로봇학 실험 4

ODE(Open Dynamics Engine)

민준서 로봇학부

KWANGWOON UNIVERSITY

Code – RobotExp_4.cpp

```cpp
#include "stdafx.h"
#include "RobotExp_4.h"
#include "RobotExp_4Dlg.h"
//page 3 include
#include "ThreadWorker.h"
#include "WorkBase.h"
#include "ODEWork.h"

#include "SystemMemory.h"
#include "DataType.h"

#include <thread>
#include <chrono>
#include "Comm.h"
```

```cpp
DataType_t jointData;
memset(&jointData, 0, sizeof(DataType_t));
CREATE_SYSTEM_MEMORY("JointData", DataType_t);
SET_SYSTEM_MEMORY("JointData", jointData);

CThreadedWorker cODEWorker;
cODEWorker.SetWork(CreateWork<CODEWork>("ODEWork"));

cODEWorker.StartWork();

CRobotExp_4Dlg dlg;
m_pMainWnd = &dlg;
INT_PTR nResponse = dlg.DoModal();

cODEWorker.StopWork();
```

코드에서 스레드 작동을 위한 구현부 내용으로는 work flow를 구성하고 해당 task를 수행하도록 실행함.

CRobotExp4App::InitInstance() 함수 내부에서 실행하도록 설정함. Thread 를 run 하는 부분에 해당.

Thread 로 작업을 할당하기 위해서 thread api를 include함.

Code –ODE.cpp

KWANGWOON UNIVERSITY

```cpp
#include "stdafx.h"
#include "ODE.h"
#include "SystemMemory.h"
#include "DataType.h"

#ifndef DRAWSTUFF_TEXTURE_PATH
#define DRAWSTUFF_TEXTURE_PATH "./../ode-0.13/drawstuff/textures"
#endif

#define GRAVITY 9.81
#define MAX_JOINT_NUM 3      // maximum joint num

#define DEG2RAD 0.0174533
#define RAD2DEG 57.2958

dsFunctions g_Fn;

//set addresses
static dWorldID g_World;
static dSpaceID g_Space;
static dJointGroupID g_Contactgroup;
static dGeomID g_Ground;


Object g_oObj[MAX_JOINT_NUM + 1];   // link global parameter
static dJointID g_oJoint[MAX_JOINT_NUM + 1]; // joint global parameter

double g_tar_q[MAX_JOINT_NUM] = { 0.0 , 45. * DEG2RAD, 30. * DEG2RAD }; // joint trarget position
double g_cur_q[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };  // current joint position
```

Include 및 global parameter 선언부.

각자 알맞은 객체 형태로 World, space 등등의 할당을 객체에 맞게 할당하기 위한 선언을 진행함.

해당 과제에 맞도록 MAX_JOINT_NUM을 3으로 재정의 함.

Target position을 과제에서 목표로 한 45도와 30도로 할당함.

```
void InitODE() {
    // ode main function
    // initiating world options
    dInitODE();
    g_World = dWorldCreate();
    g_Space = dHashSpaceCreate(0);
    g_Contactgroup = dJointGroupCreate(0);
    g_Ground = dCreatePlane(g_Space, 0, 0, 1, 0);

    dWorldSetGravity(g_World, 0, 0, -GRAVITY);  // gravity setting X,Y,Z
    dWorldSetERP(g_World, 0.2);
    dWorldSetCFM(g_World, 1e-5);

}



void RunODE(size_t width, size_t height) {
    //page 5
    //TO DO
    InitDrawStuff();
    InitODE();

    InitRobot();

    dsSimulationLoop(0, 0, width, height, &g_Fn);
}



void ReleaseODE() {

    dJointGroupDestroy(g_Contactgroup);
    dSpaceDestroy(g_Space);
    dWorldDestroy(g_World);
    dCloseODE();
}
```

ODE 기본 initialize 구현부

Ode에서 시뮬레이션 할 world와 space에 대한 기본적인 구현이 나타난 부분, ode를 실행할 경우, 할당 해제할 경우의 메소드가 구현되어있음.

