TI DSP,MCU 및 Xilinux Zynq FPGA

프로그래밍 전문가 과정

이름	문지희
학생 이메일	mjh8127@naver.com
날짜	2018/5/16
수업일수	55 일차
담당강사	Innova Lee(이상훈)
강사 이메일	gcccompil3r@gmail.com

목차

벡터

- 벡터의 덧셈, 뺄셈
- 벡터의 곱셈

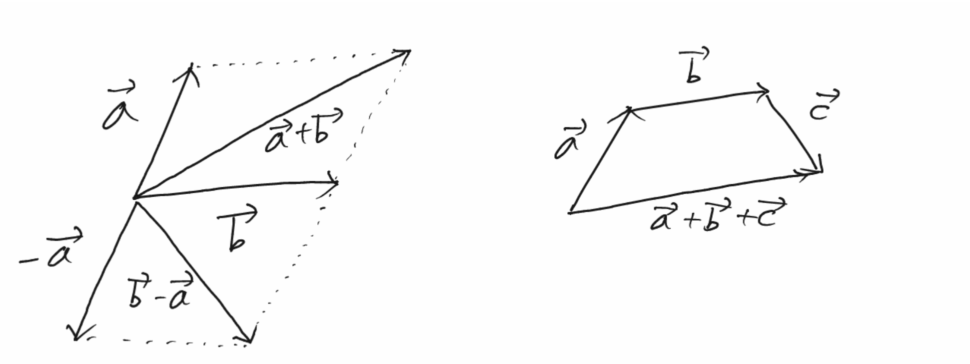
벡터의 연산 코드화하기

벡터

스칼라 : 크기만을 나타내는 물리량.

벡터 : 크기와 방향을 나타내는 물리량

- 벡터의 덧셈, 뺄셈



$\underset{a}{\rightarrow} + \underset{b}{\rightarrow}$

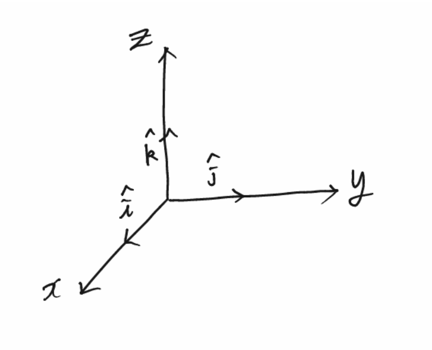
$\underset{a}{\rightarrow} - \underset{b}{\rightarrow}$

$\underset{a}{\rightarrow} + \underset{b}{\rightarrow} + \underset{c}{\rightarrow}$

벡터의 -는 반대 방향을 의미

벡터 a(3,2) , b(1,1)일 때 $\underset{a}{\rightarrow} + \underset{b}{\rightarrow}$ = (4, 3)

단위벡터,기저 : 크기가 1인 벡터



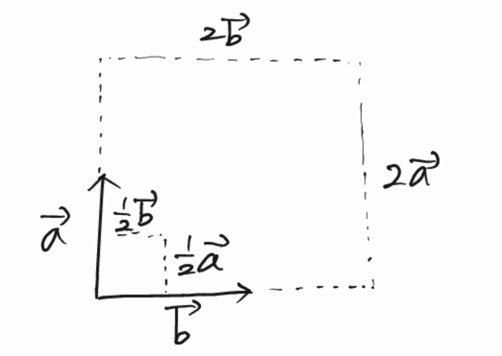
x축의 기저는 $\hat{\imath}$

y축의 기저는 $\hat{\jmath}$

z축의 기저는 $\hat{k}$

- 벡터의 곱셈

1. 스칼라 곱



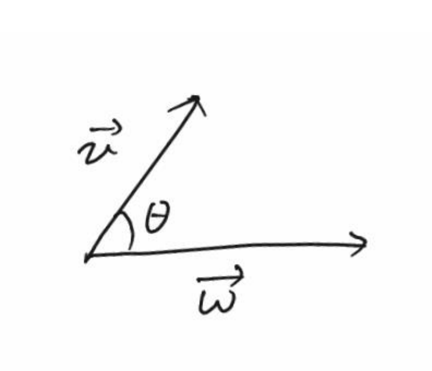
벡터에 대해 배수를 취함

2. 내적

: 내적의 결과는 스칼라이다. 방향이 없음.

