


STARLINO


- [Articles](#)
- [SerialChart](#)
- [Products](#)
 - [Motion Gamepad](#)
 - [Acc_Gyro 6DOF Analog IMU: 3 Axis Accelerometer + 3 Axis Gyro](#)
 - [Single Axis Analog Gyroscope with Noise and Drift Filters](#)
 - [Usb Thumb Sized Pic Development Platform \(PIC18F14K50\)](#)
- [About](#)
- [Contact](#)



Conductivity Measurement

Theory Guide

All you need to know about conductivity. Get a guide now!



DCM Tutorial – An Introduction to Orientation Kinematics

Posted on: May 27, 2011 by *starlino*

Category: [Featured](#), [IMU Theory and Experiments](#)

Tags: [Acc_Gyro](#), [accelerometer](#), [dcm](#), [gyro](#), [gyroscope](#), [imu](#), [inclination](#), [kinematics](#), [mems](#), [orientation](#), [sensor](#)

Introduction

This article is a continuation of my [IMU Guide](#), covering additional orientation kinematics topics. I will go through some theory first and then I will present a practical example with code build around an Arduino and a 6DOF IMU sensor (acc_gyro_6dof). The scope of this experiment is to create an algorithm for fusing gyroscope and accelerometer data in order to create an estimation of the device orientation in space. Such an algorithm was already presented in part 3 of my “IMU Guide” and a practical Arduino experiment with code was presented in the “[Using a 5DOF IMU](#)” article and was nicknamed “Simplified Kalman Filter”, providing a simple alternative to the well known Kalman Filter algorithm. In this article we’ll use another approach utilizing the DCM (Direction Cosine Matrix). For the reader that is unfamiliar with MEMS sensors it is recommended to read Part 1 and 2 of the IMU Guide article. Also for following the experiments presented in this text it is recommended to acquire an Arduino board and an [acc_gyro_6dof](#) sensor.

Prerequisites

No really advanced math is necessary. Find a good book on matrix operations, that’s all you might need above school math course. If you would like to refresh your knowledge below are some quick articles:

Cartesian Coordinate System – http://en.wikipedia.org/wiki/Cartesian_coordinate_system

Rotation – http://en.wikipedia.org/wiki/Rotation_%28mathematics%29

Vector scalar product – http://en.wikipedia.org/wiki/Dot_product

Vector cross product – http://en.wikipedia.org/wiki/Cross_product

Matrix Multiplication – http://en.wikipedia.org/wiki/Matrix_multiplication

Block Matrix – http://en.wikipedia.org/wiki/Block_matrix

Transpose Matrix – <http://en.wikipedia.org/wiki/Transpose>

Triple Product – http://en.wikipedia.org/wiki/Triple_product

Notations

Vectors are marked in **bold text** – so for example “**v**” is a vector and “v” is a scalar (if you can’t distinguish the two there’s problem with the text formatting wherever you’re reading this).

Part 1. The DCM Matrix

Generally speaking orientation kinematics deals with calculating the relative orientation of a body relative to a global coordinate system. It is useful to attach a coordinate system to our body frame and call it Oxyz, and another one to our global frame and call it OXYZ. Both the global and the body frames have the same fixed origin O (see *Fig. 1*). Let’s also define **i**, **j**, **k** to be unity vectors co-directional with the body frame’s x, y, and z axes – in other words they are versors of Oxyz and let **I**, **J**, **K** be the versors of global frame OXYZ.

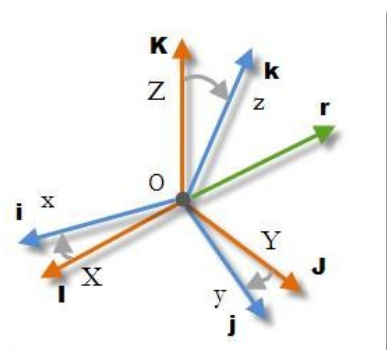


Figure 1

Thus, by definition, expressed in **terms of global coordinates** vectors **I, J, K** can be written as:

$$\mathbf{I}^G = \{1, 0, 0\}^T, \mathbf{J}^G = \{0, 1, 0\}^T, \mathbf{K}^G = \{0, 0, 1\}^T$$

Note: we use $\{\dots\}^T$ notation to denote a column vector, in other words a column vector is a translated row vector. The orientation of vectors (row/column) will become relevant once we start multiplying them by a matrix later on in this text.

And similarly, in **terms of body coordinates** vectors **i, j, k** can be written as:

$$\mathbf{i}^B = \{1, 0, 0\}^T, \mathbf{j}^B = \{0, 1, 0\}^T, \mathbf{k}^B = \{0, 0, 1\}^T$$

Now let's see if we can write vectors **i, j, k** in terms of global coordinates. Let's take vector **i** as an example and write its global coordinates:

$$\mathbf{i}^G = \{i_x^G, i_y^G, i_z^G\}^T$$

Again, by example let's analyze the X coordinate i_x^G , it's calculated as the length of projection of the **i** vector onto the global X axis.

$$i_x^G = |\mathbf{i}| \cos(X, \mathbf{i}) = \cos(\mathbf{I}, \mathbf{i})$$

Where $|\mathbf{i}|$ is the norm (length) of the **i** unity vector and $\cos(\mathbf{I}, \mathbf{i})$ is the cosine of the angle formed by the vectors **I** and **i**. Using the fact that $|\mathbf{I}| = 1$ and $|\mathbf{i}| = 1$ (they are unit vectors by definition). We can write:

$$i_x^G = \cos(\mathbf{I}, \mathbf{i}) = |\mathbf{I}||\mathbf{i}| \cos(\mathbf{I}, \mathbf{i}) = \mathbf{I} \cdot \mathbf{i}$$

Where $\mathbf{I} \cdot \mathbf{i}$ is the scalar (dot) product of vectors **I** and **i**. For the purpose of calculating scalar product $\mathbf{I} \cdot \mathbf{i}$ it doesn't matter in which coordinate system these vectors are measured as long as they are both expressed in the same system, since a rotation does not modify the angle between vectors so: $\mathbf{I} \cdot \mathbf{i} = \mathbf{I}^B \cdot \mathbf{i}^B = \mathbf{I}^G \cdot \mathbf{i}^G = \cos(\mathbf{I}^B, \mathbf{i}^B) = \cos(\mathbf{I}^G, \mathbf{i}^G)$, so for simplicity we'll skip the superscript in scalar products $\mathbf{I} \cdot \mathbf{i}$, $\mathbf{J} \cdot \mathbf{j}$, $\mathbf{K} \cdot \mathbf{k}$ and in cosines $\cos(\mathbf{I}, \mathbf{i})$, $\cos(\mathbf{J}, \mathbf{j})$, $\cos(\mathbf{K}, \mathbf{k})$.

Similarly we can show that:

$$i_y^G = \mathbf{J} \cdot \mathbf{i}, i_z^G = \mathbf{K} \cdot \mathbf{i}, \text{ so now we can write vector } \mathbf{i} \text{ in terms of global coordinate system as:}$$

$$\mathbf{i}^G = \{\mathbf{I} \cdot \mathbf{i}, \mathbf{J} \cdot \mathbf{i}, \mathbf{K} \cdot \mathbf{i}\}^T$$

$$\text{Furthermore, similarly it can be shown that } \mathbf{j}^G = \{\mathbf{I} \cdot \mathbf{j}, \mathbf{J} \cdot \mathbf{j}, \mathbf{K} \cdot \mathbf{j}\}^T, \mathbf{k}^G = \{\mathbf{I} \cdot \mathbf{k}, \mathbf{J} \cdot \mathbf{k}, \mathbf{K} \cdot \mathbf{k}\}^T.$$

We now have a complete set of global coordinates for our body's vectors **i, j, k** and we can organize these values in a convenient matrix form:

$$[\mathbf{i}^G, \mathbf{j}^G, \mathbf{k}^G] = \begin{bmatrix} \mathbf{I} \cdot \mathbf{i} & \mathbf{I} \cdot \mathbf{j} & \mathbf{I} \cdot \mathbf{k} \\ \mathbf{J} \cdot \mathbf{i} & \mathbf{J} \cdot \mathbf{j} & \mathbf{J} \cdot \mathbf{k} \\ \mathbf{K} \cdot \mathbf{i} & \mathbf{K} \cdot \mathbf{j} & \mathbf{K} \cdot \mathbf{k} \end{bmatrix} = \begin{bmatrix} \cos(\mathbf{I}, \mathbf{i}) & \cos(\mathbf{I}, \mathbf{j}) & \cos(\mathbf{I}, \mathbf{k}) \\ \cos(\mathbf{J}, \mathbf{i}) & \cos(\mathbf{J}, \mathbf{j}) & \cos(\mathbf{J}, \mathbf{k}) \\ \cos(\mathbf{K}, \mathbf{i}) & \cos(\mathbf{K}, \mathbf{j}) & \cos(\mathbf{K}, \mathbf{k}) \end{bmatrix} = \text{DCM}^G \quad (\text{Eq. 1.1})$$

This matrix is called Direction Cosine Matrix for now obvious reasons – it consists of cosines of angles of all possible combinations of body and global vectors.

The task of expressing the global frame vectors $\mathbf{I}^G, \mathbf{J}^G, \mathbf{K}^G$ in body frame coordinates is symmetrical in nature and can be achieved by simply swapping the notations **I, J, K** with **i, j, k**, the results being:

$$\mathbf{I}^B = \{\mathbf{I} \cdot \mathbf{i}, \mathbf{I} \cdot \mathbf{j}, \mathbf{I} \cdot \mathbf{k}\}^T, \mathbf{J}^B = \{\mathbf{J} \cdot \mathbf{i}, \mathbf{J} \cdot \mathbf{j}, \mathbf{J} \cdot \mathbf{k}\}^T, \mathbf{K}^B = \{\mathbf{K} \cdot \mathbf{i}, \mathbf{K} \cdot \mathbf{j}, \mathbf{K} \cdot \mathbf{k}\}^T$$

and organized in a matrix form:

$$[\mathbf{I}^B, \mathbf{J}^B, \mathbf{K}^B] = \begin{bmatrix} \mathbf{I} \cdot \mathbf{i} & \mathbf{J} \cdot \mathbf{i} & \mathbf{K} \cdot \mathbf{i} \\ \mathbf{I} \cdot \mathbf{j} & \mathbf{J} \cdot \mathbf{j} & \mathbf{K} \cdot \mathbf{j} \\ \mathbf{I} \cdot \mathbf{k} & \mathbf{J} \cdot \mathbf{k} & \mathbf{K} \cdot \mathbf{k} \end{bmatrix} = \begin{bmatrix} \cos(\mathbf{I}, \mathbf{i}) & \cos(\mathbf{J}, \mathbf{i}) & \cos(\mathbf{K}, \mathbf{i}) \\ \cos(\mathbf{I}, \mathbf{j}) & \cos(\mathbf{J}, \mathbf{j}) & \cos(\mathbf{K}, \mathbf{j}) \\ \cos(\mathbf{I}, \mathbf{k}) & \cos(\mathbf{J}, \mathbf{k}) & \cos(\mathbf{K}, \mathbf{k}) \end{bmatrix} = \text{DCM}^B \quad (\text{Eq. 1.2})$$

It is now easy to notice that $\text{DCM}^B = (\text{DCM}^G)^T$ or $\text{DCM}^G = (\text{DCM}^B)^T$, in other words the two matrices are transposes of each other, we'll use this important property later on.

Also notice that $\text{DCM}^B \cdot \text{DCM}^G = (\text{DCM}^G)^T \cdot \text{DCM}^G = \text{DCM}^B \cdot (\text{DCM}^B)^T = \mathbf{I}_3$, where \mathbf{I}_3 is the 3×3 identity matrix. In other words the DCM matrices are orthogonal.

This can be proven by simply expanding the matrix multiplication in block matrix form:

$$\begin{aligned} \text{DCM}^B \text{DCM}^G &= \text{DCM}^{GT} \text{DCM}^G = \begin{bmatrix} \mathbf{i}^{GT} \\ \mathbf{j}^{GT} \\ \mathbf{k}^{GT} \end{bmatrix} [\mathbf{i}^G, \mathbf{j}^G, \mathbf{k}^G] = \\ &= \begin{bmatrix} \mathbf{i}^{GT} \cdot \mathbf{i}^G & \mathbf{i}^{GT} \cdot \mathbf{j}^G & \mathbf{i}^{GT} \cdot \mathbf{k}^G \\ \mathbf{j}^{GT} \cdot \mathbf{i}^G & \mathbf{j}^{GT} \cdot \mathbf{j}^G & \mathbf{j}^{GT} \cdot \mathbf{k}^G \\ \mathbf{k}^{GT} \cdot \mathbf{i}^G & \mathbf{k}^{GT} \cdot \mathbf{j}^G & \mathbf{k}^{GT} \cdot \mathbf{k}^G \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \mathbf{I}_3 \end{aligned} \quad (\text{Eq. 1.3})$$

To prove this we use such properties as for example: $\mathbf{i}^{GT} \cdot \mathbf{i}^G = |\mathbf{i}^G| |\mathbf{i}^G| \cos(0) = 1$ and $\mathbf{i}^{GT} \cdot \mathbf{j}^G = 0$ because (**i** and **j** are orthogonal) and so forth.

The DCM matrix (also often called the rotation matrix) has a great importance in orientation kinematics since it defines the rotation of one frame relative to another. It can also be used to determine the global coordinates of an arbitrary vector if we know its coordinates in the body frame (and vice versa).

Let's consider such a vector with body coordinates:

$\mathbf{r}^B = \{ r_x^B, r_y^B, r_z^B \}^T$ and let's try to determine its coordinates in the global frame, by using a known rotation matrix DCM^G .

We start by doing following notation:

$\mathbf{r}^G = \{ r_x^G, r_y^G, r_z^G \}^T$.

Now let's tackle the first coordinate r_x^G :

$r_x^G = |\mathbf{r}^G| \cos(\mathbf{I}^G, \mathbf{r}^G)$, because r_x^G is the projection of \mathbf{r}^G onto X axis that is co-directional with \mathbf{I}^G .

Next let's note that by definition a rotation is such a transformation that does not change the scale of a vector and does not change the angle between two vectors that are subject to the same rotation, so if we express some vectors in a different rotated coordinate system the norm and angle between vectors will not change:

$|\mathbf{r}^G| = |\mathbf{r}^B|$, $|\mathbf{I}^G| = |\mathbf{I}^B| = 1$ and $\cos(\mathbf{I}^G, \mathbf{r}^G) = \cos(\mathbf{I}^B, \mathbf{r}^B)$, so we can use this property to write

$r_x^G = |\mathbf{r}^G| \cos(\mathbf{I}^G, \mathbf{r}^G) = |\mathbf{I}^B| |\mathbf{r}^B| \cos(\mathbf{I}^B, \mathbf{r}^B) = \mathbf{I}^B \cdot \mathbf{r}^B = \mathbf{I}^B \cdot \{ r_x^B, r_y^B, r_z^B \}^T$, by using one the two definition of the scalar product.

Now recall that $\mathbf{I}^B = \{ \mathbf{I.i}, \mathbf{I.j}, \mathbf{I.k} \}^T$ and by using the other definition of scalar product:

$r_x^G = \mathbf{I}^B \cdot \mathbf{r}^B = \{ \mathbf{I.i}, \mathbf{I.j}, \mathbf{I.k} \}^T \cdot \{ r_x^B, r_y^B, r_z^B \}^T = r_x^B \mathbf{I.i} + r_y^B \mathbf{I.j} + r_z^B \mathbf{I.k}$

In same fashion it can be shown that:

$r_y^G = r_x^B \mathbf{J.i} + r_y^B \mathbf{J.j} + r_z^B \mathbf{J.k}$

$r_z^G = r_x^B \mathbf{K.i} + r_y^B \mathbf{K.j} + r_z^B \mathbf{K.k}$

Finally let's write this in a more compact matrix form:

$$\mathbf{r}^G = \begin{bmatrix} r_x^G \\ r_y^G \\ r_z^G \end{bmatrix} = \begin{bmatrix} \mathbf{I.i} & \mathbf{I.j} & \mathbf{I.k} \\ \mathbf{J.i} & \mathbf{J.j} & \mathbf{J.k} \\ \mathbf{K.i} & \mathbf{K.j} & \mathbf{K.k} \end{bmatrix} \begin{bmatrix} r_x^B \\ r_y^B \\ r_z^B \end{bmatrix} = \text{DCM}^G \mathbf{r}^B \quad (\text{Eq. 1.4})$$

Thus the DCM matrix can be used to convert an arbitrary vector \mathbf{r}^B expressed in one coordinate system B, to a rotated coordinate system G.

