Robot Bartender

Robotics and Human-Machine Interfaces Project

Master's Degree in Embedded Computing Systems | A.Y. 2016/2017

Giulia Ferri

1 SOMMARIO

2 Scope 2
3 System Overview 2
3.1 Manipulator 2
3.2 Environment 3
4 Task of manipulator 3
5 Path Planning 4
5.1 Artificial Potential Field 4
5.1.1 Attraction Potential Function 5
5.1.2 Repulsive Potential Function 6
5.1.3 Total Potential 7
5.1.4 Defects of Potential Field 7
5.1.5 Planning Technique 8
6 Joints Space Trajectory Generation 12
6.1 Implementation 14
6.1.1 Non-optimized procedure 16
6.1.2 Optimized procedure 16
6.2 Comparison of joints Space trajectory generation methods 18
6.2.1 Inverse Kinematic 18
6.2.2 Procedure not optimized for collision avoidance 19
6.2.3 Procedure optimized for collision avoidance 20
6.2.4 Manipulability 23
7 Use Cases 23
7.1 Graphic Interfaces 24
8 Conclusion 25
9 References 25

2 SCOPE

The aim of the project is to simulate a bartender robot that assembles, mixes and serves cocktails to customers, knowing the recipes and the positions of the ingredients.

3 SYSTEM OVERVIEW

The project is implemented with the Peter Corke's Robotics Toolbox for MATLAB that provides many useful functions to develop and simulate classical arm type robotics, for example kinematics, dynamics, and trajectory generation.

3.1 MANIPULATOR

The manipulator is composed by 8 joints of which one prismatic and seven revolute.

It has 8 DOFs (Degrees of Freedom), and since the 3D space has only 6 DOF (three positional, and three angular) it has two DOFs redundant.

Robot (8 axis, PRRRRRRR)					
j	**theta**	**d**	**a**	**alpha**	**qlim**
1	0	q1	0	-1.571	[-1.500 , 0.300]
2	q2	0	0	-1.571	[-1.221 , 4.363]
3	q3	0	0	1.571	[-6.283 , 6.283]
4	q4	0	0.7	0	[-3.927 , 4.363]
5	q5	0	0.02	-1.571	[-4.101 , 1.047]
6	q6	0.7	0	1.571	[-6.283 , 6.283]
7	q7	0	0	-1.571	[-6.283 , 6.283]
8	q8	0	0	1.571	[-3.142 , 3.142]

Table 1 - Denavit-Hartenberg parameters of the robot and joints limits.

The limits of the joints are chosen to avoid collision with each other during the execution of the task for which the robot is created.

The robot is designed as hanging from the ceiling. For this reason, the robot is located on a base which moves the initial pose of the robot:

$$base = \begin{bmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2 ENVIRONMENT

The work environment consists of a bar table. On it are placed the bottles, the shaker and the glass in which the cocktail will be served to the customer.

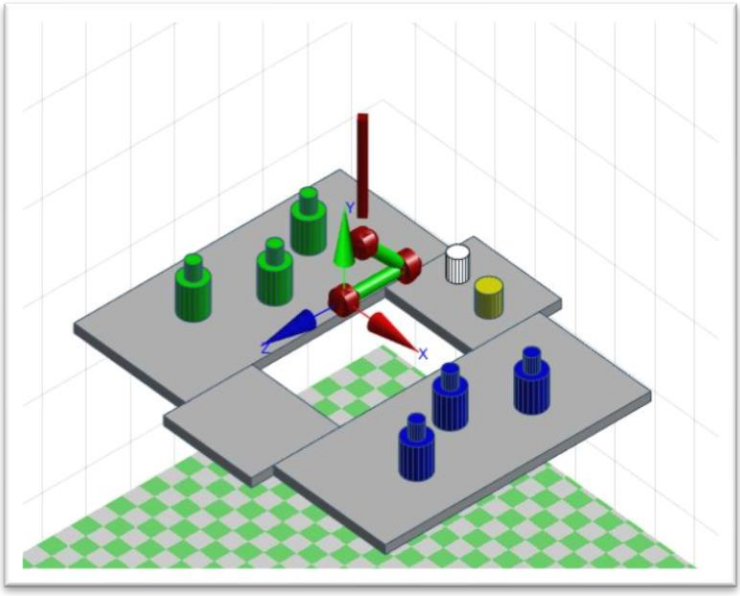

Figure 1 - Robot and environment

4 TASKS OF MANIPULATOR

The tasks that the robot should perform consists in:

- for each ingredient, take the right bottle, pour the content into the shaker and then put the bottle back in its place;
- mix the ingredients;
- pour the result obtained into the glass;
- bring the glass to the final position where you are supposed to find the customer.

Obviously, it is important that during path the robot avoids obstacles, therefore all the other objects in the environment except the goal that must reach.

Furthermore, during transport of the item, its content shall be not spilled. For this reason, it is decided that the correct pose for the end effector is with the y-axis upwards. When an object is taken it need to verify that the z axis is facing the target.

5 PATH PLANNING

To create the path that the robot shall follow, is possible to exploit the ***ctraj*** function of the toolbox since the task of the robot is to bring the end effector from a given initial position to a final one with a predetermined orientation.

The ***ctraj*** function computes a Cartesian trajectory (4x4xN) from pose T0 (initial transformation matrix of end-effector) to T1 (final transformation matrix of end-effector) with N points that follow a trapezoidal velocity profile along the path.

However, this method operates under the simplifying assumption that the workspace is empty. In presence of obstacles, it is necessary to plan the motion that enable the robot to execute the assigned task without colliding with them.

So, to determinate of a collision avoidance path of end-effector, starting from the robot position to the targeted goal, is used the planning via ***Artificial Potential Field***(APF). The concept about APF is to find a mathematical function to represent the energy of the system based on the idea of physical rules in potential fields. APF consists of two fields: attractive field generated by the goal and repulsive field by each of the obstacles. The influence of potential field U made by the combination of attractive and repulsive potential control the motion of the robot in a safer path while keeping it away from the obstacles.

The ***ctraj*** function is used only to interpolate the rotation trajectory of end-effector.

5.1 ARTIFICIAL POTENTIAL FIELD

The general APF equation is

$$U(p) = U_{attr}(p) + U_{rep}(p)$$

where p is the end-effector position, $U_{attr}(p)$ is the attractive function and $U_{rep}(p)$ is the repulsion function.

When the robot is immersed in the potential field, attractive force and repulsive force guide the robot towards the goal point.

The attractive force is generated between the robot and the goal and it is responsible for attracting the robot to the goal. The repulsive force is between the robot and the obstacles. Its main function is to avoid them.

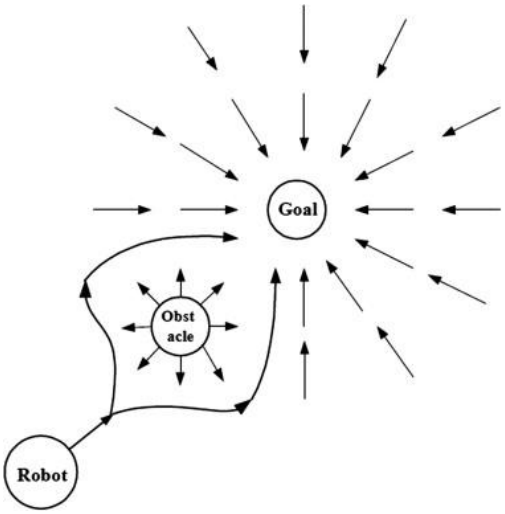

Figure 2 - Artificial Potential Field

5.1.1 Attraction Potential Function

The Attractive Potential Function is divided in two terms, conical potential and Quadratic potential. The conical potential is used when the robot is far away from the goal. On the other hand, the quadratic potential is used when the robot is near to the goal. The reference that will define whether the robot is far or near is the term d_{th}.

$$U_{attr}(p) = \begin{cases} \frac{1}{2} k_a d(p, p_{goal})^2, & d(p, p_{goal}) < d_{th} \\ k_a d(p, p_{goal}), & d(p, p_{goal}) \geq d_{th} \end{cases}$$

where p is the end-effector position variable, p_{goal} is the goal position in the end-effector frame, $d(p, p_{goal})$ is the distance function (Euclidean distance), and $k_a$is the attractive scaling factor.

