Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет систем управления и робототехники

Отчет по лабораторной работе №3, 4

«Прямая и обратная задача кинематики. Траекторное управление и задача планирования траектории»

по дисциплине «Введение в профессиональную деятельность»

Выполнили: студенты гр. R3135
Дупак А. А.,
Щтенников Р. А.,
Зорькина А. А.

Преподаватель: Перегудин А. А.

Санкт-Петербург
2019

Цель работы

Ознакомиться со способом нахождения параметров манипулятора и научитсья переходить из декартовых координат в обобщенные и обратно. Ознакомиться с методами планирования траектории, решить проблему следования за траекторией.

Материалы работы

Результаты необходимых расчетов и построений

Таблица 1. DH-параметры манипулятора.

Узел	a_i, m	α_i, rad	d_i, m	θ_i
1	0	π/2	0.14	θ_1*
2	0.12	0	0	θ_2*
3	0.12	0	0	θ_3*

* переменные

Таблица 2. Коэффициенты ПИД-регулятора.

k_p	k_i	k_d	anti-windup
5	0.07	0.5	10

Таблица 3. Координаты точек в декартовой и обобщенной системе.

n	*X, m*	*Y, m*	*Z, m*	θ_1, °	θ_2, °	θ_3, °
1	0.1	0.1	0.1	45	54	104
2	0.1	0.1	0.3	45	14	54
3	0.1	-0.1	0.3	-45	14	54
4	0.1	-0.1	0.1	-45	54	104

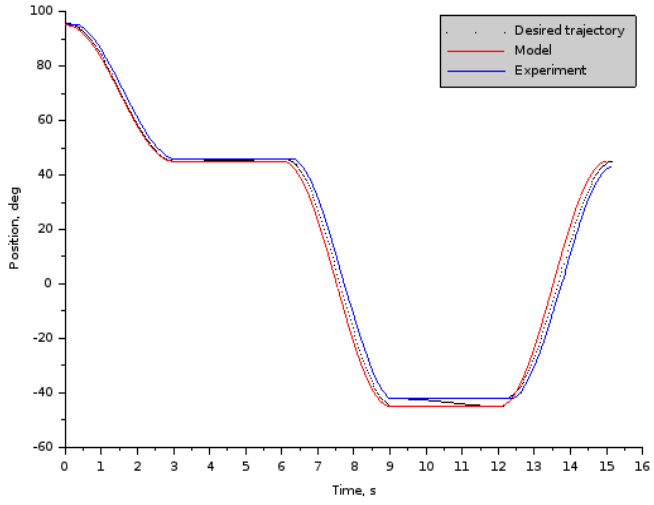

Рис. 1. График переходного процесса регулятора угла поворота первого узла манипулятора.

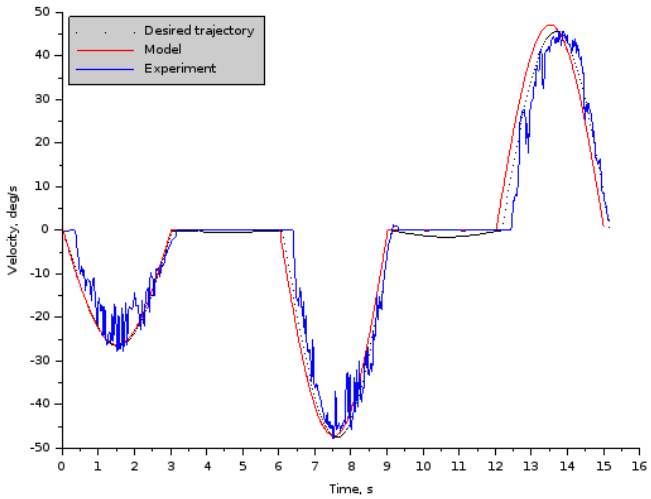

Рис. 2. График переходного процесса регулятора угловой скорости первого узла манипулятора.