We can use similar logic to prove the reverse process:

$$\mathbf{r}^B = \text{DCM}^B \mathbf{r}^G \quad (\text{Eq. 1.5})$$

Or we can arrive at the same conclusion by multiplying both parts in (Eq. 1.4) by DCM^B which equals to DCM^{GT} , and using the property that $\text{DCM}^{GT} \cdot \text{DCM}^G = \mathbf{I}_3$, see (Eq. 1.3):

$$\text{DCM}^B \mathbf{r}^G = \text{DCM}^B \text{DCM}^G \mathbf{r}^B = \text{DCM}^{GT} \text{DCM}^G \mathbf{r}^B = \mathbf{I}_3 \mathbf{r}^B = \mathbf{r}^B$$

Part 2. Angular Velocity

So far we have a way to characterize the orientation of one frame relative to another rotated frame, it is the DCM matrix and it allows us to easily convert the global and body coordinates back and forth using (Eq. 1.4) and (Eq. 1.5). In this section we'll analyze the rotation as a function of time that will help us establish the rules of updating the DCM matrix based on a characteristic called angular velocity. Let's consider an arbitrary rotating vector \mathbf{r} and define its coordinates at time t to be $\mathbf{r}(t)$. Now let's consider a small time interval dt and make the following notations: $\mathbf{r} = \mathbf{r}(t)$, $\mathbf{r}' = \mathbf{r}(t+dt)$ and $d\mathbf{r} = \mathbf{r}' - \mathbf{r}$:

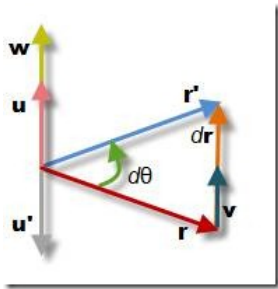


Figure 2

Let's say that during a very small time interval $dt \rightarrow 0$ the vector \mathbf{r} has rotated about an axis co-directional with a unit vector \mathbf{u} by an angle $d\theta$ and ended up in the position \mathbf{r}' . Since \mathbf{u} is our axis of rotation it is perpendicular to the plane in which the rotation took place (the plane formed by \mathbf{r} and \mathbf{r}') so \mathbf{u} is orthogonal to both \mathbf{r} and \mathbf{r}' . There are two unit vectors that are orthogonal to the plane formed by \mathbf{r} and \mathbf{r}' , they are shown on the picture as \mathbf{u} and \mathbf{u}' since we're still defining things we'll choose the one that is co-directional with the cross product $\mathbf{r} \times \mathbf{r}'$, following the rule of [right-handed coordinate system](#). Thus because \mathbf{u} is a unit vector $|\mathbf{u}| = 1$ and is co-directional with $\mathbf{r} \times \mathbf{r}'$ we can deduct it as follows:

$$\mathbf{u} = (\mathbf{r} \times \mathbf{r}') / |\mathbf{r} \times \mathbf{r}'| = (\mathbf{r} \times \mathbf{r}') / (|\mathbf{r}| |\mathbf{r}'| \sin(d\theta)) = (\mathbf{r} \times \mathbf{r}') / (|\mathbf{r}|^2 \sin(d\theta)) \quad (Eq. 2.1)$$

Since a rotation does not alter the length of a vector we used the property that $|\mathbf{r}'| = |\mathbf{r}|$.

The linear velocity of the vector \mathbf{r} can be defined as the vector:

$$\mathbf{v} = d\mathbf{r} / dt = (\mathbf{r}' - \mathbf{r}) / dt \quad (Eq. 2.2)$$

Please note that since our dt approaches 0 so does $d\theta \rightarrow 0$, hence the angle between vectors \mathbf{r} and $d\mathbf{r}$ (let's call it α) can be found from the isosceles triangle contoured by \mathbf{r} , \mathbf{r}' and $d\mathbf{r}$:

$$\alpha = (\pi - d\theta) / 2 \text{ and because } d\theta \rightarrow 0, \text{ then } \alpha \rightarrow \pi/2$$

What this tells us is that \mathbf{r} is perpendicular to $d\mathbf{r}$ when $dt \rightarrow 0$ and hence \mathbf{r} is perpendicular to \mathbf{v} since \mathbf{v} and $d\mathbf{r}$ are co-directional from (Eq. 2.2):

$$\mathbf{v} \perp \mathbf{r} \quad (Eq. 2.21)$$

We are now ready to define the angular velocity vector. Ideally such a vector should define the rate of change of the angle θ and the axis of the rotation, so we define it as follows:

$$\mathbf{w} = (d\theta/dt) \mathbf{u} \quad (Eq. 2.3)$$

Indeed the norm of the \mathbf{w} is $|\mathbf{w}| = d\theta/dt$ and the direction of \mathbf{w} coincides with the axis of rotation \mathbf{u} . Let's expand (Eq. 2.3) and try to establish a relationship with the linear velocity \mathbf{v} :

Using (Eq. 2.3) and (Eq. 2.1):

$$\mathbf{w} = (d\theta/dt) \mathbf{u} = (d\theta/dt) (\mathbf{r} \times \mathbf{r}') / (|\mathbf{r}|^2 \sin(d\theta))$$

Now note that when $dt \rightarrow 0$, so does $d\theta \rightarrow 0$ and hence for small $d\theta$, $\sin(d\theta) \approx d\theta$, we end up with:

$$\mathbf{w} = (\mathbf{r} \times \mathbf{r}') / (|\mathbf{r}|^2 dt) \quad (Eq. 2.4)$$

Now because $\mathbf{r}' = \mathbf{r} + d\mathbf{r}$, $d\mathbf{r}/dt = \mathbf{v}$, $\mathbf{r} \times \mathbf{r} = \mathbf{0}$ and using the distributive property of cross product over addition:

$$\mathbf{w} = (\mathbf{r} \times (\mathbf{r} + d\mathbf{r})) / (|\mathbf{r}|^2 dt) = (\mathbf{r} \times \mathbf{r} + \mathbf{r} \times d\mathbf{r}) / (|\mathbf{r}|^2 dt) = \mathbf{r} \times (d\mathbf{r}/dt) / |\mathbf{r}|^2$$

And finally:

$$\mathbf{w} = \mathbf{r} \times \mathbf{v} / |\mathbf{r}|^2 \quad (Eq. 2.5)$$

This equation establishes a way to calculate angular velocity from a known linear velocity \mathbf{v} .

We can easily prove the reverse equation that lets us deduct linear velocity from angular velocity:

$$\mathbf{v} = \mathbf{w} \times \mathbf{r} \quad (Eq. 2.6)$$

This can be proven simply by expanding \mathbf{w} from (Eq. 2.5) and using vector [triple product](#) rule $(\mathbf{a} \times \mathbf{b}) \times \mathbf{c} = (\mathbf{a} \cdot \mathbf{c})\mathbf{b} - (\mathbf{b} \cdot \mathbf{c})\mathbf{a}$. Also we'll use the fact that \mathbf{v} and \mathbf{r} are perpendicular (Eq. 2.21) and thus $\mathbf{v} \cdot \mathbf{r} = 0$

$$\mathbf{w} \times \mathbf{r} = (\mathbf{r} \times \mathbf{v} / |\mathbf{r}|^2) \times \mathbf{r} = (\mathbf{r} \times \mathbf{v}) \times \mathbf{r} / |\mathbf{r}|^2 = ((\mathbf{r} \cdot \mathbf{r})\mathbf{v} + (\mathbf{v} \cdot \mathbf{r})\mathbf{r}) / |\mathbf{r}|^2 = (|\mathbf{r}|^2 \mathbf{v} + 0) / |\mathbf{r}|^2 = \mathbf{v}$$

So we just proved that (Eq. 2.6) is true. Just to check (Eq. 2.6) intuitively – from Figure 2 indeed \mathbf{v} has the direction of $\mathbf{w} \times \mathbf{r}$ using the right hand rule and indeed $\mathbf{v} \perp \mathbf{r}$ and $\mathbf{v} \perp \mathbf{w}$ because it is in the same plane with \mathbf{r} and \mathbf{r}' .

Part 3. Gyroscopes and angular velocity vector

A 3-axis MEMS gyroscope is a device that senses rotation about 3 axes attached to the device itself (body frame). If we adopt the device's coordinate system (body's frame), and analyze some vectors attached to the earth (global frame), for example vector \mathbf{K} pointing to the zenith or vector \mathbf{I} pointing North – then it would appear to an observer inside the device that these vector rotate about the device center. Let w_x , w_y , w_z be the outputs of a gyroscope expressed in rad/s – the measured rotation about axes x , y , z respectively. Converting from the raw output of the gyroscope to physical values is discussed for example here: http://www.starlino.com/imu_guide.html. If we query the gyroscope at regular, small time intervals dt , then what gyroscope output tells us is that during this time interval the earth rotated about gyroscope's x axis by an angle of $d\theta_x = w_x dt$, about y axis by an angle of $d\theta_y = w_y dt$ and about z axis by an angle of $d\theta_z = w_z dt$. These rotations can be characterized by the angular velocity vectors: $\mathbf{w}_x = w_x \mathbf{i} = \{w_x, 0, 0\}^T$, $\mathbf{w}_y = w_y \mathbf{j} = \{0, w_y, 0\}^T$, $\mathbf{w}_z = w_z \mathbf{k} = \{0, 0, w_z\}^T$, where $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are versors of the local coordinate frame (they are co-directional with body's axes x, y, z respectively). Each of these three rotations will cause a linear displacement which can be expressed by using (Eq. 2.6):

$$d\mathbf{r}_1 = dt \mathbf{v}_1 = dt (\mathbf{w}_x \times \mathbf{r}); d\mathbf{r}_2 = dt \mathbf{v}_2 = dt (\mathbf{w}_y \times \mathbf{r}); d\mathbf{r}_3 = dt \mathbf{v}_3 = dt (\mathbf{w}_z \times \mathbf{r}).$$

The combined effect of these three displacements will be:

$$d\mathbf{r} = d\mathbf{r}_1 + d\mathbf{r}_2 + d\mathbf{r}_3 = dt (\mathbf{w}_x \times \mathbf{r} + \mathbf{w}_y \times \mathbf{r} + \mathbf{w}_z \times \mathbf{r}) = dt (\mathbf{w}_x + \mathbf{w}_y + \mathbf{w}_z) \times \mathbf{r} \text{ (cross product is distributive over addition)}$$

Thus the equivalent linear velocity resulting from these 3 transformations can be expressed as:

$$\mathbf{v} = d\mathbf{r}/dt = (\mathbf{w}_x + \mathbf{w}_y + \mathbf{w}_z) \times \mathbf{r} = \mathbf{w} \times \mathbf{r}, \text{ where we introduce } \mathbf{w} = \mathbf{w}_x + \mathbf{w}_y + \mathbf{w}_z = \{w_x, w_y, w_z\}$$

Which looks exactly like (Eq. 2.6) and suggests that the combination of three **small** rotations about axes x, y, z characterized by angular rotation vectors \mathbf{w}_x , \mathbf{w}_y , \mathbf{w}_z is equivalent to one **small** rotation characterized by angular rotation vector $\mathbf{w} = \mathbf{w}_x + \mathbf{w}_y + \mathbf{w}_z = \{w_x, w_y, w_z\}$. Please note that we're stressing out that these are **small** rotations, since in general when you combine large rotations the order in which rotations are performed become important and you cannot

simply sum them up. Our main assumption that let us go from a linear displacement to a rotation by using (Eq. 2.6) was that dt is really small, and thus the rotations $d\theta$ and linear displacement dr are small as well. In practice this means that the larger the dt interval between gyro queries the larger will be our accumulated error, we'll deal with this error later on. Now, since w_x, w_y, w_z are the output of the gyroscope, then we arrive at the conclusion that in fact a 3 axis gyroscope measures the instantaneous angular velocity of the world rotating about the device's center.

Part 4. DCM complimentary filter algorithm using 6DOF or 9DOF IMU sensors

In the context of this text a 6DOF device is an IMU device consisting of a 3 axis gyroscope and a 3 axis accelerometer. A 9DOF device is an IMU device of a 3 axis gyroscope, a 3 axis accelerometer and a 3 axis magnetometer. Let's attach a global right-handed coordinate system to the Earth's frame such that the **I** versor points North, **K** versor points to the Zenith and thus, with these two versors fixed, the **J** versor will be constrained to point West.

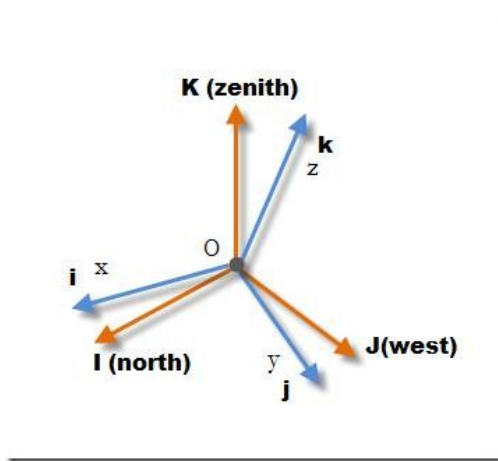


Figure 3

Also let's consider the body coordinate system to be attached to our IMU device (acc_gyro used as an example),

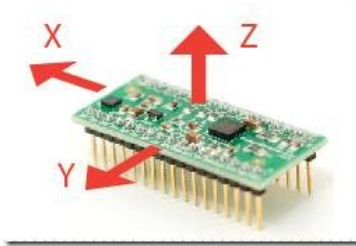


Figure 4

We already established the fact that gyroscopes can measure the angular velocity vector. Let's see how accelerometer and magnetometer measurements will fall into our model.

Accelerometers are devices that can sense gravitation. Gravitation vector is pointing towards the center of the earth and is opposite to the vector pointing to Zenith \mathbf{K}^B . If the 3 axis accelerometer output is $\mathbf{A} = \{A_x, A_y, A_z\}$ and we assume that there are no external accelerations or we have corrected them then we can estimate that $\mathbf{K}^B = -\mathbf{A}$. (See this IMU Guide for more clarifications http://www.starlino.com/imu_guide.html).

Magnetometers are devices that are really similar to accelerometers, except that instead of gravitation they can sense the Earth's magnetic North. Just like accelerometers they are not perfect and often need corrections and initial calibration. If the corrected 3-axis magnetometer output is $\mathbf{M} = \{M_x, M_y, M_z\}$, then according to our model \mathbf{I}^B is pointing North, thus $\mathbf{I}^B = \mathbf{M}$.

Knowing \mathbf{I}^B and \mathbf{K}^B allows us calculate $\mathbf{J}^B = \mathbf{K}^B \times \mathbf{I}^B$.

Thus an accelerometer and a magnetometer alone can give us the DCM matrix, expressed either as DCM^B or DCM^G

$$\text{DCM}^G = \text{DCM}^{BT} = [\mathbf{I}^B, \mathbf{J}^B, \mathbf{K}^B]^T$$

The DCM matrix can be used to convert any vector from body's(devices) coordinate system to the global coordinate system. Thus for example if we know that the nose of the plane has some fixed coordinates expressed in body's coordinate system as $\mathbf{r}^B = \{1, 0, 0\}$, then we can find where the device is heading in other words the coordinates of the nose in global coordinate systems using (Eq. 1.4):

$$\mathbf{r}^G = \text{DCM}^G \mathbf{r}^B$$

So far you're asking yourself if an accelerometer and a magnetometer gives us the DCM matrix at any point in time, why do we need the gyroscope? The gyroscope is actually a more precise device than the accelerometer and magnetometer are, it is used to "fine-tune" the DCM matrix returned by the accelerometer and magnetometer.

Gyroscopes have no sense of absolute orientation of the device, i.e. they don't know where north is and where zenith is (things that we can find out using the accelerometer and magnetometer), instead if we know the orientation of the device at time t , expressed as a DCM matrix $\text{DCM}(t)$, we can find a more precise orientation $\text{DCM}(t+dt)$ using the gyroscope, then the one estimated directly from the accelerometer and magnetometer direct readings which are subject to a lot of noise in form of external (non-gravitational) inertial forces (i.e. acceleration) or magnetically forces that are not caused by the earth's magnetic field.

These facts call for an algorithm that would combine the readings from all three devices (accelerometer, magnetometer and gyroscope) in order to create our best guess or estimate regarding the device orientation in space (or space's orientation in device's coordinate systems), the two orientations are related since

they are simply expressed using two DCM matrices that are transpose of one another ($\text{DCM}^G = \text{DCM}^{BT}$).

We'll now go ahead and introduce such an algorithm.

We'll work with the DCM matrix that consists of the versors of the global (earth's) coordinate system aligned on each row:

$$\text{DCM}^G = [\mathbf{i}^G, \mathbf{j}^G, \mathbf{k}^G] = \begin{bmatrix} \mathbf{I.i} & \mathbf{I.j} & \mathbf{I.k} \\ \mathbf{J.i} & \mathbf{J.j} & \mathbf{J.k} \\ \mathbf{K.i} & \mathbf{K.j} & \mathbf{K.k} \end{bmatrix} = \begin{bmatrix} \cos(\mathbf{I,i}) & \cos(\mathbf{I,j}) & \cos(\mathbf{I,k}) \\ \cos(\mathbf{J,i}) & \cos(\mathbf{J,j}) & \cos(\mathbf{J,k}) \\ \cos(\mathbf{K,i}) & \cos(\mathbf{K,j}) & \cos(\mathbf{K,k}) \end{bmatrix} = \begin{bmatrix} \mathbf{I}^{BT} \\ \mathbf{J}^{BT} \\ \mathbf{K}^{BT} \end{bmatrix}$$

If we read the rows of DCM^G we get the vectors $\mathbf{I}^B, \mathbf{J}^B, \mathbf{K}^B$. We'll work mostly with vectors \mathbf{K}^B (that can be directly estimated by accelerometer) and vector \mathbf{I}^B (that can be directly estimated by the magnetometer). The vector \mathbf{J}^B is simply calculated as $\mathbf{J}^B = \mathbf{K}^B \times \mathbf{I}^B$, since it's orthogonal to the other two vectors (remember versors are unity vectors with same direction as coordinate axes).

Let's say we know the zenith vector expressed in body frame coordinates at time t_0 and we note it as \mathbf{K}^B_0 . Also let's say we measured our gyro output and we have determined that our angular velocity is $\mathbf{w} = \{w_x, w_y, w_z\}$. Using our gyro we want to know the position of our zenith vector after a small period of time dt has passed we'll note it as \mathbf{K}^B_{1G} . And we find it using (Eq. 2.6):

$$\mathbf{K}^B_{1G} \approx \mathbf{K}^B_0 + dt \mathbf{v} = \mathbf{K}^B_0 + dt (\mathbf{w}_g \times \mathbf{K}^B_0) = \mathbf{K}^B_0 + (d\theta_g \times \mathbf{K}^B_0)$$

Where we noted $d\theta_g = dt \mathbf{w}_g$. Because \mathbf{w}_g is angular velocity as measured by the gyroscope. We'll call $d\theta_g$ angular displacement. In other words it tells us by what **small** angle (given for all 3 axis in form of a vector) has the orientation of a vector \mathbf{K}^B changed during this **small** period of time dt .

Obviously, another way to estimate \mathbf{K}^B is by making another reading from accelerometer so we can get a reading that we note as \mathbf{K}^B_{1A} .

In practice the values \mathbf{K}^B_{1G} will be different from \mathbf{K}^B_{1A} . One was estimated using our gyroscope and the other was estimated using our accelerometer.

