Санкт-Петербургский политехнический университет Петра Великого
Институт машиностроения, материалов и транспорта
Высшая школа автоматизации и робототехники

К У Р С О В А Я Р А Б О Т А

Дисциплина: объектно-ориентированное программирование

Тема: разработка программы для управления манипулятором «OmegaMan»

Выполнили

Студенты группы 3331506/00401 _____А. С. Кондратьев

Преподаватель _____М. С. Ананьевский

Санкт-Петербург

2023

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ МАНИПУЛЯТОРА

Манипулятор «OmegaMan» состоит из металлических звеньев, перемещающихся за счет севроприводов. Крайнее звено древовидной структуры манипулятора является схватом, который способен удерживать объекты массой до 100 г.

Управляющей частью манипулятора является платформа OpenCM9.04 (аналог Arduino Nano). Данная платформа поддерживается в среде разработки ArduinoIDE. Кроме того, платформа *OpenCM*9.04 поддерживает *UART*, что позволяет управлять манипулятором с персонального компьютера.

Для более информативного управления манипулятором было принято решение использовать библиотеку *ncurses* для организации информативного вывода параметров манипулятора в терминал компьютера.

Таким образом, необходимо разработать программу, способную передавать команду управления на манипулятор по *UART* и выводить актуальные значения параметров манипулятора в терминал.

ОПИСАНИЕ ПРОГРАММЫ

Во время работы программы пользователь вводит с клавиатуры команду, которая с помощью модуля *graphics* записывается класс *str*, после чего класс *Connect* обрабатывает команду и при отсутствии синтаксических ошибок кодирует ее с помощью алгоритма crc8 и отправляет на платформу *OpenCM*9.04. Управление графической составляющей предоставляет модуль *graphics*. Хранение команды во время обработки осуществляет класс *str*. Хранение истории ввода обеспечивают классы *History* и *List*. Хранение передаваемых с манипулятора параметров осуществляется за счет класса *Gservo*.

ИНСТРУКЦИЯ ПО РАБОТЕ С ПРОГРАММОЙ

1. Ограничения

Манипулятор "OmegaMan" имеет 4 вращательные кинематические пары, каждая кинематическая пара имеет свои ограничения, которые задаются в следующем диапазоне: 0 -- 1023 (0 -- 5П/3). Ограничения записаны в файле Arduino/Config.h. Для нормальной работы программы изменять эти ограничения запрещено!

Согласно описанию манипулятора, схват выдерживает полезную нагрузку массой 100г.
В реальности он может выдержать без отключения только один цветной кубик из аудитории В2.15. При подъеме груза массой выше кубика возможно отключение одного или нескольких сервоприводов манипулятора.

2. Описание программы для платформы OpenCM9.04

Платформа OpenCM9.04 можно программировать в ArduinoIDE. Детали подключения
необходимых библиотек описаны в [3]. Наша программа в ArduinoIDE разделена на несколько файлов: Arduino.ino - основной файл, в котором описываются функции setup() и loop(); Config.h - здесь описаны все константы, необходимые для корректной работы программы; Connection.h - содержит протокол для общения с ПК через терминал; Joint.h - модуль для расчета геометрических характеристик манипулятора; Servo.h - модуль для взаимодействия с сервоприводами манипулятора.
Модуль Calibration.h содержит экспериментальные возможности для калибровки манипулятора. Для обеспечения нормальной работы робота использовать данный модуль не рекомендуется. Далее представлено более подробное описание методов, позволяющих управлять манипулятором.

2.1. Servo.h

Существующие объекты:
servo1
servo2
servo3
servo4

Методы:
obj.set_angle(uint16_t _angle) - задать сервоприводу значение угла;
obj.set_speed(uint16_t _speed) - задать сервоприводу скорость перемещения;
Servo::setStartPosition() - Возвращает манипулятор в стартовое положение;
toolPush() - схватить объект;
toolPop() - отпустить объект;

obj.get_DXL_ID() - возвращает значение DXL_ID сервопривода;
obj.get_angle() - возвращает реальное значение угла;
obj.get_goal() - возвращает знчение угла, переданное в качестве целевого;
obj.get_load() - возвращает текущую нагрузку;
obj.is_moving() - 1 - сервопривод двигается, 2 - не двигается;
obj.get_speed() - возвращает значение заданной скорости;

Отдельного рассмотрения требует метод Servo::mv(uint16_t msg). Для его работы в функции loop следует раскомментировать метод Servo::mv(Serial.parseInt()). На вход данный метод принимает 5-тиразрядное число. Старший разряд кодирует DXL_ID
сервопривода, остальные разряды кодируют значение угла (0 -- 1023). Таким образом, для задания 1-му сервоприводу угла 256 необходимо в Serial Monitor ввести число 10256; для задания 3-му сервоприводу угла 1023 необходимо ввести число 31023; для задания 4-му сервоприводу угла 0 необходимо ввести 40000.

Обращаем внимание, что перемещения сервоприводов будут выполнены с учетом ограничений минимальных и максимальных значений угла. Важно помнить, что данный
метод может использоваться только для отладки и не имеет совместимости с классом
Connection, поэтому совместное их использование приведет к некорректной работе программы.

