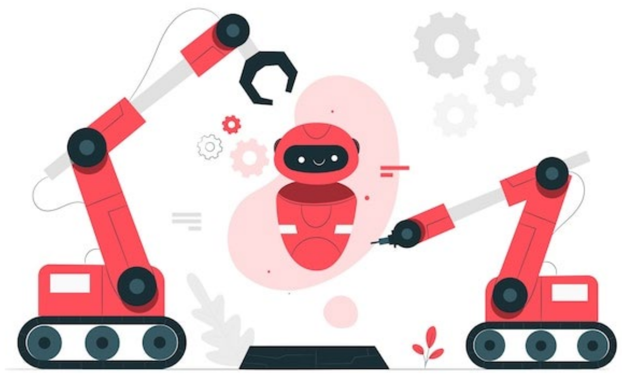
INDUSTRIAL ROBOTICS AND MOBILE ROBOTICS PROJECT



A.Y.: 2022/2023

Riccardo Florio

209502

Summary

Introduction

Robotics is a rapidly developing field that is playing an increasingly important role in automating processes and realizing intelligent solutions. Within this project, the work is divided into two parts: industrial robotics and mobile robotics.

In the first part, the focus is on industrial robotics, analyzing the structure of an anthropomorphic robot and studying its movement capabilities. It is widely used in various industries, such as production line automation, component assembly, and heavy object handling. During this phase of the project, the mechanical structure of an anthropomorphic robot is examined, including its articulations and control methods. Algorithms have also been developed to make it follow specific trajectories, in order to simulate activities such as picking and placing objects in an industrial environment.

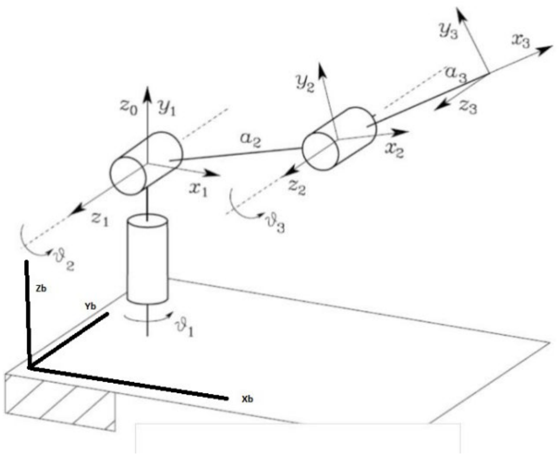
In the second part of the project, the focus is on mobile robotics and on the development of a robot capable of moving autonomously in a pre-established environment. It is becoming increasingly important in various contexts, such as logistics, exploration of hostile environments and care for the elderly. During this phase, a robot equipped with navigation algorithms is designed and built, which allow it to perceive its surroundings and plan its movements. A control system was also developed to ensure that the robot moves safely and efficiently in the pre-set path.

Through this robotics project, technical and conceptual challenges in the field of automation are addressed. The main objective is, therefore, to explore the potential of anthropomorphic and mobile robots, applying them to specific contexts such as industrial and mobile robotics. It is hoped that this report will provide a detailed overview of the activities carried out, the results obtained and the future prospects in the research and development of advanced robots.

INDUSTRIAL ROBOTICS

Request

Consider the anthropomorphic robot in the figure:



whose structure is modeled by the Denavit – Hartenberg table (Figure *1*) and in which the reference systems (0) and (b) are linked by the roto – translation matrix (*Figure 2*).

FIGURE 1

link	a (m)	α	d (m)	θ
1	0	$\frac{\pi}{2}$	0	θ_1
2	0.9	0	0	θ_2
3	0.9	0	0	θ_3

FIGURE 2

$$\hat{R}_0^b = \begin{pmatrix} 1 & 0 & 0 & 0.5 \\ 0 & 1 & 0 & 0.5 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Consider the following points:

$$P_1 = \begin{pmatrix} 0.8 \\ 0.8 \\ 0.5 \end{pmatrix}; \; P_2 = \begin{pmatrix} 1.2 \\ 0.8 \\ 0.5 \end{pmatrix}; \; P_3 = \begin{pmatrix} 1.0 \\ 1.2 \\ 0.5 \end{pmatrix};$$

Determine the time trends of the joint variables (position and speed) so that the origin of the CRS$_3$:

1. describes a triangle according to the sequence P $_1$ -> P2 -> P3 -> P1.
2. describes a circle passing through the points P1 -> P2 -> P3 -> P1.

in both cases the total travel time of the bend must be 40 [sec].

PS - distances are expressed in metres; angles in radians.

