Chapter 1

Agent code

1.1 car.h

```
#ifndef CAR_H_
#define CAR_H_

#include "motor.h"
#include <pthread.h>

#define NO_TURN          0
#define LEFT_TURN        1
#define RIGHT_TURN       2

#define TURN_MAGNITUDE   0.5f

class Car{

public:
        Car(int FR,int FL,int BR,int BL) : frontRightWheel(FR, WHEELMODE), frontLeftW
        backRightWheel(BR, WHEELMODE), backLeftWheel(BL, WHEELMODE){turn = NO_TURN; s
        void setSpeed(int, bool);
        int getSpeed();
        void turnCar(int);
        void setMode(int);
        int getMode();
        void startPing();
private:
        int direction;
        int speed;
        int turn;
        int mode;
```

```
        Motor frontRightWheel;
        Motor frontLeftWheel;
        Motor backRightWheel;
        Motor backLeftWheel;
        pthread_t thread_car;
        static void * staticEntryPoint(void * c);
        void ping();
};

#endif
```

1.2 car.cpp

```
#include "car.h"
#include <stdio.h>
#include <unistd.h>

pthread_mutex_t mutex_car = PTHREAD_MUTEX_INITIALIZER;

void Car::setSpeed(int theSpeed, bool dir){

        if(getMode() == FAILSAFE_MODE)
                return;

        try{
                switch(turn)
                {
                case NO_TURN:
                        //set all wheels same speed
                        frontLeftWheel.setSpeed(theSpeed, !dir);
                        backLeftWheel.setSpeed(theSpeed, !dir);
                        frontRightWheel.setSpeed(theSpeed, dir);
                        backRightWheel.setSpeed(theSpeed, dir);
                        break;
                case LEFT_TURN:
                        //set left wheels TURN_MAGNITUDE of right wheels
                        frontLeftWheel.setSpeed(theSpeed*TURN_MAGNITUDE, !di
                        backLeftWheel.setSpeed(theSpeed*TURN_MAGNITUDE, !dir
                        frontRightWheel.setSpeed(theSpeed, dir);
                        backRightWheel.setSpeed(theSpeed, dir);
                        break;
                case RIGHT_TURN:
                        //set right wheels TURN_MAGNITUDE of left wheels
                        frontLeftWheel.setSpeed(theSpeed, !dir);
                        backLeftWheel.setSpeed(theSpeed, !dir);
                        frontRightWheel.setSpeed(theSpeed*TURN_MAGNITUDE, di
```

```
                        backRightWheel.setSpeed(theSpeed*TURN_MAGNITUDE, dir);
                        break;
                }
                speed = theSpeed;
                direction = dir;
        }
        catch(MotorException e) {
                printf("ID: %d lost\n",e.ID);
                printError(e.status);
                setMode(FAILSAFE_MODE);
                printf("Wheels lost!\n");
                startPing();
        }

}

void Car::turnCar(int theTurn){

        if(getMode() == FAILSAFE_MODE)
                return;

        try{
                turn = theTurn;
                if(speed != 0){
                        setSpeed(speed, direction);
                        return;
                }
                if(turn == NO_TURN){
                        setSpeed(0,1);
                        return;
                }
                bool dir;
                if(turn == LEFT_TURN)
                        dir = 1;
                if(turn == RIGHT_TURN)
                        dir = 0;

                printf("direction %d\n",direction);
                frontLeftWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
                backLeftWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
                frontRightWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
                backRightWheel.setSpeed(1023*TURN_MAGNITUDE, dir);
        }
        catch(MotorException e) {
                printf("ID: %d lost\n",e.ID);
                printError(e.status);
```

```
                setMode(FAILSAFE_MODE);
                printf("Wheels lost!\n");
                startPing();
        }

}

void Car::setMode(int theMode){
        pthread_mutex_lock( &mutex_car );
        mode = theMode;
        pthread_mutex_unlock( &mutex_car );
}

int Car::getMode(){
        pthread_mutex_lock( &mutex_car );
        int temp = mode;
        pthread_mutex_unlock( &mutex_car );
        return temp;
}

void Car::ping(){
        printf("Ping Car\n");
        while(1){
                int count = 0;
                count += frontLeftWheel.ping();
                count += backLeftWheel.ping();
                count += frontRightWheel.ping();
                count += backRightWheel.ping();

                if(count == 4){
                        printf("All wheels active!\n");
                        setMode(IDLE_MODE);
                        return;
                }
        }
}

void Car::startPing(){
        pthread_create(&thread_car, NULL, Car::staticEntryPoint, this);
}

void * Car::staticEntryPoint(void * c)
{
    ((Car *) c)->ping();
    return NULL;
}
```

1.3 motor.h

```
#ifndef MOTOR_H_
#define MOTOR_H_

#include <dynamixel.h>
#include <pthread.h>

// Control table address
#define CW_ANGLE_LIMIT_L        6
#define CW_ANGLE_LIMIT_H        7
#define CCW_ANGLE_LIMIT_L       8
#define CCW_ANGLE_LIMIT_H       9
#define MAX_TORQUE_L            14
#define MAX_TORQUE_H            15
#define HIGH_LIMIT_VOLTAGE      13
#define GOAL_POSITION_L         30
#define GOAL_POSITION_H         31
#define MOVING_SPEED_L          32
#define MOVING_SPEED_H          33
#define PRESENT_POSITION_L      36
#define PRESENT_POSITION_H      37
#define PRESENT_SPEED_L         38
#define PRESENT_SPEED_H         39
#define MOVING                  46

#define WHEELMODE               0
#define SERVOMODE               1

#define CW                      1
#define CCW                     0

#define IDLE_MODE               0
#define FAILSAFE_MODE           1


class MotorException{
public:
        MotorException(int theID, int theStatus) : ID(theID), status(theStatus){};
        int ID;
        int status;
};

class Motor{
public:
```

```
        Motor(int, int);
        int getMode();
        int getPosition();
        int getSpeed();
        void setGoalPosition(int);
        void setSpeed(int,bool);
        void setMode(int);
        void setRotateDirection(int);
        void printErrorCode(void);
        void checkStatus();
        int ping();
private:
        int position;
        int speed;
        int mode;
        int ID;
        int commStatus;
        int rotateDirection;
};

void pingAll();
void printError(int status);

#endif
```

1.4 motor.cpp

```
#include "motor.h"
#include "dynamixel.h"
#include "stdio.h"
#include "communication.h"

Motor::Motor(int theID, int theMode){
        ID = theID;
        mode = theMode;
        commStatus = COMM_RXSUCCESS;
        setMode(mode);
}

int Motor::getMode(){
        return mode;
}

int Motor::getPosition(){

        int temp = readWord( ID, PRESENT_POSITION_L );
```

```
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
                throw MotorException(ID,commStatus);
        printErrorCode();
        position = temp;
        return position;
}

int Motor::getSpeed(){

        unsigned short temp = readWord( ID, PRESENT_SPEED_L );
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
                throw MotorException(ID,commStatus);
        printErrorCode();
        speed = temp & 1023;
        return speed;
}

void Motor::setGoalPosition(int thePosition){

        writeWord( ID, GOAL_POSITION_L, thePosition );
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
                throw MotorException(ID,commStatus);
        printErrorCode();
}


void Motor::setMode(int theMode){

        switch(theMode)
        {
        case WHEELMODE:
                writeWord( ID, CW_ANGLE_LIMIT_L, 0 );
                writeWord( ID, CCW_ANGLE_LIMIT_L, 0 );
                break;
        case SERVOMODE:
                writeWord( ID, CW_ANGLE_LIMIT_L, 0 );
                writeWord( ID, CCW_ANGLE_LIMIT_L, 1023 );
                break;
        default:
                printf("unknown mode: %d\n", theMode);
                return;
        }
        mode = theMode;
```

```
}

void Motor::setSpeed(int theSpeed, bool theDirection){

        writeWord( ID, MOVING_SPEED_L, theSpeed | (theDirection<<10) );
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
                throw MotorException(ID,commStatus);
        printErrorCode();
}

void Motor::setRotateDirection(int direction){

        switch(direction)
        {
        case CW:
                writeWord(ID, MOVING_SPEED_L, 1024);
                break;
        case CCW:
                writeWord(ID, MOVING_SPEED_L, 0);
                break;
        default:
                printf("invalid input: %d\n", direction);
                return;
        }
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
                throw MotorException(ID,commStatus);
        printErrorCode();

        rotateDirection = direction;
}

// Print error bit of status packet
void Motor::printErrorCode()
{
        if(getRXpacketError(ERRBIT_VOLTAGE) == 1)
                printf("Input voltage error!\n");

        if(getRXpacketError(ERRBIT_ANGLE) == 1)
                printf("Angle limit error!\n");

        if(getRXpacketError(ERRBIT_OVERHEAT) == 1)
                printf("Overheat error!\n");

        if(getRXpacketError(ERRBIT_RANGE) == 1)
```

```
                printf("Out of range error!\n");

        if(getRXpacketError(ERRBIT_CHECKSUM) == 1)
                printf("Checksum error!\n");

        if(getRXpacketError(ERRBIT_OVERLOAD) == 1)
                printf("Overload error!\n");

        if(getRXpacketError(ERRBIT_INSTRUCTION) == 1)
                printf("Instruction code error!\n");
}

void Motor::checkStatus(){

        unsigned char temp;
        for(int i = 0; i<50; i++)
        {
                if(i == 10 || i == 45)
                        continue;
                temp = readByte( ID, i );
                printf("%d:\t%d\t%d\n", ID, i, temp);
        }
        printf("\n");
}

int Motor::ping(){
        pingID(ID);
        commStatus = getResult();
        if( commStatus == COMM_RXSUCCESS )
        {
                //printf("Motor ID: %d active!\n",ID);
                return 1;
        }
        //printf("Motor ID: %d NOT active!\n",ID);
        return 0;
}

void pingAll(){
        for(int i = 0; i<254; i++){
                dxl_ping(i);
                if( dxl_get_result( ) == COMM_RXSUCCESS )
                {
                        printf("ID: %d active!\n",i);
                }
        }
}
```

```
void printError(int status){
        switch(status)
        {
        case COMM_TXFAIL:

                printf("COMM_TXFAIL: Failed transmit instruction packet!\n"
                break;

        case COMM_TXERROR:
                printf("COMM_TXERROR: Incorrect instruction packet!\n");
                break;

        case COMM_RXFAIL:
                printf("COMM_RXFAIL: Failed get status packet from device!\n
                break;

        case COMM_RXWAITING:
                printf("COMM_RXWAITING: Now recieving status packet!\n");
                break;

        case COMM_RXTIMEOUT:
                printf("COMM_RXTIMEOUT: There is no status packet!\n");
                break;

        case COMM_RXCORRUPT:
                printf("COMM_RXCORRUPT: Incorrect status packet!\n");
                break;

        default:
                printf("This is unknown error code!\n");
                break;
        }
}
```