내적의 표기

$\vec{v} \cdot \vec{w}$ == $\langle \vec{v}, \vec{w} \rangle$



$$\vec{v} = (v_x + v_y + v_z) \quad , \quad \vec{w} = (w_x + w_y + w_z)$$

$$\vec{v} \cdot \vec{w} = v_x w_x + v_y w_y + v_z w_z$$

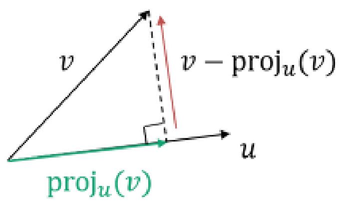
$$= \left\| \vec{v} \right\| \left\| \vec{w} \right\| \cos\theta$$

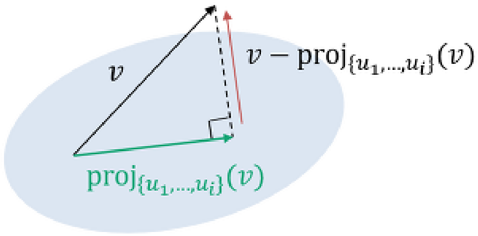
$$= \frac{\langle \vec{v}, \vec{w} \rangle}{\|w\|^2} \vec{w}$$

내적을 90도인지 아닌지 파악할 때 사용할 수 있음. 내적의 결과 값이 0이면 직교

orthogonal projection(정사영)

$$\left|\vec{w}\right|\cos\theta = proj_{\vec{w}}\vec{v} = \frac{\left|\vec{w}\right|\left|\vec{v}\right|\cos\theta}{\left|\vec{w}\right|^2}\vec{w} = \frac{\vec{v}\cdot\vec{w}}{\left|\vec{w}\right|^2}\vec{w}$$ 라고 표현





그람-슈미트 정규 직교화

주어진 벡터들을 이용해서 서로 수직인 벡터들을 만드는 방법이다 다른 말로 표현하면 주어진 벡터들에 대한 직교기저(orthogonal basis) 또는 정규직교기저(orthonormal basis)를 구하는 과정이다.

그람-슈미트 직교화(Gram-Schmidt orthogonalization): 주어진 벡터 v1, v2, ... 로부터 이 벡터들을 생성할 수 있는 직교기저(orthogonal basis)를 구하는 과정
그람-슈미트 정규직교화(Gram-Schmidt orthonormalization): 주어진 벡터 v1, v2, ... 로부터 이 벡터들을 생성할 수 있는 정규직교기저(orthonormal basis)를 구하는 과정

$$\begin{aligned}
\mathbf{u}_1 &= \mathbf{v}_1, \\
\mathbf{u}_2 &= \mathbf{v}_2 - \mathrm{proj}_{u_1}(\mathbf{v}_2), \\
\mathbf{u}_3 &= \mathbf{v}_3 - \mathrm{proj}_{u_1}(\mathbf{v}_3) - \mathrm{proj}_{u_2}(\mathbf{v}_3), \\
&\vdots \\
\mathbf{u}_k &= \mathbf{v}_k - \sum_{i=1}^{k-1}\mathrm{proj}_{u_i}(\mathbf{v}_k)
\end{aligned}$$

$$\begin{aligned}
\mathbf{e}_1 &= \mathbf{u}_1/\|\mathbf{u}_1\|, \\
\mathbf{e}_2 &= \mathbf{u}_2/\|\mathbf{u}_2\|, \\
&\vdots \\
\mathbf{e}_k &= \mathbf{u}_k/\|\mathbf{u}_k\|
\end{aligned}$$

3. 외적

: 3차원 상에서만 정의가 가능하다.

