Industrial & Collaborative Robotics Project

Giovanardi Simone - 275785@studenti.unimore.it - ID:195442

1 Parametrization

In order to create the parametrization of the project, my registration number is used. Since my ID is $N = 195442$, ordering the first five digits from the most significative to the least significative, the list of parameter is obtained:

$$C_1 = 1;$$
$$C_2 = 9;$$
$$C_3 = 5;$$
$$C_4 = 4;$$
$$C_5 = 4.$$

2 Project_1

2.1 Project_1.1

For the project, the manipulator Universal UR5 was considered and the goal was to pass from the initial configuration $C_{\text{start}} = (0.0, 0.0, 0.0, 0.0, 0.0, 0.0)^{\text{T}}$ to the final configuration $C_{\text{goal}} = \frac{1}{2\pi}(c1, c2, c3, c4, c5, c1)^{\text{T}}$ in 2 seconds where the joint velocities are zero for both initial and final configuration. In order to achieve the goal, the Matlab code was generated. As first, C_{start} and C_{goal} was defined, considering them as a row vector 1x6 instead of a column vector, because otherwise in the following steps some errors due to computations occurred; also the robot was loaded:

```
startConfiguration = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0];
goalConfiguration = 1/(2 * pi)[c1, c2, c3, c4, c5, c1];
robot = loadrobot("universalUR5");
robot.DataFormat = "row";
```

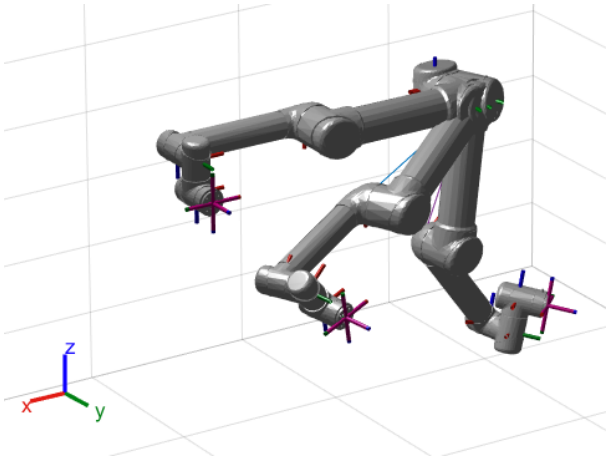

Figure 1: Robot posture evolution.

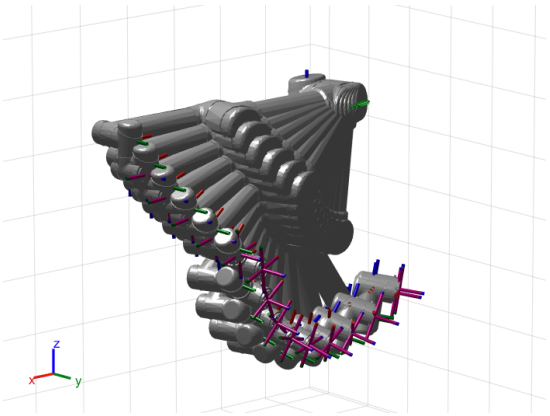

Figure 2: More dense Robot posture evolution.

For the purpose of generate a trajectory for each joint, the preliminary step of the motion planning was performed, using the RRT algorithm to find a sequence of configuration (path) to be followed. The choice of this algorithm is due to the fact that is computationally fast and easy to implement. An other idea could be to use directly a trapezoidal trajectory without using the motion planning RRT, since this case requires to reach a single configuration and no obstacles are present, but the methodology chosen is stronger and more usable for many other cases, comprised that in Project 1.2 .

```
rrt = manipulatorRRT(robot,{}, "IgnoreSelfCollision",true,
      MaxConnectionDistance=1);
rng(0);
path = plan(rrt,startConfiguration,goalConfiguration);
```

In the manipulatorRRT() the robot is inserted, with the additional information that collisions between joints must be ignored, because otherwise, an error of auto-collision occurred in the C_{goal} configuration. The ConnectHeuristic inside the manipulatorRRT() object was kept enabled and a maximum connection distance equal to 1 was chosen, in order to have a configuration approximately in the middle (looking at the Robot posture in Figure 1) with respect to C_{start} and C_{goal}. Figure 2 shows a more dense representation to better understand the evolution. Deactivating the heuristic, too many configuration were generated. A fixed seed was used to have a reproducible code. The plan() function is used to generate the path, having the position of each joint in each configuration (where a joint is represented by a column):

```
path =
    0         0         0         0         0         0
0.3907    0.5038   -0.2315    0.5132    0.1643   -0.4996
0.1592    1.4324    0.7958    0.6366    0.6366    0.1592
```