2.2. Connection.h

Класс позволяет обмениваться сообщениями с ПК через UART. Скорость передачи сообщений составляет 9600 baud. Интерфейс со стороны ПК будет описан позднее.

Для корректной работы класса необходимо в функции loop() раскомментировать метод
Connection::receiveCommand(). Кроме того, во время работы программы с использованием класса Connection запрещается использование методов класса Serial и метода Servo::mv(uint16_t msg). Данный класс работает в автономном режиме, и единственное, что может требовать корректировки - метод Connection::findCommand(), который позволяет принимать с ПК новые задания, созданные пользователем.

Для добавления нового задания следует выполнить 2 шага:
а) в файл Config.h добавить строку следующего вида
#define NEW_TASK X
где X - id нового задания (должен быть уникальным, минимальное значение - 6, максимальное - 99).
б) в метод Connection::findCommand() добавить следующий блок

```
if (com == NEW_TASK) {
    return new_function(value);
```

}

3. Описание программы для ПК

Данная программа позволяет оценивать параметры манипулятора в режиме реального
времени и управлять манипулятором через терминал без использования ArduinoIDE.

Установка данной программы возможна только на ПК под управлением ОС Linux с установленной библиотекой "ncurses". для компиляции программы необходимо выполнить следующие шаги:
$ mkdir App/build && cd App/build
$ cmake ../CMakeLists.txt
$ make -j4
После чего в директории App/build будет доступен исполняемый файл Manipulator.
Для запуска программы необходимо перейти в директорию App/build и ввести в терминал следующую команду:
$ ./Manipulator

При запуске программы в случае неудачной попытки подключения ПК к OpenCM9.04
будет выведена следующее сообщение:
Unable to connect

Ввод команды осуществляется за счет нажатия на клавиши клавиатуры. В случае нажатия клавиш Backspace и Delete будет произведено удаление соответствующего введенного символа. В случае нажатия клавиши Enter будет произведена обработка команды. Клавиши KEY_LEFT и KEY_RIGHT позволяют перемещаться по введенной

команде. Клавиши KEY_UP и KEY_DOWN позволяют просматривать историю введенных в
рамках текущего сеанса команд. Для выхода из программы необходимо нажать комбинацию клавиш Ctrl + C.

Для успешной отправки команды в случае, если в программе для платформы ArduinoIDE задание уже существует, достаточно ввести в командную строку данной программы следующее сообщение
STTVVVV
где S - DXL_ID сервопривода, TT - задание, VVVV - value (константа, в случае отсутствия необходимости ее использования необходимо ввести сообщение следующего вида: STT0000).

Помимо числовых команд поддерживаются текстовые команды:
home - возврат манипулятора в стартовое положение;
push - схватить объект;
pop - отпустить объект.

Для добавления пользовательских числовых команд достаточно добавить соответствующую команду в программу для платформы OpenCM9.04 по алгоритму, описанному выше.

Для добавления текстовой команды необходимо добавить соответствующую числовую
команду, в класс Connect (файл App/Connect.h) добавить метод, заполняющий массив
Connect::command соответствующим образом, в метод Connect::decodeKeyInput() добавить обработку текстовой команды. Подробнее см. обработку текстовой команды
"home" в App/Connect.cpp.

ПРИЛОЖЕНИЕ 1. Exception.h

```
//
// Created by user on 3/23/23.
//

#ifndef MANIPULATOR_EXCEPTION_H
#define MANIPULATOR_EXCEPTION_H


#include <string>


class Exception : std::exception {
private:
    std::string message;
public:
    explicit Exception(std::string _message) {message = std::move(_message);};

    [[maybe_unused]] std::string getMessage() const {return message;};
};


#endif //MANIPULATOR_EXCEPTION_H
```

ПРИЛОЖЕНИЕ 2. Connect.h

```
//
// Created by user on 05.03.23.
//

#ifndef MANIPULATOR_CONNECT_H
#define MANIPULATOR_CONNECT_H


#include <chrono>
#include <cstring>
#include <fcntl.h>
#include <iostream>
#include <string>
#include <termios.h>
#include <unistd.h>
#include "../Arduino/Config.h"
#include "Exception.h"
#include "Gservo.h"
#include "str.h"


class Connect {
private:
    inline static int Arduino = open("/dev/ttyACM0", O_RDWR | O_NOCTTY | O_NONBLOCK);
    inline static uint8_t command[COMMAND_SIZE];
    inline static uint8_t message[MESSAGE_SIZE];

public:
    inline static str key_cmd;

private:
    static void resetCommand();

    static bool openArduino();

public:
    static bool setConnection();
    static void disconnectArduino();

private:
    static uint8_t crc8(const uint8_t pocket[], uint64_t size);
    static void calcCommandCheckSum();
    static uint8_t calcMessageCheckSum(uint8_t buffer[]);

public:
    static void sendCommand();
```

```
private:
    static void setId(uint8_t id);
    static void setTask(uint8_t task);
    static void setValue(uint16_t value);
    static void encodeCommand(uint64_t cmd);

    static void decodeMessage();
    static Gservo* findGservo(uint8_t id);

public:
    static bool receiveMessage();

private:
    static uint64_t checkNumberCommand();

public:
    static void toolPush();
    static void toolPop();
    static void goHome();

    static void decodeKeyInput();
};


inline struct termios SerialPortSettings;


#endif //MANIPULATOR_CONNECT_H
```