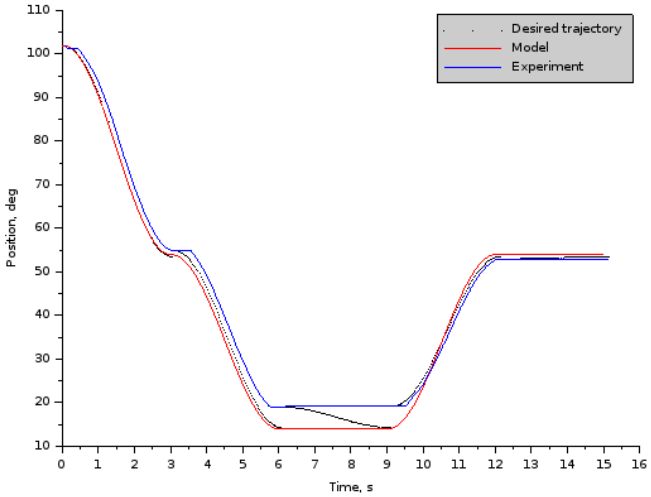

Рис. 3. График переходного процесса регулятора угла поворота второго узла манипулятора.

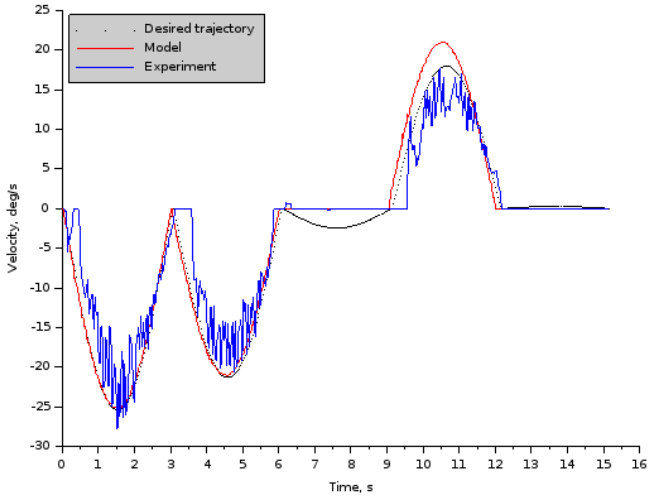

Рис. 4. График переходного процесса регулятора угловой скорости второго узла манипулятора.

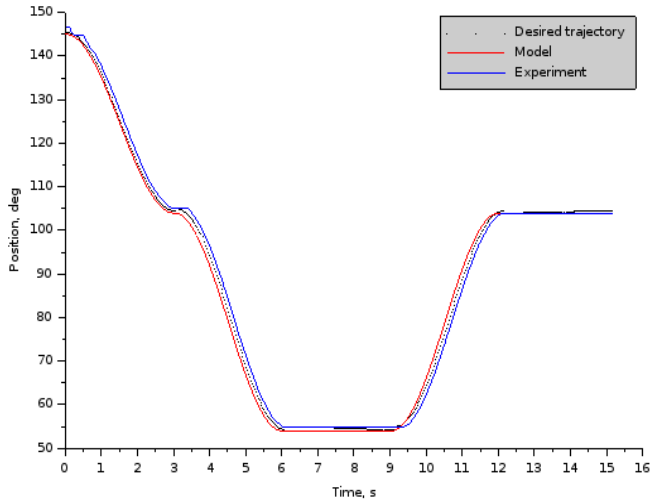

Рис. 5. График переходного процесса регулятора угла поворота третьего узла манипулятора.

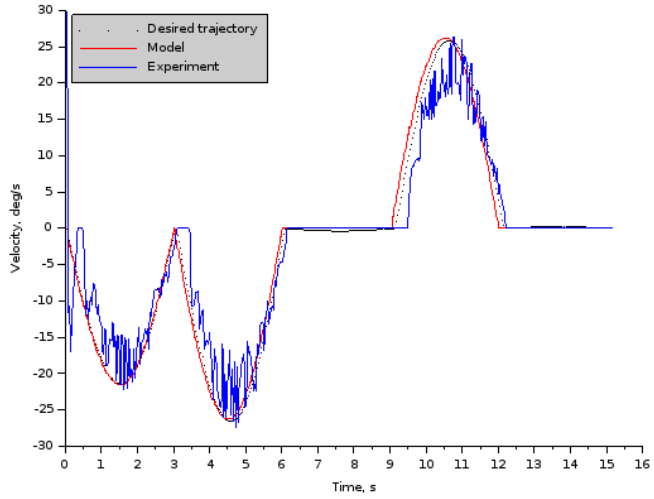

Рис. 6. График переходного процесса регулятора угловой скорости третьего узла манипулятора.

