable chance to bring those ideas into actual control problems of satellite attitude. (Sakai-sensei, as far as I know, has given several lectures on these themes) Today's lecture is divided into two parts: brief lectures on attitude determination, which may be omitted you have known enought to proceed this lecture, and excercises of software programming with minimum tutorials. Although materials used in this lecture are easy to trace, you have much times to go through step by step to complete. In the end of each section of the matrials you can find instructions to do. You are expected to write codes of functions in C language. After completing all the excercises and gathering missing parts of programs, you would have a set of the attitude determination software to run on an iPhone/iPad.

In most of remote systems, such as Spacecraft, UAV, AUV, Rovers, and etc..., have a set of sensors, actuators, and data processors to perform their designed activities with/without human operator's interventions. The data processing structure can be illustrated as follows. In the next page, you can see MMX S/C and iPhone illustrated as sensor-actuator-data Proccessing embedded systems. From this point of view, both are similar, and you can analogize systems inside S/C once you have experienced working on iPhone.

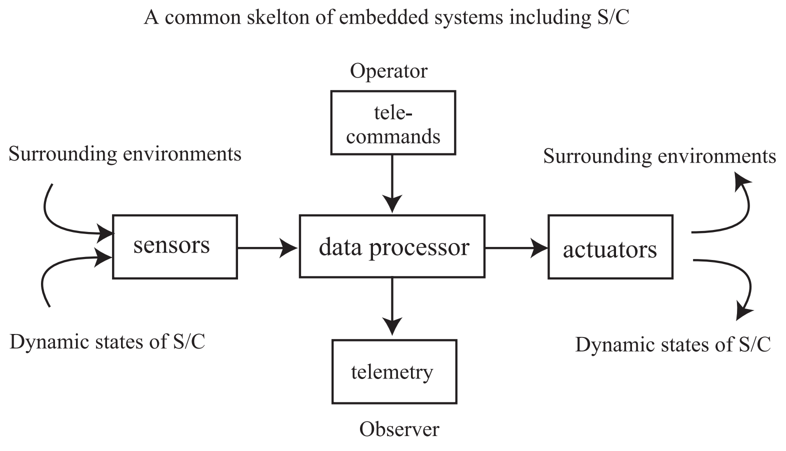
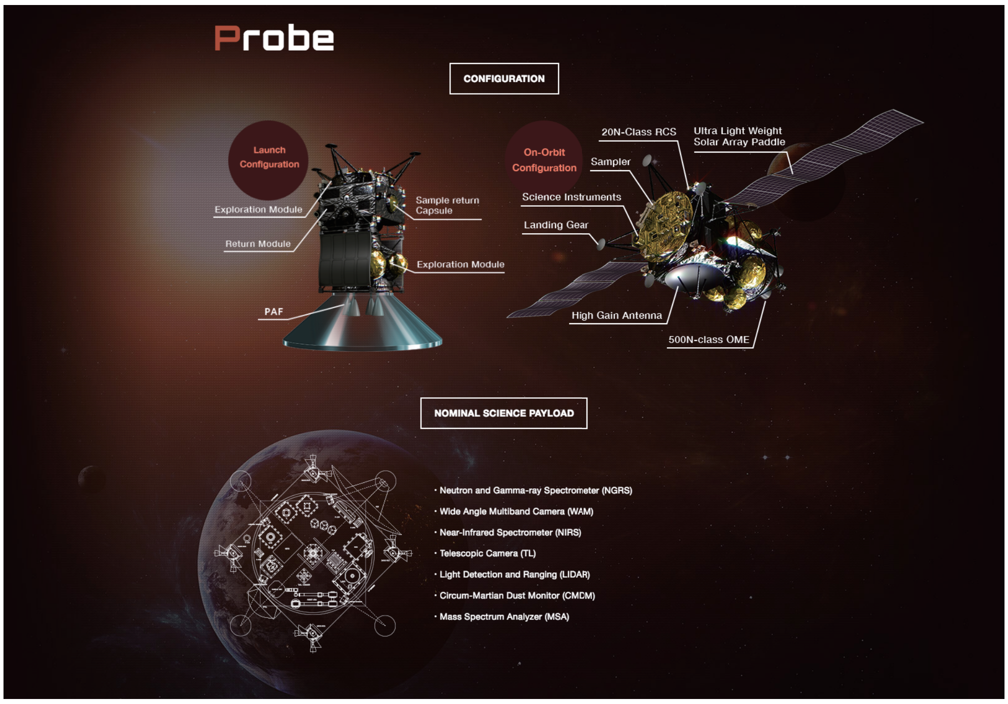
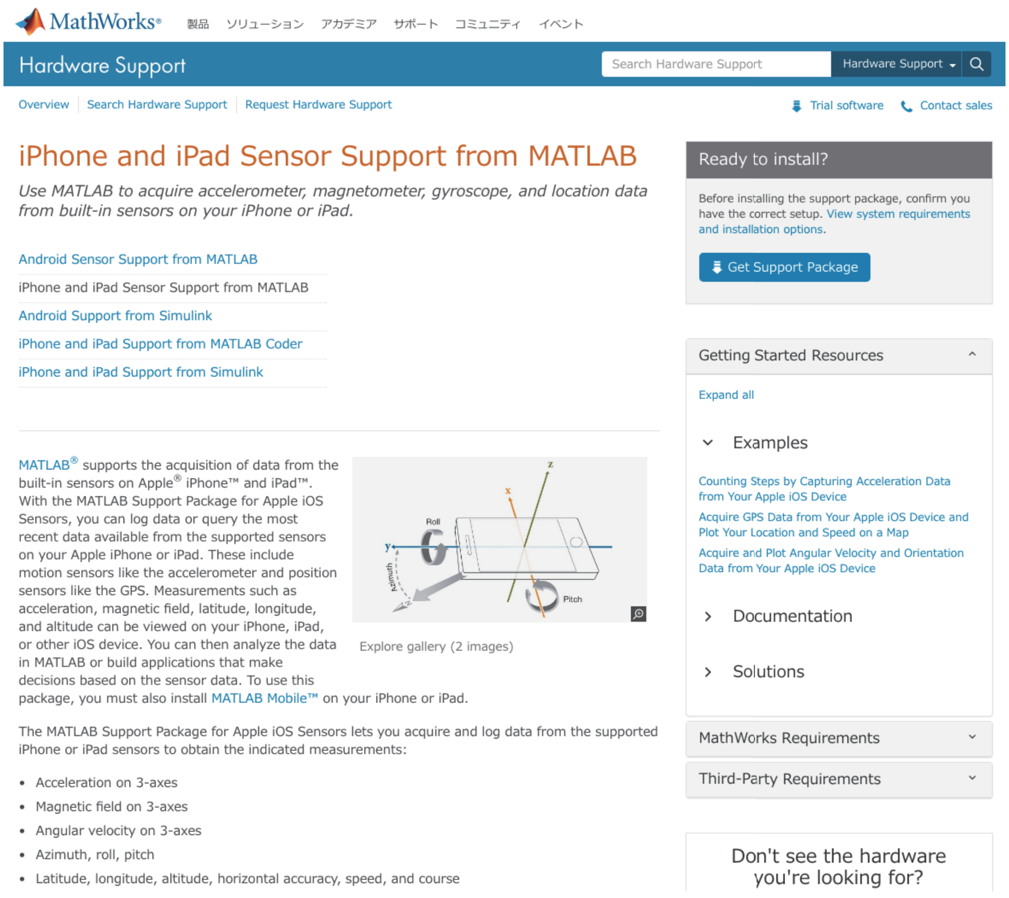


