

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

Учебный курс «Моделирование и исследование робототехнических систем»

ОТЧЁТ по лабораторной работе №2 на тему:

«Изучение решения обратной задачи кинематики»

Выполнил:

студент группы АДБ-17-11 \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ Абдулзагиров М.М.

(дата) (подпись) (ФИО)

Принял

преподаватель: \_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ Прохоренко Л.С.

(дата) (подпись) (ФИО)

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

Москва 2021

Цель работы: Изучить аналитическое решение обратной задачи кинематики на примере манипуляторов SCARA и PUMA.

# Обратная задача кинематики PUMA



Рисунок 1. Робот SCARA.

Листинг 1.

from matplotlib import pyplot as plt from matplotlib import animation import numpy as np from IPython.display import HTML

%matplotlib notebook

from kinematics import Vector, Quaternion, Transform import graphics

irb\_l = [352.0, 70.0, 350.0, 380.0, 65.0] irb\_lim = [ (-180, 180),

(-90, 110),

(-230, 50),

(-200, 200),

(-115, 115),

(-400, 400)

]

#ПЗК 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

]

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

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

#ОЗК 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(200,50+ 1000 \* t / total, 500) if t / total < 0.5 else Vec

tor(200 + (t / total - 0.5) \* 500, 550, 500), Quaternion.from\_angle\_axis( t / total \* np.pi/2 ,

Vector(0, 1, 0)

)

)

# флаги конфиругации irb\_i = [1, 1, 1]

# Вывод анимации

(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") rt, gt, bt = graphics.axis(ax, Transform.identity(), 1) rf, gf, bf = graphics.axis(ax, Transform.identity(), 1)

total = 100

def animate(frame):

t = target(frame, total) q = irb\_ik( t, irb\_l, irb\_i )

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

(x, y, z) = graphics.chain\_to\_points(chain) lines.set\_data\_3d(x, y, z) global rt, gt, bt, rf, gf, bf rt.remove(); gt.remove(); bt.remove(); rf.remove(); gf.remove(); bf.r emove() rt, gt, bt = graphics.axis(ax, t, 100) rf, gf, bf = graphics.axis(ax, chain[-1], 100)

animate(0) fps = 25 irb\_ani = animation.FuncAnimation( fig, animate, frames=total, interval=1000.0/fps

)

HTML(irb\_ani.to\_jshtml())

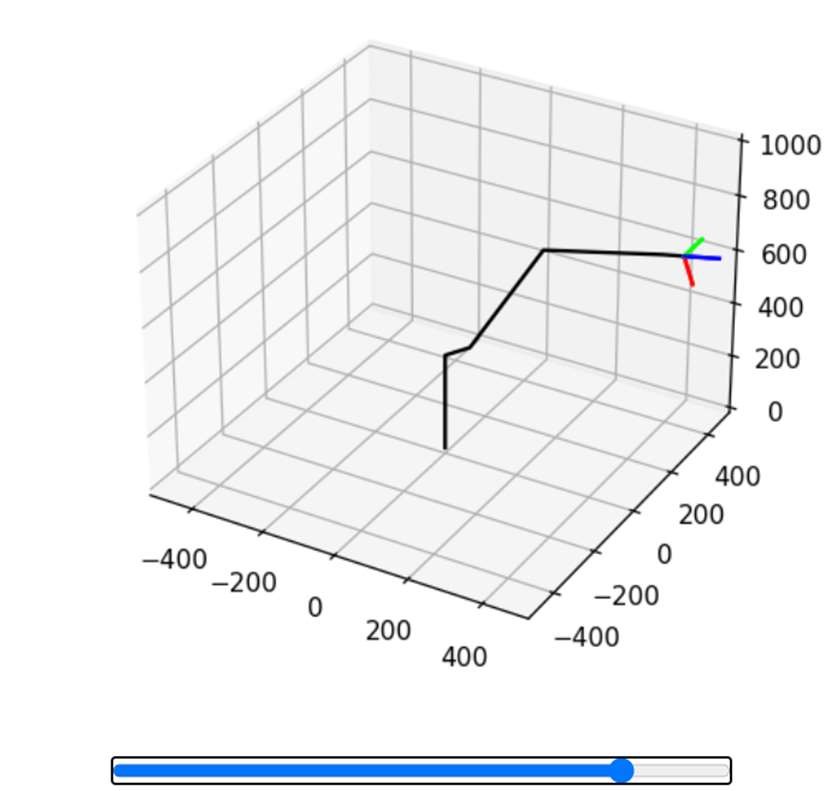
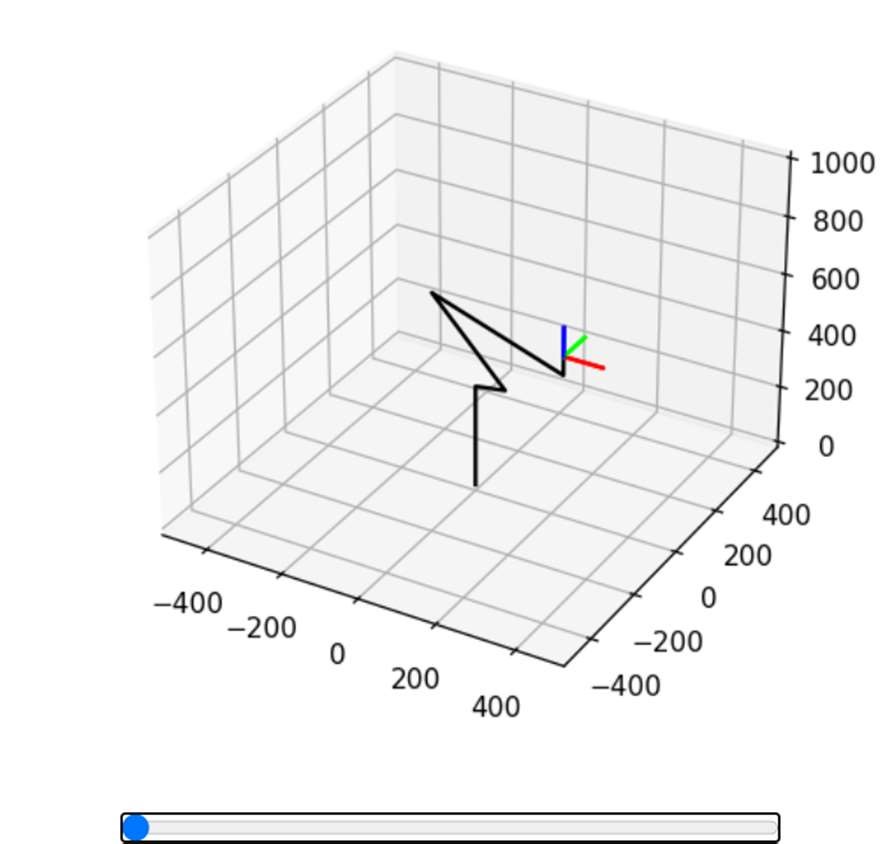


