TI DSP, MCU 및 Xilinx Zynq FPGA
프로그래밍 전문가 과정

강사 – Innova Lee(이상훈)
gcccompil3r@gmail.com
학생 – 하성용
accept0108@naver.com

55 일차

벡터

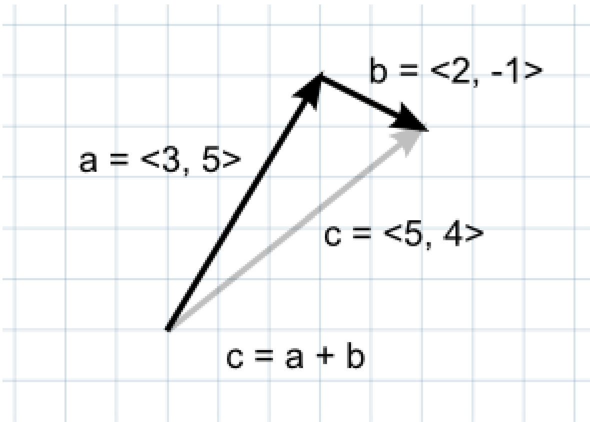
Vector : Vector 는 크기와 방향을 가진다
크기가 클 수록 화살표의 길이가 길어지고

크기와 방향만 있으면 벡터,
크기값만 나타내는게 스칼라

벡터의 연산은
v + w = w + v(교환법칙)
u + (v +w) = (u+v) +w(결합법칙)
v+0=v(덧셈의 항등원)
v+(-v)=0 (덧셈의 역원)

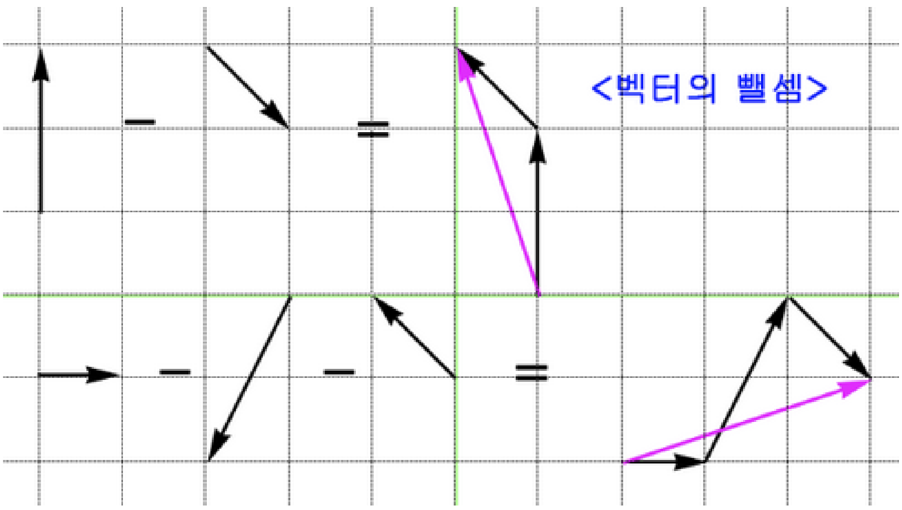
벡터의 덧셈(Addition)

벡터의 덧셈은 벡터를 하나씩 더 하는것과 같다. (더하는 순서는 상관없음)
벡터를 전부 더한 후에 시작점에서 최종점을 연결하면,
벡터의 합으로 구한 최종 벡터가 된다.