図 1: Simplified data processing structure model of remote systems

If you have any question, you can ask it to the lecturer when you think it better to do so. Your intervetions are always welcome. Note that you have been requested to bring your Laptop PC to edit the program source code files. If you have any C comipler in your PC, you can work with most of excercise except the last one by yourself. If not, each time you finish the excercise, you copy your files onto the lecturer's PC and compile them on it. The resultant binary code for target iPhone/iPad is generated on a MacBook, that is also used to download it to the iPhone/iPad. Those operation of MacBook is performed by the lecturer.




MathWorks® 製品 ソリューション アカデミア サポート コミュニティ イベント

Hardware Support

Search Hardware Support | Hardware Support

Overview | Search Hardware Support | Request Hardware Support | Trial software | Contact sales

iPhone and iPad Sensor Support from MATLAB

Use MATLAB to acquire accelerometer, magnetometer, gyroscope, and location data from built-in sensors on your iPhone or iPad.

Android Sensor Support from MATLAB
iPhone and iPad Sensor Support from MATLAB
Android Support from Simulink
iPhone and iPad Support from MATLAB Coder
iPhone and iPad Support from Simulink

MATLAB® supports the acquisition of data from the built-in sensors on Apple® iPhone™ and iPad™. With the MATLAB Support Package for Apple iOS Sensors, you can log data or query the most recent data available from the supported sensors on your Apple iPhone or iPad. These include motion sensors like the accelerometer and position sensors like the GPS. Measurements such as acceleration, magnetic field, latitude, longitude, and altitude can be viewed on your iPhone, iPad, or other iOS device. You can then analyze the data in MATLAB or build applications that make decisions based on the sensor data. To use this package, you must also install MATLAB Mobile™ on your iPhone or iPad.

Explore gallery (2 images)

The MATLAB Support Package for Apple iOS Sensors lets you acquire and log data from the supported iPhone or iPad sensors to obtain the indicated measurements:

- Acceleration on 3-axes
- Magnetic field on 3-axes
- Angular velocity on 3-axes
- Azimuth, roll, pitch
- Latitude, longitude, altitude, horizontal accuracy, speed, and course

Ready to install?

Before installing the support package, confirm you have the correct setup. View system requirements and installation options.

Get Support Package

Getting Started Resources

Expand all

Examples

Counting Steps by Capturing Acceleration Data from Your Apple iOS Device
Acquire GPS Data from Your Apple iOS Device and Plot Your Location and Speed on a Map
Acquire and Plot Angular Velocity and Orientation Data from Your Apple iOS Device

Documentation

Solutions

MathWorks Requirements

Third-Party Requirements

Don't see the hardware you're looking for?

図 2: MMX S/C and iPhone as embedded systems (https://jp.mathworks.com/hardware-support/iphone-sensor.html, http://mmx.isas.jaxa.jp/en/index.html)

2 Attitude Representation

2.1 Reference frame and body-fixed frame

Attitude of an object is expressed as a coordinate transformation from a reference frame, which is usuall an inertial frame $\{\mathbf{i}\}$, to an body-fixed frame, $\{\mathbf{b}\}$, where $\{\mathbf{e}\}$ is *vector array* expression of a triplet of mutually orthogonal unit vectors as $\mathbf{e_1}, \mathbf{e_2}, \mathbf{e_3}$. There are two major ways of attitude represetation: *Direction Cosine Matrix(DCM) and Queternion.*

A vector **a** can be expressed in many framaes. For example, let **a** be expressed in an inerital frame $\{\mathbf{i}\}$ and a body-fixed frame $\{\mathbf{b}\}$ as

$$\{\mathbf{b}\}^{\mathbf{T}}[a'] = \{\mathbf{i}\}^T[a], \tag{1}$$

where $[a] = [a_1, a_2, a_3]^T$, $\{\mathbf{b}\}.\{\mathbf{b}\}^T = diag(1,1,1)$, $[a'] = [a'_1, a'_2, a'_3]^T$, and $\{\mathbf{i}\}.\{\mathbf{i}\}^T = diag(1,1,1)$.

