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 _____ (дата) _____ (подпись) Абдулзагиров М.М. (ФИО)

Принял
преподаватель: _____ (дата) _____ (подпись) Порунов М.Ю. (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 степенямиподвижности.

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

```
#длины звеньев
irb_l = [352.0, 70.0, 350.0, 380.0, 65.0]

#диапазон изменения обобщенных координат (град.)
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
    ]
```

#ОЗК

```
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)
    )
```

#закон изменения положения

```
def target(t, total):     return Transform(
        Vector(200,50+ 1000 * t / total, 500) if t / total < 0.5
      else Vector(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())
```

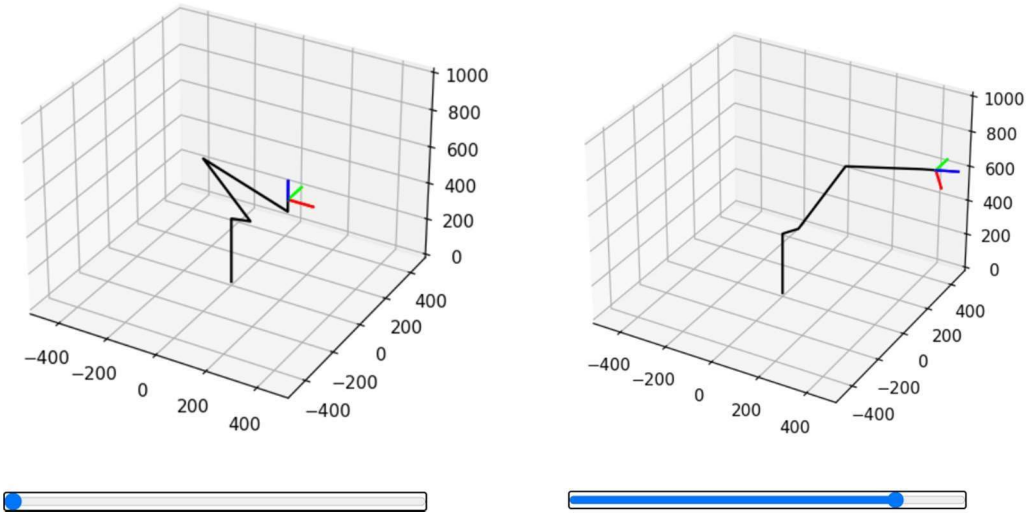

Рис. 2 Результат выполнения программы

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()
```

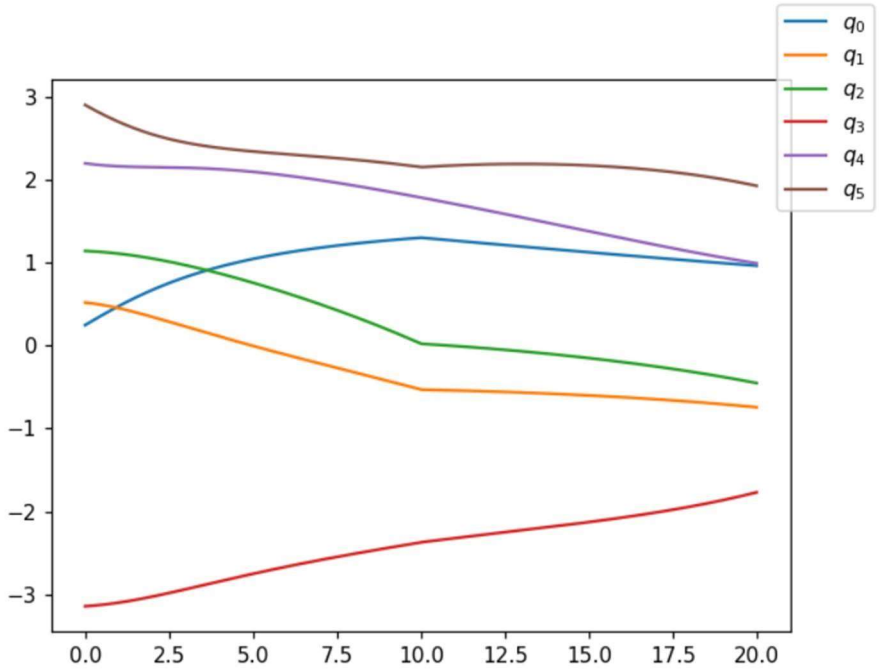

Рис. 3 График изменения обобщенных координат

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()
```