*외적의 성질

$\hat{\imath} \times \hat{\jmath} = \hat{k}$

$\hat{\jmath} \times \mathrm{i} = -\hat{k}$

$\hat{\jmath} \times \hat{k} = \hat{\imath}$

$\hat{k} \times \hat{\jmath} = -\hat{\imath}$

$\hat{k} \times \hat{\imath} = \hat{\jmath}$

$\hat{\imath} \times \hat{k} = -\hat{\jmath}$

외적 공식

$$\underset{a}{\rightarrow} \times \underset{b}{\rightarrow} \Rightarrow \begin{vmatrix} \hat{\imath} & \hat{\jmath} & \hat{k} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix} = \left(a_y b_{z} - a_z b_y\right)\hat{\imath} - (a_x b_z - a_z b_x)\hat{\jmath} + (a_x b_y - a_y b_x)\hat{k}$$

벡터의 연산 코드화 하기

-vector_3d.c

```
#include
"vector_3d.h"
        #include <stdio.h>


        int main(void)
        {
                vec3 A = {3, 2, 1};
                vec3 B = {1, 1, 1};
                vec3 X = {1, 0, 0};
                vec3 Y = {0, 1, 0};
                vec3 v[3] = {{0, 4, 0}, {2, 2, 1}, {1, 1, 1}};
                vec3 w[3] = {};
                vec3 R = {0, 0, 0,
                         vec3_add, vec3_sub, vec3_scale,
                         vec3_dot, vec3_cross, print_vec3,
                         gramschmidt_normalization};


                R.add(A, B, &R);
                R.print(R);


                R.sub(A, B, &R);
                R.print(R);


                R.scale(3, R, &R);
                R.print(R);
```

```
        printf("A dot B = %f₩n", R.dot(A, B));


        R.cross(X, Y, &R);
        R.print(R);


        R.gramschmidt(v, w, R);


        return 0;
}
```

-vector_3d.h

```
#ifndef
__VECTOR_3D_H__
        #define __VECTOR_3D_H__


        #include <stdio.h>
        #include <math.h>


        typedef struct vector3d vec3;


        struct vector3d
        {
                float x;
                float y;
                float z;


                void (* add)(vec3, vec3, vec3 *);
                void (* sub)(vec3, vec3, vec3 *);
                void (* scale)(float, vec3, vec3 *);
```

```
        float (* dot)(vec3, vec3);
        void (* cross)(vec3, vec3, vec3 *);
        void (* print)(vec3);


        void (* gramschmidt)(vec3 *, vec3 *, vec3);
};


void vec3_add(vec3 a, vec3 b, vec3 *r)
{
        r->x = a.x + b.x;
        r->y = a.y + b.y;
        r->z = a.z + b.z;
}


void vec3_sub(vec3 a, vec3 b, vec3 *r)
{
        r->x = a.x - b.x;
        r->y = a.y - b.y;
        r->z = a.z - b.z;
}


void vec3_scale(float factor, vec3 a, vec3 *r)
{
        r->x = a.x * factor;
        r->y = a.y * factor;
        r->z = a.z * factor;
}


float vec3_dot(vec3 a, vec3 b)
{
```

```
        return a.x * b.x + a.y * b.y + a.z * b.z;
}


void vec3_cross(vec3 a, vec3 b, vec3 *r)
{
        r->x = a.y * b.z - a.z * b.y;
        r->y = a.z * b.x - a.x * b.z;
        r->z = a.x * b.y - a.y * b.x;
}


void print_vec3(vec3 r)
{
        printf("x = %f, y = %f, z = %f₩n", r.x, r.y, r.z);
}


float magnitude(vec3 v)
{
        return sqrt(v.x * v.x + v.y * v.y + v.z * v.z);
}


void gramschmidt_normalization(vec3 *arr, vec3 *res, vec3 r)
{
        vec3 scale1 = {};
        float dot1, mag1;


        mag1 = magnitude(arr[0]);
        r.scale(1.0 / mag1, arr[0], &res[0]);
        r.print(res[0]);
```

```
			mag1 = magnitude(res[0]);
			dot1 = r.dot(arr[1], res[0]);
			r.scale(dot1 * (1.0 / mag1), res[0], &scale1);
			r.sub(arr[1], scale1, &res[1]);
			r.print(res[1]);
		}


		#endif
```