If the attitude of the object is expressed by C, we can get the following equation.

$$\{\mathbf{b}\} = C\{\mathbf{i}\}. \tag{2}$$

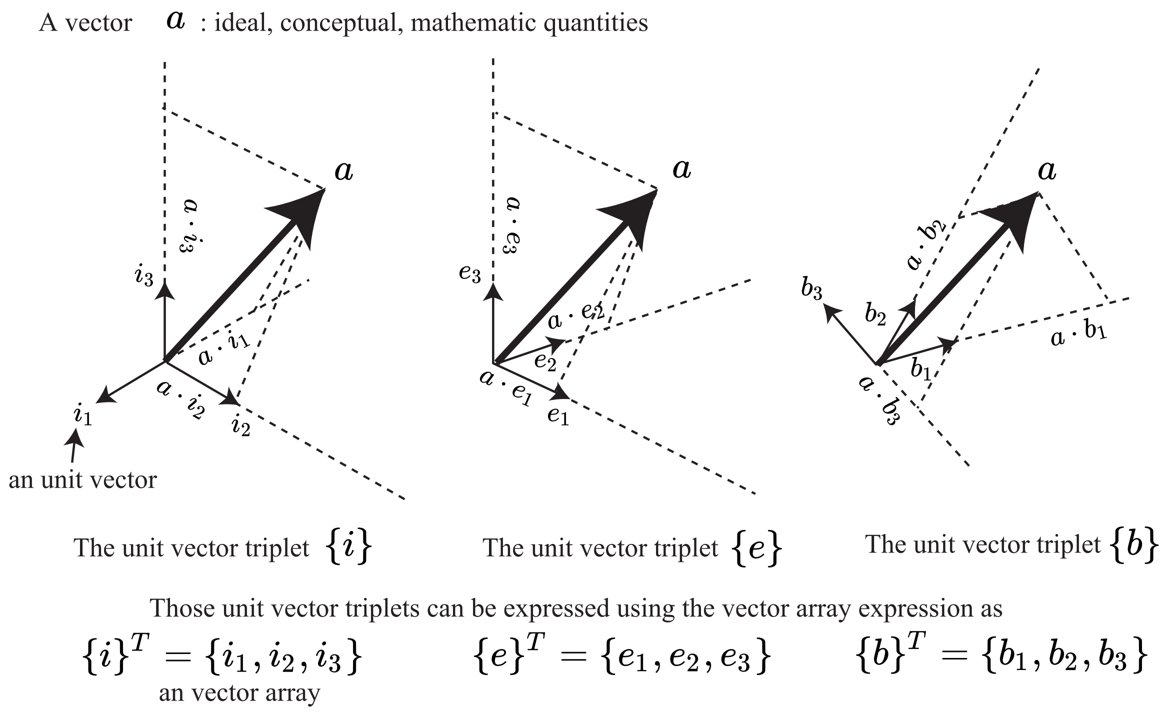
So we can obtain

$$C = \{\mathbf{b}\}.\{\mathbf{i}\}^T = C\{\mathbf{i}\}.\{\mathbf{i}\}^T, \tag{3}$$

where C is called DCM and expressed as

$$C = \begin{pmatrix} \mathbf{b_1}\cdot\mathbf{i_1} & \mathbf{b_1}\cdot\mathbf{i_2} & \mathbf{b_1}\cdot\mathbf{i_3} \\ \mathbf{b_2}\cdot\mathbf{i_1} & \mathbf{b_2}\cdot\mathbf{i_2} & \mathbf{b_2}\cdot\mathbf{i_3} \\ \mathbf{b_3}\cdot\mathbf{i_1} & \mathbf{b_3}\cdot\mathbf{i_2} & \mathbf{b_3}\cdot\mathbf{i_3} \end{pmatrix} \tag{4}$$

The components of DCM are inner product of corresponding vector basis of each frame, illustred as follows.



Those unit vector triplets can be expressed using the vector array expression as

$$\{i\}^T = \{i_1, i_2, i_3\} \qquad \{e\}^T = \{e_1, e_2, e_3\} \qquad \{b\}^T = \{b_1, b_2, b_3\}$$

図 3: Vector array expression and DCM components

2.2 Conversion operation from DCM to Q and vice versa

We have another way to express the attitude: using quaternions (or Euler parameters). Quaternion is derived from a different point of view from that we have when we think about *attitude*. The definition is that *an attitude can be expressed by a simple rotation with respect to an axis from any reference frame*. This definition is a theorem found by Euler, and mathematically formulated by Euler, Gauss, and Hamilton(Quaternion is forumulated by Halmiton).

