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

КУРСОВАЯ РАБОТА

по дисциплине: объектно-ориентированное программирование
Тема: разработка программы для управления манипулятором «OmegaMan»

Выполнил
Студент группы 3331506/00401 _____В. О. Малыхин

Преподаватель. _____М. С. Ананьевский

«___»_____2023 г.

Санкт-Петербург
2023

Цель работы – разработать программное обеспечение для робота-манипулятора OmegaMan компании OmegaBot.

В данной части курсовой работы были разработаны классы для калибровки и расчета геометрических характеристик.

Выполнение работы

Манипулятор OmegaMan имеет 4 вращательные кинематические пары, каждая кинематическая пара имеет свои ограничения, которые задаются в следующем диапазоне: 0 – 1023 (0 – 5π/3). Ограничения записаны в файле Arduino/Config.h. Согласно описанию манипулятора, схват выдержит нагрузку массой 100г.

Платформа OpenCM9.04 моно программировать в ArduinoIDE. 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 расчета геометрических характеристик манипулятора; Servo.h – модуль взаимодействия с сервоприводами манипулятора. Модуль Calibration.h содержит экспериментальные возможности для калибровки манипулятора.

Разработанные классы:

1) Класс Joint.h – используется для расчета геометрических характеристик.

2) Класс Calibration.h – используется для калибровки.

Текст кода приведён в приложении.

Класс Joint.h

Существующие объекты:

joint1

joint2

joint3

joint4

Методы:

obj.get_radians() – обновляет значения углов в радианах;

obj.get_x() – возвращает координату x данного соединения;

obj.get_y() – возвращает координату y данного соединения;

obj.get_z() м возвращает координату z данного соединения;

obj.get_coordinates() – выводит в Serial Port значения всех координат звена.

Ось X манипулятора направлена от контроллера в сторону основания манипулятора. Оси Y и Z направлены так, что образуется правая тройка векторов, причем ось Y лежит в горизонтальной плоскости.

Погрешность – 5 мм.

Класс Calibration.h

Методы:

obj.calibration_min(Servo servo) – Задаёт значения крайних положений, основываясь на текущем моменте на двигателе;

obj.calibration_max(Servo servo) – Задаёт значения крайних положений, основываясь на текущем моменте на двигателе;

obj.calibration_setup() – Производит полную калибровку манипулятора;

Список литературы

1. Руководство по эксплуатации OmegaMan.mini v1.5;

2. Бьёрн Страуструп – Язык программирования C++ (четвёртое издание), 2013;

3. А.Н. Евграфов, М.З. Коловский, Г.Н. Петров – Теория механизмов и машин, 2020;

4.Компания Omegabot //Главная страница: [Электронный ресурс]. URL: https://omegabot.ru/ (дата обращения 01.05.23);

5. C and C++ reference // Main page: [Электронный ресурс]. URL: https://en.cppreference.com/w/ (дата обращения 01.05.23);

6. Repository with files for course work "Manipulator": [Электронный ресурс]. URL: https://github.com/artem-kondratew/Manipulator.git (дата обращения 01.05.23).