1.5 manipulator.h

```
#ifndef MANIPULATOR_H_
#define MANIPULATOR_H_

#include "motor.h"
#include <pthread.h>

#define PI 3.14159265

#define XSTART           0
```

```
#define YSTART          155
#define ZSTART          77

class Manipulator{
public:
        Manipulator(int IDOne ,int IDTwo,int IDThree, int IDGrip_left, int IDGrip_rig
        one(IDOne, SERVOMODE), two(IDTwo, SERVOMODE), three(IDThree, SERVOMODE),
        grip_left(IDGrip_left, SERVOMODE), grip_right(IDGrip_right, SERVOMODE) {theta
        void goToPosition(int, int, int);
        void setAngles(float, float, float);
        void setGripper(bool);
        void drawLine(int, int, int, int, int);
        void drawCircle(int, int, int, int, float, float);
        void setMode(int);
        int getMode();
        void startPing();
private:
        float theta1;
        float theta2;
        float theta3;
        int mode;
        Motor one;
        Motor two;
        Motor three;
        Motor grip_left;
        Motor grip_right;
        pthread_t thread;
        static void * staticEntryPoint(void * c);
        void ping();
};

#endif
```

1.6 manipulator.cpp

```
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include "manipulator.h"

using namespace std;

#define D2      77      //length of first arm in mm
#define D3      155     //length of second arm in mm

#define ANGLE_TO_VALUE  (float)511*6/(5*PI)
```

```
#define GRIPPER_LEFT_ZERO       511-140
#define GRIPPER_RIGHT_ZERO      511+140
#define MAX_COUNT               5

pthread_mutex_t mutex_man = PTHREAD_MUTEX_INITIALIZER;

void Manipulator::goToPosition(int x, int y, int z){

        //return error if beyond max
//      if((x*x+y*y+z*z) > (D2+D3)*(D2+D3))
//      {
//              printf("invalid position!\n");
//              return;
//      }

        if(getMode() == FAILSAFE_MODE)
                return;

        float s3, c3, l;

        l = sqrt(x*x+y*y);
        c3 = (z*z + l*l - D2*D2 - D3*D3)/(2*D2*D3);
        s3 = sqrt(1-c3*c3);

        theta3 = atan2(s3,c3);
        theta2 = PI/2 - atan2(D3*s3, D2+D3*c3)-atan2(z,l);
        theta1 = atan2(x,y);

        setAngles(theta1, theta2, theta3);
}

void Manipulator::setAngles(float t1, float t2, float t3){

        if(getMode() == FAILSAFE_MODE)
                return;

        try{
                int dummy;

                if(t1 != t1)
                        printf("nan theta 1\n");
                else if(t1 > 5*PI/6){
                        one.setGoalPosition(1023);
                        printf("Theta 1 too high\n");
                }
```

```
            else if(t1 < -5*PI/6){
                one.setGoalPosition(0);
                printf("Theta 1 too low\n");
            }
            else{
                dummy = (float)(t1*ANGLE_TO_VALUE+511);
                one.setGoalPosition(dummy);
                //printf("one: %d\n",dummy);
            }

            if(t2 != t2)
                printf("nan theta 2\n");
            else if(t2 > 5*PI/6){
                two.setGoalPosition(1023);
                printf("Theta 2 too high\n");
            }
            else if(t2 < 0){
                two.setGoalPosition(511);
                printf("Theta 2 too low\n");
            }
            else{
                dummy = (float)(t2*ANGLE_TO_VALUE+511);
                two.setGoalPosition(dummy);
                //printf("two: %d\n",dummy);
            }

            if(t3 != t3)
                printf("nan theta 3\n");
            else if(t3 > 0.78*PI){
                three.setGoalPosition(989);
                printf("Theta 3 too high\n");
            }
            else if(t3 < -0.5*PI){
                three.setGoalPosition(51);
                printf("Theta 3 too low\n");
            }
            else{
                dummy = (float)(t3*ANGLE_TO_VALUE+511);
                three.setGoalPosition(dummy);
                //printf("three: %d\n",dummy);
            }
        }
        catch(MotorException e) {
            printf("ID: %d lost\n",e.ID);
            printError(e.status);
            setMode(FAILSAFE_MODE);
```

```
                printf("Manipulator lost!\n");
                startPing();
        }
}

void Manipulator::setGripper(bool on){

        if(getMode() == FAILSAFE_MODE)
                return;

        try{
                if(!on){
                        grip_left.setGoalPosition(511-50);
                        grip_right.setGoalPosition(511+50);
                        return;
                }

                int positionL, positionR, lastPositionL, lastPositionR;
                int counter = 0;
                //put servo set point to zero degrees
                grip_left.setGoalPosition(GRIPPER_LEFT_ZERO);
                grip_right.setGoalPosition(GRIPPER_RIGHT_ZERO);
                lastPositionR = grip_right.getPosition();
                lastPositionL = grip_left.getPosition();
                while(1){
                        positionL = grip_left.getPosition();
                        positionR = grip_right.getPosition();
                        printf("left: %d\tright: %d\n",positionL,positionR);

                        if(lastPositionL == positionL || lastPositionR == po
                                counter++;
                        else
                                counter = 0;
                        if(counter == MAX_COUNT)
                                return;
                        lastPositionL = positionL;
                        lastPositionR = positionR;
                        usleep(10000);
                }
        }
        catch(MotorException e) {
                printf("ID: %d lost\n",e.ID);
                printError(e.status);
                setMode(FAILSAFE_MODE);
                printf("Manipulator lost!\n");
                startPing();
```

```
        }
}

void Manipulator::drawLine(int xstart, int ystart, int xend, int yend, int z){

        if(getMode() == FAILSAFE_MODE)
                return;

        try{
                goToPosition(xstart,ystart,z+50);
                sleep(1);
                goToPosition(xstart,ystart,z);
                usleep(100000);
                int x = xend-xstart;
                int y = yend-ystart;
                int length = sqrt(x*x+y*y);
                x /= length;     //normalize
                y /= length;     //normalize
                for(int i = 0; i<length; i++){
                        printf("x: %d\ty: %d\n",xstart+i*x, ystart+i*y);
                        goToPosition(xstart+i*x, ystart+i*y, z);
                        usleep(10000);
                }
        }
        catch(MotorException e) {
                printf("ID: %d lost\n",e.ID);
                printError(e.status);
                setMode(FAILSAFE_MODE);
                printf("Manipulator lost!\n");
                startPing();
        }
}

void Manipulator::drawCircle(int xcenter, int ycenter, int z, int radius, float start

        if(getMode() == FAILSAFE_MODE)
                return;

        try{
                float t = startAngle;
                float stepSize = 0.01;
                while(t <= endAngle){
                        goToPosition(radius*sin(t) + xcenter, radius*cos(t) + ycenter
                        t += stepSize;
                        usleep(10000);
                }
```

```
        }
        catch(MotorException e) {
                printf("ID: %d lost\n",e.ID);
                printError(e.status);
                setMode(FAILSAFE_MODE);
                printf("Manipulator lost!\n");
                startPing();

        }
}

void Manipulator::setMode(int theMode){
        pthread_mutex_lock( &mutex_man );
        mode = theMode;
        pthread_mutex_unlock( &mutex_man );
}
int Manipulator::getMode(){
        pthread_mutex_lock( &mutex_man );
        int temp = mode;
        pthread_mutex_unlock( &mutex_man );
        return temp;
}

void Manipulator::ping(){
        printf("Ping Manipulators\n");
        while(1){
                int count = 0;
                count += one.ping();
                count += two.ping();
                count += three.ping();
                count += grip_left.ping();
                count += grip_right.ping();

                if(count == 5){
                        printf("All manipulator motors active!\n");
                        setMode(IDLE_MODE);
                        //printf("Returning to start position\n");
                        //goToPosition(XSTART,YSTART,ZSTART);
                        //setGripper(0);
                        return;
                }
        }
}

void Manipulator::startPing(){
```

```
        pthread_create(&thread, NULL, Manipulator::staticEntryPoint, this);
}

void * Manipulator::staticEntryPoint(void * c)
{
    ((Manipulator *) c)->ping();
    return NULL;
}
```

1.7 sensor.h

```
#ifndef SENSOR_H_
#define SENSOR_H_

#include <dynamixel.h>

//control table adress
#define IR_LEFT_FIRE_DATA       26
#define IR_CENTER_FIRE_DATA     27
#define IR_RIGHT_FIRE_DATA      28
#define LIGHT_LEFT_DATA         29
#define LIGHT_CENTER_DATA       30
#define LIGHT_RIGHT_DATA        31
#define IR_OBSTACLE_DETECTED    32
#define LIGHT_DETECTED          33
#define SOUND_DATA              35
#define BUZZER_DATA_NOTE        40
#define BUZZER_DATA_TIME        41

#define LEFT                    0
#define CENTER                  1
#define RIGHT                   2

/*melody:
        0: Rising
        1: Falling
        2: Fight
        4: Fail
        5: sad
        6: bip bip
        7: sad 2
        10: whistle rise
        11: bip bop
        15: bip bip 2
        16: phone
        21: whistle
```

```
        24: rtrtrrtrt
*/

class Sensor{
public:
        Sensor(int);
        int getIR(int);
        int getLight(int);       //only infrared light
        void playMelody(int);    //input range 0-26
        void playMelody(unsigned char*, int);    //play from arrays in songs.
        void ping();
        void setMode(int);
        int getMode();
private:
        int ID;
        int commStatus;
        int mode;
};

#endif
```