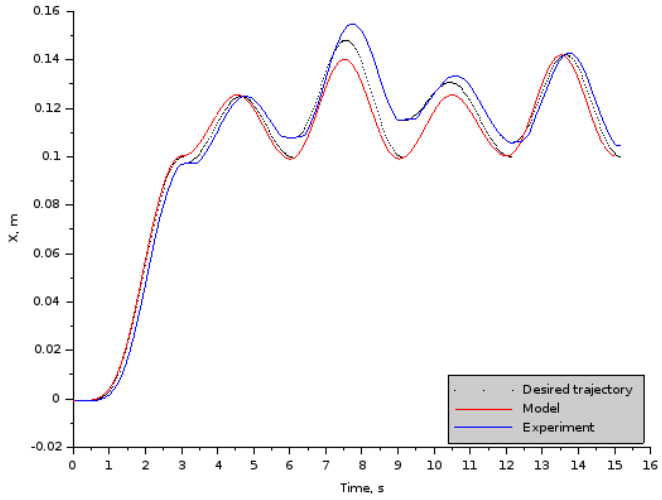

Рис. 7. График зависимости координаты X рабочего инструмента манипулятора от веремени.

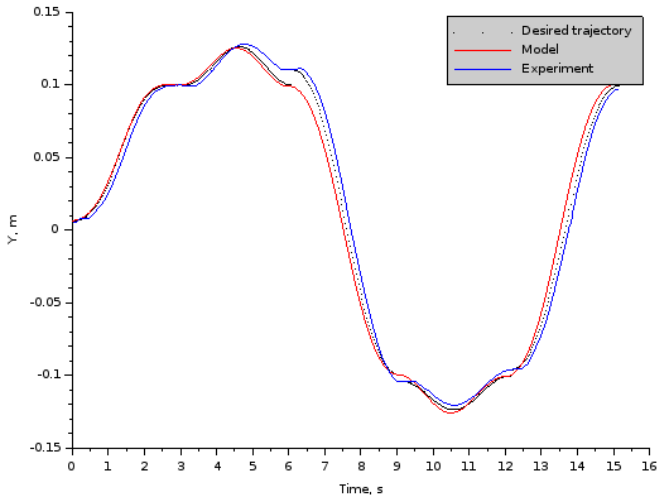

Рис. 8. График зависимости координаты Y рабочего инструмента манипулятора от веремени.

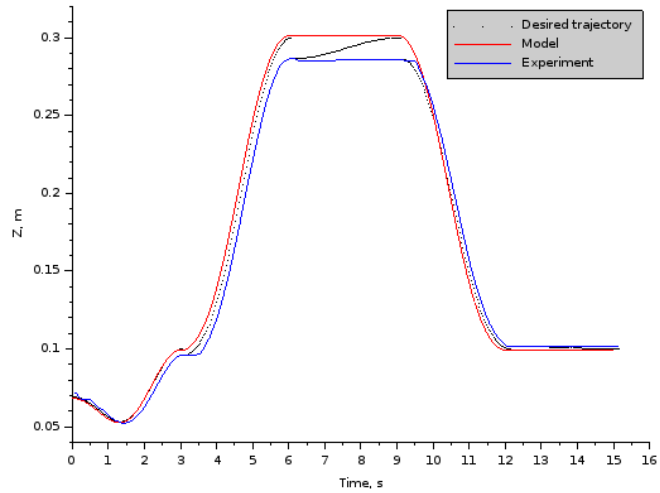

Рис. 9. График зависимости координаты Z рабочего инструмента манипулятора от веремени.

