Одеський національний університет імені І.І.Мечникова
(повне найменування вищого навчального закладу)

Інститут математики, економіки і механіки
(повне найменування інституту/факультету)

Кафедра теоретичної механіки
(повна назва кафедри)

Дипломна робота

Магістр
(освітньо-кваліфікаційний рівень)

на тему: «Чисельне моделювання складної механічної системи на прикладі маніпулятора»
«Численное моделирование сложной механической системы на примере манипулятора»
«Numerical modeling of complex mechanical systems on the example of the manipulator "»

Виконав: студент денної форми навчання
напряму підготовки 8.04020201 Теоретична та прикладна механіка
Бондаренко Максим Олександрович

Керівник Професор Асланов.С.К. _____

Рецензент доцент Рачинська А.Л. _____

Рекомендовано до захисту:
Протокол засідання кафедри
№ 10 від 13 червня 2016 р.

Захищено на засіданні ЕК № 2
протокол № 2 від 22.06 2016 р.
Оцінка добре / B / 85
(за національною шкалою, шкалою ECTS, бали)

Завідувач кафедри д.ф.-м. наук, проф. Асланов С.К.
_____
(підпис)

Голова ЕК д. техн.наук, проф. Волков В.Е.
_____
(підпис)

Одеса – 2016

Содержание

Введение

Одной из важнейших проблем современной робототехники является создание эффективных методов, моделей и алгоритмов для решения задач механики его исполнительного органа - манипулятора. Манипуляционная система робота, как правило, представляет собой сложный пространственный механизм с множеством степеней подвижности, позволяющий его рабочему органу (схвату или инструменту) совершать разнообразные движения в пространстве, а также обеспечивать его ориентацию. Конструктивно манипулятор состоит из следующих основных узлов: несущих конструкций, приводов, передаточных механизмов и исполнительных механизмов (рука манипулятора), снабженных захватными устройствами, в которых закрепляются перемещаемые объекты (груз, деталь, инструменты и т.п.). Рука манипулятора состоит из совокупности подвижно соединенных звеньев и служит для непосредственной реализации транспортных и ориентирующих движений перемещаемого объекта. Отдельные звенья манипулятора соединяются между собой кинематическими парами - устройствами, ограничивающими число степеней свободы относительного перемещения звеньев. В большинстве конструкций манипуляционных роботов используются так называемые пары пятого класса (вращательные или поступательные), обеспечивающие одну степень свободы относительного перемещения каждой пары подвижно соединенных звеньев.

Важнейшей задачей современной робототехники является создание более совершенных систем управления роботов, что требует, прежде всего, развития исследований в области кинематики и динамики, а также синтеза алгоритмов управления движением манипулятора. При этом большинство возникающих технических задач робототехники можно свести к двум взаимосвязанным научным проблемам - механике роботов (задачи кинематики и динамики) и управлению их движением.

Первоочередными научными задачами, относящимися к первой проблеме, являются математическое описание роботов и, прежде всего, их манипуляторов, включая разработку методов их математического моделирования, обоснование и формулирование критериев качества, в том числе кинематических критериев, построение программных траекторий движения манипуляторов, разработка методов кинематического и динамического их анализа и синтеза. Решение перечисленных задач требует применения хорошо развитого математического аппарата и алгоритмов, ориентированных на использование ЭВМ. Современная робототехника позволяет успешно решать основные задачи кинематики, динамики и управления роботами. В работах отечественных исследователей достаточно подробно рассмотрены указанные проблемы, показаны принципы построения математических моделей (ММ) манипуляторов и основные методы решения задач робототехники. Вместе с тем, сложность механической системы манипулятора, а также ряд существенных особенностей, присущих роботу как объекту управления (упругая податливость звеньев и передаточных механизмов, взаимовлияние степеней подвижности, неоднозначность решения некоторых задач, наличие ограничений различного рода и др.) зачастую приводит к чрезвычайной сложности получаемых моделей, что затрудняет решение многих задач механики роботов. В связи с этим весьма актуальным является поиск новых методов и подходов к построению более компактных и удобных ММ манипуляторов.

В современной робототехнике сложилась тенденция решать задачи кинематики и динамики так, чтобы достаточно полно изложить теорию робототехнических систем, разработать методы исследования кинематики и динамики управления манипуляторами и создать предпосылки для построения систем автоматизированного проектирования роботов. Так, при описании кинематики и динамики манипуляторов наиболее удобным и хорошо разработанным может считаться аппарат, использующий понятие однородных координат. Однако, при большом числе степеней подвижности манипулятора

возникает проблема, связанная с чрезвычайной громоздкостью и сложностью получаемых моделей, вызванных существенной вычислительной избыточностью метода. Это обуславливает поиск новых методов и подходов для более рационального описания пространственного движения звеньев манипулятора.

Одним из перспективных путей получения ММ манипуляторов является использование таких кинематических параметров, которые позволят получать наиболее рациональные с точки зрения вычислительной сложности модели манипуляторов, удобные для практического использования. Так, в последние годы для описания движения и ориентации твердых тел и, в частности, летательных аппаратов, успешно используются такие кинематические параметры, как Родрига-Гамильтона, Кейли-Клейна, кватернионы и др. Их использование в сочетании с традиционными параметрами (углы Эйлера-Крылова, направляющие косинусы) позволяет в ряде случаев получить весьма эффективные модели и алгоритмы, описывающие движение и ориентацию твердого тела в пространстве. Использование дуальных аналогов различных кинематических параметров, применяемых для описания вращательного движения твердого тела, позволяет в задачах кинематики перейти к описанию произвольного пространственного движения. Аппарат винтового исчисления также повышает эффективность решения задач пространственного движения.

Увеличивающаяся сложность управления современными системами, наличие процессов, характеризующихся неопределенностями, которые не могут быть описаны статистически, все возрастающая размерность решаемых задач и другие факторы привели к попыткам применения методов искусственного интеллекта (ИИ) к решению сложных технических, социальных, экономических и других проблем. Попытки использования таких подходов, в частности, нейросетевого, в ряде случаев дают положительные результаты и показывают их перспективность. В задачах робототехники применение методов ИИ, и, в частности, искусственных нейронных сетей отражено в работах Юревича Е.И., Макарова И.М., Лохина В.М., и др. авторов.

Вместе с тем, использование методов ИИ в механике роботов встречается весьма редко и положительный опыт их применения в данной сфере можно считать незначительным и недостаточно осмысленным. Более того, немногочисленные положительные попытки использования методов ИИ в задачах механики роботов дают основание считать, что их использование без связи с другими перспективными подходами, не всегда дает желаемый положительный эффект.

Таким образом, можно предположить, что наиболее сложные задачи современной робототехники могут быть успешно решены, во-первых, на основе создания и выбора новых, нетрадиционных математических объектов и методов, позволяющих наиболее компактно описывать угловое и пространственное движение твердого тела. Во-вторых, с помощью использования современных подходов и, в частности, методов ИИ. И, в-третьих, что наиболее важно, благодаря созданию комбинированных методов, позволяющих системно использовать достоинства как применяемого математического аппарата для создания ММ манипулятора, так и новые подходы и математические модели, включая методы ИИ. Более того, положительный опыт применения этих методов при решении задач механики роботов целесообразно накапливать, систематизировать и использовать при решении вновь возникающих задач.

Объектом исследования данной работы является конструкция семизвенного робота-манипулятора типа биотической руки.

Предметом исследования являются кинематика и динамика конструкции, изменение пространственной конфигурации манипулятора в процессе его движения под программным управлением.

Целью работы является разработка компьютерной модели семизвенного робота-манипулятора типа биотической руки, реализация программы расчёта его кинематических и динамических характеристик, визуализация движения механизма средствами пространственной компьютерной графики, отражающей

реально наблюдаемые параметры относительного движения элементов конструкции манипулятора.

Заключение

В соответствии с поставленной целью работы построена кинематическая модель семизвенного манипулятора, реализованная в виде матричных уравнений, связывающих обобщённые координаты механизма с декартовыми координатами узлов сочленения звеньев. Эта модель используется для решения задачи определения положения звеньев замкнутой цепи исполнительного механизма по обобщённым координатам, и обратной задачи определения значений углов относительных поворотов звеньев по координатам заданного положения рабочего органа.

Разработаны алгоритмы решения прямой и обратной задач с возможностью их модификации по числу звеньев и классу вращательных кинематических пар.

Построена динамическая модель семизвенного манипулятора в форме системы уравнений Лагранжа 2-го рода с начальными условиями. Уравнения разрешены относительно вторых производных обобщённых координат системы. Выбрана стратегия управления рабочим органом манипулятора, а именно закон изменения управляющих моментов.

Однако численно реализовать до конца построенную модель движения манипулятора не удалось. Анализ результатов работы программы показал, что при выбранных законах управляющих моментов, углы поворотов звеньев манипулятора принимают значения, не соответствующие условиям задачи.

Приложение

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Microsoft.DirectX;
using Microsoft.DirectX.Direct3D;
using CLM = WindowsFormsApplication10.Class_Matrix;

namespace WindowsFormsApplication10
{
    public partial class Form1 : Form
    {
        Device d3d;
        Mesh Link1;
        Material Link1Material;
        Mesh Link2;
        Material Link2Material;
        Mesh Link3;
        Material Link3Material;
        Mesh Link4;
        Material Link4Material;
        Mesh Link5;
        Material Link5Material;
        Mesh Link6;
        Material Link6Material;
        Mesh Link7;
        Material Link7Material;
        Mesh Link81;
        Material Link81Material;
        Mesh Link82;
        Material Link82Material;
        Mesh Link83;
        Material Link83Material;
        Mesh Link91;
        Material Link91Material;
        Mesh Link92;
        Material Link92Material;
        Mesh Link93;
        Material Link93Material;
        Mesh grip1;
        Material grip1Material;
        Mesh grip2;
        Material grip2Material;
        Mesh grip3;
        Material grip3Material;
        Mesh vectorA;
        Material vectorAMaterial;
        Mesh axisX;
        Material axisXMaterial;
        Mesh axisY;
        Material axisYMaterial;
        Mesh axisZ;
        Material axisZMaterial;

        int N;
        int M = 0;
        double dt;
        public double e = Math.Pow(10, -5);
        double xm0 = 0, ym0 = 0, zm0 = 0;
        double d0 = 0;
        double[] xm = new double[14];
```

```
        double[] ym = new double[14];
        double[] zm = new double[14];
        // длина звена 1
        double d_1 = 1;
        // длина звена 2
        double d_2 = 0.05;
        //масса звена 2
        //double m_2=0;
        // длина звена 3
        double d_3 = 0.5;
        //масса звена 3
        double m_3=0.5;
        // длина звена 4
        double d_4 = 0.05;
        //масса звена 4
        //double m_4;
        // длина звена 5
        double d_5 = 0.25;
        //масса звена 5
        double m_5=0.25;
        // длина звена 6
        double d_6 = 0.05;
        //масса звена 6
        //double m_6;
        // длина звена 7
        double d_7 = 0.15;
        //масса звена 7
        double m_7=0.15;

        // длина первой фаланги пальца схвата
        double d_8 = 0.05;
        // длина второй фаланги пальца схвата
        double d_9 = 0.05;

        //длина радиус-вектора точки A
        double d_A = 0;
        //масса груза
        double m_g;

        // переменная величина
        double angle;

        double gamma_x1 = Math.PI / 2, gamma_y1 = Math.PI / 6, gamma_z1 = Math.PI /
3;
        double gamma_x2, gamma_y2, gamma_z2;
        double gamma_x3, gamma_y3, gamma_z3;

        float xs = -0.5f, ys = -0.8f, zs = 1.8f;

        // координаты начала звена 1 в неподвижной системы
        double[] Segment_O0O0 = new double[3];
        // вектор звена 1 в координатах неподвижной системы   OXYZ
        double[] Segment_O0O1 = new double[3];
        // вектор звена 2 в координатах неподвижной системы   OXYZ
        double[] Segment_O0O2 = new double[3];
        // вектор звена 2 в координатах подвижной системы   O1X1Y1Z1
        double[] Segment_O1O2 = new double[3];
        // вектор звена 3 в координатах неподвижной системы   OXYZ
        double[] Segment_O0O3 = new double[3];
        // вектор звена 3 в координатах подвижной системы   O2X2Y2Z2
        double[] Segment_O2O3 = new double[3];
        // вектор звена 4 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O4 = new double[3];
        // вектор звена 4 в координатах подвижной системы   03X3Y3Z3
        double[] Segment_O3O4 = new double[3];
        // вектор звена 5 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O5 = new double[3];
```

```
        // вектор звена 5 в координатах подвижной системы O4X4Y4Z4
        double[] Segment_O4O5 = new double[3];
        // вектор звена 6 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O6 = new double[3];
        // вектор звена 6 в координатах подвижной системы   O5X5Y5Z5
        double[] Segment_O5O6 = new double[3];
        // вектор звена 7 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O7 = new double[3];
        // вектор звена 7 в координатах подвижной системы O6X6Y6Z6
        double[] Segment_O6O7 = new double[3];


        // вектор звена 8_1 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O81 = new double[3];
        // вектор звена 8_1 в координатах подвижной системы O7X7Y7Z7
        double[] Segment_O7O81 = new double[3];

        // вектор звена 8_2 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O82 = new double[3];
        // вектор звена 8_2 в координатах подвижной системы O7X7Y7Z7
        double[] Segment_O7O82 = new double[3];

        // вектор звена 8_3 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O83 = new double[3];
        // вектор звена 8_3 в координатах подвижной системы O7X7Y7Z7
        double[] Segment_O7O83 = new double[3];

        // вектор звена 9_1 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O91 = new double[3];
        // вектор звена 9_1 в координатах подвижной системы O81X81Y81Z81
        double[] Segment_O81O91 = new double[3];

        // вектор звена 9_2 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O92 = new double[3];
        // вектор звена 9_2 в координатах подвижной системы O82X82Y82Z82
        double[] Segment_O82O92 = new double[3];

        // вектор звена 9_3 в координатах неподвижной системы  OXYZ
        double[] Segment_O0O93 = new double[3];
        // вектор звена 9_3 в координатах подвижной системы O83X83Y83Z83
        double[] Segment_O83O93 = new double[3];