This function represents the potential that affect the robot while the force that will drive the robot to reach the goal will be generated from the negative gradient of this function.

$$F_{attr}(p) = -\nabla U_{attr}(p)$$

As:

$$F_{attr}(p) = \begin{cases} k_a d(p, p_{goal}), & d(p, p_{goal}) < d_{th} \\ k_a \frac{(p_{goal} - p)}{d(p, p_{goal})}, & d(p, p_{goal}) \geq d_{th} \end{cases}$$

5.1.2 Repulsive Potential Function

There is always one goal at a time for the robot, but the obstacles are more than one. That is why, the repulsive potential function consists of all the repulsive fields of each obstacle exists in the environment. Every obstacle has a specific limited region that has a repulsive field, so that when the robot comes in that region, it will be repelled from that obstacle.

The latter are represented by a cloud of points as it is possible to see in figure below. The distance between end-effector and each obstacle is

$$d(p, Obstacle_i) = \min_{point \in Obstacle_i} d(p, point)$$

namely, the minimum distance between end effector position and all points of the i-th obstacles.

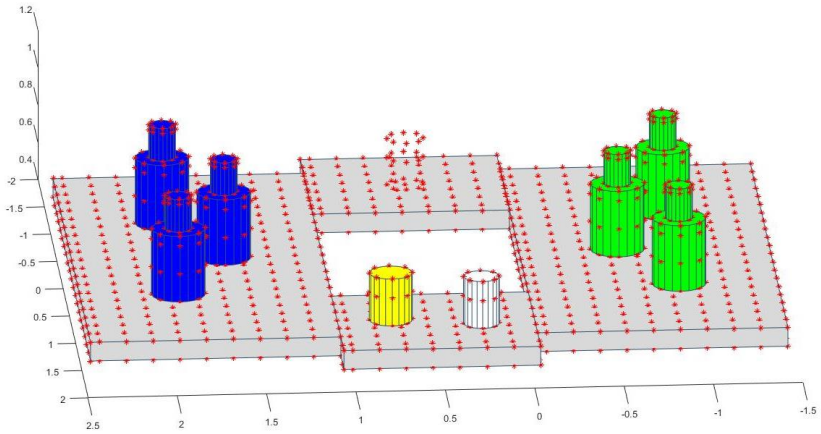

Figure 3 - Representation of obstacles

The repulsive field for one obstacle is:

$$U_{rep,i}(p) = \begin{cases} \frac{1}{2} k_r \left(\frac{1}{d(p, Obstacle_i)} - \frac{1}{d_{th}} \right)^2, & d(p, Obstacle_i) < d_{th} \\ 0, & d(p, Obstacle_i) \geq d_{th} \end{cases}$$

where $d(p, Obstacle_i)$ is the distance to the obstacle, k_r is the scaling factor, and i represent the order number of the current obstacle and d_{th} is the range of influence.

The repulsive force would be represented as:

$$F_{rep,i}(p) = -\nabla U_{rep,i}(p)$$

and

$$F_{rep,i}(p) = \begin{cases} \frac{k_r}{d(p, Obstacle_i)^2}\left(\frac{1}{d(p, Obstacle_i)} - \frac{1}{d_{th}}\right)\nabla d(p, Obstacle_i), & d(p, Obstacle_i) < d_{th} \\ 0, & d(p, Obstacle_i) \geq d_{th} \end{cases}$$

The total repulsive function for n number of obstacles is:

$$U_{rep}(p) = \sum_{i=1}^{n} U_{rep,i}(p)$$

5.1.3 Total Potential

The total potential is obtained by the superposition of the attractive and the aggregate repulsive potentials:

$$U_{Tot}(p) = U_{att}(p) + U_{rep}(p)$$

This results in the force field:

$$F_{tot}(p) = -\nabla U_{Tot}(p) = F_{attr}(p) + \sum_{i=1}^{n} F_{rep,i}(p)$$

5.1.4 Defects of Potential Field

The technique of artificial potential field has its own disadvantages. Some problems that are inherent in the application of potential fields are:

- ***Local Minima***: robot can be in a position where the potential field is zero, which means that the mobile system is blocked in local minimum.

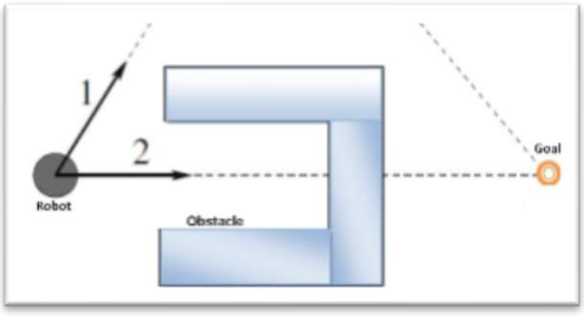

Figure 4 - Local Minima problem

- ***Problem of Closely Spaced***: This situation is like the previous, if two obstacles are placed nearby, such as a door, the repulsive forces of each obstacle are combined into one repulsive force that points out of the opening between obstacles

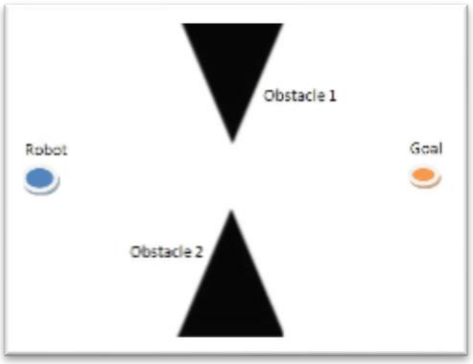

Figure 5 - Problem of closely spaced

- ***Oscillations Problem***: sum of force is calculated by summing components of the environmental Impact. However, this situation causes the oscillations in the path of the robot.

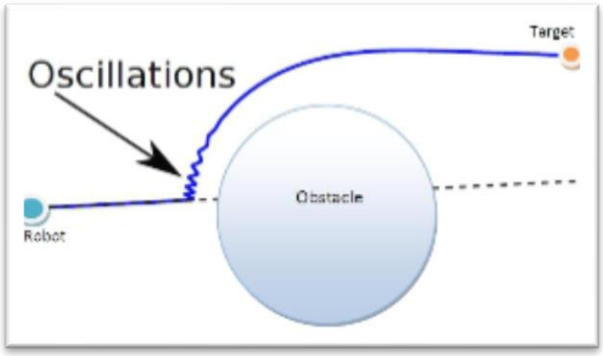

Figure 6 - Oscillation Problem

5.1.5 Planning Technique

To plan collision free motions, the total artificial potential and the associated force field is interpreted as a desired velocity potential for the end effector with direction and module influenced in an attractive way by goal and repulsive way by the obstacles.

$$\dot{p} = F_{tot}(p)$$

Therefore, the next step of end-effector is compute by numerical integration of before equation via Euler method:

$$p_{k+1} = p_k + T\, F_{tot}(p_k)$$

where p_k and p_{k+1} represent respectively the current and the next end effector position and T is the integration step choice equal to 1.

The result is a points trajectory that avoid obstacles and reaches the target as desired.

The major defect founded for this algorithm due to the environment used, is the oscillations as showed in Figure 12. In this case, it is accentuated also by the representation of the obstacles made by points. In fact, it is possible that in two consecutive steps the computation of the distance between end-effector and the same obstacle select two different points. As defined before the distance is the minimum distance between the end effector and every point of the obstacle thus the repulsive force is given by the chosen point. Caused by this, the algorithm may lead for example to an upward and downward movement of the end effector.