Now it turns out we can go the reverse way and estimate the angular velocity \mathbf{w}_a or angular displacement $d\theta_a = dt \mathbf{w}_a$, from the new accelerometer reading \mathbf{K}^B_{1A} , we'll use (Eq. 2.5):

$$\mathbf{w}_a = \mathbf{K}^B_0 \times \mathbf{v}_a / |\mathbf{K}^B_0|^2$$

Now $\mathbf{v}_a = (\mathbf{K}^B_{1A} - \mathbf{K}^B_0) / dt$, and is basically the linear velocity of the vector \mathbf{K}^B_0 . And $|\mathbf{K}^B_0|^2 = 1$, since \mathbf{K}^B_0 is a unity vector. So we can calculate:

$$d\theta_a = dt \mathbf{w}_a = \mathbf{K}^B_0 \times (\mathbf{K}^B_{1A} - \mathbf{K}^B_0)$$

The idea of calculating a new estimate \mathbf{K}^B_1 that combines both \mathbf{K}^B_{1A} and \mathbf{K}^B_{1G} is to first estimate $d\theta$ as a weighted average of $d\theta_a$ and $d\theta_g$:

$d\theta = (s_a d\theta_a + s_g d\theta_g) / (s_a + s_g)$, we'll discuss about the weights later on, but shortly they are determined and tuned experimentally in order to achieve a desired response rate and noise rejection.

Note for advanced readers: To improve algorithm we could divide $d\theta_g$ into two components: one orthogonal to \mathbf{K}^B_0 and one parallel to it. We'd then only apply averaging formula on the component orthogonal to \mathbf{K}^B_0 , the idea being that the accelerometer cannot sense rotation about \mathbf{K}^B_0 (azimuth to zenith) axis, thus the equivalent parallel component of $d\theta_a$ will always be zero.

And then \mathbf{K}^B_1 is calculated similar to how we calculated \mathbf{K}^B_{1G} :

$$\mathbf{K}^B_1 \approx \mathbf{K}^B_0 + (d\theta \times \mathbf{K}^B_0)$$

Why we went all the way to calculate $d\theta$ and did not apply the weighted average formula directly to \mathbf{K}^B_{1A} and \mathbf{K}^B_{1G} ? Because $d\theta$ can be used to calculate the other elements of our DCM matrix in the same way:

$$\mathbf{I}^B_1 \approx \mathbf{I}^B_0 + (d\theta \times \mathbf{I}^B_0)$$

$$\mathbf{J}^B_1 \approx \mathbf{J}^B_0 + (d\theta \times \mathbf{J}^B_0)$$

The idea is that all three versors $\mathbf{I}^B, \mathbf{J}^B, \mathbf{K}^B$ are attached to each other and will follow the same angular displacement $d\theta$ during our small interval dt . So in a nutshell this is the algorithm that allows us to calculate the DCM_1 matrix at time t_1 from our previous estimated DCM_0 matrix at time t_0 . It is applied recursively at regular small time intervals dt and gives us an updated DCM matrix at any point in time. The matrix will not drift too much because it is fixed to the absolute position dictated by the accelerometer and will not be too noisy from external accelerations because we also use the gyroscope data to update it.

So far we didn't mention a word about our magnetometer. One reason being that it is not available on all IMU units (6DOF) and we can go away without using it, but our resulting orientation will then have a drifting heading (i.e. it will not show if we're heading north, south, west or east), or we can introduce a virtual magnetometer that is always pointing North, to introduce stability in our model. This situation is demonstrated in the accompanying source code that used a 6DOF IMU.

Now we'll show how to integrate magnetometer readings into our algorithm. As it turns out it is really simple since magnetometer is really similar to accelerometer (they even use similar calibration algorithms), the only difference being that instead of estimating the Zenith vector \mathbf{K}^B vector it estimates the

vector pointing North \mathbf{I}^B . Following the same logic as we did for our accelerometer we can determine the angular displacement according to the updated magnetometer reading as being:

$$d\mathbf{\theta}_m = dt \mathbf{w}_m = \mathbf{I}_{00}^B \times (\mathbf{I}_{1M}^B - \mathbf{I}_{00}^B)$$

Now let's incorporate it into our weighted average:

$$d\mathbf{\theta} = (s_a d\mathbf{\theta}_a + s_g d\mathbf{\theta}_g + s_m d\mathbf{\theta}_m) / (s_a + s_g + s_m)$$

From here we go the same path to calculate the updated DCM₁

$$\mathbf{I}_{11}^B \approx \mathbf{I}_{00}^B + (d\mathbf{\theta} \times \mathbf{I}_{00}^B), \mathbf{K}_{11}^B \approx \mathbf{K}_{00}^B + (d\mathbf{\theta} \times \mathbf{K}_{00}^B) \text{ and } \mathbf{J}_{11}^B \approx \mathbf{J}_{00}^B + (d\mathbf{\theta} \times \mathbf{J}_{00}^B),$$

In practice we'll calculate $\mathbf{J}_{11}^B = \mathbf{K}_{11}^B \times \mathbf{I}_{11}^B$, after correcting \mathbf{K}_{11}^B and \mathbf{I}_{11}^B to be perpendicular unity vectors again, note that all our logic is approximated and dependent on dt being small, the larger the dt the larger the error we'll accumulate.

So if vectors $\mathbf{I}_{00}^B, \mathbf{J}_{00}^B, \mathbf{K}_{00}^B$ form a valid DCM matrix, in other words they are orthogonal to each other and are unity vectors, then we can't say the same about $\mathbf{I}_{11}^B, \mathbf{J}_{11}^B, \mathbf{K}_{11}^B$, the formulas used for calculating them does not guarantee the orthogonality or length of the vector to be preserved, however we will not get a big error if dt is small, all we need to do is to correct them after each iteration.

First let's see how we can ensure that two vectors are orthogonal again. Let's consider two unity vectors \mathbf{a} and \mathbf{b} that are "almost orthogonal" in other words the angle between these two vectors is close to 90° , but not exactly 90° . We're looking to find a vector \mathbf{b}' that is orthogonal to \mathbf{a} and that is in the same plane formed by the vectors \mathbf{a} and \mathbf{b} . Such a vector is easy to find as shown in Figure 5. First we find vector $\mathbf{c} = \mathbf{a} \times \mathbf{b}$ that by the rules of cross product is orthogonal to both \mathbf{a} and \mathbf{b} and thus is perpendicular to the plane formed by \mathbf{a} and \mathbf{b} . Next the vector $\mathbf{b}' = \mathbf{c} \times \mathbf{a}$ is calculated as the cross product of \mathbf{c} and \mathbf{a} . From the definition of cross product \mathbf{b}' is orthogonal to \mathbf{a} and because it is also orthogonal to \mathbf{c} – it end up in the plane orthogonal to \mathbf{c} , which is the plane formed by \mathbf{a} and \mathbf{b} . Thus \mathbf{b}' is the corrected vector we're seeking that is orthogonal to \mathbf{a} and belongs to the plane formed by \mathbf{a} and \mathbf{b} .

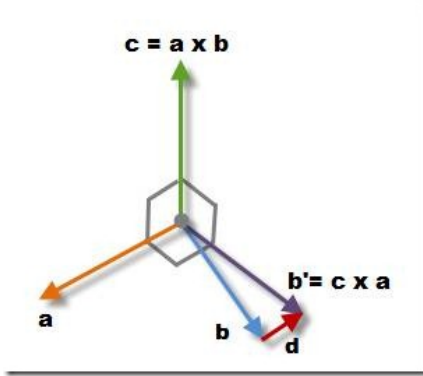


Figure 5

We can extend the equation using [the triple product rule](#) and the fact that $\mathbf{a} \cdot \mathbf{a} = |\mathbf{a}|^2 = 1$:

$$\mathbf{b}' = \mathbf{c} \times \mathbf{a} = (\mathbf{a} \times \mathbf{b}) \times \mathbf{a} = -\mathbf{a}(\mathbf{a} \cdot \mathbf{b}) + \mathbf{b}(\mathbf{a} \cdot \mathbf{a}) = \mathbf{b} - \mathbf{a}(\mathbf{a} \cdot \mathbf{b}) = \mathbf{b} + \mathbf{d}, \text{ where } \mathbf{d} = -\mathbf{a}(\mathbf{a} \cdot \mathbf{b}) \text{ (Scenario 1, } \mathbf{a} \text{ is fixed } \mathbf{b} \text{ is corrected)}$$

You can reflect a little bit on the results ... So we obtain corrected vector \mathbf{b}' from vector \mathbf{b} by adding a "correction" vector $\mathbf{d} = -\mathbf{a}(\mathbf{a} \cdot \mathbf{b})$. Notice that \mathbf{d} is parallel to \mathbf{a} . Its direction is dependent upon the angle between \mathbf{a} and \mathbf{b} , for example in Figure 5 $\mathbf{a} \cdot \mathbf{b} = \cos(\mathbf{a}, \mathbf{b}) > 0$, because angle between \mathbf{a} and \mathbf{b} is less than 90° thus \mathbf{d} has opposite direction from \mathbf{a} and a magnitude of $\cos(\mathbf{a}, \mathbf{b}) = \sin(\mathbf{b}, \mathbf{b}')$.

In the scenario above we considered that vector \mathbf{a} is fixed and we found a corrected vector \mathbf{b}' that is orthogonal to \mathbf{a} . We can consider the symmetric problem – we fix \mathbf{b} and find the corrected vector \mathbf{a}' :

$$\mathbf{a}' = \mathbf{a} - \mathbf{b}(\mathbf{a} \cdot \mathbf{b}) = \mathbf{a} - \mathbf{b}(\mathbf{a} \cdot \mathbf{b}) = \mathbf{a} + \mathbf{e}, \text{ where } \mathbf{e} = -\mathbf{b}(\mathbf{a} \cdot \mathbf{b}) \text{ (Scenario 2, } \mathbf{b} \text{ is fixed } \mathbf{a} \text{ is corrected)}$$

Finally in the third scenario we want both vectors to move towards their corrected state, we consider them both "equally wrong", so intuitively we apply half correction to both vectors from scenario 1 and 2:

$$\mathbf{a}' = \mathbf{a} - \mathbf{b}(\mathbf{a} \cdot \mathbf{b}) / 2 \text{ (Scenario 3, both } \mathbf{a} \text{ and } \mathbf{b} \text{ are corrected)}$$

$$\mathbf{b}' = \mathbf{b} - \mathbf{a}(\mathbf{a} \cdot \mathbf{b}) / 2$$

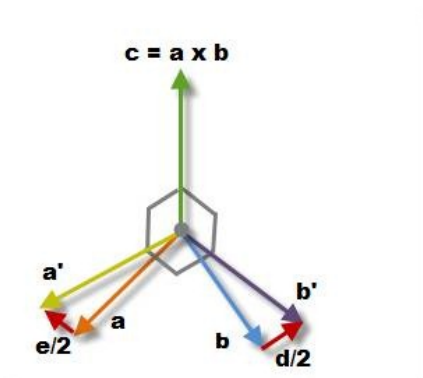


Figure 6

This is an relatively easy formula to calculate on a microprocessor since we can pre-compute $\text{Err} = (\mathbf{a} \cdot \mathbf{b})/2$ and then use it to correct both vectors:

$$\mathbf{a}' = \mathbf{a} - \text{Err} * \mathbf{b}$$

$$\mathbf{b}' = \mathbf{b} - \text{Err} * \mathbf{a}$$

Please note that we're not proving that \mathbf{a}' and \mathbf{b}' are orthogonal in Scenario 3, but we presented the intuitive reasoning why the angle between \mathbf{a}' and \mathbf{b}' will get closer to 90° if we apply the above corrective transformations.

Now going back to our updated DCM matrix that consists of three vectors $\mathbf{I}_1^B, \mathbf{J}_1^B$, we apply the following corrective actions before reintroducing the DCM matrix into the next loop:

$$\text{Err} = (\mathbf{I}_1^B \cdot \mathbf{J}_1^B) / 2$$

$$\mathbf{I}_1^{B'} = \mathbf{I}_1^B - \text{Err} * \mathbf{J}_1^B$$

$$\mathbf{J}_1^{B'} = \mathbf{J}_1^B - \text{Err} * \mathbf{I}_1^B$$

$$\mathbf{I}_1^{B''} = \text{Normalize}[\mathbf{I}_1^{B'}]$$

$$\mathbf{J}_1^{B''} = \text{Normalize}[\mathbf{J}_1^{B'}]$$

$$\mathbf{K}_1^{B''} = \mathbf{I}_1^{B''} \times \mathbf{J}_1^{B''}$$

Where $\text{Normalize}[\mathbf{a}] = \mathbf{a} / |\mathbf{a}|$, is the formula calculating the unit vector co-directional with \mathbf{a} .

So finally our corrected DCM₁ matrix can be recomposed from vectors $\mathbf{I}_1^{B''}, \mathbf{J}_1^{B''}, \mathbf{K}_1^{B''}$ that have been ortho-normalized (each vector constitutes a row of the updated and corrected DCM matrix).

We repeat the loop to find DCM₂, DCM₃, or in general DCM_n, at any time interval n.

References

1. [Theory of Applied Robotics: Kinematics, Dynamics, and Control \(Reza N. Jazar\)](#)
2. [Linear Algebra and Its Applications \(David C. Lay\)](#)
3. [Fundamentals of Matrix Computations \(David S. Watkins\)](#)
4. [Direction Cosine Matrix IMU: Theory \(W. Premerlani\)](#)

Additional Notes

For the implementation of the algorithm for now see my quadcopter project in particular releases 6/7 have a nice [Processing](#) program for visual display of the DCM matrix and a model plane. The entire code is on SVN repository:

<https://code.google.com/p/picquadcontroller/source/browse/trunk/>

The code is in imu.h file:

<https://code.google.com/p/picquadcontroller/source/browse/trunk/imu.h>

A PDF Version of this article is available here

[DCM Tutorial – An Introduction to Orientation Kinematics by Starlino \(PDF, Rev 0.1 Draft\)](#)

Please mention and link to the source when using information in this article:

http://www.starlino.com/dcm_tutorial.html

Starlino Electronics // Spring, 2011

[Comments \(140\)](#)

140 COMMENTS | [Add Comment](#) | [RSS](#)

1. Peter | June 7, 2011

Hi Starlino,

I'm trying to understand this. As per diydrones post...

The thing that I keep struggling with is why I don't see a Euler rotation equation anywhere.

When you say:

"Please note that we're stressing out that these are small rotations, since in general when you combine large rotations the order in which rotations are performed become important and you cannot simply sum them up."

I just can't work out why small rotations would make any difference. Could this hold true for a gyro only solution? Can you help me understand this?

2. starlino | June 13, 2011

Peter, really good question ...

To understand why the order of rotations becomes more important with angle magnitude and less important for small angles try to perform following experiments. Pick an object like a book and choose the local X and Y axis attached to the object, for example X axis is along bottom side and Y axis is along

left side.

Now being each experiment from a fixed start position.

1) Rotate the object roughly 5 degrees about X axis then , 5 degrees about Y axis remember end position.

2) Rotate the object roughly 5 degrees about Y axis then , 5 degrees about X axis remember end position.

Compare the end position on experiments 1) and 2) they will be roughly the same.

Now let's see what happens with larger angles, by performing:

3) Rotate the object roughly 90 degrees about X axis then , 90 degrees about Y axis remember end position.

4) Rotate the object roughly 90 degrees about Y axis then , 90 degrees about X axis remember end position.

The end positions of 3) and 4) are obviously different, so order of rotation is important for larger angles.

Now what I am trying to explain “with small angles” is actually decomposition of angular velocity vector. And what I am trying to avoid is vector derivatives in order to make it simple for all folks to grasp it. However if you prefer, a more robust proof is given for example in [Jazar, Ex. 200, 201 pg 385-387, see References section of the article] using vector analysis.

The problem arises from the fact that a 3 axis gyro tells us the angular velocity about each axis separately w_1 about X, w_2 about Y, w_3 about Z, after each interval dt we don't know in which order these rotations happened, but for small rotation we're showing that we can consider these three rotations to be (roughly) equivalent to one rotation about a vector $w = \{w_1, w_2, w_3\}$, with an angle of $|w|$. This proves to be convenient in updating the DCM matrix in one step, however theoretically one could apply 3 separate updates (again order is not important as long as we update really often and w_1, w_2, w_3 are not large).

Yes I think you could use this algorithm with gyro only, just set the weight of magnetometer and accelerometer to 0 and reduce it to see what happens. You will basically skip the step where w_G , w_A , w_M are combined by a weighted average and just use w_G to update the DCM matrix.

3. [Direct Cosine Method Article nice « Industrial Engineering](#) | August 23, 2011

[...] http://www.starlino.com/dcm_tutorial.html Share this:TwitterFacebookLike this:LikeBe the first to like this post. [...]

4. [fahdovski](#) | August 23, 2011

Thx a lot Starlino , you rock !!

5. [fahdovski](#) | August 26, 2011

hi,

Plz i need to calculate the Inertia matrix of the Quadcopter to simulate the Quad on Matlab
any help plz

6. [Tricopter: gyro damping and PID control \(fail\) « Design, Build, Test, Iterate](#) | October 27, 2011

[...] been working on stabilization code for the tricopter using direction cosine matrices but haven't yet gotten too far because the DCM updates too slowly. I must be doing something [...]

7. [fahdovski](#) | November 6, 2011

I have a question

What is the best method

DCM

or

Kalman Filter (Acc+Gyro)

8. [starlino](#) | November 6, 2011

DCM is not really an algorithm for IMU fusion, I think you mean complimentary filter applied to DCM (direction cosine matrix). Complimentary filter + DCM matrix is easier to understand and tune-up in my opinion. You can also “upgrade” your complimentary filter with auto-adjusting weights to trigger even better results.

9. [Orientation Kinematics with Direction Cosine Matrices « Design, Build, Test, Iterate](#) | November 13, 2011

[...] The Starlino DCM Tutorial [...]