If you notate the direction axis of rotation and its angle as λ, θ, respectively, quaterion, $q = (q_1, q_2, q_3, q_4)$, is as follows:

$$q_1 = \lambda_1 \sin(\theta/2), q_2 = \lambda_2 \sin(\theta/2), q_3 = \lambda_3 \sin(\theta/2), q_4 = \cos(\theta/2) \tag{5}$$

We have one constraint on quoternion as,

$$q_1^2 + q_2^2 + q_3^3 + q_4^4 = 1. \tag{6}$$

If a DCM C indicates the same orientation as q, there is a equation of trasnformation between two:

$$C = \begin{pmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_2q_1 - q_3q_4) & q_2^2 - q_3^2 - q_1^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_3q_1 + q_2q_4) & 2(q_3q_2 - q_1q_4) & q_3^2 - q_1^2 - q_2^2 + q_4^2 \end{pmatrix} \tag{7}$$

I'm not going through the characteristics of quaternion father now. The only thing you should remember is that the quaternions is equivalent for the DCM, and they can be converted mathematically each other.

Now you are requested to work the follwing three excersise as instruected.

2.2.1 [excercise] q to DCM

Complete the function **q2dcm** in the file named **step1.c**. There are some information on this transformation in the appendix.

int q2dcm(double c[3*3], double q[4]);

2.2.2 [excercise] additive operation

Complete the function **qadd** in the file named **step1.c**. There are some information on this operation in the appendix.

int qadd(double ans[q], double q[4], double dq[4]);

2.2.3 [excercise] DCM to q

Complete the function **dcm2q** in the file named **step1.c**. There are some information on this transformation in the appndix.

int dcm2q(double q[4], double c[3*3]);

3 Vector Operation Functions

To get familar with the programming of vector operations in C, write the following fuctions. Some are partially written, so you should complete all the functions. Those functions will be used as important subsystems of the today's goal.

3.1 [excercise] copy

Complete the function **copy3** and **copy4** in the file named **step2.c**. These functions copy the 2nd argument into 1st argument.

void copy3(double ans[3], double v[3]); void copy4(double ans[4], double v[4]);

3.2 [excercise] norm

Complete the function **norm** and **norm3** in the file named **step2.c**. These functions return the norm-2 values of the argument 3x1 array and 4x1 array.

double norm3(double v[3]); double norm4(double v[4]);

3.3 [excercise] normalize

Complete the function **normalize3** and **normalize4** in the file named **step2.c**. These fuctions modify the argument of 3x1 array into a unit vector and return the norm of the argument vector.

double normalize3(double v[3]); double normalize4(double v[4]);

3.4 [excercise] dot

Complete the function **dot** in the file named **step2.c**. This function returns the dot-proct value of 2 3x1 arrays.

double dot3(double u[3], double v[3]);

3.5 [excercise] cross

Complete the function **cross** in the file named **step2.c**. This functions set the first argument of 3x1 array by the cross-proct of the 2nd and 3rd argument of 3x1 array. The return values are norm values of the cross-proct vectors.

double cross3(double ans[3], double u[3], double v[3]);

3.6 [excercise] mul

Complete the function **mult** in the file named **step2.c**. This function set the frist argument of 3*3 array by the product of the 2nd and the 3rd agument of 3*3 array.

void mul33(double ans[3*3], double a[3*3], double b[3*3]);

3.7 [excercise] tr

Complete the function **tr** in the file named **step2.c**. This function modifies the argument of 3*3 array by the transposed matrix of it.

void tr33(doub m[3*3]);

4 Simple Attitude Determination Algorithm

4.1 TRIAD

One of the easy-to-implement algorithms of attitude detemination is “TRIAD”. TRIADI is easy to understand, simple, deterministic and straight-forward, so that there is no difficaulty to implement even when we use a standard programming language, such as C.

With a set of two non-coaxial (non parallel) vectors that are expressed both in the reference and body-fixed frames, the attitude (pose) of the object, with respect to the reference frame, can be expressed. Let us think two vectors **G** and **M**, expressed in both {**b**} and {**i**} as

$$\mathbf{g} = \frac{\mathbf{G}}{|\mathbf{G}|} = \{\mathbf{b}\}^T[g'] = \{\mathbf{i}\}^T[g] \tag{8}$$

$$\mathbf{m} = \frac{\mathbf{M}}{|\mathbf{M}|} = \{\mathbf{b}\}^T[m'] = \{\mathbf{i}\}^T[m]. \tag{9}$$

Normalized two vectors, **g** and **m**, are not in pararallel, the third vector, **s**, can be calculated by the cross-product operation as

$$\mathbf{s} = \frac{1}{\sin(\theta)}(\mathbf{g} \times \mathbf{m}), \tag{10}$$

where θ is the angle between **g** and **m**.
Also, we can obtain **t** by

$$\mathbf{t} = \mathbf{g} \times \mathbf{s}. \tag{11}$$

By applying the vector $\mathbf{i_i}$ from the left sides of **g**, **s**, **t**, we have the following equations:

$$\mathbf{i_1} \cdot \mathbf{g} = \mathbf{i_1} \cdot \{\mathbf{b}\}^T[g'] = g_1 \tag{12}$$

$$\mathbf{i_2} \cdot \mathbf{g} = \mathbf{i_2} \cdot \{\mathbf{b}\}^T[g'] = g_2 \tag{13}$$

$$\mathbf{i_3} \cdot \mathbf{g} = \mathbf{i_3} \cdot \{\mathbf{b}\}^T[g'] = g_3 \tag{14}$$

$$\mathbf{i_1} \cdot \mathbf{s} = \mathbf{i_1} \cdot \{\mathbf{b}\}^T[s'] = s_1 \tag{15}$$

$$\mathbf{i_2} \cdot \mathbf{s} = \mathbf{i_2} \cdot \{\mathbf{b}\}^T[s'] = s_2 \tag{16}$$

$$\mathbf{i_3} \cdot \mathbf{s} = \mathbf{i_3} \cdot \{\mathbf{b}\}^T[s'] = s_3 \tag{17}$$

$$\mathbf{i_1} \cdot \mathbf{t} = \mathbf{i_1} \cdot \{\mathbf{b}\}^T[t'] = t_1 \tag{18}$$

$$\mathbf{i_2} \cdot \mathbf{t} = \mathbf{i_2} \cdot \{\mathbf{b}\}^T[t'] = t_2 \tag{19}$$

$$\mathbf{i_3} \cdot \mathbf{t} = \mathbf{i_3} \cdot \{\mathbf{b}\}^T[t'] = t_3. \tag{20}$$

Above equations are identical to the conponets of DCM between two frames expressed as

$$\{b\} = C\{i\}. \tag{21}$$

The following figure illustrates relations between those components and two frames.