Рисунок 2. Результат работы программы.

Построение графиков обобщённых координат и их скоростей:

## Листинг 2

#углы

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()

#скороссть

ax.plot(t[:-1], np.diff(q[0]), label="$vq\_0$") ax.plot(t[:-1], np.diff(q[1]), label="$vq\_1$") ax.plot(t[:-1], np.diff(q[2]), label="$vq\_2$") ax.plot(t[:-1], np.diff(q[3]), label="$vq\_3$") ax.plot(t[:-1], np.diff(q[4]), label="$vq\_4$") ax.plot(t[:-1], np.diff(q[5]), label="$vq\_5$") fig.legend() fig.show()

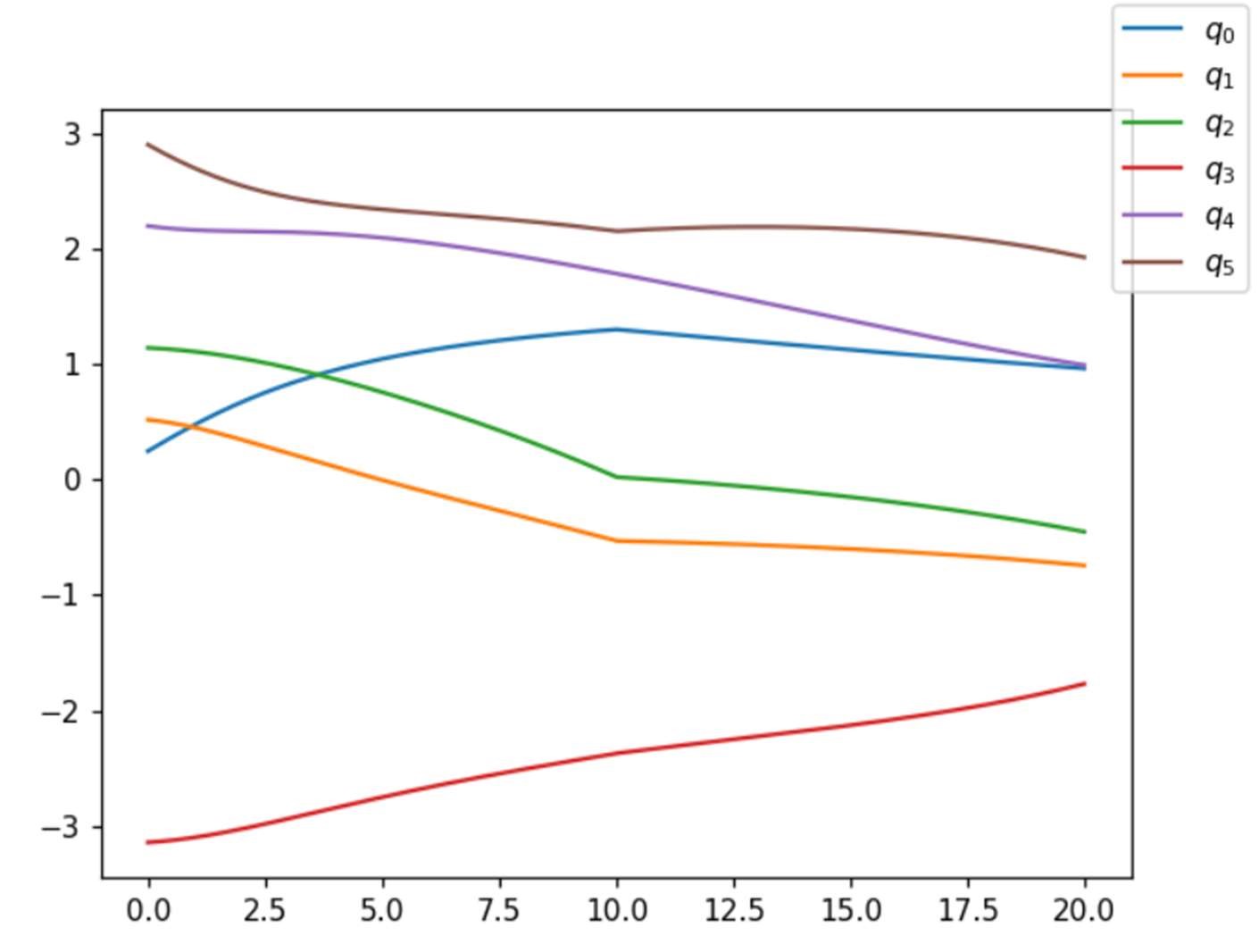


Рисунок 3. График изменения обобщённых координат.

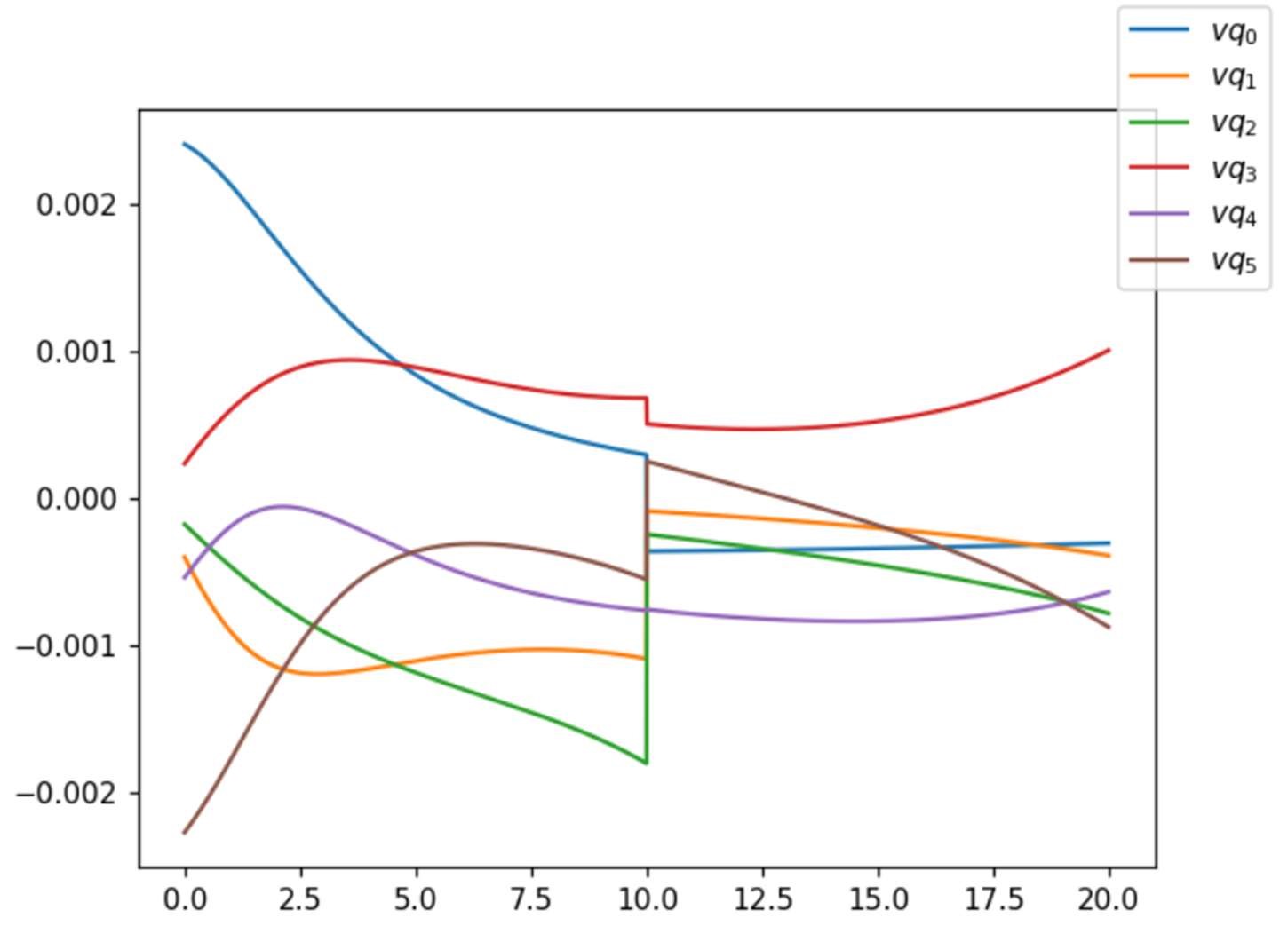


Рисунок 4. График изменения скоростей.

