

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 №3 на тему:

«Изучение методов траекторного управления»

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 математический аппарат, применяемый для решения задач траекторного управления.

# Управление роботом 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]

#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

]

#ограничитель 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)

) irb\_lim = [ (-180, 180),

(-90, 110),

(-230, 50),

(-200, 200),

(-115, 115),

(-400, 400)

]

#возвращает None если невозможно достичь точки def irb\_ik\_lim(target, l, i=[1, 1, 1]): solution = irb\_ik(target, l, i) for index in range(len(solution)):

if solution[index] < np.deg2rad(irb\_lim[index][0]) or\ solution[index] > np.deg2rad(irb\_lim[index][1]) or\ np.isnan(solution[index]):

return None return solution

#интерполяция def lin(start, end, t, total): return Transform.lerp( start, end, t / total

)

s = Transform(

Vector(200, 400, 600),

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

)

e = Transform(

Vector(200, -300, 800),

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

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

Реализация линейного движения:

#LIN

(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

)

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

#углы

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()

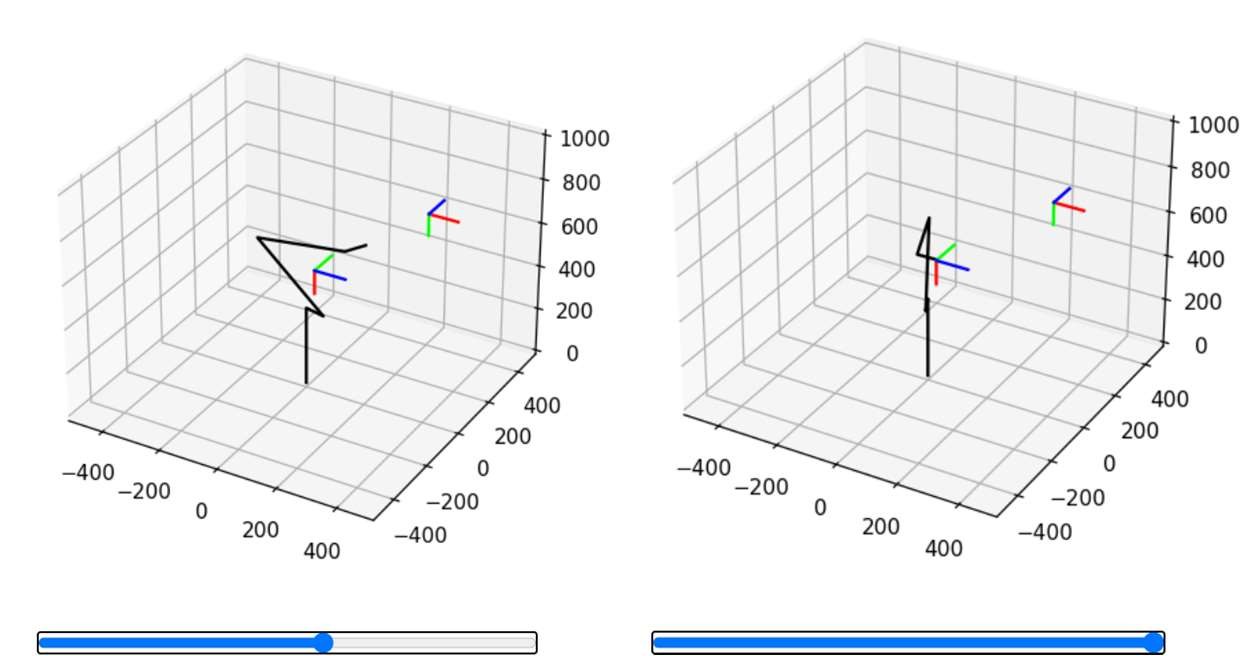
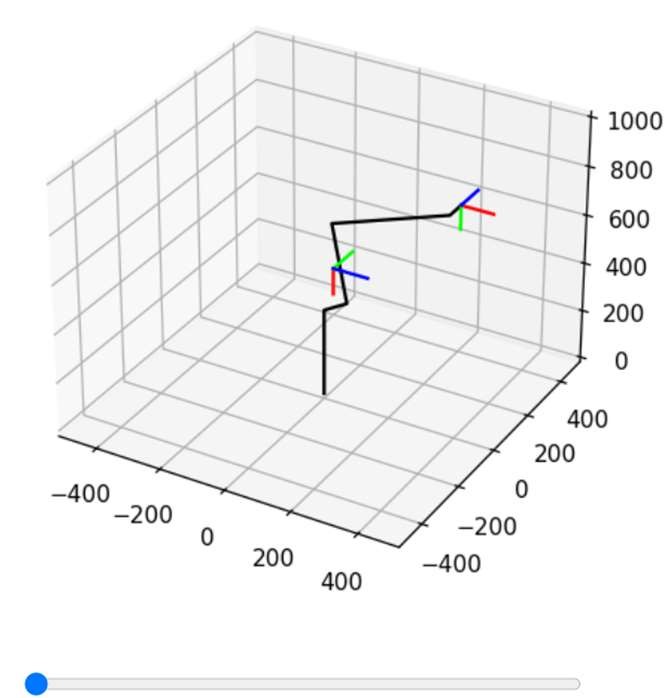


Рисунок 2. Линейное движение.

