

**Министерство науки и высшего образования**

**Российской Федерации**

**федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технологический университет «СТАНКИН»**

**(ФГБОУ ВО «МГТУ «СТАНКИН»)**

Институт цифровых интеллектуальных систем

Кафедра робототехники и мехатроники

Дисциплина «Управление роботами и робототехническими системами»

**ОТЧЁТ**

**по лабораторной работе №2**

**на тему:**

## «Изучение решения обратной задачи кинематики и методов траекторного управления»

Выполнил:

студент группы АДМ-21-05 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ Пивкин Д.П.

(дата) (подпись) (ФИО)

Принял

преподаватель: \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ Порунов М.Ю.

(дата) (подпись) (ФИО)

Оценка: \_\_\_\_\_\_\_\_ Дата:\_\_\_\_\_\_\_\_\_

Москва 2022

**Цель работы:** Изучить аналитическое решение обратной задачи кинематики на примере манипуляторов SCARA и PUMA и кастомного робота. Также изучить математический аппарат, применяемый для решения задач траекторного управления.

**Задание №1**

Решим обратную задачу кинематики для ABB IRB 140, и для своего закона изменения целевого положения построим графики изменения обобщенных координат, а также оценим скорость изменения обобщенных координат (проведя, численное дифференцирование).

На рисунке 1 представлен робот ABB IRB 140, который обладает шестью степенями подвижности.



Рис. 1. Робот-манипулятор ABB IRB 140

В листинге 1 приведен код для реализации работы кинематики манипулятора, а в листинге 2 построение графика изменения обобщенных координат. Оценка скорости изменения обобщенных координат показана в листинге 3.

Листинг 1 – Решение задания 2.1

*//****Укажем длины звеньев***

irb\_l **=** [352.0, 70.0, 350.0, 380.0, 65.0]

*//****Учтем диапазон изменения обобщенных координат (град.)***

irb\_lim **=** [

(**-**180, 180),

(**-**90, 110),

(**-**230, 50),

(**-**200, 200),

(**-**115, 115),

(**-**400, 400)

]

***//Решим прямую задачу кинематики***

**def** irb\_chain(q, l):

base **=** Transform**.**identity()

column **=** base **+** Transform(

Vector(0, 0, l[0]),

Quaternion**.**from\_angle\_axis(q[0], Vector(0, 0, 1))

)

shoulder **=** column **+** Transform(

Vector(l[1], 0, 0),

Quaternion**.**from\_angle\_axis(q[1], Vector(0, **-**1, 0))

)

elbow **=** shoulder **+** Transform(

Vector(0, 0, l[2]),

Quaternion**.**from\_angle\_axis(q[2], Vector(0, 1, 0))

)

wrist **=** elbow **+** Transform(

Vector(l[3], 0, 0),

Quaternion**.**from\_angle\_axis(q[3], Vector(1, 0, 0)) **\***

Quaternion**.**from\_angle\_axis(q[4], Vector(0, 1, 0))

)

flange **=** wrist **+** Transform(

Vector(l[4], 0, 0),

Quaternion**.**from\_angle\_axis(q[5], Vector(1, 0, 0)) **\***

Quaternion**.**from\_angle\_axis(np**.**pi **/** 2, Vector(0, 1, 0))

)

**return** [

base,

column,

shoulder,

elbow,

wrist,

flange

]

***//Решим обратную задачу кинематики***

def wrap\_from\_to(value, s, e):

r = e - s

return value - (r \* np.floor((value - s) / r))

def irb\_ik(target, l, i=[1, 1, 1]):

wrist = target + Vector(0, 0, -l[4]) + Vector(0, 0, -l[0])

projection = Vector(wrist.x, wrist.y, 0)

q0 = Vector(0, 1, 0).angle\_to(projection, Vector(0, 0, 1)) - np.pi / 2 \* i[0] + np.pi

d = ((projection.magnitude() - i[0] \* l[1]) \*\* 2 + wrist.z \*\* 2) \*\* 0.5

q2 = -i[1] \* np.arccos(

(l[2] \*\* 2 + l[3] \*\* 2 - d \*\* 2) /\

(2 \* l[2] \* l[3])

) + np.pi / 2

triangle\_angle = np.arcsin(

l[3] \* i[0] \* np.sin(q2 - np.pi / 2) / d

)

lift\_angle = np.arctan2(

wrist.z,

(projection.magnitude() - i[0] \* l[1])

)

q1 = -i[0] \* (np.pi / 2 + triangle\_angle - lift\_angle)

ori = Quaternion.from\_angle\_axis(q0, Vector(0, 0, 1)) \*\

Quaternion.from\_angle\_axis(q1, Vector(0, -1, 0)) \*\

Quaternion.from\_angle\_axis(q2, Vector(0, 1, 0))

ez = ori \* Vector(1, 0, 0)

ey = ori \* Vector(0, 1, 0)

tz = target.rotation \* Vector(0, 0, 1)

ty = target.rotation \* Vector(0, 1, 0)

wy = ez.cross(tz)

q3 = ey.angle\_to(wy, ez) + np.pi / 2 - np.pi / 2 \* i[2]

q4 = ez.angle\_to(tz, wy) \* i[2]

q5 = wy.angle\_to(ty, tz) + np.pi / 2 -np.pi / 2 \* i[2]

return (

wrap\_from\_to(q0, -np.pi, np.pi),

wrap\_from\_to(q1, -np.pi, np.pi),

wrap\_from\_to(q2, -np.pi, np.pi),

wrap\_from\_to(q3, -np.pi, np.pi),

wrap\_from\_to(q4, -np.pi, np.pi),

wrap\_from\_to(q5, -np.pi, np.pi)

)

**//Зададим закон изменения положения**

def target(t, total):

return Transform(

Vector(100, 500, 1000 \* t / total) if t / total < 0.5 else Vector(100 + (t / total - 0.5) \* 500, 500 + (t / total - 0.5) \* 100, 500),