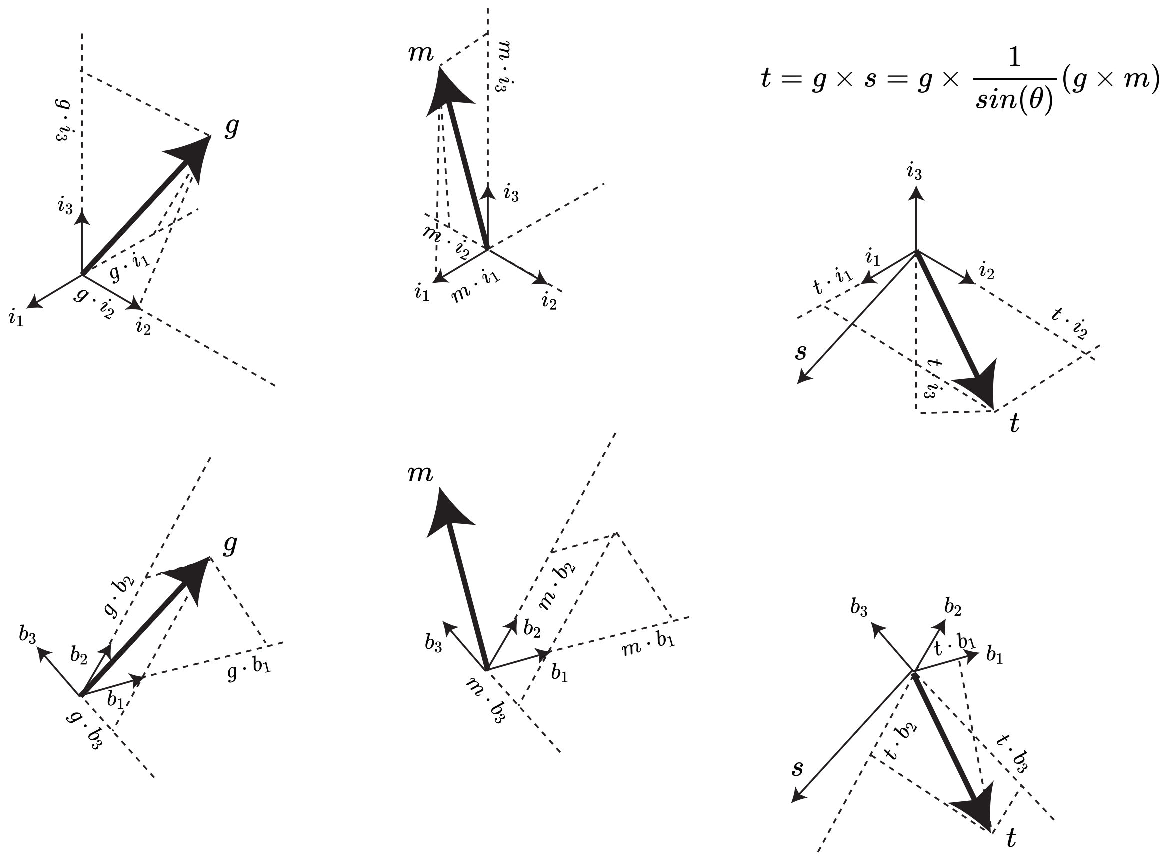


図 4: Inner Product of two vectors selected from two reference frames

Composing all relations between vectors as a matrix, we can derive a DCM as,

$$[[g][s][t]] = \{\mathbf{i}\} \cdot \{\mathbf{b}\}^T [[g'][s'][t']] = \{\mathbf{i}\} \cdot \{\mathbf{i}\}^T C^T [[g'][s'][t']] = C^T [[g'][s'][t']], \tag{22}$$

which results

$$[[g][s][t]] = C^T [[g'][s'][t']], \tag{23}$$

or

$$\begin{aligned} C &= [[[g][s][t]][[g'][s'][t']]^{-1}]^T \\ &= [[[g][s][t]][[g'][s'][t']]^T]^T. \end{aligned} \tag{24}$$

Note that the transpose matrix of orthogonal matrix is identical to the inverse matrix of it. Once we obtained C, its corresponding quaternion can be obtained by the format converstion operation developed in section 2.

4.2 [execercise] triad

After understanding the path of developping of all the equations above, complete the function 'triad' in the file step3.c. This function sets the frist argument of 4x1 array through the sequence of operations using the the 2nd to 5th aguments of 3x1 arrays and returns the angle between two vectors in radian.

double triad(double q[4], double Gdash[3], double Mdash[3], double G[3], double M[3]);

5 Vector Observation

With four vectors, two in a reference frame and two in a body-fixed frame, we can determine the attitude of any object by in DCM or quaternion. In actual satellite cases, these are two observation vecotors and two are theoritical vectors. An typical set is a couple of Sun direction and Geomagnetic field direction, which are frequently used in corse accuracy attitude estimation since theoritical calculation using well-known models can be done onboard.