1.8 sensor.cpp

```
#include "motor.h"
#include "sensor.h"
#include "stdio.h"
#include <unistd.h>
#include "communication.h"

Sensor::Sensor(int theID){
        ID = theID;
        commStatus = COMM_RXSUCCESS;
        mode = IDLE_MODE;
}

int Sensor::getLight(int pos){

        int data = readByte( ID, LIGHT_LEFT_DATA + pos );
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
        {
                mode = FAILSAFE_MODE;
                printf("sensor lost\n");
        }
        return data;
}
```

```
int Sensor::getIR(int pos){

        int data = readByte( ID, IR_LEFT_FIRE_DATA + pos );
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
        {
                mode = FAILSAFE_MODE;
                printf("sensor lost\n");
        }


        return data;
}

void Sensor::playMelody(int song){

        if(song < 0 || song > 26){
                printf("invalid input\n");
                return;
        }
        writeByte(ID, BUZZER_DATA_TIME, 255);
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
        {
                mode = FAILSAFE_MODE;
                printf("sensor lost\n");
        }
        writeByte(ID, BUZZER_DATA_NOTE, song);
        commStatus = getResult();
        if(commStatus != COMM_RXSUCCESS)
        {
                mode = FAILSAFE_MODE;
                printf("sensor lost\n");
        }
}

void Sensor::playMelody(unsigned char* song, int length){



        for(int i = 0; i<length; i+=2)
        {

                if(song[i+1] != 100)
                {
```

```
                        writeByte(ID, BUZZER_DATA_TIME, 254);
                        writeByte(ID, BUZZER_DATA_NOTE, song[i+1]);
                        usleep(40000*song[i]);
                }
                else
                {
                        writeByte(ID, BUZZER_DATA_TIME, 0);
                        usleep(40000*song[i]);
                }


        }
        writeByte(ID, BUZZER_DATA_TIME, 0);

}

void Sensor::ping(){
        pingID(ID);
        commStatus = getResult();
        if( commStatus == COMM_RXSUCCESS )
        {
                printf("Sensor ID: %d active!\n",ID);
                setMode(IDLE_MODE);
        }
        else{
                setMode(FAILSAFE_MODE);
        }
}

void Sensor::setMode(int theMode){
        mode = theMode;
}

int Sensor::getMode(){
        return mode;
}
```

1.9 interface.h

```
#ifndef INTERFACE_H_
#define INTERFACE_H_

#include "manipulator.h"
#include "car.h"

void windowInit();
```

```
void checkEvent(Manipulator *, Car *);

#endif
```

1.10 interface.cpp

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <stdio.h>
#include <stdlib.h>
#include "interface.h"
#include "manipulator.h"

#define KEYMASK ButtonPressMask | KeyPressMask | KeyReleaseMask | ButtonReleaseMask |

#define FORWARD 25
#define BACKWARD 39
#define LEFT 38
#define RIGHT 40

#define LEFT_MOUSE_BUTTON       1
#define RIGHT_MOUSE_BUTTON      3
#define MOUSE_WHEEL             2
#define MOUSE_WHEEL_FORWARD     4
#define MOUSE_WHEEL_BACKWARD    5

Display *display;
Window window;
XEvent event;
bool button = 0;
bool buttonR = 0;
int xpos = XSTART;
int ypos = YSTART;
int zpos = ZSTART;
int xzero = 0;
int yzero = 0;

void windowInit()
{
    int s;
    /* open connection with the server */
    display = XOpenDisplay(NULL);
    if (display == NULL)
    {
        fprintf(stderr, "Cannot open display\n");
        exit(1);
```

```
    }

    s = DefaultScreen(display);

    /* create window */
    window = XCreateSimpleWindow(display, RootWindow(display, s), 10, 10, 50
                            BlackPixel(display, s), WhitePixel(display, s));

    /* select kind of events we are interested in */
    XSelectInput(display, window, KEYMASK);

    /* map (show) the window */
    XMapWindow(display, window);

    //do not detect autorepeating events from keyboard
    XAutoRepeatOff(display);
    printf("Display open\n");
}
void checkEvent(Manipulator *man, Car *car){
        XNextEvent (display, & event);
        switch(event.type){
                case MotionNotify:
                        if(button){
                                xpos -= event.xmotion.x - xzero;
                                ypos -= event.xmotion.y - yzero;
                                xzero = event.xmotion.x;
                                yzero = event.xmotion.y;
                                //printf("xpos: %d\t ypos: %d\n", xpos, ypos
                                man->goToPosition(xpos,ypos,zpos);
                        }
                        break;
                case ButtonPress:
                        if(event.xkey.keycode == LEFT_MOUSE_BUTTON)
                        {
                                button = 1;
                                xzero = event.xbutton.x;
                                yzero = event.xbutton.y;
                        }
                        if(event.xkey.keycode == RIGHT_MOUSE_BUTTON)
                        {
                                buttonR ^= 1;
                                man->setGripper(buttonR);
                        }
                        if(event.xkey.keycode == MOUSE_WHEEL_FORWARD)
                        {
                                zpos-=10;
```

```
                man->goToPosition(xpos,ypos,zpos);
            }
            if(event.xkey.keycode == MOUSE_WHEEL_BACKWARD)
            {
                zpos+=10;
                man->goToPosition(xpos,ypos,zpos);
            }

            printf( "KeyPress: %d\n", event.xkey.keycode );
            break;
        case ButtonRelease:
            if(event.xkey.keycode == LEFT_MOUSE_BUTTON)
                button = 0;
            break;
        case KeyPress:
            //printf( "KeyPress: %d\n", e.xkey.keycode );
            switch(event.xkey.keycode){
                case FORWARD:
                    printf("forward\n");
                    car->setSpeed(1023,1);
                    break;
                case BACKWARD:
                    car->setSpeed(1023,0);
                    printf("backward\n");
                    break;
                case RIGHT:
                    car->turnCar(RIGHT_TURN);
                    printf("right\n");
                    break;
                case LEFT:
                    car->turnCar(LEFT_TURN);
                    printf("left\n");
                    break;
                default:
                    printf("unknown:%d\n",event.xkey.keycode);
            }
            break;
        case KeyRelease:
            //printf( "KeyRelease: %d\n", e.xkey.keycode );
            switch(event.xkey.keycode){
                case FORWARD:
                    car->setSpeed(0,1);
                    printf("forward released\n");
                    break;
                case BACKWARD:
                    car->setSpeed(0,1);
```

```
                                        printf("backward released\n");
                                        break;
                                case RIGHT:
                                        car->turnCar(NO_TURN);
                                        printf("right released\n");
                                        break;
                                case LEFT:
                                        car->turnCar(NO_TURN);
                                        printf("left released\n");
                                        break;
                                default:
                                        printf("unknown:%d\n",event.xkey.key
                        }
                        break;
        }
}
```

1.11 dynamixel.h

```
#ifndef _DYNAMIXEL_HEADER
#define _DYNAMIXEL_HEADER

#ifdef __cplusplus
extern "C" {
#endif


////////////// device control methods /////////////////////////
int dxl_initialize(int deviceIndex, int baudnum );
void dxl_terminate();


////////////// set/get packet methods //////////////////////////
#define MAXNUM_TXPARAM          (150)
#define MAXNUM_RXPARAM          (60)

void dxl_set_txpacket_id(int id);
#define BROADCAST_ID            (254)

void dxl_set_txpacket_instruction(int instruction);
#define INST_PING                       (1)
#define INST_READ                       (2)
#define INST_WRITE                      (3)
#define INST_REG_WRITE          (4)
#define INST_ACTION                     (5)
#define INST_RESET                      (6)
```

```
#define INST_SYNC_WRITE         (131)

void dxl_set_txpacket_parameter(int index, int value);
void dxl_set_txpacket_length(int length);

int dxl_get_rxpacket_error(int errbit);
#define ERRBIT_VOLTAGE          (1)
#define ERRBIT_ANGLE            (2)
#define ERRBIT_OVERHEAT         (4)
#define ERRBIT_RANGE            (8)
#define ERRBIT_CHECKSUM         (16)
#define ERRBIT_OVERLOAD         (32)
#define ERRBIT_INSTRUCTION      (64)

int dxl_get_rxpacket_length(void);
int dxl_get_rxpacket_parameter(int index);


// utility for value
int dxl_makeword(int lowbyte, int highbyte);
int dxl_get_lowbyte(int word);
int dxl_get_highbyte(int word);


////////// packet communication methods ///////////////////////
void dxl_tx_packet(void);
void dxl_rx_packet(void);
void dxl_txrx_packet(void);

int dxl_get_result(void);
#define COMM_TXSUCCESS          (0)
#define COMM_RXSUCCESS          (1)
#define COMM_TXFAIL             (2)
#define COMM_RXFAIL             (3)
#define COMM_TXERROR            (4)
#define COMM_RXWAITING          (5)
#define COMM_RXTIMEOUT          (6)
#define COMM_RXCORRUPT          (7)


////////////// high communication methods ////////////////////////
void dxl_ping(int id);
int dxl_read_byte(int id, int address);
void dxl_write_byte(int id, int address, int value);
int dxl_read_word(int id, int address);
void dxl_write_word(int id, int address, int value);
```

```
#ifdef __cplusplus
}
#endif

#endif
```

1.12 dynamixel.c

```
#include "dxl_hal.h"
#include "dynamixel.h"

#define ID                                  (2)
#define LENGTH                              (3)
#define INSTRUCTION                         (4)
#define ERRBIT                              (4)
#define PARAMETER                           (5)
#define DEFAULT_BAUDNUMBER      (1)

unsigned char gbInstructionPacket[MAXNUM_TXPARAM+10] = {0};
unsigned char gbStatusPacket[MAXNUM_RXPARAM+10] = {0};
unsigned char gbRxPacketLength = 0;
unsigned char gbRxGetLength = 0;
int gbCommStatus = COMM_RXSUCCESS;
int giBusUsing = 0;


int dxl_initialize(int deviceIndex, int baudnum )
{
        float baudrate;
        baudrate = 2000000.0f / (float)(baudnum + 1);

        if( dxl_hal_open(deviceIndex, baudrate) == 0 )
                return 0;

        gbCommStatus = COMM_RXSUCCESS;
        giBusUsing = 0;
        return 1;
}

void dxl_terminate(void)
{
        dxl_hal_close();
}
```

```
void dxl_tx_packet(void)
{
        unsigned char i;
        unsigned char TxNumByte, RealTxNumByte;
        unsigned char checksum = 0;

        if( giBusUsing == 1 )
                return;

        giBusUsing = 1;

        if( gbInstructionPacket[LENGTH] > (MAXNUM_TXPARAM+2) )
        {
                gbCommStatus = COMM_TXERROR;
                giBusUsing = 0;
                return;
        }

        if( gbInstructionPacket[INSTRUCTION] != INST_PING
                && gbInstructionPacket[INSTRUCTION] != INST_READ
                && gbInstructionPacket[INSTRUCTION] != INST_WRITE
                && gbInstructionPacket[INSTRUCTION] != INST_REG_WRITE
                && gbInstructionPacket[INSTRUCTION] != INST_ACTION
                && gbInstructionPacket[INSTRUCTION] != INST_RESET
                && gbInstructionPacket[INSTRUCTION] != INST_SYNC_WRITE )
        {
                gbCommStatus = COMM_TXERROR;
                giBusUsing = 0;
                return;
        }

        gbInstructionPacket[0] = 0xff;
        gbInstructionPacket[1] = 0xff;
        for( i=0; i<(gbInstructionPacket[LENGTH]+1); i++ )
                checksum += gbInstructionPacket[i+2];
        gbInstructionPacket[gbInstructionPacket[LENGTH]+3] = ~checksum;

        if( gbCommStatus == COMM_RXTIMEOUT || gbCommStatus == COMM_RXCORRUPT )
                dxl_hal_clear();