Quaternion.from\_angle\_axis(

(t / total) / 2 \* np.pi + np.pi,

Vector(1, 0, 0)

)

)

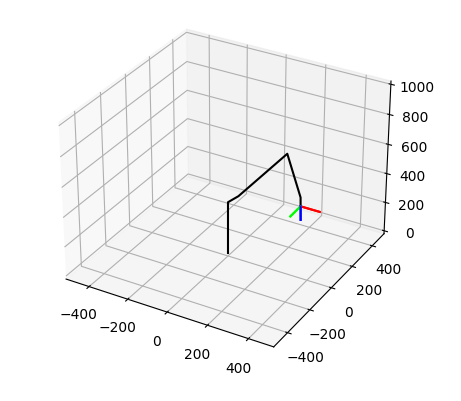


Рис. 2 Результат выполнения программы

Листинг 2 – График изменения обобщенных координат

v\_target **=** np**.**vectorize(target, excluded**=**{1})

v\_irb\_ik **=** np**.**vectorize(irb\_ik, excluded**=**{1, 2})

total **=** 20

step **=** 0.01

t **=** np**.**arange(0, total, step)

fig **=** plt**.**figure()

ax **=** fig**.**add\_subplot()

q **=** v\_irb\_ik(

v\_target(t, total),

irb\_l,

irb\_i

);

ax**.**plot(t, q[0], label**=**"$q\_0$")

ax**.**plot(t, q[1], label**=**"$q\_1$")

ax**.**plot(t, q[2], label**=**"$q\_2$")

ax**.**plot(t, q[3], label**=**"$q\_3$")

ax**.**plot(t, q[4], label**=**"$q\_4$")

ax**.**plot(t, q[5], label**=**"$q\_5$")

fig**.**legend()

fig**.**show()

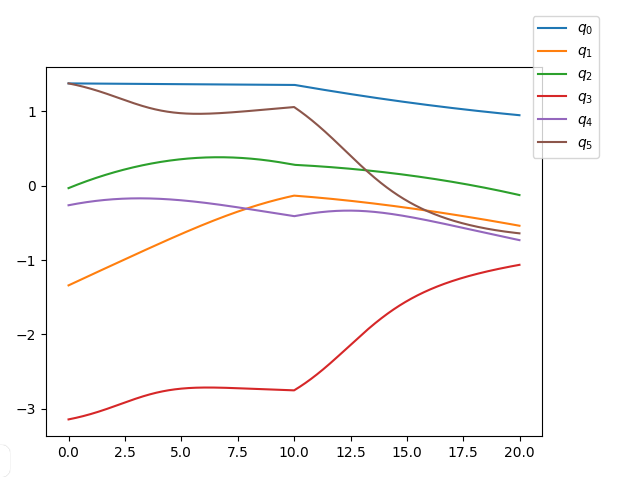


Рис. 3 График изменения обобщенных координат

Листинг 3 - Оценим скорость изменения обобщенных координат

v\_target **=** np**.**vectorize(target, excluded**=**{1})

v\_irb\_ik **=** np**.**vectorize(irb\_ik, excluded**=**{1,2})

total **=** 20

step **=** 0.01

t **=** np**.**arange (0, total, step)

fig **=** plt**.**figure()

ax **=** fig**.**add\_subplot()

q **=** v\_irb\_ik (

v\_target(t, total),

irb\_l,

irb\_i

);

ax**.**plot(t[:**-**1], np**.**diff(q[0])**/**step, label**=**"$w\_q0$")

ax**.**plot(t[:**-**1], np**.**diff(q[1])**/**step, label**=**"$w\_q1$")

ax**.**plot(t[:**-**1], np**.**diff(q[2])**/**step, label**=**"$w\_q2$")

ax**.**plot(t[:**-**1], np**.**diff(q[3])**/**step, label**=**"$w\_q3$")

ax**.**plot(t[:**-**1], np**.**diff(q[4])**/**step, label**=**"$w\_q4$")

ax**.**plot(t[:**-**1], np**.**diff(q[5])**/**step, label**=**"$w\_q5$")

fig**.**legend()

fig**.**show()

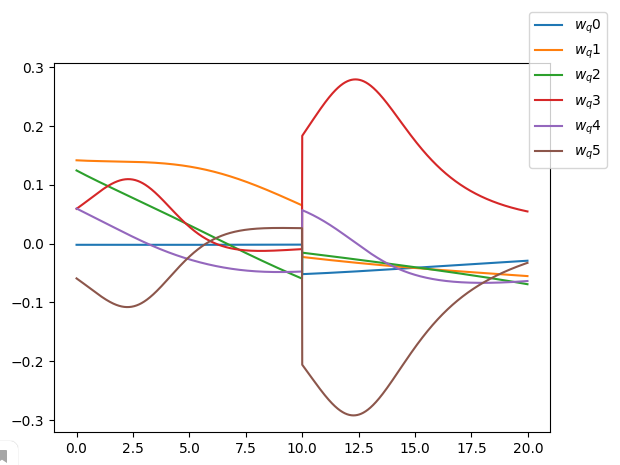


Рис. 4 Скорость изменения обобщенных координат

# **Задание №2**

Решим обратную задачу кинематики для SCARA, и для своего закона изменения целевого положения построим графики изменения обобщенных координат, а также оценим скорость изменения обобщенных координат (проведя, численное дифференцирование).

На рисунке 5 представлен робот SCARA, который обладает четырьмя степенями подвижности.

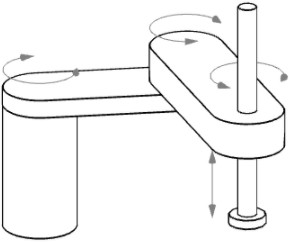


Рис. 5 Робот SCARA

В листинге 4 приведен код для реализации работы кинематики манипулятора, построения графика изменения обобщенных координат и оценки скорости изменения обобщенных координат. Аналитическое решение показано на рис. 6.

