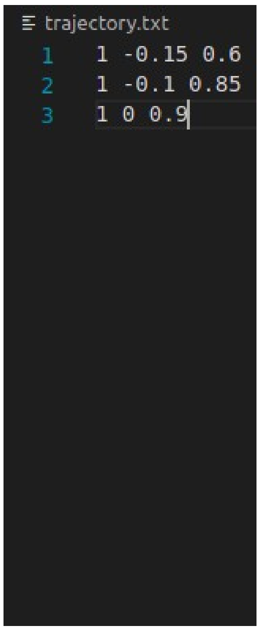
LAB 3 : Trajectory

Soulaïman Marsou

Manipulator for this lab :
For this lab, I use the manipulator of the LAB2, the one in the file LV1_manipulator5.py for which I computed the Pieper's solution myself.

1. Generate the trajectory :

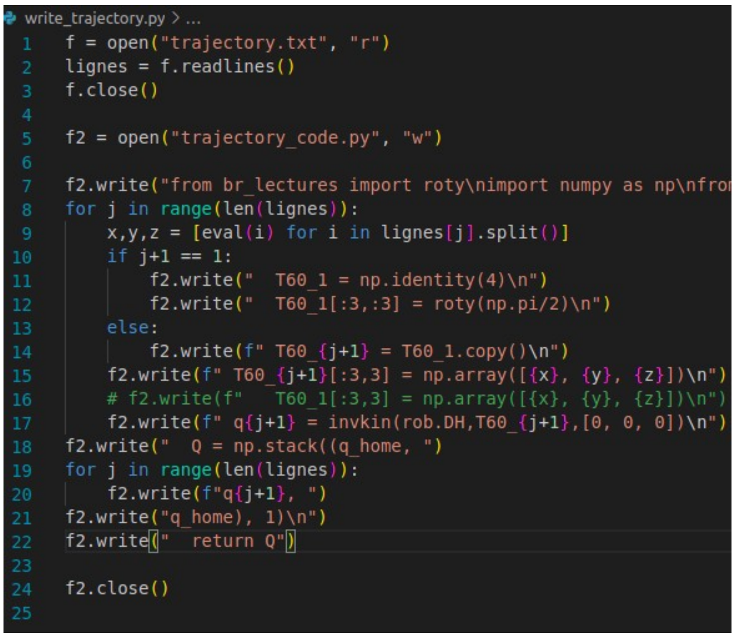
To be more convenient and easy to generate the trajectory, I implemented a function **"write_trajectory.py"** which, given a set of points in the file **"trajectory.txt",** implement a function in the file **"trajectory_code.py"** that return the **Q** variable used in **hocook** function.



trajectory.txt

```
1 -0.15 0.6
1 -0.1 0.85
1 0 0.9
```

Fig1 : Set of points



write_trajectory.py > ...

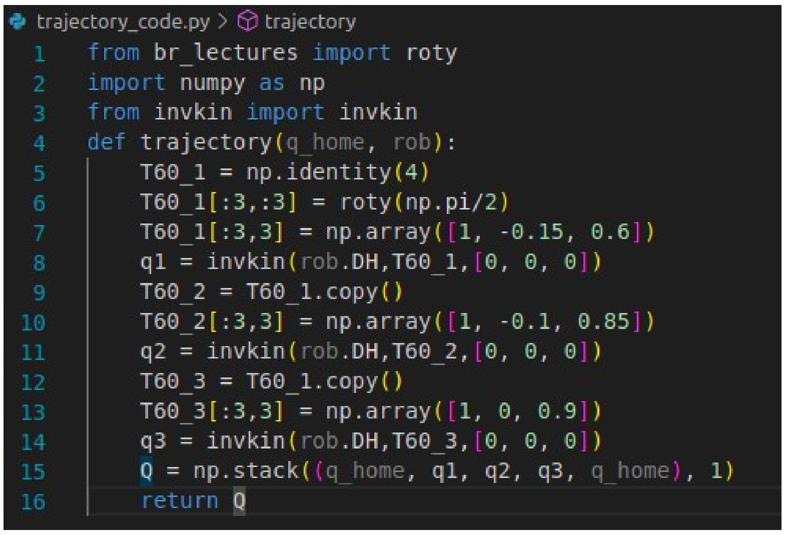
```python
f = open("trajectory.txt", "r")
lignes = f.readlines()
f.close()

f2 = open("trajectory_code.py", "w")

f2.write("from br_lectures import roty\nimport numpy as np\nfro
for j in range(len(lignes)):
    x,y,z = [eval(i) for i in lignes[j].split()]
    if j+1 == 1:
        f2.write("  T60_1 = np.identity(4)\n")
        f2.write("  T60_1[:3,:3] = roty(np.pi/2)\n")
    else:
        f2.write(f" T60_{j+1} = T60_1.copy()\n")
    f2.write(f" T60_{j+1}[:3,3] = np.array([{x}, {y}, {z}])\n")
    # f2.write(f"   T60_1[:3,3] = np.array([{x}, {y}, {z}])\n")
    f2.write(f" q{j+1} = invkin(rob.DH,T60_{j+1},[0, 0, 0])\n")
f2.write("  Q = np.stack((q_home, ")
for j in range(len(lignes)):
    f2.write(f"q{j+1}, ")
f2.write("q_home), 1)\n")
f2.write("  return Q")

f2.close()
```