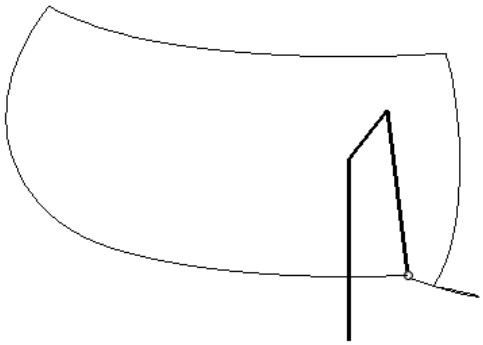

Рис. 10. Изображение траектории движения рабочего инструмента манипулятора через точки задающие квадрат.

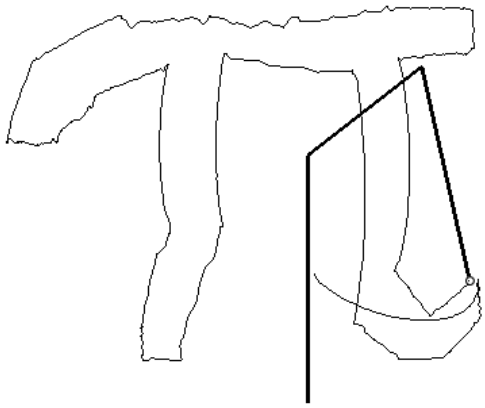

Рис. 11. Изображение траектории движения рабочего инструмента манипулятора через точки задающие число π.

Плинирование траектории[1]

Гармоническая траектория основанная на тригонометрических функциях имеет следующую математическую формулировку:

$$s(\theta)=R(1-\cos\theta) \quad ,$$

где R – радиус окружности.

В более общем случае, гармоническая траектория может быть определена как

$$q(t)=\frac{q_1-q_0}{2}(1-\cos\frac{t-t_0}{t_p})+q_0 \quad ,$$

где q_0, q_1 – начальное и конечное значение угла поворота вала двигателя, t_0, t_1 - начальное и конечное значение времени, t – текущее время, t_p – время переходного процесса.

Продифференцировав представленное выражение по времени получим формулу для угловой скорости вражения вала двигателя:

$$\dot{q}(t)=\frac{\pi(q_1-q_0)}{2t_p}\sin(\frac{\pi(t-t_0)}{t_p})$$

Код программы для EV3

а) Независимое управление узлами манипулятора с помощью ПИД-регулятора.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
import math

route = [[0.1, 0.1, 0.1], [0.1, 0.1, 0.3], [0.1, -0.1, 0.3], [0.1, -0.1,
0.1], [0.1, 0.1, 0.1]]

kp = 5
ki = 0.07
kd = 0.5
anti_windup = 10

d1 = 0.14
a2 = 0.12
a3 = 0.12
```

1 Biagiotti L., Melchiorri C. Trajectory planning for automatic machines and robots. – Springer Science & Business Media, 2008.

MLA

```
gear = [-60/8, 40/8, -40/8]
limit = [[90*math.pi/180, -90*math.pi/180],[90*math.pi/180,-
30*math.pi/180],[135*math.pi/180,-30*math.pi/180]]
motor = [LargeMotor('outA'), LargeMotor('outB'), LargeMotor('outC')]
sensor = [TouchSensor('in1'), TouchSensor('in2'), TouchSensor('in3')]
file_obj = open("data_ind.txt", "w")

integral = err_old = {}
for i in range(3):
    integral[i] = 0
    err_old[i] = 0


def calibrate():
    for i in range(3):
        while (sensor[i].is_pressed == False):
            motor[i].run_direct(duty_cycle_sp = 25*(-1)**(i+1))
        motor[i].position = limit[i][0]*180/math.pi*gear[i]


def inverse_kinematics(x, y, z):
    r1=math.sqrt(x**2+y**2)
    r3=math.sqrt(x**2+y**2+(z-d1)**2)
    if r3 > a2 + a3:
        return False, {}