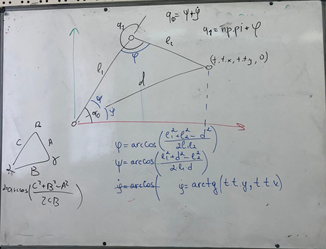


Рис. 6 Решение обратной задачи кинематики

Листинг 4 – Решение задания 2.2

def wrap\_from\_to(value, s, e):  
 r = e - s  
 return value - (r \* np.floor((value - s) / r))  
def scara\_ik(target, l):  
 d = (target.translation.x \*\* 2 + target.translation.y \*\* 2) \*\* 0.5  
 fi = np.arccos((l[1] \*\* 2 - l[2] \*\* 2 + d \*\* 2) / (2 \* l[1] \* d ))  
 psi = np.arccos((l[1] \*\* 2 + l[2] \*\* 2 - d \*\* 2) / (2 \* l[1] \* l[2]))  
 jo = np.arctan2(target.translation.x, target.translation.y)  
 q0 = Vector(1, 0, 0).angle\_to(  
 Vector(target.translation.x, target.translation.y, 0),  
 Vector(0, 0, 1)  
 ) + fi  
 q1 = np.pi + psi  
 q2 = target.angle - q0 - q1  
 q3 = l[0] - target.translation.z  
 return (  
 wrap\_from\_to(q0, -np.pi, np.pi),

wrap\_from\_to(q1, -np.pi, np.pi),  
 wrap\_from\_to(q2, -np.pi, np.pi),

q3  
 )

**//Зададим закон изменения целевого положения**

def target(t, total):

omega = t / total \* np.pi \* 2

return Target(

Vector(200, 0, 100) + 100 \* Vector(np.sin(omega), np.sin(omega), 0),

4 \* omega

)

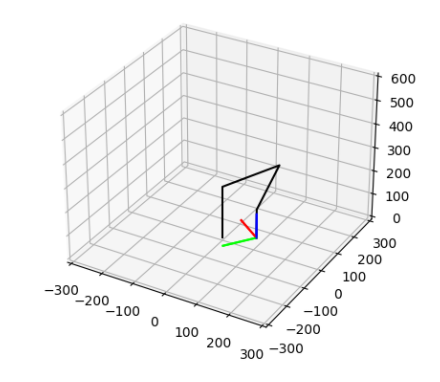


Рис. 7 Пример выполнения программы

В листинге 5 показан реализация метод оценки рабочей зоны. Графическое отображение показано на рис. 8.

Листинг 5 - Оценка рабочей зоны

size = 600

step = 25

fig, ax = graphics.figure(size \* 2)

px = []; py = []; pz = []

for x in np.arange(-size, size, step):

for y in np.arange(-size, size, step):

for z in np.arange(0, size, step):

t = Target(Vector(x, y, z), 0)

if scara\_limited\_ik(t, scara\_l) != None:

px += [x]

py += [y]

pz += [z]

ax.scatter(px, py, pz)

fig.show()

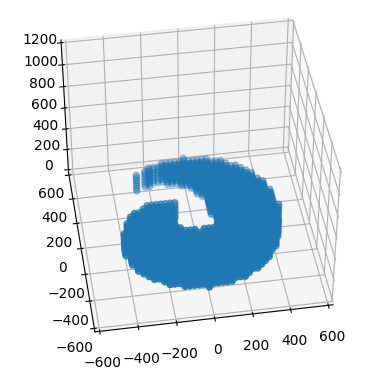


Рис. 8 Рабочая зона

Изменение обобщенных координат (рис. 9), а рабочая зона (рис. 10).

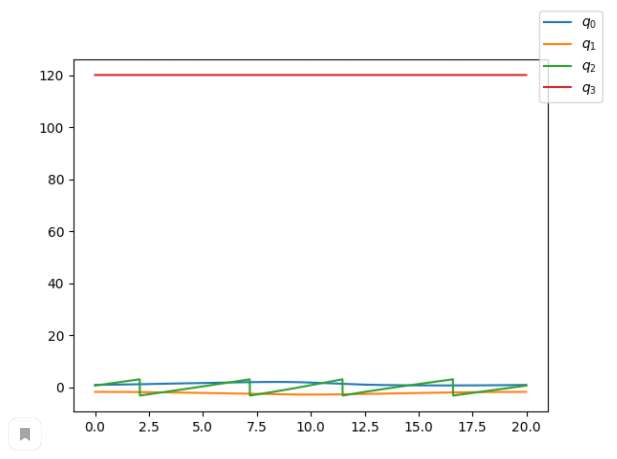


Рис. 9 График изменения обобщенных координат

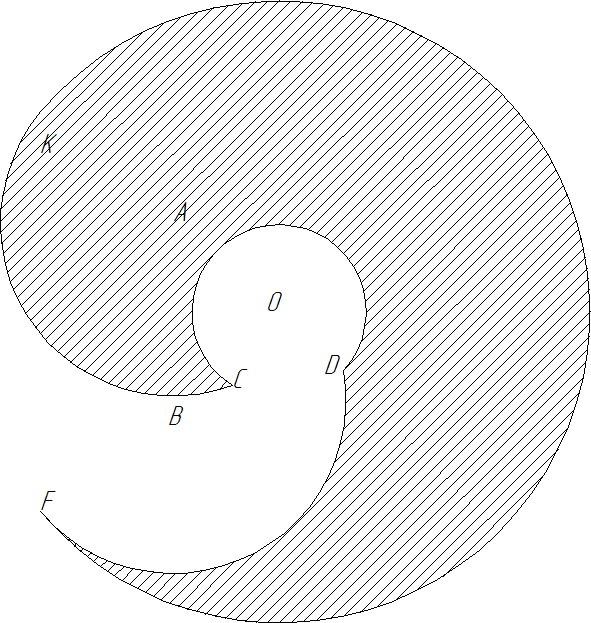


Рис. 10 Рабочая зона

**Задание №3.**

Решим обратную задачу кинематики для «кастомного» робота (листинг 6), графическое представление выполнения программы (рис. 12), оценка рабочей зоны (рис. 13). Аналитическое решение (рис. 11).