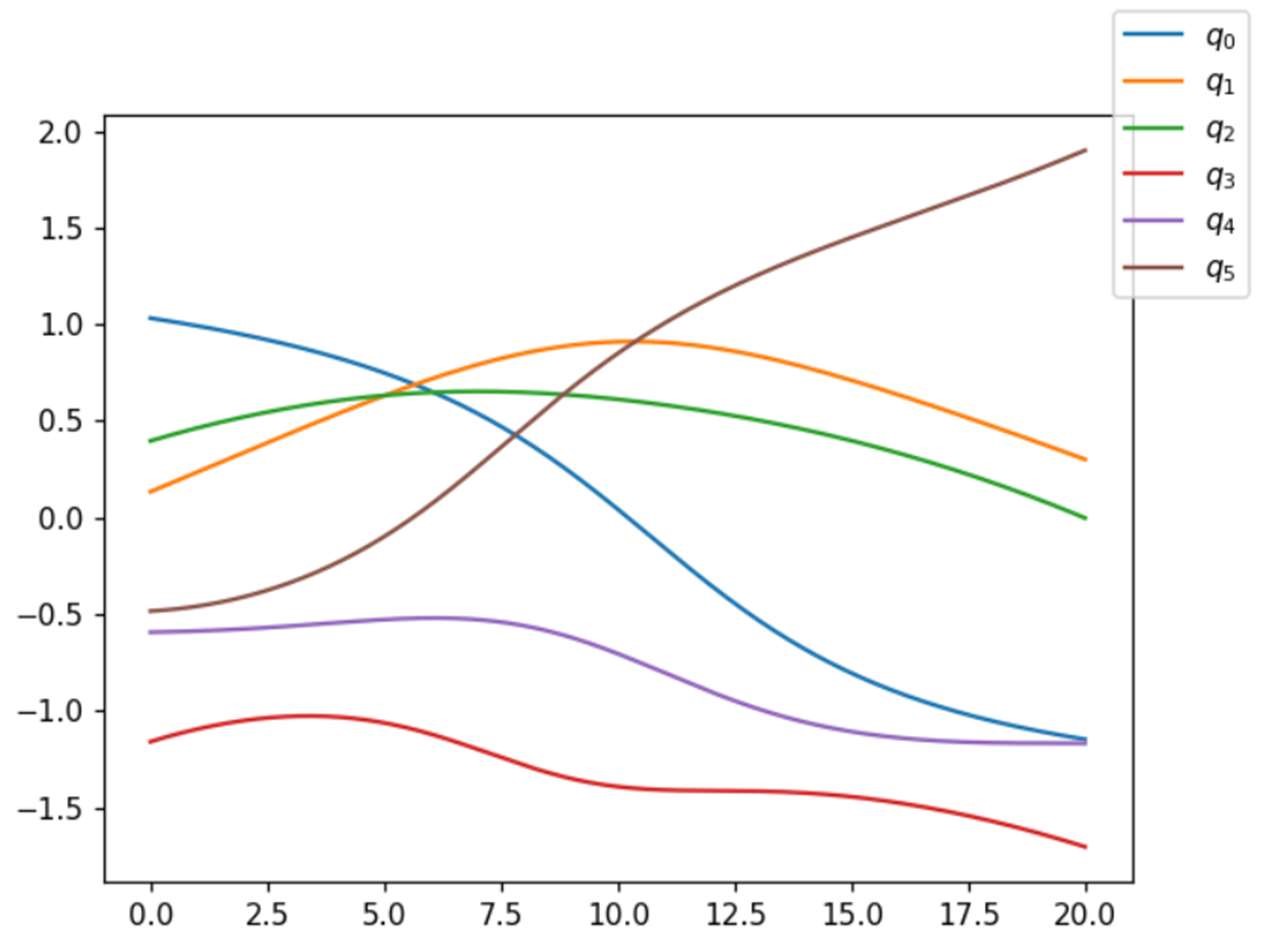


Рисунок 3. График изменения обобщённых координат.

Реализация движения в режиме переброски:

#PTP

(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

)

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()

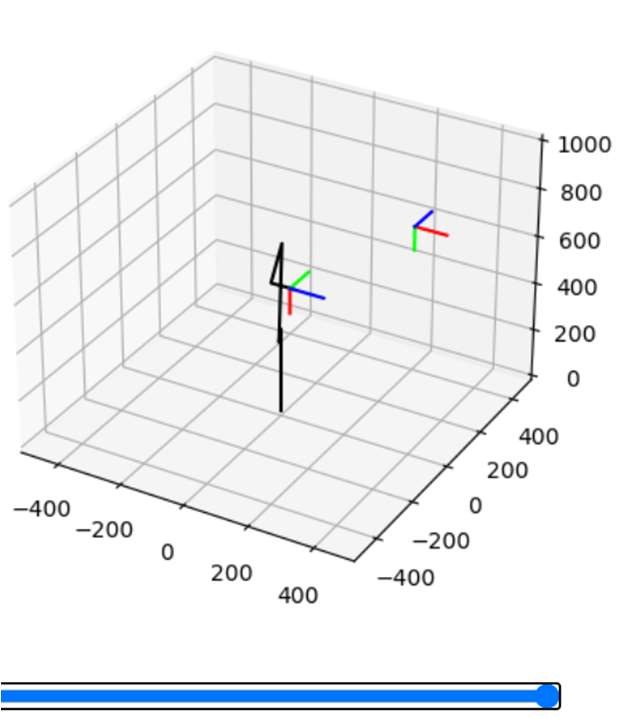
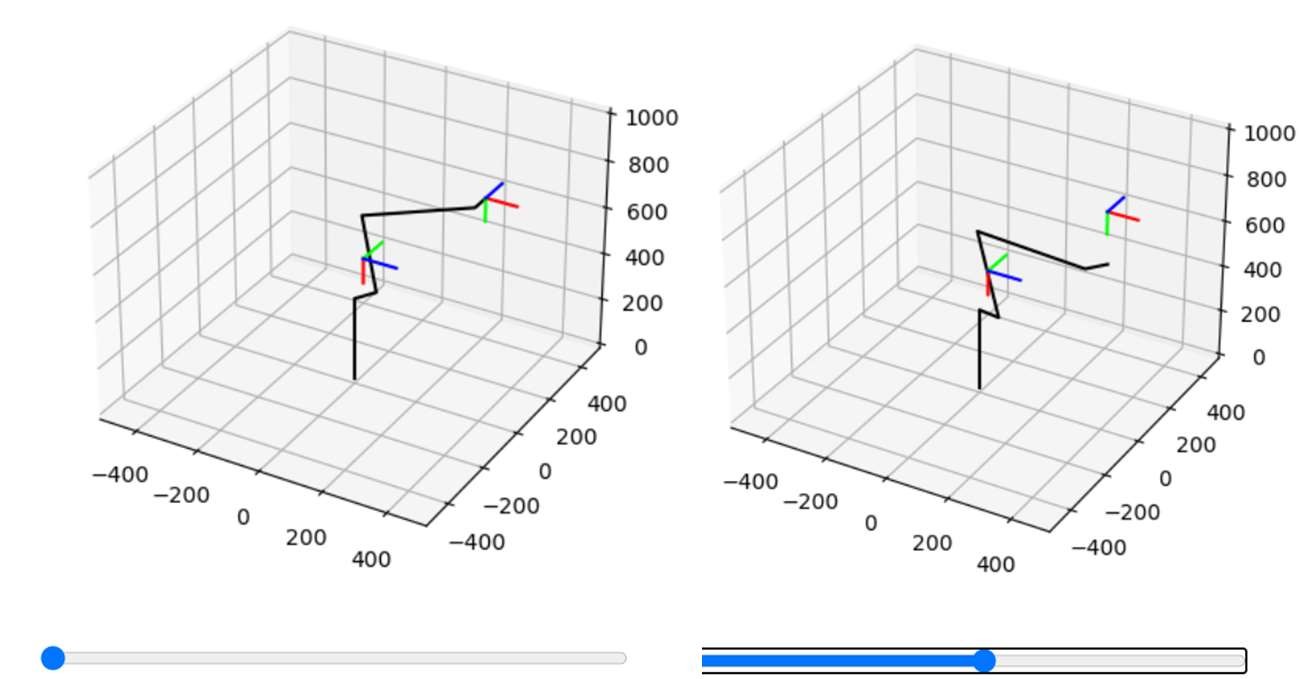


Рисунок 4. Движение в режиме переброски.