        // радиус-вектор конца первого звена в неподвижной системе
        double[] RadiusVector_O0O0 = new double[3];
        // радиус-вектор конца второго звена (начала первого звена) в неподвижной
системе
        double[] RadiusVector_O0O1 = new double[3];
        // радиус-вектор конца третьего звена (начала второго звена) в неподвижной
системе
        double[] RadiusVector_O0O2 = new double[3];
        // радиус-вектор конца четвёртого звена (начала третьего звена) в
неподвижной системе
        double[] RadiusVector_O0O3 = new double[3];
        // радиус-вектор конца пятого звена (начала четвёртого звена) в неподвижной
системе
        double[] RadiusVector_O0O4 = new double[3];
        // радиус-вектор конца шестого звена (начала пятого звена)в неподвижной
системе
        double[] RadiusVector_O0O5 = new double[3];
        // радиус-вектор конца седьмого звена (начала шестого звена) в неподвижной
системе
        double[] RadiusVector_O0O6 = new double[3];
        // радиус-вектор конца седьмого звена в неподвижной системе
        double[] RadiusVector_O0O7 = new double[3];
        // радиус-вектор конца первой фаланги первого пальца схвата в неподвижной
системе
```

```
        double[] RadiusVector_O0O81 = new double[3];
        // радиус-вектор конца первой фаланги второго пальца схвата в неподвижной
системе
        double[] RadiusVector_O0O82 = new double[3];
        // радиус-вектор конца первой фаланги третьего пальца схвата в неподвижной
системе
        double[] RadiusVector_O0O83 = new double[3];
        // радиус-вектор конца второй фаланги первого пальца схвата в неподвижной
системе
        double[] RadiusVector_O0O91 = new double[3];
        // радиус-вектор конца второй фаланги второго пальца схвата в неподвижной
системе
        double[] RadiusVector_O0O92 = new double[3];
        // радиус-вектор конца второй фаланги третьего пальца схвата в неподвижной
системе
        double[] RadiusVector_O0O93 = new double[3];
        // вектор A ориентации плоскости окружности-траектории
        double[] Vector_A = new double[3];
        //радиус R окружности-траектории
        double R;
        // радиус-вектор заданной точки A в неподвижной системе
        double[] RadiusVector_O0A = new double[3];
        //угол поворота (j+1)-го звена относительно оси OjZj
        double[] AngleRotation = {0,0,0,0,0,0};
        //угол поворота (j+1)-го звена относительно оси OjZj при перемещении между i
и (i+1) опорными точками
        double[] AngleRotationi = { 0, 0, 0, 0, 0, 0 };
        //угловая скорость поворота (j+1)-го звена относительно оси OjZj
        double[] Speed_AngleRotation = { 0, 0, 0, 0, 0, 0 };
        //угловая скорость поворота (j+1)-го звена относительно оси OjZj при
перемещении между i и (i+1) опорными точками
        double[] Speed_AngleRotationi = { 0, 0, 0, 0, 0, 0 };
        // угол поворота второй фаланги i-го пальца схвата относительно оси O8iZ8i
        double AngleRotation_haper = 0;

        // матрица преобразования координат системы O1X1Y1Z1 в координаты системы
OXYZ
        double[,] MatrixTransformation_O1O0 = { { -1, 0, 0 }, {0, 0, 1 }, { 0, 1, 0
} };
        // матрица преобразования координат системы O2X2Y2Z2 в координаты системы
OXYZ
        double[,] MatrixTransformation_O2O0 = new double[3, 3];
        // матрица преобразования координат системы O3X3Y3Z3 в координаты системы
OXYZ
        double[,] MatrixTransformation_O3O0 = new double[3, 3];
        // матрица преобразования координат системы O4X4Y4Z4 в координаты системы
OXYZ
        double[,] MatrixTransformation_O4O0 = new double[3, 3];
        // матрица преобразования координат системы O5X5Y5Z5 в координаты системы
OXYZ
        double[,] MatrixTransformation_O5O0 = new double[3, 3];
        // матрица преобразования координат системы O6X6Y6Z6 в координаты системы
OXYZ
        double[,] MatrixTransformation_O6O0 = new double[3, 3];
        // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
OXYZ
        double[,] MatrixTransformation_O7O0 = new double[3, 3];
        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы OXYZ
        double[,] MatrixTransformation_O81O0 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы OXYZ
        double[,] MatrixTransformation_O82O0 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы OXYZ
        double[,] MatrixTransformation_O83O0 = new double[3, 3];
```

```
        // матрица преобразования координат системы O2X2Y2Z2 в координаты системы
O1X1Y1Z1
        double[,] MatrixTransformation_O2O1 = { { 1, 0, 0 }, { 0, 0, -1 }, { 0, 1, 0
} };
        // матрица преобразования координат системы O3X3Y3Z3 в координаты системы
O1X1Y1Z1
        double[,] MatrixTransformation_O3O1 = new double[3, 3];
        // матрица преобразования координат системы O4X4Y4Z4 в координаты системы
O1X1Y1Z1
        double[,] MatrixTransformation_O4O1 = new double[3, 3];
        // матрица преобразования координат системы O5X5Y5Z5 в координаты системы
O1X1Y1Z1
        double[,] MatrixTransformation_O5O1 = new double[3, 3];
        // матрица преобразования координат системы O6X6Y6Z6 в координаты системы
O1X1Y1Z1
        double[,] MatrixTransformation_O6O1 = new double[3, 3];
        // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
O1X1Y1Z1
        double[,] MatrixTransformation_O7O1 = new double[3, 3];
        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O1X1Y1Z1
        double[,] MatrixTransformation_O81O1 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O1X1Y1Z1
        double[,] MatrixTransformation_O82O1 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O1X1Y1Z1
        double[,] MatrixTransformation_O83O1 = new double[3, 3];

        // матрица преобразования координат системы O3X3Y3Z3 в координаты системы
O2X2Y2Z2
        double[,] MatrixTransformation_O3O2 = { { -1, 0, 0 }, { 0, 0, 1 }, { 0, 1, 0
} };
        // матрица преобразования координат системы O4X4Y4Z4 в координаты системы
O2X2Y2Z2
        double[,] MatrixTransformation_O4O2 = new double[3, 3];
        // матрица преобразования координат системы O5X5Y5Z5 в координаты системы
O2X2Y2Z2
        double[,] MatrixTransformation_O5O2 = new double[3, 3];
        // матрица преобразования координат системы O6X6Y6Z6 в координаты системы
O2X2Y2Z2
        double[,] MatrixTransformation_O6O2 = new double[3, 3];
        // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
O2X2Y2Z2
        double[,] MatrixTransformation_O7O2 = new double[3, 3];
        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O2X2Y2Z2
        double[,] MatrixTransformation_O81O2 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O2X2Y2Z2
        double[,] MatrixTransformation_O82O2 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O2X2Y2Z2
        double[,] MatrixTransformation_O83O2 = new double[3, 3];


        // матрица преобразования координат системы O4X4Y4Z4 в координаты системы
O3X3Y3Z3
        double[,] MatrixTransformation_O4O3 = { { 1, 0, 0 }, { 0, 0, -1 }, { 0, 1, 0
} };
        // матрица преобразования координат системы O5X5Y5Z5 в координаты системы
O3X3Y3Z3
        double[,] MatrixTransformation_O5O3 = new double[3, 3];
        // матрица преобразования координат системы O6X6Y6Z6 в координаты системы
O3X3Y3Z3
        double[,] MatrixTransformation_O6O3 = new double[3, 3];
```

```
        // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
O3X3Y3Z3
        double[,] MatrixTransformation_O7O3 = new double[3, 3];
        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O3X3Y3Z3
        double[,] MatrixTransformation_O81O3 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O3X3Y3Z3
        double[,] MatrixTransformation_O82O3 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O3X3Y3Z3
        double[,] MatrixTransformation_O83O3 = new double[3, 3];


        // матрица преобразования координат системы O5X5Y5Z5 в координаты системы
O4X4Y4Z4
        double[,] MatrixTransformation_O5O4 = { { -1, 0, 0 }, { 0, 0, 1 }, { 0, 1, 0
} };
        // матрица преобразования координат системы O6X6Y6Z6 в координаты системы
O4X4Y4Z4
        double[,] MatrixTransformation_O6O4 = new double[3, 3];
        // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
O4X4Y4Z4
        double[,] MatrixTransformation_O7O4 = new double[3, 3];
        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O4X4Y4Z4
        double[,] MatrixTransformation_O81O4 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O4X4Y4Z4
        double[,] MatrixTransformation_O82O4 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O4X4Y4Z4
        double[,] MatrixTransformation_O83O4 = new double[3, 3];

        // матрица преобразования координат системы O6X6Y6Z6 в координаты системы
O5X5Y5Z5
        double[,] MatrixTransformation_O6O5 = { { 1, 0, 0 }, { 0, 0, -1 }, { 0, 1, 0
} };
        // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
O5X5Y5Z5
        double[,] MatrixTransformation_O7O5 = new double[3, 3];
        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O5X5Y5Z5
        double[,] MatrixTransformation_O81O5 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O5X5Y5Z5
        double[,] MatrixTransformation_O82O5 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O5X5Y5Z5
        double[,] MatrixTransformation_O83O5 = new double[3, 3];

        // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
O6X6Y6Z6
        double[,] MatrixTransformation_O7O6 = { { 1, 0, 0 }, { 0, 0, -1 }, { 0, 1, 0
} };
        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O6X6Y6Z6
        double[,] MatrixTransformation_O81O6 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O6X6Y6Z6
        double[,] MatrixTransformation_O82O6 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O6X6Y6Z6
        double[,] MatrixTransformation_O83O6 = new double[3, 3];

        // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O7X7Y7Z7
```

```
        double[,] MatrixTransformation_O81O7 = new double[3, 3];
        // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O7X7Y7Z7
        double[,] MatrixTransformation_O82O7 = new double[3, 3];
        // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O7X7Y7Z7
        double[,] MatrixTransformation_O83O7 = new double[3, 3];
        // матрица преобразования координат системы, связаной с плоскостью
траектории схвата в координаты системы O7X7Y7Z7
        double[,] MatrixTransformation_OAO0 = new double[3, 3];

        // матрица поворота системы координат O1X1Y1Z1 в системе координат OXYZ
        double[,] MatrixRotation_1 = new double[3, 3];
        // матрица поворота системы координат O2X2Y2Z2 в системе координат O1X1Y1Z1
        double[,] MatrixRotation_2 = new double[3, 3];
        // матрица поворота системы координат O3X3Y3Z3 в системе координат O2X2Y2Z2
        double[,] MatrixRotation_3 = new double[3, 3];
        // матрица поворота системы координат O4X4Y4Z4 в системе координат O3X3Y3Z3
        double[,] MatrixRotation_4 = new double[3, 3];
        // матрица поворота системы координат O5X5Y5Z5 в системе координат O4X4Y4Z4
        double[,] MatrixRotation_5 = new double[3, 3];
        // матрица поворота системы координат O6X6Y6Z6 в системе координат O5X5Y5Z5
        double[,] MatrixRotation_6 = new double[3, 3];
        // матрица поворота системы координат O81X81Y81Z81 в системе координат
O7X7Y7Z7
        double[,] MatrixRotation_81 = new double[3, 3];
        // матрица поворота системы координат O82X82Y82Z82 в системе координат
O7X7Y7Z7
        double[,] MatrixRotation_82 = new double[3, 3];
        // матрица поворота системы координат O83X83Y83Z83 в системе координат
O7X7Y7Z7
        double[,] MatrixRotation_83 = new double[3, 3];
        // матрица коэффициентов при вторых производных углов поворотов звеньев
манипулятора
        double[,] Matrix_A =new double[6, 6];
        double[,] Matrix_AI = new double[6, 6];
        double[,] Matrix_D = new double[6, 6];
        //определитель матрицы Matrix_A
        double Det_Matrix_A;
        //= { { Math.Pow(d_2, 2) * (m_3 + m_5 + m_7 + m_g), d_2 * d_3 * (m_3 / 2 +
m_5 + m_7 + m_g), -d_2 * d_4 * (m_5 + m_7 + m_g), -d_2 * d_5 * (m_5 / 2 + m_7 +
m_g), d_2 * d_6 * (m_7 + m_g), d_2 * d_7 * (m_7 / 2 + m_g) }, { d_2 * d_3 * (m_3 / 2
+ m_5 + m_7 + m_g), Math.Pow(d_3, 2) * (m_3 / 3 + m_5 + m_7 + m_g), -d_3 * d_4 *
(m_5 + m_7 + m_g), -d_3 * d_5 * (m_5 / 2 + m_7 + m_g), d_3 * d_6 * (m_7 + m_g), d_3
* d_7 * (m_7 / 2 + m_g) }, { -d_2 * d_4 * (m_5 + m_7 + m_g), -d_3 * d_4 * (m_5 + m_7
+ m_g), Math.Pow(d_4, 2) * (m_5 + m_7 + m_g), d_4 * d_5 * (m_5 / 2 + m_7 + m_g), -
d_4 * d_6 * (m_7 + m_g), -d_4 * d_7 * (m_7 / 2 + m_g) }, { -d_2 * d_5 * (m_5 / 2 +
m_7 + m_g), -d_3 * d_5 * (m_5 / 2 + m_7 + m_g), d_4 * d_5 * (m_5 / 2 + m_7 + m_g),
Math.Pow(d_5, 2) * (m_5 / 3 + m_7 + m_g), -d_5 * d_6 * (m_7 + m_g), -d_5 * d_7 *
(m_7 / 2 + m_g) }, { d_2 * d_6 * (m_7 + m_g), d_3 * d_6 * (m_7 + m_g), -d_4 * d_6 *
(m_7 + m_g), -d_5 * d_6 * (m_7 + m_g), Math.Pow(d_6, 2) * (m_7 + m_g), d_6 * d_7 *
(m_7 / 2 + m_g) }, { d_2 * d_7 * (m_7 / 2 + m_g), d_3 * d_7 * (m_7 / 2 + m_g), -d_4
* d_7 * (m_7 / 2 + m_g), -d_5 * d_7 * (m_7 / 2 + m_g), d_6 * d_7 * (m_7 / 2 + m_g),
Math.Pow(d_7, 2) * (m_7 / 3 + m_g) } };
        double Det_Matrix_D;

        double [] Vector_Moment= new double[6];
        // вектор коэффициентов при второй степени обобщённой переменной
        double[] a = { 0, 0, 0, 0, 0, 0 };
        // вектор коэффициентов при первой степени обобщённой переменной
        double[] b = { 0, 0, 0, 0, 0, 0 };
        // вектор коэффициентов при нулевой степени обобщённой переменной
        double[] c = { 0, 0, 0, 0, 0, 0 };