    desired_angle = {}
    if x == 0:
        if y == 0:
            desired_angle[0] = 0
        elif y > 0:
            desired_angle[0] = math.pi/2
        else:
            desired_angle[0] = -math.pi/2
    elif y == 0:
        if x > 0:
            desired_angle[0] = 0
        else:
            desired_angle[0] = math.pi
    else:
        desired_angle[0] = math.atan2(y, x)

    if z - d1 == 0:
        desired_angle[1] = math.pi/2 - math.acos((a2**2+r3**2-a3**2)/
(2*a2*r3))
    elif z - d1 > 0:
        desired_angle[1] = math.atan(r1/(z-d1)) -
math.acos((a2**2+r3**2-a3**2)/(2*a2*r3))
    else:
        desired_angle[1] = math.pi + math.atan(r1/(z-d1)) -
math.acos((a2**2+r3**2-a3**2)/(2*a2*r3))

    desired_angle[2] = math.pi - math.acos((a2**2+a3**2-r3**2)/
(2*a2*a3))

    for i in range(3):
```

```
        if desired_angle[i] > limit[i][0] or desired_angle[i] < limit[i]
[1]:
            return False, desired_angle
        desired_angle[i] = desired_angle[i]*gear[i]
    return True, desired_angle


try:
    calibrate()
    start_time = time.time()
    for j in range(len(route)):
        reachable, desired_angle = inverse_kinematics(route[j][0],
route[j][1], route[j][2])
        if reachable:
            err = derivative = u = {}
            t0 = time.time() - start_time
            while (time.time() - t0 - start_time < 3):
                s = ""
                for i in range(3):
                    err[i] = desired_angle[i] -
motor[i].position*math.pi/180
                    integral[i] += err[i]*ki
                    if abs(integral[i]) > anti_windup:
                        integral[i] = math.copysign(anti_windup,
integral[i])
                    derivative[i] = (err[i] - err_old[i])*kd
                    err_old[i] = err[i]
                    u[i] = kp * err[i] + integral[i] + derivative[i]
                    u[i] = u[i]*100/8.4
                    if abs(u[i]) > 100:
                        u[i] = math.copysign(100, u[i])
                    motor[i].run_direct(duty_cycle_sp = u[i])
                    s += str(motor[i].position) + " "
                file_obj.write(s + str(time.time() - start_time) +
'\n \n')
            print("moved to (" + str(route[j][0]) + ", " + str(route[j]
[1]) + ", " + str(route[j][2]) + ")")
        else:
            print("unreachable point (" + str(route[j][0]) + ", " +
str(route[j][1]) + ", " + str(route[j][2]) + ")")