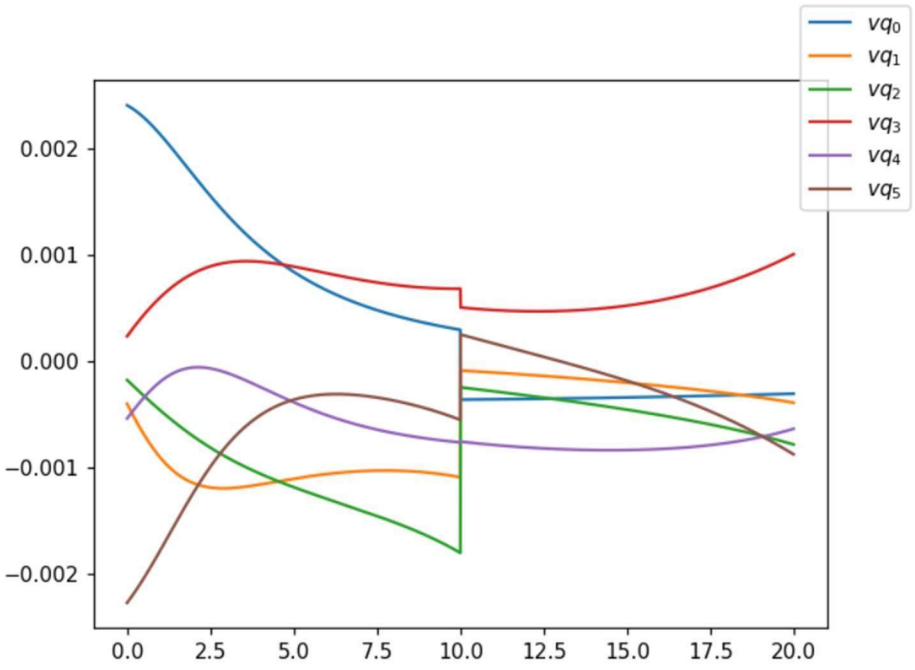

Рис. 4 Скорость изменения обобщенных координат

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.

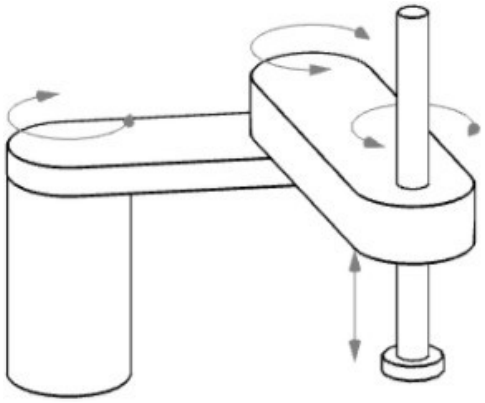

Рис. 5 Робот SCARA

В листинге 4 приведен код для реализации работы кинематики манипулятора, построения графика изменения обобщенных координат и оценки скорости изменения обобщенных координат.

Листинг 4 – Решение задания 2.

```
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):
          self.angle = angle # угол поворота вокруг вертикальной
оси
        super(Target, self).__init__()
          self.translation = translation

    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.remove()
     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
)
```

Пример выполнения программы приведён на рис 6.

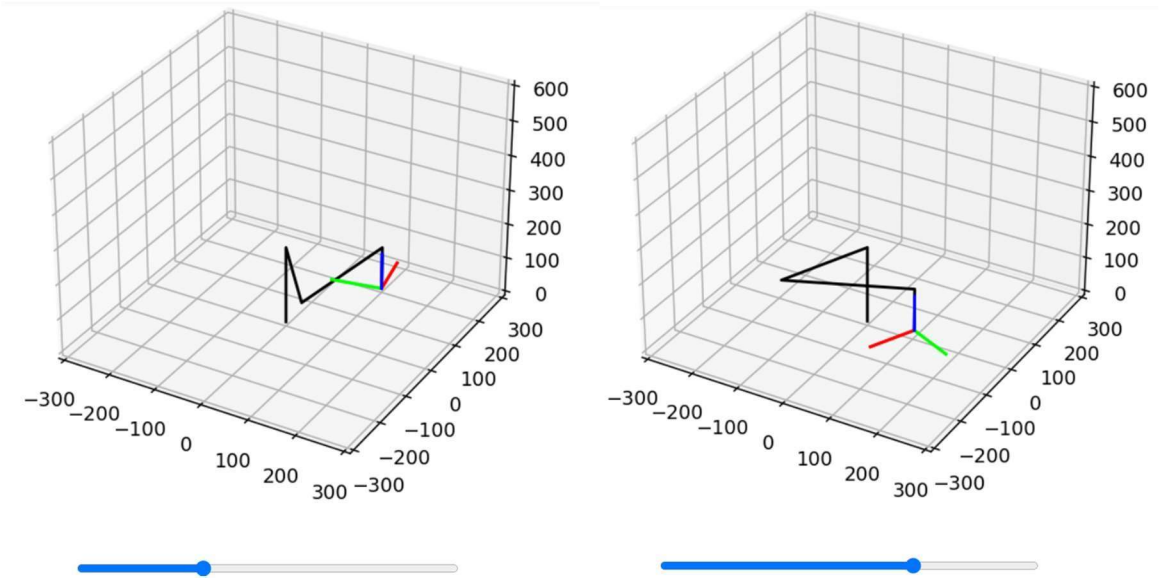

Рис. 6 Пример выполнения программы

В листинге 5 показан реализация метод оценки рабочей зоны. Графическое отображение показано на рис. 7.