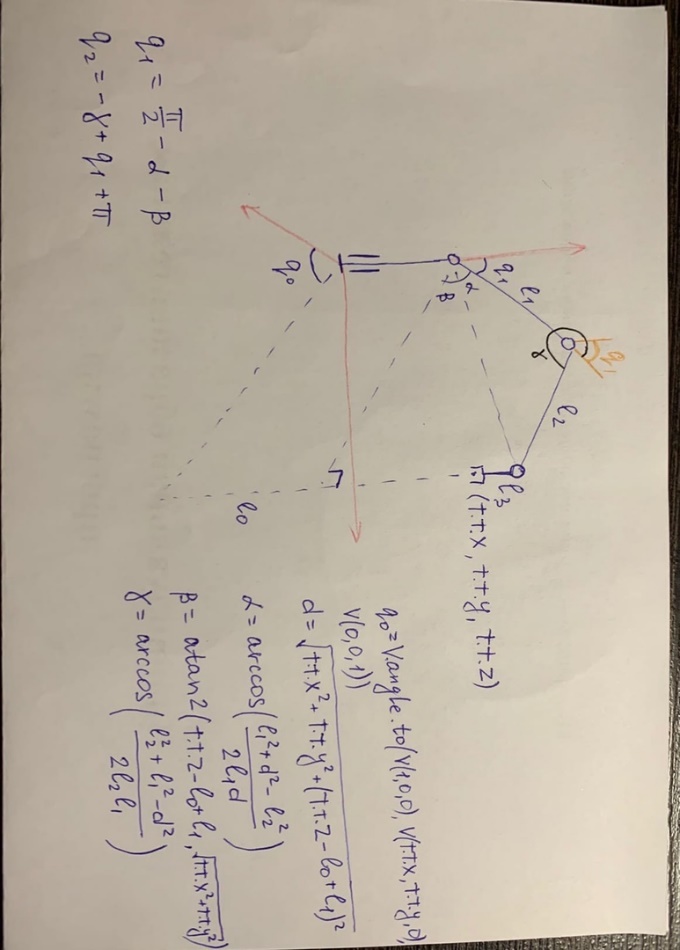


Рис. 11 Решение обратной задачи кинематики

Листинг 6 – ОЗК для кастомного робота

def wrap\_from\_to(value, s, e):

r = e - s

return value - (r \* np.floor((value - s) / r))

def custom\_ik(target, l):

d = (target.translation.x \*\* 2 + target.translation.y \*\* 2 + (target.translation.z - l[0]+l[3])\*\*2) \*\* 0.5

alfa = np.arccos((l[1] \*\* 2 + d \*\* 2 - l[2] \*\* 2) /(2 \* l[1] \* d))

beta = np.arctan2(target.translation.z - l[0] + l[3], (target.translation.x \*\* 2 + target.translation.y \*\* 2) \*\* 0.5)

psi = np.arccos((l[2] \*\* 2 + l[1] \*\* 2 - d \*\* 2) / (2 \* l[1] \* l[2]))

q0 = Vector(1, 0, 0).angle\_to(

Vector(target.translation.x, target.translation.y, 0),

Vector(0, 0, 1)

)

q1 = np.pi / 2 - alfa - beta

q2 = -psi + q1 + np.pi

q3 = -target.angle + q0

return (

wrap\_from\_to(q0, -np.pi, np.pi),

wrap\_from\_to(q1, -np.pi, np.pi),

wrap\_from\_to(q2, -np.pi, np.pi),

wrap\_from\_to (q3, -np.pi, np.pi),

)

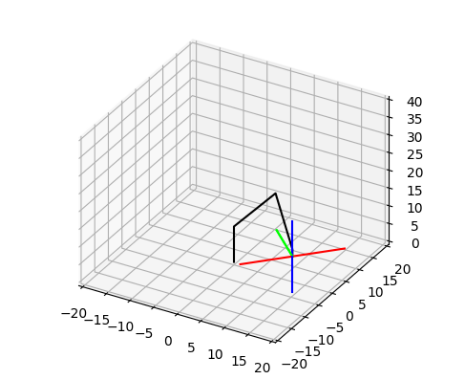


Рис. 12 Пример выполнения программы

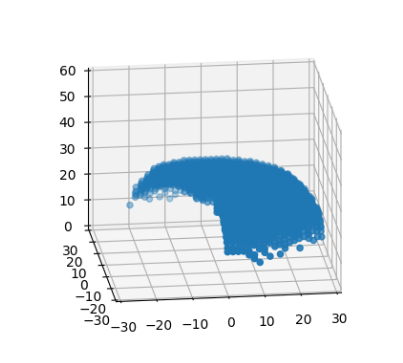


Рис. 13 Рабочая зона для данных ограничений обобщенных координат

**Вывод:** в ходе лабораторной работы мы изучил аналитическое решение обратной задачи кинематики на примере манипуляторов SCARA и PUMA

**Задание №4.**

Для робота PUMA проведем исследование для своих точек траектории, проанализируем влияние параметра blend на скорость обобщенных координат, оценим ускорения обобщенных координат (через дифференцирование второго порядка diff(. . ,2)). Пример выполнения программы (рис. 14).

#Возьмем готовые функции для моделирования кинематики

#Добавим функцию, учитывающую ограничения степеней подвижности

#Объявим функцию для описания линейной траектории

Объявим функцию для описания линейной траектории (листинг 7).