        TxNumByte = gbInstructionPacket[LENGTH] + 4;
        RealTxNumByte = dxl_hal_tx( (unsigned char*)gbInstructionPacket, TxNumByte );

        if( TxNumByte != RealTxNumByte )
        {
                gbCommStatus = COMM_TXFAIL;
```

```
                giBusUsing = 0;
                return;
        }

        if( gbInstructionPacket[INSTRUCTION] == INST_READ )
                dxl_hal_set_timeout( gbInstructionPacket[PARAMETER+1] + 6 );
        else
                dxl_hal_set_timeout( 6 );

        gbCommStatus = COMM_TXSUCCESS;
}

void dxl_rx_packet(void)
{
        unsigned char i, j, nRead;
        unsigned char checksum = 0;

        if( giBusUsing == 0 )
                return;

        if( gbInstructionPacket[ID] == BROADCAST_ID )
        {
                gbCommStatus = COMM_RXSUCCESS;
                giBusUsing = 0;
                return;
        }

        if( gbCommStatus == COMM_TXSUCCESS )
        {
                gbRxGetLength = 0;
                gbRxPacketLength = 6;
        }

        nRead = dxl_hal_rx( (unsigned char*)&gbStatusPacket[gbRxGetLength],
        gbRxGetLength += nRead;
        if( gbRxGetLength < gbRxPacketLength )
        {
                if( dxl_hal_timeout() == 1 )
                {
                        if(gbRxGetLength == 0)
                                gbCommStatus = COMM_RXTIMEOUT;
                        else
                                gbCommStatus = COMM_RXCORRUPT;
                        giBusUsing = 0;
                        return;
                }
```

```
        }

        // Find packet header
        for( i=0; i<(gbRxGetLength-1); i++ )
        {
                if( gbStatusPacket[i] == 0xff && gbStatusPacket[i+1] == 0xff )
                {
                        break;
                }
                else if( i == gbRxGetLength-2 && gbStatusPacket[gbRxGetLength-1] == 0
                {
                        break;
                }
        }
        if( i > 0 )
        {
                for( j=0; j<(gbRxGetLength-i); j++ )
                        gbStatusPacket[j] = gbStatusPacket[j + i];

                gbRxGetLength -= i;
        }

        if( gbRxGetLength < gbRxPacketLength )
        {
                gbCommStatus = COMM_RXWAITING;
                return;
        }

        // Check id pairing
        if( gbInstructionPacket[ID] != gbStatusPacket[ID])
        {
                gbCommStatus = COMM_RXCORRUPT;
                giBusUsing = 0;
                return;
        }

        gbRxPacketLength = gbStatusPacket[LENGTH] + 4;
        if( gbRxGetLength < gbRxPacketLength )
        {
                nRead = dxl_hal_rx( (unsigned char*)&gbStatusPacket[gbRxGetLength], g
                gbRxGetLength += nRead;
                if( gbRxGetLength < gbRxPacketLength )
                {
                        gbCommStatus = COMM_RXWAITING;
                        return;
                }
```

```
        }

        // Check checksum
        for( i=0; i<(gbStatusPacket[LENGTH]+1); i++ )
                checksum += gbStatusPacket[i+2];
        checksum = ~checksum;

        if( gbStatusPacket[gbStatusPacket[LENGTH]+3] != checksum )
        {
                gbCommStatus = COMM_RXCORRUPT;
                giBusUsing = 0;
                return;
        }

        gbCommStatus = COMM_RXSUCCESS;
        giBusUsing = 0;
}

void dxl_txrx_packet(void)
{
        dxl_tx_packet();

        if( gbCommStatus != COMM_TXSUCCESS )
                return;

        do{
                dxl_rx_packet();
        }while( gbCommStatus == COMM_RXWAITING );
}

int dxl_get_result(void)
{
        return gbCommStatus;
}

void dxl_set_txpacket_id( int id )
{
        gbInstructionPacket[ID] = (unsigned char)id;
}

void dxl_set_txpacket_instruction( int instruction )
{
        gbInstructionPacket[INSTRUCTION] = (unsigned char)instruction;
}

void dxl_set_txpacket_parameter( int index, int value )
```

```
{
        gbInstructionPacket[PARAMETER+index] = (unsigned char)value;
}

void dxl_set_txpacket_length( int length )
{
        gbInstructionPacket[LENGTH] = (unsigned char)length;
}

int dxl_get_rxpacket_error( int errbit )
{
        if( gbStatusPacket[ERRBIT] & (unsigned char)errbit )
                return 1;

        return 0;
}

int dxl_get_rxpacket_length(void)
{
        return (int)gbStatusPacket[LENGTH];
}

int dxl_get_rxpacket_parameter( int index )
{
        return (int)gbStatusPacket[PARAMETER+index];
}

int dxl_makeword( int lowbyte, int highbyte )
{
        unsigned short word;

        word = highbyte;
        word = word << 8;
        word = word + lowbyte;
        return (int)word;
}

int dxl_get_lowbyte( int word )
{
        unsigned short temp;

        temp = word & 0xff;
        return (int)temp;
}

int dxl_get_highbyte( int word )
```

```
{
        unsigned short temp;

        temp = word & 0xff00;
        temp = temp >> 8;
        return (int)temp;
}

void dxl_ping( int id )
{
        while(giBusUsing);

        gbInstructionPacket[ID] = (unsigned char)id;
        gbInstructionPacket[INSTRUCTION] = INST_PING;
        gbInstructionPacket[LENGTH] = 2;

        dxl_txrx_packet();
}

int dxl_read_byte( int id, int address )
{
        while(giBusUsing);

        gbInstructionPacket[ID] = (unsigned char)id;
        gbInstructionPacket[INSTRUCTION] = INST_READ;
        gbInstructionPacket[PARAMETER] = (unsigned char)address;
        gbInstructionPacket[PARAMETER+1] = 1;
        gbInstructionPacket[LENGTH] = 4;

        dxl_txrx_packet();

        return (int)gbStatusPacket[PARAMETER];
}

void dxl_write_byte( int id, int address, int value )
{
        while(giBusUsing);

        gbInstructionPacket[ID] = (unsigned char)id;
        gbInstructionPacket[INSTRUCTION] = INST_WRITE;
        gbInstructionPacket[PARAMETER] = (unsigned char)address;
        gbInstructionPacket[PARAMETER+1] = (unsigned char)value;
        gbInstructionPacket[LENGTH] = 4;

        dxl_txrx_packet();
}
```

```
int dxl_read_word( int id, int address )
{
        while(giBusUsing);

        gbInstructionPacket[ID] = (unsigned char)id;
        gbInstructionPacket[INSTRUCTION] = INST_READ;
        gbInstructionPacket[PARAMETER] = (unsigned char)address;
        gbInstructionPacket[PARAMETER+1] = 2;
        gbInstructionPacket[LENGTH] = 4;

        dxl_txrx_packet();

        return dxl_makeword((int)gbStatusPacket[PARAMETER], (int)gbStatusPacket[PARAM
}

void dxl_write_word( int id, int address, int value )
{
        while(giBusUsing);

        gbInstructionPacket[ID] = (unsigned char)id;
        gbInstructionPacket[INSTRUCTION] = INST_WRITE;
        gbInstructionPacket[PARAMETER] = (unsigned char)address;
        gbInstructionPacket[PARAMETER+1] = (unsigned char)dxl_get_lowbyte(value);
        gbInstructionPacket[PARAMETER+2] = (unsigned char)dxl_get_highbyte(value);
        gbInstructionPacket[LENGTH] = 5;

        dxl_txrx_packet();
}
```

1.13 dxl_ hal.h

```
#ifndef _DYNAMIXEL_HAL_HEADER
#define _DYNAMIXEL_HAL_HEADER


#ifdef __cplusplus
extern "C" {
#endif


int dxl_hal_open(int deviceIndex, float baudrate);
void dxl_hal_close();
int dxl_hal_set_baud( float baudrate );
void dxl_hal_clear();
int dxl_hal_tx( unsigned char *pPacket, int numPacket );
```

```
int dxl_hal_rx( unsigned char *pPacket, int numPacket );
void dxl_hal_set_timeout( int NumRcvByte );
int dxl_hal_timeout();