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()
```

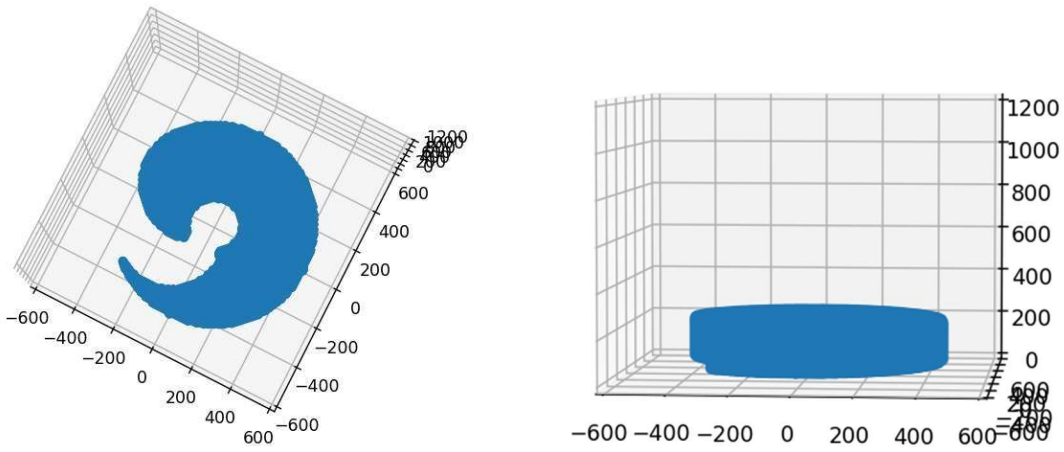

Рис. 7 Рабочая зона

В листинге 5 показан реализация метода для отображения графика изменения обобщенных координат. Графическое отображение показано на рис. 8. На рис 9 изображена рабочая зона робота.

Листинг 6 - построение графика.

```
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()
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
    )
```

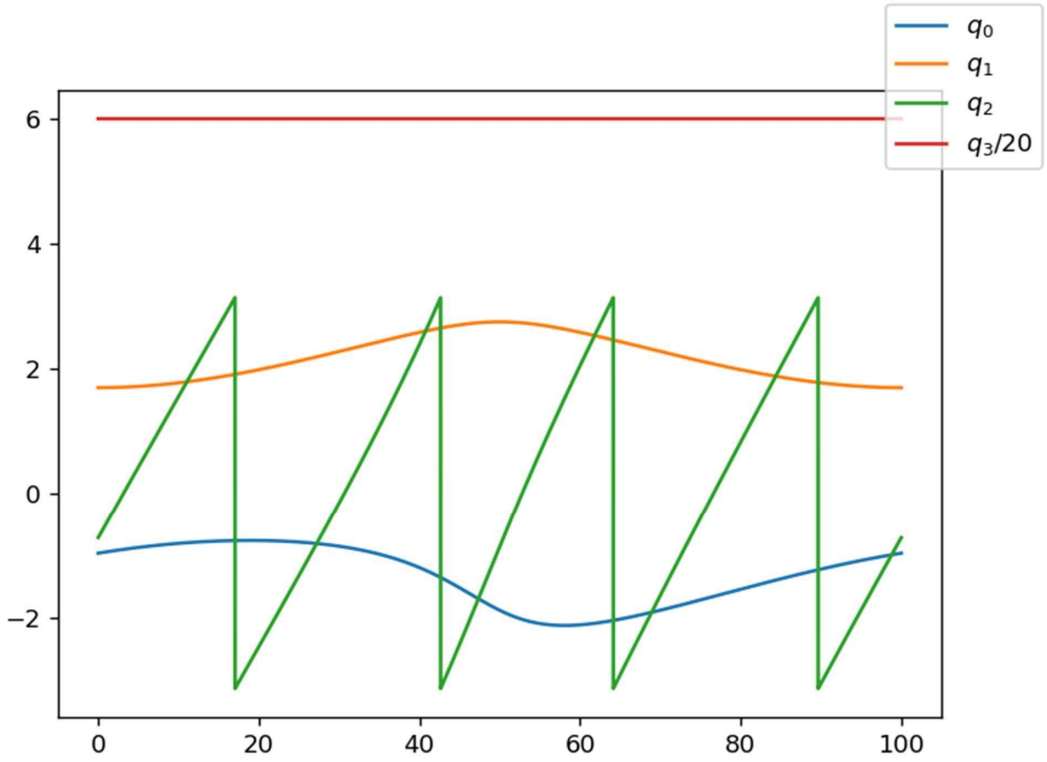

Рис. 8 График изменения обобщенных координат

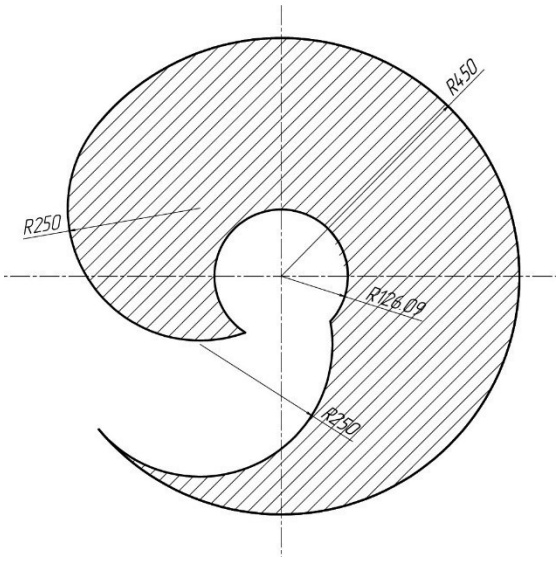

Рис. 9 Рабочая зона

Задание №3.

Для робота PUMA проведем исследование для своих точек траектории, проанализируем влияние параметра blend на скорость обобщенных координат, оценим ускорения обобщенных координат. Пример выполнения программы приведён на рис 10.

Объявим функцию для описания линейной траектории (листинг 7).

Листинг 7 – Функция описания линейной траектории