To improve the path and avoiding this problem, the data obtained are processed by data fitting algorithm in which a "smooth" function is constructed. For this purpose, the algorithm creates polynomial curve that approximate data points through the MATLAB function ***polyfit*** and extract the new points for the end effector path again via ***polyval*** function.

```
%% fitData.m
function [newPath] = fitData(data)
    sd=size(data);
    t=linspace(0,1,sd(1))';
    t2=linspace(0,1,500)';
    polyOrder=[10 10 10];
        xcoefs=polyfit(t,data(:,1),polyOrder(1));
            x=polyval(xcoefs,t2);
        ycoefs=polyfit(t,data(:,2),polyOrder(2));
            y=polyval(ycoefs,t2);
        zcoefs=polyfit(t,data(:,3),polyOrder(3));
            z=polyval(zcoefs,t2);
    newPath = [x';y';z'];
    % delete points too close
                                i = 2;
    while i < size(newPath,2)
        if norm(newPath(:,i-1)-newPath(:,i)) < 0.02
            newPath = horzcat(newPath(:,1:i-1),
                                    newPath(:,i+1:end));
        else
            i=i+1;
        end
    end
end
```

Figure 7 - Data fitting code

For orientation of end effector, the interpolated trajectory of the ***ctraj*** function is maintain since this does not create issues for collision with obstacles.

Since that the contents shall not be spilled during trajectory it is established that throughout the trajectory the y-axis of the end effector is facing upwards, i.e.

directed as z-axis of the base frame. To do this it is necessary that the second column of the rotation matrix is always [0; 0; 1]. Clearly, this condition is false when the action to be performed is to pour.

5.1.6 Comparison of path generation techniques

The following figures show in various views the comparison of paths computed by:

- ***Ctraj*** function;
- Artificial Potential Field algorithm;
- Data fitting of point obtained by artificial potential field algorithm.

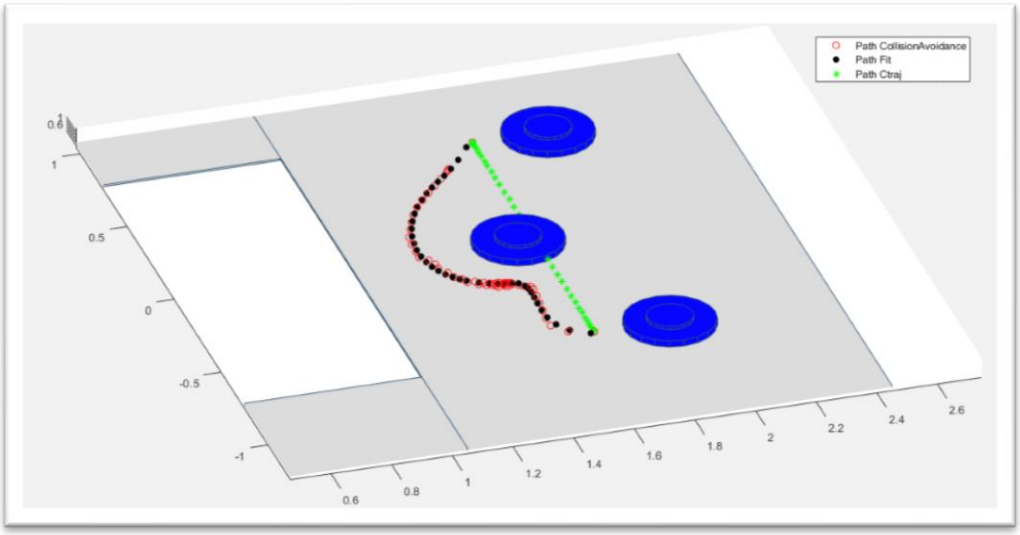

Figure 8 - Comparison of paths generation techniques (1)

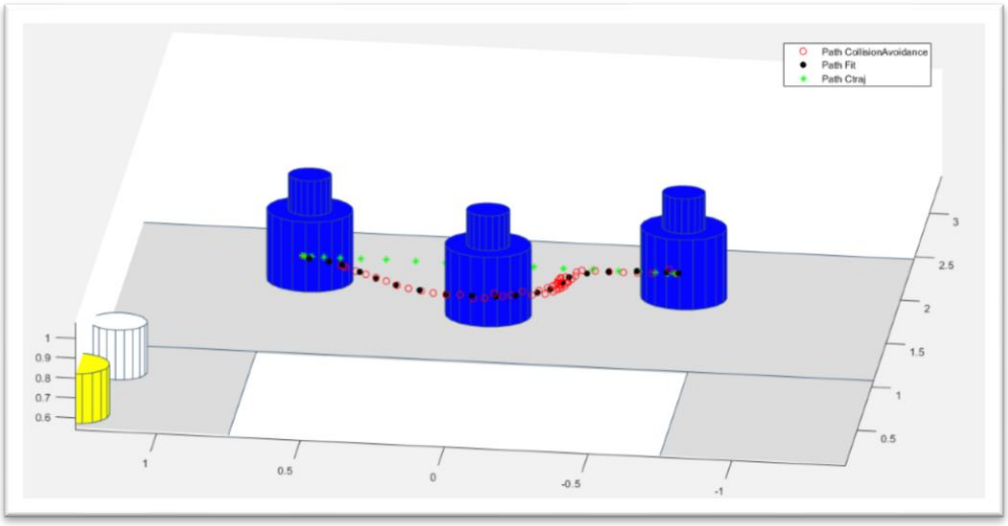

Figure 9 - Comparison of paths generation techniques (2)

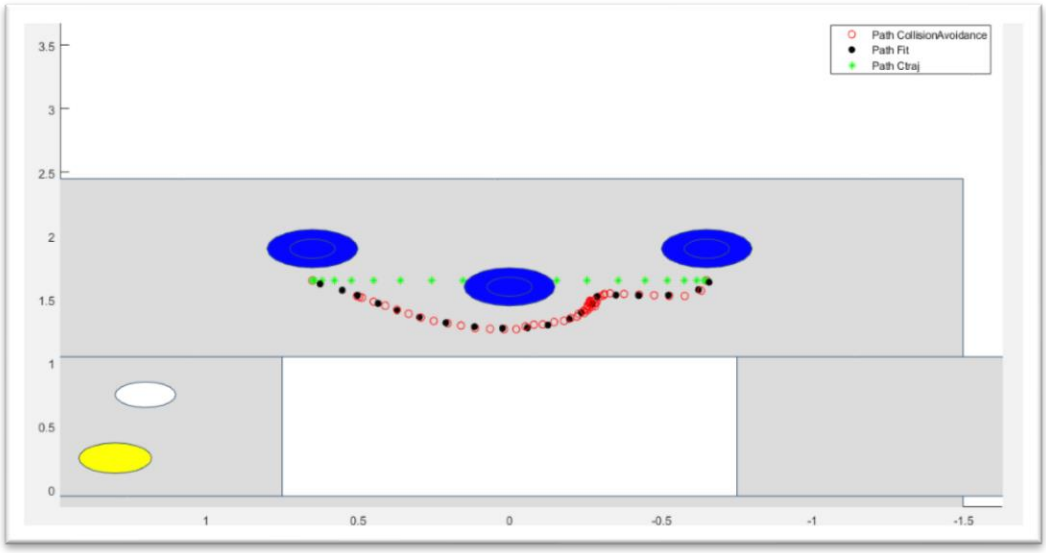

Figure 10 - Comparison of paths generation techniques (3)