# Обратная задача кинематики для SCARA

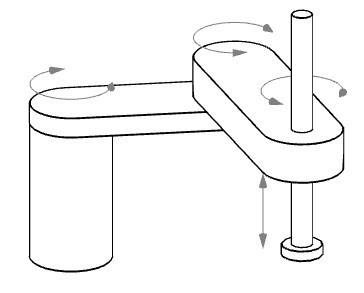


Рисунок 5. Робот SCARA.

## Листинг 3

from matplotlib import pyplot as plt from matplotlib import animation import numpy as np from IPython.display import HTML

%matplotlib notebook from kinematics import Vector, Quaternion, Transform import graphics scara\_l = [220.2, 200, 250] scara\_lim = [ (-140, 140),

(-150, 150),

(-400, 400),

(0, 180)

]

#ПЗК def scara\_chain(q, l):

base = Transform.identity() column = base + Transform(

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

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

)

elbow = column + Transform(

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

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

) tool = elbow + Transform(

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

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

)

flange = tool + Transform( Vector(0, 0, -q[3]),

Quaternion.identity()

) return [ base, column, elbow, tool, flange

]

# Класс для описания целевого положения class Target:

def \_\_init\_\_(self, translation, angle):

super(Target, self).\_\_init\_\_() self.translation = translation # вектор self.angle = angle # угол поворота вокруг вертикальной оси

def to\_transform(self): return Transform( self.translation, Quaternion.from\_angle\_axis( self.angle,

Vector(0, 0, 1)

)

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

r = e - s

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

# ОЗК def scara\_ik(target, l): d = (target.translation.x \*\* 2 + target.translation.y \*\* 2) \*\* 0.5 q1 = np.pi -np.arccos(

(l[2] \*\* 2 + l[1] \*\* 2 - d \*\* 2) /\

(2 \* l[2] \* l[1])

)

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

(2 \* l[1] \* d)

)

lift\_angle = np.arctan2( target.translation.y, target.translation.x

) q0 = -triangle\_angle + lift\_angle 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

)

# закон изменения целевого положения def target(t, total): omega = t / total \* np.pi \* 2 return Target(

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

4 \* omega

)

# вывод анимации

(x, y, z) = graphics.chain\_to\_points( scara\_chain([0, 0, 0, 0], scara\_l)

) fig, ax = graphics.figure(600) lines, = ax.plot(x, y, z, color="#000000") rt, gt, bt = graphics.axis(ax, Transform.identity(), 1) rf, gf, bf = graphics.axis(ax, Transform.identity(), 1)

total = 100

def animate(frame): t = target(frame, total) q = scara\_ik( t, scara\_l

)

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

(x, y, z) = graphics.chain\_to\_points(chain) lines.set\_data\_3d(x, y, z) global rt, gt, bt, rf, gf, bf rt.remove(); gt.remove(); bt.remove(); rf.remove(); gf.remove(); bf.remov e() rt, gt, bt = graphics.axis(ax, t.to\_transform(), 100) rf, gf, bf = graphics.axis(ax, chain[-1], 100)

animate(0) fps = 25

scara\_ani = animation.FuncAnimation( fig, animate,

frames=total,

interval=1000.0/fps

)

