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 _____Е. Д. Пыхалов

Преподаватель _____М. С. Ананьевский

Санкт-Петербург
2023

ИССЛЕДОВАНИЕ ВОЗМОЖНОСТЕЙ МАНИПУЛЯТОРА

Манипулятор «OmegaMan» состоит из металлических звеньев, перемещающихся за счет сервоприводов. 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.

Робот не имеет встроенной защиты от столкновений с рабочей поверхностью, другими собственными звеньями. Кроме того, отсутствует упрощенная возможность получения данных о текущем состоянии приводов.

Таким образом, необходимо разработать программу для OpenCM9.04, которая будет контролировать перемещения робота, а также позволит в упрощенном порядке взаимодействовать с роботом.

ОПИСАНИЕ ПРОГРАММЫ

Получение и отправка данных реализуется с помощью стороннего приложения. Класс Connection.h используется для соединения с приложением. Все поступающие команды обрабатываются с помощью класса Servo.h, который после проверки доступности положения отправляет нужные сигналы на сервопривод. На данный момент реализовано осевое движение манипулятора (отдельно каждой осью). Класс Joint.h задуман как реализация движения в определенной системе координат, но на данный момент реализован только функционал пересчета текущего положения в декартову систему координат. Класс Calibration.h запускается при необходимости в получении новых крайних положений. Реализован с помощью датчика момента в каждом сервоприводе.

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]. Сервоприводы Dynamixel AX-12A имеют встроенный EEPROM, что позволяет вносить и извлекать параметры каждого привода, напрямую получая доступ к регистру.
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 ПКчерез 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:

- `init()` – инициализация сервоприводов;
- `pingServos()` – проверить отклик сервоприводов;
- `set_angle(uint16_t angle, uint8_t ID)` – задать сервоприводу значение угла;
- `set_speed(uint16_t speed, uint8_t ID)` – задать сервоприводу значение скорости;
- `safe_move(uint16_t msg)` – функция контроля доступности конечного положения;
- `set_max_angle(uint16_t max_angle)` – устанавливает максимальный угол для сервопривода
- `set_min_angle(uint16_t min_angle)` – устанавливает минимальный угол для сервопривода
- `reformatAngle(uint16_t angle)` – ограничение угла, подаваемого на вход
- `set_angle(uint16_t _angle)` - задать сервоприводу значение угла;
- `set_speed(uint16_t _speed)` - задать сервоприводу скорость перемещения;
- `setStartPosition()` - Возвращает манипулятор в стартовое положение;
- `toolPush()` - схватить объект;
- `toolPop()` - отпустить объект;

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

Стоит также заметить, что методы, такие как get_angle, is_moving и тд. , реализованы путем обращения к регистру EEPROM необходимого сервопривода.

Отдельного рассмотрения требует метод Servo::safe_move(uint16_t msg). Манипулятор построен таким образом, что 1 и 4 сервоприводы двигаются независимо от остальных. Вращение вокруг вертикальной оси или же схват объектов не требует работы остальных сервоприводов. На рисунке 1 представлена схема расположения сервоприводов на роботе, подобном OmegaMan.