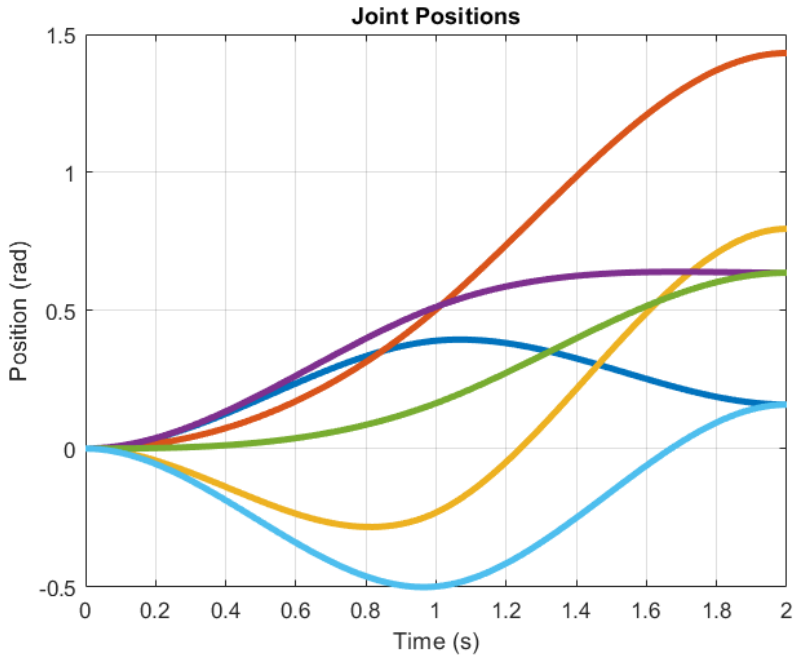

Figure 3: Joints positions evolution in time.

Now the trajectory was created, using a spline function, which was preferred to other methods such as trapezoidal trajectories, since there is a "mid" configuration in which the robot has to pass through and in this way the outcome will be smoother. The spline() function is used in a for loop, for generating the trajectory for each joint, keeping as zero the boundaries in order to have a null velocity in those points; ppval() provides the values of the joint position at each evaluation time tt.

```
tf = 2;
t = linspace(0, tf, size(path, 1)); % Original time points
tt = 0:0.02:tf; % Evaluation time points

pp = spline(t, [0, path(:, j).', 0]);
q(:, j) = ppval(pp, tt);
```

Joint positions trajectories are now generated and showed in Figure 3, noting that the final values are as expected and that two pairs of joints have the same position, since the ID number provided the same number for them.

The final step was to create the velocity and acceleration profiles:

```
dpp = fnder(pp, 1);
ddpp = fnder(pp, 2);
dq(:, j) = ppval(dpp, tt);
ddq(:, j) = ppval(ddpp, tt);
```

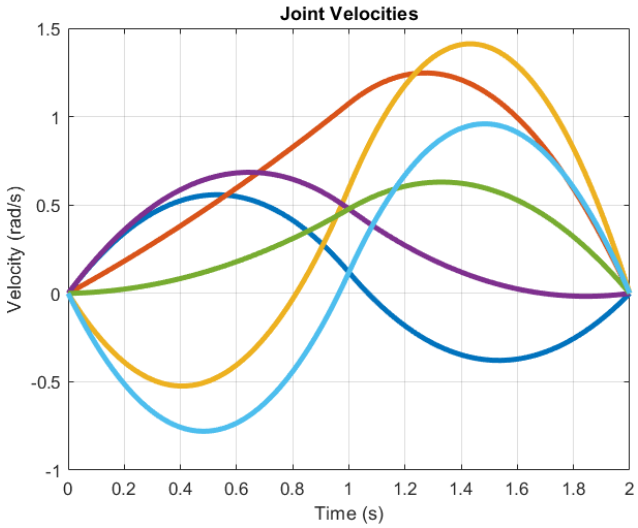

Figure 4: Joints velocities evolution in time.