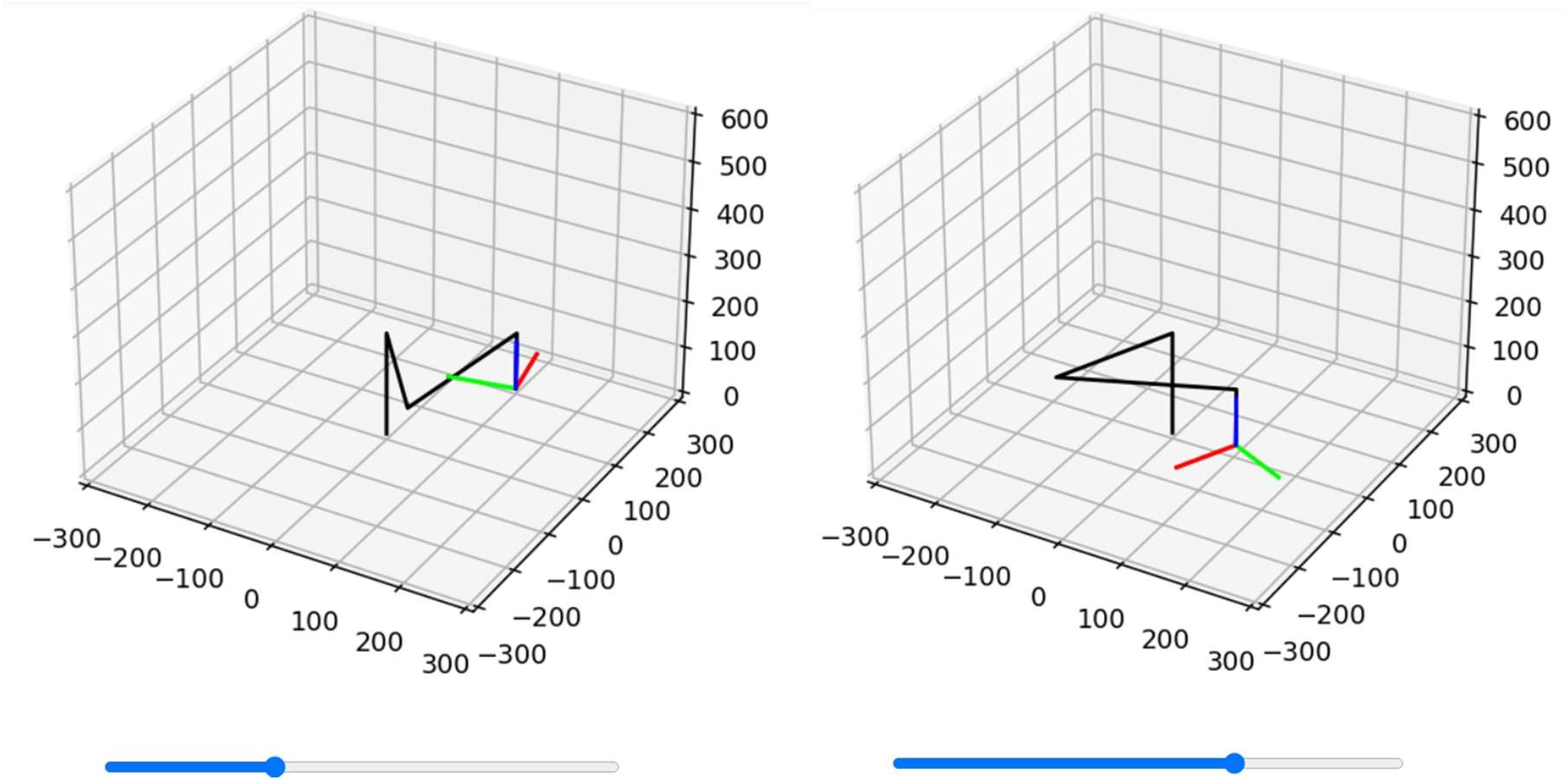


Рисунок 6. Результат работы программы.

## Листинг 4

# построение граффика

v\_target = np.vectorize(target, excluded={1}) v\_irb\_ik = np.vectorize(scara\_ik, excluded={1, 2}) total = 100 step = 0.01 t = np.arange(0, total, step)

fig = plt.figure() ax = fig.add\_subplot() q = v\_irb\_ik( v\_target(t, total), scara\_l );

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]/20, label="$q\_3/20$") fig.legend()

fig.show()