해당 과제에선, 선언된 GRAVITY상수로 중력을 설정함. RunODE 메소드에서 시뮬레이션 실행 루프가 진행됨.

```
void nearCB(void* data, dGeomID o1, dGeomID o2) {
    int i;
    static const int N = 5;
    dContact contact[N];

    if ((g_Ground == o1) || (g_Ground == o2))   // if object collide with ground
    {
        if ((o1 == g_oObj[0].geom) && (o2 == g_Ground) || (o1 == g_Ground) && (o2 == g_oObj[0].geom))
        {
            // obj 1 collide with ground (fixed)
            return;
        }
        bool isGround = ((g_Ground == o1) || (g_Ground == o2));
        int n = dCollide(o1, o2, N, &contact[0].geom, sizeof(dContact));

        if (isGround)
        {
            for (int i = 0; i < n; i++)
            {
                contact[i].surface.mu = 0.1;
                contact[i].surface.mode = dContactBounce;
                contact[i].surface.bounce = 0.0;
                contact[i].surface.bounce_vel = 0.0;

                dJointID c = dJointCreateContact(g_World, g_Contactgroup, &contact[i]);

                dJointAttach(c, dGeomGetBody(contact[i].geom.g1), dGeomGetBody(contact[i].geom.g2));
            }
        }
    }
    else
    {
        //obstacle collide between object codes need collision box
    }
}
```

충돌 상황 구현부

ODE에서는 callback 함수 구현을 통해 충돌 처리를 할 수 있음 현재 구현부에서는 ground와 충돌할 경우에 대한 구현을 진행 함..

<instances>

mu : 마찰계수 (0.1)

mode : 충돌 방식 (bounce)

bounce : 반발계수

bounce_vel : 반발 속도

본 과제에서는 geom0에 대한 예외를 처리 함.

```
void SimLoopDrawStuff(int pause)
{
    //page 6
    //TO DO
    //dSpaceCollide(g_Space, 0, &nearCB);

    PControl();

    double dt = 0.1;
    dWorldStep(g_World, dt);

    dReal r, length;
    dVector3 length_box;

    //draw capsule object

    /* ... */

    // for visualization
    dsSetColor(0., 0.4, 0.4);
    dGeomCapsuleGetParams(g_oObj[0].geom, &r, &length); //set capsule params load from g_oObj
    dsDrawCapsuleD(dBodyGetPosition(g_oObj[0].body), dBodyGetRotation(g_oObj[0].body), (float)length, (float)r);    //load geom from init robot function : type casted

    dsSetColor(0.5, 0.0, 0.0);
    dGeomCapsuleGetParams(g_oObj[1].geom, &r, &length); //set capsule params load from g_oObj
    dsDrawCapsuleD(dBodyGetPosition(g_oObj[1].body), dBodyGetRotation(g_oObj[1].body), (float)length, (float)r);    //load geom from init robot function : type casted

    dsSetColor(0.5, 0., 0.7);
    dGeomCapsuleGetParams(g_oObj[2].geom, &r, &length); //set capsule params load from g_oObj
    dsDrawCapsuleD(dBodyGetPosition(g_oObj[2].body), dBodyGetRotation(g_oObj[2].body), (float)length, (float)r);    //load geom from init robot function : type casted
}
```

Simloop 구현부

주석된 부분에선 for문을 이용하여 3개의 capsule을 구현한 부분이 포함 되어있음.

Pointer를 이용하여 변수값을 동적으로 할당받은 후, dreal value를 typecasting하여 할당함.

```
void InitRobot()
{
    //body
    dMass mass;
    dMatrix3 R;

    //mass point
    dReal x[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
    dReal y[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
    dReal z[MAX_JOINT_NUM] = { 0.25, 1.0, 1.75 };

    //link pose
    dReal ori_x[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
    dReal ori_y[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
    dReal ori_z[MAX_JOINT_NUM] = { 1.0, 1.0, 1.0 };
    dReal ori_q[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };

    //link length
    dReal length[MAX_JOINT_NUM] = { 0.5, 1.0, 0.5 };

    //mass of A
    dReal weight[MAX_JOINT_NUM] = { 0.5, 1.0, 0.5 };

    dReal r[MAX_JOINT_NUM];
    for (int i = 0; i < MAX_JOINT_NUM; i++)
    {
        r[i] = 0.125;
    }

    //creating body
    for (int i = 0; i < MAX_JOINT_NUM; i++)
    {
        // Make Body gunctions
        g_oObj[i].body = dBodyCreate(g_World);  // creating body and return address of body
        dBodySetPosition(g_oObj[i].body, x[i], y[i], z[i]); //move body to position x,y,z (body address, x, y, z)
        dMassSetZero(&mass);    // set body mass zero (initalize)
        dMassSetCapsuleTotal(&mass, weight[i], 1, r[i], length[i]); //capsule mass setting (mass, heading, radious, length)
        dBodySetMass(g_oObj[i].body, &mass);    // set body mass
        g_oObj[i].geom = dCreateCapsule(g_Space, r[i], length[i]);  // save gobj (capsule)
        dGeomSetBody(g_oObj[i].geom, g_oObj[i].body);   //creating geometry fit to geom option (capsule)
        dRFromAxisAndAngle(R, ori_x[i], ori_y[i], ori_z[i], ori_q[i]);  // body angular setting
        dBodySetRotation(g_oObj[i].body, R);    // change body with angular matrix
    }
```