Fig2 : "write_trajectory.py"



trajectory_code.py > trajectory

```python
from br_lectures import roty
import numpy as np
from invkin import invkin
def trajectory(q_home, rob):
    T60_1 = np.identity(4)
    T60_1[:3,:3] = roty(np.pi/2)
    T60_1[:3,3] = np.array([1, -0.15, 0.6])
    q1 = invkin(rob.DH,T60_1,[0, 0, 0])
    T60_2 = T60_1.copy()
    T60_2[:3,3] = np.array([1, -0.1, 0.85])
    q2 = invkin(rob.DH,T60_2,[0, 0, 0])
    T60_3 = T60_1.copy()
    T60_3[:3,3] = np.array([1, 0, 0.9])
    q3 = invkin(rob.DH,T60_3,[0, 0, 0])
    Q = np.stack((q_home, q1, q2, q3, q_home), 1)
    return Q
```

Fig3 : file written by "write_trajctory.py"

2.Display result

I chose to change the board to have the x-axis as the normal. So the board is not on the ground, it is in front of the robot, and I changed the **planecontact** function.

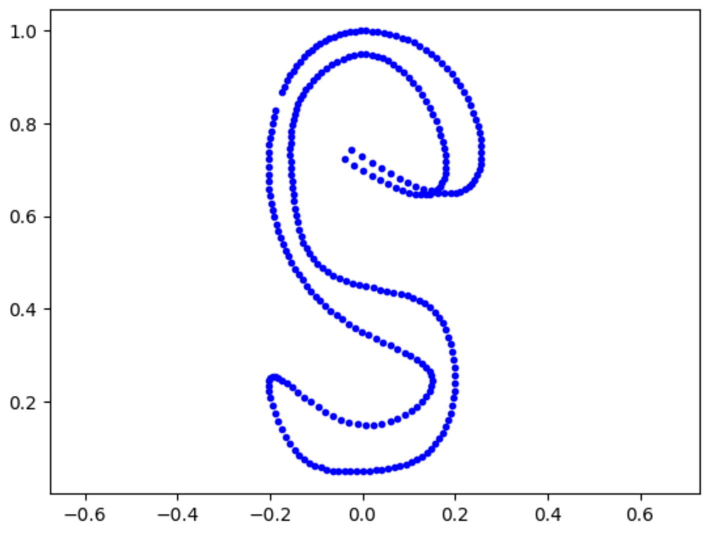


Fig4 : result with the planecontact function

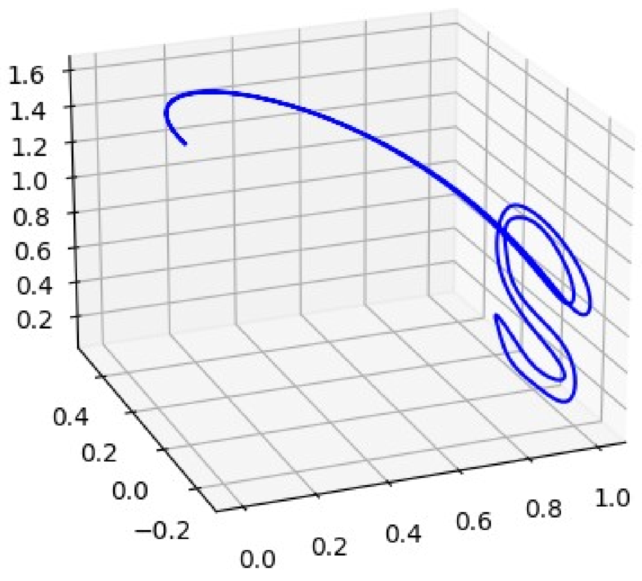


Fig5 : trajectory in 3D

3.Values, velocities, accelerations of joint variables

For my robot and the trajectory I wanted, I had to change the limite of the acceleration. I also raised the velocity to be more convenient for the tests.
I also tried with values as minimum as possible, and it was working for a velocity not changed and an acceleration multiplied by 2,1.
Here, the velocity was multiplied by 2, and the acceleration by 100.