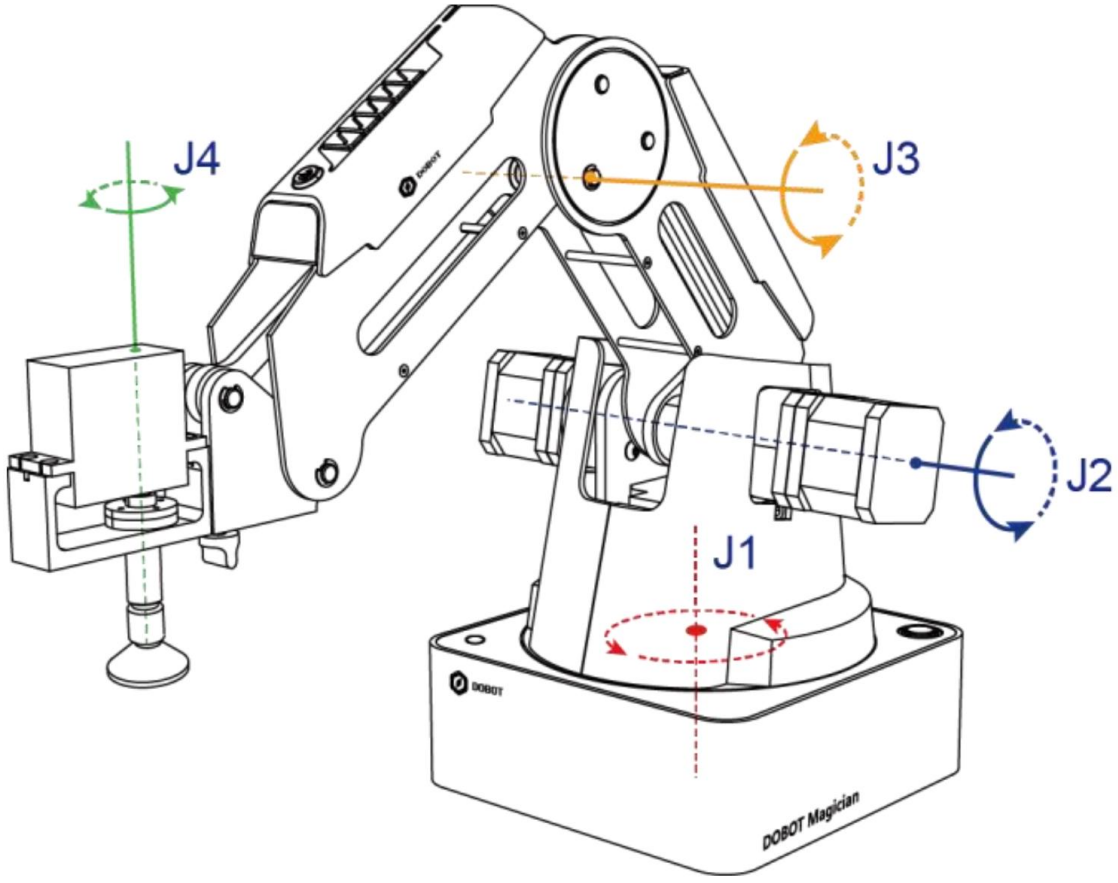

Рисунок 1 – Расположение сервоприводов

Каждый сервопривод в момент, когда он не двигается, находится в режиме

удержания момента, чтобы сохранять заданное положение. Однако, кинематика робота устроена так, что существуют такие положения, для перемещения в которые требуется работа двух сервоприводов (2 и 3) одновременно.

При попытке использовать только один сервопривод (например 2), создается ситуация, когда один сервопривод, пытаясь переместиться в заданное положение, встречает сопротивление со стороны другого привода(3), который не используется и находится в режиме удержания момента. Спустя какое-то время, один из сервоприводов уходит в режим защита из-за превышения допустимой нагрузки.

Чтобы этого не происходило, было принято решение разработать функцию, которая при движении одного сервопривода, проверяет, нужно ли двигать и второй.

В начале необходимо было получить зависимости второго привода от третьего и наоборот. На рисунке 2 представлена зависимость максимального и минимального углов 3 сервопривода, от положения 2 сервопривода.