body구현부

과제에 맞도록 변수를 할당하고 body를 구현함. 무게중심을 알맞게 설정한 후 , 길이에 맞는 무게를 할당함. 반지름은 0.125로 동일하도록 구현하였음.

```
//Joints

//center of hindge
dReal c_x[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
dReal c_y[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
dReal c_z[MAX_JOINT_NUM] = { 0.0, 0.5, 1.5 };

//hindge axis
dReal axis_x[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
dReal axis_y[MAX_JOINT_NUM] = { 0.0, 1.0, 1.0 };
dReal axis_z[MAX_JOINT_NUM] = { 1.0, 0.0, 0.0 };

// fixed axis setting
g_oJoint[0] = dJointCreateFixed(g_World, 0);    //set gobj[0] fixed joint
dJointAttach(g_oJoint[0], 0, g_oObj[0].body);   // if second arg is 0, attach with world
dJointSetFixed(g_oJoint[0]);


// joint settings : can converted after with adding joint
for (int i = 1; i < MAX_JOINT_NUM; i++) // (joint 0 setted before)
{
    g_oJoint[i] = dJointCreateHinge(g_World, 0);
    dJointAttach(g_oJoint[i], g_oObj[i].body, g_oObj[i - 1].body);  // attach joint 0 and joint 1 (obj0 and obj1)
    dJointSetHingeAnchor(g_oJoint[i], c_x[i], c_y[i], c_z[i]);  // attach objects with hindge joint
    dJointSetHingeAxis(g_oJoint[i], axis_x[i], axis_y[i], axis_z[i]);
}
```

Joint 구현부

구현한 로봇의 joint회전은 0번 joint를 제외하고 y방향으로 가능하도록 설정함. 0번 joint는 groun에 고정되도록 설정하고, 이외의 joint들은 높이에 맞게 위치를 설정함..

0번 joint는 0,0,0에 위치하도록 설정하였으며 해당 joint는 회전축을 x축으로 설정하여 고정하였음

```
void PControl() //hindge joint Pcontrol
{
    dReal Kp = 10, fMax = 5000.0;
    dReal a_error_q[MAX_JOINT_NUM];

    for (int i = 1; i < MAX_JOINT_NUM; i++)
    {
        g_cur_q[i] = dJointGetHingeAngle(g_oJoint[i]);

        a_error_q[i] = g_tar_q[i] - g_cur_q[i]; // feedback

        //gymbol lock control
        if (g_tar_q[i] - g_cur_q[i] > 180.0*DEG2RAD)
        {
            g_cur_q[i] += 359.9*DEG2RAD;
        }

        if (g_tar_q[i] - g_cur_q[i] < -180.0*DEG2RAD)
        {
            g_cur_q[i] -= 359.9*DEG2RAD;
        }

        dJointSetHingeParam(g_oJoint[i], dParamVel, Kp*a_error_q[i]);
        dJointSetHingeParam(g_oJoint[i], dParamFMax, fMax);
    }
}
```