ПРИЛОЖЕНИЕ 3. Connect.cpp

```
//
// Created by user on 05.03.23.
//

#include "Connect.h"


bool Connect::openArduino() {
    if (Arduino == -1) {
        Arduino = open("/dev/ttyACM1", O_RDWR | O_NOCTTY | O_NONBLOCK);
        if (Arduino == -1) {
            return false;
        }
    }

    tcgetattr(Arduino, &SerialPortSettings);

    SerialPortSettings.c_cflag |= (CLOCAL | CREAD);    // Ignore modem controls
    SerialPortSettings.c_cflag &= ~CSIZE;
    SerialPortSettings.c_cflag |= CS8;     // 8 bit chars
    SerialPortSettings.c_cflag &= ~(PARENB | PARODD);  // shut off parody
    SerialPortSettings.c_cflag &= ~CSTOPB; //no scts stop
    SerialPortSettings.c_iflag &= ~IGNBRK; //disable break processing
    SerialPortSettings.c_iflag = 0;        // no echo
    SerialPortSettings.c_iflag &= ~(IXON | IXOFF | IXANY); // no software flow control
    SerialPortSettings.c_oflag = 0;        // no remapping
    SerialPortSettings.c_lflag &= ~(ECHO | ECHONL | ICANON | ISIG |IEXTEN);
    SerialPortSettings.c_cc[VMIN] = 0;     // read doesn't block
    SerialPortSettings.c_cc[VTIME] = 0;    // 0s read timeout

    tcsetattr(Arduino,TCSANOW,&SerialPortSettings);

    return true;
}


void Connect::resetCommand() {
    command[COMMAND_START_BYTE1_CELL] = START_BYTE;
    command[COMMAND_START_BYTE2_CELL] = START_BYTE;
    command[COMMAND_ID_CELL] = PING_DXL_ID;
    command[COMMAND_TASK1_CELL] = PING_TASK;
    command[COMMAND_TASK2_CELL] = PING_TASK;
    command[COMMAND_VALUE1_CELL] = PING_VALUE1;
    command[COMMAND_VALUE2_CELL] = PING_VALUE2;
    calcCommandCheckSum();
}
```

```
bool Connect::setConnection() {
    if (!openArduino()) {
        std::cout << "Unable to connect" << std::endl;
        return false;
    }

    resetCommand();
    bool message_flag = false;
    auto start_timer = std::chrono::system_clock::now();
    while (!message_flag) {
        auto end_timer = std::chrono::system_clock::now();
        if (std::chrono::duration_cast<std::chrono::milliseconds>(end_timer - start_timer).count() >
int(TIMER)) {
            sendCommand();
            message_flag = receiveMessage();
            start_timer = std::chrono::system_clock::now();
        }
    }
    std::cout << "connected" << std::endl;
    sleep(1);
    return true;
}


void Connect::disconnectArduino() {
    close(Arduino);
}


uint8_t Connect::crc8(const uint8_t pocket[], uint64_t size) {
    uint8_t BYTE_SIZE = 8;
    uint8_t MSB_MASK = 0x80;
    uint8_t byte;
    uint8_t POLY = 0x7;
    uint8_t crc8 = 0xFF;

    for (int cell = 0; cell < size; cell++) {

        byte = pocket[cell];
        crc8 = crc8 ^ byte;

        for (int byte_number = 0; byte_number < BYTE_SIZE; byte_number++) {

            if (crc8 & MSB_MASK) {
                crc8 = (crc8 << 1) ^ POLY;
            }
```

```
            else {
                crc8 = crc8 << 1;
            }
        }
    }
    return crc8;
}


void Connect::calcCommandCheckSum() {
    command[COMMAND_CHECKSUM_CELL] = crc8(command, COMMAND_SIZE - 1);
}


uint8_t Connect::calcMessageCheckSum(uint8_t buffer[]) {
    return crc8(buffer, MESSAGE_SIZE);
}


void Connect::sendCommand() {
    if (!openArduino()) {
        return;
    }

    calcCommandCheckSum();
    write(Arduino, command, COMMAND_SIZE);
    resetCommand();
}


void Connect::setId(uint8_t id) {
    command[COMMAND_ID_CELL] = id;
}


void Connect::setTask(uint8_t task) {
    command[COMMAND_TASK1_CELL] = task / 10;
    command[COMMAND_TASK2_CELL] = task % 10;
}


void Connect::setValue(uint16_t value) {
    command[COMMAND_VALUE1_CELL] = uint8_t(value / 100);
    command[COMMAND_VALUE2_CELL] = uint8_t(value % 100);
}


void Connect::encodeCommand(uint64_t cmd) {
```

```
    auto id = static_cast<uint8_t>(cmd / 1000000);
    setId(id);

    auto task = static_cast<uint8_t>((cmd % 1000000) / 10000);
    setTask(task);

    uint16_t value = cmd % 10000;
    setValue(value);
}


Gservo* Connect::findGservo(uint8_t id) {
    if (id == 1) {
        return &gservo1;
    }
    if (id == 2) {
        return &gservo2;
    }
    if (id == 3) {
        return &gservo3;
    }
    if (id == 4) {
        return &gservo4;
    }
}


void Connect::decodeMessage() {
    Gservo* gservo = findGservo(message[MESSAGE_ID_CELL]);
    gservo->set_goal(message[MESSAGE_GOAL1_CELL], message[MESSAGE_GOAL2_CELL]);
    gservo->set_angle(message[MESSAGE_ANGLE1_CELL],
message[MESSAGE_ANGLE2_CELL]);
    gservo->set_speed(message[MESSAGE_SPEED1_CELL],
message[MESSAGE_SPEED2_CELL]);
    gservo->set_torque(message[MESSAGE_TORQUE1_CELL],
message[MESSAGE_TORQUE2_CELL]);
    gservo->set_is_moving(message[MESSAGE_IS_MOVING_CELL]);
    Gservo::set_x(message[MESSAGE_X1_CELL], message[MESSAGE_X2_CELL],
message[MESSAGE_X_SIGN]);
    Gservo::set_y(message[MESSAGE_Y1_CELL], message[MESSAGE_Y2_CELL],
message[MESSAGE_Y_SIGN]);
    Gservo::set_z(message[MESSAGE_Z1_CELL], message[MESSAGE_Z2_CELL],
message[MESSAGE_Z_SIGN]);
    Gservo::set_q0(message[MESSAGE_Q01_CELL], message[MESSAGE_Q02_CELL]);
    Gservo::set_q1(message[MESSAGE_Q11_CELL], message[MESSAGE_Q12_CELL]);
    Gservo::set_q2(message[MESSAGE_Q21_CELL], message[MESSAGE_Q22_CELL]);
}
```

```
bool Connect::receiveMessage() {
    if (!openArduino()) {
        return false;
    }

    uint8_t buf[MESSAGE_SIZE];
    read(Arduino, buf, MESSAGE_SIZE);

    if (buf[MESSAGE_START_BYTE1_CELL] == START_BYTE &&
buf[MESSAGE_START_BYTE2_CELL] == START_BYTE) {
        if (!calcMessageCheckSum(buf)) {
            std::memcpy(message, buf, sizeof(uint8_t) * MESSAGE_SIZE);
            Connect::decodeMessage();
            memset(buf,0,MESSAGE_SIZE);
            return true;
        }
    }
    return false;
}


uint64_t Connect::checkNumberCommand() {
    uint8_t numbers[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
    uint64_t flag = 0;
    for (int i = 0; i < key_cmd.size(); i++) {
        for (uint8_t number: numbers) {
            if (key_cmd.get_str()[i] == number) {
                flag++;
                break;
            }
        }
    }
    return flag;
}


void Connect::toolPush() {
    resetCommand();
    setId(DXL_ID4);
    setTask(TOOL_PUSH_TASK);
}


void Connect::toolPop() {
    resetCommand();
    setId(DXL_ID4);
    setTask(TOOL_POP_TASK);
```

```
}


void Connect::goHome() {
    setTask(GO_HOME_TASK);
}


void Connect::decodeKeyInput() {

    if (checkNumberCommand() == key_cmd.size()) {
        Connect::encodeCommand(stoi(key_cmd.get_str()));
        return;
    }
    if (key_cmd.get_str() == "push") {
        return toolPush();
    }
    if (key_cmd.get_str() == "pop") {
        return toolPop();
    }
    if (key_cmd.get_str() == "home") {
        return goHome();
    }
}
```