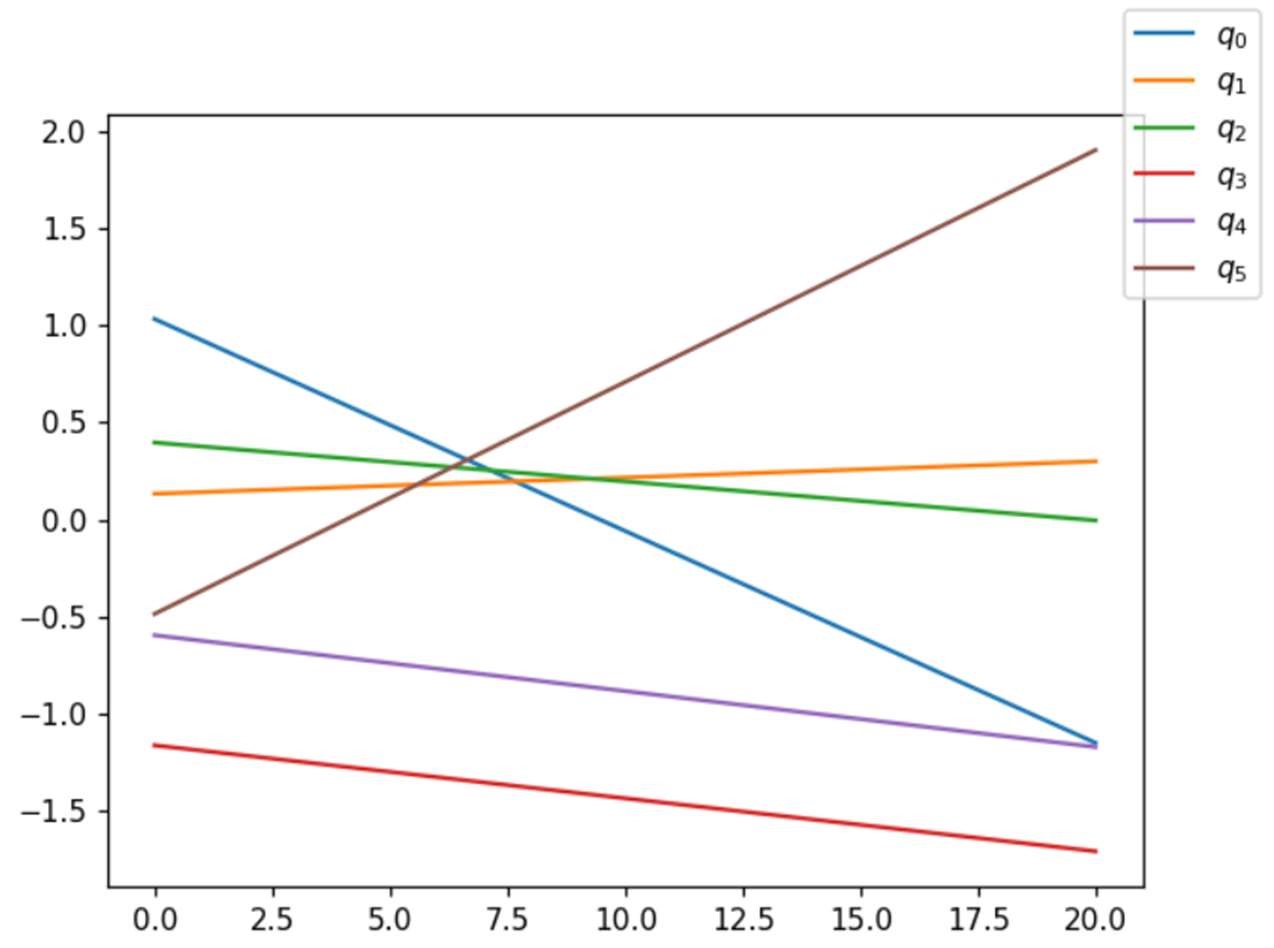


Рисунок 5. График изменения обобщённых координат.

При линейном движении и в режиме переброски наблюдаются следующие различия:

* в при линейном движении рабочий орган двигается по линии из точки s в точку e, при этом ориентация так же изменяется линейно. Алгоритм интерполяции в данном случае интерполирует координаты и ориентацию, только потом переводя их в обобщенные координаты.
* при переброске рабочий орган может отклоняться от линейной траектории, но при этом обобщенные координаты изменяются линейно, что позволяет быстрее дойти до нужной позиции.

Два линейных движения в цепочке:

## Листинг 4

s = Transform(

Vector(200, 400, 200),

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

)

i = Transform(

Vector(650, -100, 800),

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

)

e = Transform(

Vector(300,300, 500),

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

)

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

# функция для объединения 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

)

(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

)

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()

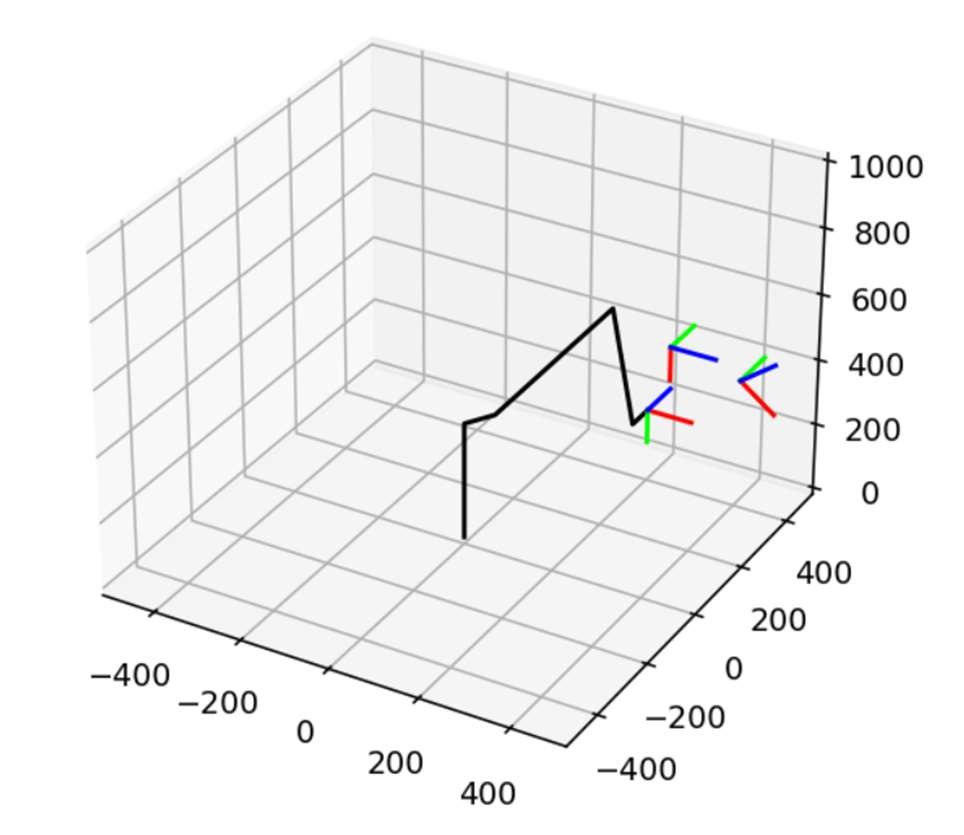


Рисунок 6. Движение в режиме переброски.