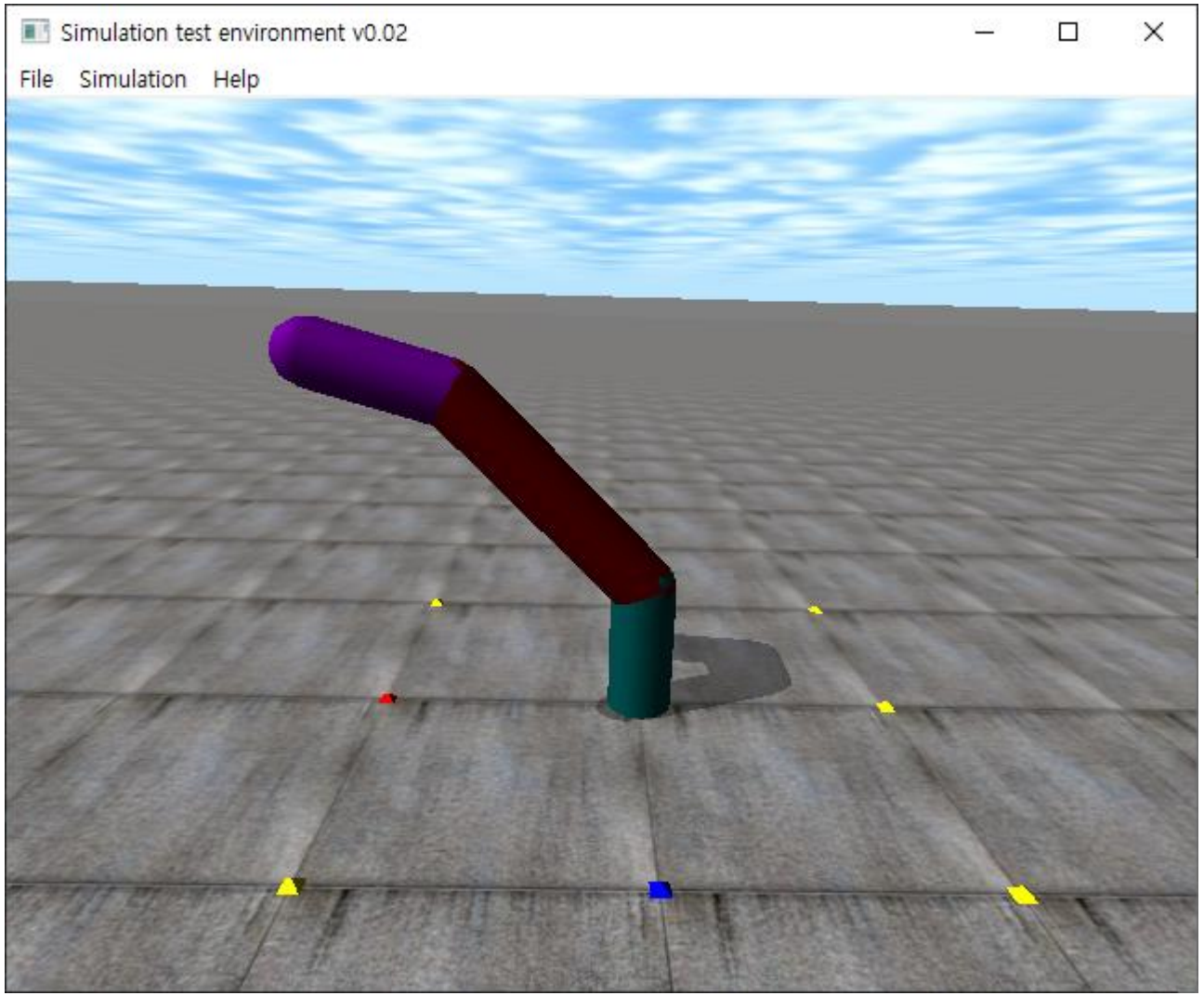
P제어 구현부

해당 구현부에서는 2dof 의 물체를 set함수를 이용하여 제어함.

구현 시 gimbol lock을 고려하여 구현하였음.