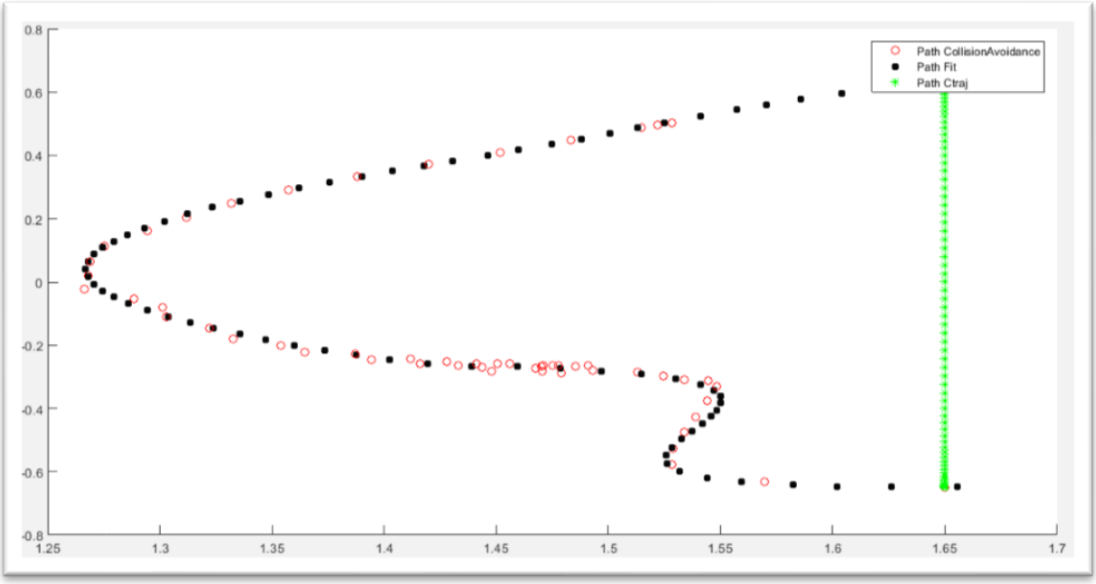

Figure 11 - Comparison of paths generation techniques (4)

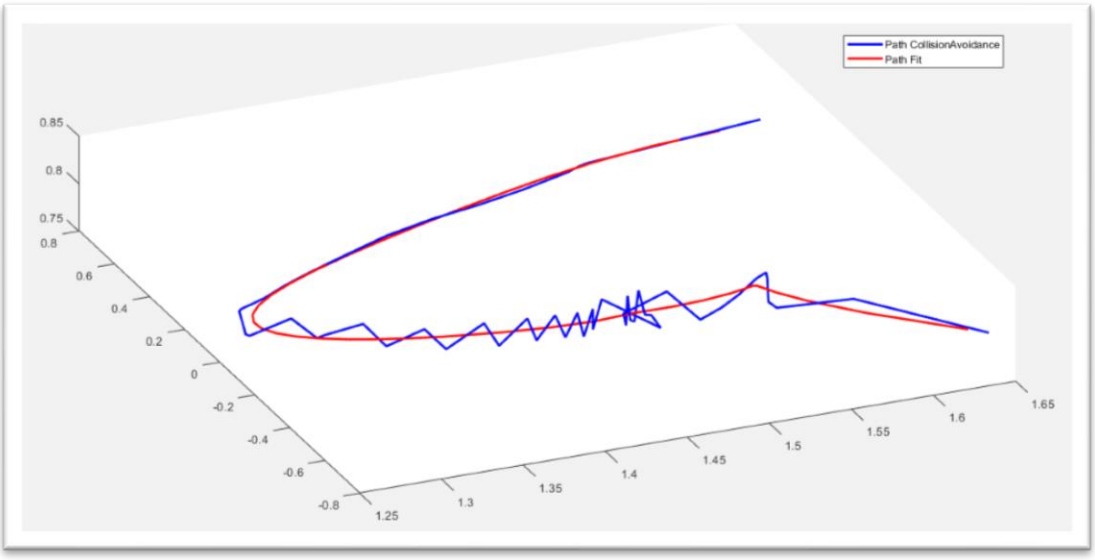

Figure 12 - Union of the points obtained with the collision avoidance function and union of the points obtained after fitting to highlight the problem of oscillations

6 JOINTS SPACE TRAJECTORY GENERATION

Given a motion trajectory of the end-effector the aim is to determine a feasible joint space trajectory $(q(t), \dot{q}(t))$ that reproduces it. The simplest solution would be to use inverse kinematics, but this does not allow to take advantage of redundancy to modify the joints space trajectory and optimize it to maintain the robot away from obstacles or from singularities.

For this, the differential cinematics is used. It gives the way to express the end-effector liner velocity $\dot{p_e}$ and angular velocity w_e as a function of the joints velocities $\dot{q}$. This mapping is described by a matrix termed geometric Jacobian J.

$$\dot{p_e} = J_p(q)\dot{q}$$

$$w_e = J_o(q)\dot{q}$$

Where J_p and J_o are respectively the contribute of joint velocities at liner velocity $\dot{p_e}$ and at angular velocity w_e.

So, the differential kinematics equation is

$$v_e = \begin{bmatrix} \dot{p_e} \\ w_e \end{bmatrix} = J(q)\dot{q} \qquad \text{where } J = \begin{bmatrix} J_p \\ J_o \end{bmatrix}$$

To determine an admissible joints trajectory is exploited the inverse differential kinematics.

The joint velocities can be obtained via inversion of the Jacobian matrix:

$$\dot{q} = J^{-1}(q)v_e$$

In case of a redundant manipulator, this inversion is not possible because the Jacobian is not a square matrix, hence there exists infinite solutions of the differential kinematics equation. A common way to resolve this problem is to find the solutions $\dot{q}$ that minimize the quadratic cost functional of joint velocities:

$$g(\dot{q}) = \frac{1}{2}\dot{q}^T W \dot{q}$$

where W is a symmetric positive definite weighting matrix. This problem can be solved using the *Lagrange multipliers* method, which gives the optimal solution

$$\dot{q} = W^{-1}J^T(JW^{-1}J^T)^{-1}v_e$$

and, if $W = I$ the formula becomes

$$\dot{q} = J^{\dagger}v_e$$

where

$$J^{\dagger} = J^T(JJ^T)^{-1}$$

is the *right pseudo-inverse of the Jacobian*.

Both solutions described above to compute $\dot{q}$ can be used only when Jacobian has full rank. Hence, it becomes meaningless when manipulator is at a singular configuration. A simple solution overcoming the problem of inverting differential kinematics in the neighbourhood of a singularity is by ***damped least square inverse***

$$J^{\dagger} = J^T(JJ^T + k^2 I)^{-1}$$

where k is a damping factor that renders the inversion better conditioned.

However, since the manipulator is redundant, if $\dot{q}^*$ is a solution for differential kinematics equation, also $\dot{q}^* + P\dot{q}_0$ is a solution where $\dot{q}_0$ is a vector of arbitrary joint velocities and P is a projector on the null space of J.

In fact, in redundant manipulator the number of components of the end effector velocities needed to perform the task (r) is less than the DOF of the manipulator (equal to the number of joint, n). In this case, the null space of the Jacobian matrix is not empty and has dimension $n - r$ (except in the case of singularity where dimension of range space decreases while the dimension of null space increases). The existence of a not empty null space allows to exploit the redundant DOF to optimize additional objective functions, through $\dot{q}_0$. In fact, the effect of $\dot{q}_0$ is to generate internal motions of the structure that not change the end effector position and orientation.

In this case, it is necessary to consider a new cost functional

$$g'(\dot{q}) = \frac{1}{2}(\dot{q} - \dot{q}_0)^T(\dot{q} - \dot{q}_0)$$

so that the norm of the vector $(\dot{q} - \dot{q}_0)$ is minimized. The solutions are sought which satisfy the differential kinematics constraint (primary objective) and are as close as possible to $\dot{q}_0$. Hence, the secondary objective function can be specified by $\dot{q}_0$ vector. Solving the optimization problem, it is possible to obtain

$$\dot{q} = J^{\dagger}v_e + (I - J^{\dagger}J)\dot{q}_0$$

where $J^{\dagger}v_e$ is relative to minimum norm joint velocities and $(I - J^{\dagger}J)\dot{q}_0$ called *homogeneous solution*, attempts to satisfy the additional constraint to specify via $\dot{q}_0$.