Листинг 7 – Функция описания линейной траектории

def lin(start, end, t, total):

return Transform.lerp(

start,

end,

t / total

)

s = Transform(

Vector(200, 400, 500),

Quaternion.from\_angle\_axis(np.pi / 2 , Vector(-1, 0, 0))

)

e = Transform(

Vector(200, -300, 800),

Quaternion.from\_angle\_axis(np.pi / 4 , Vector(0, 1, 0))

)

irb\_i = [1, 1, -1]

***//Рассмотрим линейной движение***

(x, y, z) = graphics.chain\_to\_points(

irb\_chain([0, 0, 0, 0, 0, 0], irb\_l)

)

fig, ax = graphics.figure(1000)

lines, = ax.plot(x, y, z, color="#000000")

graphics.axis(ax, s, 100)

graphics.axis(ax, e, 100)

total = 100

def animate(frame):

trs = lin(s, e, frame, total)

q = irb\_ik\_lim(

trs,

irb\_l,

irb\_i

)

if q != None:

chain = irb\_chain(q, irb\_l)

(x, y, z) = graphics.chain\_to\_points(chain)

lines.set\_data\_3d(x, y, z)

animate(0)

fps = 25

irb\_ani = animation.FuncAnimation(

fig,

animate,

frames=total,

interval=1000.0/fps

)

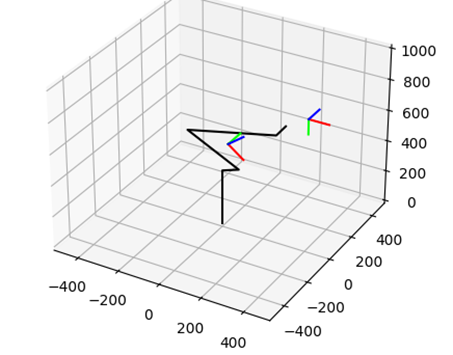


Рис. 14 Пример выполнения программы

Повторим движение в режиме переброски (листинг 8). Работа программы отображена на рис. 15.

Листинг 8 – Движение в режиме переброски

(x, y, z) = graphics.chain\_to\_points (

irb\_chain([0, 0, 0, 0, 0, 0], irb\_l)

)

fig, ax = graphics.figure(1000)

lines, = ax.plot(x, y, z, color="#000000")

graphics.axis(ax, s, 100)

graphics.axis(ax, e, 100)

total = 100

s\_q = irb\_ik\_lim(s, irb\_l, irb\_i)

e\_q = irb\_ik\_lim(e, irb\_l, irb\_i)

def animate(frame):

q = []

for index in range(len(s\_q)):

t = frame / total

q += [s\_q[index] + t \* (e\_q[index] - s\_q[index])]

chain = irb\_chain(q, irb\_l)

(x, y, z) = graphics.chain\_to\_points(chain)

lines.set\_data\_3d(x, y, z)

animate(0)

fps = 25

irb\_ani = animation.FuncAnimation(

fig,

animate,

frames=total,

interval=1000.0/fps

)

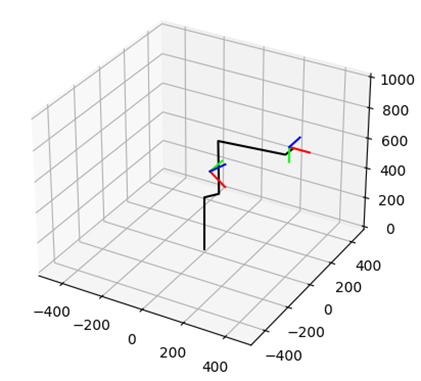


Рис. 15 Пример работы программы

Построим график изменения обобщенных координат для переброски (листинг 9). График показан на рис.16.

Листинг 9 – Реализация графика изменения обобщенных координат

total = 20

step = 0.01

t = np.arange(0, total, step)

fig = plt.figure()

ax = fig.add\_subplot()

s\_q = irb\_ik\_lim(s, irb\_l, irb\_i)

e\_q = irb\_ik\_lim(e, irb\_l, irb\_i)

q = []

for index in range(6):

q += [s\_q[index] + t / total \* (e\_q[index] - s\_q[index])]

ax.plot(t, q[0], label="$q\_0$")

ax.plot(t, q[1], label="$q\_1$")

ax.plot(t, q[2], label="$q\_2$")

ax.plot(t, q[3], label="$q\_3$")

ax.plot(t, q[4], label="$q\_4$")

ax.plot(t, q[5], label="$q\_5$")

fig.legend()

fig.show()

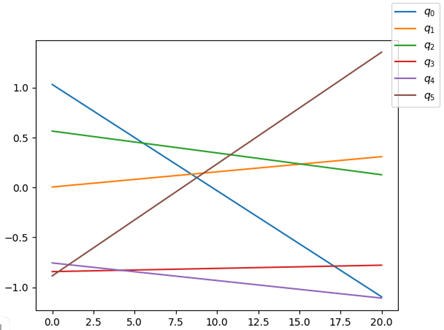


Рис. 16 График изменения обобщенных координат

Построим график изменения обобщенных координат для линейного движения (листинг 10). График показан на рис. 17.

Листинг 10 – График изменения обобщенных координат линейного движения

v\_lin = np.vectorize(lin, excluded={0, 1, 3})

v\_irb\_ik = np.vectorize(irb\_ik\_lim, excluded={1, 2})

total = 20

step = 0.01

t = np.arange(0, total, step)

fig = plt.figure()

ax = fig.add\_subplot()

q = v\_irb\_ik( v\_lin(s, e, t, total), irb\_l, irb\_i );

ax.plot(t, q[0], label="$q\_0$")

ax.plot(t, q[1], label="$q\_1$")

ax.plot(t, q[2], label="$q\_2$")

ax.plot(t, q[3], label="$q\_3$")

ax.plot(t, q[4], label="$q\_4$")

ax.plot(t, q[5], label="$q\_5$")

fig.legend()

fig.show()

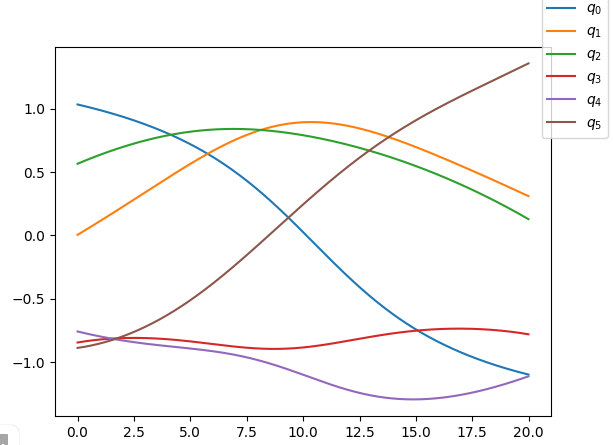


Рис. 17 График изменения обобщенных координат

Объединим оба движения (листинг 11). Выполнение программы рис. 18.