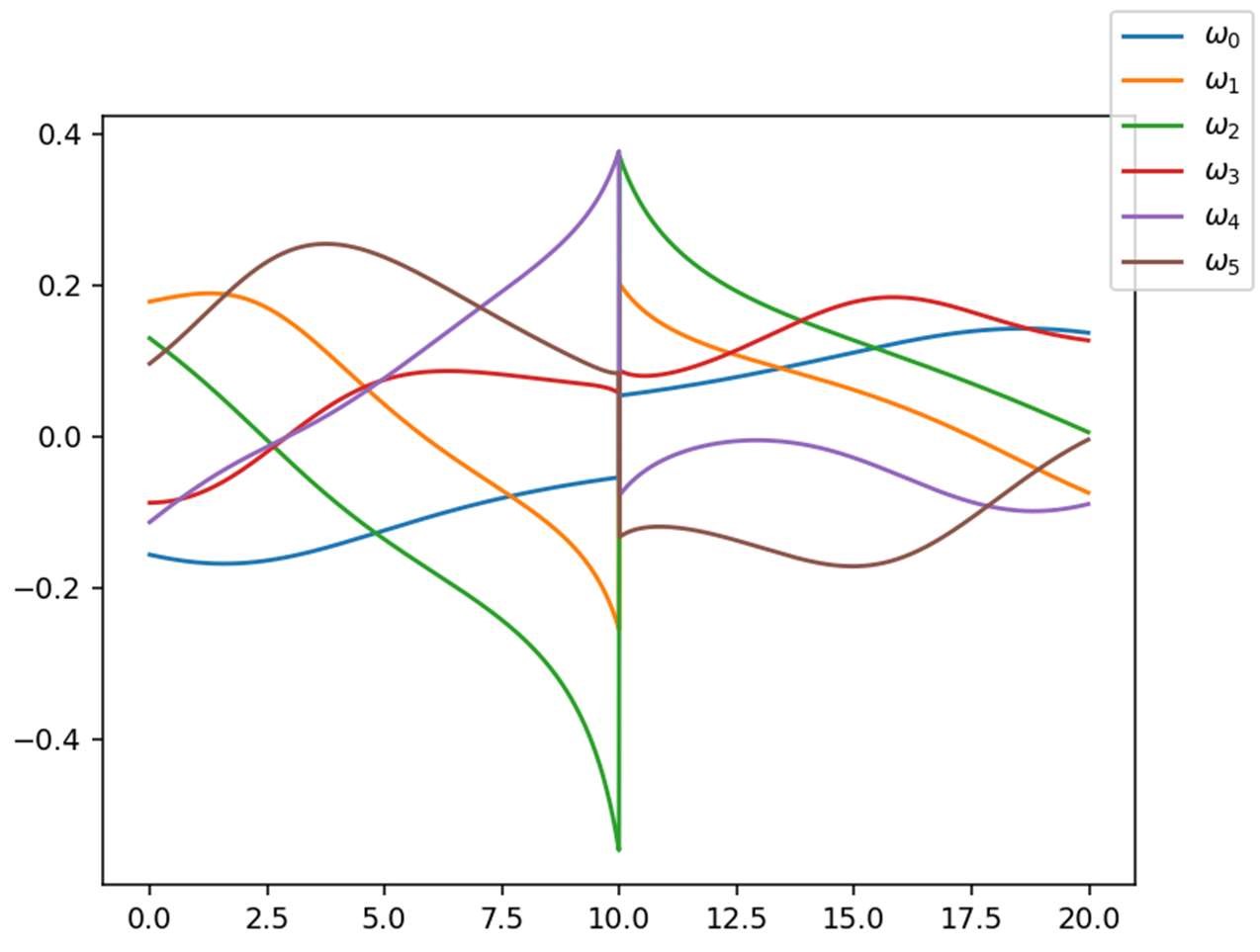


Рисунок 7. График скорости обобщённых координат.

Т.к. движение происходит по линейному закону, то графики скорости не линейны. Смена направления на движение к следующей точку сопровождается резкими изменениями скорости.

Напишем функцию для объединения двух линейных движений со сглаживанем:

## Листинг 5

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 = 0.55

(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

)

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

#скорость

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() #ускорение 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 ),2) / step;

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

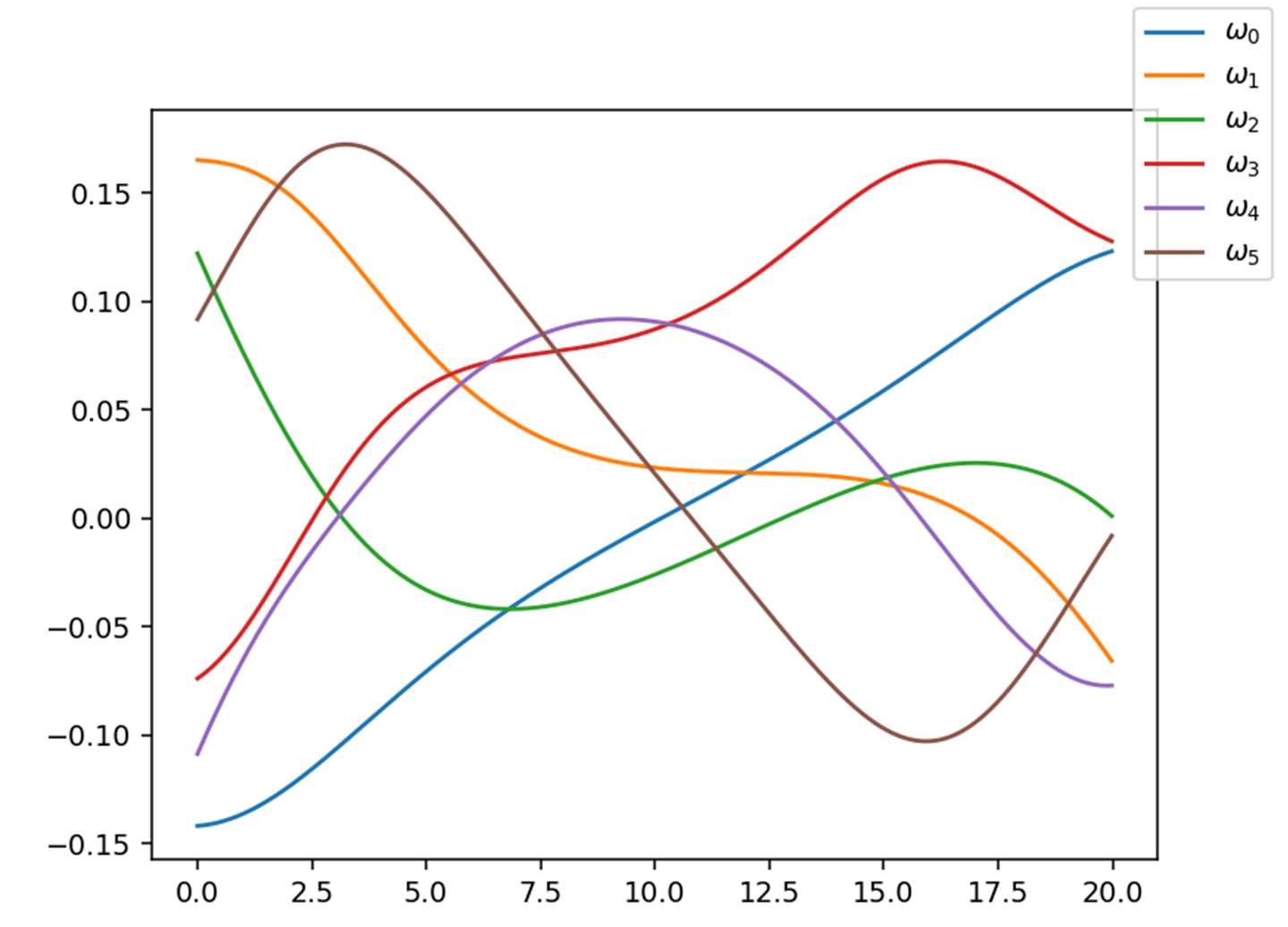


Рисунок 8. График скорости обобщённых координат.