        // вектор коэффициентов при второй степени обобщённой переменной
        double[] fa = { 0, 0, 0, 0, 0, 0 };
        // вектор коэффициентов при первой степени обобщённой переменной
```

```
        double[] fb = { 0, 0, 0, 0, 0, 0 };
        // вектор коэффициентов при нулевой степени обобщённой переменной
        double[] fc = { 0, 0, 0, 0, 0, 0 };

        // вектор функций правой части системы дифур в каноническом виде
        double[] F = { 0, 0, 0, 0, 0, 0 };

        // вспомогательные углы поворота вокруг осей OX, OY, OZ
        double RotX0 = 0, RotY0 = 0, RotZ0 = 0;

        // массив углов поворотов вокруг оси OX
        double[] RotX = new double[14];
        // массив углов поворотов вокруг оси OY
        double[] RotY = new double[14];
        // массив углов поворотов вокруг оси OZ
        double[] RotZ = new double[14];

      //  double[,] MatrixTest = { {1,0,0,0,0,0}, {0,1,0,0,0,0 }, {0,0,1,0,0,0 },
{0,0,0,1,0,0 }, { 0,0,0,0,1,1}, {0,0,0,0,0,1 } };

        public Form1()
        {
            InitializeComponent();
            this.SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.Opaque,
true);
            d3d = null;
            Link1 = null;
            Link2 = null;
            Link3 = null;
            Link4 = null;
            Link5 = null;
            Link6 = null;
            Link7 = null;
            Link81 = null;
            Link82 = null;
            Link83 = null;
            Link91 = null;
            Link93 = null;
            grip1 = null;
            grip2 = null;
            grip3 = null;
            axisX = null;
            axisY = null;
            axisZ = null;


            //     MatrixDeterminantN(MatrixTest, 6);


            Matrix_A[0, 0] = Math.Pow(d_2, 2) * (m_3 + m_5 + m_7 + m_g);
            Matrix_A[0, 1] = d_2 * d_3 * (m_3 / 2 + m_5 + m_7 + m_g);
            Matrix_A[0, 2] = -d_2 * d_4 * (m_5 + m_7 + m_g);
            Matrix_A[0, 3] = -d_2 * d_5 * (m_5 / 2 + m_7 + m_g);
            Matrix_A[0, 4] = d_2 * d_6 * (m_7 + m_g);
            Matrix_A[0, 5] = d_2 * d_7 * (m_7 / 2 + m_g);

            Matrix_A[1, 0] = d_2 * d_3 * (m_3 / 2 + m_5 + m_7 + m_g);
            Matrix_A[1, 1] = Math.Pow(d_3, 2) * (m_3 / 3 + m_5 + m_7 + m_g);
            Matrix_A[1, 2] = -d_3 * d_4 * (m_5 + m_7 + m_g);
            Matrix_A[1, 3] = -d_3 * d_5 * (m_5 / 2 + m_7 + m_g);
            Matrix_A[1, 4] = d_3 * d_6 * (m_7 + m_g);
            Matrix_A[1, 5] = d_3 * d_7 * (m_7 / 2 + m_g);

            Matrix_A[2, 0] = -d_2 * d_4 * (m_5 + m_7 + m_g);
            Matrix_A[2, 1] = -d_3 * d_4 * (m_5 + m_7 + m_g);
            Matrix_A[2, 2] = Math.Pow(d_4, 2) * (m_5 + m_7 + m_g);
            Matrix_A[2, 3] = d_4 * d_5 * (m_5 / 2 + m_7 + m_g);
```

```
            Matrix_A[2, 4] = -d_4 * d_6 * (m_7 + m_g);
            Matrix_A[2, 5] = -d_4 * d_7 * (m_7 / 2 + m_g);

            Matrix_A[3, 0] = -d_2 * d_5 * (m_5 / 2 + m_7 + m_g);
            Matrix_A[3, 1] = -d_3 * d_5 * (m_5 / 2 + m_7 + m_g);
            Matrix_A[3, 2] = d_4 * d_5 * (m_5 / 2 + m_7 + m_g);
            Matrix_A[3, 3] = Math.Pow(d_5, 2) * (m_5 / 3 + m_7 + m_g);
            Matrix_A[3, 4] = -d_5 * d_6 * (m_7 + m_g);
            Matrix_A[3, 5] = -d_5 * d_7 * (m_7 / 2 + m_g);

            Matrix_A[4, 0] = d_2 * d_6 * (m_7 + m_g);
            Matrix_A[4, 1] = d_3 * d_6 * (m_7 + m_g);
            Matrix_A[4, 2] = -d_4 * d_6 * (m_7 + m_g);
            Matrix_A[4, 3] = -d_5 * d_6 * (m_7 + m_g);
            Matrix_A[4, 4] = Math.Pow(d_6, 2) * (m_7 + m_g);
            Matrix_A[4, 5] = d_6 * d_7 * (m_7 / 2 + m_g);

            Matrix_A[5, 0] = d_2 * d_7 * (m_7 / 2 + m_g);
            Matrix_A[5, 1] = d_3 * d_7 * (m_7 / 2 + m_g);
            Matrix_A[5, 2] = -d_4 * d_7 * (m_7 / 2 + m_g);
            Matrix_A[5, 3] = -d_5 * d_7 * (m_7 / 2 + m_g);
            Matrix_A[5, 4] = d_6 * d_7 * (m_7 / 2 + m_g);
            Matrix_A[5, 5] = Math.Pow(d_7, 2) * (m_7 / 3 + m_g);

            //for(int i=0; i <= 5; i++)
            //{
            //   for (int j = 0; j <= 5; j++ )
            //   {
            //       Matrix_D[i, j] = Matrix_A[i, j];
            //   }
            //}


          //MatrixDeterminantN(Matrix_A, 6);
            //Det_Matrix_A = Det_Matrix_D;

            Det_Matrix_A = CLM.Determinant_J_0(Matrix_A);

            MatrixInvertion_N(Matrix_A, Matrix_AI,6);

            //координаты начала отсчёта системы OXYZ
            Segment_O0O0[0] = 0; Segment_O0O0[1] = 0; Segment_O0O0[2] = 0;
            //координаты радиус-вектора первого звена в системе OXYZ
            Segment_O0O1[0] = 0; Segment_O0O1[1] = d_1; Segment_O0O1[2] = 0;
            //координаты радиус-вектора второго звена в системе O1X1Y1Z1
            Segment_O1O2[0] = 0; Segment_O1O2[1] = d_2; Segment_O1O2[2] = 0.0;
            //координаты радиус-вектора третьего звена в системе O2X2Y2Z2
            Segment_O2O3[0] = 0; Segment_O2O3[1] = d_3; Segment_O2O3[2] = 0.0;
            //координаты радиус-вектора четвёртого звена в системе O3X3Y3Z3
            Segment_O3O4[0] = 0; Segment_O3O4[1] = d_4; Segment_O3O4[2] = 0.0;
            //координаты радиус-вектора пятого звена в системе O4X4Y4Z4
            Segment_O4O5[0] = 0; Segment_O4O5[1] = d_5; Segment_O4O5[2] = 0.0;
            //координаты радиус-вектора шестого звена в системе O5X5Y5Z5
            Segment_O5O6[0] = 0; Segment_O5O6[1] = d_6; Segment_O5O6[2] = 0.0;
            //координаты радиус-вектора седьмого звена в системе O6X6Y6Z6
            Segment_O6O7[0] = 0; Segment_O6O7[1] = d_7; Segment_O6O7[2] = 0.0;

            // орт k7 в системе O7X7Y7Z7
            double[] k7 = { 0, 0, 1 };


            //координаты радиус-вектора первой фаланги первого пальца схвата в
системе O7X7Y7Z7
            Segment_O7O81[0] = d_8 * Math.Cos(gamma_x1); Segment_O7O81[1] = d_8 *
Math.Cos(gamma_y1); Segment_O7O81[2] = d_8 * Math.Cos(gamma_z1);
            double alfa1 = Math.Atan(Segment_O7O81[1] / Segment_O7O81[0]);
```

```
            double AbsO7O81 = Math.Sqrt(Math.Pow(Segment_O7O81[0], 2) +
Math.Pow(Segment_O7O81[1], 2) + Math.Pow(Segment_O7O81[2], 2));
            // орт j81
            double[] j81 = { Segment_O7O81[0] / AbsO7O81, Segment_O7O81[1] /
AbsO7O81, Segment_O7O81[2] / AbsO7O81 };
            // орт k81
            double[] k1 = new double[3];
            VMultiplicationVector(j81, k7, k1);
            double Absk1 = Math.Sqrt(Math.Pow(k1[0], 2) + Math.Pow(k1[1], 2) +
Math.Pow(k1[2], 2));
            double[] k81 = { k1[0] / Absk1, k1[1] / Absk1, k1[2] / Absk1 };
            // орт i81
            double[] i81 = new double[3];
            VMultiplicationVector(k81, j81, i81);

            MatrixTransformation_O81O7[0, 0] = i81[0];
            MatrixTransformation_O81O7[0, 1] = i81[1];
            MatrixTransformation_O81O7[0, 2] = i81[2];
            MatrixTransformation_O81O7[1, 0] = j81[0];
            MatrixTransformation_O81O7[1, 1] = j81[1];
            MatrixTransformation_O81O7[1, 2] = j81[2];
            MatrixTransformation_O81O7[2, 0] = k81[0];
            MatrixTransformation_O81O7[2, 1] = k81[1];
            MatrixTransformation_O81O7[2, 2] = k81[2];

            //координаты радиус-вектора первой фаланги второго пальца схвата в
системе O7X7Y7Z7
            double alfa2 = alfa1 + 2 * Math.PI / 3;
            gamma_z2 = gamma_z1;
            Segment_O7O82[0] = d_8 * Math.Sin(gamma_z2) * Math.Cos(alfa2);
            gamma_x2 = Math.Acos(Segment_O7O82[0]);
            Segment_O7O82[1] = d_8 * Math.Sin(gamma_z2) * Math.Sin(alfa2);
            gamma_y2 = Math.Acos(Segment_O7O82[1]);
            Segment_O7O82[2] = Segment_O7O81[2];
            double AbsO7O82 = Math.Sqrt(Math.Pow(Segment_O7O82[0], 2) +
Math.Pow(Segment_O7O82[1], 2) + Math.Pow(Segment_O7O82[2], 2));
            // орт j82
            double[] j82 = { Segment_O7O82[0] / AbsO7O82, Segment_O7O82[1] /
AbsO7O82, Segment_O7O82[2] / AbsO7O82 };
            // орт k82
            double[] k2 = new double[3];
            VMultiplicationVector(j82, k7, k2);
            double Absk2 = Math.Sqrt(Math.Pow(k2[0], 2) + Math.Pow(k2[1], 2) +
Math.Pow(k2[2], 2));
            double[] k82 = { k2[0] / Absk2, k2[1] / Absk2, k2[2] / Absk2 };
            // орт i82
            double[] i82 = new double[3];
            VMultiplicationVector(k82, j82, i82);

            MatrixTransformation_O82O7[0, 0] = i82[0];
            MatrixTransformation_O82O7[0, 1] = i82[1];
            MatrixTransformation_O82O7[0, 2] = i82[2];
            MatrixTransformation_O82O7[1, 0] = j82[0];
            MatrixTransformation_O82O7[1, 1] = j82[1];
            MatrixTransformation_O82O7[1, 2] = j82[2];
            MatrixTransformation_O82O7[2, 0] = k82[0];
            MatrixTransformation_O82O7[2, 1] = k82[1];
            MatrixTransformation_O82O7[2, 2] = k82[2];

            //координаты радиус-вектора первой фаланги третьего пальца схвата в
системе O7X7Y7Z7
            double alfa3 = alfa2 + 2 * Math.PI / 3;
            gamma_z3 = gamma_z1;
            Segment_O7O83[0] = d_8 * Math.Sin(gamma_z3) * Math.Cos(alfa3);
            gamma_x3 = Math.Acos(Segment_O7O83[0]);
            Segment_O7O83[1] = d_8 * Math.Sin(gamma_z3) * Math.Sin(alfa3);
            gamma_y3 = Math.Acos(Segment_O7O83[1]);
```

```
            Segment_O7O83[2] = Segment_O7O81[2];
            double AbsO7O83 = Math.Sqrt(Math.Pow(Segment_O7O83[0], 2) +
Math.Pow(Segment_O7O83[1], 2) + Math.Pow(Segment_O7O83[2], 2));
            // орт j83
            double[] j83 = { Segment_O7O83[0] / AbsO7O83, Segment_O7O83[1] /
AbsO7O83, Segment_O7O83[2] / AbsO7O83 };
            // орт k83
            double[] k3 = new double[3];
            VMultiplicationVector(j83, k7, k3);
            double Absk3 = Math.Sqrt(Math.Pow(k3[0], 2) + Math.Pow(k3[1], 2) +
Math.Pow(k3[2], 2));
            double[] k83 = { k3[0] / Absk3, k3[1] / Absk3, k3[2] / Absk3 };
            // орт i82
            double[] i83 = new double[3];
            VMultiplicationVector(k83, j83, i83);

            MatrixTransformation_O83O7[0, 0] = i83[0];
            MatrixTransformation_O83O7[0, 1] = i83[1];
            MatrixTransformation_O83O7[0, 2] = i83[2];
            MatrixTransformation_O83O7[1, 0] = j83[0];
            MatrixTransformation_O83O7[1, 1] = j83[1];
            MatrixTransformation_O83O7[1, 2] = j83[2];
            MatrixTransformation_O83O7[2, 0] = k83[0];
            MatrixTransformation_O83O7[2, 1] = k83[1];
            MatrixTransformation_O83O7[2, 2] = k83[2];