Листинг 11 – Объединение двух линейных движений

def lin\_lin(start, inter, end, t, total):

progress = t / total

if progress < 0.5:

return Transform.lerp(

start,

inter,

progress \* 2

)

else:

return Transform.lerp(

inter,

end,

(progress - 0.5) \* 2

)

***//Добавим промежуточную точку и рассмотрим два линейных движения***

i = Transform(

Vector(400, 100, 900),

Quaternion.from\_angle\_axis(np.pi / 4, Vector(0, 1, 0))

)

(x, y, z) = graphics.chain\_to\_points(

irb\_chain([0, 0, 0, 0, 0, 0], irb\_l)

)

fig, ax = graphics.figure(1000)

lines, = ax.plot(x, y, z, color="#000000")

graphics.axis(ax, s, 100)

graphics.axis(ax, i, 100)

graphics.axis(ax, e, 100)

total = 100

def animate(frame):

trs = lin\_lin(s, i, e, frame, total)

q = irb\_ik\_lim(

trs,

irb\_l,

irb\_i

)

if q != None:

chain = irb\_chain(q, irb\_l)

(x, y, z) = graphics.chain\_to\_points(chain)

lines.set\_data\_3d(x, y, z)

animate(0)

fps = 25

irb\_ani = animation.FuncAnimation(

fig,

animate,

frames=total,

interval=1000.0/fps

)

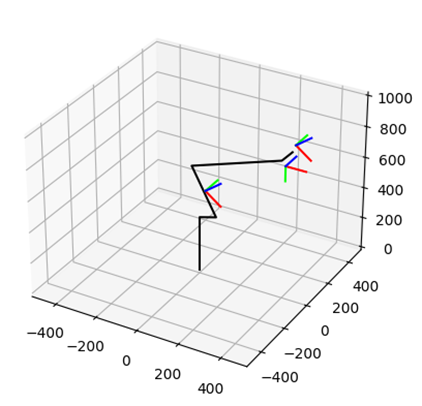


Рис. 18 Пример выполнения программы

Далее проанализируем скорости обобщенных координат (листинг 12). График показан на рис. 19.

Листинг 12 – Анализ скорости обобщенных координат

v\_lin\_lin = np.vectorize(lin\_lin, excluded={0, 1, 2, 4})

v\_irb\_ik = np.vectorize(irb\_ik\_lim, excluded={1, 2})

total = 20

step = 0.01

t = np.arange(0, total, step)

fig = plt.figure()

ax = fig.add\_subplot()

w = np.diff(v\_irb\_ik(

v\_lin\_lin(s, i, e, t, total),

irb\_l,

irb\_i

)) / step;

ax.plot(t[:-1], w[0], label="$\omega\_0$")

ax.plot(t[:-1], w[1], label="$\omega\_1$")

ax.plot(t[:-1], w[2], label="$\omega\_2$")

ax.plot(t[:-1], w[3], label="$\omega\_3$")

ax.plot(t[:-1], w[4], label="$\omega\_4$")

ax.plot(t[:-1], w[5], label="$\omega\_5$")

fig.legend()

fig.show()

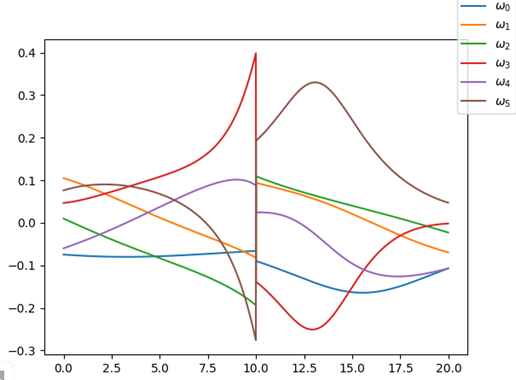


Рис. 19 График изменения скорости обоб.координат

Напишем функцию для объединения двух линейных движений со сглаживанием (листинг 13), выполнение программы показано на рис. 20.

Листинг 13 – Объединение линейных движений со сглаживанием

def bezier\_transform(a, b, c, t):

return Transform.lerp(

Transform.lerp(a, b, t),

Transform.lerp(b, c, t),

t

)

def lin\_lin\_smooth(start, inter, end, t, total, blend=0.1):

progress = t / total

if np.abs(progress - 0.5) < blend:

progress = (progress - 0.5 + blend) / 2 / blend

a = lin(start, inter, 1.0 - 2 \* blend, 1)

b = inter

c = lin(inter, end, 2 \* blend, 1)

return bezier\_transform(

a,

b,

c,

progress

)

else:

return lin\_lin(start, inter, end, t, total)

blending = 0.9

(x, y, z) = graphics.chain\_to\_points(

irb\_chain([0, 0, 0, 0, 0, 0], irb\_l)

)

fig, ax = graphics.figure(1000)

lines, = ax.plot(x, y, z, color="#000000")

graphics.axis(ax, s, 100)

graphics.axis(ax, i, 100)

graphics.axis(ax, e, 100)

total = 100

def animate(frame):

trs = lin\_lin\_smooth(s, i, e, frame, total, 0.1)

q = irb\_ik\_lim(

trs,

irb\_l,

irb\_i

)

if q != None:

chain = irb\_chain(q, irb\_l)

(x, y, z) = graphics.chain\_to\_points(chain)

lines.set\_data\_3d(x, y, z)

animate(0)

fps = 25

irb\_ani = animation.FuncAnimation(

fig,

animate,

frames=total,

interval=1000.0/fps

)

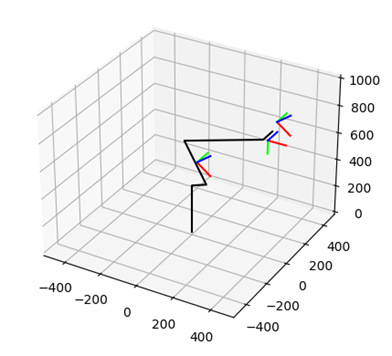


Рис. 20 Пример выполнения программы

Проанализируем скорости обобщенных координат (листинг 14). Графики показаны на рис. 21 и 22.