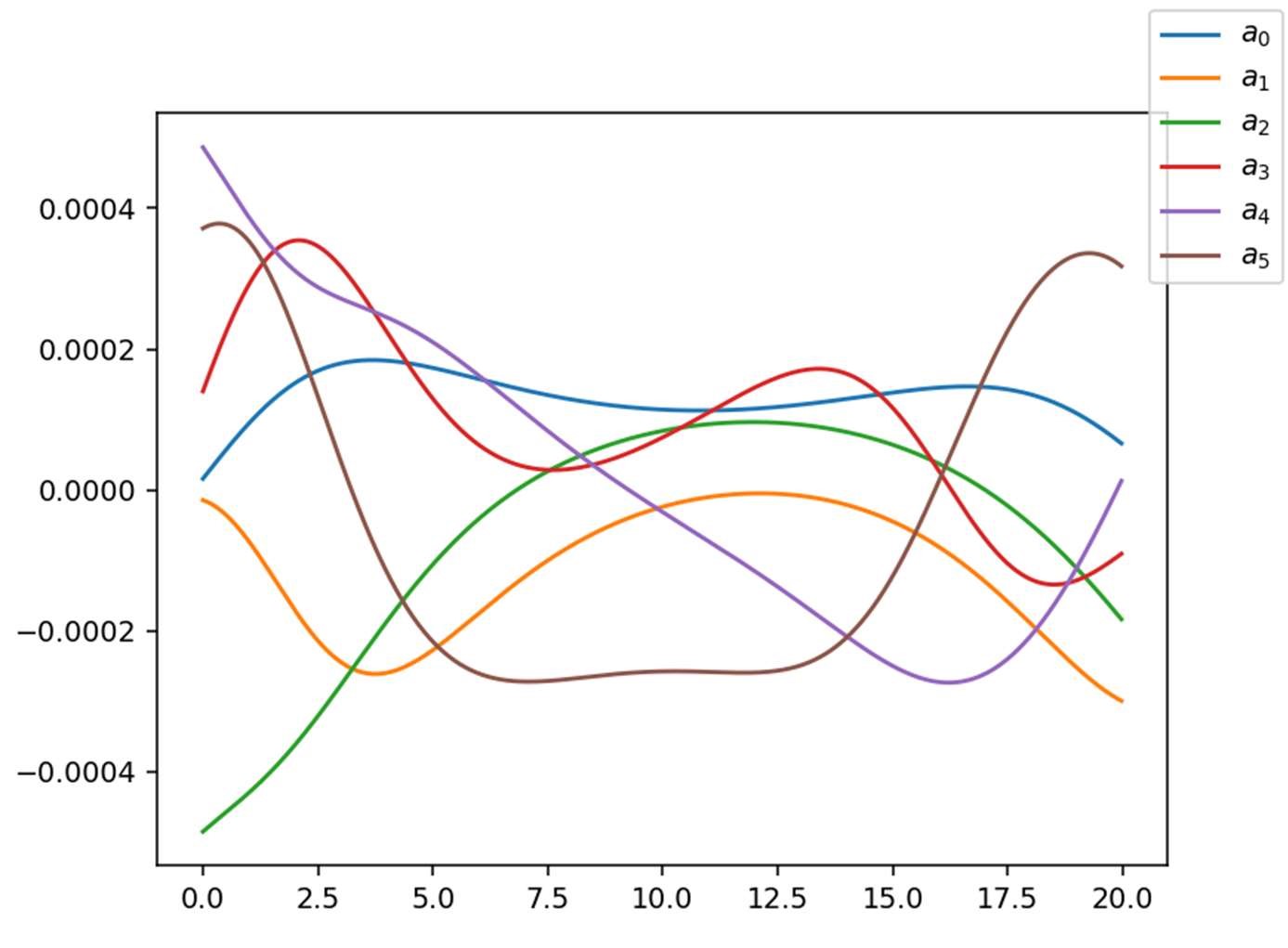


Рисунок 9. График ускорений обобщённых координат.

При данного алгоритма можно управлять сглаживанием изменением параметра blend. При его увеличении скорость обобщенных координат изменяется более плавно, тем самым можно избежать резких скачков скоростей и ускорения.

# Управление роботом SCARA

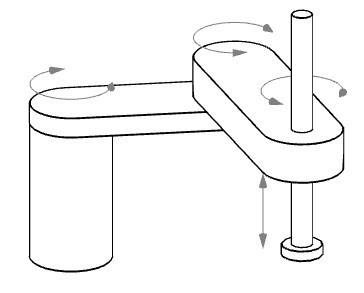


Рисунок 10. Робот SCARA.

## Листинг 6

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]

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

] 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 q0 = Vector(1, 0, 0).angle\_to( Vector(target.translation.x, target.tr anslation.y, 0), Vector(0, 0, 1) ) - np.arccos((l[1] \*\* 2 + d \*\* 2 - l[2]

\*\* 2) / (2 \* l[1] \* d)) q1 = np.pi -

np.arccos( (l[1] \*\* 2 + l[2] \*\* 2 - d \*\* 2) / (2 \* l[1] \* l[2]) ) triangle\_angle = np.arcsin( l[2] \* np.sin(q1 - np.pi ) / d ) lift\_angle = np.arctan2( target.translation.y, target.translation.x )

q2 = target.angle - q0 - q1 q3 = l[0] - target.translation.z 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

) scara\_lim = [ (-140, 140),

(-150, 150),

(-400, 400),

(0, 180)

] def scara\_ik\_lim(target, l):

solution = scara\_ik(target, l) for index in range(len(solution) - 1):

if solution[index] < np.deg2rad(scara\_lim[index][0]) or\ solution[index] > np.deg2rad(scara\_lim[index][1]) or\ np.isnan(solution[index]):

return None return solution

class Target:

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

super(Target, self).\_\_init\_\_() self.translation = translation self.angle = angle

def to\_transform(self): return Transform( self.translation, Quaternion.from\_angle\_axis( self.angle, Vector(0, 0, 1)

)

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

progress = t / total return Target(

Vector.lerp(start.translation, end.translation, progress), start.angle + (end.angle - start.angle) \* progress

)

s = Target(

Vector(200, 300, 120),

0 ) e = Target(

Vector(200, -200, 200), np.pi / 2

)

Линейное движение:

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

)

fig, ax = graphics.figure(1000) lines, = ax.plot(x, y, z, color="#000000") graphics.axis(ax, s.to\_transform(), 100) graphics.axis(ax, e.to\_transform(), 100)

r, g, b = graphics.axis(ax, Transform.identity(), 1)

total = 100

def animate(frame): trs = lin(s, e, frame, total) q = scara\_ik\_lim( trs, scara\_l ) if q != None: chain = scara\_chain(q, scara\_l)

(x, y, z) = graphics.chain\_to\_points(chain) lines.set\_data\_3d(x, y, z) global r, g, b

r.remove(); g.remove(); b.remove() r, g, b = graphics.axis(ax, chain[-1], 100)

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

)