            //координаты радиус-вектора второй фаланги первого пальца схвата в
системе O81X81Y81Z81
            Segment_O81O91[0] = d_9 * Math.Cos(AngleRotation_haper);
            Segment_O81O91[1] = d_9 * Math.Sin(AngleRotation_haper);
            Segment_O81O91[2] = 0;

            //координаты радиус-вектора второй фаланги второго пальца схвата в
системе O82X82Y82Z82
            Segment_O82O92[0] = d_9 * Math.Cos(AngleRotation_haper);
            Segment_O82O92[1] = d_9 * Math.Sin(AngleRotation_haper);
            Segment_O82O92[2] = 0;

            //координаты радиус-вектора второй фаланги третьего пальца схвата в
системе O83X83Y83Z83
            Segment_O83O93[0] = d_9 * Math.Cos(AngleRotation_haper);
            Segment_O83O93[1] = d_9 * Math.Sin(AngleRotation_haper);
            Segment_O83O93[2] = 0;

            Search_Node_Coordinates_Links_Of_Manipulator(AngleRotation,
AngleRotation_haper);

        }

//*************************************************************************************
************************************************

        private void button1_Click(object sender, EventArgs e)
        {
            N = Convert.ToInt32(textBox4.Text);
            dt = Convert.ToDouble(textBox5.Text);
            m_g = Convert.ToDouble(textBox6.Text);
            // матрица радиус-векторов опорных точек Ai в неподвижной системе
            double[,] RadiusVector_O0Ai = new double[N, 3];
            double[] AB = new double [3];
            double AbsAB;
            double[] ia = new double[3];
            double[] ja = new double[3];
            double[] ka = new double[3];
            //координаты перемещения схвата
            RadiusVector_O0A[0] = Convert.ToDouble(textBox1.Text);
```

```
            RadiusVector_O0A[1] = Convert.ToDouble(textBox2.Text);
            RadiusVector_O0A[2] = Convert.ToDouble(textBox3.Text);
            //текущие координаты схвата RadiusVector_O0O7[i];
            //параметрическое  задание прямой траектории схвата
            AB[0] = RadiusVector_O0A[0] - RadiusVector_O0O7[0];
            AB[1] = RadiusVector_O0A[1] - RadiusVector_O0O7[1];
            AB[2] = RadiusVector_O0A[2] - RadiusVector_O0O7[2];
            AbsAB = Math.Pow((Math.Pow(AB[0], 2) + Math.Pow(AB[1], 2) +
Math.Pow(AB[2], 2)), 0.5);
          //цикл по количеству опорных точек: для каждой оторной точки O0Ai
определяем движение манипулятора
            for (int i=0; i<=N-1; i++)
             {
               //задание опорных точек
              RadiusVector_O0Ai[i, 0] = RadiusVector_O0O7[0] + (i + 1) * AB[0] / N;
              RadiusVector_O0Ai[i, 1] = RadiusVector_O0O7[1] + (i + 1) * AB[1] / N;
              RadiusVector_O0Ai[i, 2] = RadiusVector_O0O7[2] + (i + 1) * AB[2] / N;

             }
            for (int i = 0; i <= N-1; i++)
            {
                double[] Vector_O0Ai = new double[3];
                Vector_O0Ai[0] = RadiusVector_O0Ai[i, 0];
                Vector_O0Ai[1] = RadiusVector_O0Ai[i, 1];
                Vector_O0Ai[2] = RadiusVector_O0Ai[i, 2];

                Console.Write("координаты "); Console.Write(i + 1);
Console.WriteLine("-ой опорной точки");
                Console.Write(Vector_O0Ai[0]); Console.Write("    ");
Console.Write(Vector_O0Ai[1]); Console.Write("    ");
Console.WriteLine(Vector_O0Ai[2]);

                Console.WriteLine("начальные координаты схвата в цикле по опорным
точкам");
                Console.Write(RadiusVector_O0O7[0]); Console.Write("    ");
Console.Write(RadiusVector_O0O7[1]); Console.Write("    ");
Console.WriteLine(RadiusVector_O0O7[2]);

                Search_Link_Rotation_Angles_Of_Manipulator(Vector_O0Ai);

                Console.WriteLine("углы поворота звеньев");
                Console.Write(AngleRotation[0]); Console.Write("    ");
Console.Write(AngleRotation[1]); Console.Write("    ");
Console.Write(AngleRotation[2]); Console.Write("    ");
Console.Write(AngleRotation[3]); Console.Write("    ");
Console.Write(AngleRotation[4]); Console.Write("    ");
Console.WriteLine(AngleRotation[5]);

                // моменты в узлах звеньев
                Moment(AngleRotation);

                //правая часть канонической системы
                //Canonic_Sistem_Eqwil_Motion_Of_Manipulator(AngleRotation);

                for (int j = 0; j <= 5; j++)
                {
                    AngleRotationi[j] = 0;
                    Speed_AngleRotationi[j] = 0;
                }

                while (Math.Abs(RadiusVector_O0O7[0] - Vector_O0Ai[0]) > 0.0001 ||
Math.Abs(RadiusVector_O0O7[1] - Vector_O0Ai[1]) > 0.0001 ||
Math.Abs(RadiusVector_O0O7[2] - Vector_O0Ai[2]) > 0.0001)
                {
                    M = M + 1;
```

```
                    Console.Write("перемещение схвата между "); Console.Write(i +
1); Console.Write(" и "); Console.Write(i + 2); Console.WriteLine("опорными
тосками");
                    Console.Write("точка "); Console.WriteLine(M);

                    Solve_Sistem_Eqwil_Runge_Kutta(AngleRotationi,
Speed_AngleRotationi, 6);

                    Search_Node_Coordinates_Links_Of_Manipulator(AngleRotationi,
AngleRotation_haper);

                    Console.Write("текущие координаты схвата в цикле между ");
Console.Write(i + 1); Console.Write(" и "); Console.Write(i + 2);
Console.WriteLine("опорными тосками");
                    Console.Write(RadiusVector_0007[0]); Console.Write("    ");
Console.Write(RadiusVector_0007[1]); Console.Write("    ");
Console.WriteLine(RadiusVector_0007[2]);

                    Drawing_Motion_Of_Manipulator();

                }
            }
        }
        private void Solve_Sistem_Eqwil_Runge_Kutta(double[] AngleRot, double[]
Speed_AngleRot,int n)
        {

            double[] k1 = new double [n];
            double[] k2 = new double [n];
            double[] k3 = new double [n];
            double[] k4 = new double [n];

            double[] m1 = new double [n];
            double[] m2 = new double [n];
            double[] m3 = new double [n];
            double[] m4 = new double [n];

            double[] AngleRot_2 = new double[n];
            double[] AngleRot_1 = new double[n];
            double[] AngleRot_0 = new double[n];

            double[] Speed_AngleRot_1 = new double[n];

            for (int j = 0; j <= n-1; j++)
            {
                AngleRot_0[j] = 1;
            }

            double[,] Matrix_ai = new double[n, n];
            double[,] Matrix_bi = new double[n, n];
            double[,] Matrix_ci = new double[n, n];

            double[,] Matrix_fa = new double[n, n];
            double[,] Matrix_fb = new double[n, n];
            double[,] Matrix_fc = new double[n, n];

            double[,] Matrix_faT = new double[n, n];
            double[,] Matrix_fbT = new double[n, n];
            double[,] Matrix_fcT = new double[n, n];

            double[,] Matrix_Test = new double[n, n];

            for (int i = 0; i <= n-1; i++)
            {
                for (int j = 0; j <= n-1; j++)
                {
                    if (i == j)
```

```
            {
                Matrix_ai[i, j] = a[i];
                Matrix_bi[i, j] = b[i];
                Matrix_ci[i, j] = c[i];
            }
            else
            {
                Matrix_ai[i, j] = 0;
                Matrix_bi[i, j] = 0;
                Matrix_ci[i, j] = 0;

            }
        }
    }

    MatrixMultiplication(Matrix_AI, Matrix_ai, Matrix_fa);
    MatrixMultiplication(Matrix_AI, Matrix_bi, Matrix_fb);
    MatrixMultiplication(Matrix_AI, Matrix_ci, Matrix_fc);

    MatrixTransponation(Matrix_fa, Matrix_faT, n);
    MatrixTransponation(Matrix_fb, Matrix_fbT, n);
    MatrixTransponation(Matrix_fc, Matrix_fcT, n);

    MatrixMultiplication(Matrix_AI, Matrix_A, Matrix_Test);

    for (int j = 0; j <= n-1; j++)
    {
        AngleRot_2[j] = Math.Pow(AngleRot[j], 2);
        AngleRot_1[j] = AngleRot[j];
        Speed_AngleRot_1[j] = Speed_AngleRot[j];
    }

    MatrixMultiplicationVector(AngleRot_2, Matrix_faT, fa);
    MatrixMultiplicationVector(AngleRot_1, Matrix_fbT, fb);
    MatrixMultiplicationVector(AngleRot_0, Matrix_fcT, fc);

    for (int j = 0; j <= n-1; j++)
    {
        F[j] = fa[j] + fb[j] + fc[j];
        k1[j] = F[j] * dt;
        m1[j] = Speed_AngleRot_1[j]*dt;
    }

    for (int j = 0; j <= n-1; j++)
    {
        AngleRot_2[j] = Math.Pow(AngleRot[j] + m1[j] / 2.0, 2);
        AngleRot_1[j] = AngleRot[j] + m1[j] / 2.0;
        Speed_AngleRot_1[j] = Speed_AngleRot[j] + k1[j] / 2.0;
    }

    MatrixMultiplicationVector(AngleRot_2, Matrix_faT, fa);
    MatrixMultiplicationVector(AngleRot_1, Matrix_fbT, fb);
    MatrixMultiplicationVector(AngleRot_0, Matrix_fcT, fc);

    for (int j = 0; j <= n-1; j++)
    {
        F[j] = fa[j] + fb[j] + fc[j];
        k2[j] = F[j] * dt;
        m2[j] = Speed_AngleRot_1[j] * dt;
    }

    for (int j = 0; j <= n-1; j++)
    {
        AngleRot_2[j] = Math.Pow(AngleRot[j] + m2[j] / 2.0, 2);
        AngleRot_1[j] = AngleRot[j] + m2[j] / 2.0;
        Speed_AngleRot_1[j] = Speed_AngleRot[j] + k2[j] / 2.0;
    }
```

```
            MatrixMultiplicationVector(AngleRot_2, Matrix_faT, fa);
            MatrixMultiplicationVector(AngleRot_1, Matrix_fbT, fb);
            MatrixMultiplicationVector(AngleRot_0, Matrix_fcT, fc);

            for (int j = 0; j <= n-1; j++)
            {
                F[j] = fa[j] + fb[j] + fc[j];
                k3[j] = F[j] * dt;
                m3[j] = Speed_AngleRot_1[j] * dt;
            }

            for (int j = 0; j <= n-1; j++)
            {
                AngleRot_2[j] = Math.Pow(AngleRot[j] + m3[j], 2);
                AngleRot_1[j] = AngleRot[j] + m3[j];
                Speed_AngleRot_1[j] = Speed_AngleRot[j] + k3[j] ;
            }

            MatrixMultiplicationVector(AngleRot_2, Matrix_faT, fa);
            MatrixMultiplicationVector(AngleRot_1, Matrix_fbT, fb);
            MatrixMultiplicationVector(AngleRot_0, Matrix_fcT, fc);

            for (int j = 0; j <= n-1; j++)
            {
                F[j] = fa[j] + fb[j] + fc[j];
                k4[j] = F[j] * dt;
                m4[j] = Speed_AngleRot_1[j] * dt;
            }

            for (int i = 0; i <= n-1; i++ )
            {
              Speed_AngleRot[i] = Speed_AngleRot[i] + 1 / 6.0 * (m1[i] + 2 * m2[i] +
2 * m3[i] + m4[i]);
               AngleRot[i] = AngleRot[i] + 1 / 6.0 * (k1[i] + 2 * k2[i] + 2 * k3[i] +
k4[i]);
            }
            Console.WriteLine("углы поворота звеньев");
            Console.Write(AngleRot[0]); Console.Write("    ");
Console.Write(AngleRot[1]); Console.Write("    "); Console.Write(AngleRot[2]);
Console.Write("    "); Console.Write(AngleRot[3]); Console.Write("    ");
Console.Write(AngleRot[4]); Console.Write("    "); Console.WriteLine(AngleRot[5]);
            //Search_Node_Coordinates_Links_Of_Manipulator();
            //Drawing_Motion_Of_Manipulator();
        }

        private void Moment(double[] AngleRot)
        {
            for (int i = 0; i <= 5; i++ )
            {
                if (Math.Abs(AngleRot[i]) < e)
                {
                    a[i] = 0;
                    b[i] = 0;
                    c[i] = 0;
                }
                else
                {
                    a[i] = 3 / AngleRot[i];
                    b[i] = -4;
                    c[i] = AngleRot[i];
                }
               }
        }
        private void MatrixTransponation(double[,] A, double[,] B, int n)
        {
            for(int i=0; i<=n-1; i++)
```

```
            {
             for(int j=0; j<=n-1; j++)
             {
                 B[i,j]=A[j,i];
             }
            }
        }
       private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Axis();
            //Search_Node_Coordinates_Links_Of_Manipulator();
            // анимация
            Drawing_Motion_Of_Manipulator();
          }

        private void Axis()
        {
            d3d.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Green, 1.0f, 0);
            d3d.BeginScene();
            SetupProekcii();
            // рисуем оси координат со сдвигом изображения вглубь экрана
            d3d.Material = axisXMaterial;
            d3d.Transform.World = Matrix.RotationX((float)(0 * Math.PI / 180.0)) *
Matrix.RotationY((float)(90 * Math.PI / 180.0)) * Matrix.RotationZ((float)(0 *
Math.PI / 180.0)) * Matrix.Translation(xs, ys, zs);
            axisX.DrawSubset(0);
            d3d.Material = axisYMaterial;
            d3d.Transform.World = Matrix.RotationX((float)(90 * Math.PI / 180.0)) *
Matrix.RotationY((float)(0 * Math.PI / 180.0)) * Matrix.RotationZ((float)(0 *
Math.PI / 180.0)) * Matrix.Translation(xs, ys, zs);
            axisY.DrawSubset(0);
            d3d.Material = axisZMaterial;
            d3d.Transform.World = Matrix.RotationX((float)(0 * Math.PI / 180.0)) *
Matrix.RotationY((float)(0 * Math.PI / 180.0)) * Matrix.RotationZ((float)(0 *
Math.PI / 180.0)) * Matrix.Translation(xs, ys, zs);
            axisZ.DrawSubset(0);
            d3d.EndScene();
            //Показываем содержимое дублирующего буфера
            d3d.Present();
        }