10. [Frantch](#) | December 8, 2011

Hello Starlino,

Big thanks for this tutorial. I've been brought here because I'm developing an Android app that is trying to calculate best YAW (azimuth) angle possible. The Android SDK provide a method to get the OrientationMatrix but it's really buggy. This is way I wanted to give a try to the DCM. My phone is 9DOF (3axes accel + gyro + magneto)

I tried to implement the code you provide in Java. I display the rotation using a cube in openGL. But I'm running into a some problem:

1. Sometimes but not always when I start the App the cube is flicking from 0 to 180 on the yaw and pitch angle (the roll seems OK..)

if a shake a bit the phone “sometimes” it stabilies !! and seemed to work really well “visually” the cube behave exactly how I want (pitch, roll yaw)

TO debug I tried to print the Euler angle with

```
angle[YAW] = atan2(dcmMatrix[3], dcmMatrix[0]);
```

```
angle[PITCH] = -asin(dcmMatrix[6]);
```

```
angle[ROLL] = atan2(dcmMatrix[7], dcmMatrix[8]);
```

When the cube is stabilized and the phone still on the table (screen facing up) the values are

yaw: 0 pitch = 0 roll = 180 (it should be 0 no) ??

if a yaw the phone the printed yaw increase (OK) if I roll the phone.. the printed PITCH increase???? and if a pitch the phone the printed pitch seemed to be wrong :S

WHy is there that are the axes mixup ??

2. Now.. in your code the Imag vector is always pointing at the same direction because you don't have magnetometer. But since i have, I put the magnetometer value on the Imag vector
 And now.. (well the problem of the flicking is still here I have to shake the phone I don't know How and it stabilizes..) But now the cube is not reacting correctly "visually"
 When i YAW the phone the cube is rotating around XY (YAW) AND AT THE SAME TIME around XZ (PITH) ?? But when I roll the phone not problem! the cube roll

Why is this happening ?

TO test i'm check if the matrix is orthogonal ($i^2 + j^2 + k^2 = 1$) all do it on all the line and column

thanks

11. Frantch | December 8, 2011

if I print out the first line of the DCM at each iteration I get that

```
-0.99600416 0.028452467 0.08465337
-0.99596983 0.02799713 0.085207015
-0.99591947 0.027649324 0.085906744
-0.9958669 0.027320543 0.08661838
-0.9958056 0.027040107 0.087407134
0.9956389 -0.026996411 -0.089299254
0.9955801 -0.02681771 -0.09000592
0.99550337 -0.026562588 -0.09092565
0.9954371 -0.026288053 -0.09172788
0.99537075 -0.026074352 -0.09250489
```

I have a signed problem that why it is flicking... that might come from the accelerometer

12. Frantch | December 8, 2011

Sorry Starlino As you haven't yet validated my comment I just post this on to say that the problem I ha when adding the magnetometer has been fixed.. (I just forgot to normalize the vector Imag.. !)

But the problem of the flicking is still here !

13. Frantch | December 8, 2011

Humm hello again Starlino... I think I found out the flicking problem..
 The problem is the orthonormalization !
 When I do the dot product between the line I and J the error is reaaaaaly small (something like 0.00004)

```
So if a comment out this part
temporary_dot_I = DCM_I.scale(-err / 2);
DCM_J.add(temporary_dot_I);

temporary_dot_J = DCM_J.scale(-err / 2);
DCM_I.add(temporary_dot_J);
```

No flicking anymore !! And the cube seems to always be quite orthogonal (not deformed...) if I a shake the phone really hard :)

So actually we don't need to do this correction EVER TIME ? but maybe only off error > 0.5 or something ?

14. [Anonymous](#) | January 1, 2012

[...] [...]

15. [Aswin](#) | January 10, 2012

Hi Starlino,

This is a very nice article. It explains the concepts of a DCM very well.
 I wonder about one thing though. The North vector the magnetometer is pointing at is not perpendicular to the Zenith vector. It is pointing into the ground in a northerly direction. The angle with the XY plane, or surface of the earth, is the inclination of the earth magnetic field.

I myself am in the middle of incorporating the magnetometer into a 6DOF IMU so I am not really sure about how to cope with this. But I think you have to transform the compass vector to the world frame, then you can correct for the inclination to make the Vector perpendicular to the Zenith vector. The result is transformed back to the body frame. After that it can be used like you describe.

16. starlino | January 10, 2012

Aswin: Good question, let Z be the zenith UNIT vector (obtained from accelerometer) and M be the raw magnetometer reading UNIT vector that is not necessarily perpendicular to Z. To obtain true North perpendicular to Z and parallel to the ground use the following transformation.
 First obtain a vector that is perpendicular to plain formed by Z and M, that happens to be W (West):

$W = Z \times M$ (where "x" is the vector cross product), note order is important, use right-hand coordinate system rule.

Then obtain North (N) , as a vector perpendicular to bot Z and W simply get:

$N = W \times Z$

Or in one formula

$N = (Z \times M) \times Z$, using triple product formula

$\mathbf{N} = (\mathbf{Z} \cdot \mathbf{Z}) \mathbf{M} - (\mathbf{M} \cdot \mathbf{Z}) \mathbf{Z}$, where “.” is the scalar dot product, since \mathbf{Z} is a unit vector this becomes

$$\mathbf{N} = \mathbf{M} - (\mathbf{M} \cdot \mathbf{Z}) \mathbf{Z}$$

The scalar $-(\mathbf{M} \cdot \mathbf{Z}) = -\cos(\mathbf{M}, \mathbf{Z})|\mathbf{M}||\mathbf{Z}| = -\cos(\mathbf{M}, \mathbf{Z})$ (since \mathbf{M}, \mathbf{Z} unit vectors) is the correction term, you can visualize this as if it pulls the uncorrected vector \mathbf{M} away from \mathbf{Z} depending on the angle between \mathbf{M} and \mathbf{Z} .

Note that in particular if \mathbf{M} is perpendicular to \mathbf{Z} , then $\mathbf{M} \cdot \mathbf{Z} = 0$, so $\mathbf{N} = \mathbf{M}$ as expected.

Finally you may want to normalize \mathbf{N} to make sure it is a unit vector:

$$\mathbf{N}' = \text{Normalize}(\mathbf{N})$$

17. [A Guide To using IMU \(Accelerometer and Gyroscope Devices\) in Embedded Applications.](#) « Starlino Electronics | January 21, 2012

[...] – LIS331AL (datasheet) – analog 3-axis 2G accelerometer – LPR550AL (datasheet) – a dual-axis (Pitch and Roll), 500deg/second gyroscope – LY550ALH (datasheet) – a single axis (Yaw) gyroscope (this last device is not used in this tutorial but it becomes relevant when you move on to DCM Matrix implementation) [...]

18. [Alfiansyah](#) | May 17, 2012

You rocks Starlino!!

:-)

your writings help me very much . . Don't know what to say, without you, i will not be able to finish my Bachelor degree Final Project.

bless you :'-)

19. [Alfiansyah](#) | May 22, 2012

May i ask you something?

How to convert your DCMg / DCMb to euler angles? (Pitch, Yaw, Roll)
I'am really confused with that problem. . .

Thx Starlino :-)

20. starlino | May 22, 2012

Alfiansyah, You'll find DCM to Euler angles conversion in this book: <http://astore.amazon.com/librarian06-20/detail/1441917497> page 21, section 3.6.3 “Euler Angles”, it is not a one-to-one conversion so I would recommend reading the entire chapter, to be aware of potential pitfalls. <http://www.starlino.com/wp-admin/edit-comments.php?p=226&approved=1#comments-form>

21. [Alfiansyah](#) | May 22, 2012

Thx Starlino! :D
I will soon take a look at your suggestion,

But how about an explanation from this page : <http://www.weizmann.ac.il/matlab/toolbox/aeroblks/directioncosinematrixtoeulerangles.html>

Simple and clear explanation there, but are the final formula could be directly applied to your DCM?

Thank you, for your replies :-)

22. starlino | May 22, 2012

Yes, this is same DCM (Direction Cosine Matrix).

23. [Alfiansyah](#) | May 22, 2012

Great book you suggest me, i'm still reading it, but i got really confused (like the “first time” i read your(DCM tutorial) article) because it uses different formula compared to articles i read before to get euler angles from Rotation Matrices.

I think i will close the book for today, it's 2 am in Indonesia, Time to Sleep, Thx for your help, thx for the book :D

Thank you very much :D

24. imunoob | May 28, 2012

Hallo starlino, this is a real good tutorial, thx!!!

But i have a small question:

If i have computed my DCM_(n), do i need to multiply DCM_(n) with DCM_(n-1)? insofar i understand DCM_(n) is the rotation matrix from step n-1 to n. If i want get the orientation DCM must i multiply them?

ps: sry for my english, i hope could understand me.

25. starlino | May 28, 2012

imunoob: no you don't need to multiply DCM_(n) with DCM_(n-1). DCM_(n) already contains absolute rotation from initial position. The initial position is chosen as DCM₍₀₎ = I (where I is 3×3 identity matrix, all 1-s on the diagonal). So all DCM-s after that are relative to that position.

26. [Alfiansyah](#) | June 6, 2012

Starlino, i almost forgot, i'm keeping this question by my self till now but the harder i think more confused i get.

Ok here it is, there is some difference between your DCM tutorial and your Code.

In your algorithm tutorial, the way you produce Angular displacement from Acc or Magneto is like this :

$d\theta_a = KB_0 \times (KB_{1A} - KB_0)$

But in your Code it look like this :

`vector3d_cross(dcmGyro[2],Kacc,wA);`

doesn't it has the same meaning as Doing cross Product between `dcm_est[2]` and `dcm_acc[2]` to get angular displacement? ($d\theta_a = KB_0 \times KB_{1A}$)

Did i miss something? In my program run on STM32 i build a program based on your DCM tutorial , and every things goes as expected. . . but till now i can't figure out what are you planning to calculate in your program. . (This line of code = `vector3d_cross(dcmGyro[2],Kacc,wA);`)

Can you help me to explain your line of code that i'm asking you? Thx :-)

27. starlino | June 7, 2012

Alfiansyah : $KB_0 \times (KB_{1A} - KB_0) = KB_0 \times KB_{1A} - KB_0 \times KB_0 = KB_0 \times KB_{1A}$, since $KB_0 \times KB_0 = 0$

28. Alfiansyah | July 5, 2012

Part 1:

Thx Starlino :D

several test about your dynamic weight algorithm were done . . .

one disadvantage i found is, because of high vibration from quadrotor, acc value became noisy. . very noisy. And it is possible to cause 1 g value in opposite direction of stable condition, causing the IMU to trust the accelerometer just in time were it should not be trusted , causing the angle estimation jump crazy to opposite direction it should be.

Example, in my first experiment i manually tilted my quad pitch to -9 degree, using maximum acc_w 0.1 (large indeed , just for test) and linearly decrease it to zero using $0.8 > X < 1.2$ range.

And then i got this result as shown in figure : http://4.bp.blogspot.com/-eBINhwn0M_k/T_YBFLHtBfI/AAAAAAAAABDg/cplun_IWVzA/s1600/IMU+Pitch.png

29. Alfiansyah | July 5, 2012

Part 2:

Realtime 10 sec sensor reading in that condition shown in figure:

http://1.bp.blogspot.com/-T64uV4etfP8/T_X9Oz1FtkI/AAAAAAAAABDQ/aAKuTibESSQ/s1600/IIII.png

you can see that quadrotor (i'am using the same draganflyer as yours) motor vibration really make the accelerometer became crazy. next interesting point from that image is magnetometer an gyroscope is steady as nothing happen (no magnetic field analysis for magnetometer were made though)

Second experiment is to remove the dynamic weight algorithm and manually change the acc weight from 0.1 to 0.01,

One trial sample is shown in this figure : http://2.bp.blogspot.com/-rJLCDqT8fUM/T_YMWn-5OgI/AAAAAAAAABDs/rFt2A-Mrhzs/s1600/imu_est.png

I got RMS error result Moving from 4.4degree to 4.0 degree (decrease) but i know that "if i decrease the acc weight more, gyro drift will dominate estimation result" and "if i keep increasing acc weight, more noise will dominate estimation result"

30. Alfiansyah | July 5, 2012

Part 3:

So in third experiment i decide to low pass filter the accelerometer raw data before using it to produce zenith vector

(lpf i use is $\Rightarrow acc_x = (coef1)acc_x + (coef2)raw_acc_x$; were $coef1 + coef2 = 1$) and i got better result in static condition. RMS error became 0.3 degree.

As shown is this figure : http://1.bp.blogspot.com/-Pbm54grQzxo/T_YOtu6wsyI/AAAAAAAAABD0/hr3CRHdzzu8/s1600/imu_est2.png

But i believe it will produce huge lag in dynamic analysis, and that is what gyroscope exist for. . . to remove lag in dynamic condition. so we need your dynamic weight algorithm once again but in different structure

I propose using Gyroscope and accelerometer raw data to detect translation, rotation movement and noisy environment, so that we can decide when and how much we change the weight of "angular displacement produce from gyro" and "angular displacement produce from lowpassfiltered accelerometer (not raw accelerometer)" in fusing angular displacement phase. i still don't have an idea how to take an advantage from "stable raw magnetometer data due to high vibration" is this analysis.

what do you think about my opinion sir?

31. starlino | July 6, 2012

Alfiansyah: the dynamic weighting algorithm purpose is to remove or decrease the acceleration data from the equation during SHORT periods of external acceleration for example a turn, a fall or a speed-up. It is not a solution for a constant noise. If the dynamic weighting algorithm is excluding accelerometer data all the time you're left with gyro only that will drift over time. As you noticed a noisy and uncalibrated accelerometer is worst than no accelerometer. Another good thing you noticed is that the low pass filter will introduce a delay. So what is the solution ?

It is actually not in the code or electronics, it's in the mechanics. First: the draganfly frame is really noisy and shaky, it's a real challenge. Nowadays you can buy good 'solid' frames for under \$20, look at 'hobbyking' site for example. Second: how is your accelerometer mounted ? Did you use solid screws ?

Because I would recommend soft foam mounting tape , use several layers to dampen the effect of vibration.

Treat the problem at the source first. Eliminate noise, with your software I am sure you can measure it. So get a good frame and mounting that will minimize noise. Next, treat the remaining noise with a low pass-filter (you can use by-pass caps on an analog accelerometer or implement it in code). My acc_gyro_6dof for example has 0.1uF by pass caps that can be increased by external caps. Finally, don't use dynamic weighting to eliminate something that would be constant (noise from motors). On the opposite you must include noise in the acceptable accelerometer data range since it's a constant during flight.

32. Alfiansyah | July 7, 2012

I really like your concept of "Treat the problem at the source first", I'am actually mounted it by softfoam but just one layer. . . I'am not thinking to find a solution from mechanics point of view, And now i realize it is important :P as you say.

Btw my friend and I successfully implemented draganflyer to fly low using 4 IR range sensor. . i want to share the video but i think it is not good enough to show to you, i will improve it so than i will share it.

33. Ash | October 2, 2012

Hi Starlino,

Must thank you for the clear explanations you have given. I have a question regarding the code that you used to calculate the pitch and roll of your quadcopter.

```
float errorPitch = dcmEst[0][0] - RwTrg[0]; (1)
```

```
float errorRoll = dcmEst[0][1] - RwTrg[1]; (2)
```

I gather that `dcmEst[0][x]` is the bodyvector expressed in the global cordinate system and you have taken the pitch and roll as the i and j components of this vector. However

if we are to derive the euler angles as per

<http://gentlenav.googlecode.com/files/EulerAngles.pdf>

the calculations are different, needing a negative inverse sin and a tan inverse to get the roll and pitch angles respectively.

Have you used the simpler appoximation as the quadcopter is constrained from going beyond a certain (0.3 in the code) roll and pitch limits ?

Many Thanks

Ash

34. starlino | October 2, 2012

You must be looking at `rev6` of `main.c` (that part of the code was still work in progress , and yes I believe the intention was just to get an estimate of the angle since for small angles $\arcsin(x) \sim x$ since I was probably just trying to get the device stabilized and hovering, although I should have used `dcm[2][0]` and `dcm[2][1]` which is the global Z axis in local coordinates ...), please have a look at `rev 8`,

<http://code.google.com/p/picquadcontroller/source/browse/trunk/main.c?r=8>

hopefully it makes more sense. Also `rev7` of `imu.h` has some fixes based on few comments here.

35. DeWitt Payne | October 4, 2012

I've ported your `imu.h` `rev 7` code to R to use to process gyro, accelerometer and GPS data collected while driving a car using an iPod with an external GPS dongle(Emprum Ultimate). I wanted to get more accurate lateral and lineal acceleration corrected for gravity while driving on a non-level surface and errors in orienting the iPod axes to the vehicle axes. After many stumbles, I think I've actually got it working. I've even managed to back out the approximate change in altitude. Your code and articles were a big help.

Thanks.

DeWitt Payne

36. DeWitt Payne | October 11, 2012

By the way, the `dcm_rotate` function in `rev 7` and `8` didn't work correctly for me. I went back to the code you commented out, matrix multiplication of the old `dcm` times the skew plus identity matrix. You had that coded incorrectly too. You were multiplying `W*dcm`, which is the wrong order. Matrix multiplication isn't commutative. It's `dcm*W` in Premierlani's article. The R language has matrix multiplication built in. The code is `dcm*%*%W`. Once I fixed that, everything fell into place.

37. starlino | October 11, 2012

DeWitt, the reason Premierlani has it the other way <http://gentlenav.googlecode.com/files/DCMDraft2.pdf> is because of the reference coordinate system . In fact he explains it after Eqn.13 on page 14 .

"...Ideally,

we would like to track the axes of the aircraft in the earth frame of reference, but the gyro measurements are made in the aircraft frame of reference. There is an easy solution to the issue by recognizing the symmetry in the rotation. In the frame of reference of the plane, the earth frame is rotating equal and opposite to the rotation of the plane in the earth frame. So we can track the earth axes as seen in the plane frame by flipping the sign of the gyro signals."