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

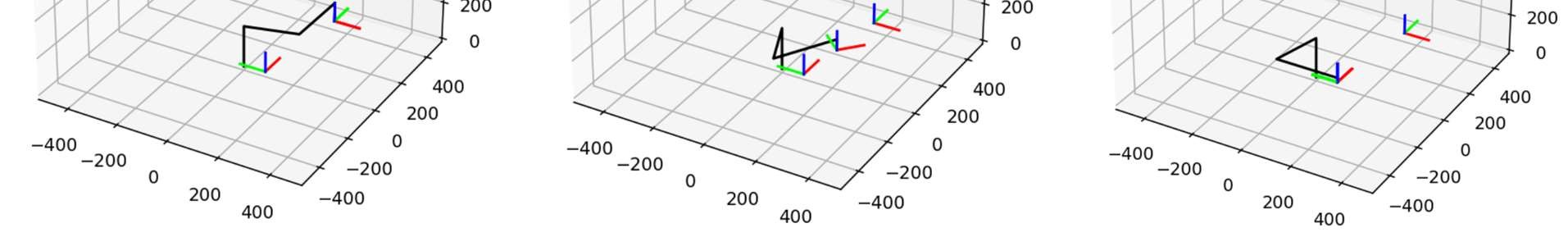
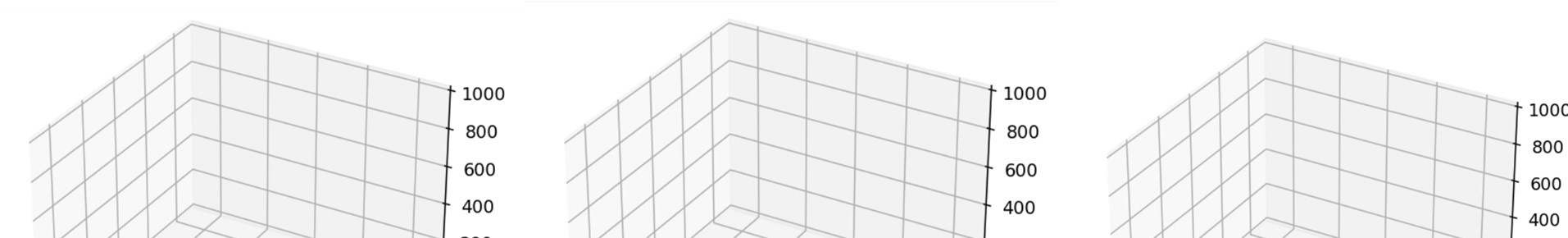


Рисунок 11. Линейное движение.

Движение в режиме переброски:

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

)

fig, ax = graphics.figure(1000) lines, = ax.plot(x, y, z, color="#000000") graphics.axis(ax, s.to\_transform(), 100) graphics.axis(ax, e.to\_transform(), 100)

total = 100

s\_q = scara\_ik\_lim(s, scara\_l) e\_q = scara\_ik\_lim(e, scara\_l)

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 = scara\_chain(q, scara\_l)

(x, y, z) = graphics.chain\_to\_points(chain) lines.set\_data\_3d(x, y, z)

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

)

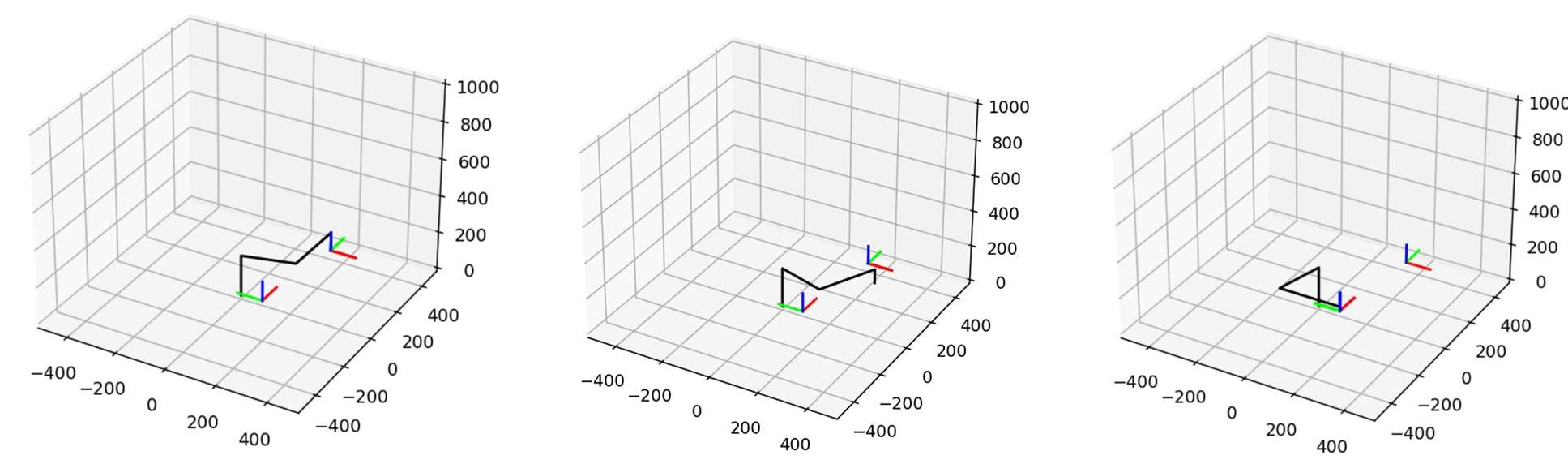


Рисунок 12. Движение в режиме переброски.

При линейном движении и в режиме переброски наблюдаются те же различия, что и в прошлом примере.

Добавим промежуточную точку:

## Листинг 7

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

progress = t / total if progress < 0.5:

return lin(start, inter, progress \* 2, 1) else:

return lin(inter, end, (progress - 0.5) \* 2, 1)

i = Target(

Vector(400, 100, 0), np.pi,

)

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

)

fig, ax = graphics.figure(1000) lines, = ax.plot(x, y, z, color="#000000") graphics.axis(ax, s.to\_transform(), 100) graphics.axis(ax, i.to\_transform(), 100) graphics.axis(ax, e.to\_transform(), 100)

total = 100

def animate(frame): trs = lin\_lin(s, i, e, frame, total) q = scara\_ik\_lim(

trs, scara\_l ) if q != None:

chain = scara\_chain(q, scara\_l) (x, y, z) = graphics.chain\_to\_points(chain) lines.set\_data\_3d(x, y, z)

animate(0) fps = 25

scara\_ani = animation.FuncAnimation( fig, animate, frames=total, interval=1000.0/fps

)

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

v\_lin\_lin = np.vectorize(lin\_lin, excluded={0, 1, 2, 4}) v\_irb\_ik = np.vectorize(scara\_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), scara\_l, )) / 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]/20, label="$\omega\_3/20$") fig.legend() fig.show()