ПРИЛОЖЕНИЕ 4. List.h

```
//
// Created by user on 11.02.23.
//

#ifndef HELLO_WORLD_LIST_H
#define HELLO_WORLD_LIST_H


#include <cstdint>
#include <iostream>
#include <utility>
#include "Exception.h"
#include "str.h"


class List;


class ListIterator;


class Node {
    friend class List;
    friend class ListIterator;
private:
    str data;
    Node* next;
public:
    explicit Node(str _data, Node* _next=nullptr);
    str get();
};


class List {
    friend class ListIterator;
private:
    uint64_t size;
    Node* head;
    Node* tail;

public:
    List();
    ~List() = default;

    Node* operator[](uint64_t index);
```

```
    bool isEmpty() const;
    void push(str data);
    void push(const str& data, uint64_t index);
    void pushHead(const str& data);
    void pushTail(str data);
    str pop();
    str pop(uint64_t index);
    void print();
    uint64_t getSize() const;
    std::string getData(uint64_t index);
    void clear();
};


class ListIterator {
private:
    List* list;
    Node* iterator;

public:
    explicit ListIterator(List *_list, Node *_iterator);
    void pushNext(str data);
    str popNext();
    void find(str data);
    str get();
};


Node::Node(str _data, Node* _next) {
    data = std::move(_data);
    next = _next;
}


str Node::get() {
    return data;
}


List::List() {
    size = 0;
    head = nullptr;
    tail = nullptr;
}


Node *List::operator[](const uint64_t index) {
    if(isEmpty() || index >= size) {
```

```
        throw Exception("error: bad index");
    }
    Node* node = head;
    for (int i = 0; i < index; i++) {
        node = node->next;
    }
    return node;
}


bool List::isEmpty() const {
    return size == 0;
}


void List::push(str data) {
    auto* node = new Node(std::move(data));
    if(isEmpty()) {
        size++;
        head = node;
        tail = node;
        return;
    }
    size++;
    tail->next = node;
    tail = node;
}


void List::push(const str& data, uint64_t index) {
    if (index > size) {
        throw Exception("error: bad index");
    }
    if (isEmpty() || index == size) {
        return push(data);
    }
    size++;
    auto* node = new Node(data);
    if (index == 0) {
        node->next = head;
        head = node;
        return;
    }
    node->next = this->operator[](index);
    this->operator[](index - 1)->next = node;
}
```

```
void List::pushHead(const str& data) {
    push(data, 0);
}


void List::pushTail(str data) {
    push(std::move(data));
}


str List::pop() {
    if (isEmpty()) {
        throw Exception("error: List is empty");
    }
    if (size == 1) {
        str data = tail->data;
        head = nullptr;
        tail = nullptr;
        size--;
        return data;
    }
    Node* node = head;
    for (uint64_t i = 0; i < size - 2; i++) {
        node = node->next;
    }
    node->next = nullptr;
    tail = node;
    size--;
    return tail->data;
}


str List::pop(uint64_t index) {
    if (index >= size) {
        throw Exception("error: bad index");
    }
    if (index == size - 1) {
        return pop();
    }
    if (index == 0) {
        str data = head->data;
        head = head->next;
        size--;
        return data;
    }
    Node* node = head;
    for (uint64_t i = 0; i < index - 1; i++) {
        node = node->next;
```

```
    }
    node->next = node->next->next;
    size--;
    return this->operator[](index)->data;
}


void List::print() {
    if(isEmpty()) {
        return;
    }
    std::cout << "[";
    Node* node = head;
    while (node != tail) {
        std::cout << node->data.get() << ", ";
        node = node->next;
    }
    std::cout << tail->data.get() << "]" << std::endl;
}


uint64_t List::getSize() const {
    return size;
}


std::string List::getData(uint64_t index) {
    if (index >= size) {
        throw Exception("error: bad index");
    }
    return this->operator[](index)->data.get();
}


void List::clear() {
    size = 0;
    head = nullptr;
    tail = nullptr;
}


ListIterator::ListIterator(List *_list, Node* _iterator) {
    list = _list;
    iterator = _iterator;
}


void ListIterator::pushNext(str data) {
```

```
    auto* new_node = new Node(std::move(data), iterator->next);
    iterator->next = new_node;
    if (iterator == list->tail) {
        list->tail = iterator->next;
    }
    list->size++;
}


str ListIterator::popNext() {
    if (!iterator->next) {
        throw Exception("error: bad access");
    }
    list->size--;
    str data = iterator->next->get();
    if (iterator->next == list->tail) {
        list->tail = iterator;
    }
    iterator->next = iterator->next->next;
    return data;
}


void ListIterator::find(str data) {
    Node* node = list->head;
    while(node) {
        if (node->data.get() == data.get()) {
            iterator = node;
            return;
        }
        node = node->next;
    }
    iterator = nullptr;
}


str ListIterator::get() {
    return iterator->get();
}


#endif //HELLO_WORLD_LIST_H
```