In general,

$$\dot{q}_0 = k_0\left(\frac{\partial W(q)}{\partial q}\right)^T$$

where $k_0 > 0$ and $W(q)$ is the secondary objective function, that will be maximized locally.

Finally, obtained joint velocities, joints configuration can be computed by integrating velocities over time. This integration can be performed in discrete time by Euler method:

$$q(t_{k+1}) = q(t_k) + \dot{q}(t_k)\Delta T$$

6.1 IMPLEMENTATION

For the main task of manipulator, the optimization used is the distance from an obstacle. This because the path built above only avoids the obstacles with respect to end effector pose.

In order to compute this distance, it is necessary define points on the structure for each step. Since the joints move at each step of the path, to find their position in the base frame is use the composition of transform matrices of all previous joints to the base.

Once that the joint position is found, a cylinder is built on it including the link to the next joint. The control points are obtained from cylinder in the same way as for objects in the environment.

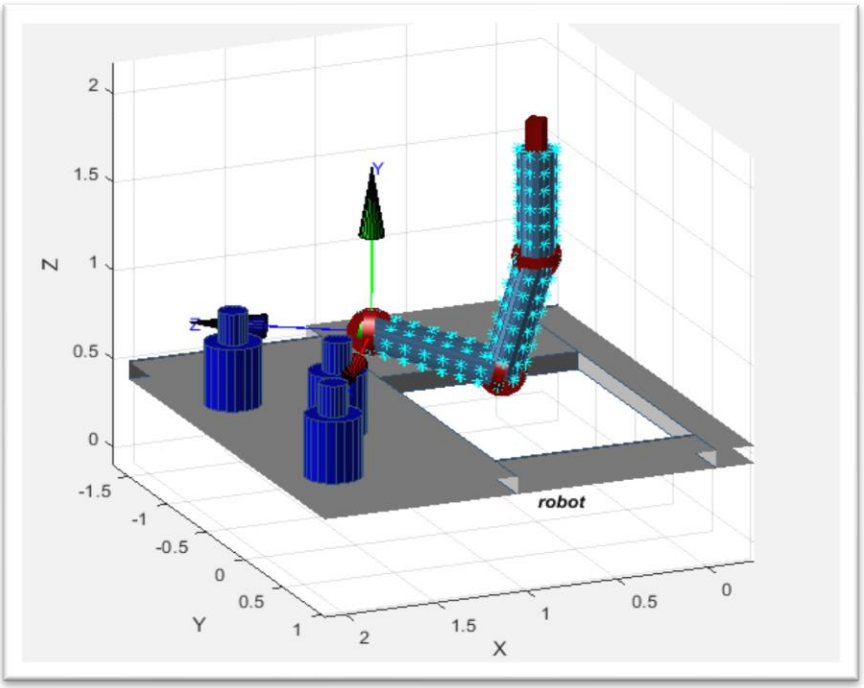

Figure 13 - Cylinders built on the manipulator and the respective control points

Hence, to know the minimum distance of the manipulator from an obstacle, for every joints the minimum distance is computed between every points on this and every point on each obstacle.

$$d(Joint_i, Obstacle_j) = \min_{\substack{p_{joint} \in Joint_i \\ p_{obs} \in Obstacle_j}} d(p_{joint}, p_{obs})$$

where $Joint_i$ is the current configuration of the i-th joint of the manipulator. The result is a distance for each joint and the minimum is taken from these. This is the secondary objective function $W(q)$ which need to be **maximized**.

To create a joint space trajectory that keeps the manipulator step by step far from the obstacles, as first is computed the path with the artificial potential algorithm as described in the previous chapter. Afterwards for each step of path a **non-optimized procedure** is initially performed and then checked on it if the minimum distance manipulator-obstacle is smaller than a threshold. If it happens, it starts again from the last valid configuration and now follows an **optimized procedure**. This is done because the optimized procedure is performed using the MATLAB function ***fmincon*** which can have a very high computational cost. Optimization is strictly performed when necessary.

6.1.1 Non-optimized procedure

The following steps are performed for the unoptimized procedure:

- compute the **velocity vector** v_e for the end effector. Since the path is generated in the Cartesian Space, it is possible to determine velocity subtracting two consecutive position or orientation and dividing by time, to obtain an infinitesimal displacement
$$\frac{p_i - p_{i+1}}{\Delta T}$$
where p_i is the position or orientation of the end effector at the step i, and ΔT is chosen equal to one. For greater precision, it is compute using the toolbox function ***tr2delta (T0, T1)*** that gives the differential motion corresponding to infinitesimal motion from pose T0 to T1 which are homogeneous transformations. As the destination T (T1), the next step of the Cartesian trajectory is taken, whereas as the source T (T0) the current pose of the end effector is taken. This is because there may be an error and the end effector may not have reached exactly the desired pose;
- compute **Jacobian** matrix J through the toolbox function ***jacob0 (q)***;
- compute **damped least square right pseudo-inverse** of the Jacobian $J^{\dagger} = J^T(JJ^T + k^2 I)^{-1}$;
- compute the **joint velocities vector** solution $\dot{q} = J^{\dagger} v_e$;
- compute the **next configuration q** adding the term to the previous joint coordinates $q = q + \dot{q}$;
- **check** if any joint has left the **limits** and if this happen, changes the value with the maximum possible.

At this point it is checked whether the configuration obtained is or is not below the threshold for the minimum distance from the obstacles. If it is below of threshold, the following optimized procedure is performed, otherwise the configuration just computed is retained.

6.1.2 Optimized procedure

For the optimized procedure, instead the following steps were performed:

- compute the **vector q_0** through an optimization problem;
- compute the **joint velocities vector** solution $\dot{q} = J^{\dagger} v_e + (I - J^{\dagger} J)\dot{q}_0$;
- compute the **next configuration q** adding the term to the previous joint coordinates $q = q + \dot{q}$.

Now we analyse in detail the optimization problem used to compute $\dot{q}_0$. The function used, as already said, is ***fmincon*** that finds minimum of constrained nonlinear multivariable function.

The optimization problem set is like:

$$\begin{cases} \min\limits_{x} -f(x) \\ c(x) \leq 0 \end{cases}$$

where $f(x)$ is the objective function and $c(x)$ are nonlinear constrains. Recall that the objective function ($W(q)$) need to be maximized and ***fmincon*** is a minimization function. For this it is necessary the minimum of minus the function which is equivalent to maximum.

```
%% optimazeTrajectory.m

function [q0dot_opt] =
optimazeTrajectory(robot,Ve,J,Jc,q_old,Tnew,pointsCloud,Goal)

    x0 = zeros(robot.n, 1);
    options = optimoptions('fmincon','Display','iter',...
            'Algorithm','active-set', 'FunctionTolerance', 1e-3);
    q0dot_opt = fmincon(
            @(q0dot)obj_function(q0dot,robot,q_old,Ve,J,Jc,pointsCloud,Goal),
            x0,[],[],[],[],[],[],
            @(q0dot)confun(q0dot,robot,Ve,J,Jc,q_old,Tnew),options);

end
```

Figure 14 - Optimization code

In this case the variable of the problem is $\dot{q}_0$.

The objective function is the function to maximize. It computes the minimum distance manipulator-obstacles as described before.

The nonlinear constrained imposed are:

- limits of joints;
- limits of velocities of joints;
- position and orientation of end-effector is about equal (deviation less than a threshold) to desired one.
- The algorithm used is ***Active-Set*** because compared to the other possibility it can take large steps, which increase speed.