...

so $(-w) \times r = r \times w$

Rather than change the formula I change the sign of gyro's w, see <http://code.google.com/p/picquadcontroller/source/browse/trunk/imu.h?r=7>

```
//_____
```

```
//dcmGyro
```

```
//_____
```

```
float w[3]; //gyro rates (angular velocity of a global vector in local coordinates)
```

```
w[0] = -getGyroOutput(1); //rotation rate about accelerometer's X axis (GY output) in rad/ms
```

```
w[1] = -getGyroOutput(0); //rotation rate about accelerometer's Y axis (GX output) in rad/ms
```

```
w[2] = -getGyroOutput(2); //rotation rate about accelerometer's Z axis (GZ output) in rad/ms
```

note the - sign in front of `getGyroOutput`.

Hope this clarifies the issue. You gotta pay attention to the sign of gyro output. The best way to test is to set the weights of Acc and Mag to 0 and see if the algorithm triggers correct results.

38. William Premierlani | October 12, 2012

Hi Starlino,

I really enjoyed your tutorial. It is well written, interesting, comprehensive, and accurate. Nice work!

Bill Premierlani

39. DeWitt Payne | October 12, 2012

I see the difference now. But what I want is the lateral and lineal acceleration in the vehicle frame corrected for changes in the gravity vector caused mainly by driving on a non-level surface and for any error in orientation of the sensor axes relative to the vehicle axes. Pitch and roll of the vehicle itself from lineal

and lateral acceleration because the center of mass is above the vehicle roll and pitch axes is usually smaller. Since I'm post processing, I don't care all that much if the code isn't the most efficient. That might change if I try to do this in real time, but I'm not there yet.

Setting Acc and Mag to zero was indeed how I tested my code.

40. starlino | October 12, 2012

DeWitt have a look at Bill's simple dead reckoning algorithm: <http://diydrones.com/profiles/blogs/a-simple-deadreckoning> it might provide you some clues. Basically you first need to compute an accurate orientation using DCM algorithm, then extract the gravitation vector from it (G). Then: $\text{True Acceleration}(\text{vector}) = \text{Accelerometer Reading}(\text{vector}) - G(\text{vector})$.

41. DeWitt Payne | October 12, 2012

That's sort of what I'm doing except right now I'm just use the GPS speed and direction data. The implementation of the dead reckoning algorithm isn't all that obvious to me from the code, particularly the rather important correction for GPS latency. Right now I use a fixed latency determined by comparing the accelerometer data to acceleration calculated from the GPS speed and direction data. I adjust the latency to make the curves overlap. I only just realized that the dead reckoning code contains assembly language commands and uses integer data in the registers with left and right shifts to multiply and divide by factors of 2. Efficient code, but hard to read for someone who hasn't done assembly language programming for over forty years.

When I calculate the Euler angles from the calculated DCM, I get what looks like reasonable values. I can integrate the pitch angle and velocity to calculate altitude and get a plot that looks reasonable. It's not perfect because I don't include measured altitude in the correction feedback. The altitude changes are not large and the GPS I'm using doesn't measure altitude as well as it does horizontal position even when it says it has a 3D lock. Specifically, when I drive around a loop, the difference in GPS measured altitude at the start and finish is larger than the difference in horizontal position. It's about as far off as my calculated altitude. I'm using an Emprum Ultimate GPS on a gen 4 iPod and collecting the accelerometer, gyro rate and GPS data using the Sensor Data app.

42. William Premerlani | October 13, 2012

There have been a few recent improvements to the basic DCM algorithm. One of them is a method for accounting for acceleration in the roll-pitch drift compensation without requiring any sort of dynamic model of the vehicle or aircraft. So, it could be used on anything, such as a motorcycle, automobile, boat, luge sled, etc. It has been successfully used by the Ardu team. Here are a couple of links:

<http://diydrones.com/forum/topics/an-improved-method-for-accounting-for-acceleration-in-gyro-roll>

<http://www.diydrones.com/profiles/blogs/acceleration-compensation-flight-testing>

Best regards,
Bill Premerlani

43. DeWitt Payne | October 13, 2012

On inverting the axes:

I'm using the iPod in portrait orientation with the dock down. Because the z axis of the iPod is perpendicular to the face, I transpose the data so that the z axis data is treated as x axis data, the x axis is the y axis and the z axis is x axis. The directions of the axes are such that in this position, the z and x axis are already inverted, so if I invert the y axis data, I'm home. I did this to make things come out right, but now I understand why it needed to be done.

The correct accelerometer axis orientation would be face up with the dock to the right. But that's not exactly the most convenient position for use.

44. DeWitt Payne | October 13, 2012

Bill Premerlani,

Since I'm currently doing the calculations off-line because I don't have the time to deal with real time data while I'm driving, I can fake faster GPS updates by applying a cubic spline smooth to the GPS data and using that fit to interpolate between readings. Then I have GPS data at the same rate as accelerometer data. So for me, it makes little difference whether I'm doing the calculations in body or earth frame.

45. Ken | November 20, 2012

Hi Starlino,
Many thanks for the wonderful tutorial. I've got a question about the getting Euler angles from the gyro output. I've read somewhere that Gyro output (rad/s) is not the derivative of Euler angles.

So can we actually integrate Gyro output (rad/s), $w=[w_x, w_y, w_z]$ to get Euler angles?? (Euler angles = $w_x(dt), w_y(dt), w_z(dt)$??)
Thank you!! Really appreciate it.

46. starlino | November 20, 2012

Ken, you're right you need to obtain first the DCM by integration, by following the algorithm in this article for example. Then you convert the DCM to Euler angles. "Theory of applied robotics" has all the formulas also some cautionary notes when working with Euler angles. Amazon Link: http://www.amazon.com/gp/product/1441917497/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=librarian06-20&linkCode=as2&camp=217145&creative=399353&creativeASIN=1441917497

47. starlino | November 22, 2012

Also Ken, here is a pdf explaining one way to convert DCM matrix to Euler angles: <http://gentlenav.googlecode.com/files/EulerAngles.pdf>

48. Ken | November 21, 2012

Starlino, thanks for the fast reply! In your algorithm, when calculating the weighted average $d(\theta)$, how can we determine the weight 'sa' and 'sg'?
Many thanks!

49. starlino | November 21, 2012

$sa=1$, $sg=20$ will give good results in most cases. You can decreased sa dynamically if there's external acceleration detected (when acc is noisy it's modulus usually != 1g), this was discussed somewhere in the comments in one of the articles under "IMU Theory and Experiments".

50. Jess | November 22, 2012

Hi Starlino, GREAT tutorial! I understand how your DCM algorithm works by making use of accelerometer, magnetometer and gyro readings. This might sound like a stupid question, but would it be possible for you to explain a bit on Eq 15 and Eq 17 (page 14-15) of "Direction Cosine Matrix IMU: Theory (W. Premerlani)" ? i.e. where's the term $R(t)$ on the RHS of Eq17 comes from? and is $R(t+dt)$ the same as $DCM[G]$ which is Eq1.1 in your tutorial? Also, the only way to know $r_{\text{earth}}(0)$ in Eq 15 is to query accelerometer right? Thanks a bunch! :)

51. starlino | November 22, 2012

Jess, William put up a file explaining those equations, see: <http://gentlenav.googlecode.com/files/MatrixRotation.pdf>, hope this will lead you to the answers to the rest of the questions.

More docs related to DCMDraft2 can be found here: <http://code.google.com/p/gentlenav/downloads/list> (including the above document).

52. Miika | March 5, 2013

I'm hopefully misunderstanding something here but shouldn't the angular displacement be (let's ignore magnetometer)

$$d\theta = d\theta_g + s_a d\theta_a$$

as $d\theta_g$ is the gyro-based displacement and $d\theta_a$ is just a correction term which is ideally zero?

I think Premerlani had it so in <http://gentlenav.googlecode.com/files/DCMDraft2.pdf>?

When you have

$$d\theta = (s_a d\theta_a + s_g d\theta_g) / (s_a + s_g)$$

with $s_a=1$ and $s_g=20$ it's almost correct because $20/21 \approx 1$.

53. naga | March 19, 2013

this topic is very good

54. Matt | March 31, 2013

Awesome tutorial. Thank you!

55. Matt | April 17, 2013

I'm a little confused with your derivation of equation 2.5. In the derivation, you assume that your instantaneous axis of rotation of r is orthogonal to r . But this is not the case in general.

The equation

$$v = w \times r$$

does work in general, as long as the origin of your coordinate system lies on the axis of rotation (parallel to w). But if we manipulate this equation, cross multiplying both sides by r :

$$r \times v = r \times (w \times r)$$

$$r \times v = w(r \cdot r) - r(r \cdot w)$$

Rearranging,

$$w = ((r \times v) - r(r \cdot w)) / |r|^2$$

We obtain your equation 2.5 only if $(r \cdot w) = 0$.

Now if you are measuring the yaw component of w I can see why this would work, because you would have $r = I_b = (I_{bx}, I_{by})$ and $w = (0, 0, w_z)$, so $(r \cdot w) = 0$.

Care to elaborate?

56. starlino | April 19, 2013

Matt, the coordinate system was chosen to have same origin as w and r on purpose. Also w is perpendicular to r and r' by definition (it's perpendicular to the plain of rotation formed by r and r'), hence $w \cdot r = |w||r| \cos(w, r) = 0$. If you need to work with a different coordinate system you will need to perform a linear translation and then perform the rotation and then translate the result back.

57. David | April 21, 2013

Dear Starlino, thank you for this good article!

I have a question, please help me!

I want to extract the Euler angles relative to earth frame from gyro's outputs in a simulation study by using eqs: 17-21. Hence, in my code it is assumed that the gyro rotate about its x axis with angular velocity of 1 deg/sec and $dt=20\text{msec}$.

I expect that after 1 second; pitch, roll and yaw values to be 0, 1 and 1 degree respectively. But I never receive to this result. I am very confused. Are my exception is correct? Furthermore, these angles have large variation. What is my mistake point?

Best regards

58. Ajit | May 19, 2013

How does DCM do a basic complementary filter? I am not able to understand which block of the DCM does this. Could someone throw light on this?

59. [DCM Tutorial | arducopterarcic](#) | June 13, 2013

[...] DCM Tutorial [...]

60. Peter | June 14, 2013

Dear Sergiu!

Thank you very much for your efforts in publishing your knowledge about DCM theory!!

I used the sensors LSM330 (acc & gyro) and HMC5983 (compass), with your imu.h.

The calculations of pitch and roll are working fine!

Only yaw is kind of messed up: when i turn the device (let's say 90°), the yaw angle changes to 90 – then goes back to it's original value smoothly.

What could be wrong here??

I used this page to convert DCM to euler angles: [1]

I had another problem integrating the magnetometer..

In your code you used this:

```
//in the absense of magnetometer let's assume North vector (I) is always in XZ plane of the device (y coordinate is 0)
```

```
Imag[0] = sqrt(1-dcmEst[0][2]*dcmEst[0][2]);
```

```
Imag[1] = 0;
```

```
Imag[2] = dcmEst[0][2];
```

When i feed the Magnetometer readings into Imag[0-2] (instead of your values from the DCM matrix) and normalize them, i get messed up angles.

What could have gone wrong here?

I would really appreciate, if you could help me out a little here!

Thank you,

Peter

[1] <http://www.weizmann.ac.il/matlab/toolbox/aeroblks/directioncosinematrixtoeulerangles.html>

61. starlino | June 14, 2013

Peter, regarding the yaw angle, it hard to say why it is 90 degrees at first but it could be because you are not innitalizing the DCM matrix with the magnetometer data This is why it is pointing into a fixed direction first.

As far as second question is concerned – did you calibrate your magnetometer ? What values are you expexting ? Please read up in the comments of IMU guide how to innitalize your absolute North vector (Imag). The magnetic field is not pointing north ! In usa it is pointing almost down and north, this is why you need to do some processing. Look at comment #16 above

62. b1 | June 20, 2013

Peter, starlino.

I have the same problem even after using triple product from comment 16. Under small angle rotation (<90) it works as intended, but then if angle become more than 90, body coordinate system flips slowly. I can see work of complementary filter, slow flip comes from magnetometer vector that changes direction. Effect can only be seen from rotation by Y axis. Maybe we need to build stabilized compass with accelerometer and magnetometer first and then perform triple product?

Anyway, thanks for good explanation!

63. Alex Zarenin | July 17, 2013

Hi Sergiu,

Your great tutorial gave me a refreshes course on the DCM algorithm, which I first found in Bill Premerlani article. I am working now on my third IMU board and third quad – and this one I managed to put in the air and land successfully without crashing (yet)! My DCM is quite stable and I am using PID parameters that I adjusted on the stand, mu quad is in the "+" configuration – that is one axis goes front to back and the other one – left to right.

The problem hat I am dealing with is the precession – just by itself the quad is quite stable, but if I apply some tilt control (or wind or any other disturbance tilts it) it quite often starting to get into the precession mode where pitch and roll follow a sinusoids with roughly the same amplitude of about +/-0.2 rad with the phase shift of about Pi/2. I collect the full telemetry of the flight (RC control values, all the angles, proprtional, derivative and integral components for 3 PID controllers by axis, actual motor control values, etc) onto the on-board data logger and when I analyze it in Excel it looks like the problem comes from the fact that contrary to the test stand, which limits model movement to one degree of freedom, in a real flight all three controls become tied together.

For example, if the model has simultaneouslu some pitch and roll trying to adjust pitch by changing balance between the front and back motors also generates a Yaw component; controlling Yaw will affect both Roll and Pitch etc. So my struggle now is in how to move away from controlling each axis individually to something more integral.

Any advice would be appreciated!

Thank you,

–Alex

64. [OlliW's Bastelseiten » IMU Data Fusing: Complementary-, Kalman- and Mahony Filter](#) | September 17, 2013

[...] and Gyroscope Devices) in Embedded Applications (Dez. '09) [link] – by Starlino [St2] DCM Tutorial – An Introduction to Orientation Kinematics (May '11) [link] – by Starlino (or as [...])

65. [HennSun](#) | December 9, 2013

I have translated it into chinese ,and mentioned the original link.

<http://www.cnblogs.com/dreamfactory/p/3424953.html>

66. Leonardo Garberoglio | December 24, 2013

Hi Starlino, how are you?

I have code working with just gyro's. On still position I obtain this data:

<http://www.4shared.com/office/KWxaj4q/dcm.html>

i put gyro data (deg/ms) and then 9 dcm values. Sample time is 20msec

Is it ok? what can I do to verify my data is correct?

I see small variation on each sample. But each element on DCM matrix change a lot over the time....

Thk and best regards!

67. Leonardo Garberoglio | December 24, 2013

Starlino, I think I make a progress. I add accel correction according to your code. First, with 0.01 weight seems to not work. I put 0.05 on accel weight and now I can see that dcmEst[2][2] is always almost 1.

I don't know why other itmes on dcmEst varies form 1 to -1 and to 1 again over the time... it is stable for about 15-20 second, and then gradually change dcmEst[0][0] from 1 to lower values....

This is my new data dump. I put 3 gyro data and 9 dcmEst value. Always on still position.

http://www.4shared.com/office/6th1bFpX/DCM_St_A_3.html

Is it normal? is the the magnetometer the solution or there is no problem?

Thk and best regards!

68. Leonardo Garberoglio | December 27, 2013

My brain is just about to exlote... I have read every implementation that i found on the web but i just can't make it work.

I log gyro and accel data and, on still position, gyro data y almost 0 deg/seg (0,01), accel y about 0,0,97 g (x, y, z)

I make my DCM matrix just like you y copy your code and paste on my project, then make litle change on sensor reading.

DCM matrix start with [1,0,0][0,1,0][0,0,1] and change a few from the first 10-20 seconds. Then first and second row's start to change. I.i vector start to decrease while I.j vector increase. normalization is ok, becouse module of each row is 1. and orthonormalization is ok too becouse de cross product of first and second row is equal to 3rd row:

```
0.511  0.859  -0.024
-0.860  0.510  -0.021
-0.006  0.032  0.999
```

but, why change first and second row's change a lot over time? And when I rotate the IMU on one axis and then put ir on still position again i don't obtain identity matrix (or almost) again. And finally if a obtain pitch roll and yaw angles they don't represent angular position of the imu....

I don't know what is wrong! i'm thinking on put magnetometer... but most paper that i read say that it have to work with gyro and accel only....

Could you please help me to find the problem?

best regards!

69. starlino | December 27, 2013

Leonardo, to isolate the problem try setting the Magnetometer and Accelerometer weights to 0. Whithout moving the device your DCM output should stay close to identity matrix:

```
1 0 0
0 1 0
0 0 1
```

If you move around any X axis then only 2 rows or columns should update (the other one should stay 1 0 0).

If you don't get this result – you have a problem in gyro sensitivity calculations, or gyro is not calibrated and drifts while stationary.

70. Leonardo Garberoglio | December 28, 2013

Ok, It doesn't work.

With gyro data only, the matrix is close to identity for almost 20 sec.