ПРИЛОЖЕНИЕ 5. str.h

```
//
// Created by user on 3/17/23.
//

#ifndef MANIPULATOR_STR_H
#define MANIPULATOR_STR_H


#include <iostream>
#include <utility>
#include <cstring>


class str {
private:
    std::string string;
    int curs;

public:
    str();

    void push(uint8_t symbol, int index);
    void pop(int index);
    void reset();
    uint64_t size();
    const char* get();
    std::string get_str();
    void keyBackspace();
    void keyDelete();
    void setCurs(int CURS_X);
    int getCurs() const;
    void set(std::string _string);
};


#endif //MANIPULATOR_STR_H
```

ПРИЛОЖЕНИЕ 6. str.cpp

```
//
// Created by user on 3/17/23.
//

#include "str.h"


str::str() {
    string = "";
    curs = -1;
}


void str::push(uint8_t symbol, int index) {
    curs = index;
    if (index >= string.size()) {
        string += static_cast<char>(symbol);
        return;
    }
    std::string buffer = string;
    string = "";
    for (int i = 0; i < index; i++) {
        string += buffer[i];
    }
    string += static_cast<char>(symbol);
    for (int i = index; i < buffer.size(); i++) {
        string += buffer[i];
    }
}


void str::pop(int index) {
    std::string buffer = string;
    string = "";
    for (int i = 0; i < buffer.size(); i++) {
        if (i != index) {
            string += buffer[i];
        }
    }
}


void str::reset() {
    curs = -1;
    string = "";
}
```

```
uint64_t str::size() {
    return string.size();
}


const char* str::get() {
    return string.c_str();
}


std::string str::get_str() {
    return string;
}


void str::keyBackspace() {
    pop(curs);
}


void str::keyDelete() {
    pop(curs + 1);
}


void str::setCurs(int CURS_X) {
    if (CURS_X >= string.size()) {
        curs = (int)string.size() - 1;
        return;
    }
    curs = CURS_X - 1;
}


int str::getCurs() const {
    return curs;
}


void str::set(std::string _string) {
    string = std::move(_string);
}
```