6.2 COMPARISON OF JOINTS SPACE TRAJECTORY GENERATION METHODS

In this part, the comparison of the methods of trajectory generation in the space of the joints is made:

- inverse kinematics;
- procedure not optimized to avoid obstacles (but optimized for joint speed and to avoid singularities)
- procedure optimized also to avoid obstacles;

For each of this, is show a plot of configurations of the joints along a trajectory taken as example. Furthermore, the final position of the robot is plotted at the end of the trajectory. This case shows how the optimized procedure is the only one able to avoid the obstacle of the bar table.

6.2.1 Inverse Kinematic

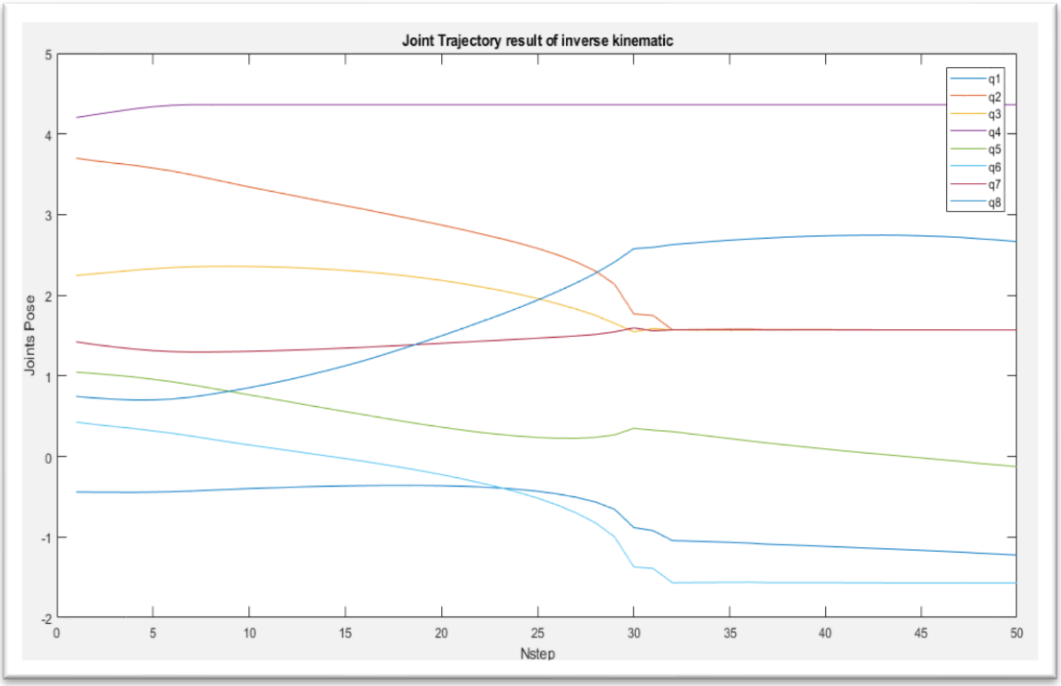

Figure 15 - Configurations of joints in an example trajectory computed by inverse kinematics

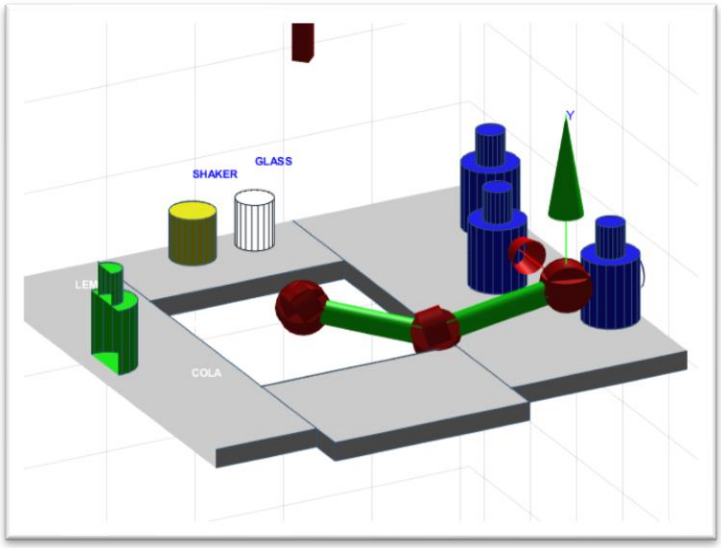

Figure 16 - Final pose of robot after an example trajectory computed by inverse kinematics

6.2.2 Not optimized procedure for collision avoidance

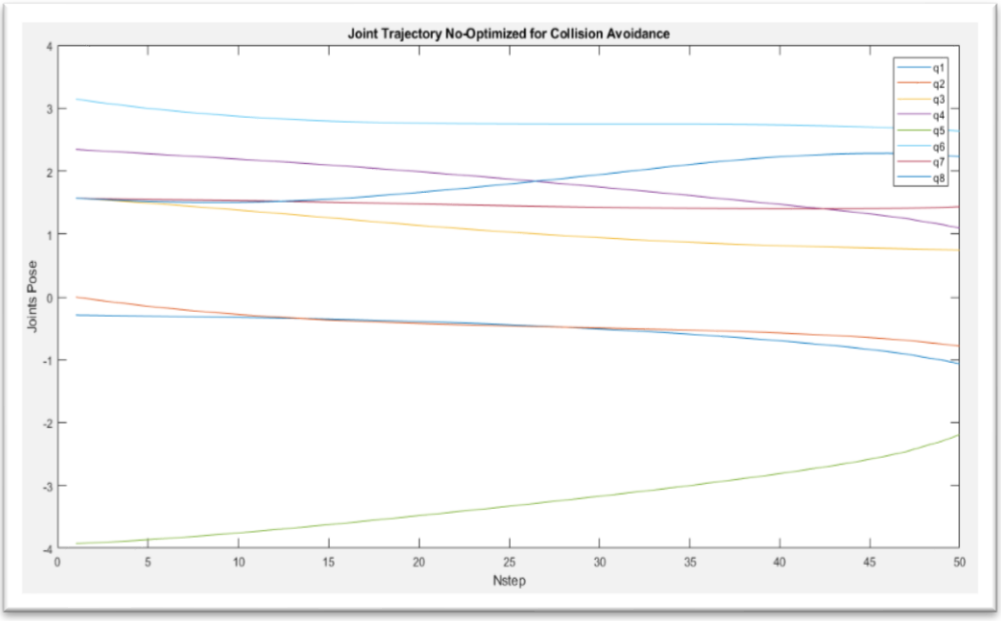

Figure 17 - Configurations of joints in an example trajectory computed by Non-optimized procedure for collision avoidance

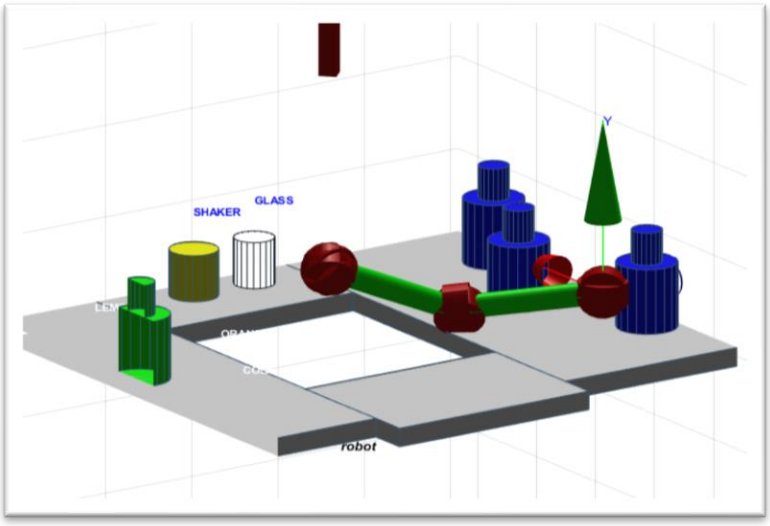

Figure 18 - Final pose of robot after an example trajectory computed by Non-optimized procedure for collision avoidance