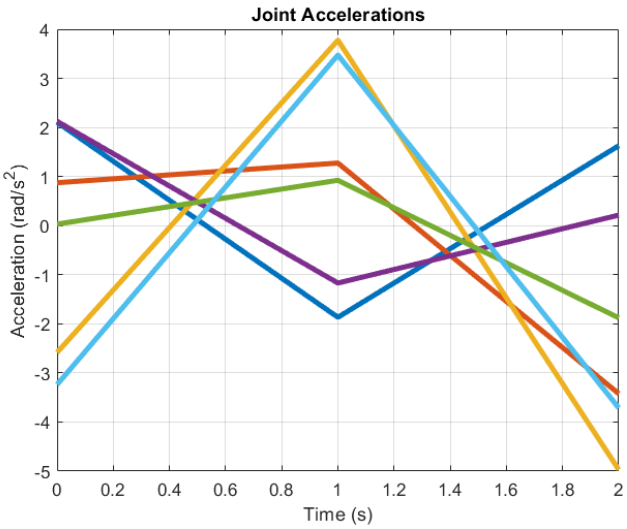

Figure 5: Joint accelerations evolution in time.

The fnder() was used to derive the position curve for each joint and the ppval() used again to discretize the curve and investigate it in a more dense pattern. Figure 4 and Figure 5 show the velocity and accelerations profiles respectively for each joint. It is possible to see that velocities are zero when the Robot reaches C_{goal} as wanted and the accelerations are not zero at the beginning and at the end. Since they are not required to be null, these discontinuities are maintained.

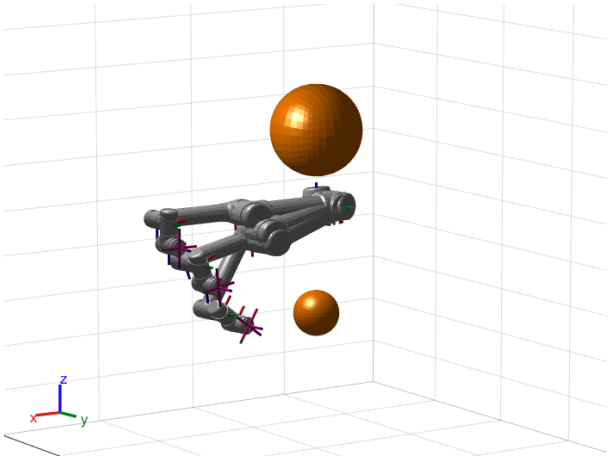

Figure 6: Robot posture evolution.

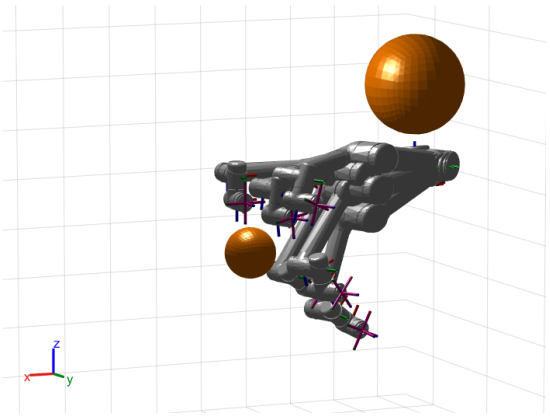

Figure 7: Robot posture evolution with moved obstacle.

2.2 Project_1.2

Now, the goal differs with respect to the previous case because the goal configuration is $C_{\text{goal},2} = \frac{1}{2\pi}(c1, c1, c3, 0, 0, 0)^{\text{T}}$ and must be reached in 3 seconds starting from the same C_{start} as before; the same applies for velocities (zero at C_{start} and $\text{C}_{\text{goal},2}$). Also two spherical obstacles are now present, where the first has a radius $r_1 = 0.2$ and center in [0, 0, 0.4] and the second is centered in [0, 0, −0.4] with radius $r_2 = 0.1$. The trajectory problem is solved exactly with the same procedure as the previous, hence with a RRT motion planning and a spline trajectory tracking; In Figure 6 is showed the Robot with the initial, final and generated configurations. Just to test the performance with different obstacles, the smaller obstacle was moved in a position comprised between the C_{start} and the and $\text{C}_{\text{goal},2}$ and indeed more configurations were generated in order to avoid the collision with it (Figure 7). Different from the previous exercise is the choice of the maximum connection distance in order to perform a shorter configuration changing (the heuristic is still enabled):

```
rrt = manipulatorRRT(robot,{}, "IgnoreSelfCollision",true,
      MaxConnectionDistance=0.5);
```

With the same discretization and the same procedure, the splines were generated, but with the only difference that now the final time is $T_{\text{F}} = 3$. In Figure 8 the position trajectory for each joint is plotted. It can be seen that even if three joint doesn't have to change position from C_{start} to $\text{C}_{\text{goal},2}$, the curve profile is not constant. It could be forced to be imposed fixed, but is not a good idea, since it would no longer be able to generalize its behavior for a more general case. The other goal positions are as expected. Also the velocity profile is plotted (Figure 9) and as wanted the velocity is zero at T_{F}. Figure 10 shows accelerations.