ПРИЛОЖЕНИЕ 7. History.h

```
//
// Created by user on 3/21/23.
//

#ifndef MANIPULATOR_HISTORY_H
#define MANIPULATOR_HISTORY_H


#include <utility>
#include "List.h"


class History {
private:
    inline static List list;
    inline static uint64_t index = 0;
    inline static str current_command;
public:
    static void moveUp();
    static void moveDown();
    static std::string get();
    static void append(const str &command);
    static uint64_t getIndex();
    static void setCurrentCommand(std::string command);
    static void resetIndex();
};


void History::moveUp() {
    if (index == list.getSize()) {
        return;
    }
    index++;
}


void History::moveDown() {
    if (index == 0) {
        return;
    }
    index--;
}


std::string History::get() {
    if (index == 0) {
```

```
        return current_command.get();
    }
    return list.getData(index - 1);
}


void History::append(const str &command) {
    list.pushHead(command);
}


uint64_t History::getIndex() {
    return index;
}


void History::setCurrentCommand(std::string command) {
    current_command.set(std::move(command));
}


void History::resetIndex() {
    index = 0;
}


#endif //MANIPULATOR_HISTORY_H
```

ПРИЛОЖЕНИЕ 8. Gservo.h

```
//
// Created by user on 3/18/23.
//

#ifndef MANIPULATOR_GSERVO_H
#define MANIPULATOR_GSERVO_H


#include <cstdint>


class Gservo {
private:
    uint8_t id;
    uint16_t goal;
    uint16_t angle;
    uint16_t speed;
    uint16_t torque;
    uint8_t is_moving;
    inline static int x;
    inline static int y;
    inline static int z;
    inline static uint16_t q0;
    inline static uint16_t q1;
    inline static uint16_t q2;

public:
    explicit Gservo(uint8_t _id);

    uint8_t get_id() const;
    uint16_t get_goal() const;
    uint16_t get_angle() const;
    uint16_t get_speed() const;
    uint16_t get_torque() const;
    uint16_t get_is_moving() const;
    static uint16_t get_x();
    static uint16_t get_y();
    static uint16_t get_z();
    static uint16_t get_q0() ;
    static uint16_t get_q1() ;
    static uint16_t get_q2() ;

    void set_goal(uint8_t _goal1, uint8_t _goal2);
    void set_angle(uint8_t _angle1, uint8_t _angle2);
    void set_speed(uint8_t _speed1, uint8_t _speed2);
    void set_torque(uint8_t _torque1, uint8_t _torque2);
```

```
    void set_is_moving(uint8_t _is_moving);
    static void set_x(uint8_t _x1, uint8_t _x2, bool x_sign);
    static void set_y(uint8_t _y1, uint8_t _y2, bool y_sign);
    static void set_z(uint8_t _z1, uint8_t _z2, bool z_sign);
    static void set_q0(uint8_t _q01, uint8_t _q02);
    static void set_q1(uint8_t _q11, uint8_t _q12);
    static void set_q2(uint8_t _q21, uint8_t _q22);
};


inline Gservo gservo1(1);
inline Gservo gservo2(2);
inline Gservo gservo3(3);
inline Gservo gservo4(4);


#endif //MANIPULATOR_GSERVO_H
```

ПРИЛОЖЕНИЕ 9. Gservo.cpp

```
//
// Created by user on 3/18/23.
//

#include "Gservo.h"


Gservo::Gservo(uint8_t _id) {
    id = _id;
    goal = 0;
    angle = 0;
    speed = 0;
    torque = 0;
    is_moving = 0;
    x = 0;
    y = 0;
    z = 0;
}


uint8_t Gservo::get_id() const {
    return id;
}


uint16_t Gservo::get_goal() const {
    return goal;
}


uint16_t Gservo::get_angle() const {
    return angle;
}


uint16_t Gservo::get_speed() const {
    return speed;
}


uint16_t Gservo::get_torque() const {
    return torque;
}


uint16_t Gservo::get_is_moving() const {
```

```
    return is_moving;
}


uint16_t Gservo::get_x(){
    return x;
}


uint16_t Gservo::get_y(){
    return y;
}


uint16_t Gservo::get_z(){
    return z;
}


uint16_t Gservo::get_q0() {
    return q0;
}


uint16_t Gservo::get_q1() {
    return q1;
}


uint16_t Gservo::get_q2() {
    return q2;
}


void Gservo::set_goal(uint8_t _goal1, uint8_t _goal2) {
    goal = _goal1 * 100 + _goal2;
}


void Gservo::set_angle(uint8_t _angle1, uint8_t _angle2) {
    angle = _angle1 * 100 + _angle2;
}


void Gservo::set_speed(uint8_t _speed1, uint8_t _speed2) {
    speed = _speed1 * 100 + _speed2;
}
```

```
void Gservo::set_torque(uint8_t _torque1, uint8_t _torque2) {
  torque = _torque1 * 100 + _torque2;
}


void Gservo::set_is_moving(uint8_t _is_moving) {
  is_moving = _is_moving;
}


void Gservo::set_x(uint8_t _x1, uint8_t _x2, bool x_sign) {
  x = (_x1 * 100 + _x2);
  x = (x_sign) ? x : -x;
}


void Gservo::set_y(uint8_t _y1, uint8_t _y2, bool y_sign) {
  y = (_y1 * 100 + _y2);
  y = (y_sign) ? y : -y;
}


void Gservo::set_z(uint8_t _z1, uint8_t _z2, bool z_sign) {
  z = (_z1 * 100 + _z2);
  z = (z_sign) ? z : -z;
}


void Gservo::set_q0(uint8_t _q01, uint8_t _q02) {
  q0 = _q01 * 100 + _q02;
}


void Gservo::set_q1(uint8_t _q11, uint8_t _q12) {
  q1 = _q11 * 100 + _q12;
}


void Gservo::set_q2(uint8_t _q21, uint8_t _q22) {
  q2 = _q21 * 100 + _q22;
}
```