#ifdef __cplusplus
}
#endif

#endif
```

1.14 dxl_ hal.c

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <fcntl.h>
#include <termios.h>
#include <linux/serial.h>
#include <sys/ioctl.h>
#include <sys/time.h>

#include "dxl_hal.h"

int     gSocket_fd      = -1;
long    glStartTime     = 0;
float   gfRcvWaitTime   = 0.0f;
float   gfByteTransTime = 0.0f;

char    gDeviceName[20];

int dxl_hal_open(int deviceIndex, float baudrate)
{
        struct termios newtio;
        struct serial_struct serinfo;
        char dev_name[100] = {0, };

        sprintf(dev_name, "/dev/ttyUSB%d", deviceIndex);

        strcpy(gDeviceName, dev_name);
        memset(&newtio, 0, sizeof(newtio));
        dxl_hal_close();

        if((gSocket_fd = open(gDeviceName, O_RDWR|O_NOCTTY|O_NONBLOCK)) < 0)
                fprintf(stderr, "device open error: %s\n", dev_name);
```

```
        goto DXL_HAL_OPEN_ERROR;
    }

    newtio.c_cflag          = B38400|CS8|CLOCAL|CREAD;
    newtio.c_iflag          = IGNPAR;
    newtio.c_oflag          = 0;
    newtio.c_lflag          = 0;
    newtio.c_cc[VTIME]      = 0;    // time-out    (TIME * 0.1   ) 0 : disable
    newtio.c_cc[VMIN]       = 0;    // MIN    read    return

    tcflush(gSocket_fd, TCIFLUSH);
    tcsetattr(gSocket_fd, TCSANOW, &newtio);

    if(gSocket_fd == -1)
        return 0;

    if(ioctl(gSocket_fd, TIOCGSERIAL, &serinfo) < 0) {
        fprintf(stderr, "Cannot get serial info\n");
        return 0;
    }

    serinfo.flags &= ~ASYNC_SPD_MASK;
    serinfo.flags |= ASYNC_SPD_CUST;
    serinfo.custom_divisor = serinfo.baud_base / baudrate;

    if(ioctl(gSocket_fd, TIOCSSERIAL, &serinfo) < 0) {
        fprintf(stderr, "Cannot set serial info\n");
        return 0;
    }

    dxl_hal_close();

    gfByteTransTime = (float)((1000.0f / baudrate) * 12.0f);

    strcpy(gDeviceName, dev_name);
    memset(&newtio, 0, sizeof(newtio));
    dxl_hal_close();

    if((gSocket_fd = open(gDeviceName, O_RDWR|O_NOCTTY|O_NONBLOCK)) < 0) {
        fprintf(stderr, "device open error: %s\n", dev_name);
        goto DXL_HAL_OPEN_ERROR;
    }

    newtio.c_cflag          = B38400|CS8|CLOCAL|CREAD;
    newtio.c_iflag          = IGNPAR;
    newtio.c_oflag          = 0;
```

```
        newtio.c_lflag          = 0;
        newtio.c_cc[VTIME]      = 0;    // time-out     (TIME * 0.1    ) 0 :
        newtio.c_cc[VMIN]       = 0;    // MIN     read     return

        tcflush(gSocket_fd, TCIFLUSH);
        tcsetattr(gSocket_fd, TCSANOW, &newtio);

        return 1;

DXL_HAL_OPEN_ERROR:
        dxl_hal_close();
        return 0;
}

void dxl_hal_close()
{
        if(gSocket_fd != -1)
                close(gSocket_fd);
        gSocket_fd = -1;
}

int dxl_hal_set_baud( float baudrate )
{
        struct serial_struct serinfo;

        if(gSocket_fd == -1)
                return 0;

        if(ioctl(gSocket_fd, TIOCGSERIAL, &serinfo) < 0) {
                fprintf(stderr, "Cannot get serial info\n");
                return 0;
        }

        serinfo.flags &= ~ASYNC_SPD_MASK;
        serinfo.flags |= ASYNC_SPD_CUST;
        serinfo.custom_divisor = serinfo.baud_base / baudrate;

        if(ioctl(gSocket_fd, TIOCSSERIAL, &serinfo) < 0) {
                fprintf(stderr, "Cannot set serial info\n");
                return 0;
        }

        //dxl_hal_close();
        //dxl_hal_open(gDeviceName, baudrate);

        gfByteTransTime = (float)((1000.0f / baudrate) * 12.0f);
```

```
        return 1;
}

void dxl_hal_clear(void)
{
        tcflush(gSocket_fd, TCIFLUSH);
}

int dxl_hal_tx( unsigned char *pPacket, int numPacket )
{
        return write(gSocket_fd, pPacket, numPacket);
}

int dxl_hal_rx( unsigned char *pPacket, int numPacket )
{
        memset(pPacket, 0, numPacket);
        return read(gSocket_fd, pPacket, numPacket);
}

static inline long myclock()
{
        struct timeval tv;
        gettimeofday (&tv, NULL);
        return (tv.tv_sec * 1000 + tv.tv_usec / 1000);
}

void dxl_hal_set_timeout( int NumRcvByte )
{
        glStartTime = myclock();
        gfRcvWaitTime = (float)(gfByteTransTime*(float)NumRcvByte + 5.0f);
}

int dxl_hal_timeout(void)
{
        long time;

        time = myclock() - glStartTime;

        if(time > gfRcvWaitTime)
                return 1;
        else if(time < 0)
                glStartTime = myclock();

        return 0;
}
```

1.15 communication.h

```
#ifndef COMMUNICATION_H_
#define COMMUNICATION_H_

int readWord(int, int);
int readByte(int, int);
int getResult();
int getRXpacketError(int);
void writeWord(int,int,int);
void writeByte(int,int,int);
void pingID(int);

#endif
```

1.16 communication.cpp

```
#include <dynamixel.h>
#include <pthread.h>

//Mutex is used for multiple access from threads
//Best way would be to make communication atomic
//such that the communication would finnish without
//being interrupted. That way yould could avoid timeout error
pthread_mutex_t mutex_comm = PTHREAD_MUTEX_INITIALIZER;

int readWord(int id, int adress){
        pthread_mutex_lock( &mutex_comm );
        int temp = dxl_read_word(id, adress);
        pthread_mutex_unlock( &mutex_comm );
        return temp;
}

int readByte(int id, int adress){
        pthread_mutex_lock( &mutex_comm );
        int temp = dxl_read_byte(id, adress);
        pthread_mutex_unlock( &mutex_comm );
        return temp;
}

int getResult(){
        pthread_mutex_lock( &mutex_comm );
        int temp = dxl_get_result();
        pthread_mutex_unlock( &mutex_comm );
        return temp;
```

```
}

int getRXpacketError(int errbit){
        pthread_mutex_lock( &mutex_comm );
        int temp = dxl_get_rxpacket_error(errbit);
        pthread_mutex_unlock( &mutex_comm );
        return temp;
}

void writeWord(int id, int adress, int value){
        pthread_mutex_lock( &mutex_comm );
        dxl_write_word(id, adress, value);
        pthread_mutex_unlock( &mutex_comm );
}

void writeByte(int id,int adress,int value){
        pthread_mutex_lock( &mutex_comm );
        dxl_write_byte(id, adress, value);
        pthread_mutex_unlock( &mutex_comm );
}

void pingID(int id){
        pthread_mutex_lock( &mutex_comm );
        dxl_ping(id);
        pthread_mutex_unlock( &mutex_comm );
}
```

1.17 json_ processing.h

```
//defines
#define BUFFER_SIZE   (256 * 1024)  /* 256 KB */
#define URL_FORMAT    "https://wodinaz.com/%s"
#define URL_SIZE      256

//includes
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <string>
#include <vector>
#include <map>

using namespace std;

//functions
```

```
void json_test_function();
//example code that uses the four basic functions to communicate with the se

void debug_print_map(map<string,double> mymap);
// a debug function used to print maps received from the server

void debug_print_vector(vector<string> myvector);
//debug function used to print vectors


void json_send_data(map<string,double> mymap);
// Uploads the provided map of sensor values to the server

map<string,double> json_get_data(int id);
// Downloads sensor data from the server. The user must choose which agent (

void json_send_command(string cmd,int id);
// Uploads a command to the server.
//The agent with the corresponding id will download this command

vector<string> json_get_commands(int id);
//Download commands from the server.
```