       // процедура поиска координат узлов манипулятора
        private void Search_Node_Coordinates_Links_Of_Manipulator(double[] AngleRot,
double AngleRot_h)
        {
          // матрица преобразования координат системы O1X1Y1Z1 в координаты системы
OXYZ после вращения с системе O1X1Y1Z1
           double[,] MatrixTransformAfterRot_O1O0 = new double[3, 3];
           // матрица преобразования координат системы O2X2Y2Z2 в координаты системы
O1X1Y1Z1 после вращения с системе O2X2Y2Z2
           double[,] MatrixTransformAfterRot_O2O1 = new double[3, 3];
           // матрица преобразования координат системы O3X3Y3Z3 в координаты системы
O2X2Y2Z2 после вращения с системе O3X3Y3Z3
           double[,] MatrixTransformAfterRot_O3O2 = new double[3, 3];
           // матрица преобразования координат системы O4X4Y4Z4 в координаты системы
O3X3Y3Z3 после вращения с системе O4X4Y4Z4
           double[,] MatrixTransformAfterRot_O4O3 = new double[3, 3];
           // матрица преобразования координат системы O5X5Y5Z5 в координаты системы
O4X4Y4Z4 после вращения с системе O5X5Y5Z5
           double[,] MatrixTransformAfterRot_O5O4 = new double[3, 3];
           // матрица преобразования координат системы O6X6Y6Z6 в координаты системы
O5X5Y5Z5 после вращения с системе O6X6Y6Z6
           double[,] MatrixTransformAfterRot_O6O5 = new double[3, 3];
           // матрица преобразования координат системы O7X7Y7Z7 в координаты системы
O6X6Y6Z6 после вращения с системе O7X7Y7Z7
           double[,] MatrixTransformAfterRot_O7O6 = {{1,0,0},{0,1,0},{0,0,1}};
```

```
            // матрица преобразования координат системы O81X81Y81Z81 в координаты
системы O7X7Y7Z7 после вращения с системе O81X81Y81Z81
            double[,] MatrixTransformAfterRot_O81O7 = new double[3, 3];
            // матрица преобразования координат системы O82X82Y82Z82 в координаты
системы O7X7Y7Z7 после вращения с системе O82X82Y82Z82
            double[,] MatrixTransformAfterRot_O82O7 = new double[3, 3];
            // матрица преобразования координат системы O83X83Y83Z83 в координаты
системы O7X7Y7Z7 после вращения с системе O83X83Y83Z83
            double[,] MatrixTransformAfterRot_O83O7 = new double[3, 3];


            ValueMatrixRotationZ(MatrixRotation_1, AngleRot[0]);
            ValueMatrixRotationZ(MatrixRotation_2, AngleRot[1]);
            ValueMatrixRotationZ(MatrixRotation_3, AngleRot[2]);
            ValueMatrixRotationZ(MatrixRotation_4, AngleRot[3]);
            ValueMatrixRotationZ(MatrixRotation_5, AngleRot[4]);
            ValueMatrixRotationZ(MatrixRotation_6, AngleRot[5]);
            ValueMatrixRotationZ(MatrixRotation_81, AngleRot_h);
            ValueMatrixRotationZ(MatrixRotation_82, AngleRot_h);
            ValueMatrixRotationZ(MatrixRotation_83, AngleRot_h);

            // преобразование матрицы перехода из системы OXYZ в систему O1X1Y1Z1,
полученную в следствие поворота ортов вокруг оси O1Z1
            MatrixMultiplication(MatrixRotation_1, MatrixTransformation_O1O0,
MatrixTransformAfterRot_O1O0);
            // переобозначение
            MatrixTransformation_O1O0 = MatrixTransformAfterRot_O1O0;
            // координаты вектора второго звена после поворота в неподвижной системе
OXYZ
            MatrixMultiplicationVector(Segment_O1O2, MatrixTransformation_O1O0,
Segment_O0O2);

            // преобразование матрицы перехода из системы O1X1Y1Z1 в систему O2X2Y2Z2,
отражающее поворот ортов вокруг оси O2Z2
            MatrixMultiplication(MatrixRotation_2, MatrixTransformation_O2O1,
MatrixTransformAfterRot_O2O1);
            // переобозначение
            MatrixTransformation_O2O1 = MatrixTransformAfterRot_O2O1;
            // матрица перехода из системы OXYZ в систему O2X2Y2Z2
            MatrixMultiplication(MatrixTransformation_O2O1, MatrixTransformation_O1O0,
MatrixTransformation_O2O0);
            // координаты вектора третьего звена после поворота в неподвижной системе
OXYZ
            MatrixMultiplicationVector(Segment_O2O3, MatrixTransformation_O2O0,
Segment_O0O3);

            // преобразование матрицы перехода из системы O2X2Y2Z2 в систему O3X3Y3Z3,
отражающее поворот ортов вокруг оси O3Z3
            MatrixMultiplication(MatrixRotation_3, MatrixTransformation_O3O2,
MatrixTransformAfterRot_O3O2);
            // переобозначение
            MatrixTransformation_O3O2 = MatrixTransformAfterRot_O3O2;
            // матрица перехода из системы OXYZ в систему O3X3Y3Z3
            MatrixMultiplication(MatrixTransformation_O3O2, MatrixTransformation_O2O0,
MatrixTransformation_O3O0);
            // координаты вектора четвёртого звена в неподвижной системе OXYZ
            MatrixMultiplicationVector(Segment_O3O4, MatrixTransformation_O3O0,
Segment_O0O4);

            // преобразование матрицы перехода из системы O3X3Y3Z3 в систему O4X4Y4Z4,
отражающее поворот ортов вокруг оси O4Z4
            MatrixMultiplication(MatrixRotation_4, MatrixTransformation_O4O3,
MatrixTransformAfterRot_O4O3);
            // переобозначение
            MatrixTransformation_O4O3 = MatrixTransformAfterRot_O4O3;
            // матрица перехода из системы OXYZ в систему O4X4Y4Z4
```

```
            MatrixMultiplication(MatrixTransformation_O4O3, MatrixTransformation_O3O0,
MatrixTransformation_O4O0);
            // координаты вектора пятого звена в неподвижной системе OXYZ
            MatrixMultiplicationVector(Segment_O4O5, MatrixTransformation_O4O0,
Segment_O0O5);

            // преобразование матрицы перехода из системы O4X4Y4Z4 в систему O5X5Y5Z5,
отражающее поворот ортов вокруг оси O5Z5
            MatrixMultiplication(MatrixRotation_5, MatrixTransformation_O5O4,
MatrixTransformAfterRot_O5O4);
            // переобозначение
            MatrixTransformation_O5O4 = MatrixTransformAfterRot_O5O4;
            // матрица перехода из системы OXYZ в систему O5X5Y5Z5
            MatrixMultiplication(MatrixTransformation_O5O4, MatrixTransformation_O4O0,
MatrixTransformation_O5O0);
            // координаты вектора пятого звена в неподвижной системе OXYZ
            MatrixMultiplicationVector(Segment_O5O6, MatrixTransformation_O5O0,
Segment_O0O6);

            // преобразование матрицы перехода из системы O5X5Y5Z5 в систему O6X6Y6Z6,
отражающее поворот ортов вокруг оси O6Z6
            MatrixMultiplication(MatrixRotation_6, MatrixTransformation_O6O5,
MatrixTransformAfterRot_O6O5);
            // переобозначение
            MatrixTransformation_O6O5 = MatrixTransformAfterRot_O6O5;
            // матрица перехода из системы OXYZ в систему O6X6Y6Z6
            MatrixMultiplication(MatrixTransformation_O6O5, MatrixTransformation_O5O0,
MatrixTransformation_O6O0);
            // определение координат вектора шестого звена в неподвижной системе OXYZ
            MatrixMultiplicationVector(Segment_O6O7, MatrixTransformation_O6O0,
Segment_O0O7);

            // система O7X7Y7Z7 не поворачивается относительно осей системы O6X6Y6Z6,
поэтому матрица  MatrixTransformation_O7O6 не изменяется
            MatrixMultiplication(MatrixTransformation_O7O6, MatrixTransformation_O6O0,
MatrixTransformation_O7O0);
            // определение координат вектора первой фаланги первого пальца схвата в
неподвижной системе OXYZ
            MatrixMultiplicationVector(Segment_O7O81, MatrixTransformation_O7O0,
Segment_O0O81);
            // определение координат вектора первой фаланги второго пальца схвата в
неподвижной системе OXYZ
            MatrixMultiplicationVector(Segment_O7O82, MatrixTransformation_O7O0,
Segment_O0O82);
            // определение координат вектора первой фаланги третьего пальца схвата в
неподвижной системе OXYZ
            MatrixMultiplicationVector(Segment_O7O83, MatrixTransformation_O7O0,
Segment_O0O83);


            MatrixMultiplication(MatrixRotation_81, MatrixTransformation_O81O7,
MatrixTransformAfterRot_O81O7);
            // переобозначение
            MatrixTransformation_O81O7 = MatrixTransformAfterRot_O81O7;
            // матрица перехода из системы OXYZ в систему O81X81Y81Z81
            MatrixMultiplication(MatrixTransformation_O81O7,
MatrixTransformation_O7O0, MatrixTransformation_O81O0);
            // координаты вектора второй фаланги первого пальца схвата в неподвижной
системе OXYZ
            MatrixMultiplicationVector(Segment_O81O91, MatrixTransformation_O81O0,
Segment_O0O91);

            MatrixMultiplication(MatrixRotation_82, MatrixTransformation_O82O7,
MatrixTransformAfterRot_O82O7);
            // переобозначение
            MatrixTransformation_O82O7 = MatrixTransformAfterRot_O82O7;
            // матрица перехода из системы OXYZ в систему O82X82Y82Z82
```

```
            MatrixMultiplication(MatrixTransformation_O82O7,
MatrixTransformation_O7O0, MatrixTransformation_O82O0);
            // координаты вектора второй фаланги второго пальца схвата в неподвижной
системе OXYZ
            MatrixMultiplicationVector(Segment_O82O92, MatrixTransformation_O82O0,
Segment_O0O92);

            MatrixMultiplication(MatrixRotation_83, MatrixTransformation_O83O7,
MatrixTransformAfterRot_O83O7);
            // переобозначение
            MatrixTransformation_O83O7 = MatrixTransformAfterRot_O83O7;
            // матрица перехода из системы OXYZ в систему O83X83Y83Z83
            MatrixMultiplication(MatrixTransformation_O83O7,
MatrixTransformation_O7O0, MatrixTransformation_O83O0);
            // координаты вектора второй фаланги третьего пальца схвата в неподвижной
системе OXYZ
            MatrixMultiplicationVector(Segment_O83O93, MatrixTransformation_O83O0,
Segment_O0O93);

            //определение радиус-векторов узлов манипулятора в неподвижной системе
координат
            RadiusVector_O0O0 = Segment_O0O0;
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O1[i] = RadiusVector_O0O0[i]
+ Segment_O0O1[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O2[i] = RadiusVector_O0O1[i]
+ Segment_O0O2[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O3[i] = RadiusVector_O0O2[i]
+ Segment_O0O3[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O4[i] = RadiusVector_O0O3[i]
+ Segment_O0O4[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O5[i] = RadiusVector_O0O4[i]
+ Segment_O0O5[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O6[i] = RadiusVector_O0O5[i]
+ Segment_O0O6[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O7[i] = RadiusVector_O0O6[i]
+ Segment_O0O7[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O81[i] =
RadiusVector_O0O7[i] + Segment_O0O81[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O82[i] =
RadiusVector_O0O7[i] + Segment_O0O82[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O83[i] =
RadiusVector_O0O7[i] + Segment_O0O83[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O91[i] =
RadiusVector_O0O81[i] + Segment_O0O91[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O92[i] =
RadiusVector_O0O82[i] + Segment_O0O92[i]; }
            for (int i = 0; i <= 2; i++) { RadiusVector_O0O93[i] =
RadiusVector_O0O83[i] + Segment_O0O93[i]; }
        }

        // процедура поиска углов поворота звеньев манипулятора
        private void Search_Link_Rotation_Angles_Of_Manipulator(double[]
RadiusVector_O0Ai)
        {
            double sinusF1, sinusF2, sinusF3, sinusF4, sinusF5, sinusF6;
            // радиус-вектор OAi в системе OXYZ
            double[] RadiusVector_O0Ai_0 = new double[3];
            // радиус-вектор O1Ai в системе OXYZ
            double[] RadiusVector_O1Ai_0 = new double[3];
            // радиус-вектор O1Ai в системе O1X1Y1Z1
            double[] RadiusVector_O1Ai_1 = new double[3];
            // радиус-вектор O2Ai в системе O1X1Y1Z1
            double[] RadiusVector_O2Ai_1 = new double[3];
            // радиус-вектор O2Ai в системе O2X2Y2Z2
            double[] RadiusVector_O2Ai_2 = new double[3];
            // радиус-вектор O3Ai в системе O2X2Y2Z2
            double[] RadiusVector_O3Ai_2 = new double[3];
```