Листинг 14 – Анализ скорости обобщенных координат

v\_lin\_lin = np.vectorize(lin\_lin\_smooth, excluded={0, 1, 2, 4, 5})

v\_irb\_ik = np.vectorize(irb\_ik\_lim, excluded={1, 2})

total = 20

step = 0.01

t = np.arange(0, total, step)

fig = plt.figure()

ax = fig.add\_subplot()

w = np.diff(v\_irb\_ik(

v\_lin\_lin(s, i, e, t, total, blending),

irb\_l,

irb\_i

)) / step;

ax.plot(t[:-1], w[0], label="$\omega\_0$")

ax.plot(t[:-1], w[1], label="$\omega\_1$")

ax.plot(t[:-1], w[2], label="$\omega\_2$")

ax.plot(t[:-1], w[3], label="$\omega\_3$")

ax.plot(t[:-1], w[4], label="$\omega\_4$")

ax.plot(t[:-1], w[5], label="$\omega\_5$")

fig.legend()

fig.show()

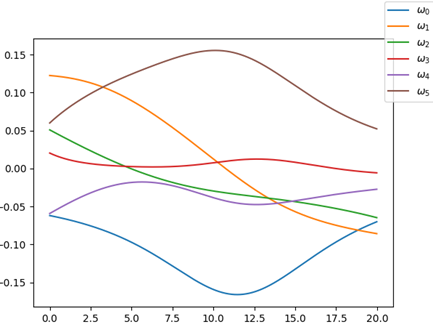
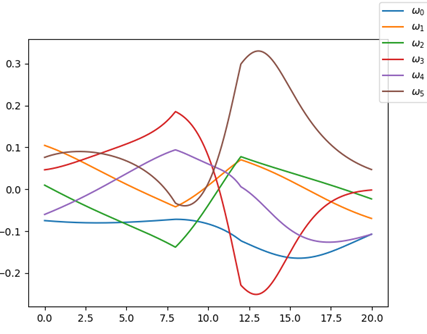


Рис. 21 Скорости при blending = 0.1 Рис. 22 Скорости при blending = 0.9

Проанализируем ускорения обобщенных координат (листинг 15), график (рис. 23).

Листинг 15 – Анализ ускорений обобщенных координат

v\_lin\_lin = np.vectorize(lin\_lin\_smooth, excluded={0, 1, 2, 4, 5})

v\_irb\_ik = np.vectorize(irb\_ik\_lim, excluded={1, 2})

total = 20

step = 0.01

t = np.arange(0, total, step)

fig = plt.figure()

ax = fig.add\_subplot()

a = np.diff(v\_irb\_ik(

v\_lin\_lin(s, i, e, t, total, blending),

irb\_l,

irb\_i

),2) / (step\*0.001);

ax.plot(t[:-2], a[0], label="$a\_0$")

ax.plot(t[:-2], a[1], label="$a\_1$")

ax.plot(t[:-2], a[2], label="$a\_2$")

ax.plot(t[:-2], a[3], label="$a\_3$")

ax.plot(t[:-2], a[4], label="$a\_4$")

ax.plot(t[:-2], a[5], label="$a\_5$")

fig.legend()

fig.show()

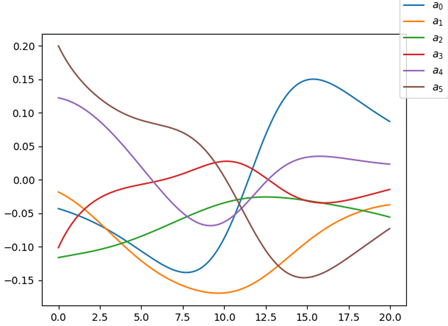


Рис. 23 График изменения ускорений обобщенных координат

Если сравнить графики с рисунков, то видно, что при увеличении значения параметра сглаживания с blending=0.1 до blending=0.9 у нас вовсе пропали какие-либо скачки, процесс изменения скоростей и ускорений стал абсолютно плавным.

**Задание №5**

Для робота SCARA проведем исследование для своих точек траектории, проанализируем влияние параметра blend на скорость обобщенных координат, оценим ускорения обобщенных координат (через дифференцирование второго порядка diff(. . ,2)).

Все действия производятся аналогично заданию №4. Результаты представлены на рис. 24-30.