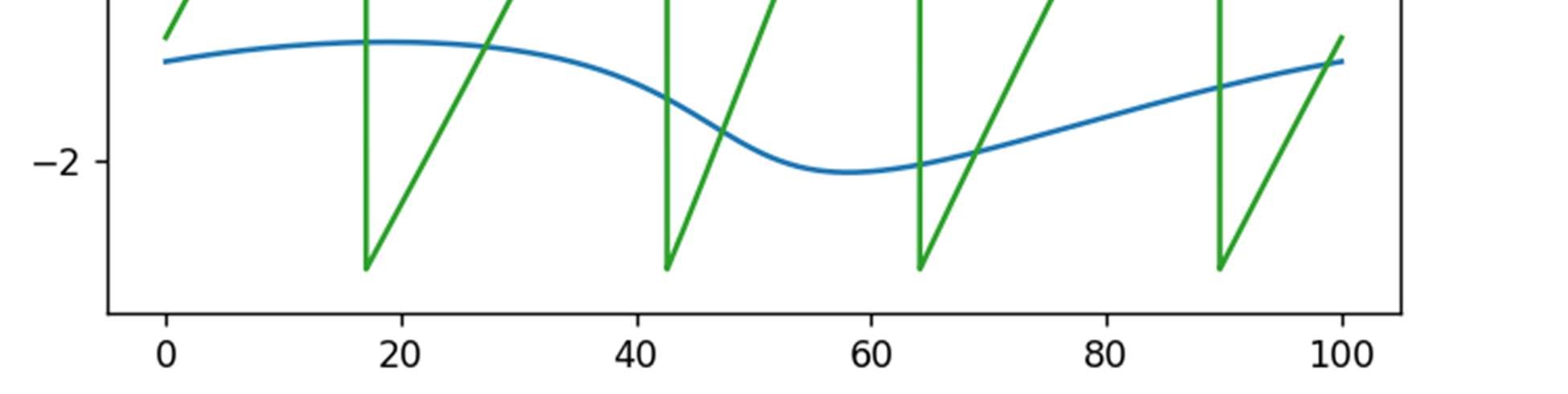
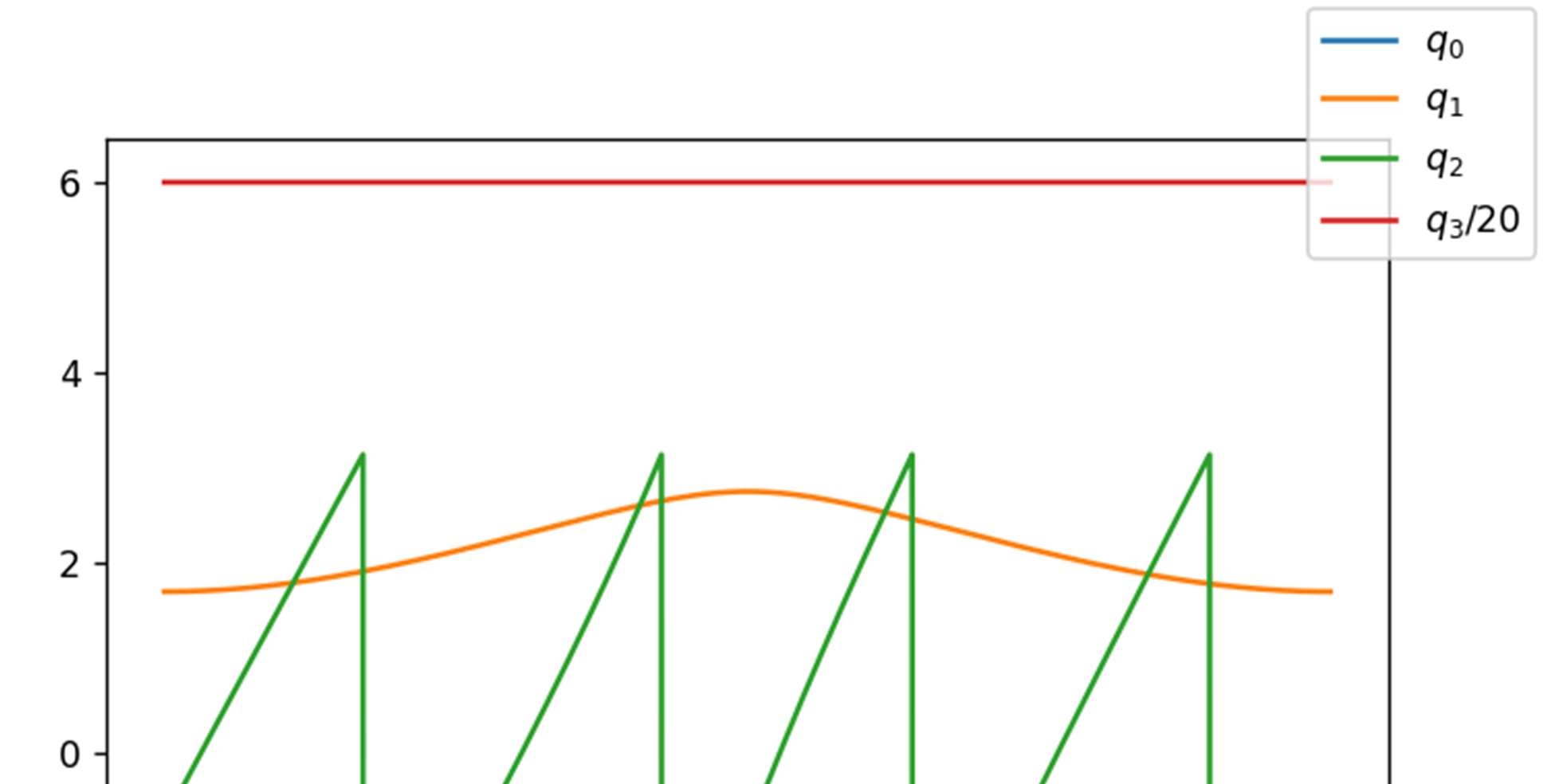


Рисунок 7. График изменения обобщённых координат.

Изменим изменения обобщённых координат:

## Листинг 5

def target(t, total): omega = t / total \* np.pi \* 2

return Target(

Vector(200,30, 100+ 50\*np.sin(omega)) + 100 \* Vector(np.cos(omega

)/2, np.sin(omega)\*2, 0), np.pi

)