This is my init code for gyro:

```
void L3G4200D_init(){
    L3G4200D_write(L3G4200D_CTRL_REG1, 0b00001111); // 100Hz, 12.5 CO, all axis enable
    // L3G4200D_write(L3G4200D_CTRL_REG1, 0b01001111); // 200Hz, 12.5 CO, all axis enable
    L3G4200D_write(L3G4200D_CTRL_REG4, 0x20); // 0x00 250 dps, 0x10 500 dps, 0x20 2000dps
}
```

This is my calibration code:

// Determine zero bias for all axes of both sensors by averaging 50 measurements

```
void L3G4200D_GetBiass(void){
```

```
    int i;
    for (i = 0; i < 600; i += 1) {
        L3G4200D_read_data();
        config.gyroOffs[0] += gyro_X;
        config.gyroOffs[1] += gyro_Y;
        config.gyroOffs[2] += gyro_Z;
        delay_ms(1);
    }
    for (i = 0; i < 3; i++) {
        config.gyroOffs[i] /= 600;
    }
}
```

and this is a few data for my log file:

```
20.000, -0.001, -0.001, -0.001, 0.832, 0.123, 0.541, -0.104, 0.992, -0.067, -0.546, -0.001, 0.838
20.000, -0.001, 0.006, -0.006, 0.836, 0.118, 0.536, -0.098, 0.993, -0.067, -0.540, 0.003, 0.842
20.000, -0.002, -0.006, -0.005, 0.833, 0.115, 0.541, -0.092, 0.993, -0.070, -0.546, 0.008, 0.838
20.000, -0.002, -0.005, 0.005, 0.830, 0.121, 0.545, -0.097, 0.993, -0.072, -0.550, 0.007, 0.835
20.000, 0.003, 0.001, -0.004, 0.830, 0.116, 0.545, -0.093, 0.993, -0.069, -0.549, 0.006, 0.836
20.000, -0.001, -0.006, -0.009, 0.828, 0.109, 0.550, -0.084, 0.994, -0.070, -0.554, 0.012, 0.832
20.000, -0.004, -0.002, -0.002, 0.827, 0.109, 0.551, -0.082, 0.994, -0.074, -0.556, 0.016, 0.831
20.000, 0.002, 0.001, 0.002, 0.827, 0.110, 0.551, -0.084, 0.994, -0.072, -0.556, 0.014, 0.831
20.000, -0.001, 0.005, 0.001, 0.830, 0.111, 0.547, -0.085, 0.994, -0.073, -0.552, 0.014, 0.834
20.000, -0.002, -0.006, 0.005, 0.826, 0.116, 0.552, -0.089, 0.993, -0.075, -0.557, 0.013, 0.830
20.000, 0.008, -0.001, 0.003, 0.825, 0.115, 0.553, -0.092, 0.993, -0.068, -0.558, 0.005, 0.830
20.000, -0.005, 0.001, 0.002, 0.825, 0.119, 0.552, -0.094, 0.993, -0.073, -0.557, 0.008, 0.830
20.000, -0.001, -0.004, -0.002, 0.823, 0.118, 0.555, -0.092, 0.993, -0.074, -0.560, 0.010, 0.828
```

20 is my imu_update loop (msec)

first 3 value is scaled, unbiased gyro data. deg/sec

Last 9 value is DCM matrix. as you can see first and last vector are drift from identity. only 2nd vector is stable.

Have you got any idea where the problem is?

By the way, with gyro data only, how much time dcm matrix is close to identity before it start to drift?

Thk for your time!

71. Leonardo Garberoglio | December 28, 2013

Starlino, how are you?

I still working on it, but I don't make any progress.

I'm trying to obtain lower values from my gyro.

I change a little my code to test it. This is my main loop:

```
if((msTicks-timer) >= 20 ){
    adcAvg[3]=0;
    adcAvg[4]=0;
    adcAvg[5]=0;
    for(i=0;i<10;i++){
        L3G4200D_read_data();
        adcAvg[3]+=gyro_X;
        adcAvg[4]+=gyro_Y;
        adcAvg[5]+=gyro_Z;
    }
    for(i=0;i<3;i++){
        adcAvg[3+i]/=10;
    }

    timer=msTicks;
    imu_update();
}
```

every 20msec I read ten times my gyro and average the values. then I call imu_update. because of 10's reading my imu loop is about 28msec.

But I'm still having +0.001 to +0.008 from my gyro. With this values, DCM matrix is close to identity for a few seconds, and then start to drift. I check normalization and orthogonality of new calculated DCM and it's Ok. DCM code seems to be ok.... so I don't know if my gyro data is ok and it's normal to DCM drift after a few seconds or my gyros is wrong and DCM should be un-drift for more time....

How do you think? have you got log file for your raw data to compare with mine?

Sorry by all of this, but you are the only one that answer me!!!!

Thk again for your time!

72. Leonardo Garberoglio | December 28, 2013

Starlino, forget all of my questions.... I just read a tons of web's talking about noise on this specific Gyro... So my raw data is Ok and DCM drift to fast because of gyro noise...

So righth now I am searching for some filter to the Gyro.... Have you got any filter code that I can use?

By the way, I see a lot of project using my IMU (GY-80) working. So I'm wondering my self to implement accelerometer and magnetometer corrections... what do you think?

73. Leonardo Garberoglio | December 31, 2013

Well, I continue to DCM test. It's doesn't work yet.

I found a new problem.

I read only one gyro axis and working with DCM with 3 values. So I expect that only 4 values of DCM matrix will change while I rotate my IMU. When I test it I found that only 4 values change, but they change to much. I mean, while I rotate from 0 to 10 degrees each DCM value changes from 1 to 0 then to -1 then to 0 and to 1 again several times... I log gyro output, angle integration and DCM values. I obtain this:

```
7.7733 -14.1717 -15.2583 18.2745 4.0199 2.7472

20.000, 0.000, 0.000, 0.000, 0.000, 0.999, 0.026, 1.000, 0.000, 0.000, 0.000, 1.000, 0.000, 0.000, 0.000, 1.000
20.000, 0.000, 0.000, 0.000, 0.000, 0.999, 0.026, 1.000, 0.000, 0.000, 0.000, 1.000, 0.000, 0.000, 0.000, 1.000
20.000, 0.030, 0.000, 0.000, -0.030, 1.007, 0.026, 1.000, 0.000, 0.000, 0.000, 1.000, 0.030, 0.000, -0.030, 1.000
20.250, 0.041, 0.000, 0.000, -0.071, 1.017, 0.026, 1.000, 0.000, 0.000, 0.000, 0.997, 0.071, 0.000, -0.071, 0.997
20.750, 0.040, 0.000, 0.000, -0.112, 1.026, 0.021, 1.000, 0.000, 0.000, 0.000, 0.994, 0.111, 0.000, -0.111, 0.994
20.250, 0.055, 0.000, 0.000, -0.167, 1.032, 0.021, 1.000, 0.000, 0.000, 0.000, 0.986, 0.166, 0.000, -0.166, 0.986
20.750, 0.051, 0.000, 0.000, -0.217, 1.040, 0.021, 1.000, 0.000, 0.000, 0.000, 0.977, 0.215, 0.000, -0.215, 0.977
20.250, 0.056, 0.000, 0.000, -0.273, 1.047, 0.021, 1.000, 0.000, 0.000, 0.000, 0.963, 0.270, 0.000, -0.270, 0.963
20.750, 0.071, 0.000, 0.000, -0.344, 1.056, 0.014, 1.000, 0.000, 0.000, 0.000, 0.941, 0.337, 0.000, -0.337, 0.941
20.250, 0.058, 0.000, 0.000, -0.402, 1.056, 0.014, 1.000, 0.000, 0.000, 0.000, 0.920, 0.391, 0.000, -0.391, 0.920
20.750, 0.058, 0.000, 0.000, -0.460, 1.056, 0.009, 1.000, 0.000, 0.000, 0.000, 0.896, 0.443, 0.000, -0.443, 0.896
20.250, 0.046, 0.000, 0.000, -0.506, 1.066, 0.009, 1.000, 0.000, 0.000, 0.000, 0.875, 0.485, 0.000, -0.485, 0.875
20.750, 0.037, 0.000, 0.000, -0.544, 1.072, 0.009, 1.000, 0.000, 0.000, 0.000, 0.856, 0.517, 0.000, -0.517, 0.856
20.250, 0.035, 0.000, 0.000, -0.579, 1.078, 0.009, 1.000, 0.000, 0.000, 0.000, 0.837, 0.547, 0.000, -0.547, 0.837
20.750, 0.030, 0.000, 0.000, -0.609, 1.078, 0.009, 1.000, 0.000, 0.000, 0.000, 0.820, 0.572, 0.000, -0.572, 0.820
20.250, 0.000, 0.000, 0.000, -0.609, 1.084, 0.009, 1.000, 0.000, 0.000, 0.000, 0.820, 0.572, 0.000, -0.572, 0.820
20.500, 0.033, 0.000, 0.000, -0.642, 1.092, 0.013, 1.000, 0.000, 0.000, 0.000, 0.801, 0.598, 0.000, -0.598, 0.801
20.500, 0.031, 0.000, 0.000, -0.673, 1.092, 0.013, 1.000, 0.000, 0.000, 0.000, 0.782, 0.623, 0.000, -0.623, 0.782
20.500, 0.000, 0.000, 0.000, -0.673, 1.103, 0.013, 1.000, 0.000, 0.000, 0.000, 0.782, 0.623, 0.000, -0.623, 0.782
20.500, 0.038, 0.000, 0.000, -0.711, 1.103, 0.013, 1.000, 0.000, 0.000, 0.000, 0.758, 0.652, 0.000, -0.652, 0.758
20.500, 0.037, 0.000, 0.000, -0.748, 1.103, 0.013, 1.000, 0.000, 0.000, 0.000, 0.733, 0.680, 0.000, -0.680, 0.733
20.500, 0.051, 0.000, 0.000, -0.800, 1.109, 0.013, 1.000, 0.000, 0.000, 0.000, 0.697, 0.717, 0.000, -0.717, 0.697
20.500, 0.046, 0.000, 0.000, -0.845, 1.115, 0.008, 1.000, 0.000, 0.000, 0.000, 0.664, 0.748, 0.000, -0.748, 0.664
20.500, 0.050, 0.000, 0.000, -0.895, 1.124, 0.013, 1.000, 0.000, 0.000, 0.000, 0.626, 0.780, 0.000, -0.780, 0.626
20.500, 0.063, 0.000, 0.000, -0.958, 1.124, 0.004, 1.000, 0.000, 0.000, 0.000, 0.576, 0.818, 0.000, -0.818, 0.576
20.500, 0.066, 0.000, 0.000, -1.024, 1.130, 0.004, 1.000, 0.000, 0.000, 0.000, 0.521, 0.854, 0.000, -0.854, 0.521
20.500, 0.069, 0.000, 0.000, -1.092, 1.140, 0.004, 1.000, 0.000, 0.000, 0.000, 0.461, 0.887, 0.000, -0.887, 0.461
20.500, 0.054, 0.000, 0.000, -1.146, 1.148, 0.004, 1.000, 0.000, 0.000, 0.000, 0.413, 0.911, 0.000, -0.911, 0.413
20.500, 0.084, 0.000, 0.000, -1.231, 1.148, -0.002, 1.000, 0.000, 0.000, 0.000, 0.335, 0.942, 0.000, -0.942, 0.335
20.750, 0.094, 0.000, 0.000, -1.325, 1.148, 0.004, 1.000, 0.000, 0.000, 0.000, 0.245, 0.970, 0.000, -0.970, 0.245
20.250, 0.113, 0.000, 0.000, -1.438, 1.159, 0.004, 1.000, 0.000, 0.000, 0.000, 0.134, 0.991, 0.000, -0.991, 0.134
20.500, 0.122, 0.000, 0.000, -1.560, 1.165, 0.004, 1.000, 0.000, 0.000, 0.000, 0.014, 1.000, 0.000, -1.000, 0.014
20.500, 0.116, 0.000, 0.000, -1.676, 1.180, 0.004, 1.000, 0.000, 0.000, 0.000, -0.101, 0.995, 0.000, -0.995, -0.101
```

First value is time step. Then GyroX, GyroY, GyroZ (dps). Next is integral of previous values, angle X, angle Y, angle Z. Finally DCM matrix. As you can see I rotate only 1.6 degrees and dcm matrix change from 1 to 0. I think that each 4 values that is changing where cos X and sin X.... but it seems to be another think....

Someone have any idea where the problem is? Is my asumtion right?

Best regards! and happy new year!

74. Leonardo Garberoglio | January 2, 2014

I make some matlab script to test my code:

```
clear;
imu_interval=0.01975;
dcmEst=[1 0 0; 0 1 0; 0 0 1];
gyro=[-0.000476962508 -0.00100231264 -0.000903233362];
g=[1 -gyro(3) gyro(2); gyro(3) 1 -gyro(1); -gyro(2) gyro(1) 1];
dcmEst=dcmEst*g;
error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
x_est=[0 0 0];
y_est=[0 0 0];
x_est = dcmEst(2,:) * error;
y_est = dcmEst(1,:) * error;
dcmEst(1,:) = dcmEst(1,:) + x_est;
dcmEst(2,:) = dcmEst(2,:) + y_est;
dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
dcmEst(1,:)=dcmEst(1,:)/norm(dcmEst(1,:));
dcmEst(2,:)=dcmEst(2,:)/norm(dcmEst(2,:));
dcmEst(3,:)=dcmEst(3,:)/norm(dcmEst(3,:));
gyro
dcmEst

gyro=[-0.00204776251 -0.00414881203 0.00357279973];
g=[1 -gyro(3) gyro(2); gyro(3) 1 -gyro(1); -gyro(2) gyro(1) 1];
dcmEst=dcmEst*g;
error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
x_est=[0 0 0];
y_est=[0 0 0];
x_est = dcmEst(2,:) * error;
y_est = dcmEst(1,:) * error;
dcmEst(1,:) = dcmEst(1,:) + x_est;
dcmEst(2,:) = dcmEst(2,:) + y_est;
dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
dcmEst(1,:)=dcmEst(1,:)/norm(dcmEst(1,:));
dcmEst(2,:)=dcmEst(2,:)/norm(dcmEst(2,:));
dcmEst(3,:)=dcmEst(3,:)/norm(dcmEst(3,:));
gyro
dcmEst

gyro=[0.0533423312 0.399783999 0.0112587996];
g=[1 -gyro(3) gyro(2); gyro(3) 1 -gyro(1); -gyro(2) gyro(1) 1];
dcmEst=dcmEst*g;
error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
x_est=[0 0 0];
y_est=[0 0 0];
x_est = dcmEst(2,:) * error;
y_est = dcmEst(1,:) * error;
dcmEst(1,:) = dcmEst(1,:) + x_est;
dcmEst(2,:) = dcmEst(2,:) + y_est;
dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
dcmEst(1,:)=dcmEst(1,:)/norm(dcmEst(1,:));
dcmEst(2,:)=dcmEst(2,:)/norm(dcmEst(2,:));
dcmEst(3,:)=dcmEst(3,:)/norm(dcmEst(3,:));
gyro
dcmEst

gyro=[0.0533423312 1.1 0.012587996];
g=[1 -gyro(3) gyro(2); gyro(3) 1 -gyro(1); -gyro(2) gyro(1) 1];
dcmEst=dcmEst*g;
error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
x_est=[0 0 0];
y_est=[0 0 0];
x_est = dcmEst(2,:) * error;
y_est = dcmEst(1,:) * error;
dcmEst(1,:) = dcmEst(1,:) + x_est;
dcmEst(2,:) = dcmEst(2,:) + y_est;
dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
dcmEst(1,:)=dcmEst(1,:)/norm(dcmEst(1,:));
dcmEst(2,:)=dcmEst(2,:)/norm(dcmEst(2,:));
dcmEst(3,:)=dcmEst(3,:)/norm(dcmEst(3,:));
gyro
```

dcmEst

As you can see I put 4 DCM loop, first with low values from my gyro (still position), next with 0.4 deg/sec and 1.1 deg/sec on Y axis.

This is my matlab outputs:

```
>> DCM_01

gyro =

    -0.0005   -0.0010   -0.0009

dcmEst =

    1.0000    0.0009   -0.0010
   -0.0009    1.0000    0.0005
    0.0010   -0.0005    1.0000

gyro =

   -0.0020   -0.0041    0.0036

dcmEst =

    1.0000   -0.0027   -0.0051
    0.0027    1.0000    0.0025
    0.0051   -0.0025    1.0000

gyro =

    0.0533    0.3998    0.0113

dcmEst =

    0.9306   -0.0035    0.3661
    0.0234    0.9987   -0.0456
   -0.3655    0.0510    0.9294

gyro =

    0.0533    1.1000    0.0126

dcmEst =

    0.3567    0.0203    0.9340
    0.0996    0.9943   -0.0370
   -0.9297    0.1062    0.3527

>>
```

As you can see first two (or three) dcm matrix is close to identity and it is what I expected.

But when I rotate my gyro (slowly) and put this value on dcm calculation I obtain something that I don't expect. If I rotate 1.1 deg/sec dcm matrix is far away from identity... Is that right? Am I make wrong math???

Thk and best regards!

75. starlino | January 2, 2014

Leonardo, it is ok and even expected for DCM matrix to be different from identity matrix once you rotate away from initial position. The DCM matrix should always be orthonormal however (module of any column/row should be 1 and dot product of any row/column combination should be 0).

One test you could do – try rotating the device one full loop (slowly but steady) and come back to initial position. You should be back to identity matrix when you are back to initial position. How far away you are from identity matrix – this will tell you how well your gyros are calibrated.

Once you achieved this to some level of precision of course, you could move on and add the other sensors to equation.

76. Leonardo Garberoglio | January 2, 2014

Thk Starlino, once again for your answer.

I'm seeing this formula from wikipedia: <http://upload.wikimedia.org/math/8/8/6/8867b40e7d7d69e2db9cba8a7b42d2db.png>

I focusing on Δy individual rotation. If I understand, what I expect for a rotation around Y axis is $dcm[0][0]$ will change from 1 to -1 while I rotate the gyro 180° following cosine function and $dcm[0][2]$ will change from 0 to 0 (passing through -1) following sine function. Same thing will happen with $dcm[2][0]$ and $dcm[2][2]$, and $dcm[1]$ will stay [0 1 0]. Am I right?

What I see is with little angle my first dcm's row change from [1 0 0] to [0 0 1] to [-1 0 0] several times when I rotate a few degrees...

I think that I misunderstood the algorithm....

What I understand is:

Let's assume that my gyro is rotating at a constant speed around Y axis with a rate of 15 deg/sec (is it ok or is it too fast?). So my gyro data will be [0 15 0] right? My timeStep is 20msec, so if I pause my uC after 1 sec. I have to see

$[\cos 15^\circ \ 0 \ -\sin 15^\circ]$

$[0 \ 1 \ 0]$

$[\sin 15^\circ \ 0 \ \cos 15^\circ]$

Is that ok?

