# Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 – Innova Lee(이상훈)

gcccompil3r@gmail.com

학생 – hoseong Lee(이호성)

hslee00001@naver.com

목차

- ✓ 행렬의 덧셈, 뺄셈, 곱셈, 스케일링
- ✓ 정방, 대각, 전치, 대칭행렬 (-> 칼만필터)
- ✓ 회전행렬(e^ix = cos x+ I sinx 와 같은 오일러 공식)
- ✓ Guass-jordan 소거법 ->연립방정식, 역행렬
- ✓ 행렬의 판별식
- ✓ 수반행렬
- ✓ 역행렬 구하기(정석)
- ✓ 크래머공식
- ✓ Determinant 칼만필터

Determinant –칼만필터

Guass-jordan 소거법 -> 역행렬구하기

쌤말씀-> 가우스 소거법은 로봇팔 제어를 수행하기 위해서도 모르면 안된다. 여기에 미분 방정식 형태로 식이 만들어지니 라플라스 변환이 필요하다.

(연립 미분방정식 해석엔 라플라스 변환과 행렬만한게 없다.)

C로 구현

```c
#include <stdio.h>
#include "matrix_op.h"

int main(void)
{

    mat3 A = {{{3,2,1},{5,4,3},{7,6,5}}};
    mat3 B = {{{1,2,3},{3,4,5},{5,6,7}}};
    mat3 R = {{{0,0,0},{0,0,0},{0,0,0}},matrix_determinant,matrix_printf,
              matrix_add,matrix_sub,matrix_scale,matrix_inverse};

    R.Determinant();

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

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

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

    R.inverse(A,&R);
    R.print(R);

    return 0;
}
```

```c
#ifndef __Matrix_
#define __Matrix_

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

typedef struct matric3 mat3;
struct matric3
{
    float x[3][3];

    void (* Determinant)(void);
    void (* print)(mat3);
    void (* add)(mat3, mat3, mat3 *);
    void (* sub)(mat3, mat3, mat3 *);
    void (* scale)(mat3, mat3, mat3 *);
    void (* inverse)(void);
    void (* Gaussian_elimination)(void);
    void (* Cramer_rule)(void);
    void (* Transpose)(void);

};

void matrix_printf(mat3 r)
{
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            printf("x[%d][%d] = %0.1f\n",i,j,r.x[i][j]);
        }
    }
    printf("\n");
}

void matrix_add(mat3 a, mat3 b, mat3 *r)
{
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            r->x[i][j] =  a.x[i][j] + b.x[i][j];
        }
    }
}
```

```c
void matrix_sub(mat3 a, mat3 b, mat3 *r)
{
    int i,j;
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
        {
            r->x[i][j] =  a.x[i][j] - b.x[i][j];
        }
    }
}

void matrix_scale(mat3 a, mat3 b, mat3 *r)
{
    r->x[0][0] = a.x[0][0]*b.x[0][0]+a.x[0][1]*b.x[1][0]+a.x[0][2]*b.x[2][0];
    r->x[0][1] = a.x[0][0]*b.x[0][1]+a.x[0][1]*b.x[1][1]+a.x[0][2]*b.x[2][1];
    r->x[0][2] = a.x[0][0]*b.x[0][2]+a.x[0][1]*b.x[1][2]+a.x[0][2]*b.x[2][2];

    r->x[1][0] = a.x[1][0]*b.x[0][0]+a.x[1][1]*b.x[1][0]+a.x[1][2]*b.x[2][0];
    r->x[1][1] = a.x[1][0]*b.x[0][1]+a.x[1][1]*b.x[1][1]+a.x[1][2]*b.x[2][1];
    r->x[1][2] = a.x[1][0]*b.x[0][2]+a.x[1][1]*b.x[1][2]+a.x[1][2]*b.x[2][2];

    r->x[2][0] = a.x[2][0]*b.x[0][0]+a.x[2][1]*b.x[1][0]+a.x[2][2]*b.x[2][0];
    r->x[2][1] = a.x[2][0]*b.x[0][1]+a.x[2][1]*b.x[1][1]+a.x[2][2]*b.x[2][1];
    r->x[2][2] = a.x[2][0]*b.x[0][2]+a.x[2][1]*b.x[1][2]+a.x[2][2]*b.x[2][2];
}

void matrix_inverse(mat3 a, mat3 *r)
{

}


#endif
```