ПРИЛОЖЕНИЕ 10. graphics.h

```
//
// Created by user on 05.03.23.
//

#ifndef MANIPULATOR_GRAPHICS_H
#define MANIPULATOR_GRAPHICS_H


#include <csignal>
#include <sys/ioctl.h>
#include "ncurses.h"
#include "Connect.h"
#include "History.h"


#define KEY_RETURN     10

#define ID_X           9
#define GOAL_X         17
#define ANGLE_X        23
#define SPEED_X        30
#define TORQUE_X       37
#define IS_MOVING_X    45
#define EDGE_X         55
#define X_X            57
#define Y_X            62
#define Z_X            67
#define Q0_X           57
#define Q1_X           62
#define Q2_X           67
#define ALL_Q_TITLE_Y  4
#define ALL_Q_Y        5

#define COMMAND_Y      8
#define LAST_COMMAND_Y 9


int CURS_Y = 0;
int CURS_X = 0;


int get_columns() {
    struct winsize window{};
    ioctl(0, TIOCGWINSZ, &window);
    return window.ws_col;
}
```

```
int get_rows() {
    struct winsize window{};
    ioctl(0, TIOCGWINSZ, &window);
    return window.ws_row;
}


void finish() {
    Connect::disconnectArduino();
    resetty();
    endwin();
    exit(0);
}


void sighandler(int sig) {
    if (sig == SIGINT) {
        finish();
    }
}


void print_table() {
    for (int i = 1; i < 5; i++) {
        move(i + 1, 0);
        printw("servo%d", i);
    }

    move(0, ID_X);
    printw("id");

    move(0, GOAL_X);
    printw("goal");
    move(0, ANGLE_X);
    printw("angle");

    move(0, SPEED_X);
    printw("speed");

    move(0, TORQUE_X);
    printw("torque");

    move(0, IS_MOVING_X);
    printw("is_moving");

    move(0, X_X);
```

```
    printw("x");

    move(0, Y_X);
    printw("y");

    move(0, Z_X);
    printw("z");

    move(COMMAND_Y - 1, 0);
    printw("Set command:");

    move(ALL_Q_TITLE_Y, Q0_X);
    printw("q0");

    move(ALL_Q_TITLE_Y, Q1_X);
    printw("q1");

    move(ALL_Q_TITLE_Y, Q2_X);
    printw("q2");

    for (int i = 0; i < 7; i++) {
        move(i, EDGE_X);
        printw("|");
    }

    refresh();
}


void print_id() {
    for (int i = 1; i < 5; i++) {
        move(i + 1, ID_X + 1);
        printw("%d", i);
    }
    refresh();
}


void print_exit_option() {
    move(get_rows() - 1, 0);
    printw("Press 'Ctrl+C' to exit");
    refresh();
}


void init_graphics() {
    initscr();
    savetty();  //  save terminal settings
```

```
    //nonl();  //  deny going to the new line

    cbreak();  //  send buffer after pressing enter
    echo();  //  visible printing
    timeout(0);

    //leaveok(stdscr, TRUE);
    //curs_set(0);  //  hide cursor
    keypad(stdscr, TRUE);

    signal(SIGINT, sighandler);

    clear();
    print_table();
    print_id();
    print_exit_option();
    move(COMMAND_Y, 0);
    refresh();
}


void clear_command_line() {
    move(COMMAND_Y, 0);
    for (int i = 0; i < get_columns(); i++) {
        printw(" ");
    }
    move(COMMAND_Y, 0);
    refresh();
}


void print_last_command() {
    move(LAST_COMMAND_Y, 0);
    for (int i = 0; i < get_columns(); i++) {
        printw(" ");
    }
    move(LAST_COMMAND_Y, 0);
    printw("%s", Connect::key_cmd.get());
}


void print_command_line() {
    move(COMMAND_Y, 0);
    printw("%s", Connect::key_cmd.get());
}
```

```
void key_return_proc() {
    if (Connect::key_cmd.get_str().empty()) {
        return;
    }
    History::append(Connect::key_cmd);
    Connect::decodeKeyInput();
    print_last_command();
    clear_command_line();
    Connect::key_cmd.reset();
    History::resetIndex();
}


void key_backspace_proc() {
    getsyx(CURS_Y, CURS_X);
    Connect::key_cmd.setCurs(CURS_X + 1);
    Connect::key_cmd.keyBackspace();
    clear_command_line();
    print_command_line();
    move(CURS_Y, CURS_X);
}


void key_delete_proc() {
    getsyx(CURS_Y, CURS_X);
    Connect::key_cmd.setCurs(CURS_X);
    Connect::key_cmd.keyDelete();
    clear_command_line();
    print_command_line();
    move(CURS_Y, CURS_X);
}


void key_left_proc() {
    getsyx(CURS_Y, CURS_X);
    move(CURS_Y, CURS_X - 1);
}


void key_right_proc() {