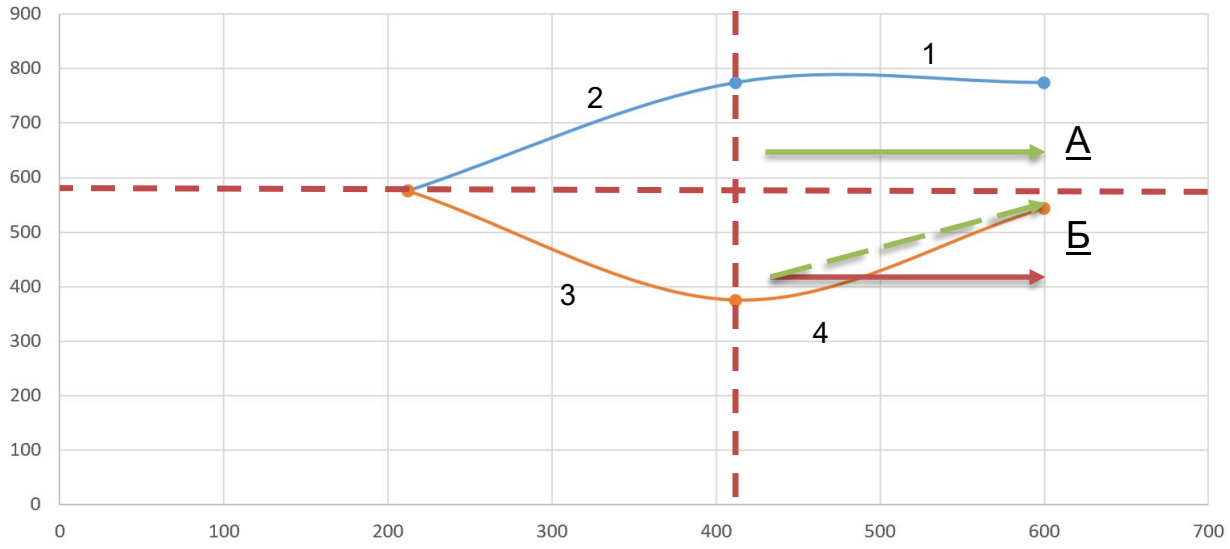

Рисунок 2 - Зависимость положений третьего привода от положения второго

Как можно понять из рисунка, наибольший диапазон работы у третьего привода наблюдается при значении угла второго привода около 400.

Для реализации функции было принято решение разделить рабочую область на 4 четверти, чтобы в каждой области находилась простая линейная зависимость. После этого мы получили 4 уравнения, которые описывают зависимость на каждом участке.

Функция реализована следующим образом: при получении команды на движение второго сервопривода, функция проверяет, в какой четверти находится

точка целевого положения (заданное положение для второго, текущее положение третьего). Затем, с помощью уравнения проверяет, находится ли третий сервопривод в зоне допустимых значений. Если же нет – двигает его до минимально/максимально допустимого.

В качестве примера, на рисунке 2 под А обозначено движение, не требующее корректировки положения третьего привода, а под Б – движение, для которого требуется корректировка (обычная линия – без функции, пунктирная – работа функции).

Аналогичный алгоритм реализован и при движении третьего сервопривода в целевое положение.