```
            // радиус-вектор O3Ai в системе O3X3Y3Z3
            double[] RadiusVector_O3Ai_3 = new double[3];
            // радиус-вектор O4Ai в системе O3X3Y3Z3
            double[] RadiusVector_O4Ai_3 = new double[3];
            // радиус-вектор O4Ai в системе O4X4Y4Z4
            double[] RadiusVector_O4Ai_4 = new double[3];
            // радиус-вектор O5Ai в системе O4X4Y4Z4
            double[] RadiusVector_O5Ai_4 = new double[3];
            // радиус-вектор O5Ai в системе O5X5Y5Z5
            double[] RadiusVector_O5Ai_5 = new double[3];
            // радиус-вектор O5Ai в системе O5X5Y5Z5
            double[] RadiusVector_O6Ai_5 = new double[3];
            // радиус-вектор O5Ai в системе O6X6Y6Z6
            double[] RadiusVector_O6Ai_6 = new double[3];


            // радиус-вектор O1O7 в системе OXYZ
            double[] RadiusVector_O1O7_0 = new double[3];
            // радиус-вектор O1O7 в системе O1X1Y1Z1
            double[] RadiusVector_O1O7_1 = new double[3];
            // радиус-вектор O2O7 в системе O1X1Y1Z1
            double[] RadiusVector_O2O7_1 = new double[3];
            // радиус-вектор O2O7 в системе O2X2Y2Z2
            double[] RadiusVector_O2O7_2 = new double[3];
            // радиус-вектор O3O7 в системе O2X2Y2Z2
            double[] RadiusVector_O3O7_2 = new double[3];
            // радиус-вектор O3O7 в системе O3X3Y3Z3
            double[] RadiusVector_O3O7_3 = new double[3];
            // радиус-вектор O4O7 в системе O3X3Y3Z3
            double[] RadiusVector_O4O7_3 = new double[3];
            // радиус-вектор O4O7 в системе O4X4Y4Z4
            double[] RadiusVector_O4O7_4 = new double[3];
            // радиус-вектор O5O7 в системе O4X4Y4Z4
            double[] RadiusVector_O5O7_4 = new double[3];
            // радиус-вектор O5O7 в системе O5X5Y5Z5
            double[] RadiusVector_O5O7_5 = new double[3];
            // радиус-вектор O6O7 в системе O5X5Y5Z5
            double[] RadiusVector_O6O7_5 = new double[3];
            // радиус-вектор O6O7 в системе O6X6Y6Z6
            double[] RadiusVector_O6O7_6 = new double[3];

            // радиус-вектор O1Ai в системе O1X1Y1Z1 после поворота  на угол
AngleRotation_1
            double[] RadiusVector_O1AiR_1 = new double[3];
            // радиус-вектор O2Ai в системе O2X2Y2Z2 после поворота  на угол
AngleRotation_1
            double[] RadiusVector_O2AiR_2 = new double[3];
            // радиус-вектор O3Ai в системе O3X3Y3Z3 после поворота  на угол
AngleRotation_3
            double[] RadiusVector_O3AiR_3 = new double[3];
            // радиус-вектор O4Ai в системе O4X4Y4Z4 после поворота  на угол
AngleRotation_4
            double[] RadiusVector_O4AiR_4 = new double[3];
            // радиус-вектор O5Ai в системе O5X5Y5Z5 после поворота  на угол
AngleRotation_5
            double[] RadiusVector_O5AiR_5 = new double[3];
            // радиус-вектор O6Ai в системе O6X6Y6Z6 после поворота  на угол
AngleRotation_6
            double[] RadiusVector_O6AiR_6 = new double[3];
            double[] C = new double[3];

            double[,] InvertionMatrixTransformation_O1O0 = new double[3, 3];
            double[,] InvertionMatrixTransformation_O2O1 = new double[3, 3];
            double[,] InvertionMatrixTransformation_O3O2 = new double[3, 3];
            double[,] InvertionMatrixTransformation_O4O3 = new double[3, 3];
            double[,] InvertionMatrixTransformation_O5O4 = new double[3, 3];
            double[,] InvertionMatrixTransformation_O6O5 = new double[3, 3];
```

```
            double[,] InvertionMatrixTransformation_O7O6 = new double[3, 3];
            double[,] InvertionMatrixRotation_1 = new double[3, 3];
            double[,] InvertionMatrixRotation_2 = new double[3, 3];
            double[,] InvertionMatrixRotation_3 = new double[3, 3];
            double[,] InvertionMatrixRotation_4 = new double[3, 3];
            double[,] InvertionMatrixRotation_5 = new double[3, 3];
            double[,] InvertionMatrixRotation_6 = new double[3, 3];

            // угол вращения второго звена
            for (int i = 0; i <= 2; i++)
            {
                // вектор O1O6 в системе OXYZ
                RadiusVector_O1O7_0[i] = RadiusVector_O0O7[i] - Segment_O0O1[i];
                // вектор O1Ai в системе OXYZ
                RadiusVector_O1Ai_0[i] = RadiusVector_O0Ai[i] - Segment_O0O1[i];
            }
            // матрица преобразования из OXYZ в O1X1Y1Z1 (обратная к матрице
преобразования от O1X1Y1Z1 к OXYZ)
            MatrixInvertion_3(MatrixTransformation_O1O0,
InvertionMatrixTransformation_O1O0);
            // преобразование координат вектора O1O6 из OXYZ в O1X1Y1Z1
            MatrixMultiplicationVector(RadiusVector_O1O7_0,
InvertionMatrixTransformation_O1O0, RadiusVector_O1O7_1);
            // преобразование координат вектора O1Ai из OXYZ в O1X1Y1Z1
            MatrixMultiplicationVector(RadiusVector_O1Ai_0,
InvertionMatrixTransformation_O1O0, RadiusVector_O1Ai_1);
            double a1, b1, c1;
            // векторное произведение проекций O1O6 и O1Ai на O1X1Y1
            a1 = RadiusVector_O1O7_1[0] * RadiusVector_O1Ai_1[1] -
RadiusVector_O1O7_1[1] * RadiusVector_O1Ai_1[0];

////////////////////////////////////////////////////////////////////////////////
////////////////////////////
            // длина проекции вектора O1O6 на O1X1Y1
            b1 = Math.Sqrt(RadiusVector_O1O7_1[0] * RadiusVector_O1O7_1[0] +
RadiusVector_O1O7_1[1] * RadiusVector_O1O7_1[1]);//+ RadiusVector_1_5_1[2] *
RadiusVector_1_5_1[2]);
            // длина проекции вектора O1Ai на O1X1Y1
            c1 = Math.Sqrt(RadiusVector_O1Ai_1[0] * RadiusVector_O1Ai_1[0] +
RadiusVector_O1Ai_1[1] * RadiusVector_O1Ai_1[1]);//+ RadiusVector_1_A_1[2] *
RadiusVector_1_A_1[2]);
            // синус первого угла AngleRotation_1
            sinusF1 = a1 / (b1 * c1);
            // первый угол вращения в радианах
            AngleRotation[0] = Math.Asin(sinusF1);
            // первый угол вращения в градусах
            double fi1 = AngleRotation[0] * 180 / Math.PI;
            //Console.WriteLine("угол поворота второго звена");
            //Console.Write("fi1"); Console.Write("    "); Console.WriteLine(fi1);
            ValueMatrixRotationZ(MatrixRotation_1, AngleRotation[0]);
            // матрица, обратная к матрице поворота первого звена
            MatrixInvertion_3(MatrixRotation_1, InvertionMatrixRotation_1);
            // вектор O1Ai1 равен произведению вектора O1Ai на матрицу, обратную
матрице поворота первого звена на угол fi1
            MatrixMultiplicationVector(RadiusVector_O1Ai_1,
InvertionMatrixRotation_1, RadiusVector_O1AiR_1);
            double S1 = RadiusVector_O1O7_1[0] / RadiusVector_O1O7_1[1] -
RadiusVector_O1AiR_1[0] / RadiusVector_O1AiR_1[1];

            // угол вращения третьего звена
            for (int i = 0; i <= 2; i++)
            {
                RadiusVector_O2O7_1[i] = RadiusVector_O1O7_1[i] - Segment_O1O2[i];
                RadiusVector_O2Ai_1[i] = RadiusVector_O1AiR_1[i] - Segment_O1O2[i];
            }
            MatrixInvertion_3(MatrixTransformation_O2O1,
InvertionMatrixTransformation_O2O1);
```

```
            MatrixMultiplicationVector(RadiusVector_O2O7_1,
InvertionMatrixTransformation_O2O1, RadiusVector_O2O7_2);
            MatrixMultiplicationVector(RadiusVector_O2Ai_1,
InvertionMatrixTransformation_O2O1, RadiusVector_O2Ai_2);
            double a2, b2, c2;
            a2 = RadiusVector_O2O7_2[0] * RadiusVector_O2Ai_2[1] -
RadiusVector_O2O7_2[1] * RadiusVector_O2Ai_2[0];

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
            b2 = Math.Sqrt(RadiusVector_O2O7_2[0] * RadiusVector_O2O7_2[0] +
RadiusVector_O2O7_2[1] * RadiusVector_O2O7_2[1]);//+ RadiusVector_2_5_2[2] *
RadiusVector_2_5_2[2]);
            c2 = Math.Sqrt(RadiusVector_O2Ai_2[0] * RadiusVector_O2Ai_2[0] +
RadiusVector_O2Ai_2[1] * RadiusVector_O2Ai_2[1]);//+ RadiusVector_2_A_2[2] *
RadiusVector_2_A_2[2]);
            sinusF2 = a2 / (b2 * c2);
            AngleRotation[1] = Math.Asin(sinusF2);
            double fi2 = AngleRotation[1] * 180 / Math.PI;
            //Console.WriteLine("угол поворота третьего звена");
            //Console.Write("fi2"); Console.Write("    "); Console.WriteLine(fi2);
            ValueMatrixRotationZ(MatrixRotation_2, AngleRotation[1]);
            MatrixInvertion_3(MatrixRotation_2, InvertionMatrixRotation_2);
            MatrixMultiplicationVector(RadiusVector_O2Ai_2,
InvertionMatrixRotation_2, RadiusVector_O2AiR_2);
            double S2 = RadiusVector_O2O7_2[0] / RadiusVector_O2O7_2[1] -
RadiusVector_O2AiR_2[0] / RadiusVector_O2AiR_2[1];

            // угол вращения четвёртого звена
            for (int i = 0; i <= 2; i++)
            {
                RadiusVector_O3O7_2[i] = RadiusVector_O2O7_2[i] - Segment_O2O3[i];
                RadiusVector_O3Ai_2[i] = RadiusVector_O2AiR_2[i] - Segment_O2O3[i];
            }
            MatrixInvertion_3(MatrixTransformation_O3O2,
InvertionMatrixTransformation_O3O2);
            MatrixMultiplicationVector(RadiusVector_O3O7_2,
InvertionMatrixTransformation_O3O2, RadiusVector_O3O7_3);
            MatrixMultiplicationVector(RadiusVector_O3Ai_2,
InvertionMatrixTransformation_O3O2, RadiusVector_O3Ai_3);

            double a3, b3, c3;
            a3 = RadiusVector_O3O7_3[2] * RadiusVector_O3Ai_3[0] -
RadiusVector_O3O7_3[0] * RadiusVector_O3Ai_3[2];

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
            b3 = Math.Sqrt(RadiusVector_O3O7_3[0] * RadiusVector_O3O7_3[0] +
RadiusVector_O3O7_3[2] * RadiusVector_O3O7_3[2]);// + RadiusVector_3_5_3[1] *
RadiusVector_3_5_3[1]);
            c3 = Math.Sqrt(RadiusVector_O3Ai_3[0] * RadiusVector_O3Ai_3[0] +
RadiusVector_O3Ai_3[2] * RadiusVector_O3Ai_3[2]);// + RadiusVector_3_A_3[1] *
RadiusVector_3_A_3[1]);
            sinusF3 = a3 / (b3 * c3);
            AngleRotation[2] = Math.Asin(sinusF3);
            double fi3 = AngleRotation[2] * 180.0 / Math.PI;
            //Console.WriteLine("угол поворота четвёртого звена");
            //Console.Write("fi3"); Console.Write("    "); Console.WriteLine(fi3);
            ValueMatrixRotationY(MatrixRotation_3, AngleRotation[2]);
            MatrixInvertion_3(MatrixRotation_3, InvertionMatrixRotation_3);
            MatrixMultiplicationVector(RadiusVector_O3Ai_3,
InvertionMatrixRotation_3, RadiusVector_O3AiR_3);
            double S3 = RadiusVector_O3O7_3[0] / RadiusVector_O3O7_3[2] -
RadiusVector_O3AiR_3[0] / RadiusVector_O3AiR_3[2];

            // угол вращения пятого звена
            for (int i = 0; i <= 2; i++)
```

```
            {
                RadiusVector_O4O7_3[i] = RadiusVector_O3O7_3[i] - Segment_O3O4[i];
                RadiusVector_O4Ai_3[i] = RadiusVector_O3AiR_3[i] - Segment_O3O4[i];
            }
            MatrixInvertion_3(MatrixTransformation_O4O3,
InvertionMatrixTransformation_O4O3);
            MatrixMultiplicationVector(RadiusVector_O4O7_3,
InvertionMatrixTransformation_O4O3, RadiusVector_O4O7_4);
            MatrixMultiplicationVector(RadiusVector_O4Ai_3,
InvertionMatrixTransformation_O4O3, RadiusVector_O4Ai_4);
            double a4, b4, c4;
            a4 = RadiusVector_O4O7_4[1] * RadiusVector_O4Ai_4[2] -
RadiusVector_O4O7_4[2] * RadiusVector_O4Ai_4[1];

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
            b4 = Math.Sqrt(RadiusVector_O4O7_4[2] * RadiusVector_O4O7_4[2] +
RadiusVector_O4O7_4[1] * RadiusVector_O4O7_4[1]);// + RadiusVector_4_5_4[0] *
RadiusVector_4_5_4[0]);
            c4 = Math.Sqrt(RadiusVector_O4Ai_4[2] * RadiusVector_O4Ai_4[2] +
RadiusVector_O4Ai_4[1] * RadiusVector_O4Ai_4[1]);// + RadiusVector_4_A_4[2] *
RadiusVector_4_A_4[2]);
            sinusF4 = a4 / (b4 * c4);
            AngleRotation[3] = Math.Asin(sinusF4);
            double fi4 = AngleRotation[3] * 180.0 / Math.PI;
            //Console.WriteLine("угол поворота пятого звена");
            //Console.Write("fi4"); Console.Write("    "); Console.WriteLine(fi4);
            ValueMatrixRotationX(MatrixRotation_4, AngleRotation[3]);
            MatrixInvertion_3(MatrixRotation_4, InvertionMatrixRotation_4);
            MatrixMultiplicationVector(RadiusVector_O4Ai_4,
InvertionMatrixRotation_4, RadiusVector_O4AiR_4);

            double S4 = RadiusVector_O4O7_4[1] / RadiusVector_O4O7_4[2] -
RadiusVector_O4AiR_4[1] / RadiusVector_O4AiR_4[2];