```
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()
```

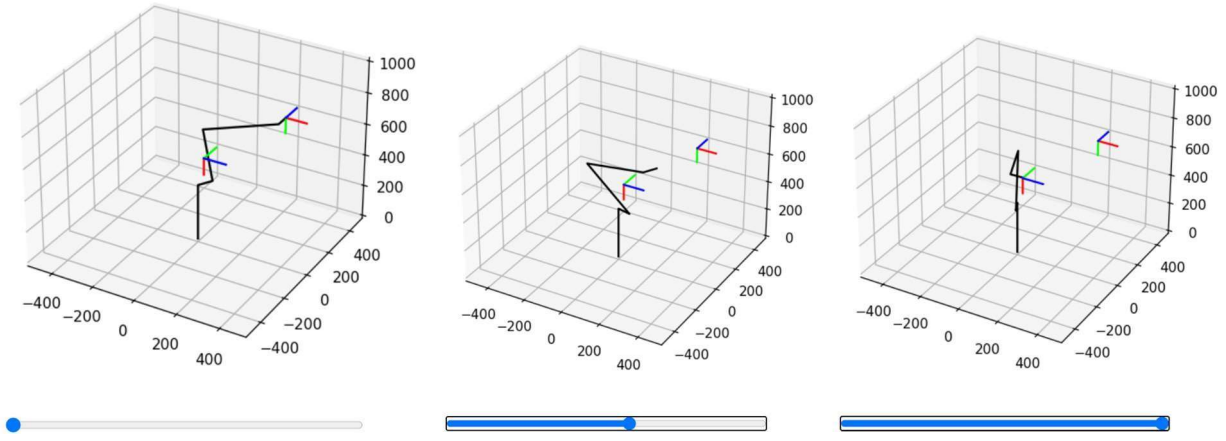

Рис. 10 Пример выполнения программы

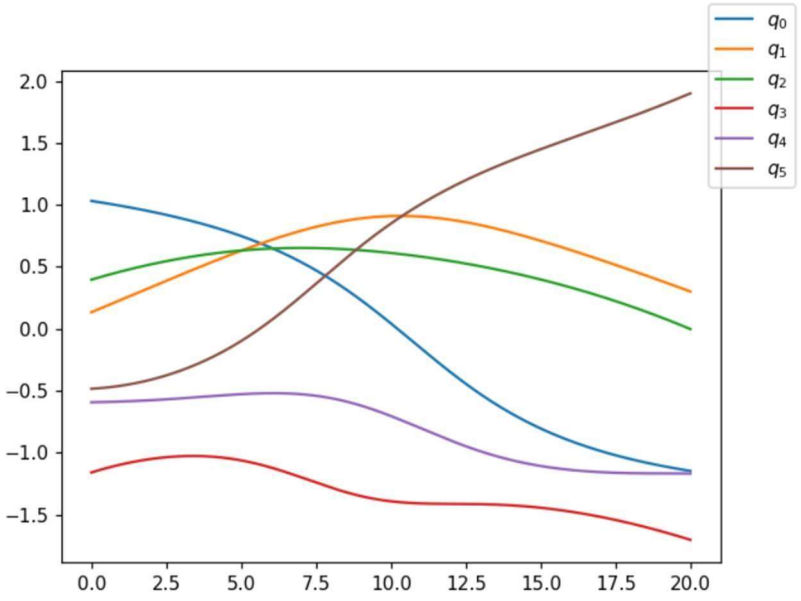

Рис 11. График изменения обобщённых координат.

Реализация движения в режиме переброски (листинг 8). Работа программы отображена на рис. 12.

Листинг 8 – Движение 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()
```