However, iPhones have no Sun sensor, we have to use another vector. In this lecture, we use the gravity direction vector. With one gravity vector (or accelemeter measure) as **G** and one magnetic field vector as **M**, we can determine the attitude of the iPhone.

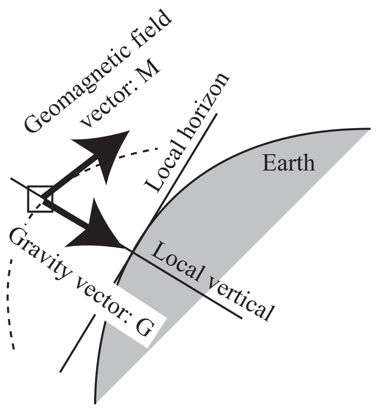


図 5: Two vectors for TRIAD are selected as the geomagnetic field vector and the gravity vector

The next figure shows the data flow from incoming source of sensors into the data processor. Once after we have obtain updated data set of gravity and magnetic field vector, we are to calculate the correspoing theoritical vectors in the referece frame, and then put those four vectors into the TRIAD logic to obtain its DCM or quaternion. The update cycle is depends on the sensor-update-cycle and the computational peformance of the onboard processor we use.

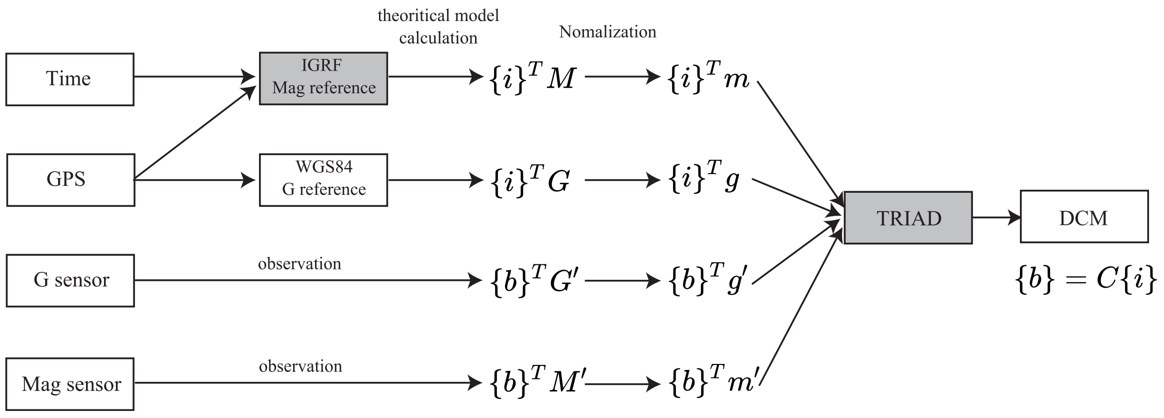


図 6: Data flow from source sensors to attitude data calculation processor

[excercise] check the observation vector handling

Before proceeding to the excercise using an actual system, we should check our software programning result by running it on a debug system. In the step4.c, there is a skeleton of TRIAD software, and you can complete the code by adding and modifying it using functions you have completed. After that, you can compile and run it to see the result on the reference PC. The dummy sensor data are stored in the following variables. There are fixed values.

double obs_g[3], obs_m[3], ref_g[3], ref_m[3];

6 iPhone Attitude Determination

After going through the section 1 to 4, you are now ready to upload your own code of attitude determination subsystem to an iPhone. The target iPhone is the lecture's, which has been registered to accept user-developed program just for testing.

Please copy all the files you have completed, step1.c, step2.c step3.c and step4.c onto the USB-memory device. Althoug the souce code can be used in several systems such as Windows, Linux, and Mac, you have to compile them using Xcode on MacOS to get the executable code for iPhones.

The downloading procedure will be done on the lecturer's MacBook and uploaded from it. After uploading, you have to check the software of attitude determination system of the iPhone by your self.

The sequece of operation is shown here:

1. upload the code
2. flick the badge "AttDet" to start the program
3. select "GPS postion" to activate the GPS sensor
4. select "G Gravitaion" to activate the G sensor (**g** vector)
5. select "Mag field" to activate the M sensor (**m** vector)
6. select "TRIAD" to start the attitude determination
7. take data of q by making it facing in several directions.

That's it.

at least one eigenvector with eigenvalue unity. That is, there exists a unit vector, $\hat{\mathbf{e}}$, that is unchanged by A:

$$A\hat{\mathbf{e}} = \hat{\mathbf{e}} \tag{12-5}$$

The vector $\hat{\mathbf{e}}$ has the same components along the body axes and along the reference axes. Thus, $\hat{\mathbf{e}}$ is a vector along the axis of rotation. The existence of $\hat{\mathbf{e}}$ demonstrates Euler's Theorem: *the most general displacement of a rigid body with one point fixed is a rotation about some axis.*

We regard the direction cosine matrix as the fundamental quantity specifying the orientation of a rigid body. However, other parameterizations, as summarized in Table 12-1 and discussed more fully below, may be more convenient for specific applications. In each case, we will relate the parameters to the elements of the direction cosine matrix. Our treatment follows earlier work by Sabroff, *et al.*, [1965].