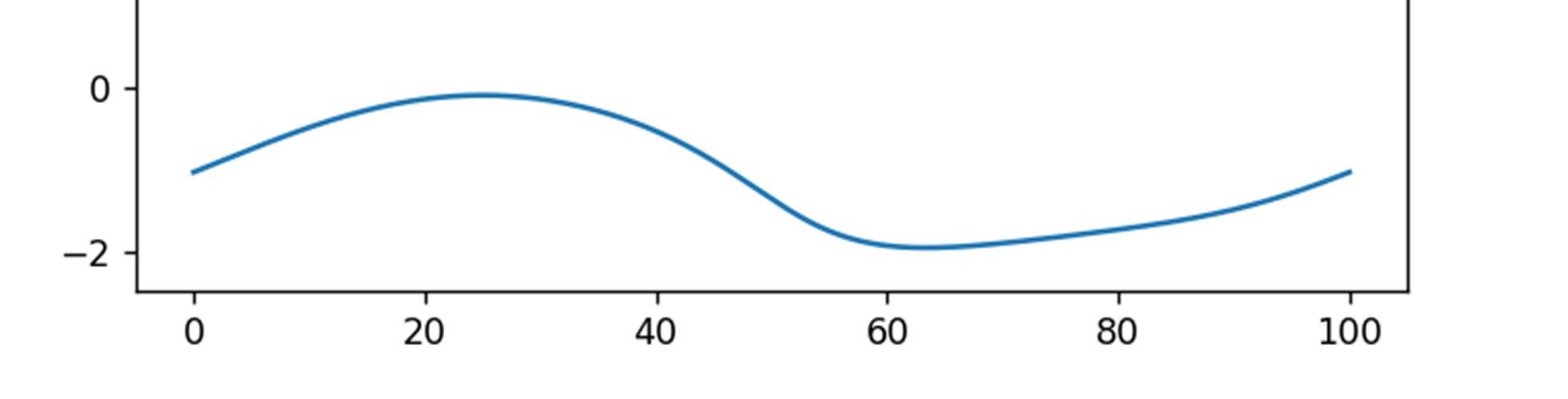
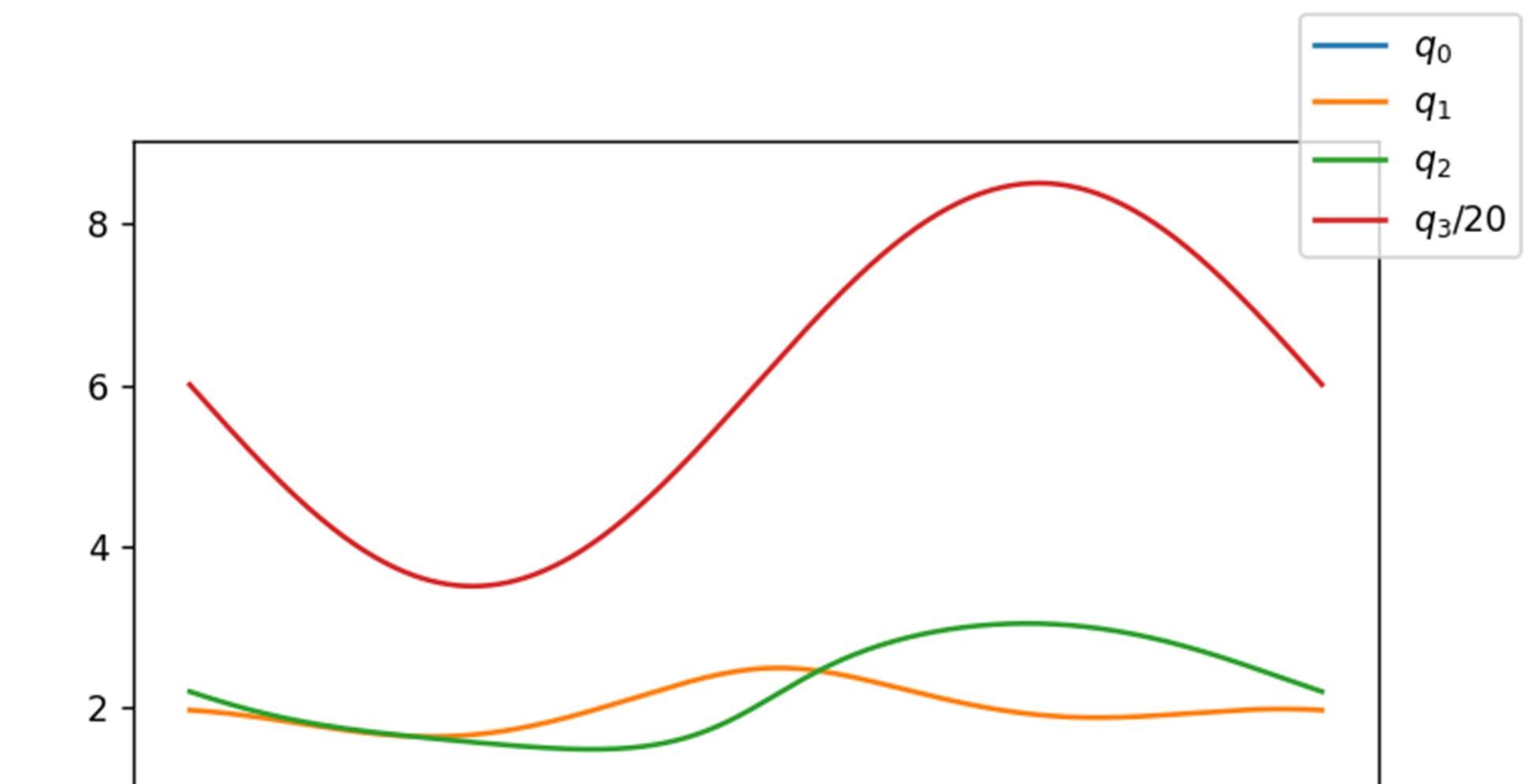


Рисунок 8. График изменения обобщённых координат.