1.18 json_ processing.cpp

```
/*
 * Copyright (c) 2009-2013 Petri Lehtinen <petri@digip.org>
 *
 * Jansson is free software; you can redistribute it and/or modify
 * it under the terms of the MIT license. See LICENSE for details.
 */

#include <stdlib.h>
#include <string.h>
#include <stdio.h>

#include <jansson.h>

#include "http_functions.h"

#define BUFFER_SIZE  (256 * 1024)  /* 256 KB */

#define URL_FORMAT   "https://wodinaz.com/%s"
#define URL_SIZE     256
int i=0;
```

```
//URL's
#define PATH_CONNECT "connect"
#define PATH_DATA "data/"
#define PATH_COMMAND "command/"

//C++ stuff
#include <string>
#include <iostream>
#include <ostream>
#include <sstream>
#include <vector>
#include <map>
using namespace std;

int myID=0;
int testID=0;

//functions



void debug_print_map(map<string,double> mymap){
    for (map<string,double>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
    {
      string key = it->first;
      double value = it->second;
      printf ("sensor %s has value %f\n",key.c_str(),value);
    }
}

void debug_print_vector(vector<string> myvector){
    for (vector<string>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
    {
      string command = *it;
      printf ("command: %s\n",command.c_str());
    }
}

string convertIntToString(int number)
{
    if (number == 0)
        return "0";
    string temp="";
    string returnvalue="";
    while (number>0)
    {
```

```
        temp+=number%10+48;
        number/=10;
    }
    for (int i=0;i<temp.length();i++)
        returnvalue+=temp[temp.length()-i-1];
    return returnvalue;
}
int convertStringToInt(string inputString){
    return atoi(inputString.c_str());
}
double convertStringToDouble(string inputString){
    stringstream ss(inputString);
    double result;
    return ss >> result ? result : 0;

}
string convertDoubleToString(double number){
    ostringstream convert;   // stream used for the conversion

    convert << number;      // insert the textual representation of 'Number

    return convert.str(); // set 'Result' to the contents of the stream

}

map<string,double> json_get_data(int id){
    printf("starting get_data\n");
    map<string,double> data_map;
    int root_length=0;
    char *text_response;
    char url[URL_SIZE];
    string id_path=PATH_DATA;

    string id_string = "client_"+convertIntToString(id);
    id_path.append(id_string);
    snprintf(url, URL_SIZE, URL_FORMAT, id_path.c_str());
    printf("url:%s\n",url);

    text_response = http_request(url);
    printf("response:%s\n",text_response);
    json_t *root;
    json_error_t error;
    root = json_loads(text_response, 0, &error);
    free(text_response);

    if(!root)
```

```
    {
        fprintf(stderr, "error: on line %d: %s\n", error.line, error.text);
        throw 202;
    }

    if(!json_is_array(root))
    {
        fprintf(stderr, "error: root is not an object\n");
        json_decref(root);
        root_length=1;
    }

    root_length=json_array_size(root);
    printf("root_length:%d\n",root_length );
    //getting the actual data
    json_t *data, *time_stamp, *entry_id, *sensor, *sensor_value, *device_id;
    double timeStamp,entryID,sensorValue, deviceID;
    string sensor_name;
    for (i=0;i<root_length;i++){ //DEBUG i<root_length
        data = json_array_get(root, i);
        if(!json_is_object(data))
        {
            fprintf(stderr, "error: commit data %d is not an object\n", i + 1);
            json_decref(root);
            throw 202;
        }

        time_stamp = json_object_get(data,"timestamp");
        if (!json_is_string(time_stamp)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {

            timeStamp = convertStringToDouble(json_string_value(time_stamp));
            printf("timeStamp:%f\n",timeStamp );
        }

        entry_id = json_object_get(data,"_id");
        if (!json_is_string(entry_id)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {
            entryID =convertStringToDouble(json_string_value(entry_id));
        }
```

```
        sensor= json_object_get(data,"sensor");
        if (!json_is_string(sensor)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {
            sensor_name = json_string_value(sensor);
            printf("sensor_name:%s\n",sensor_name.c_str() );
        }

        const char* snsr_name = sensor_name.c_str();
        sensor_value = json_object_get(data,snsr_name);
        if (!json_is_string(sensor_value)){
            printf("throwing jsonException at sensor_value\n");
            throw 202;
        }
        else {
            sensorValue= convertStringToDouble(json_string_value(sensor_valu
            printf("sensor_value:%f\n",sensorValue);
        }

        device_id = json_object_get(data,"device_id");
        if (!json_is_string(device_id)){
            printf("throwing jsonException at device_id\n");
            throw 202;
        }
        else {
            deviceID = convertStringToDouble(json_string_value(device_id));
            printf("deviceID:%f\n",deviceID);
        }
        //put stuff in returning map
        data_map[sensor_name]=sensorValue;
    }
    return data_map;
}

void json_send_data(map<string,double> mymap){
    //printf("starting send_data\n");

    char url[URL_SIZE];


    string id_string = convertIntToString(myID);
    string http_path=PATH_DATA;
    http_path.append("client_"+id_string);
```

```
    string sensor_name;
    string key;
    double value;
    string value_string;
    string json_string;
    for (map<string,double>::iterator it=mymap.begin(); it!=mymap.end(); ++it)
    {
        key = it->first;
        value = it->second;
        value_string=convertDoubleToString(value);
        sensor_name=key;

        string http_path=PATH_DATA;
        http_path.append("client_"+id_string);
        http_path.append("/");
        http_path.append(sensor_name);
        json_string="{";
        json_string.append("\"");

        json_string.append(sensor_name);
        json_string.append("\"");
        json_string.append(":");
        json_string.append("␣");
        json_string.append("\""+value_string+"\""+"}");
        snprintf(url, URL_SIZE, URL_FORMAT, http_path.c_str());
        //printf("url:%s\n",url);
        //printf("json_string:%s\n",json_string.c_str());

        char *json_cstring = new char[json_string.length() + 1];
        strcpy(json_cstring, json_string.c_str());
        // do stuff

        http_post(url,json_cstring);
        free(json_cstring);
    }
}

void json_send_command(string cmd,int id){
    printf("starting␣send_commands\n");

    char url[URL_SIZE];
    string command=cmd;
    string http_path=PATH_COMMAND;
    string id_string = convertIntToString(id);
    http_path.append("client_"+id_string);
    string json_string;
```

```
    http_path=PATH_COMMAND;
    http_path.append("client_"+id_string);
    json_string="{";
    json_string.append("\"");

    json_string.append("command");
    json_string.append("\"");
    json_string.append(":");
    json_string.append("␣");
    json_string.append("\""+command+"\""+"}");
    snprintf(url, URL_SIZE, URL_FORMAT, http_path.c_str());
    printf("url:%s\n",url);
    printf("json_string:%s\n",json_string.c_str());

    char *json_cstring = new char[json_string.length() + 1];
    strcpy(json_cstring, json_string.c_str());
    // do stuff

    http_post(url,json_cstring);
    free(json_cstring);
}

vector<string> json_get_commands(int id){
    //printf("starting get_commands\n");
    vector<string> commands_vector;
    int root_length=0;
    char *text_response;
    char url[URL_SIZE];
    string id_path=PATH_COMMAND;

    string id_string = "client_"+convertIntToString(id);
    id_path.append(id_string);
    snprintf(url, URL_SIZE, URL_FORMAT, id_path.c_str());
    //printf("url:%s\n",url);

    text_response = http_request(url);
    //printf("response:%s\n",text_response);
    json_t *root;
    json_error_t error;
    root = json_loads(text_response, 0, &error);
    free(text_response);

    if(!root)
    {
        fprintf(stderr, "error:␣on␣line␣%d:␣%s\n", error.line, error.text);
        throw 202;
```

```
    }

    if(!json_is_array(root))
    {
        fprintf(stderr, "error: root is not an array\n");
        json_decref(root);
        root_length=1;
    }

    root_length=json_array_size(root);
    //printf("root_length:%d\n",root_length );
    //getting the actual data
    json_t *data, *time_stamp, *iterator;
    double timeStamp;
    string command="";
    for (i=0;i<root_length;i++){ //DEBUG i<root_length
        data = json_array_get(root, i);
        if(!json_is_object(data))
        {
            fprintf(stderr, "error: commit data %d is not an object\n", i + 1);
            json_decref(root);
            throw 202;
        }

        time_stamp = json_object_get(data,"timestamp");
        if (!json_is_string(time_stamp)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {

            timeStamp = convertStringToDouble(json_string_value(time_stamp));
            printf("timeStamp:%f\n",timeStamp );
        }
        iterator =json_object_get(data,"command");
        if (!json_is_string(iterator)){
            printf("throwing jsonException\n");
            throw 202;
        }
        else {
            command = json_string_value(iterator);
            //printf("command:%s\n",command.c_str());
        }
        commands_vector.push_back(command);
    }
    return commands_vector;
```

```
}


void json_test_function(){
    map<string,double> debug_map;
    debug_map["test1"]=8.9;
    debug_map["test2"]=5678.456;
    printf("Sending data\n");
    json_send_data(debug_map);
    printf("printing data\n");
    debug_print_map(json_get_data(testID));


    string command1="command_one";
    string command2="command_two";
    printf("sending commands\n");
    json_send_command(command1,testID);
    json_send_command(command2,testID);
    printf("printing commands\n");
    debug_print_vector(json_get_commands(testID));
}
```

1.19 http_ functions.h

```
#ifndef HTTP_FUNCTIONS
#include <stdlib.h>
#include <string.h>
#include <stdio.h>

// make HTTP request to url
char* http_request(char *url);

//make a HTTP post to url
void http_post(char* url, char* json_string);
#endif
```

1.20 http_ functions.cpp

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <curl/curl.h>
using namespace std;
```

```
#define BUFFER_SIZE  (256 * 1024)  /* 256 KB */

#define URL_FORMAT   "https://wodinaz.com/%s"
#define URL_SIZE     256

struct write_result
{
    char *data;
    int pos;
};

static size_t write_response(void *ptr, size_t size, size_t nmemb, void *stream)
{
    struct write_result *result = (struct write_result *)stream;

    if(result->pos + size * nmemb >= BUFFER_SIZE - 1)
    {
        fprintf(stderr, "error: too small buffer\n");
        return 0;
    }

    memcpy(result->data + result->pos, ptr, size * nmemb);
    result->pos += size * nmemb;

    return size * nmemb;
}

// make HTTP request to url
char* http_request(char *url)
{
    CURL *curl = NULL;
    CURLcode status;
    struct curl_slist *headers = NULL;
    char *data = NULL;
    long code;

    curl_global_init(CURL_GLOBAL_ALL);
    curl = curl_easy_init();
    if(!curl)
        goto error;

    data = (char*)malloc(BUFFER_SIZE);
    if(!data)
        goto error;

    struct write_result write_result;
```

```
    write_result.data=data;
    write_result.pos=0;

    curl_easy_setopt(curl, CURLOPT_URL, url);

    curl_easy_setopt(curl, CURLOPT_HTTPHEADER, headers);

    curl_easy_setopt(curl, CURLOPT_WRITEFUNCTION, write_response);
    curl_easy_setopt(curl, CURLOPT_WRITEDATA, &write_result);

    status = curl_easy_perform(curl);
    if(status != 0)
    {
        fprintf(stderr, "error: unable to request data from %s:\n", url);
        fprintf(stderr, "%s\n", curl_easy_strerror(status));
        goto error;
    }

    curl_easy_getinfo(curl, CURLINFO_RESPONSE_CODE, &code);
    if(code != 200)
    {
        fprintf(stderr, "error: server responded with code %ld\n", code);
        goto error;
    }

    curl_easy_cleanup(curl);
    curl_slist_free_all(headers);
    curl_global_cleanup();

    /* zero-terminate the result */
    data[write_result.pos] = '\0';

    return data;

error:
    if(data)
        free(data);
    if(curl)
        curl_easy_cleanup(curl);
    if(headers)
        curl_slist_free_all(headers);
    curl_global_cleanup();
    return NULL;
}

//post to server
```

```
void http_post(char* url, char* json_string){
      CURL *curl;
      CURLcode res;

       /* In windows, this will init the winsock stuff */
       curl_global_init(CURL_GLOBAL_ALL);
       /* get a curl handle */
       curl = curl_easy_init();
       if(curl) {
         /* First set the URL that is about to receive our POST. This URL can
            just as well be a https:// URL if that is what should receive the
            data. */
         curl_easy_setopt(curl, CURLOPT_URL, url);

         /* Now specify the POST data */
         curl_easy_setopt(curl, CURLOPT_POSTFIELDS, json_string);

         /* Perform the request, res will get the return code */
         res = curl_easy_perform(curl);

         /* Check for errors */
         if(res != CURLE_OK)
           fprintf(stderr, "curl_easy_perform() failed: %s\n",
                   curl_easy_strerror(res));