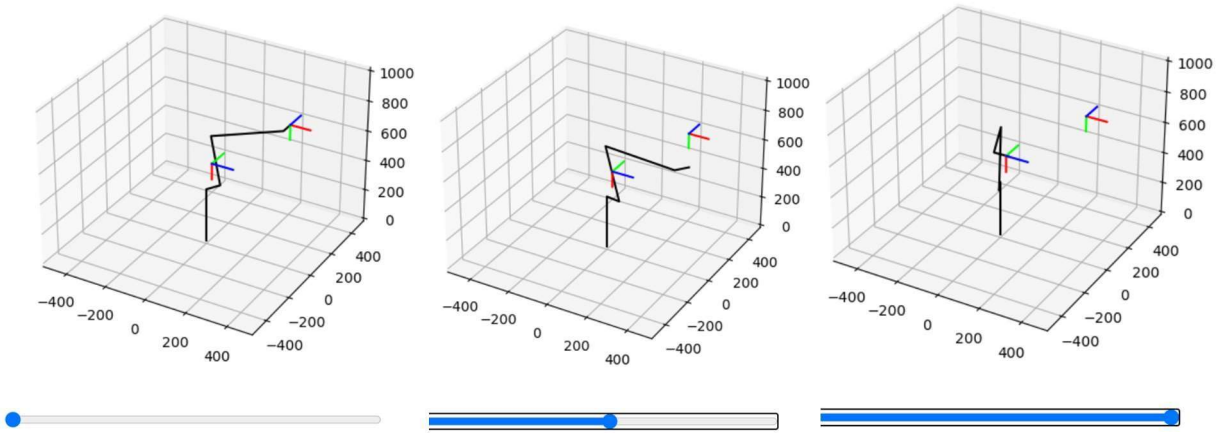

Рис. 12 Пример работы программы

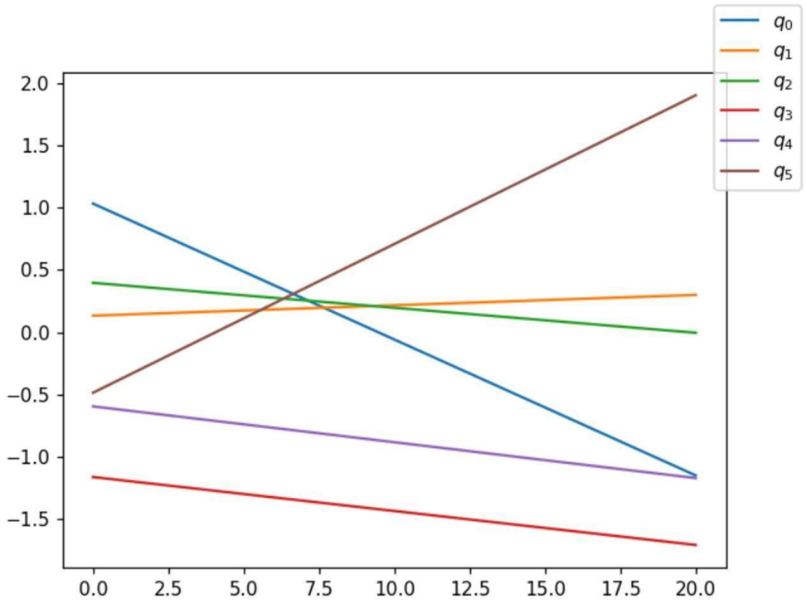

Рис. 13. График изменения обобщённых координат.

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.

Построим график изменения обобщенных координат для переброски (листинг 9). График показан на рис.15.

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()
```

Два линейных движения в цепочке:

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

Листинг 11 – Объединение двух линейных движений

```
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()
```

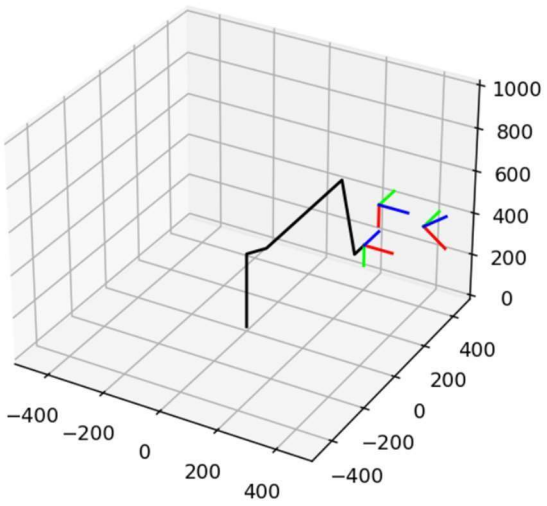

Рис. 14 Движение в режиме переброски

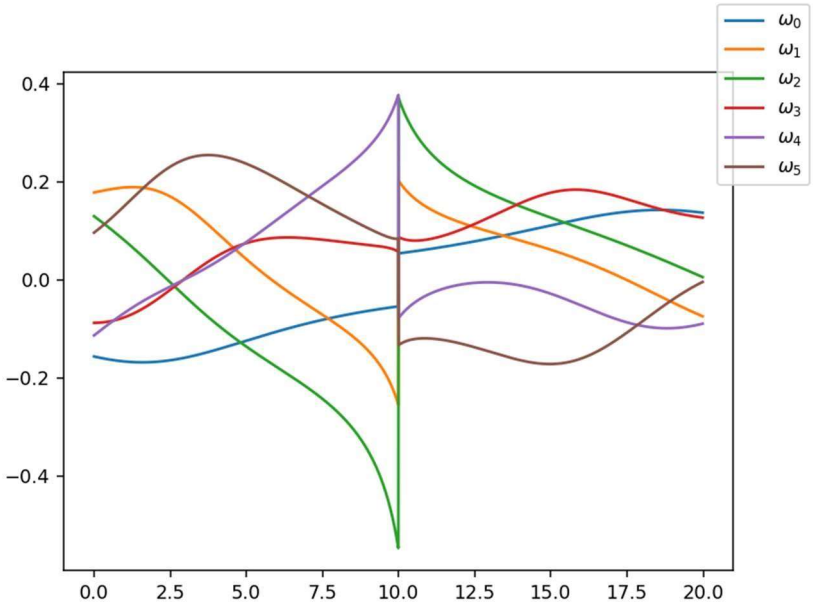

Рисунок 15. График скорости обобщённых координат.

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 сглаживанием (Листинг 12). Графики скоростей и ускорений обобщённых координат изображены на рис 16-17.

Листинг 12 – 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 = 0.55
```

```
(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=100
0.0/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()
```