6.2.3 Optimized procedure for collision avoidance

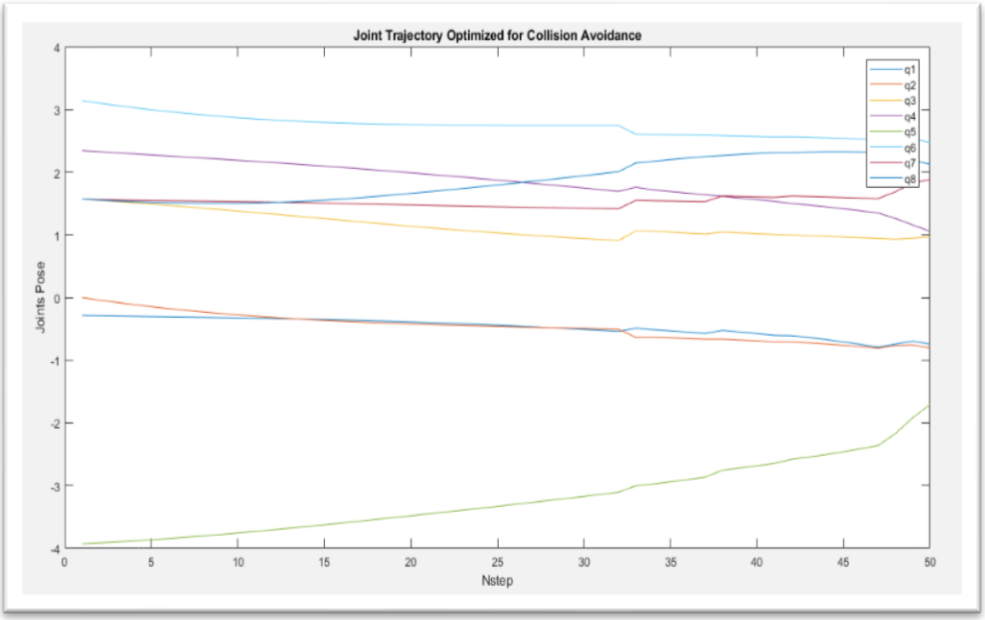

Figure 19 – Configurations of joints in an example trajectory computed by optimized procedure for collision avoidance

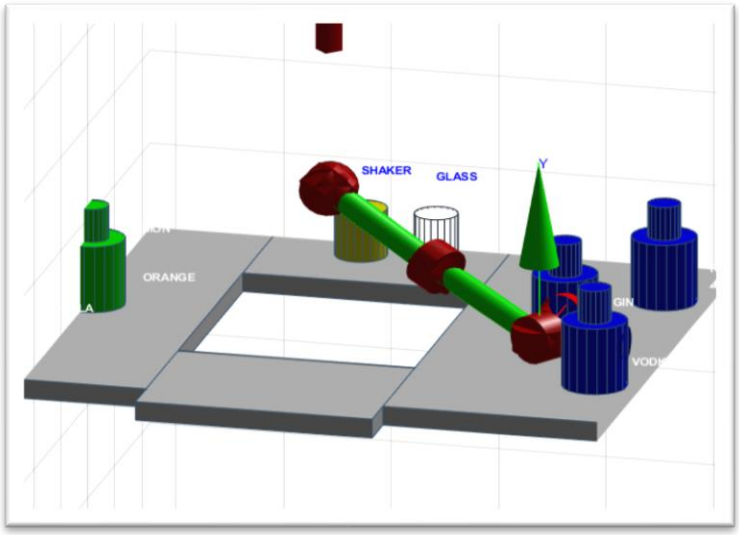

Figure 20 – Final pose of robot after an example trajectory computed by optimized procedure for collision avoidance

In the optimized procedure is exploited the non-empty null space of the Jacobian matrix that allows to use the redundant DOF to optimize the distances by the obstacles, through $\dot{q}_0$. In fact, $\dot{q}_0$ should generate internal motions of the manipulator without change the end effector pose.

Instead, the figure below, shows that the obtained pose of end effector is not always the same as that desired. This is caused by the time step which is not instantaneous unlike the applied theory which requires instantaneous conditions. Furthermore, the ***fmincon*** function uses iterative numerical optimization methods. These methods form successive approximations which leads to a generation and propagation of error.

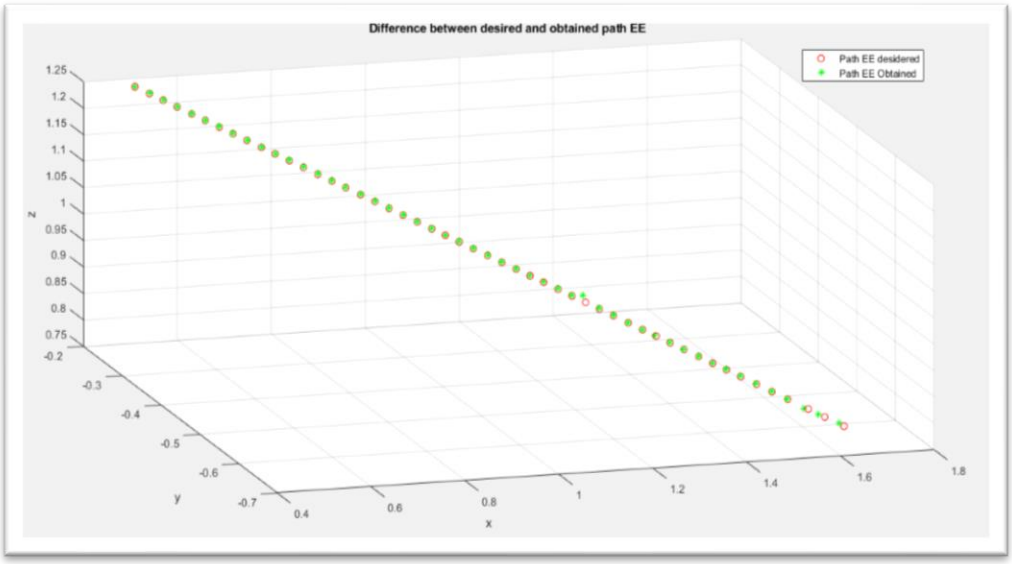

Figure 21 – End effector position desired and obtained

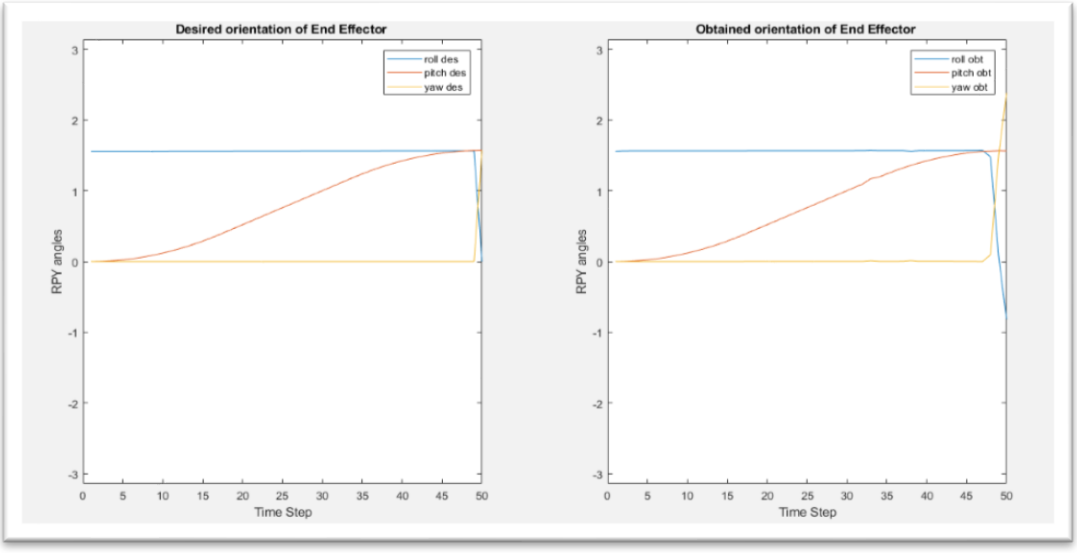

Figure 22 – Roll-pitch-yawn angle desired and obtained

The roll-pitch-yaw angles do not vary linearly with time. The discontinuity between the last two points is because the final orientation is a singularity for roll-pitch-yaw angles.