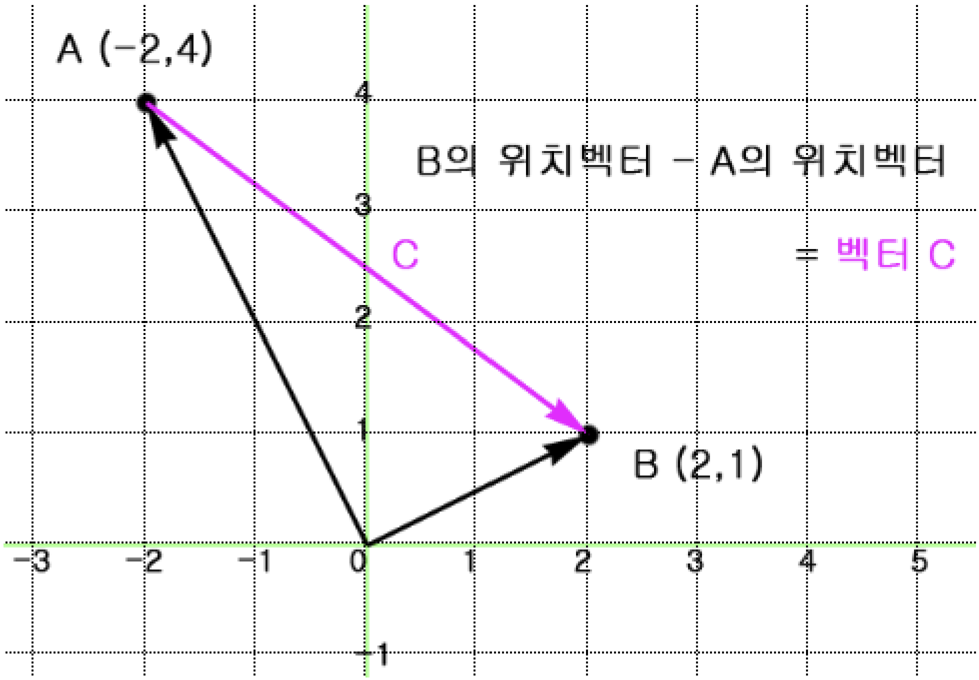


벡터의 뺄셈(Subtraction)

벡터의 뺄셈은 빼는 벡터의 **방향을 바꿔서** 더한것과 같다
벡터의 뺄셈은 두 점 사이의 거리를 구할 때 유용하게 사용된다



점 A(-2,4)와 점 B(2,1)가 있을 때, 두 점을 가리키는 위치벡터의 차를 구하면
두 점 사이의 벡터 C 가 나온다.



벡터의 곱셈
1. 스칼라곱
2. 내적
3. 외적
4. 텐서연산 //우주선 만드는거 아니면 쓰일일이없음, 엔진만든다면 필요한데 한국은 엔진기술없음
//공기역학, 외국계회사 항공기 엔진개발

스칼라곱이란 어떤 벡터에의해서 배수를 취하는것
같은 성분끼리 더하면 좌표가됨

평생사변형법을 적용
(2,1)에서(1,1) 만큼 이동하면 (3,2)
(1,1)에서 (2,1) 만큼 이동하면 (3,2)

//기저 시스템(i hat, j hat, k hat → hat 이 꼬깔모자(^)

2 를 단위벡터로 만드려면 나누기 2

(1,1)은?
대각선의 길이가 벡터의 크기
x 축 1, y 축 1
피타고라스정리로 √2
(1,1) => (√2 분에 1,√2 분에 1) → √2 분에 1 과 a->