finally:
    for i in range(3):
        motor[i].stop(stop_action='brake')
    file_obj.close
```

б) Траекторное управление и планирование траектории движения узлов манипулятора.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
import math

route = [[0.1, 0.1, 0.1], [0.1, 0.1, 0.3], [0.1, -0.1, 0.3], [0.1, -0.1,
0.1], [0.1, 0.1, 0.1]]

kc = 5
kp = 0.07
```

```
ki = 0.5
kd = 0
anti_windup = 10
tp = 3

d1 = 0.14
a2 = 0.12
a3 = 0.12
gear = [-60/8, 40/8, -40/8]
limit = [[90*math.pi/180, -90*math.pi/180],[95*math.pi/180,-
30*math.pi/180],[145*math.pi/180,-30*math.pi/180]]
motor = [LargeMotor('outA'), LargeMotor('outB'), LargeMotor('outC')]
sensor = [TouchSensor('in1'), TouchSensor('in2'), TouchSensor('in3')]
file_obj = open("robot_data.txt", "w")

derivative = u = {}
integral = err_old = {}
for i in range(3):
    integral[i] = 0
    err_old[i] = 0

def calibrate():
    for i in range(3):
        while (sensor[i].is_pressed == False):
            motor[i].run_direct(duty_cycle_sp = 25*(-1)**(i+1))
        motor[i].position = limit[i][0]*180/math.pi*gear[i]

def inverse_kinematics(x, y, z):
    r1=math.sqrt(x**2+y**2)
    r3=math.sqrt(x**2+y**2+(z-d1)**2)
    if r3 > a2 + a3:
        return False, {}

    desired_angle = {}
    if x == 0:
        if y == 0:
            desired_angle[0] = 0
        elif y > 0:
            desired_angle[0] = math.pi/2
        else:
            desired_angle[0] = -math.pi/2
    elif y == 0:
        if x > 0:
            desired_angle[0] = 0
        else:
            desired_angle[0] = math.pi
    else:
        desired_angle[0] = math.atan2(y, x)

    if z - d1 == 0:
        desired_angle[1] = math.pi/2 - math.acos((a2**2+r3**2-a3**2)/
(2*a2*r3))
    elif z - d1 > 0:
        desired_angle[1] = math.atan(r1/(z-d1)) -
math.acos((a2**2+r3**2-a3**2)/(2*a2*r3))
    else:
```

```
        desired_angle[1] = math.pi + math.atan(r1/(z-d1)) -
math.acos((a2**2+r3**2-a3**2)/(2*a2*r3))

    desired_angle[2] = math.pi - math.acos((a2**2+a3**2-r3**2)/
(2*a2*a3))

    for i in range(3):
        if desired_angle[i] > limit[i][0] or desired_angle[i] < limit[i]
[1]:
            print(str(i)+": "+str(desired_angle[i]*180/math.pi))
            return False, desired_angle
        desired_angle[i] = desired_angle[i]*gear[i]
    return True, desired_angle

try:
    calibrate()
    start_time = time.time()
    for j in range(len(route)):
        reachable, desired_angle = inverse_kinematics(route[j][0],
route[j][1], route[j][2])
        if reachable:
            start_angle = {}
            for i in range(3):
                start_angle[i] = motor[i].position*math.pi/180
            t0 = time.time() - start_time
            while (time.time() - t0 - start_time < tp):
                s = ""
                t = time.time() - start_time
                for i in range(3):
                    angle = ((desired_angle[i] - start_angle[i])/2)*(1 -
math.cos(math.pi*(t - t0)/tp)) + start_angle[i]
                    velocity = (math.pi*(desired_angle[i] -
start_angle[i])*math.sin((math.pi*(t - t0))/tp))/(2*tp)
                    err = kc*(angle - motor[i].position*math.pi/180) #+
velocity - motor[i].speed*math.pi/180
                    integral[i] += err*ki
                    if abs(integral[i]) > anti_windup:
                        integral[i] = math.copysign(anti_windup,
integral[i])
                    derivative = (err - err_old[i])*kd
                    err_old[i] = err
                    u = kp * err + integral[i] + derivative
                    u = u*100/8.4
                    if abs(u) > 100:
                        u = math.copysign(100, u)
                    motor[i].run_direct(duty_cycle_sp = u)
                    s += str(motor[i].position/gear[i]) + " " +
str(angle*180/math.pi/gear[i]) + " " + str(motor[i].speed/gear[i]) + " "
+ str(velocity*180/math.pi/gear[i]) + " "
                file_obj.write(s + str(time.time() - start_time) +
'\n \n')
            print("moved to (" + str(route[j][0]) + ", " + str(route[j]
[1]) + ", " + str(route[j][2]) + ")")
        else:
            print("unreachable point (" + str(route[j][0]) + ", " +
str(route[j][1]) + ", " + str(route[j][2]) + ")")
```

```
finally:
    for i in range(3):
        motor[i].stop(stop_action='brake')
    file_obj.close
```