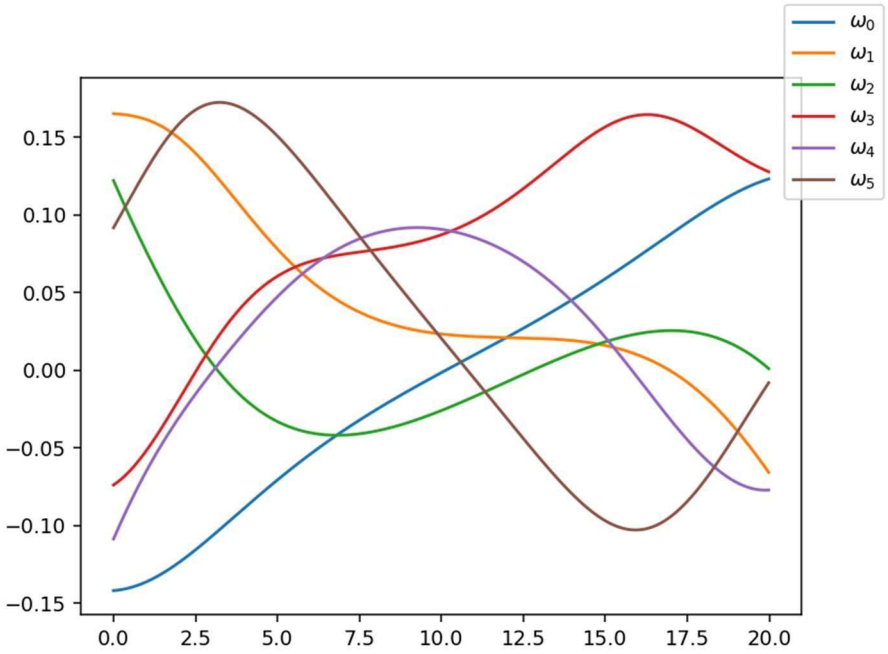

Рисунок 16. График скорости обобщённых координат.

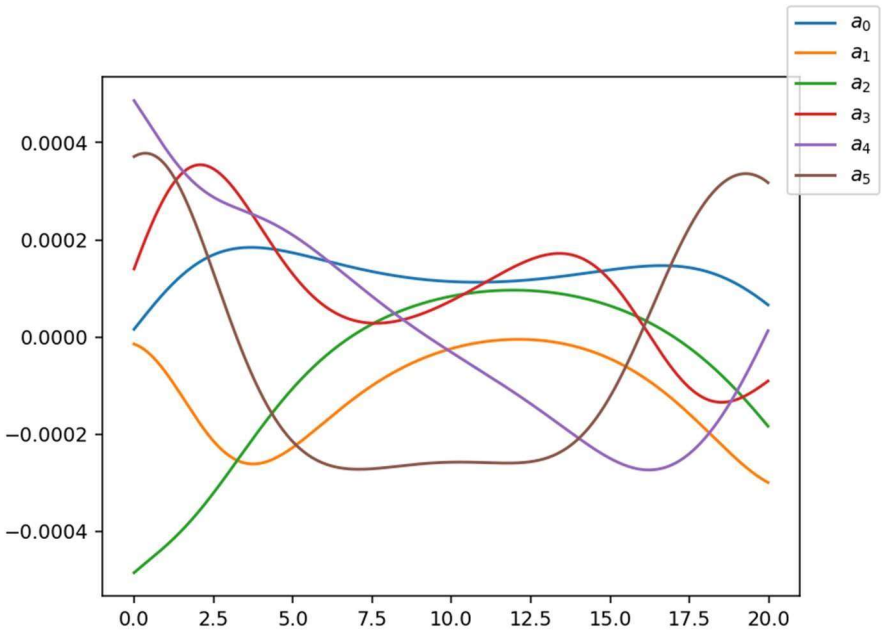

Рисунок 17. График ускорений обобщённых координат.

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

Задание №5

Для робота SCARA проведем исследование для своих точек траектории, проанализируем влияние параметра blend на скорость обобщенных координат, оценим ускорения обобщенных координат.

Все действия производятся аналогично заданию №4 (листинг 12). Результаты представлены на рис. 18.

Листинг 12 – Линейное движение

```
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),
    b) 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())
```