y=sin x

기함수
주기적분은 언제나 0
원점기준 좌우비대칭 = 기함수

y=cos x

주기적분하면 0
원점기중 좌우대칭 = 우함수

내적의 기하학적의미

v 벡터와 w 벡터를 내적을하게되면
벡터 내적의 결과는 일단 스칼라, 즉 방향이없음
내적의 표기는 표기법이 여러가지가 있는데

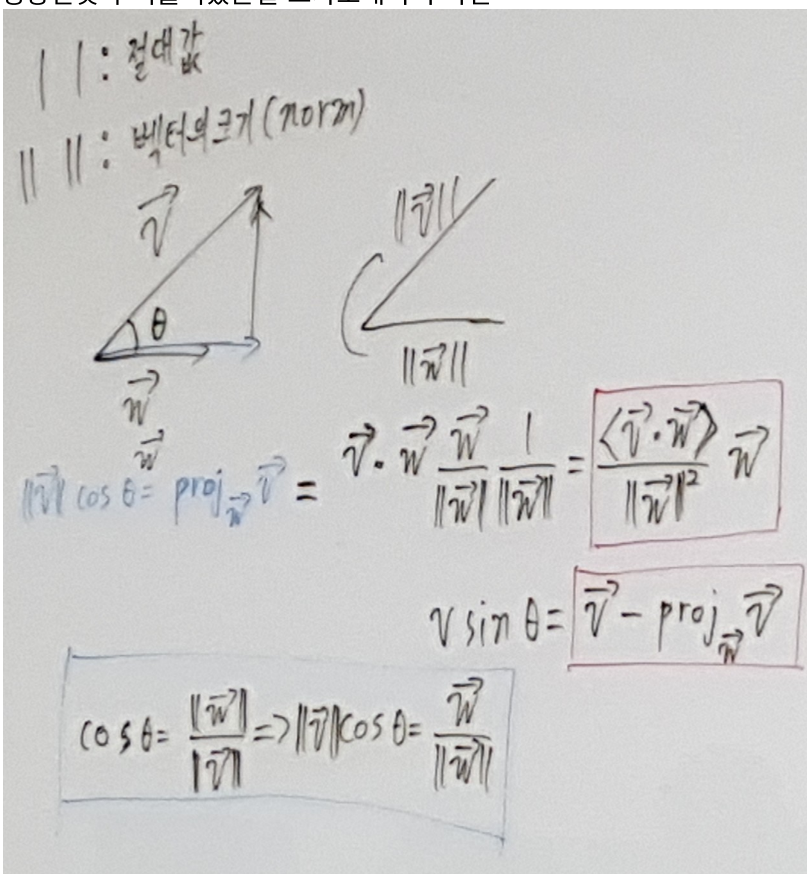
a→ . (내적사이에 쩜 오는것도 내적의표기법) b→
<a→,b→) |a→||b→| cos 0 (a 벡터와 b 벡터 절대값을 더하고 코사인세타?)

Orthogonal Projection

| 한개면 절대값
| 두개면 벡터의 크기 norm

Gram-Schmidt Orthogonalizetion
내적과 정규화의 조합으로 기저들을 정의
부동 소수점 오차 즉

가장연산하기쉬운 기저로 단위벡터를 만들고
코사인을 구하면 수직한값을 구해서
기준점이 된것과
정상인것과 비틀려있는걸 교차로해서 구하면



v0=(0,4,0)
v1=(2,2,1)
v2=(1,1,1)
꼬여있는걸 정상으로 돌리기

식안에는 내적이 들어있어서 정규화를 꼭하고 계산해야함

w0=(0,1,0) //새로운축 잡힘
w1→ =v1→ -porj→w0(→밑에) v1→
=(2,2,1) -2(0,1,0)
=(2,0,1) //2-0, 2-2,1-0

오메가 제로랑 V1 에 프로젝션
v 의 단위(변환?)이 오메가(w)

w2→ =v2→ - proj w0→ v2→ - proj w1→ v2→
=(1,1,1)-(0,1,0)-(5 분에 3(2,0,1) //제곱이니까 루트 5 가아니라 5
=(-5 분에 1, 0, 5 분에 2)
서로 정말 직교하는지 검토

내적(Inner Pro duct)
a→ · b→, <a→,b→ >
a→ =(ax, ay, az)
b→ =(bx, by, bz)
a→ · b→=ax+bx+ay+by+az+bz
=||a→ || ||b→|| cos0

= ||b||제곱 분에 <a→ ,b→>b→

외적(Outer Product)
3 차원 좌표계에서 정의 가능
a→*b→ => I/ j ^ k^
ax ay   az
bx by bz

//벡터 덧셈,뺄셈, 코드슈미트 코드

vector_3d.h

```
#ifndef __VECTOR_3D_H__
#define __VECTOR_3D_H__

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

		void (* add)(struct vector3d, struct vector3d, struct vector3d *);
		void (* sub)(struct vector3d, struct vector3d, struct vector3d *);
		void (* scale)(float, struct vector3d);
		void (* dot)(struct vector3d, struct vector3d);
		void (* cross)(struct vector3d, struct vector3d);
} 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;
}

#endif
```

vector3d.c

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

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

int main(void)
{
	vec3 A = {3, 2, 1};
	vec3 B = {1, 1, 1};
	vec3 R = {0, 0, 0, vec3_add, vec3_sub};

	R.add(A, B, &R);
	print_vec3(R);

	R.sub(A, B, &R);
	print_vec3(R);

	return 0;
}
```

```
x = 4.000000, y = 3.000000, z = 2.000000
x = 2.000000, y = 1.000000, z = 0.000000
```

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 / pow(mag1, 2.0)), res[0], &scale1);
	r.sub(arr[1], scale1, &res[1]);
```

```
		r.print(res[1]);
}

#endif
```