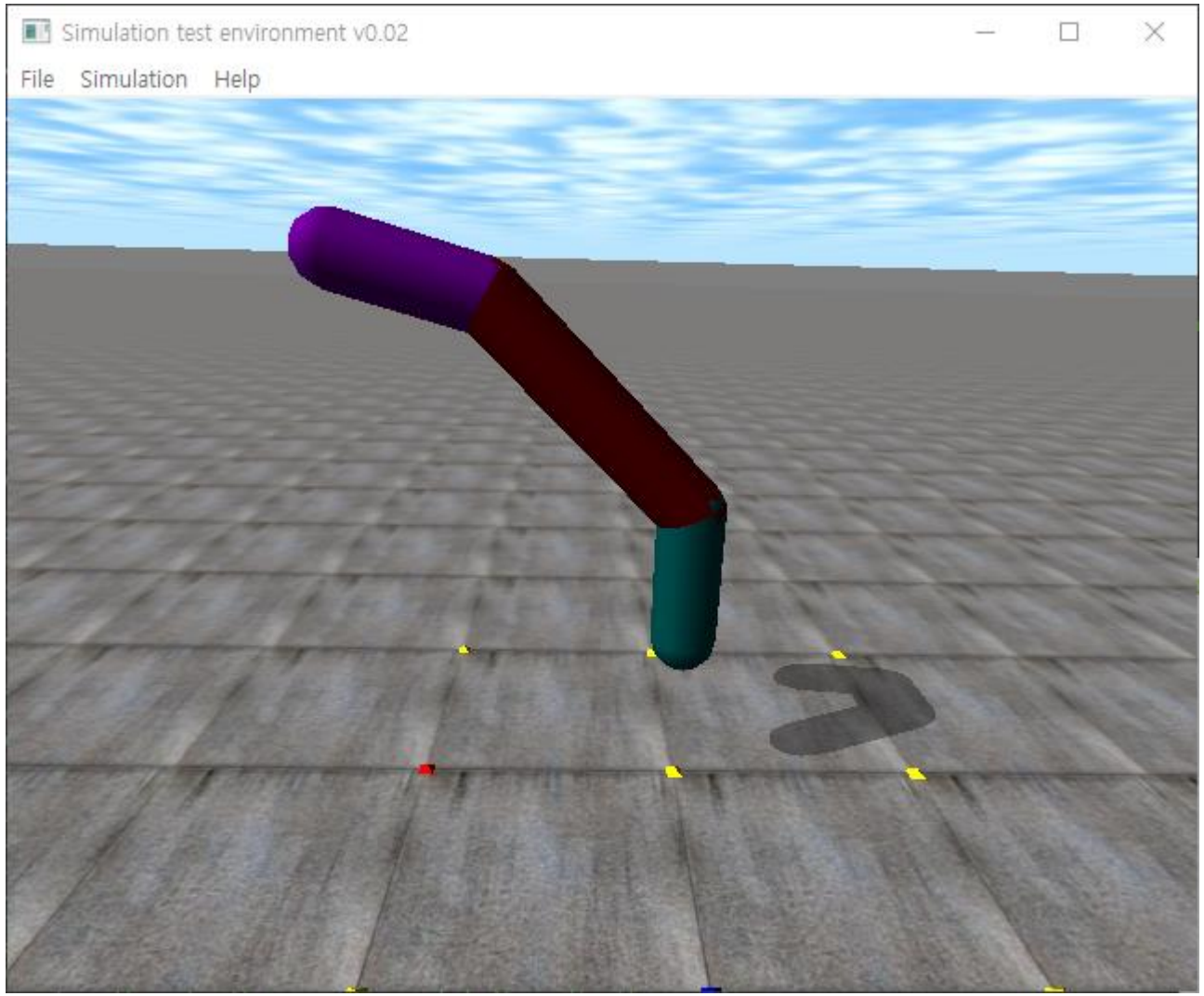
P gain은 10으로 설정하였고 잘 수렴하는 모습을 보임.

Hindge가 아닌 다른 형태의 joint일 경우에는 해당 호출 함수를 바꾸면 정상적으로 작동함.



출력모습

코드에 따라 잘 시뮬레이션이 진행된 것을 볼 수 있음.




```
void nearCB(void* data, dGeomID o1, dGeomID o2) {
    int i;
    static const int N = 5;
    dContact contact[N];

    if ((g_Ground == o1) || (g_Ground == o2))   // if object collide with ground
    {
        bool isGround = ((g_Ground == o1) || (g_Ground == o2));
        int n = dCollide(o1, o2, N, &contact[0].geom, sizeof(dContact));

        if (isGround)
        {
            for (int i = 0; i < n; i++)
            {
                contact[i].surface.mu = 0.1;
                contact[i].surface.mode = dContactBounce;
                contact[i].surface.bounce = 0.0;
                contact[i].surface.bounce_vel = 0.0;

                dJointID c = dJointCreateContact(g_World, g_Contactgroup, &contact[i]);

                dJointAttach(c, dGeomGetBody(contact[i].geom.g1), dGeomGetBody(contact[i].geom.g2));
            }
        }
    }
    else
    {
        //obstacle collide between object codes need collision box
    }
}
```

강의자료에 구현 되어있는 코드를 그대로 실행 할 경우 좌측과 같은 모습이 보임. 본 과제에서는 geom0에 대한 예외처리를 진행하여 공중에 떠오르지 않도록 구현함.

Ode를 처음 다뤄보는 상황에서 여러가지 type error나 제어를 바꾸는 과정에서 gimbal lock 현상이 나타나는 것을 확인하고 해당 부분에 대한 예외를 처리함.

```
//link pose
dReal ori_x[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
dReal ori_y[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
dReal ori_z[MAX_JOINT_NUM] = { 1.0, 1.0, 1.0 };
dReal ori_q[MAX_JOINT_NUM] = { 0.0, 0.0, 0.0 };
```

Link pose를 설정하는 부분에서 ori_q 배열에 할당되어 있는것을 보아 쿼터니안 각도를 이용하는 것으로 생각함. 처음 구현시 쿼터니안으로 진행되겠지 라고 생각하며 gimbal lock을 고려하지 않은 것을 보아 항상 gimbal lock을 고려하는 자세가 필요할 것이라 생각됨.

코드리뷰를 하며 본인의 노트북이 ubuntu 환경이라 window 데스크톱으로 옮기는 과정에서 인코딩 오류가 발생해 해당 코드에서 이상한 문자가 나타남.

(직접 작성한 주석은 모두 영어로 주석 남김)