I made this matlab code:

```
clear;
imu_interval=0.02;
dcmEst=[1 0 0; 0 1 0; 0 0 1];
gyro=[0 15 0] * imu_interval;
g=[1 -gyro(3) gyro(2); gyro(3) 1 -gyro(1); -gyro(2) gyro(1) 1];
for n = 1:50
    dcmEst=dcmEst*g;
    error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
    x_est=[0 0 0];
    y_est=[0 0 0];
    x_est = dcmEst(2, :) * error;
    y_est = dcmEst(1, :) * error;
    dcmEst(1,:) = dcmEst(1,:) + x_est;
    dcmEst(2,:) = dcmEst(2,:) + y_est;
    dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
    dcmEst(1,:)=dcmEst(1,:)/norm(dcmEst(1,:));
    dcmEst(2,:)=dcmEst(2,:)/norm(dcmEst(2,:));
    dcmEst(3,:)=dcmEst(3,:)/norm(dcmEst(3,:));
    gyro
    dcmEst
end
```

And the las dcmEst matrix is:

```
dcmEst =

    -0.4220     0    0.9066
         0    1.0000         0
    -0.9066     0   -0.4220
```

wich is not what I expected...

Starlino, once again, thk for your time. I hope I will pay with the same to you on the future!

77. Leonardo Garberoglio | January 3, 2014

I continue with my MatLab work to trying to understand DCM.

Take a look at this code:

```
clear;
imu_interval=0.02;
dcmEst=[1 0 0; 0 1 0; 0 0 1];
gyro=[0 1 0] * imu_interval;
g=[1 -gyro(3) gyro(2); gyro(3) 1 -gyro(1); -gyro(2) gyro(1) 1];
graf(5000,2)=zeros;
for n = 1:5000
    dcmEst=dcmEst*g;
    error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
    x_est=[0 0 0];
    y_est=[0 0 0];
    x_est = dcmEst(2, :) * error;
    y_est = dcmEst(1, :) * error;
    dcmEst(1,:) = dcmEst(1,:) + x_est;
    dcmEst(2,:) = dcmEst(2,:) + y_est;
    dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
    dcmEst(1,:)=dcmEst(1,:)/norm(dcmEst(1,:));
    dcmEst(2,:)=dcmEst(2,:)/norm(dcmEst(2,:));
    dcmEst(3,:)=dcmEst(3,:)/norm(dcmEst(3,:));
    graf(n,1)=n*imu_interval;
    graf(n,2)=dcmEst(1,1);
end
figure
plot(graf(:,1),graf(:,2));
```

In this case I put 1 deg/sec on my Y gyro, my timeStep is 20 msec. I run a loop for $0.02 \times 5000 = 10$ sec. I save DCM[0][0] and time to make a graph. This is what I obtain:

<http://imageshack.us/a/img197/506/bj9i.jpg>

As you can see DCM[0][0] is going from 1 to 0 to -1 to 0 to 1 several times... Is it ok? I think that DCM[0][0] will going from 1 to 0 while gyro rotate from 0 to 90°... So is my asumtion right or is my graph right?

Regards!

78. Leonardo Garberoglio | January 4, 2014

Starlino, I think I found my mystake!!!! I'm on work righth now, when I get home I will test it.

I think it is on gyro unit. You say this on your imu.h:

```
//-----
//Get gyro reading (rate of rotation expressed in deg/ms)
//-----
float getGyroOutput(unsigned char w){
    static float tmpf; //temporary variable for complex calculations
    tmpf = adcAvg[3+w] - config.gyroOffs[w]; //remove offset
    tmpf /= config.gyroSens[w]; //divide by sensitivity
    if( config.gyroInv[w]) tmpf = - tmpf; //invert axis value if needed
    return tmpf;
}
```

So I use gyro data in deg/msec

But in your config.h you write:

```
// Gyro X,Y axes (PITCH, ROLL)
// Gyro Zero Level 1.23V @ 0 deg/s = 1023 * 1.23 / 3.3 = 381.3 ADC
// 1 mV / (deg/s) = 1023/3300 ADC / ( (PI/180)rad / 1000ms ) = 17761.69165 ADC /(rad/ms)
// Gyro Sensitivity 2 mV/(deg/s) = 2 * 17761.69165 ADC /(rad/ms) = 35523.3833 ADC /(rad/ms)
// Gyro Resolution 1ADC <=> 0.000028 rad/ms

config.gyroOffs[0] = 383.15;
config.gyroOffs[1] = 385.57;

config.gyroSens[0] = 35523.3833;
config.gyroSens[1] = 35523.3833;
```

If I understund your sensitivity convert gyro data to RAD/msec.

I change my last matlab code to this:

```
clear;
imu_interval=0.02;
dcmEst=[1 0 0; 0 1 0; 0 0 1];
gyro=[0 1*0.017453293 0] * imu_interval;
g=[1 -gyro(3) gyro(2); gyro(3) 1 -gyro(1); -gyro(2) gyro(1) 1];
graf(18000,2)=zeros;
for n = 1:18000
    dcmEst=dcmEst*g;
    error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
    x_est=[0 0 0];
    y_est=[0 0 0];
    x_est = dcmEst(2,:) * error;
    y_est = dcmEst(1,:) * error;
    dcmEst(1,:) = dcmEst(1,:) + x_est;
    dcmEst(2,:) = dcmEst(2,:) + y_est;
    dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
    dcmEst(1,:)=dcmEst(1,:)/norm(dcmEst(1,:));
    dcmEst(2,:)=dcmEst(2,:)/norm(dcmEst(2,:));
    dcmEst(3,:)=dcmEst(3,:)/norm(dcmEst(3,:));
    graf(n,1)=n*imu_interval;
    graf(n,2)=dcmEst(1,1);
end
figure
plot(graf(:,1),graf(:,2));
```

I put 1°/sec on my gyro and convert it into rad/sec. Then I run the loop for 18000 steps (18.000*0,02 = 360). I expect that DCM[0][0] change from 1 to 0 to -1 to 0 and finally to 1 again and this is what I obtain.

<http://imageshack.us/a/img585/4056/w5ic.jpg>

So it's seems to work in simulation right now!!!!!!

I will test it on board and report back!!!!

79. Leonardo Garberoglio | January 4, 2014

I'm crying like a baby!!!!!! after 1 month of test, reading, more test and more reading it's working now!!!!

I'm a highschool teacher and always sed to my class "use the correct unit's!!!!, don't forget to check the unit's!!!!" And now I made this mistake.... Gyro data must be on rad/sec not °/sec....

I'm really crying....

80. Leonardo Garberoglio | January 6, 2014

Starlino, with your permission I'm translating part of your work to spanish on this forum:

<http://www.todopic.com.ar/foros/index.php?topic=41978.msg348892#msg348892>

Regards!

81. Leonardo Garberoglio | January 9, 2014

Starlino, I have a new question:

If i rotate the IMU at a rate of [1 2 3] deg/sec for 10 second, do I expect to obtain 10deg of pitch, 20deg of roll and 30deg for yaw?

I know about euler angles singularities, but I rotate 30deg max on yaw...

Is this equation valid to obtain euler angles $< 90^\circ$?

Pitch = $-\text{asin}(\text{dcmEst}(3,1))$;

Roll = $\text{atan2}(\text{dcmEst}(2,1), \text{dcmEst}(1,1))$;

Yaw = $\text{atan2}(\text{dcmEst}(3,2), \text{dcmEst}(3,3))$;

Best regards!

82. starlino | January 9, 2014

Leonard , couple of pointers that might answer your questions

<http://www.soi.city.ac.uk/~sbbh653/publications/euler.pdf> (especially see the section What if $\cos \theta = 0$?)

<http://www.chrobotics.com/docs/AN-1005-UnderstandingEulerAngles.pdf>

<https://code.google.com/p/gentlenav/downloads/detail?name=EulerAngles.pdf&can=2&q=>

83. Leonardo Garberoglio | January 10, 2014

I already read euler.pdf and I'm limiting my experiment to pitch, roll and yaw from 0 to 30° just to be sure I understand DCM. I already understand and make code to work with one rotation at time.

I mean, I rotate $1^\circ/\text{sec}$ for 15sec about X axis and I obtain $\cos(15)$ and $\sin(15)$ on DCM. That's fine.

Rigth now I'm testing simultaneous rotation rate.

So I use this matlab code:

```
clear;
clc;
imu_interval=0.02;
%La posicion inicial en reposo, los marcos de referencia coincide, por lo
%que la DCM inicial es la matriz identidad
dcmEst=[1 0 0; 0 1 0; 0 0 1];
wx=1; %Velocidad angular en °/seg en cada eje
wy=2;
wz=3;
W=[wx wy wz]*pi/180; %Vector velocidad angular en rad/seg
Theta=W*imu_interval; %Multiplicamos por el tiempo de muestreo para obtener ángulo
%Creamos la matriz de rotacion actual
g=[1 -Theta(3) Theta(2); Theta(3) 1 -Theta(1); -Theta(2) Theta(1) 1];
graf(500,4)=zeros; %Array para gráfico
for n = 1:500 %50 pasos * 0,02seg = 10 segundos
    %Actualizamos la DCM girándola de acuerdo al array g
    dcmEst=dcmEst*g;
    %Tenemos que hacerla orthogonal nuevamente.
    %Calculamos el error de ortogonalidad
    error=-dot(dcmEst(1,:),dcmEst(2,:))*0.5;
    %Repartimos el error escalando el vetor X e Y de la DCM
    x_est = dcmEst(2,:) * error;
    y_est = dcmEst(1,:) * error;
    dcmEst(1,:) = dcmEst(1,:) + x_est;
    dcmEst(2,:) = dcmEst(2,:) + y_est;
    %Para obtener Z orthogonal, hacemos producto vectorial entre X e Y
    dcmEst(3,:) = cross(dcmEst(1,:), dcmEst(2,:));
    %Ahora hay que renormalizar cada vector fila de la DCM
    dcmEst(1,:)=0.5*(3-dot(dcmEst(1,:),dcmEst(1,:))) * dcmEst(1,:);
    dcmEst(2,:)=0.5*(3-dot(dcmEst(2,:),dcmEst(2,:))) * dcmEst(2,:);
    dcmEst(3,:)=0.5*(3-dot(dcmEst(3,:),dcmEst(3,:))) * dcmEst(3,:);

    graf(n,1)=n*imu_interval;
    graf(n,2)=atan2(dcmEst(3,2),dcmEst(3,3)); %guinada
    graf(n,3)=-asin(dcmEst(3,1)); %cabeceo
    graf(n,4)=atan2(dcmEst(2,1),dcmEst(1,1)); %banqueo
```

```

end
figure
hold on
plot(graf(:,1),graf(:,2)*(180/pi),'+b');%guinada
plot(graf(:,1),graf(:,3)*(180/pi),'r');%cabeceo
plot(graf(:,1),graf(:,4)*(180/pi),'g');%banqueo
grid

```

I put 1°/sec on X axis, 2°/sec on Y axis and 3°/sec on Z axis. I run the code for 10 second. I expect that final Euler angles will be 10°, 20°, 30°. Righth?

But It's not what I have:

<http://imageshack.com/a/img594/2481/8x1y.jpg>

My DCM matrix at the end of the test is:

```

0.8090  -0.4578  0.3688
0.5166   0.8530 -0.0744
-0.2805  0.2507  0.9265

```

So I want to extract euler angles for to chek that it is correct:

$-\text{asin}(\text{dcmEst}(3,1)) * 180/\pi = 16,3^\circ$

$\text{atan2}(\text{dcmEst}(3,2), \text{dcmEst}(3,3)) * 180/\pi = 15,12^\circ$

$\text{atan2}(\text{dcmEst}(2,1), \text{dcmEst}(1,1)) * 180/\pi = 32,56^\circ$

What do I do wrong?

84. starlino | January 10, 2014

Lenoardo:

"I put 1°/sec on X axis, 2°/sec on Y axis and 3°/sec on Z axis. I run the code for 10 second. I expect that final Euler angles will be 10°, 20°, 30°. Righth?"

Wrong, Euler angles must be applied in a specific sequence not at the same time, so you will need to rotate 10 seconds about one axis , 10 seconds about another on and finally 10 seconds about the third one. Depending on your convention I believe it must be in this order :

1. Rotate the body about its zb axis through the yaw angle ψ
2. Rotate the body about its yb axis through the pitch angle θ
3. Rotate the body about its xb axis through the roll angle ϕ

I am personally not a big fan of Euler angles, this being the main reason. In this notation – the axis angles are not “equivalent”, they must be applied in specific order. It has regions of high sensibility around the poles. The equivalent would be the Longitude/Latitude measurments of the globe, if you're in north pole or south pole area , you can change longitudes by simply making one step, wheras if you're on equator you have to travel thosands of miles for same longitude change....

85. Leonardo Garberoglio | January 11, 2014

Starlino, once again, thk for your answer. I understand now.

So, I need to finde a GUI to show a dcm parametres and rotation graphically.

regards

86. starlino | January 12, 2014

I have written a little utility in Processing for that, see:

https://code.google.com/p/picquadcontroller/source/browse/#svn%2Ftrunk%2Fprocessing%2FPICQUADCONTROLLER_DEBUG1

87. Leonardo Garberoglio | January 14, 2014

Well, is there something that you have not done?

I'll test it!!! by the way, it expect 9 float from uC, comma separated? I mean i need to send "DCM[0][0], DCM[0][1]," and so on?

Thank and best regards!

88. starlino | January 14, 2014

Leo,

It actually expects at least 13 comma separated values per line.

interval, accX,accY,accZ, DCM ... (9 values)

You can find sample output here:

https://code.google.com/p/picquadcontroller/source/browse/trunk/processing/PICQUADCONTROLLER_DEBUG1/PICQUADCONTROLLER_DEBUG1.pde

```

/*
PICQUADCONTROLLER_DEBUG1.pde
//Output for:
(double)imu_interval_ms);
printf("%.2f, ",
(float*)kacc);
printf(" ");
print_float_list(3,
(float*)dcmEst);
printf(" ",%.2f,%.2f\n",
(double)pitch.value,(double)roll.value);
*/

```

And here is where it reads the values in processing:

```

if(s != null){
    float a[] = parse_numbers(s);
    if(a.length >= 4 + 3*3)
        for(int i=0;i< 3;i++){
            acc[i] = a[1+i];
            for(int j=0;j< 3;j++)
                dcm[i][j] = a[4 + i*3 + j];
        }
    }
}

```

As you can see the DCM is starting with 4th value ...

89. Peter | January 14, 2014

Has anybody compiled the <https://code.google.com/p/picquadcontroller/source/browse/trunk/imu.h> code using the new XC16 V1.2 compiler from microchip.

When I run the software it ends up crashing with a address trap error (after sometime 5-6 seconds), I have traced it to the imu.h code

Address Error 0x220

9223372036854775808.7758081.3 9223372036854775808.7758081.3 9223372036854775808.7758081.3

9223372036854775808.7758081.3 9223372036854775808.7758081.3 9223372036854775808.7758081.3
 9223372036854775808.7758081.3 9223372036854775808.7758081.3 9223372036854775808.7758081.3

the above is what happens the nine numbers are the DCM matrix at the time of the crash

Has anybody had any experience of this problem?

90. Leonardo Garberoglio | January 15, 2014

Starlino, thk a lot. I am re writing my code to try out your processin sketch.

But firs of all I need to figure out wy my processing seems to don't work. I have an error: "the funciont readStringUntil(int) does not exist". If I comment this line to test the resto of the sketch i have other error:

```
java.lang.NullPointerException
  at processing.mode.java.runner.Runner.findException(Runner.java:926)
  at processing.mode.java.runner.Runner.reportException(Runner.java:871)
  at processing.mode.java.runner.Runner.exceptionEvent(Runner.java:797)
  at processing.mode.java.runner.Runner$2.run(Runner.java:686)
```

But my be somthing from instalation because i can't run some examples sketch that came with proccessing....

I need to unzip zip file on C drive? or i need to install something else? i already have java runtime install...

Thk, and best regards!

91. Leonardo Garberoglio | January 15, 2014

Starlino, the problem is on size(500, 500, P3D);

I can't use P3D. I don't know if ther is something missing on my processing instalation....

Regards!

92. Tuan | February 17, 2014

Hi starlino, after reading this tutorial, i think this is great tut. And i have a question relate theoretically, i hope you will explain for me :D

In part1, I see that DCM is matrix of cosin

```
ex :      cos(I,i)  cos(J,i)  cos(K,i)
DCMb =    cos(I,j)  cos(J,j)  cos(K,j)
          cos(I,k)  cos(J,k)  cos(K,k)
```

But in part4, I see DCM is values from Accelerometer and magnetometer

```
DCMb = [Ib, Jb, Kb]  -> Kb from Accelerometer
          -> Ib from magnetometer
          -> Jb = Ib x Kb
```

So, my question is : DCM in part 1 is the same DCM in part 4, isn't it? Because, i think value from Accelerometer and magnetometer will difference values of cosin in part 1. Please clarify for me this confusion!

Thanks and Best Regards!

93. vikrant p. more | March 1, 2014

Hello Starlino,

Why to use DCM if we directly caculate Euler angles using IMU data and atan2() function ? What I did is I calculate roll and pitch angle using accelerometer with atan2() and using gyroscope I did data fusion by using complemetary filter for removing noise from accelerometer reading. Similarly using Magnetometer and gyro I calculate yaw angle. Now I have yaw, pitch, roll filtered angle which are less effected by circuit board vibration. So what is point in using DCM ? Because in matrix caculation there are many multiplications and divisions which reduces the microcontroller performance ?

Thanks in advanced.

94. vikrant | March 1, 2014

Why to use DCM algorithm if we easily calculate Euler angle using atan2() function and do data fusion and filtering using complementary filter?? For eg. acc. and gyro data fusion gives filtered pitch and roll angles and magnetometer and gyro gives filtered yaw angles. So why to complicate the things and load microcontroller for unnecessary matrix multiplication ?

95. starlino | March 1, 2014

vikrant: If you're using Euler angles, yaw angle becomes undetermined or very sensitive when your pitch is close to 90 degrees. DCM offer a measurement that is simmetric in regards to all axis (all axis are treated the same).