            // угол вращения шестого звена
            for (int i = 0; i <= 2; i++)
            {
                RadiusVector_O5O7_4[i] = RadiusVector_O4O7_4[i] - Segment_O4O5[i];
                RadiusVector_O5Ai_4[i] = RadiusVector_O4AiR_4[i] - Segment_O4O5[i];
            }
            MatrixInvertion_3(MatrixTransformation_O5O4,
InvertionMatrixTransformation_O5O4);
            MatrixMultiplicationVector(RadiusVector_O5O7_4,
InvertionMatrixTransformation_O5O4, RadiusVector_O5O7_5);
            MatrixMultiplicationVector(RadiusVector_O5Ai_4,
InvertionMatrixTransformation_O5O4, RadiusVector_O5Ai_5);
            double a5, b5, c5;
            a5 = RadiusVector_O5O7_5[1] * RadiusVector_O5Ai_5[2] -
RadiusVector_O5O7_5[2] * RadiusVector_O5Ai_5[1];

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
            b5 = Math.Sqrt(RadiusVector_O5O7_5[2] * RadiusVector_O5O7_5[2] +
RadiusVector_O5O7_5[1] * RadiusVector_O5O7_5[1]);// + RadiusVector_4_5_4[0] *
RadiusVector_4_5_4[0]);
            c5 = Math.Sqrt(RadiusVector_O5Ai_5[2] * RadiusVector_O5Ai_5[2] +
RadiusVector_O5Ai_5[1] * RadiusVector_O5Ai_5[1]);// + RadiusVector_4_A_4[2] *
RadiusVector_4_A_4[2]);
            sinusF5 = a5 / (b5 * c5);
            AngleRotation[4] = Math.Asin(sinusF5);
            double fi5 = AngleRotation[4] * 180.0 / Math.PI;
            //Console.WriteLine("угол поворота шестого звена");
            //Console.Write("fi5"); Console.Write("    "); Console.WriteLine(fi5);
            ValueMatrixRotationX(MatrixRotation_5, AngleRotation[4]);
            MatrixInvertion_3(MatrixRotation_5, InvertionMatrixRotation_5);
```

```
            MatrixMultiplicationVector(RadiusVector_O5Ai_5,
InvertionMatrixRotation_5, RadiusVector_O5AiR_5);

            double S5 = RadiusVector_O5O7_5[1] / RadiusVector_O5O7_5[2] -
RadiusVector_O5AiR_5[1] / RadiusVector_O5AiR_5[2];

            // угол вращения седьмого звена
            for (int i = 0; i <= 2; i++)
            {
                RadiusVector_O6O7_5[i] = RadiusVector_O5O7_5[i] - Segment_O5O6[i];
                RadiusVector_O6Ai_5[i] = RadiusVector_O5AiR_5[i] - Segment_O5O6[i];
            }
            MatrixInvertion_3(MatrixTransformation_O6O5,
InvertionMatrixTransformation_O6O5);
            MatrixMultiplicationVector(RadiusVector_O6O7_5,
InvertionMatrixTransformation_O6O5, RadiusVector_O6O7_6);
            MatrixMultiplicationVector(RadiusVector_O6Ai_5,
InvertionMatrixTransformation_O6O5, RadiusVector_O6Ai_6);
            double a6, b6, c6;
            a6 = RadiusVector_O6O7_6[1] * RadiusVector_O6Ai_6[2] -
RadiusVector_O6O7_6[2] * RadiusVector_O6Ai_6[1];

////////////////////////////////////////////////////////////////////////////////
//////////////////////////
            b6 = Math.Sqrt(RadiusVector_O6O7_6[2] * RadiusVector_O6O7_6[2] +
RadiusVector_O6O7_6[1] * RadiusVector_O6O7_6[1]);// + RadiusVector_4_5_4[0] *
RadiusVector_4_5_4[0]);
            c6 = Math.Sqrt(RadiusVector_O6Ai_6[2] * RadiusVector_O6Ai_6[2] +
RadiusVector_O6Ai_6[1] * RadiusVector_O6Ai_6[1]);// + RadiusVector_4_A_4[2] *
RadiusVector_4_A_4[2]);
            sinusF6 = a6 / (b6 * c6);
            AngleRotation[5] = Math.Asin(sinusF6);
            double fi6 = AngleRotation[5] * 180.0 / Math.PI;
            //Console.WriteLine("угол поворота шестого звена");
            //Console.Write("fi6"); Console.Write("    "); Console.WriteLine(fi6);
            ValueMatrixRotationX(MatrixRotation_6, AngleRotation[5]);
            MatrixInvertion_3(MatrixRotation_6, InvertionMatrixRotation_6);
            MatrixMultiplicationVector(RadiusVector_O6Ai_6,
InvertionMatrixRotation_6, RadiusVector_O6AiR_6);
            double S6 = RadiusVector_O6O7_6[1] / RadiusVector_O6O7_6[2] -
RadiusVector_O6AiR_6[1] / RadiusVector_O6AiR_6[2];
        }

        // аппроксимация функции параболой по 3 точкам

        private void Drawing_Motion_Of_Manipulator()
        {
            d3d.Clear(ClearFlags.Target | ClearFlags.ZBuffer, Color.Green, 1.0f, 0);
            d3d.BeginScene();
            // рисование вектора
            AngleRotationVector(RadiusVector_O0O1, RadiusVector_O0O0);
            xm[0] = xm0; ym[0] = ym0; zm[0] = zm0; d_1 = d0;
            RotX[0] = RotX0; RotY[0] = RotY0; RotZ[0] = RotZ0;
            Link1 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_1, 50, 10);
            Link1Material = new Material();
            Link1Material.Diffuse = Color.White;
            Link1Material.Specular = Color.White;
            d3d.Material = Link1Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link1.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O2, RadiusVector_O0O1);
            xm[1] = xm0; ym[1] = ym0; zm[1] = zm0; d_2 = d0;
            RotX[1] = RotX0; RotY[1] = RotY0; RotZ[1] = RotZ0;
            Link2 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_2, 50, 10);
```

```
            Link2Material = new Material();
            Link2Material.Diffuse = Color.White;
            Link2Material.Specular = Color.White;
            d3d.Material = Link2Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link2.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O3, RadiusVector_O0O2);
            xm[2] = xm0; ym[2] = ym0; zm[2] = zm0; d_3 = d0;
            RotX[2] = RotX0; RotY[2] = RotY0; RotZ[2] = RotZ0;
            Link3 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_3, 50, 10);
            Link3Material = new Material();
            Link3Material.Diffuse = Color.White;
            Link3Material.Specular = Color.White;
            d3d.Material = Link3Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link3.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O4, RadiusVector_O0O3);
            xm[3] = xm0; ym[3] = ym0; zm[3] = zm0; d_4 = d0;
            RotX[3] = RotX0; RotY[3] = RotY0; RotZ[3] = RotZ0;
            Link4 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_4, 50, 10);
            Link4Material = new Material();
            Link4Material.Diffuse = Color.White;
            Link4Material.Specular = Color.White;
            d3d.Material = Link4Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link4.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O5, RadiusVector_O0O4);
            xm[4] = xm0; ym[4] = ym0; zm[4] = zm0; d_5 = d0;
            RotX[4] = RotX0; RotY[4] = RotY0; RotZ[4] = RotZ0;
            Link5 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_5, 50, 10);
            Link5Material = new Material();
            Link5Material.Diffuse = Color.White;
            Link5Material.Specular = Color.White;
            d3d.Material = Link5Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link5.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O6, RadiusVector_O0O5);
            xm[5] = xm0; ym[5] = ym0; zm[5] = zm0; d_6 = d0;
            RotX[5] = RotX0; RotY[5] = RotY0; RotZ[5] = RotZ0;
            Link6 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_6, 50, 10);
            Link6Material = new Material();
            Link6Material.Diffuse = Color.White;
            Link6Material.Specular = Color.White;
            d3d.Material = Link6Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link6.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O7, RadiusVector_O0O6);
            xm[6] = xm0; ym[6] = ym0; zm[6] = zm0; d_7 = d0;
            RotX[6] = RotX0; RotY[6] = RotY0; RotZ[6] = RotZ0;
            Link7 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_7, 50, 10);
            Link7Material = new Material();
            Link7Material.Diffuse = Color.White;
```

```
            Link7Material.Specular = Color.White;
            d3d.Material = Link7Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link7.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O81, RadiusVector_O0O7);
            xm[7] = xm0; ym[7] = ym0; zm[7] = zm0; d_8 = d0;
            RotX[7] = RotX0; RotY[7] = RotY0; RotZ[7] = RotZ0;
            Link81 = Mesh.Cylinder(d3d, 0.005f, 0.005f, (float)d_8, 50, 10);
            Link81Material = new Material();
            Link81Material.Diffuse = Color.Red;
            Link81Material.Specular = Color.Red;
            d3d.Material = Link81Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link81.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O82, RadiusVector_O0O7);
            xm[8] = xm0; ym[8] = ym0; zm[8] = zm0; d_8 = d0;
            RotX[8] = RotX0; RotY[8] = RotY0; RotZ[8] = RotZ0;
            Link82 = Mesh.Cylinder(d3d, 0.005f, 0.005f, (float)d_8, 50, 10);
            Link82Material = new Material();
            Link82Material.Diffuse = Color.Blue;
            Link82Material.Specular = Color.Blue;
            d3d.Material = Link82Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link82.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O83, RadiusVector_O0O7);
            xm[9] = xm0; ym[9] = ym0; zm[9] = zm0; d_8 = d0;
            RotX[9] = RotX0; RotY[9] = RotY0; RotZ[9] = RotZ0;
            Link83 = Mesh.Cylinder(d3d, 0.005f, 0.005f, (float)d_8, 50, 10);
            Link83Material = new Material();
            Link83Material.Diffuse = Color.Yellow;
            Link83Material.Specular = Color.Yellow;
            d3d.Material = Link83Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link83.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O91, RadiusVector_O0O81);
            xm[10] = xm0; ym[10] = ym0; zm[10] = zm0; d_9 = d0;
            RotX[10] = RotX0; RotY[10] = RotY0; RotZ[10] = RotZ0;
            Link91 = Mesh.Cylinder(d3d, 0.005f, 0.005f, (float)d_9, 10, 1);
            Link91Material = new Material();
            Link91Material.Diffuse = Color.White;
            Link91Material.Specular = Color.White;
            d3d.Material = Link91Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link91.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O92, RadiusVector_O0O82);
            xm[11] = xm0; ym[11] = ym0; zm[11] = zm0; d_9 = d0;
            RotX[11] = RotX0; RotY[11] = RotY0; RotZ[11] = RotZ0;
            Link92 = Mesh.Cylinder(d3d, 0.005f, 0.005f, (float)d_9, 10, 1);
            Link92Material = new Material();
            Link92Material.Diffuse = Color.White;
            Link92Material.Specular = Color.White;
            d3d.Material = Link92Material;
```

```
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link92.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0O93, RadiusVector_O0O83);
            xm[12] = xm0; ym[12] = ym0; zm[12] = zm0; d_9 = d0;
            RotX[12] = RotX0; RotY[12] = RotY0; RotZ[12] = RotZ0;
            Link93 = Mesh.Cylinder(d3d, 0.005f, 0.005f, (float)d_9, 10, 1);
            Link93Material = new Material();
            Link93Material.Diffuse = Color.White;
            Link93Material.Specular = Color.White;
            d3d.Material = Link93Material;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            Link93.DrawSubset(0);

            AngleRotationVector(RadiusVector_O0A, Segment_O0O0);
            xm[13] = xm0; ym[13] = ym0; zm[13] = zm0; d_A = d0;
            RotX[13] = RotX0; RotY[13] = RotY0; RotZ[13] = RotZ0;
            vectorA = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_A, 50, 10);
            vectorAMaterial = new Material();
            vectorAMaterial.Diffuse = Color.Red;
            vectorAMaterial.Specular = Color.Red;
            d3d.Material = vectorAMaterial;
            d3d.Transform.World = Matrix.RotationY((float)(RotY0)) *
Matrix.RotationZ((float)(RotZ0)) * Matrix.RotationX((float)(RotX0)) *
Matrix.Translation((float)(xs + xm0), (float)(ys + ym0), (float)(zs + zm0));
            vectorA.DrawSubset(0);
            //Показываем содержимое дублирующего буфера
            d3d.EndScene();
            d3d.Present();
        }

//*****************************************************************************
****

        private void AngleRotationVector(double[] a, double[] b)
        {
            // координаты вектора
            double dX = (b[0] - a[0]), dY = (b[1] - a[1]), dZ = (b[2] - a[2]);
            // длина вектора
            d0 = (Math.Sqrt(dX * dX + dY * dY + dZ * dZ));
            // единичный вектор
            double x1 = 0, y1 = 0, z1 = 0;
            double x = dX / d0;
            double y = dY / d0;
            double z = dZ / d0;
            // координаты середины вектора
            double Xm = (b[0] + a[0]) / 2.0;
            xm0 = Xm;
            double Ym = (b[1] + a[1]) / 2.0;
            ym0 = Ym;
            double Zm = (b[2] + a[2]) / 2.0;
            zm0 = Zm;
            bool xl, yl, zl;
            // определение углов поворотов
            if (Math.Abs(x) < e) { xl = true; }
            else { xl = false; }
            if (Math.Abs(y) < e) { yl = true; }
            else { yl = false; }
            if (Math.Abs(z) < e) { zl = true; }
            else { zl = false; }
            if ((xl && yl && zl) || (xl && yl && !zl))
            {
                RotX0 = 0; RotY0 = 0; RotZ0 = 0;
```