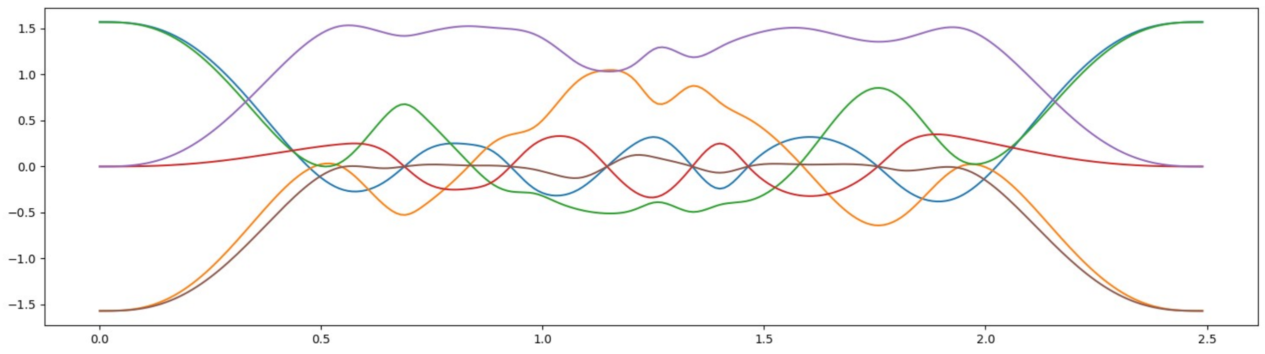


Fig6 : Values

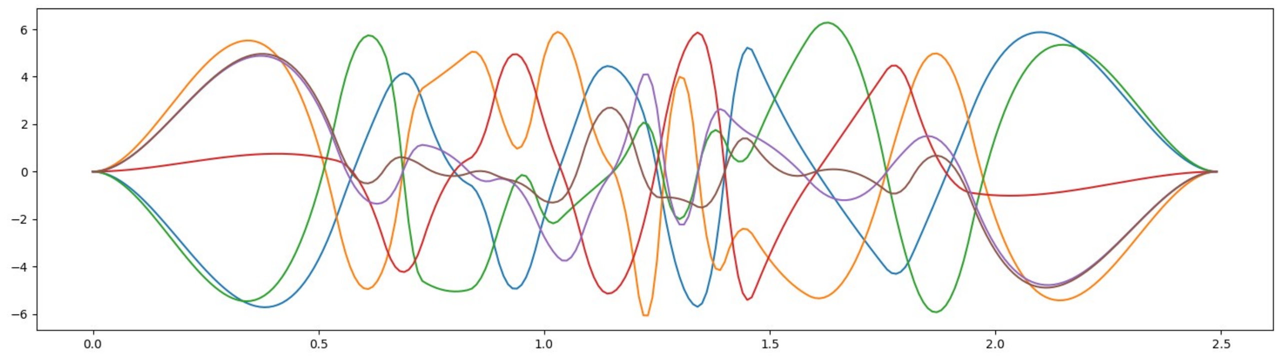


Fig7 : Velocities

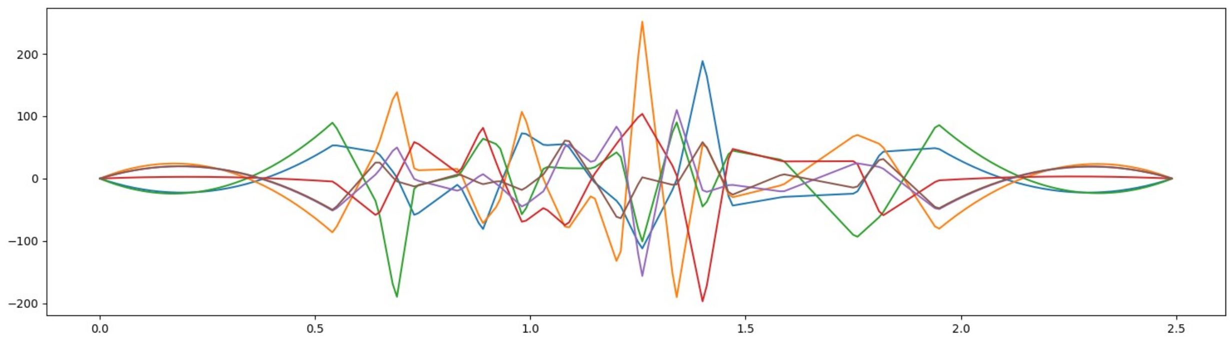


Fig8 : Accelerations

Comments :
The values, their first derivative and their second derivative are all continous. The minimum requirement for the trajectory to be smooth are completed.

However, the third derivative is not continous, we can see some peaks in the second derivatives. It could be a requirement depending on what we need.