96. vikrant | March 2, 2014

Yes you are right starlino but we can limit our pitch angle in Quadcopter program that means for eg. don't go beyond +/-50 degree. Hmm...but in that case if PID parameter are not properly set then and then only in unstable state Quadcopter reach at 90 degree pitch angle. So please give me your point of view on this if we limit our pitch angle and properly set PID parameter even after that is DCM required ?? And is DCM also consider linear acceleration effect on accelerometer and update itself accordingly ? Thanks for your fast response.

97. vikrant | March 3, 2014

Hello Starlino, If I program the Quadcopter such a way that it always stay in limit of +/- 50 degree pitch angle then also is DCM required (by assuming PID parameters are properly set) ?? And is DCM also consider linear acceleration effect ? Thank you for your fast response !!

98. [Mikael Tulldahl](#) | October 24, 2014

maybe it has been said in an earlier comment, but I just wanted to add that it's not very good to assume that the magnetometer will give you the North direction as it is used in this article. the vector you get from the magnetometer is going to point slightly into or out of the ground and is not horizontal. it depends on where you are in the world.

99. Sajad | February 8, 2015

Hi

I need a MATLAB code for 9 DOF IMU to extract DCM matrix using raw data of magnet, gyro and accelerometer. I know there are some method to obtain this matrix, but I need a appoche to calculate this EXCEPT PI CONTROLLER method.

Thanks

100. pavel | April 1, 2015

can you help me in this problem ?

How to convert IMU output data to quaternion?
I'am really confused with that problem. . .

thanks a lot Starlino :-)

101. starlino | April 1, 2015

Pavel: <http://www.euclideanspace.com/maths/geometry/rotations/conversions/matrixToQuaternion/index.htm>

102. Michal | December 7, 2015

It's great tutorial. Your method is very smart.

Thanks a lot

103. [Matrix Direct Reviews](#) | January 26, 2016

[...] DCM Tutorial – An Introduction to ... – DCM Tutorial – An Introduction to Orientation Kinematics – Introduction This article is a continuation of my IMU Guide, covering additional orientation ... [...]

104. [B.b.a Degree](#) | March 2, 2016

[...] DCM Tutorial – An Introduction to ... – DCM Tutorial – An Introduction to Orientation Kinematics – Introduction This article is a continuation of my IMU Guide, covering additional orientation ... [...]

105. Gary Wade | April 6, 2016

Great tutorial. What if I do a big angle roation? How should I write the DCM then.

Thx,

106. starlino | April 6, 2016

Unlike Euler Angles, DCM do not suffer from any anomalies at large angles close to 180 degrees. So you can use them the same way with large and small angles of rotation.

If you're refering to the fusion algorithm in this article, then it assumes that rotation from one interval to another is small. If you're sampling at higher intervals you're probably missing a lot of Gyro data so there's little point to use this algorithm at all (you could still use it with just the magnetomer and accelerometer weights).

107. george | July 11, 2016

Hello Starduino

In section four of the tutorial the DCM is construced using magnetometer and accelerometer components for two of the DCM colum vectors $K=-A$, $I=M$, and their cross poduct for the J column. This would be fine if A and M were orthogonal but they are not. The M vector dips towards the A vector by about 60 degrees. The second column (J) of the DCM should be $M \times A$ and the first (I) column $(M \times A) \times A$ this way all columns are orthogonal, normalization would of course follow. Or am I missing any assumptions?

108. starlino | July 11, 2016

This has been previously asked, but anyways to obtain corrected magnetometer vector that is perpendicular to A:

1) Get the “West/East” vector as $W = A \times M$

2) Now get $N = W \times A$, N will be corrected magnetometer vector and A,W,N are mutually perpendicular and form an orthogonal coordinate system (if normalized to modulus 1)

Some food for thought : by doing so you’re losing an important piece of information the angle between A and M, when A is noisy this information could be used to estimate true position of gravitational vector G (normally coinciding with A, when device is not moving much).

109. george | July 11, 2016

I agree with the above but that is not what the Tut says it says:

DCM=[M -AxM -A]

I'm looking at the PDF version btw.

110. starlino | July 11, 2016

This tutorial does not go into the topic of correcting or calibrating M, so it is assumed $M = N$ (corrected magnetometer output). Maybe one day I will write a tutorial about correcting/calibrating Mag and Acc devices, this is an extended topic.

111. george | July 11, 2016

Regarding the DCM updates using averaged angle increments, how do you initialise the DCM or is the process self correcting if the initial DCM was in error?

112. starlino | July 11, 2016

You can init DCM with unity matrix, it will self-correct on startup.

113. George | July 12, 2016

I'm not talking about calibration (offsets, gain, hard, soft ...) I'm on about the basic non-orthogonality between M and A which the tutorial uses as column vectors in the DCM.

114. Jimson | September 4, 2016

Hello, dear Starlino.

I am reading the W.P DCM theory paper, and I saw your help website and file. I must say thank you and I have a problem now: how to make the equation 17, Equation 17 is a recipe for updating the direction cosine matrix from gyro signals.

please help me, thanks!

115. Bibihexium | November 8, 2016

I see a blind spot in using the accelerometer to get $d\mathbf{w}_a$. If accelerometers read $\mathbf{KB0} = \{0,0,1\}$ and $\mathbf{w}_g = \{0,0,e\}$ then

$\mathbf{w}_a = \{0,0,0\}$ since accelerometers still read $\mathbf{KB1A} = \{0,0,1\}$. Using weighted average just reduces e by a fraction.

Am I confused?

116. starlino | November 9, 2016

Bibihexium – this is a good question. Yes accelerometer is “blind” to rotations around its Z (axis perpendicular to ground in Earth coordinate system). In practice rarely an object will rotate just around that axis, so this tends to average out. However if you want to improve on this you can only use the complimentary filter on the X, Y components of \mathbf{w}_a (assuming \mathbf{w}_a is in earth’s coordinate system). This also is somehow alleviated if you bring \mathbf{w}_m (from magnetometer) into equation.

As a reminder you can switch between earth/body coordinate system by using the DCM matrix (from previous loop as a good estimate).

117. Lavender | February 15, 2017

Hi Starlino,

If I am understanding correctly, at the very top of this tutorial, the subscript of $\mathbf{i}_G = \{i_xG, i_yG, i_zG\}$ T should be $\mathbf{i}_G = \{i_XG, i_YG, i_ZG\}$ T. Therefore, $i_XG = \mathbf{I.i}$, $i_YG = \mathbf{J.i}$, $i_ZG = \mathbf{K.i}$.

118. starlino | February 15, 2017

I am not sure if uppercase of subscript is necessary since the G superscript already implies the coordinates are in the global/ earth frame. I guess you can use uppercase if you want.

119. [PES4 | Andreas' Blog](#) | March 8, 2017

[...] Direction Cosine Matrix IMU: Theory A free and open implementation of DCM based on FreeIMU DCM Tutorial – An Introduction to Orientation Kinematics Understanding the various attitude estimation [...]

120. Jayson | March 10, 2017

shouldn't $W_a = k_B \times V_a / k_B^2$ be $W_a = k_B \times V_a / (k_B^2 \sin(x))$

Where x is the angle between k_B and W_a ??

121. starlino | March 10, 2017

Jayson, I don't think so since this results directly from eq. 2.5, but can you explain the reason behind your logic am I missing something ?

122. Jayson | March 10, 2017

I might be wrong as well, but I will give you my derivation.

"The Magnitude of V_a " = $|V_a| = |W_a| |k_B| \sin(x)$ *again x is the angle between k_B and W_a

solving for $|W_a|$: $|W_a| = |V_a| / |k_B| \sin(x)$

" W_a has direction" = $k_B \times V_a$

"The Magnitude of $k_B \times V_a$ " = $|k_B| |V_a| \sin(\pi/2) = |k_B| |V_a|$ * $\pi/2$ because k_B and V_a are orthogonal

therefore written as a unit vector

" W_a has direction" = $(1 / |k_B| |V_a|) k_B \times V_a$

Now we multiply this last result by $|W_a|$ (the magnitude of W_a) this gives us the magnitude and direction of W_a

$W_a = [|V_a| / (|k_B| \sin(x))] * 1 / (|k_B| |V_a|) k_B \times V_a = 1 / (k_B^2 \sin(x)) k_B \times V_a$

Thanks for the response.

123. starlino | March 10, 2017

Aren't k_B and W_a also orthogonal, thus $\sin(x) = 1$? , where x is the angle between k_B & W_a ?

124. Jayson | March 10, 2017

I can't see why they would always be orthogonal. Why do you think they are?

125. starlino | March 10, 2017

Jason , you said it yourself " " W_a has direction" = $k_B \times V_a$, thus W_a is orthogonal to the plane formed by k_B and V_a , and thus to each individual vector , see https://en.wikipedia.org/wiki/Cross_product

126. Jayson | March 10, 2017

Yeah, I'm aware that when you take the cross product of two vectors the result is orthogonal too the original two vectors. I think I'm wrong in asserting that $k_B \times V_a = W_a \times \text{constant}$

But k_B cannot be always orthogonal too W_a . For example, what if we sit an object flat on the ground and then rotate it about the Z axis. k_B will be in the Z direction and so will W_a

127. Jayson | March 10, 2017

I'm obviously missing something. I need to read your paper more closely.
Sorry for wasting your time.

128. Jayson | March 11, 2017

$$R = \begin{bmatrix} 0 & r_3 & -r_2 \\ -r_3 & 0 & r_1 \\ r_2 & -r_1 & 0 \end{bmatrix}, \quad \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = W$$

If R is a 3×3 matrix and w is a 3×1 matrix $R \cdot W = R \times W$ *where "." represents matrix multiplication

$R \cdot W = V$

Therefore

$$R^{-1} \cdot V = W$$

but R is singular and therefore not invertible. So this can't be solved

Check equation 2.4 with an example and you will see that it is wrong.

129. starlino | March 11, 2017

Jayson: r is a vector in equation 2.4, anyways if you want to arrive differently to 2.5 see https://en.wikipedia.org/wiki/Angular_velocity see "Particle in three dimensions" section.

130. Jayson | March 11, 2017

$$w = (1,1,1)$$

$$r = (1,2,3)$$

$$w \times r = v = (1,-2,1)$$

$$v \times r = (8,2,-4) \quad \text{*which is not in the same direction as } w$$

131. Jayson | March 11, 2017

R is a vector. yes.

But the cross product can be written as a matrix equation.

132. Jayson | March 12, 2017

regarding the link you posted:

https://en.wikipedia.org/wiki/Talk:Angular_velocity

Scroll down to non-circular motion.

The Matrix proof above is enough however.

Thanks for the convo.

133. starlino | March 13, 2017

Jayson, w , r , v must be mutually orthogonal

In your example $w = (1,1,1)$ $r = (1,2,3)$ are not orthogonal because their dot product is not zero:

<https://www.wolframalpha.com/input/?i=Dot+%5B%7B1,1,1%7D+,%7B1,2,3%5D>

134. Jayson | March 13, 2017

You do know that w doesn't have to be orthogonal to r right??

Or is there a particular reason you're insisting upon this?

The formula you provided is for a unique situation where r is orthogonal to w . This is not the general case. Your formula will not work in general. It will only work in the special case where w is orthogonal to r .

135. Jayson | March 13, 2017

picture a scenario where r lies on the z axis. Now we rotate about the z axis and the x axis simultaneously. w is no longer perpendicular to r and your formula will no longer work.

136. starlino | March 13, 2017

Jayson, Imagine w is divided into two components w_0 (orthogonal to r) and w_p (parallel to r), thus

$w \times r$ becomes $(w_0 + w_p) \times r = w_0 \times r + w_p \times r$, but $w_p \times r$ is 0 since $w_p \parallel r$ thus you end up with $w_0 \times r$ where w_0 is orthogonal with r .

so the rotation that the object is subject to does not have to be orthogonal to r , but we will only sense the component that is orthogonal to the Z axis with an accelerometer, you can improve somehow the algorithm by decomposing the gyro reading into two components as described above and using complimentary filter only on the component that is orthogonal to r .

I am aware of the situation you described. In this case the accelerometer's weight of the reading will act as a low-pass filter rather than giving a more exact reading of orientation. In practice this works out ok, especially when a magnetometer is used as well, things tend to average out.

137. Jayson | March 14, 2017

Yeah.. That doesn't work. Just look at the example. You can't have two different answers to the same problem just because you wrote the problem differently.

Try to figure out the matrix proof... It's all that is needed.

$$R = \begin{bmatrix} 0 & r_3 & -r_2 \\ -r_3 & 0 & r_1 \\ r_2 & -r_1 & 0 \end{bmatrix}, \quad \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = W$$

If R is a 3×3 matrix and w is a 3×1 matrix $R \cdot W = R \times W$ *where \cdot represents matrix multiplication

$$R \cdot W = V$$

Therefore

$$R^{-1} \cdot V = W$$

but R is singular and therefore not invertible. So this can't be solved

138. Jayson | March 14, 2017

Sorry, for the second message.

I did want to mention, that in the written scenario I gave where r lied on the z axis and we rotated around z and x simultaneously, can be expanded as you suggested above. That was a bad example.....

However that again will not work in general. For example try doing that for the scenario above where $w=[1,1,1]$ and $r=[1,2,3]$.

Sorry for spamming. I just felt I had to clear this up.

Thanks again for the convo.

139. Jayson | March 14, 2017

sorry again.

I was hasty in my last post. Your method won't work in either example.

Again sorry for spamming

140. G Evans | October 20, 2017

Hello

DCM Tutorial paper

I made this observation previously without a reply regarding the formation of the DCM using $K^A B = -A$ and $I^A B = M$ using acc and mag measurements. These two vectors are not orthogonal hence can not be used in the DCM. However $J = K \times I$ is legitimate being normal to K & I.

The proper DCM would be: $[(A \times M) \times A \quad A \times M \quad A]$ where A and M is acc and mag measurements respectively .

LEAVE A COMMENT

Name

E-Mail *(not published)*

Website *(optional)*

Comment

Captcha *



Type the text displayed above:

Get Updates of this post/comments by email

2016

\$ 532

2017

\$13448

2018

\$??????

2018 Top Bitcoin Brokers

Buy E

Credi

Insta

• Search

Search for:

• Categories

- [Featured](#) (7)
- [IMU Theory and Experiments](#) (7)

- [Motion Sensing USB Devices](#) (3)
- [USBThumb](#) (3)
- [Robotics](#) (4)
- [Multirotors](#) (4)
- [Soldering and DIY Fabrication](#) (5)
- [Benchmarks and Reviews](#) (9)
- [Tricks and Tips](#) (14)
- [Fun Projects](#) (5)
- [News and Discussions](#) (11)

• Sponsors

Get a NEW Plug &
Play Inertial
Measurement
Unit (IMU)

& take your projects
to the next level!

20
% OFF

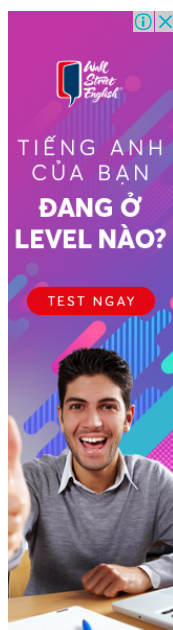
When you buy

VMU931
+
VMU Reader

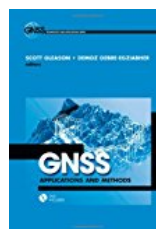
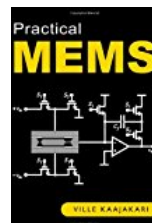
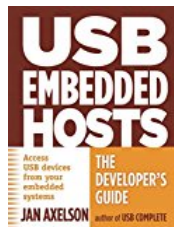


Take 20% OFF with
COUPON Code:
STARLINO20

www.variense.com



- **Books**



- **Subscribe**

Here are few ways to stay updated with my projects:

[Feedburner](#)

(sends you an email when I have a new post)

[Twitter](#)

(short messages and updates to you computer or phone)

[RSS Feed \(posts\)](#) (RSS feed for your favorite RSS reader)

[RSS Feed \(comments\)](#) (RSS feed for your favorite RSS reader)

- **Tags**

[accelerometer](#) [Acc_Gyro](#) [arduino](#) [arduino microchip](#) [ti avr](#) [business c](#) [computer](#) [desoldering](#) [driver](#) [electronics](#) [filter](#) [firmware](#) [fun](#) [game](#) [gyroscope](#) [h-bridge](#)
[howto](#) [i2c](#) [imu](#) [magenteometer](#) [motion](#) [Motion Gamedpad](#) [motor](#) [mouse](#) [op-amp](#) [parts](#) [pcb](#) [pic](#) [programmer](#) [Projects](#) [propeller](#) [quadcopter](#) [rc](#) [robot](#) [sensor](#) [serial](#) [smd](#)
[smt](#) [soldering](#) [spi](#) [tool](#) [transmitter](#) [usb](#) [UsbThumb](#)

• Categories

- [Featured](#)
- [IMU Theory and Experiments](#)
- [Motion Sensing USB Devices](#)
- [USBThumb](#)
- [Robotics](#)
- [Multirotors](#)
- [Soldering and DIY Fabrication](#)
- [Benchmarks and Reviews](#)
- [Tricks and Tips](#)
- [Fun Projects](#)
- [News and Discussions](#)

• Recent Comments

- [Mikhail](#) on [How much power is needed to hover ?](#)
- [Blake Slade](#) on [Tenma 72-7740 Autorange Multimeter Review](#)
- Michael on [How much power is needed to hover ?](#)
- joe on [A Guide To using IMU \(Accelerometer and Gyroscope Devices\) in Embedded Applications.](#)
- Hicham on [A Guide To using IMU \(Accelerometer and Gyroscope Devices\) in Embedded Applications.](#)

• Links

- [Adafruit](#)
- [EE Web](#)
- [Hackaday](#)
- [Hackedgadgets](#)
- [VoltLog](#)

© Starlino Electronics. Please mention and link to source if your use content from this site.