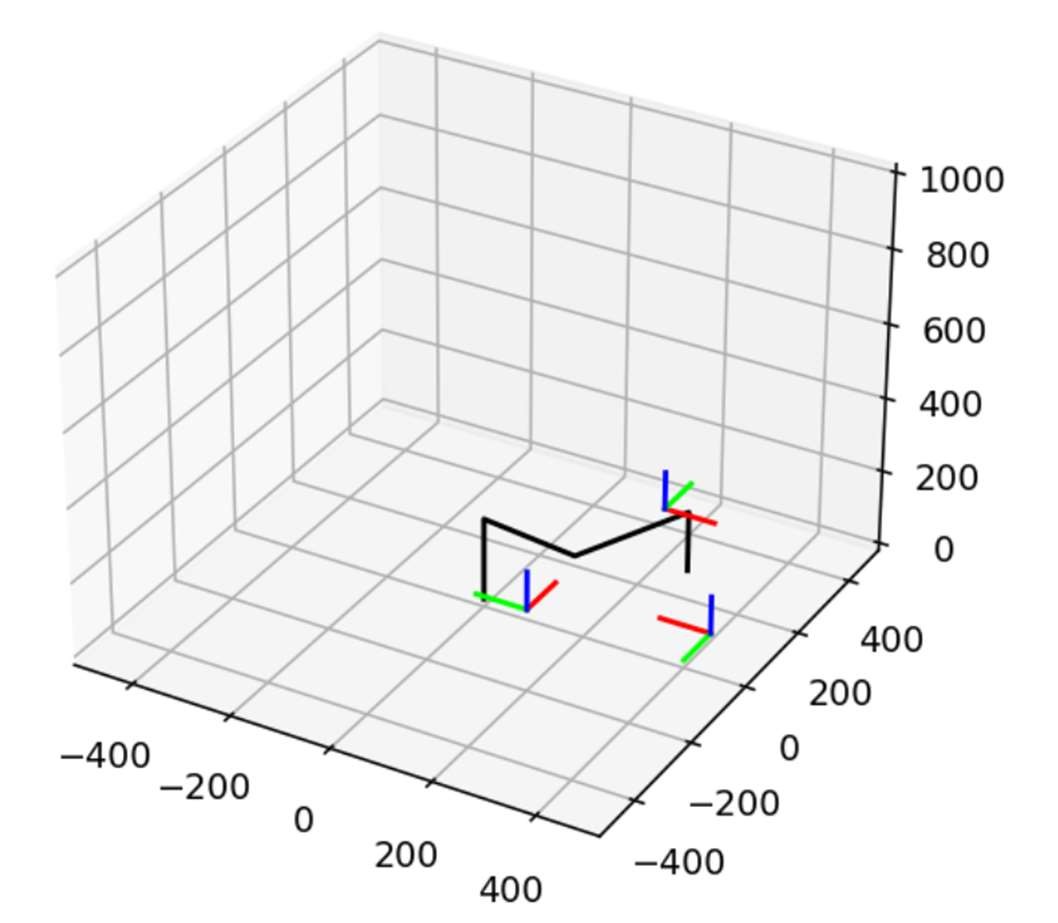


Рисунок 13. Линейное движение.

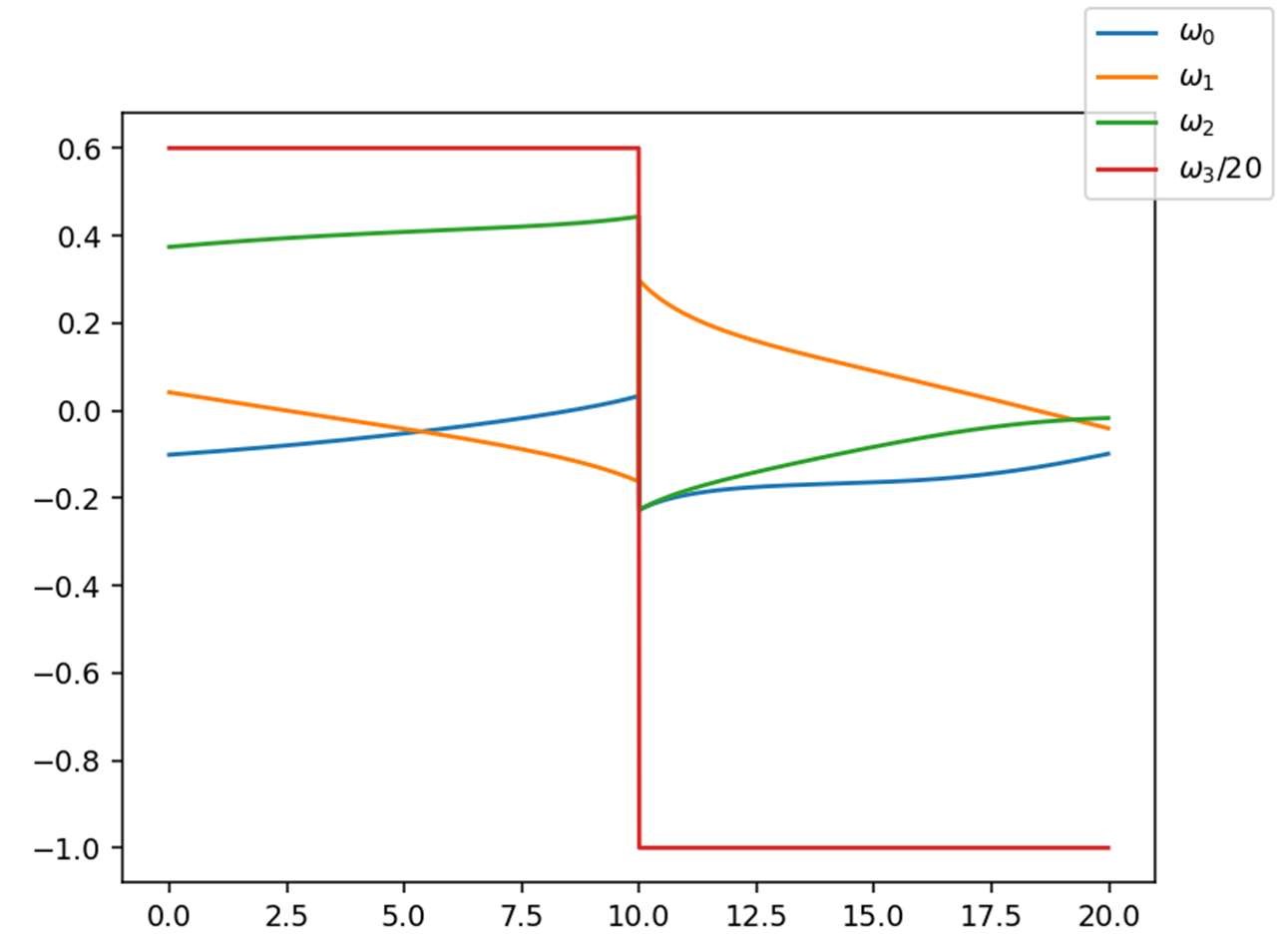


Рисунок 14. Скорость при линейном движении.

В данном случае наблюдается резкий скачок скоростей.

Напишем функцию для объединения двух линейных движений со сглаживанем:

Листинг 7 def bezier\_target(a, b, c, t): position = Vector.lerp(

Vector.lerp(a.translation, b.translation, t), Vector.lerp(b.translation, c.translation, t), t ) rotation = (1 - t) \*\* 2 \* a.angle +\ 2 \* t \* (1 - t) \*\* b.angle +\

t\*\*2 \* c.angle

return Target(position, rotation)

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\_target( a, b, c, progress ) else: return lin\_lin(start, inter, end, t, total)

blending = 0.55

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

)

fig, ax = graphics.figure(1000) lines, = ax.plot(x, y, z, color="#000000") graphics.axis(ax, s.to\_transform(), 100) graphics.axis(ax, i.to\_transform(), 100) graphics.axis(ax, e.to\_transform(), 100)

total = 100 def animate(frame): trs = lin\_lin\_smooth(s, i, e, frame, total, 0.1) q = scara\_ik\_lim( trs, scara\_l, ) if q != None:

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

(x, y, z) = graphics.chain\_to\_points(chain) lines.set\_data\_3d(x, y, z)

animate(0) fps = 25

scara\_ani = animation.FuncAnimation( fig, animate, frames=total, interval=1000.0/fps

)

#скорость

v\_lin\_lin = np.vectorize(lin\_lin\_smooth, excluded={0, 1, 2, 4, 5}) v\_irb\_ik = np.vectorize(scara\_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), scara\_l, )) / 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]/20, label="$\omega\_3/20$") fig.legend() fig.show() #ускорение

v\_lin\_lin = np.vectorize(lin\_lin\_smooth, excluded={0, 1, 2, 4, 5}) v\_irb\_ik = np.vectorize(scara\_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), scara\_l, ),2) / step;