Схема моделирования

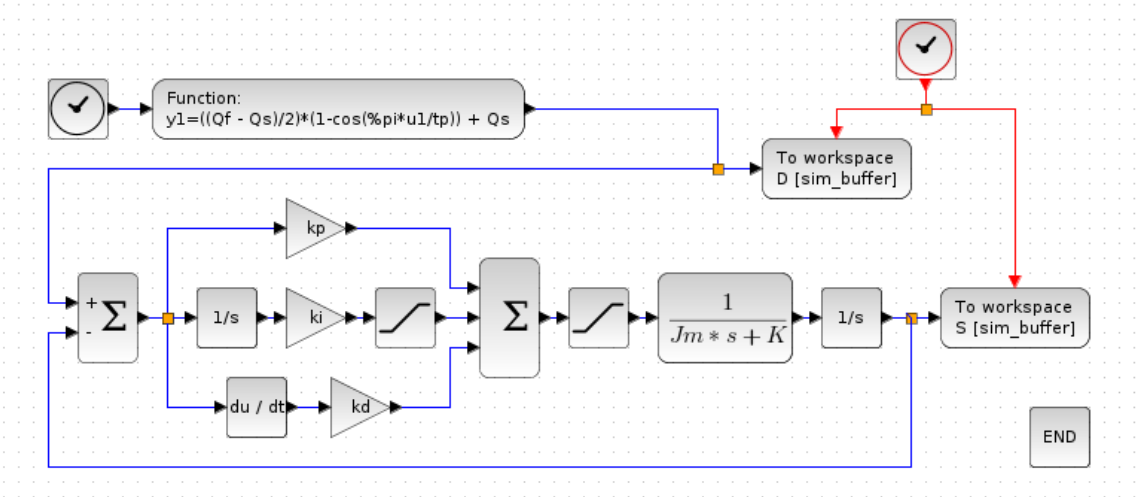

Рис. 12. Схема моделирования траекторного регулятора для узла манипулятора.

Код программы Scilab

а) Расчет приямой и обратной задачи кинематики.

```
clear()
a1=0
a2=0.12
a3=0.12
d1=0.14

function [x, y, z]=forward_kinematics(q)
   T1=[cos(q(1)-%pi),0,sin(q(1)-%pi),a1*cos(q(1)-%pi);
      sin(q(1)-%pi),0,-cos(q(1)-%pi),a1*sin(q(1)-%pi);
      0,1,0,d1;
      0,0,0,1]
   T2=[cos(q(2)+%pi/2),-sin(q(2)+%pi/2),0,a2*cos(q(2)+%pi/2);
      sin(q(2)+%pi/2),cos(q(2)+%pi/2),0,a2*sin(q(2)+%pi/2);
      0,0,1,0;
      0,0,0,1]
   T3=[cos(q(3)),-sin(q(3)),0,a3*cos(q(3));
      sin(q(3)),cos(q(3)),0,a3*sin(q(3));
      0,0,1,0;
      0,0,0,1]
   T = T1*T2*T3
   x = T(1,4)
   y = T(2,4)
   z = T(3,4)
endfunction

function q=inverse_kinematics(x, y, z)
   r1=sqrt(x^2+y^2)
   r3=sqrt(x^2+y^2+(z-d1)^2)
   if r3 > a2 + a3
      return [0; 0; 0]
   end
   if x == 0
```

```
        if y == 0
            q(1) = 0
        elseif y > 0
            q(1) = %pi/2
        else
            q(1) = -%pi/2
        end
    elseif y == 0 and x < 0
        q(1) = %pi
    else
        if x>0
            q(1) = atan(y/x)
        else
            if y>0
                q(1) = %pi+atan(y/x)
            else
                q(1) = -%pi+atan(y/x)
            end
        end
    end
    if z - d1 == 0
        q(2) = %pi/2 - acos((a2**2+r3**2-a3**2)/(2*a2*r3))
    elseif z - d1 > 0
        q(2) = atan(r1/(z-d1)) - acos((a2**2+r3**2-a3**2)/(2*a2*r3))
    else
        q(2) = %pi + atan(r1/(z-d1)) - acos((a2**2+r3**2-a3**2)/(2*a2*r3))
    end
    q(3) = %pi - acos((a2**2+a3**2-r3**2)/(2*a2*a3))
endfunction
```