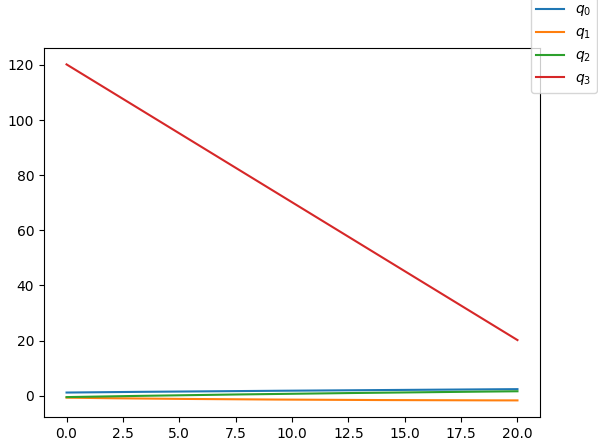


Рис. 24 График изменения обобщенных координат для линейного движения

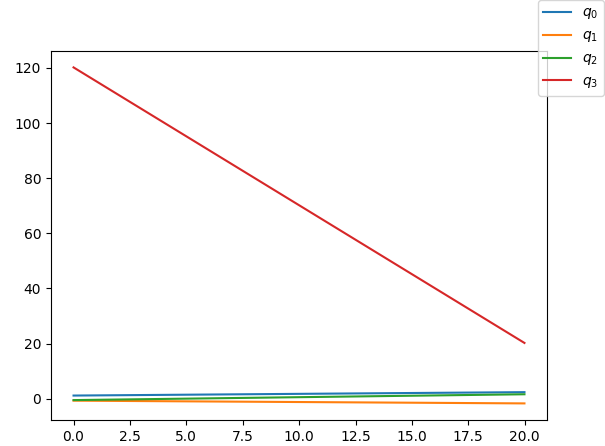


Рис. 25 График изменения обобщенных координат для переброски

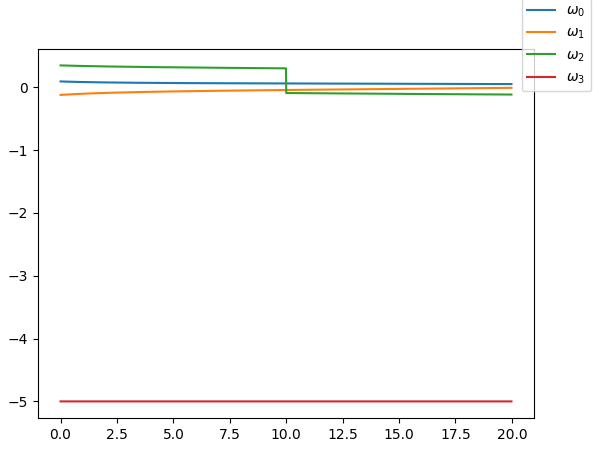


Рис. 26 График изменения скорости обобщенных координат

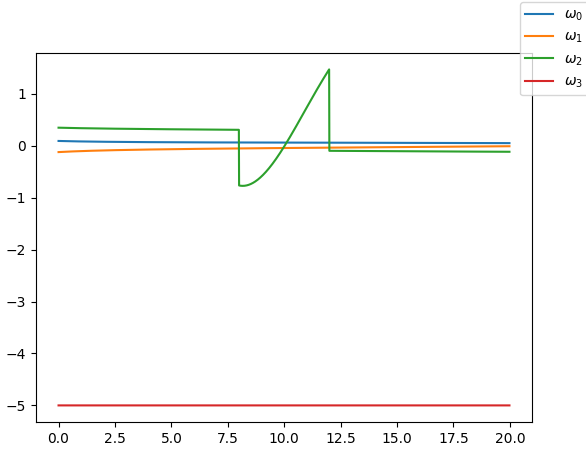
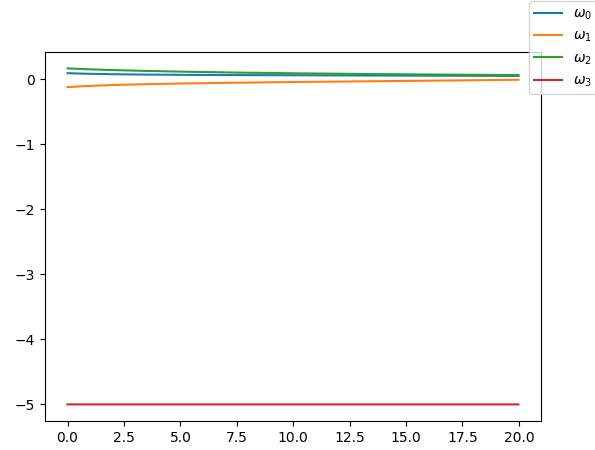
 

Рис. 27 Изменение скорости при blending = 0. Рис. 28 Изменение скорости при blending = 0.9

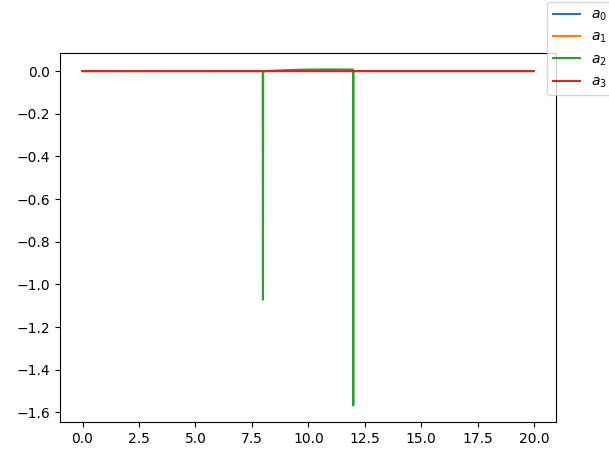
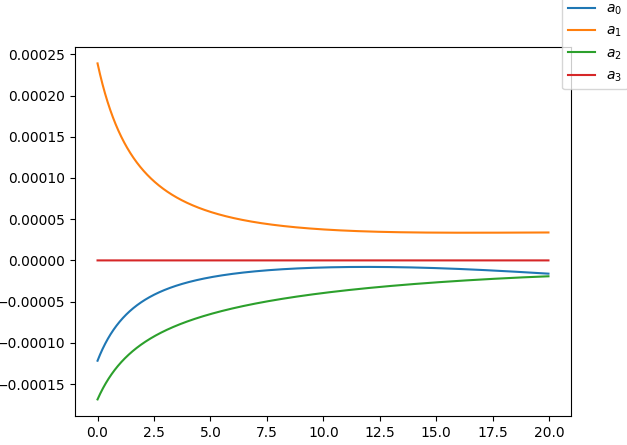
 

Рис. 29 Изменение ускорения при blending = 0.1 Рис. 30 Изменение ускорения при blending = 0.9

Если сравнить графики с рисунков, то видно, что при увеличении значения параметра сглаживания с blending=0.1 до blending=0.9 у нас вовсе пропали какие-либо скачки, процесс изменения скоростей и ускорений стал абсолютно плавным.

**Вывод:** в ходе лабораторной работы был изучен математический аппарат, применяемый для решения задач траекторного управления.