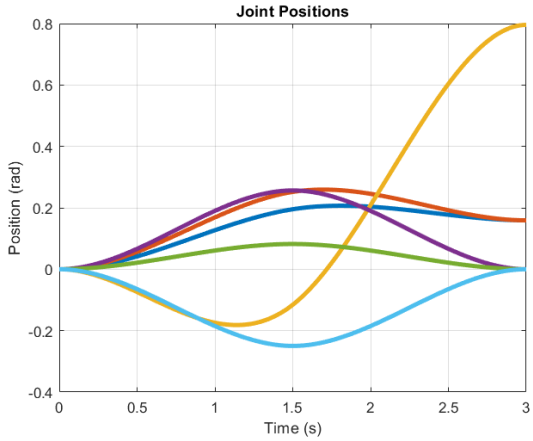

Figure 8: Joints position evolution in time.

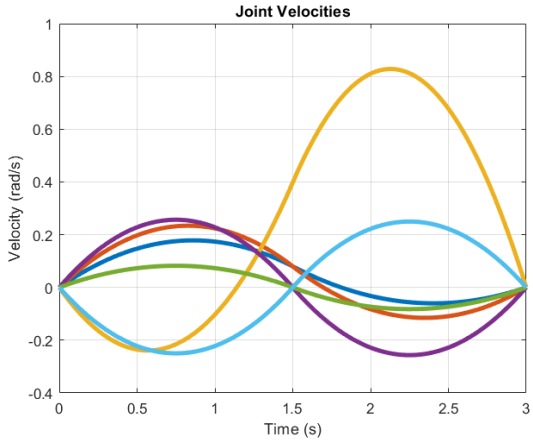

Figure 9: Joints velocity evolution in time.

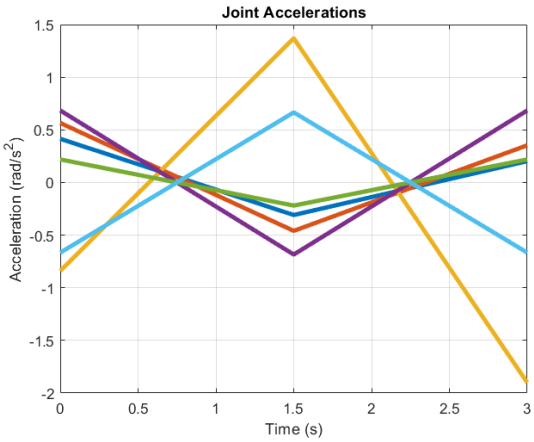

Figure 10: Joints acceleration evolution in time.

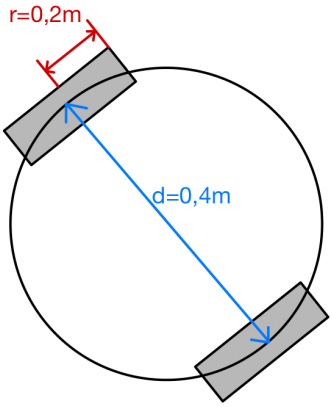

Figure 11: Schematized mobile Robot.

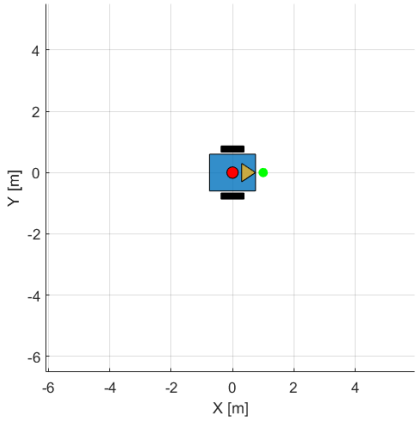

Figure 12: Mobile Robot in the start configuration.