    getsyx(CURS_Y, CURS_X);
    if (CURS_X == Connect::key_cmd.size()) {
        return;
    }
    move(CURS_Y, CURS_X + 1);
}
```

```
void key_up_proc() {
    if (History::getIndex() == 0) {
        History::setCurrentCommand(Connect::key_cmd.get());
    }
    History::moveUp();
    Connect::key_cmd.set(History::get());
    clear_command_line();
    print_command_line();
}


void key_down_proc() {
    History::moveDown();
    Connect::key_cmd.set(History::get());
    clear_command_line();
    print_command_line();
}


void key_proc(int key) {
    auto symbol = static_cast<uint8_t>(key);
    if (key == KEY_RETURN) {
        return key_return_proc();
    }
    if (key == KEY_BACKSPACE) {
        return key_backspace_proc();
    }
    if (key == KEY_DC) {
        return key_delete_proc();
    }
    if (key == KEY_LEFT) {
        return key_left_proc();
    }
    if (key == KEY_RIGHT) {
        return key_right_proc();
    }
    if (key == KEY_UP) {
        return key_up_proc();
    }
    if (key == KEY_DOWN) {
        return key_down_proc();
    }
    if (key == ERR) {
        return;
    }

    getsyx(CURS_Y, CURS_X);
    Connect::key_cmd.push(symbol, CURS_X - 1);
```

```
    clear_command_line();
    print_command_line();
    move(CURS_Y, Connect::key_cmd.getCurs() + 1);
    refresh();
}


void print_param(uint8_t gservo_id, int x, uint16_t param, bool is_q=false) {
    int y = (is_q) ? ALL_Q_Y : 1 + gservo_id;
    move(y, x);
    printw("    ");
    move(y, x);
    printw("%d", param);
}


void print_goal(uint8_t gservo_id, uint16_t goal) {
    print_param(gservo_id, GOAL_X, goal);
}


void print_angle(uint8_t gservo_id, uint16_t angle) {
    print_param(gservo_id, ANGLE_X + 1, angle);
}


void print_speed(uint8_t gservo_id, uint16_t speed) {
    print_param(gservo_id, SPEED_X + 1, speed);
}


void print_torque(uint8_t gservo_id, uint16_t torque) {
    print_param(gservo_id, TORQUE_X + 1, torque);
}


void print_is_moving(uint8_t gservo_id, uint16_t is_moving) {
    print_param(gservo_id, IS_MOVING_X, is_moving);
}


void print_x(uint16_t x) {
    print_param(DXL_ID1, X_X, x);
}


void print_y(uint16_t y) {
    print_param(DXL_ID1, Y_X, y);
```

```
}


void print_z(uint16_t z) {
    print_param(DXL_ID1, Z_X, z);
}


void print_q0(uint16_t q0) {
    print_param(ALL_Q_Y, Q0_X, q0);
}


void print_q1(uint16_t q1) {
    print_param(ALL_Q_Y, Q1_X, q1);
}


void print_q2(uint16_t q2) {
    print_param(ALL_Q_Y, Q2_X, q2);
}


void print_params_from_servo(Gservo gservo) {
    print_goal(gservo.get_id(), gservo.get_goal());
    print_angle(gservo.get_id(), gservo.get_angle());
    print_speed(gservo.get_id(), gservo.get_speed());
    print_torque(gservo.get_id(), gservo.get_torque());
    print_is_moving(gservo.get_id(), gservo.get_is_moving());
    print_x(gservo.get_x());
    print_y(gservo.get_y());
    print_z(gservo.get_z());
    print_q0(gservo.get_q0());
    print_q1(gservo.get_q1());
    print_q2(gservo.get_q2());
}

void print_params() {
    getsyx(CURS_Y, CURS_X);
    print_params_from_servo(gservo1);
    print_params_from_servo(gservo2);
    print_params_from_servo(gservo3);
    print_params_from_servo(gservo4);
    move(CURS_Y, CURS_X);
}
#endif //MANIPULATOR_GRAPHICS_H
```

ПРИЛОЖЕНИЕ 11. main.cpp

```
#include <chrono>
#include "graphics.h"


int main() {
    if (!Connect::setConnection()) {
        finish();
    }

    init_graphics();

    auto start_timer = std::chrono::system_clock::now();
    while (true) {

        key_proc(getch());

        auto end_timer = std::chrono::system_clock::now();
        if (std::chrono::duration_cast<std::chrono::milliseconds>(end_timer - start_timer).count() >
int(TIMER)) {
            Connect::sendCommand();
            for (int i = 0; i < 4; i++) {
                if (Connect::receiveMessage()) {
                    print_params();
                }
            }
            start_timer = std::chrono::system_clock::now();
        }
    }
}
```