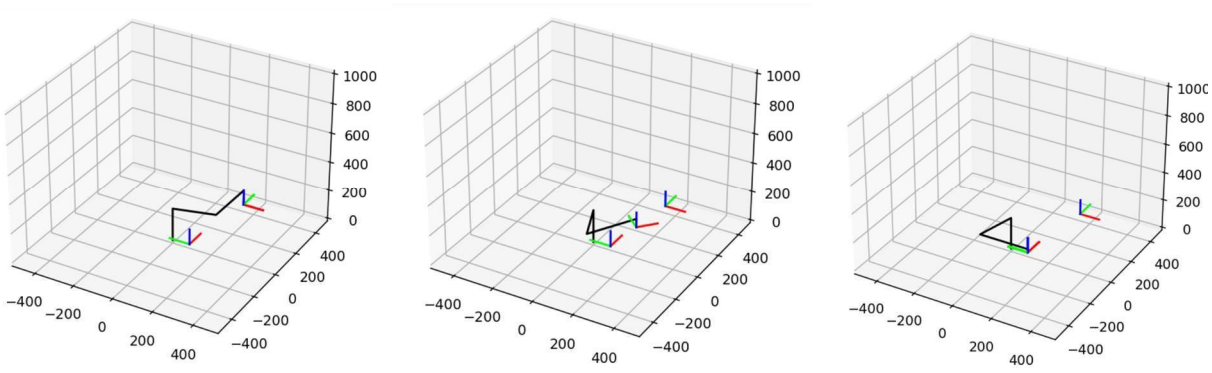

Рисунок 18. Линейное движение.

Создадим программу для движения в режиме переброски (листинг 13). Результат запуска программы представлен на рис 19.

Листинг 13 – программа движения 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,     interv
al=1000.0/fps )
```

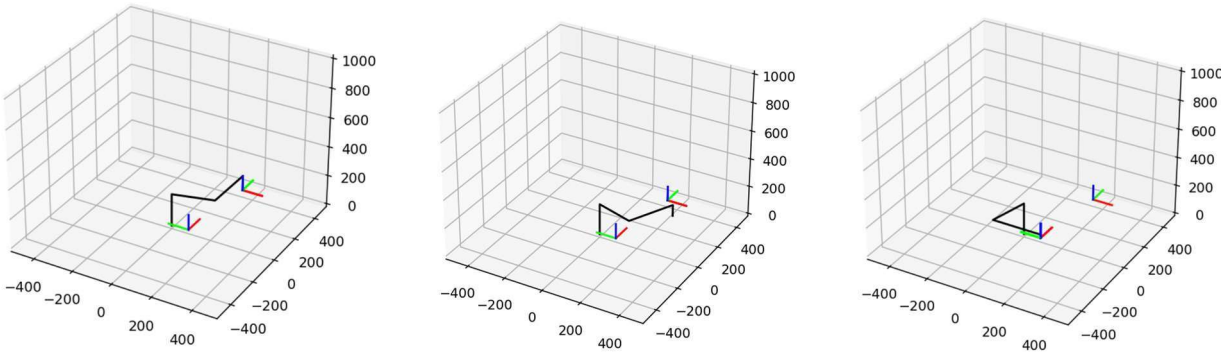

Рисунок 19. Движение в режиме переброски.

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

Добавим промежуточную точку (листинг 14). Результат запуска программы представлен на рис 20-21.

Листинг 14 – движение с промежуточной точкой

```
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,     interv
al=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()
```

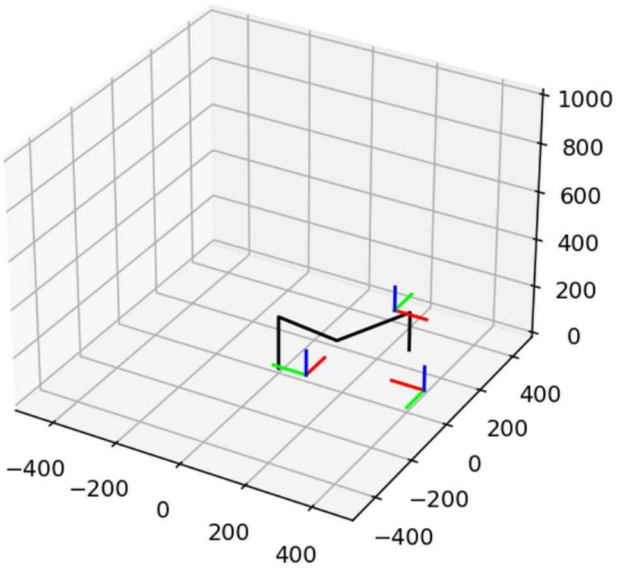

Рисунок 20. Линейное движение.

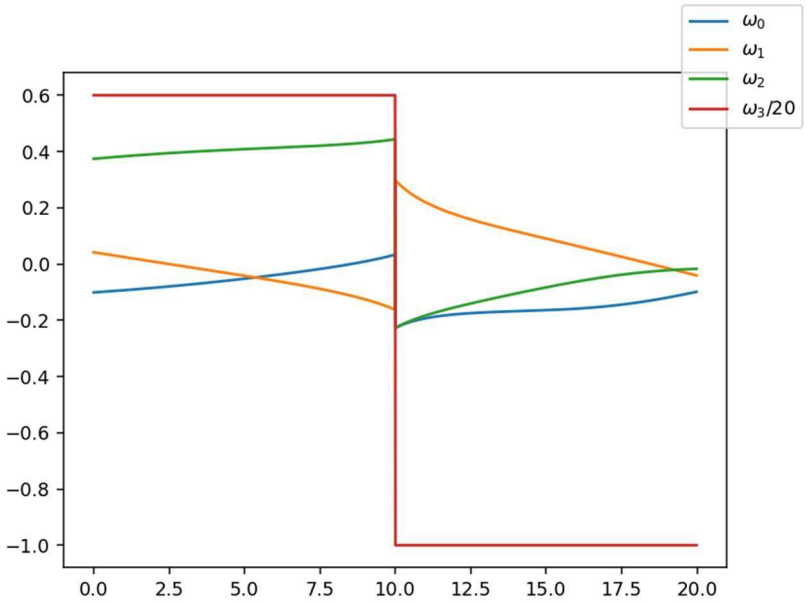

Рисунок 21. Скорость при линейном движении.

В данном случае наблюдается резкий скачок скоростей.

Напишем функцию для объединения двух линейных движений со сглаживанием (листинг 15). Результат запуска программы представлен на рис 22-24.

Листинг 15 – программа со сглаживанием перехода