         //printf("return code:%d\n",res );
         /* always cleanup */
         curl_easy_cleanup(curl);
       }
       curl_global_cleanup();
}
```

Chapter 2

Example code

2.1 Car

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <time.h>
#include "car.h"

using namespace std;

//put ID of the wheels here
#define FRONT_RIGHT_WHEEL       1
#define BACK_RIGHT_WHEEL        3
#define FRONT_LEFT_WHEEL        0
#define BACK_LEFT_WHEEL         2


int main(){

        int deviceIndex = 0;
        int baudnum = 1;

        printf("--------CAR TEST PROGRAM--------\n");

        ///////// Open USB2Dynamixel ////////////
        if( dxl_initialize(deviceIndex, baudnum) == 0 )
        {
                printf( "Failed to open USB2Dynamixel!\n" );
                printf( "Press Enter key to terminate...\n" );
```

```
                getchar();
                return 0;
        }
        else
                printf( "Succeed to open USB2Dynamixel!\n" );


        Car car1(FRONT_RIGHT_WHEEL, FRONT_LEFT_WHEEL, BACK_RIGHT_WHEEL, BACK
        sleep(1);

        car1.setSpeed(1023,1);
        sleep(2);
        car1.setSpeed(1023,0);
        sleep(2);
        car1.setSpeed(0,1);

                while(1)
                {


                }

        // Close device
        car1.setSpeed(0,1);
        dxl_terminate();
        return 0;
}
```

2.2 Interface

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <time.h>
#include "car.h"
#include "manipulator.h"
#include "interface.h"

using namespace std;

//put ID of the wheels here
#define FRONT_RIGHT_WHEEL       1
#define BACK_RIGHT_WHEEL        3
#define FRONT_LEFT_WHEEL        0
#define BACK_LEFT_WHEEL         2
```

```
#define MAN_ONE                 4               //zero at 511
#define MAN_TWO                 7               //zero at 511, not allowed to go und
#define MAN_THREE               5               //zero at 511

#define GRIPPER_LEFT            12
#define GRIPPER_RIGHT           6

int main(){

        int deviceIndex = 0;
        int baudnum = 1;

        printf("--------LOCAL INTERFACE TEST PROGRAM--------\n");

        ///////// Open USB2Dynamixel ////////////
        if( dxl_initialize(deviceIndex, baudnum) == 0 )
        {
                printf( "Failed to open USB2Dynamixel!\n" );
                printf( "Press Enter key to terminate...\n" );
                getchar();
                return 0;
        }
        else
                printf( "Succeed to open USB2Dynamixel!\n" );

        windowInit();
        Car car1(FRONT_RIGHT_WHEEL, FRONT_LEFT_WHEEL, BACK_RIGHT_WHEEL, BACK_LEFT_WHE
        Manipulator manipulator1(MAN_ONE, MAN_TWO, MAN_THREE, GRIPPER_LEFT, GRIPPER_R
        sleep(1);

        manipulator1.goToPosition(XSTART,YSTART,ZSTART);
        manipulator1.setGripper(0);

                while(1)
                {

                        checkEvent(&manipulator1, &car1);

                }



        // Close device
        car1.setSpeed(0,1);
        dxl_terminate();
```

```
        return 0;
}
```

2.3 Main

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <pthread.h>
#include <vector>
#include <string>
#include <time.h>
#include "car.h"
#include "manipulator.h"
#include "json_processing.h"
#include "sensor.h"

using namespace std;

//ID of wheels
#define FRONT_RIGHT_WHEEL       1
#define BACK_RIGHT_WHEEL        3
#define FRONT_LEFT_WHEEL        0
#define BACK_LEFT_WHEEL         2

//ID of manipulator arm
#define MAN_ONE                 4               //zero at 511
#define MAN_TWO                 7               //zero at 511, not allowed t
#define MAN_THREE               5               //zero at 511

//ID of gripper
#define GRIPPER_LEFT            12
#define GRIPPER_RIGHT           6

//ID of sensor
#define SENSOR                  100

void *sendSensorData(void *ptr);

int main(){

        pthread_t thread1;
        int deviceIndex = 0;
        int baudnum = 1;
        string command;
```

```
vector <string> commands;
string strCheck = "position";

printf("--------MAIN PROGRAM--------\n");

//////// Open USB2Dynamixel ///////////
if( dxl_initialize(deviceIndex, baudnum) == 0 )
{
        printf( "Failed to open USB2Dynamixel!\n" );
        printf( "Press Enter key to terminate...\n" );
        getchar();
        return 0;
}
else
        printf( "Succeed to open USB2Dynamixel!\n" );

Car car1(FRONT_RIGHT_WHEEL, FRONT_LEFT_WHEEL, BACK_RIGHT_WHEEL, BACK_LEFT_WHE
Manipulator manipulator1(MAN_ONE, MAN_TWO, MAN_THREE, GRIPPER_LEFT, GRIPPER_R
Sensor sensor1(SENSOR);
sleep(1);

sensor1.playMelody(6);
manipulator1.goToPosition(XSTART,YSTART,ZSTART);
manipulator1.setGripper(0);

//get old commands from server and disregard them
vector <string> dummy = json_get_commands(0);

//create thread for sending sensor data
pthread_create( &thread1, NULL, sendSensorData, &sensor1 );

        while(1)
        {

                        //get commands
                        while(commands.empty())
                        {
                                commands = json_get_commands(0);
                        }

                        //execute commands
                        while(!commands.empty())
                        {
                                command = commands.front();
                                commands.erase(commands.begin());
                                if(command == "forward")
```

```
                car1.setSpeed(1023,1);

            else if(command == "backward")
                car1.setSpeed(1023,0);

            else if(command == "stop")
                car1.setSpeed(0,1);

            else if(command == "leftTurn")
                car1.turnCar(LEFT_TURN);

            else if(command == "rightTurn")
                car1.turnCar(RIGHT_TURN);

            else if(command == "noTurn")
                car1.turnCar(NO_TURN);

            else if(command == "gripClose")
                manipulator1.setGripper(1);

            else if(command == "gripOpen")
                manipulator1.setGripper(0);

            else if(command.find(strCheck) != st
                size_t found1 = command.find
                size_t found2 = command.find
                size_t found3 = command.find
                string nr1 = command.substr(
                string nr2 = command.substr(
                string nr3 = command.substr(

                int x = atoi(nr1.c_str());
                int y = atoi(nr2.c_str());
                int z = atoi(nr3.c_str());
                manipulator1.goToPosition(x,
            }

            else
                printf("Unknown command\n");

            printf("command: %s\n", command.c_st
        }

    }
```

```
        // Close device
        car1.setSpeed(0,1);
        dxl_terminate();
        return 0;
}

//thread function for continously sending data
void *sendSensorData(void *ptr){

        //initialize sensor here?
        Sensor* p = (Sensor*)ptr;
        int data;
        map <string,double> sensorData;
        while(1){
                //sleep for 100ms
                sleep(1);

                if(p->getMode() == FAILSAFE_MODE)
                {
                        p->ping();
                        continue;
                }
                //get data and put it in the map
                data = p->getIR(CENTER);
                printf("\nIR center: %d\n",data);
                sensorData["IR center"] = data;

                data = p->getIR(LEFT);
                printf("IR left: %d\n",data);
                sensorData["IR left"] = data;

                data = p->getIR(RIGHT);
                printf("IR right: %d\n",data);
                sensorData["IR right"] = data;
                //send data
                json_send_data(sensorData);
                //clear map
                sensorData.clear();
        }
        return NULL;
}
```

2.4 Manipulator

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include <time.h>
#include "manipulator.h"

using namespace std;

#define MAN_ONE                 4               //zero at 511
#define MAN_TWO                 7               //zero at 511, not allowed t
#define MAN_THREE               5               //zero at 511

#define GRIPPER_LEFT            12
#define GRIPPER_RIGHT           6

int main(){

        int deviceIndex = 0;
        int baudnum = 1;

        printf("--------MANIPULATOR_TEST_PROGRAM--------\n");

        ///////// Open USB2Dynamixel ////////////
        if( dxl_initialize(deviceIndex, baudnum) == 0 )
        {
                printf( "Failed to open USB2Dynamixel!\n" );
                printf( "Press Enter key to terminate...\n" );
                getchar();
                return 0;
        }
        else
                printf( "Succeed to open USB2Dynamixel!\n" );

        Manipulator manipulator1(MAN_ONE, MAN_TWO, MAN_THREE, GRIPPER_LEFT,
        sleep(1);

        manipulator1.setGripper(0);

        //test drawing
//      manipulator1.setGripper(1);
//      manipulator1.drawLine(50,200,50,150,0);
//      manipulator1.drawLine(50,175,25,175,0);
//      manipulator1.drawLine(25,200,25,150,0);

        while(1)
```

```
        {

                for(int i = 0; i < 130; i+=1)
                {
                        manipulator1.goToPosition(0,170,i);
                        usleep(5000);
                }
                for(int i = 130; i > 0; i-=1)
                {
                        manipulator1.goToPosition(0,170,i);
                        usleep(5000);
                }

                for(int i = 0; i < 100; i+=1)
                {
                        manipulator1.goToPosition(i,170,0);
                        usleep(5000);
                }
                for(int i = 100; i > -100; i-=1)
                {
                        manipulator1.goToPosition(i,170,0);
                        usleep(5000);
                }
                for(int i = -100; i < 0; i+=1)
                {
                        manipulator1.goToPosition(i,170,0);
                        usleep(5000);
                }

        }



        // Close device
        dxl_terminate();
        return 0;
}
```

2.5 Motor

```
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>
#include "motor.h"
```

```
using namespace std;

#define MOTOR_ID                1

int main(){

        bool b = 0;
        int deviceIndex = 0;
        int baudnum = 1;

        printf("--------MOTOR_TEST_PROGRAM--------\n");

        //////// Open USB2Dynamixel ////////////
        if( dxl_initialize(deviceIndex, baudnum) == 0 )
        {
                printf( "Failed to open USB2Dynamixel!\n" );
                printf( "Press Enter key to terminate...\n" );
                getchar();
                return 0;
        }
        else
                printf( "Succeed to open USB2Dynamixel!\n" );


        Motor motor1(MOTOR_ID, SERVOMODE);

        while(1)
        {
                try{
                        printf( "Press Enter key to continue!(press ESC and
                        if(getchar() == 0x1b)
                                break;

                        if(b){
                                printf("motor1 to 300 degrees\n");
                                motor1.setGoalPosition(1023);
                        }

                        else{
                                printf("motor1 to 30 degrees\n");
                                motor1.setGoalPosition(0);
                        }


                        b ^= 1;             //change b
                }
```

```
                catch(MotorException e) {
                        printf("ID: %d lost\n",e.ID);
                        printError(e.status);
                        break;
                }