Приложение

Joint.h

```
#ifndef Joint_h
#define Joint_h


#include <cstdint>
#include <cmath>
#include "Servo.h"


// Lengths of links in mm
#define X0    -30
#define Z0     87
#define LEN1 94.4
#define LEN2  150
#define LEN3  150
#define LEN4  141


class Joint {
private:
    uint8_t DXL_ID;
    // Coordinates in mm
    int16_t x;
    int16_t y;
    int16_t z;
    int16_t q0;
    int16_t q1;
    int16_t q2;
    double q0_rad;
    double q1_rad;
    double q2_rad;

public:
```

```
    Joint(uint8_t _DXL_ID);
    ~Joint() = default;

    void get_radians();
    int16_t get_x();
    int16_t get_y();
    int16_t get_z();
    void get_coordinates();
};


Joint::Joint(uint8_t _DXL_ID) {
    DXL_ID = _DXL_ID;
}


void Joint::get_radians() {
    //q0 = map(servo1.get_angle(), 0, 1023, 30, 330) - 180;
    //q1 = map(servo2.get_angle(), 0, 1023, 30, 330) - 90;
    //q2 = map(servo3.get_angle(), 0, 1023, 330, 30) - 232;
    q0_rad = (map(servo1.get_angle(), 0, 1023, 30, 330) - 180) /
   180.0 * PI;
    q1_rad = (map(servo2.get_angle(), 0, 1023, 30, 330) - 90) /
   180.0 * PI;
    q2_rad = (map(servo3.get_angle(), 0, 1023, 30, 330) - 128) /
   180.0 * PI;
}


int16_t Joint::get_x() {
    get_radians();
    switch (DXL_ID) {
        case 1: {
            x = X0 * cos(q0_rad);
            break;
        }
```

```
        case 2: {
            x = (X0 + LEN2 * cos(q1_rad)) * cos(q0_rad);
            break;
        }
        case 3: {
            x = (X0 + LEN2 * cos(q1_rad) + LEN3 * cos(q2_rad)) *
   cos(q0_rad);
            break;
        }
        case 4: {
            x = (X0 + LEN2 * cos(q1_rad) + LEN3 * cos(q2_rad) +
   LEN4) * cos(q0_rad);
            break;
        }
        default:
            Serial.print("Wrong DXL_ID!");
            break;
    }

    return x;
}



int16_t Joint::get_y() {
    get_radians();
    switch (DXL_ID) {
        case 1: {
            y = X0 * sin(q0_rad);
            break;
        }
        case 2: {
            y = (X0 + LEN2 * cos(q1_rad)) * sin(q0_rad);
            break;
        }
        case 3: {
```

```
            y = (X0 + LEN2 * cos(q1_rad) + LEN3 * cos(q2_rad)) *
    sin(q0_rad);
            break;
        }
        case 4: {
            y = (X0 + LEN2 * cos(q1_rad) + LEN3 * cos(q2_rad) +
    LEN4) * sin(q0_rad);
            break;
        }
    }

    return y;
}


int16_t Joint::get_z() {
    get_radians();
    switch (DXL_ID) {
        case DXL_ID1: {
            z = Z0;
            break;
        }
        case 2: {
            z = Z0 + LEN2 * sin(q1_rad);
            break;
        }
        case 3: {
            z = Z0 + LEN2 * sin(q1_rad) + LEN3 * sin(q2_rad);
            break;
        }
        case 4: {
            z = Z0 + LEN2 * sin(q1_rad) + LEN3 * sin(q2_rad) - 17;
            break;
        }
        default:
            Serial.print("Wrong DXL_ID!");
```

```
            break;
    }

    return z;
}

void Joint::get_coordinates() {
    //Serial.print("Angle q0: ");
    //Serial.print(q0); Serial.print("  "); Serial.print(q0_rad);
   Serial.print("  "); Serial.println(servo1.get_angle());
    //Serial.print("Angle q1: ");
    //Serial.print(q1); Serial.print("  "); Serial.print(q1_rad);
   Serial.print("  "); Serial.println(servo2.get_angle());
    //Serial.print("Angle q2: ");
    //Serial.print(q2); Serial.print("  "); Serial.print(q2_rad);
   Serial.print("  "); Serial.println(servo3.get_angle());

    Serial.print("Coordinate x: ");
    Serial.println(get_x());
    Serial.print("Coordinate y: ");
    Serial.println(get_y());
    Serial.print("Coordinate z: ");
    Serial.println(get_z());
    Serial.println();
}

Joint joint1(DXL_ID1);
Joint joint2(DXL_ID2);
Joint joint3(DXL_ID3);
Joint joint4(DXL_ID4);

#undef X0
#undef Z0
```

```
#undef LEN1
#undef LEN2
#undef LEN3
#undef LEN4

#endif
```

Calibration.h

```
#ifndef Calibration_h
#define Calibration_h


#include "Servo.h"


class Calibration {
protected:
    static void calibration_min(Servo servo);
    static void calibration_max(Servo servo);
public:
    static void calibration_setup();
};


void Calibration::calibration_max(Servo servo) {
    uint16_t servo_max_angle = 512;
    servo.set_speed(100);

    while (true) {
        Serial.println(servo.get_load());  // DEBUG
        Serial.println(servo_max_angle);
        Serial.println(servo.get_angle());
        Serial.println(" ");

        servo.set_angle(servo_max_angle);
```

```
        delay(100);
        if (servo.is_moving() == 0) servo_max_angle += 50;
        else if (servo.get_load() > 670) {
            servo_max_angle -= 50;
            servo.set_angle(servo_max_angle);
            break;
        }
    }

    servo.set_max_angle(servo_max_angle);

    Serial.println(" ");
    Serial.println(servo_max_angle);
}


void Calibration::calibration_min(Servo servo) {
    uint16_t servo_min_angle = 512;
    servo.set_speed(100);
    while (true) {
        Serial.println(servo.get_load());  // DEBUG
        Serial.println(servo_min_angle);
        Serial.println(servo.get_angle());
        Serial.println(" ");

        servo.set_angle(servo_min_angle);
        delay(100);
        if (servo.is_moving() == 0) servo_min_angle -= 50;
        else if (servo.get_load() > 670) {
            servo_min_angle += 50;
            servo.set_angle(servo_min_angle);
            break;
        }
    }
```

```
    servo.set_min_angle(servo_min_angle);


    Serial.println(" ");
    Serial.println(servo_min_angle);
}



void Calibration::calibration_setup() {
    calibration_max(servo2);
    calibration_min(servo2);
    servo2.set_angle((servo2.get_min_angle()                    +
   servo2.get_max_angle()) / 2);
    delay(2000);
    calibration_max(servo3);
    calibration_min(servo3);
}



#endif
```