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]/100, label="$a\_3/100$") fig.legend() fig.show()

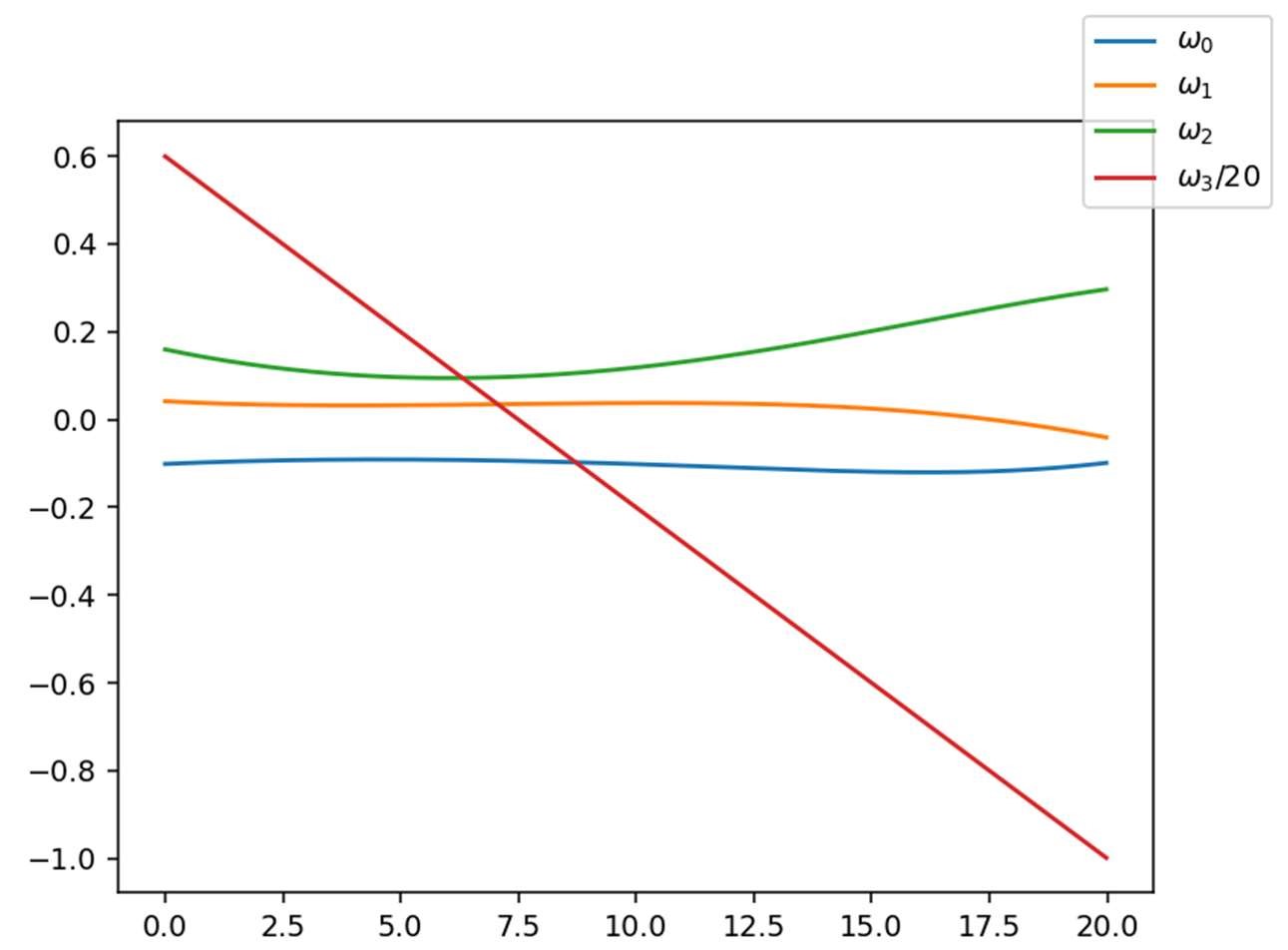


Рисунок 15. Скорость при линейном движении со сглаживанием.

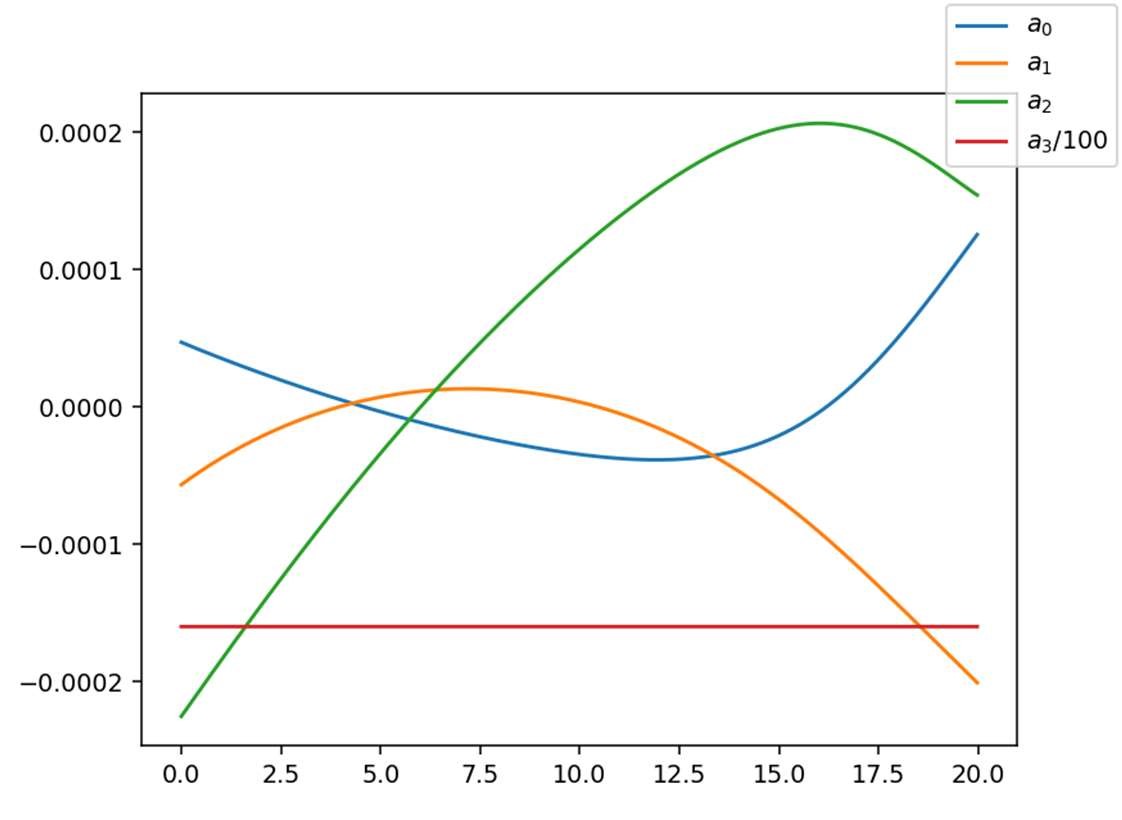


Рисунок 15. Ускорение при линейном движении.

Ускорение также стало изменяться плавно без рывков.

Вывод: в данной лабораторной работе мы изучить математический аппарат, применяемый для решения задач траекторного управления.