        }



        // Close device
        dxl_terminate();
        return 0;
}
```

2.6 ReadWrite

```
//////////////////////////////////////////////////////////////////
//##                        R O B O T I S
##
//##          ReadWrite Example code for Dynamixel.
##
//##                                                   2009.11.10 ##
//////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <termio.h>
#include <unistd.h>
#include <dynamixel.h>

// Control table address
#define P_GOAL_POSITION_L       30
#define P_GOAL_POSITION_H       31
#define P_PRESENT_POSITION_L    36
#define P_PRESENT_POSITION_H    37
#define P_MOVING                46

// Defulat setting
#define DEFAULT_BAUDNUM         1 // 1Mbps
#define DEFAULT_ID              1

void PrintCommStatus(int CommStatus);
void PrintErrorCode(void);

int main()
{
        int baudnum = 1;
```

```
int GoalPos[2] = {0, 1023};
//int GoalPos[2] = {0, 4095}; // for Ex series
int index = 0;
int deviceIndex = 0;
int Moving, PresentPos;
int CommStatus;

printf( "\n\nRead/Write example for Linux\n\n" );
/////////// Open USB2Dynamixel ////////////
if( dxl_initialize(deviceIndex, baudnum) == 0 )
{
        printf( "Failed to open USB2Dynamixel!\n" );
        printf( "Press Enter key to terminate...\n" );
        getchar();
        return 0;
}
else
        printf( "Succeed to open USB2Dynamixel!\n" );

while(1)
{
        printf( "Press Enter key to continue!(press ESC and Enter to
        if(getchar() == 0x1b)
                break;

        // Write goal position
        dxl_write_word( DEFAULT_ID, P_GOAL_POSITION_L, GoalPos[index
        do
        {
                // Read present position
                PresentPos = dxl_read_word( DEFAULT_ID, P_PRESENT_P
                CommStatus = dxl_get_result();

                if( CommStatus == COMM_RXSUCCESS )
                {
                        printf( "%d   %d\n",GoalPos[index], PresentP
                        PrintErrorCode();
                }
                else
                {
                        PrintCommStatus(CommStatus);
                        break;
                }

                // Check moving done
                Moving = dxl_read_byte( DEFAULT_ID, P_MOVING );
```

```
                        CommStatus = dxl_get_result();
                        if( CommStatus == COMM_RXSUCCESS )
                        {
                                if( Moving == 0 )
                                {
                                        // Change goal position
                                        if( index == 0 )
                                                index = 1;
                                        else
                                                index = 0;
                                }

                                PrintErrorCode();
                        }
                        else
                        {
                                PrintCommStatus(CommStatus);
                                break;
                        }
                }while(Moving == 1);
        }

        // Close device
        dxl_terminate();
        printf( "Press Enter key to terminate...\n" );
        getchar();
        return 0;
}
// Print communication result
void PrintCommStatus(int CommStatus)
{
        switch(CommStatus)
        {
        case COMM_TXFAIL:
                printf("COMM_TXFAIL: Failed transmit instruction packet!\n");
                break;

        case COMM_TXERROR:
                printf("COMM_TXERROR: Incorrect instruction packet!\n");
                break;

        case COMM_RXFAIL:
                printf("COMM_RXFAIL: Failed get status packet from device!\n");
                break;

        case COMM_RXWAITING:
```

```
                printf("COMM_RXWAITING: Now recieving status packet!\n");
                break;

        case COMM_RXTIMEOUT:
                printf("COMM_RXTIMEOUT: There is no status packet!\n");
                break;

        case COMM_RXCORRUPT:
                printf("COMM_RXCORRUPT: Incorrect status packet!\n");
                break;

        default:
                printf("This is unknown error code!\n");
                break;
        }
}

// Print error bit of status packet
void PrintErrorCode()
{
        if(dxl_get_rxpacket_error(ERRBIT_VOLTAGE) == 1)
                printf("Input voltage error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_ANGLE) == 1)
                printf("Angle limit error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_OVERHEAT) == 1)
                printf("Overheat error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_RANGE) == 1)
                printf("Out of range error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_CHECKSUM) == 1)
                printf("Checksum error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_OVERLOAD) == 1)
                printf("Overload error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_INSTRUCTION) == 1)
                printf("Instruction code error!\n");
}
```

2.7 Sensor

```
#include <stdio.h>
#include <termio.h>
```

```
#include <unistd.h>
#include <dynamixel.h>
#include "sensor.h"
#include "songs.h"


using namespace std;

#define SENSOR            100

int main(){

        int deviceIndex = 0;
        int baudnum = 1;

        printf("--------Sensor TEST PROGRAM--------\n");

        //////// Open USB2Dynamixel ///////////
        if( dxl_initialize(deviceIndex, baudnum) == 0 )
        {
                printf( "Failed to open USB2Dynamixel!\n" );
                printf( "Press Enter key to terminate...\n" );
                getchar();
                return 0;
        }
        else
                printf( "Succeed to open USB2Dynamixel!\n" );


        Sensor sensor1(SENSOR);
        sensor1.playMelody(FurElise,sizeof(FurElise));
        //sensor1.playMelody(Sirene,sizeof(Sirene));
        //sensor1.playMelody(6);

        while(1)
        {

        }

        // Close device
        dxl_terminate();
        return 0;
}
```

2.8 SyncWrite

```
//////////////////////////////////////////////////////////////////
//##                        R O B O T I S                     ##
//##          SyncWrite Example code for Dynamixel.           ##
//##                                                2009.11.10 ##
//////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <unistd.h>
#include <math.h>
#include <termio.h>

#include <dynamixel.h>

#define PI          3.141592f
#define NUM_ACTUATOR            3

// Control table address
#define P_GOAL_POSITION_L       30
#define P_GOAL_POSITION_H       31
#define P_GOAL_SPEED_L          32
#define P_GOAL_SPEED_H          33

// Defulat setting
#define DEFAULT_BAUDNUM         1 // 1Mbps
#define NUM_ACTUATOR            3 // Number of actuator
#define STEP_THETA                      (PI / 100.0f) // Large value is mor
#define CONTROL_PERIOD          (10000) // usec (Large value is more slow)

void PrintCommStatus(int CommStatus);
void PrintErrorCode(void);

int main()
{
        int id[NUM_ACTUATOR];
        int baudnum = 1;
        int deviceIndex = 0;
        float phase[NUM_ACTUATOR];
        float theta = 0;
        int AmpPos = 512;
        //int AmpPos = 2048; // for EX series
        int GoalPos;
        int i;
        int CommStatus;
        printf( "\n\nSyncWrite example for Linux\n\n" );
```

```
// Initialize id and phase
for( i=0; i<NUM_ACTUATOR; i++ )
{
        id[i] = i+1;
        phase[i] = 2*PI * (float)i / (float)NUM_ACTUATOR;
}

//////// Open USB2Dynamixel ///////////
if( dxl_initialize(deviceIndex, baudnum) == 0 )
{
        printf( "Failed to open USB2Dynamixel!\n" );
        printf( "Press Enter key to terminate...\n" );
        getchar();
        return 0;
}
else
        printf( "Succeed to open USB2Dynamixel!\n" );

// Set goal speed
dxl_write_word( BROADCAST_ID, P_GOAL_SPEED_L, 0 );
// Set goal position
dxl_write_word( BROADCAST_ID, P_GOAL_POSITION_L, AmpPos );

while(1)
{
        printf( "Press Enter key to continue!(press ESC and Enter to quit)\n"
        if(getchar() == 0x1b)
                break;

        theta = 0;
        do
        {
                // Make syncwrite packet
                dxl_set_txpacket_id(BROADCAST_ID);
                dxl_set_txpacket_instruction(INST_SYNC_WRITE);
                dxl_set_txpacket_parameter(0, P_GOAL_POSITION_L);
                dxl_set_txpacket_parameter(1, 2);
                for( i=0; i<NUM_ACTUATOR; i++ )
                {
                        dxl_set_txpacket_parameter(2+3*i, id[i]);
                        GoalPos = (int)((sin(theta+phase[i]) + 1.0) * (double
                        printf( "%d  ", GoalPos );
                        dxl_set_txpacket_parameter(2+3*i+1, dxl_get_lowbyte(G
                        dxl_set_txpacket_parameter(2+3*i+2, dxl_get_highbyte(
                }
                dxl_set_txpacket_length((2+1)*NUM_ACTUATOR+4);
```

```
                        printf( "\n" );

                        dxl_txrx_packet();
                        CommStatus = dxl_get_result();
                        if( CommStatus == COMM_RXSUCCESS )
                        {
                                PrintErrorCode();
                        }
                        else
                        {
                                PrintCommStatus(CommStatus);
                                break;
                        }

                        theta += STEP_THETA;
                        usleep(CONTROL_PERIOD);

                }while(theta < 2*PI);
        }

        dxl_terminate();
        printf( "Press Enter key to terminate...\n" );
        getchar();

        return 0;
}

// Print communication result
void PrintCommStatus(int CommStatus)
{
        switch(CommStatus)
        {
        case COMM_TXFAIL:
                printf("COMM_TXFAIL: Failed transmit instruction packet!\n"
                break;

        case COMM_TXERROR:
                printf("COMM_TXERROR: Incorrect instruction packet!\n");
                break;

        case COMM_RXFAIL:
                printf("COMM_RXFAIL: Failed get status packet from device!\n
                break;
```

```
        case COMM_RXWAITING:
                printf("COMM_RXWAITING: Now recieving status packet!\n");
                break;

        case COMM_RXTIMEOUT:
                printf("COMM_RXTIMEOUT: There is no status packet!\n");
                break;

        case COMM_RXCORRUPT:
                printf("COMM_RXCORRUPT: Incorrect status packet!\n");
                break;

        default:
                printf("This is unknown error code!\n");
                break;
        }
}

// Print error bit of status packet
void PrintErrorCode()
{
        if(dxl_get_rxpacket_error(ERRBIT_VOLTAGE) == 1)
                printf("Input voltage error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_ANGLE) == 1)
                printf("Angle limit error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_OVERHEAT) == 1)
                printf("Overheat error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_RANGE) == 1)
                printf("Out of range error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_CHECKSUM) == 1)
                printf("Checksum error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_OVERLOAD) == 1)
                printf("Overload error!\n");

        if(dxl_get_rxpacket_error(ERRBIT_INSTRUCTION) == 1)
                printf("Instruction code error!\n");
}
```