б) Построение графиков переходного процесса регулятора.

```
clear()
file_name = "sqr_traj.txt"
path = "/home/aleksandr/ITMO_lab2/ev3/lab3/X/"
res = read(path + file_name, -1, 13)
y1(:, 1) = res(:, 3)
y1(:, 2) = res(:, 7)
y1(:, 3) = res(:, 11)
y2(:, 1) = res(:, 4)
y2(:, 2) = res(:, 8)
y2(:, 3) = res(:, 12)
x1 = res(:, 13)

d1=0.14
a2=0.12
a3=0.12
a1=0

sim_time=3
tp=sim_time
sim_period=0.01
sim_buffer=sim_time/sim_period
kp=5
ki=0.07
anti_windup=10
kd=0.5
voltage=8.4
Jm=0.0025
K=0.043
gear=[60/8, 40/8, 40/8]

route=[95, 102, 145;
```

```
     45, 54, 104;
     45, 14, 54;
     -45, 14, 54;
     -45, 54, 104;
     45, 54, 104]
route=route*%pi/180

for i = 1:3
   traj = []
   sim = []
   sp=[]
   for j = 2:length(route)/3
      Qf = route(j, i)*gear(i)
      Qs = route(j-1, i)*gear(i)
      importXcosDiagram("/home/aleksandr/ITMO_lab2/ev3/lab3/scilab/model.zcos");
      xcos_simulate(scs_m, 4);
      sp=[sp;Sp.values]
      traj = [traj; D.values]
      sim = [sim; S.values]
   end
   q1(:, i) = sim
   f(i)=figure(i)
   f(i).background = 8
   plot2d(x1, y2(:, i), 0)
   plot2d((1:length(traj))/100, sp*180/%pi/gear(i), 5)
   plot2d(x1, y1(:, i), 2)
   xtitle('', 'Time, s', 'Velocity, deg/s')
   legend("Desired trajectory", "Model", "Experiment", 2)
   xs2png(i, "/home/aleksandr/ITMO_lab2/ev3/lab3/photos/2sp_"+string(i)+".png")
end
```

в) Построение 3д изображения траектории движения манипулятора.

```
clear()
file_name = "pi_traj.txt"
path = "/home/aleksandr/ITMO_lab2/ev3/lab3/X/"
res = read(path + file_name, -1, 13)
ex(:, 1) = res(:, 1)
ex(:, 2) = res(:, 5)
ex(:, 3) = res(:, 9)
time = res(:, 13)

d1=0.14
a2=0.12
a3=0.12
a1=0

for i = 1:length(time)
   q(1) = ex(i, 1)*%pi/180
   q(2) = ex(i, 2)*%pi/180
   q(3) = ex(i, 3)*%pi/180
      [x2(i),y2(i),z2(i)]=forward_kinematics(q)
      D1(i)=d1
      x1(i) = -a2*cos(q(1))*sin(-q(2))
      y1(i) = -a2*sin(q(1))*sin(-q(2))
      z1(i) = a2*cos(q(2)) + d1
end

zer = zeros(length(x1), 1)
X1=[0, 0]; Y1=[0, 0]; Z1=[0, d1]
X2=[zer'; x1']; Y2=[zer'; y1']; Z2=[D1'; z1']
X3=[x1'; x2']; Y3=[y1'; y2']; Z3=[z1'; z2']
f1=figure()
```

```
f1.background=8
f1.children.thickness=3
xsegs(X1, Y1, Z1)
xsegs(X2(:, length(x1)), Y2(:, length(x1)), Z2(:, length(x1)))
xsegs(X3(:, length(x1)), Y3(:, length(x1)), Z3(:, length(x1)))
f1.children.thickness=1
comet3d(x2, y2, z2)
```

Выводы

В результате проделанной работы были найдены DH-параметры манипулятора и расчитана прямая и обратная задачи кинематики. Также было выполнено планирование траектории с помощью тригонометрических функций и реализовано траекторное управление манипулятором с помощью ПИД-регулятора. Кроме того, была построена схема моделирования исследуемого процесса в среде Xcos. Сравнив график экспериментальных данных с графиком построенным на основании результатов моделирования схемы, мы удостоверились в том, что моделирование исследуемого процесса дает результаты схожие со значеними полученными в ходе прямых измерений.