ПРИЛОЖЕНИЕ 1. Serrvo.h

```
#ifndef Servo_h
#define Servo_h


#include <stdint.h>
#include <string>
#include <DynamixelWorkbench.h>
#include "Config.h"


DynamixelWorkbench servos;


static bool tool_flag = false;


class Servo {
private:
    uint8_t DXL_ID;
    uint16_t min_angle;
    uint16_t max_angle;
    uint16_t angle;
    uint16_t new_angle;
    uint16_t speed;

public:
    Servo(uint8_t _DXL_ID, uint16_t _min_angle, uint16_t _max_angle);

    static void init();
    static void pingServos();
    static void setStartPosition();
    uint16_t reformatAngle(uint16_t _angle);
    static Servo* findServo(uint8_t id);

    void set_angle(uint16_t _angle);
    static void set_angle(uint16_t _angle, uint8_t _DXL_ID);
    void set_max_angle(uint16_t _max_angle);
    void set_min_angle(uint16_t _min_angle);

    void set_speed(uint16_t _speed);
    static void set_speed(uint16_t _speed, uint8_t _DXL_ID);

    void set_torque(bool status);

    void set_x(int32_t _x);
    void set_y(int32_t _y);
    void set_z(int32_t _z);

    static void safe_move(uint16_t msg);
    static bool talk(uint16_t msg);
    static void mv(uint16_t msg);

private:
    int32_t readRegister(char* command);

public:
    uint8_t get_DXL_ID();

    uint16_t get_angle();
    uint16_t get_max_angle();
    uint16_t get_min_angle();

    uint16_t get_goal();
```

```
    uint16_t get_load();

    uint8_t is_moving();

    uint16_t get_speed();

    static void toolPush(bool start=false);
    static void toolPop();
};


Servo servo1(DXL_ID1, SERVO1_MIN_ANGLE, SERVO1_MAX_ANGLE);
Servo servo2(DXL_ID2, SERVO2_MIN_ANGLE, SERVO2_MAX_ANGLE);
Servo servo3(DXL_ID3, SERVO3_MIN_ANGLE, SERVO3_MAX_ANGLE);
Servo servo4(DXL_ID4, SERVO4_MIN_ANGLE, SERVO4_MAX_ANGLE);


Servo::Servo(uint8_t _DXL_ID, uint16_t _min_angle, uint16_t _max_angle) {
    DXL_ID = _DXL_ID;
    min_angle = _min_angle;
    max_angle = _max_angle;
    angle = 0;
    new_angle = 0;
}


void Servo::init() {
    servos.init(DEVICE_NAME, SERVO_BAUDRATE);
}


void Servo::pingServos() {
    servos.ping(DXL_ID1, 0, 0);
    servos.ping(DXL_ID2, 0, 0);
    servos.ping(DXL_ID3, 0, 0);
    servos.ping(DXL_ID4, 0, 0);
}


void Servo::setStartPosition() {
    servo1.set_angle(SERVO1_START_POSITION);
    servo2.set_angle(SERVO2_START_POSITION);
    servo3.set_angle(SERVO3_START_POSITION);
    delay(1000);
}


uint16_t Servo::reformatAngle(uint16_t _angle) {
    if (_angle < min_angle) {
        return min_angle;
    }
    if (_angle > max_angle) {
        return max_angle;
    }
    return _angle;
}


Servo* Servo::findServo(uint8_t id) {
    if (id == 1) {
        return &servo1;
    }
    if (id == 2) {
        return &servo2;
```

```
    }
    if (id == 3) {
        return &servo3;
    }
    if (id == 4) {
        return &servo4;
    }
}


void Servo::set_angle(uint16_t _angle) {
    uint16_t _min_angle = min_angle;;
    uint16_t _max_angle = max_angle;

    if (_angle < _min_angle) {
        angle = _min_angle;
        servos.goalPosition(DXL_ID, angle);
        return;
    }
    if (_angle > _max_angle) {
        angle = _max_angle;
        servos.goalPosition(DXL_ID, angle);
        return;
    }
    angle = _angle;
    servos.goalPosition(DXL_ID, angle);
}


void Servo::set_angle(uint16_t _angle, uint8_t _DXL_ID) {
    Servo* servo = findServo(_DXL_ID);
    servo->set_angle(_angle);
}


void Servo::set_max_angle(uint16_t _max_angle) {
    max_angle = _max_angle;
}


void Servo::set_min_angle(uint16_t _min_angle) {
    min_angle = _min_angle;
}


void Servo::set_speed(uint16_t _speed) {
    speed = _speed;
    servos.jointMode(DXL_ID, speed, DEFAULT_BOOST);
}


void Servo::set_speed(uint16_t _speed, uint8_t _DXL_ID) {
    Servo* servo = findServo(_DXL_ID);
    servo->set_speed(_speed);
}


void Servo::set_torque(bool status) {
    if (status == 1)
        servos.torqueOn(DXL_ID);
    if (status == 0)
        servos.torqueOff(DXL_ID);
}
```

```
void Servo::safe_move(uint16_t msg) {

    uint16_t neutral_angle_2 = (servo2.max_angle + servo2.min_angle) / 2 - 20;
    uint16_t neutral_angle_3 = (servo3.max_angle + servo3.min_angle) / 2 + 20;

    uint16_t num2 = servo3.max_angle - neutral_angle_2;
    uint16_t num3 = neutral_angle_2 + servo3.min_angle;
    uint16_t num4 = neutral_angle_2 - servo3.min_angle;

    uint16_t angle_3 = 1023 - servo3.get_angle();
    uint16_t angle_2 = servo2.get_angle();

    uint8_t id = msg / 10000;
    uint16_t msg_angle = msg % 10000;

    if (id == 2) {

        uint8_t quart;
        uint16_t correct_angle_3;

        msg_angle = servo2.reformatAngle(msg_angle);

        if (angle_3 > neutral_angle_3) {
            if (msg_angle > neutral_angle_2)
                quart = 1; // Определяем четверть
            else
                quart = 2;
        } else {
            if (msg_angle <= neutral_angle_2)
                quart = 3;
            else
                quart = 4;
        }

        switch (quart) { // Действия для каждой четверти
            case 1:
                servo2.set_angle(msg_angle);
                break;

            case 2:
                correct_angle_3 = num2 + msg_angle;
                if (angle_3 > correct_angle_3)
                    servo3.set_angle(correct_angle_3);
                servo2.set_angle(msg_angle);
                break;

            case 3:
                correct_angle_3 = num3 - msg_angle;
                if (angle_3 < correct_angle_3 && msg_angle < 320)
                    servo3.set_angle(correct_angle_3 + (320 - msg_angle) / 6);
                servo2.set_angle(msg_angle);
                break;

            case 4:
                correct_angle_3 = msg_angle - num4;
                if (angle_3 < correct_angle_3)
                    servo3.set_angle(correct_angle_3);
                servo2.set_angle(msg_angle);
                break;

            default:
                servo2.set_angle(msg_angle);
                break;
        }
        return;
```

```
    }

    if (id == 3) {

        uint8_t place;
        uint16_t correct_angle_2;

        msg_angle = servo3.reformatAngle(msg_angle);

        if (msg_angle < neutral_angle_3 - 200) {
            if (angle_2 > neutral_angle_2) {
                place = 7;
            } else {
                place = 8;
            }
        } else if (msg_angle < neutral_angle_3 - 100 &&
                   msg_angle >= neutral_angle_3 - 200) {
            if (angle_2 > neutral_angle_2) {
                place = 5;
            } else {
                place = 6;
            }
        } else if (msg_angle < neutral_angle_3 &&
                   msg_angle >= neutral_angle_3 - 100) {
            if (angle_2 > neutral_angle_2) {
                place = 3;
            } else {
                place = 4;
            }
        } else {
            if (angle_2 > neutral_angle_2) {
                place = 1;
            } else {
                place = 2;
            }
        }

        switch (place) { // Действия для каждой четверти
            case 1:
                servo3.set_angle(msg_angle);
                break;

            case 2:
                correct_angle_2 = msg_angle - num2;
                if (angle_2 < correct_angle_2) {
                    servo2.set_angle(correct_angle_2);
                }
                servo3.set_angle(msg_angle);
                break;

            case 3:
            case 7:
                correct_angle_2 = servo2.max_angle - (neutral_angle_3 - msg_angle) /
2;
                if (angle_2 > correct_angle_2) {
                    servo2.set_angle(correct_angle_2);
                }
                servo3.set_angle(msg_angle);
                break;

            case 4:
            case 6:
                correct_angle_2 = servo2.min_angle + (neutral_angle_3 - msg_angle) /
2;
                if (angle_2 < correct_angle_2) {
```

```
                    servo2.set_angle(correct_angle_2);
                }
                servo3.set_angle(msg_angle);
                break;

            case 5:
                correct_angle_2 = msg_angle + 50;
                if (angle_2 > correct_angle_2) {
                    servo2.set_angle(correct_angle_2);
                }
                servo3.set_angle(msg_angle);
                break;

            case 8:
                correct_angle_2 = 300;
                if (angle_2 < correct_angle_2) {
                    servo2.set_angle(correct_angle_2);
                }
                servo3.set_angle(msg_angle);
                break;

            default:
                break;
        }
    }
}


bool Servo::talk(uint16_t msg) {
    if (msg < 1) {
        return false;
    }
    if (msg == 1) {
        servo1.set_torque(0);
        return true;
    }
    if (msg == 2) {
        servo2.set_torque(0);
        return true;
    }
    if (msg == 3) {
        servo3.set_torque(0);
        return true;
    }
    if (msg == 4) {
        servo4.set_torque(0);
        return true;
    }

    uint8_t id = msg / 10000;
    Servo* servo = findServo(id);
    uint16_t msg_angle = msg % 10000;

    if (id == 2 || id == 3) safe_move(msg);
    else{
        servo3.set_angle(msg_angle);
        return true;
    }
}


void Servo::mv(uint16_t msg) {
    if (msg < 10000) {
      return;
    }
```

```
    uint8_t id = msg / 10000;
    Servo* servo = findServo(id);

    uint16_t msg_angle = msg % 10000;

    servo->set_angle(msg_angle);
}


int32_t Servo::readRegister(char* command) {
    int32_t data;
    servos.readRegister(DXL_ID, command, &data);
    return data;
}


uint8_t Servo::get_DXL_ID() {
    return DXL_ID;
}


uint16_t Servo::get_angle() {
    int32_t data;
    servos.readRegister(DXL_ID, "Present_Position", &data);
    return (uint16_t)data;
}


uint16_t Servo::get_max_angle() {
    return max_angle;
}


uint16_t Servo::get_min_angle() {
    return min_angle;
}


uint16_t Servo::get_goal() {
    return angle;
}


uint16_t Servo::get_load() {
    int32_t data;
    servos.readRegister(DXL_ID, "Present_Load", &data);
    if (data > 1023)
        data -= 1023;
    return (uint16_t)data;
}


uint8_t Servo::is_moving() {
    int32_t data;
    servos.readRegister(DXL_ID, "Moving", &data);
    return (uint8_t)data;
}


uint16_t Servo::get_speed() {
    return speed;
}


void Servo::toolPush(bool start) {
```

```
    if (start) {
        tool_flag = true;
    }
    if (!tool_flag) {
        return;
    }
    servo4.set_angle(servo4.get_goal() + 10);
    if (servo4.get_load() > TOOL_MAX_LOAD || servo4.get_goal() >= SERVO4_MAX_ANGLE) {
        servo4.set_angle(servo4.get_goal() - 10);
        tool_flag = false;
    }
}


void Servo::toolPop() {
    servo4.set_angle(SERVO4_MIN_ANGLE);
}


#endif
```