Table 12-1. Alternative Representations of Three-Axis Attitude

PARAMETERIZATION	NOTATION	ADVANTAGES	DISADVANTAGES	COMMON APPLICATIONS
DIRECTION COSINE MATRIX	$A = [A_{ij}]$	NO SINGULARITIES NO TRIGONOMETRIC FUNCTIONS CONVENIENT PRODUCT RULE FOR SUCCESSIVE ROTATIONS	SIX REDUNDANT PARAMETERS	IN ANALYSIS, TO TRANSFORM VECTORS FROM ONE REFERENCE FRAME TO ANOTHER
EULER AXIS/ANGLE	$\mathbf{e}, \Phi$	CLEAR PHYSICAL INTERPRETATION	ONE REDUNDANT PARAMETER AXIS UNDEFINED WHEN SIN Φ=0 TRIGONOMETRIC FUNCTIONS	COMMANDING SLEW MANEUVERS
EULER SYMMETRIC PARAMETERS (QUATERNION)	q_1, q_2, q_3, q_4 (q)	NO SINGULARITIES NO TRIGONOMETRIC FUNCTIONS CONVENIENT PRODUCT RULE FOR SUCCESSIVE ROTATIONS	ONE REDUNDANT PARAMETER NO OBVIOUS PHYSICAL INTERPRETATION	ONBOARD INERTIAL NAVIGATION
GIBBS VECTOR	$\mathbf{g}$	NO REDUNDANT PARAMETERS NO TRIGONOMETRIC FUNCTIONS CONVENIENT PRODUCT RULE FOR SUCCESSIVE ROTATIONS	INFINITE FOR 180-DEG ROTATION	ANALYTIC STUDIES
EULER ANGLES	ϕ, θ, ψ	NO REDUNDANT PARAMETERS PHYSICAL INTERPRETATION IS CLEAR IN SOME CASES	TRIGONOMETRIC FUNCTIONS SINGULARITY AT SOME θ NO CONVENIENT PRODUCT RULE FOR SUCCESSIVE ROTATIONS	ANALYTIC STUDIES INPUT/OUTPUT ONBOARD ATTITUDE CONTROL OF 3-AXIS STABILIZED SPACECRAFT

Euler Axis/Angle. A particularly simple rotation is one about the 3 axis by an angle Φ, in the positive sense, as illustrated in Fig. 12-2. The direction cosine matrix for this rotation is denoted by $A_3(\Phi)$; its explicit form is

$$A_3(\Phi) = \begin{bmatrix} \cos\Phi & \sin\Phi & 0 \\ \ominus\sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12-6a}$$

DCMは右下が -1
[⊖]

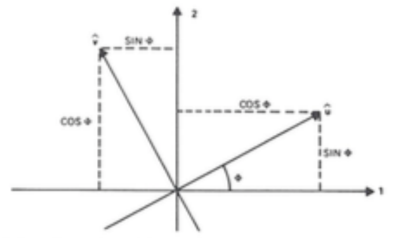


Fig. 12-2. Rotation About the Three-Axis by the Angle Φ

The direction cosine matrices for rotations by an angle Φ about the 1 or 2 axis, denoted by $A_1(\Phi)$ and $A_2(\Phi)$, respectively, are

$$A_1(\Phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{bmatrix} \tag{12-6b}$$

$$A_2(\Phi) = \begin{bmatrix} \cos\Phi & 0 & -\sin\Phi \\ 0 & 1 & 0 \\ \sin\Phi & 0 & \cos\Phi \end{bmatrix} \tag{12-6c}$$

The matrices $A_1(\Phi)$, $A_2(\Phi)$, and $A_3(\Phi)$ all have the trace

$$\operatorname{tr}(A(\Phi)) = 1 + 2\cos\Phi \tag{12-6d}$$

The trace of a direction cosine matrix representing a rotation by the angle Φ about an arbitrary axis takes the same value. This result, which will be used without proof below, follows from the observation that the rotation matrices representing rotations by the same angle about different axes can be related by an orthogonal transformation, which leaves the trace invariant (see Appendix C).

In general, the axis of rotation will not be one of the reference axes. In terms of the unit vector along the rotation axis, $\hat{\mathbf{e}}$, and angle of rotation, Φ, the most general direction cosine matrix is

$$A = \begin{bmatrix} \cos\Phi + e_1^2(1-\cos\Phi) & e_1e_2(1-\cos\Phi) + e_3\sin\Phi & e_1e_3(1-\cos\Phi) - e_2\sin\Phi \\ e_1e_2(1-\cos\Phi) - e_3\sin\Phi & \cos\Phi + e_2^2(1-\cos\Phi) & e_2e_3(1-\cos\Phi) + e_1\sin\Phi \\ e_1e_3(1-\cos\Phi) + e_2\sin\Phi & e_2e_3(1-\cos\Phi) - e_1\sin\Phi & \cos\Phi + e_3^2(1-\cos\Phi) \end{bmatrix} \tag{12-7a}$$

$$= \cos\Phi\mathbf{1} + (1-\cos\Phi)\hat{\mathbf{e}}\hat{\mathbf{e}}^{\mathrm{T}} - \sin\Phi E \tag{12-7b}$$

where $\hat{\mathbf{e}}\hat{\mathbf{e}}^{\mathrm{T}}$ is the outer product (see Appendix C) and E is the skew-symmetric matrix

$$E \equiv \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{bmatrix} \tag{12-8}$$

This representation of the spacecraft orientation is called the *Euler axis and angle* parameterization. It appears to depend on four parameters, but only three are independent because $|\hat{\mathbf{e}}| = 1$. It is a straightforward exercise to show that A defined by Eq. (12-7) is a proper real orthogonal matrix and that $\hat{\mathbf{e}}$ is the axis of rotation, that is, $A\hat{\mathbf{e}} = \hat{\mathbf{e}}$. The rotation angle is known to be Φ because the trace of A satisfies Eq. (12-6d).