Оценим рабочую зону

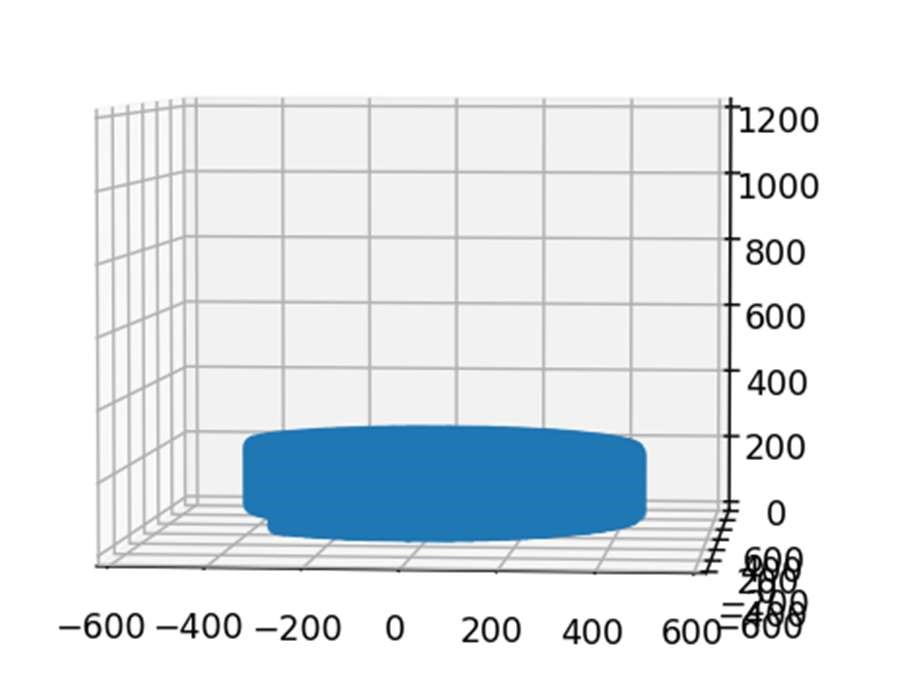
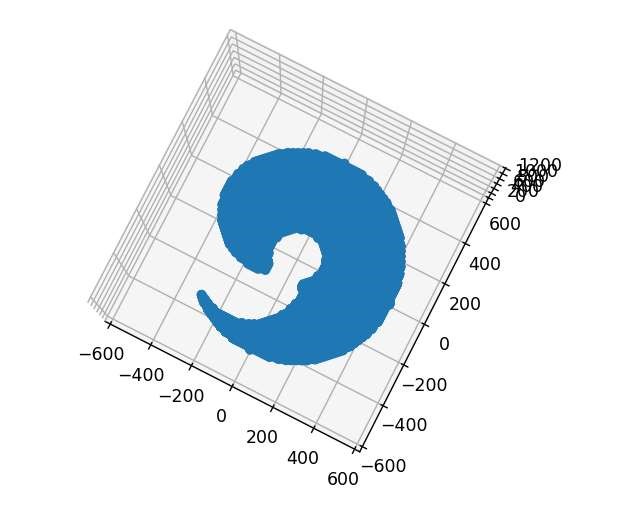


Рисунок 9. Рабочая зона SCARA.

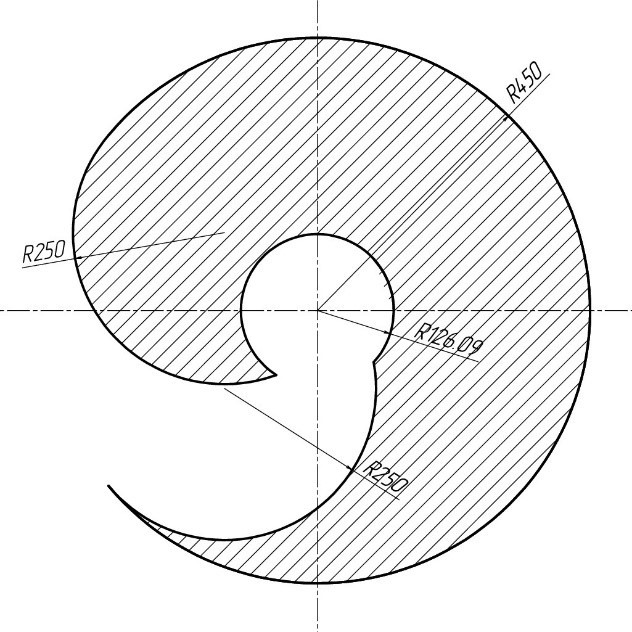


Рисунок 10. Рабочая зона SCARA.

Вывод: в данной лабораторной работе мы изучили аналитическое решение обратной задачи кинематики на примере манипуляторов SCARA и PUMA.