ПРИЛОЖЕНИЕ 2.Connection.h

```
#ifndef Connection_h
#define Connection_h


#include "Config.h"
#include "Joint.h"
#include "Servo.h"


uint8_t command[COMMAND_SIZE];
uint8_t message[MESSAGE_SIZE];


class Connection {
private:
    static uint8_t crc8(uint8_t data[], int size);
    static uint16_t calcCommandCheckSum();
    static uint16_t calcMessageCheckSum();
    static void setMsgValues(uint8_t id);

public:
static void sendMessage(uint8_t id);
    static void receiveCommand();

private:
    static void findCommand();
};


uint8_t Connection::crc8(uint8_t data[], int size) {
    uint8_t byte;
    uint8_t POLY = 0x7;
    uint8_t crc8 = 0xFF;

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

        byte = data[j];
        crc8 = crc8 ^ byte;

        for (int i = 0; i < 8; i++) {

            if (crc8 & 0x80) {
                crc8 = (crc8 << 1) ^ POLY;
            }
            else {
                crc8 = crc8 << 1;
            }
        }
    }
    return crc8;
}


uint16_t Connection::calcCommandCheckSum() {
    return crc8(command, COMMAND_SIZE);
}


uint16_t Connection::calcMessageCheckSum() {
    return crc8(message, MESSAGE_SIZE-1);
}
```

```
void Connection::setMsgValues(uint8_t id) {
    Servo* servo = Servo::findServo(id);

    message[MESSAGE_START_BYTE1_CELL] = START_BYTE;
    message[MESSAGE_START_BYTE2_CELL] = START_BYTE;

    message[MESSAGE_ID_CELL] = id;

    uint16_t goal = servo->get_goal();
    message[MESSAGE_GOAL1_CELL] = goal / 100;
    message[MESSAGE_GOAL1_CELL] = goal % 100;

    uint16_t angle = servo->get_angle();
    message[MESSAGE_ANGLE1_CELL] = angle / 100;
    message[MESSAGE_ANGLE2_CELL] = angle % 100;

    uint16_t speed = servo->get_speed();
    message[MESSAGE_SPEED1_CELL] = speed / 100;
    message[MESSAGE_SPEED2_CELL] = speed % 100;

    uint16_t load = servo->get_load();
    message[MESSAGE_TORQUE1_CELL] = load / 100;
    message[MESSAGE_TORQUE2_CELL] = load % 100;

    message[MESSAGE_IS_MOVING_CELL] = servo->is_moving();

    int16_t x = joint4.get_x();
    message[MESSAGE_X1_CELL] = abs(x / 100);
    message[MESSAGE_X2_CELL] = abs(x % 100);
    message[MESSAGE_X_SIGN] = (x >= 0) ? 1 : 0;

    int16_t y = joint4.get_y();
    message[MESSAGE_Y1_CELL] = abs(y / 100);
    message[MESSAGE_Y2_CELL] = abs(y % 100);
    message[MESSAGE_Y_SIGN] = (y >= 0) ? 1 : 0;

    int16_t z = joint4.get_z();
    message[MESSAGE_Z1_CELL] = abs(z / 100);
    message[MESSAGE_Z2_CELL] = abs(z % 100);
    message[MESSAGE_Z_SIGN] = (z >= 0) ? 1 : 0;

    message[MESSAGE_Q01_CELL] = 18;
    message[MESSAGE_Q02_CELL] = 19;
    message[MESSAGE_Q11_CELL] = 20;
    message[MESSAGE_Q12_CELL] = 21;
    message[MESSAGE_Q21_CELL] = 22;
    message[MESSAGE_Q22_CELL] = 23;

    message[MESSAGE_CHECKSUM_CELL] = calcMessageCheckSum();
}


void Connection::sendMessage(uint8_t id) {
    setMsgValues(id);

    for (int cell = MESSAGE_START_BYTE1_CELL; cell < MESSAGE_SIZE; cell++) {
        Serial.print(char(message[cell]));
    }
}


void Connection::receiveCommand() {
    if (Serial.available() >= 7) {
        command[COMMAND_START_BYTE1_CELL] = Serial.read();
        command[COMMAND_START_BYTE2_CELL] = Serial.read();
```

```
        if (command[COMMAND_START_BYTE1_CELL] == START_BYTE &&
command[COMMAND_START_BYTE2_CELL] == START_BYTE) {
            for (int cell = COMMAND_ID_CELL; cell < COMMAND_SIZE; cell++) {
                command[cell] = Serial.read();
            }
            if (!calcCommandCheckSum()) {
                findCommand();
                sendMessage(DXL_ID1);
                sendMessage(DXL_ID2);
                sendMessage(DXL_ID3);
                sendMessage(DXL_ID4);
            }
        }
    }
}


void Connection::findCommand() {
    uint16_t value = command[COMMAND_VALUE1_CELL] * 100 +
command[COMMAND_VALUE2_CELL];
    uint8_t com = command[COMMAND_TASK1_CELL] * 100 + command[COMMAND_TASK2_CELL];
    if (com == PING_TASK) {
        return;
    }
    if (com == SET_ANGLE_TASK) {
        return Servo::set_angle(value, command[COMMAND_ID_CELL]);
    }
    if (com == SET_SPEED_TASK) {
        return Servo::set_speed(value, command[COMMAND_ID_CELL]);
    }
    if (com == TOOL_PUSH_TASK) {
        return Servo::toolPush(true);
    }
    if (com == TOOL_POP_TASK) {
        return Servo::toolPop();
    }
    if (com == GO_HOME_TASK) {
        return Servo::setStartPosition();
    }
}


#endif
```