```
            }
            if ((xl && !yl && zl) || (xl && !yl && !zl))
            {
                if (y * z > 0)
                { RotX0 = -Math.Acos(Math.Abs(z / Math.Sqrt(z * z + y * y))); }
                if (y * z < 0)
                { RotX0 = Math.Acos(Math.Abs(z / Math.Sqrt(z * z + y * y))); }
                if (y * z == 0)
                { RotX0 = Math.PI / 2; }
                RotY0 = 0; RotZ0 = 0;
            }
            if ((!xl && yl && zl) || (!xl && yl && !zl))
            {
                RotX0 = 0; RotZ0 = 0;
                if (x * z > 0)
                { RotY0 = Math.Acos(Math.Abs(z / Math.Sqrt(z * z + x * x))); }
                if (x * z < 0)
                { RotY0 = -Math.Acos(Math.Abs(z / Math.Sqrt(z * z + x * x))); }
                if (x * z == 0)
                { RotY0 = Math.PI / 2; }
            }
            if ((!xl && !yl && zl))
            {
                RotX0 = 0;
                RotY0 = Math.PI / 2;
                if (y * x > 0)
                { RotZ0 = Math.Acos(Math.Abs(x / Math.Sqrt(y * y + x * x))); }
                if (y * x < 0)
                { RotZ0 = -Math.Acos(Math.Abs(x / Math.Sqrt(y * y + x * x))); }
            }
            if (Math.Abs(x) < e)
            {
                y1 = Math.Sqrt(y * y + z * z) * Math.Cos(RotX0);
                z1 = Math.Sqrt(y * y + z * z) * Math.Sin(RotX0);
            }
            if (Math.Abs(y) < e)
            {
                z1 = Math.Sqrt(x * x + z * z) * Math.Sin(RotY0);
                x1 = Math.Sqrt(x * x + z * z) * Math.Cos(RotY0);
            }
            if (Math.Abs(z) < e)
            {
                x1 = Math.Sqrt(y * y + x * x) * Math.Cos(RotZ0);
                y1 = Math.Sqrt(y * y + x * x) * Math.Sin(RotZ0);
            }
            if (!xl && !yl && !zl)
            {
                double sinusRotY, sinusRotZ;
                double cosinusRotY;
                RotZ0 = 0;
                sinusRotZ = Math.Sin(RotZ0);
                sinusRotY = x;
                RotY0 = Math.Asin(sinusRotY);
                cosinusRotY = Math.Cos(RotY0);
                if (y > 0 && z > 0) { RotX0 = -Math.Atan(Math.Abs(y / z)); }
                if (y < 0 && z > 0) { RotX0 = Math.Atan(Math.Abs(y / z)); }
                if (y < 0 && z < 0) { RotX0 = Math.PI - Math.Atan(Math.Abs(y / z));
}
                if (y > 0 && z < 0) { RotX0 = Math.PI + Math.Atan(Math.Abs(y / z));
}
                // проверка правильности углов
                double cX = Math.Cos(RotX0);
                double sX = Math.Sin(RotX0);
                double cY = Math.Cos(RotY0);
                double sY = Math.Sin(RotY0);
                double cZ = Math.Cos(RotZ0);
                double sZ = Math.Sin(RotZ0);
```

```
                x1 = sY * cZ;
                y1 = sY * sZ * cX - cY * sX;
                z1 = sY * sZ * sX + cY * cX;
            }
            double dx, dy, dz;
            dx = Math.Abs(x1 - x); dy = Math.Abs(y1 - y); dz = Math.Abs(z1 - z);
        }



//*************************************************************************************
****
        private void MatrixMultiplication(double[,] A, double[,] B, double[,] C)
        {
            int ra = A.GetLength(0), ca = A.GetLength(1);
            int rb = B.GetLength(0), cb = B.GetLength(1);
            if (ca == rb)
            {
                for (int i = 0; i <= ra - 1; i++)
                {
                    for (int j = 0; j <= cb - 1; j++)
                    {
                        double c = 0;
                        for (int r = 0; r <= ca - 1; r++)
                        {
                            double a = 0, b = 0;
                            a = A[i, r]; b = B[r, j];
                            c = c + a * b;
                        }
                        C[i, j] = c;
                    }
                }
            }
        }
        //произведение вектора на матрицу (слева)
        private void MatrixMultiplicationVector(double[] A, double[,] B, double[] C)
        {
            int cA = A.GetLength(0);
            int rB = B.GetLength(0), cB = B.GetLength(1);
            if (rB == cA)
            {
                for (int i = 0; i <= cB - 1; i++)
                {
                    double a = 0;
                    for (int r = 0; r <= cA - 1; r++)
                    {
                        a = a + A[r] * B[r, i];
                    }
                    C[i] = a;
                }
            }
        }
        // векторное произведение
        private void VMultiplicationVector(double[] A, double[] B, double[] C)
        {
          C[0] = A[1] * B[2] - A[2] * B[1]; C[1] = A[2] * B[0] - A[0] * B[2]; C[2] =
A[0] * B[1] - A[1] * B[0];
        }
        private void MatrixInvertion_3(double[,] A, double[,] B)
        {
            double D = 1;
            D = A[0, 0] * A[1, 1] * A[2, 2] - A[0, 0] * A[1, 2] * A[2, 1] + A[0, 2]
* A[1, 0] * A[2, 1] - A[0, 1] * A[1, 0] * A[2, 2] + A[0, 1] * A[1, 2] * A[2, 0] -
A[0, 2] * A[1, 1] * A[2, 0];
            if (Math.Abs(D) > 0)
            {
```

```
            for (int i = 0; i <= 2; i++)
            {
                for (int j = 0; j <= 2; j++)
                {
                    B[j, i] = A[i, j] / D;
                }
            }
        }
    }
    private void MatrixInvertion_N(double[,] A, double[,] B, int n)
    {
        double[,] D = new double[n, n];
        for (int i = 0; i <= n-1; i++)
        {
            for (int j = 0; j <= n-1; j++)
            {
                D[i, j] = A[i, j];
            }
        }
        MatrixDeterminantN(D, n);

        if (Math.Abs(Det_Matrix_D) > 0)
        {
            for (int i = 0; i <= n-1; i ++)
            {
                for (int j = 0; j <= n-1; j ++)
                {
                    B[j, i] = A[i, j] / Det_Matrix_D;
                }
            }
        }
    }

    private void MatrixDeterminantN(double[,] A, int n)
    {
        double[,] D = new double[n, n];
        double Det_D = 1;
        double a;
        int k = 0;
        for (int i = 0; i <= n-1; i++)
        {
            for (int j = 0; j <= n-1; j++)
            {
                D[i, j] = A[i, j];
            }
        }

        while (k <= n - 1)
        {
            int i = k + 1;
            while (i <= n - 1)
            {
                if (D[i, k] != 0)
                {
                    a = -D[k, k] / D[i, k];
                    for (int j = k; j <= n - 1; j++)
                    {
                        D[i, j] = A[i, j] * a + D[k, j];

                    }
                    i = i + 1;
                }
                else
                {
                    i = i + 1;
                }
            }
```

```
                Det_D = Det_D * D[k, k];
                k = k + 1;
            }
            Det_Matrix_D = Det_D;
        }

        // значения элементов матрицы вращения вокруг оси Z
        private void ValueMatrixRotationZ(double[,] MatrixRotation, double
AngleRotation)
        {
            MatrixRotation[0, 0] = Math.Cos(AngleRotation); MatrixRotation[0, 1] =
Math.Sin(AngleRotation); MatrixRotation[0, 2] = 0;
            MatrixRotation[1, 0] = -Math.Sin(AngleRotation); MatrixRotation[1, 1] =
Math.Cos(AngleRotation); MatrixRotation[1, 2] = 0;
            MatrixRotation[2, 0] = 0; MatrixRotation[2, 1] = 0; MatrixRotation[2, 2]
= 1;
        }
        // значения элементов матрицы вращения вокруг оси Y
        private void ValueMatrixRotationY(double[,] MatrixRotation, double
AngleRotation)
        {
            MatrixRotation[0, 0] = Math.Cos(AngleRotation); MatrixRotation[0, 1] =
0; MatrixRotation[0, 2] = -Math.Sin(AngleRotation);
            MatrixRotation[1, 0] = 0; MatrixRotation[1, 1] = 1; MatrixRotation[1, 2]
= 0;
            MatrixRotation[2, 0] = Math.Sin(AngleRotation); MatrixRotation[2, 1] =
0; MatrixRotation[2, 2] = Math.Cos(AngleRotation);
        }
        // значения элементов матрицы вращения вокруг оси X
        private void ValueMatrixRotationX(double[,] MatrixRotation, double
AngleRotation)
        {
            MatrixRotation[0, 0] = 1; MatrixRotation[0, 1] = 0; MatrixRotation[0, 2]
= 0;
            MatrixRotation[1, 0] = 0; MatrixRotation[1, 1] =
Math.Cos(AngleRotation); MatrixRotation[1, 2] = Math.Sin(AngleRotation);
            MatrixRotation[2, 0] = 0; MatrixRotation[2, 1] = -
Math.Sin(AngleRotation); MatrixRotation[2, 2] = Math.Cos(AngleRotation);
        }

//*************************************************************************
****
        private void Form1_Load(object sender, EventArgs e)
        {
            try
            {
                PresentParameters d3dpp = new PresentParameters();
                d3dpp.BackBufferCount = 1;
                d3dpp.SwapEffect = SwapEffect.Discard;
                d3dpp.Windowed = true; // Выводим графику в окно
                d3dpp.MultiSample = MultiSampleType.None; // Выключаем антиалиасинг
                d3dpp.EnableAutoDepthStencil = true; // Разрешаем создание z-буфера
                d3dpp.AutoDepthStencilFormat = DepthFormat.D16; // Z-буфер в 16 бит
                d3d = new Device(0, // D3D_ADAPTER_DEFAULT - видеоадаптер по
умолчанию
                DeviceType.Hardware, // Тип устройства - аппаратный ускоритель
                this, // Окно для вывода графики
                CreateFlags.SoftwareVertexProcessing, // Геометрию обрабатывает CPU
                d3dpp);
            }
            catch (Exception exc)
            {
                MessageBox.Show(this, exc.Message, "Ошибка инициализации");
                Close(); // Закрываем окно
            }
            // оси координат
            axisX = Mesh.Cylinder(d3d, 0.01f, 0.01f, 10f, 50, 10);
```

```
            axisXMaterial = new Material();
            axisXMaterial.Diffuse = Color.Blue;
            axisXMaterial.Specular = Color.White;
            axisY = Mesh.Cylinder(d3d, 0.01f, 0.01f, 10f, 50, 10);
            axisYMaterial = new Material();
            axisYMaterial.Diffuse = Color.Yellow;
            axisYMaterial.Specular = Color.White;
            axisZ = Mesh.Cylinder(d3d, 0.01f, 0.01f, 10f, 50, 10);
            axisZMaterial = new Material();
            axisZMaterial.Diffuse = Color.Orange;
            axisZMaterial.Specular = Color.White;
            //cilindr0 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_1, 50, 10);
            //cilindr0Material = new Material();
            //cilindr0Material.Diffuse = Color.White;
            //cilindr0Material.Specular = Color.White;
            Link1 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_2, 50, 10);
            Link1Material = new Material();
            Link1Material.Diffuse = Color.White;
            Link1Material.Specular = Color.White;
            Link2 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_3, 50, 10);
            Link2Material = new Material();
            Link2Material.Diffuse = Color.White;
            Link2Material.Specular = Color.White;
            Link3 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_4, 50, 10);
            Link3Material = new Material();
            Link3Material.Diffuse = Color.White;
            Link3Material.Specular = Color.White;
            Link4 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_5, 50, 10);
            Link4Material = new Material();
            Link4Material.Diffuse = Color.White;
            Link4Material.Specular = Color.White;
            Link5 = Mesh.Cylinder(d3d, 0.01f, 0.01f, (float)d_6, 50, 10);
            Link5Material = new Material();
            Link5Material.Diffuse = Color.White;
            Link5Material.Specular = Color.White;
        }
        private void SetupProekcii()
        {
            // Устанавливаем параметры источника освещения
            d3d.Lights[0].Enabled = true;    // Включаем нулевой источник освещения
            d3d.Lights[0].Diffuse = Color.White;          // Цвет источника освещения
            d3d.Lights[0].Position = new Vector3(0, 0, 0); // Задаем координаты
            d3d.Transform.Projection = Matrix.PerspectiveFovLH((float)(Math.PI / 2),
this.Width / this.Height, 1.0f, 50.0f);
            //d3d.Transform.View
        }
        private void PictureDvig()
        {
            d3d.EndScene();
            //Показываем содержимое дублирующего буфера
            d3d.Present();
        }


//*****************************************************************************
****


    }
}
```

Литература

1. Курош. А. Г. Курс высшей алгебры. Москва.: «Наука», 1965. 431 с.

2. Погорелов А.В. Аналитическая геометрия. Москва.: «Наука», 1968. 176 с.

3. Курант Р. Курс дифференциального и интегрального исчисления, том 1. Москва.: «Наука», 1967. 704 с.

4. Артоболевский И.И. Теория механизмов и машин. Москва.: «Наука», 1988. 640 с.

5. Иосилевич Г.Б. Прикладная механика. Москва.: Высшая школа, 1989. 350 с.

6. Бурдаков С.Ф., Дьяченко В.А., Тимофеев А.Н. Проектирование манипуляторов промышленных роботов и роботизированных комплексов. Москва.: «Высшая школа», 1986. 263с.

7. Юревич Е.И. Основы робототехники. Ленинград.: «Машиностроение», 1985. 271 с.

8. Макаров *И. М.,* Топчеев *Ю. И.* Робототехника: История и перспективы. Москва.: «Наука», 2003. 349 с.

9. Пуриш В.З. Основы андроавтоматики. Проектирование роботов андроидов. Николаев: Изд-во ЧГУ им. Петра Могилы, 2010. 312 с.

10. А.А. Мельник, В.Н. Хоменко, П.С. Плис, П. Энафф, В.Ф. Борисенко Кинематическая модель робота с шестью степенями свободы и возможностью учёта зазора в суставах.: Научные работы Донецкого национального технического университета № 10(180)