It is also easy to see that Eq. (12-7) reduces to the appropriate one of Eqs. (12-6) when $\hat{\mathbf{e}}$ lies along one of the reference axes. The Euler rotation angle, Φ, can be expressed in terms of direction cosine matrix elements by

$$\cos\Phi = \frac{1}{2}[\operatorname{tr}(A) - 1] \tag{12-9}$$

If $\sin\Phi \neq 0$, the components of $\hat{\mathbf{e}}$ are given by

$$e_1 = (A_{23} - A_{32})/(2\sin\Phi) \qquad (12\text{-}10a)$$

$$e_2 = (A_{31} - A_{13})/(2\sin\Phi) \qquad (12\text{-}10b)$$

$$e_3 = (A_{12} - A_{21})/(2\sin\Phi) \qquad (12\text{-}10c)$$

Equation (12-9) has two solutions for Φ, which differ only in sign. The two solutions have axis vectors $\hat{\mathbf{e}}$ in opposite directions, according to Eq. (12-10). This expresses the fact that a rotation about $\hat{\mathbf{e}}$ by an angle Φ is equivalent to a rotation about $-\hat{\mathbf{e}}$ by $-\Phi$.

Euler Symmetric Parameters. A parameterization of the direction cosine matrix in terms of *Euler symmetric parameters* q_1, q_2, q_3, q_4 has proved to be quite useful in spacecraft work. These parameters are not found in many modern dynamics textbooks, although Whittaker [1937] does introduce them and they are discussed by Sabroff, *et al.*, [1965]. They are defined by

$$q_1 \equiv e_1 \sin\frac{\Phi}{2} \qquad (12\text{-}11a)$$

$$q_2 \equiv e_2 \sin\frac{\Phi}{2} \qquad (12\text{-}11b)$$

$$q_3 \equiv e_3 \sin\frac{\Phi}{2} \qquad (12\text{-}11c)$$

$$q_4 \equiv \cos\frac{\Phi}{2} \qquad (12\text{-}11d)$$

The four Euler symmetric parameters are not independent, but satisfy the constraint equation

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \qquad (12\text{-}12a)$$

These four parameters can be regarded as the components of a quaternion,

$$\boldsymbol{q} \equiv \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \equiv \begin{bmatrix} \mathbf{q} \\ q_4 \end{bmatrix} \qquad (12\text{-}12b)$$

Quaternions are discussed in more detail in Appendix D. The Euler symmetric parameters are also closely related to the *Cayley-Klein parameters* [Goldstein, 1950].

The direction cosine matrix can be expressed in terms of the Euler symmetric parameters by

q→DCMの変換式

$$A(\boldsymbol{q}) = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \qquad (12\text{-}13a)$$

$$= (q_4^2 - \mathbf{q}^2)\mathbf{1} + 2\mathbf{q}\mathbf{q}^{\mathrm{T}} - 2q_4 Q \qquad (12\text{-}13b)$$

where Q is the skew-symmetric matrix

$$Q \equiv \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \qquad (12\text{-}13c)$$

These equations can be verified by substituting Eqs. (12-11) into them, using some trigonometric identities, and comparing them with Eq. (12-7).

The Euler symmetric parameters corresponding to a given direction cosine matrix, A, can be found from

DCM→qの変換式

$$q_4 = \pm\frac{1}{2}(1 + A_{11} + A_{22} + A_{33})^{1/2} \qquad (12\text{-}14a)$$

$$q_1 = \frac{1}{4q_4}(A_{23} - A_{32}) \qquad (12\text{-}14b)$$

$$q_2 = \frac{1}{4q_4}(A_{31} - A_{13}) \qquad (12\text{-}14c)$$

$$q_3 = \frac{1}{4q_4}(A_{12} - A_{21}) \qquad (12\text{-}14d)$$

Aはqの2次式なので、全部の符号が「正・負逆」でも(同じ[illegible])

Note that there is a sign ambiguity in the calculation of these parameters. Inspection of Eq. (12-13) shows that changing the signs of all the Euler symmetric parameters simultaneously does not affect the direction cosine matrix. Equations (12-14) express one of four possible ways of computing the Euler symmetric parameters. We could also compute

$$q_1 = \pm\frac{1}{2}(1 + A_{11} - A_{22} - A_{33})^{1/2}$$

$$q_2 = \frac{1}{4q_1}(A_{12} + A_{21})$$

これと注意は qのとりかた こと

and so forth. All methods are mathematically equivalent, but numerical inaccuracy can be minimized by avoiding calculations in which the Euler symmetric parameter appearing in the denominator is close to zero. Other algorithms for computing Euler symmetric parameters from the direction cosine matrix are given by Klumpp [1976].

Euler symmetric parameters provide a very convenient parameterization of the attitude. They are more compact than the direction cosine matrix, because only four parameters, rather than nine, are needed. They are more convenient than the Euler axis and angle parameterization (and the Euler angle parameterizations to be considered below) because the expression for the direction cosine matrix in terms of Euler symmetric parameters does not involve trigonometric functions, which require time-consuming computer operations. Another advantage of Euler symmetric parameters is the relatively simple form for combining the parameters for two individual rotations to give the parameters for the product of the two rotations. Thus, if

$$A(\boldsymbol{q}'') = A(\boldsymbol{q}')A(\boldsymbol{q}) \qquad (12\text{-}15a)$$