6.2.4 Manipulability

The plot of the manipulation is also shown only for the case of the method not optimized and optimized for collision avoidance. The computation of manipulation is done by the toolbox function ***maniplty*** which returns a manipulability index (scalar) for the robot at the joint configuration q. The measure is high when the manipulator is capable of equal motion in all directions and low when the manipulator is close to a singularity. As we can see in both cases, it is never too close to zero value, then singularities.

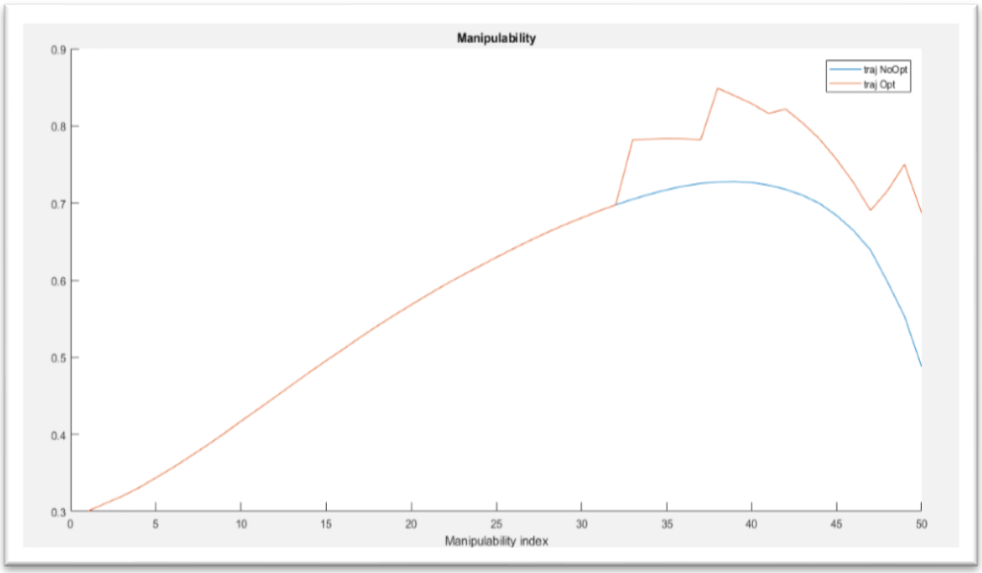

Figure 23 – Manipulability index

7 USE CASES

As an example, 3 cocktail recipes are created. The end effector homogeneous matrix T are computed manually with the help of the ***teach*** toolbox function, one for each object that shall be taken. The total trajectory is divided into 3 parts:

- pouring the ingredients into the shaker;
- mix of ingredients in the shaker;
- pouring the cocktail into the glass and moving it to its final position.

In the first part, starting from a resting pose, take the bottles indicated by the recipe to create the cocktail, pour the contents into the shaker and return the bottle to its initial position. This part ends with end effector in a suitable pose to mix the contents of the shaker.

In the second part, it moves up and down three times a precise joint to simulate the mixing. A problem encountered in this part is that the position and orientation of the end effector for mixing is correct, but the joints must also be in a predefined position. To bring the joints in this configuration, you could simply use the ***jtraj***

function of the toolbox, but it is not possible to prevent the content of the shaker is spilled. For this reason, the optimized procedure is used as explained before, in which the Cartesian trajectory to follow is simpler a trajectory of 20 identical steps for the end effector, since this must not be moved. During these steps, optimization takes advantage of redundancy to keep the end effector where it is located and change the configurations of the joints in the desired one. In fact, the new optimization problem has as an objective function to minimize the distance from the chosen configuration. The nonlinear constraints are the same as before.

In the last part, which is the same for all the cocktails, the contents of the shaker are poured into the glass and brought it to the final position to serve the customer.

At the end, the manipulator returns in the resting position, ready to prepare another cocktail.

For the computation of the parts described, a multi-trajectory function that takes as input the various steps to be reached, is created. Its purpose is to give at the trajectory function, for each step, the end effector final and initial pose. The latter is taken from the last pose of the previous step.

7.1 GRAPHIC INTERFACES

To show the calculated sample trajectories, a graphic interface created using the GUIDE of MATLAB is used, a drag-and-drop environment for laying out user interfaces(UIs).
Here are 4 buttons which, when clicked, show the trajectory of the manipulator for the preparation of 4 sample cocktails. After the click is shown in the lower right corner also the recipe, namely the bottles that the robot need to take to pour the contents into the shaker, mix all the ingredients and pour the result into the glass to serve the customer.

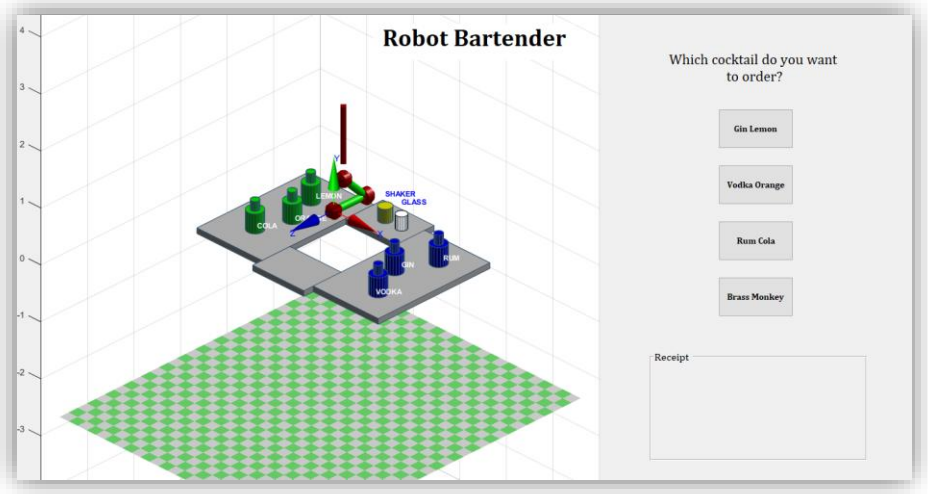

Figure 24 – Graphic interface

8 CONCLUSION

The adopted approach in the project leads to a good result compared to the task that the manipulator must perform. On the other hand, the joint space trajectory is not always smooth and has a certain error with respect to the calculated Cartesian trajectory.
These problems are probably caused by more than one reason. First, the non-optimized trajectory from the beginning but only when we get close to some obstacle. So, it's not entirely designed for that purpose.
This choice is made because the computation time ***fmincon*** function can be huge. In fact, the algorithm uses by ***fmincon*** are iterative methods. These are not expected to terminate in a finite number of steps. Starting from an initial guess, iterative methods form successive approximations that converge to the exact solution only in rare cases. A convergence test is specified to decide when a sufficiently accurate solution (hopefully) is found. Even using infinite precision arithmetic these methods would not reach the solution within a finite number of steps (in general), also for the strong dependence of the function from the starting point. This can change radically the computation time or even cause the non-convergence of algorithm to a feasible point.

Moreover, the problem to solve in this project have a high number of constraints which can make performance worse.

9 REFERENCES

[1] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, “Robotics: Modeling, Planning, and Control”
[2] Peter Corke “Robotics, Vision & Control”
[3] Oussama Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots”
[4] Y. Koren and J. Borenstein, ”Potential field methods and their inherent limitations for mobile robot navigation”, IEEE International Conference on Robotics and Automation,1991.
[5] Matoui, Boussaid, Abdelkrim, “Local minimum solution for the potential field method in multiple robot motion planning task”