```
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
```

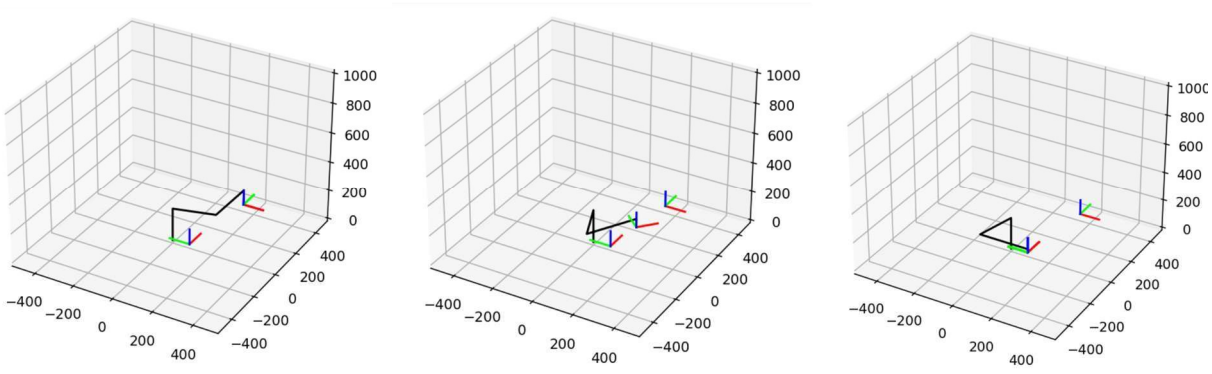

Рисунок 22. Линейное движение.

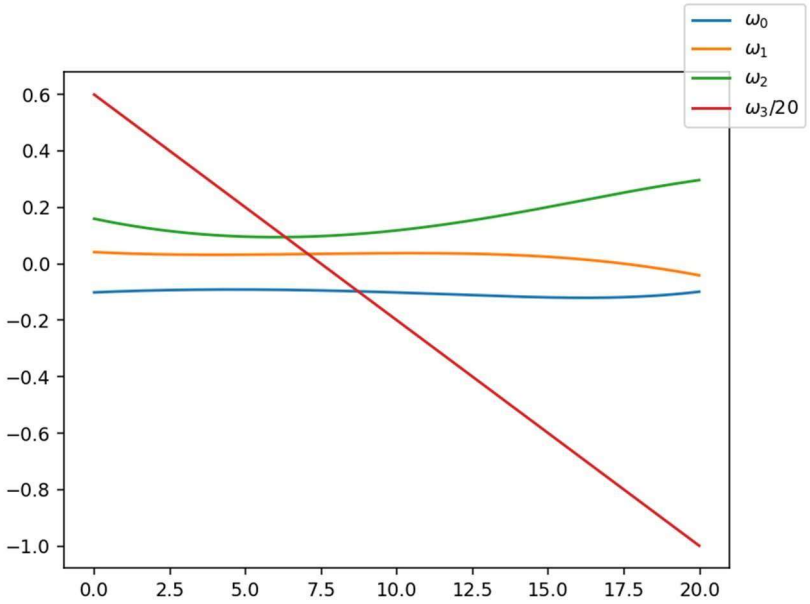

Рисунок 23. Скорость при линейном движении со сглаживанием.

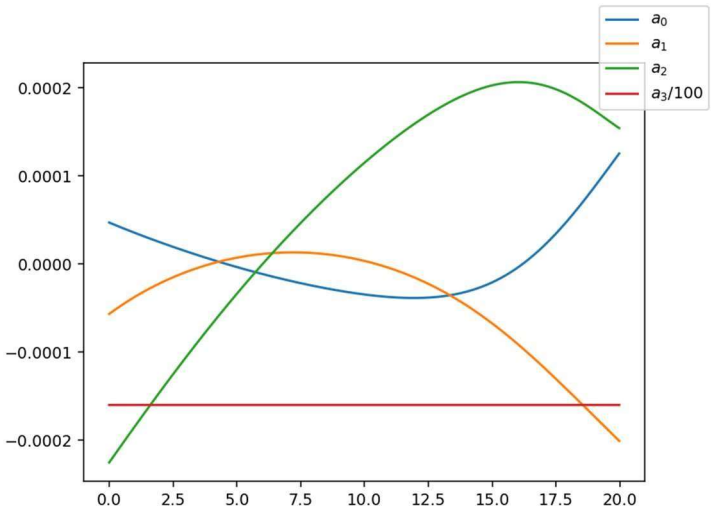

Рисунок 24. Ускорение при линейном движении.

Ускорение также стало изменяться плавно без рывков.

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