ПРИЛОЖЕНИЕ 3.Arduino.ino

```
#include <DynamixelWorkbench.h>
#include "Calibration.h"
#include "Config.h"
#include "Connection.h"
#include "Joint.h"
#include "Servo.h"


unsigned long long int tool_timer;
unsigned long long int now;


void setup() {
    Serial.begin(SERIAL_BAUDRATE);
    Serial.setTimeout(0);

    Servo::init();
    Servo::pingServos();

    servo1.set_speed(DEFAULT_SPEED);
    servo2.set_speed(DEFAULT_SPEED);
    servo3.set_speed(DEFAULT_SPEED);
    servo4.set_speed(DEFAULT_SPEED);

    Servo::setStartPosition();
    Servo::toolPop();

    tool_timer = millis();
}


void loop() {
    Connection::receiveCommand();
    //Servo::mv(Serial.parseInt());
    now = millis();
    if (now - tool_timer > TOOL_TIMER) {
        Servo::toolPush();
        tool_timer = now;
    }
}
```

СПИСОК ЛИТЕРАТУРЫ

1. Руководство по эксплуатации OmeghaMan.mini v1.5;
2. Бьёрн Страуструп – Язык программирования C++;
3. А.Н. Евграфов, М.З. Коловский, Г.Н. Петров – Теория механизмов и машин 2020;
4. https://omegabot.ru./;
5. https://en.cppreference.com/w/;
6. https://github.com/artem-kondratew/Manipulator.git.