3 Project_2

The mobile Robot that is requested to control is that schematized in Figure 11. Starting from the configuration $q_{\mathrm{i}} = (x_{\mathrm{i}}, y_{\mathrm{i}}, \theta_{\mathrm{i}}) = (0, 0, 0)$ showed in Figure12, the trajectory to be followed, is:

$$x_{\mathrm{d}} = c_1 cos(c_2 t) = 1cos(9t), \qquad y_{\mathrm{d}} = c_1 sen(c_2 t) = 1sen(9t)$$

Task 1

In order to allow to the Robot to track the desired trajectory, the IO-SFL (pure pursuit) algorithm is implemented and explained in the following. At first, some parameters parameters have been set:

```
c1 = 1;
c2 = 9;
simulationStepTime = 1/25;
simulationTime = 10.0;
time = 0.0;
q = [0.0; 0.0; 0.0];
k = [20 0; 0 20];
b = 0.5;
```

Where q is q_{i}, k is the diagonal 2x2 state matrix and b is the distance between the robot and the virtual point B to be followed. After that, the real algorithm was implemented as a while loop (the next block of code is not explained since the algorithm is assumed to be known).

```
while time < simulationTime
    % Desired positions and velocities
    x_des = c1*cos(c2*time);
    y_des = c1*sin(c2*time);
    x_des_dot = -c2*(c1*sin(c2*time));
    y_des_dot = c2*(c1*cos(c2*time));

    % Current position
    x = q(1);
    y = q(2);
    theta = q(3);

    % Virtual point (xB, yB)
    xB = x + b*cos(theta);
    yB = y + b*sin(theta);

    % Calculate vdx and vdy
    vdx = x_des_dot + k(1,1)*(x_des - xB);
    vdy = y_des_dot + k(2,2)*(y_des - yB);

    % Calculate control inputs [v; omega]
    T_inv = [cos(theta), sin(theta); -sin(theta)/b, cos(theta)/b];
    u = T_inv*[vdx; vdy];

    % Update model
    G = [cos(theta), 0.0; sin(theta), 0.0; 0.0, 1.0];
    q = q + G*u*simulationStepTime;

    % Advance simulation
    time = time + simulationStepTime;
end
```

Some attempts were made in order to chose properly k values and the distance b with an iterative procedure and k = [20 0 ; 0 20], $b = 0.5m$ was the best choice since the trajectory tracking was very good.

Task 2

Since the information relating the linear and the angular velocities are known from the code, the angular velocities for both wheels can be easily computed, also knowing the

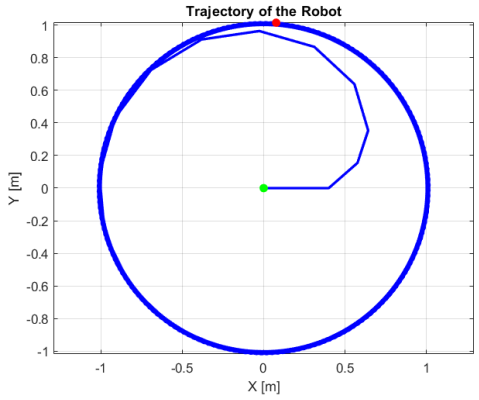

Figure 13: Mobile Robot position evolution in time.

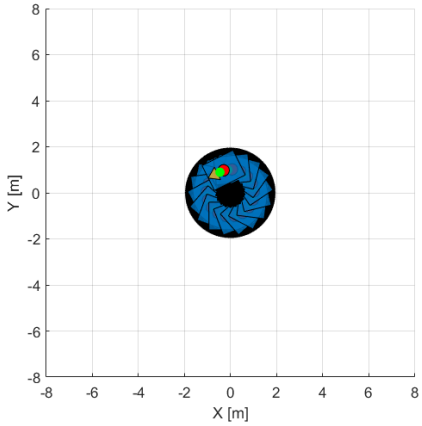

Figure 14: Mobile Robot evolution in time.

distance between the wheels and the radius of them, with the relation:

```
w = [1/r, -(d/r); -(1/r), -(d/r)]*u
```

Obviously since the trajectory is circular, the two velocities reach a constant value after a transient, which is:

```
w =
    27.5941    % wr
   -63.5941    % wl
```

It can be seen that these values are not significantly affected by the choice of k and b, varying them in a range in which the trajectory is tracked in a good way.

Task 3

The cartesian positions of the Robot can be plotted and it can be seen in Figure 13, considering the time evolution starting from the starting point q_i until the point reached at the simulation ending (the circumference is crossed many times). Figure 14 shows the positions that the robot has taken in the evaluated instances of time.