Path t triangle

First, note that the triangular path can be seen as the union of three straight paths, three segments to be precise. A segment can be parameterized as follows:

$$P(t) = P_1 + \lambda(t)[P_2 - P_1]$$

To get the lambda we use the *poly3.m function,* to which we pass the *sigma parameter*; σ is a dimensionless independent variable that varies between 0 and 1. This is done in the initial part of the *Industrial.mlx script, present in the Industrial Robotics* folder, written using the 'Matlab R2021a' software (it is recommended to use an equal or later version). The same part of the code also defines some variables that will be useful later on, such as the points P1, P2 and P3, i.e. the vertices of the triangle, the vector L, containing the lengths of the arms (extrapolated from the Denavit – Hartenberg table) and the execution times of the movement. The execution times are obtained by dividing the total time that the robot must take to travel the trajectory into three parts, approximately *13.33 sec*.

```
% Robot Arms
global L
L = [0; 0.9; 0.9];

% Vertices of the triangle
P1 = [0.8; 0.8; 0.5];
P2 = [1.2; 0.8; 0.5];
P3 = [1.0; 1.2; 0.5];

% Execution times
T1 = 0; T2 = 13.3333;
T3 = 28.6667; T4 = 40;

% sigma
DSIGMA = 0.03;
sigma = 0:DSIGMA :1;
N = length(sigma );

% lambdas
lambda = Functions.poly3(sigma);
lambda_d = Functions.poly3d(sigma);
```

In the previous piece of code, *lambda_d* variable is also defined by using the appropriate *poly3d* function, which is nothing but the derivative of *lambda*, which will be useful later on for calculating the joint speeds.

We then proceed by traversing the section from P1 to P2: at each iteration the position of the end - effector is calculated using the function $P(t)$seen before, the value of the joint variables using inverse kinematics (Anthropomorphic_*Cin_Inv.m function)* and the value of the joint speed. For the calculation of the speeds, the inverse differential kinematics is applied. In particular, the following relationship is exploited:

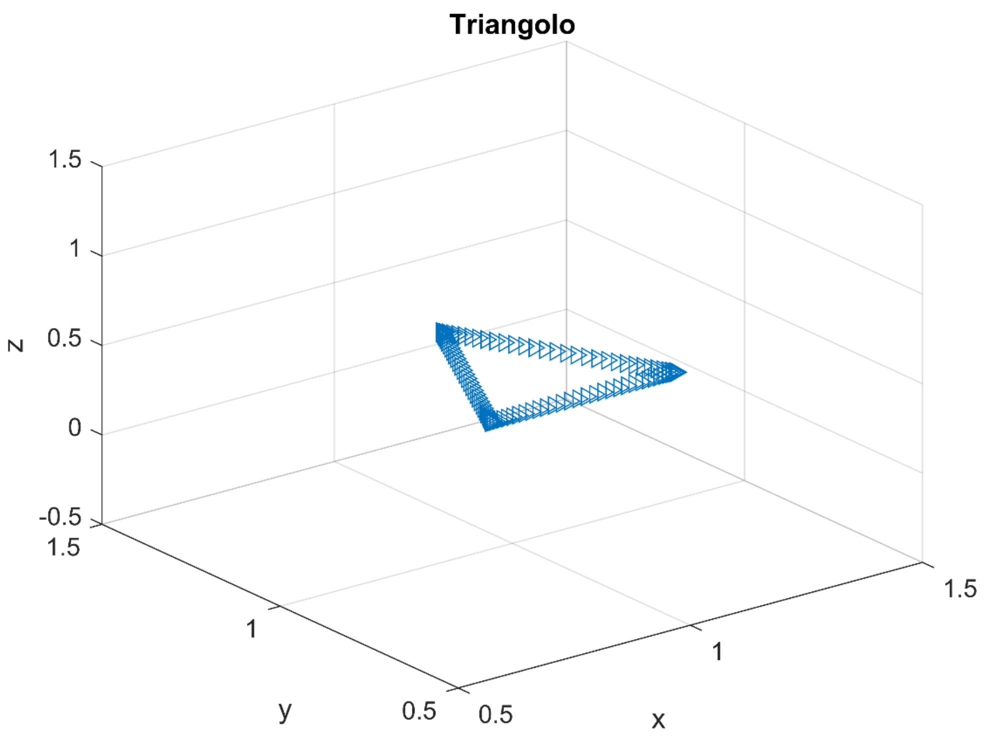
$$\dot{P} = J(Q) \cdot \dot{Q} \rightarrow \dot{Q} = J(Q)^{-1} \cdot \dot{P}$$

```
% Path P1->P2
for i = 1:N
    % Parameterisation of the route
    P = P1+lambda(i)*(P2-P1);
    % Calculation of inverse kinematics
Q = Functions.Anthropomorphic_Cin_ Inv(L,P);
    % Calculation of J(Q)
J = Functions.Jacobian_Anthropomorphic(L,Q);
    % Resolution of the system of eqs. linear J(Q)* Qd =Pd
Pd = (P2-P 1)* lambda_d (i)/(T2-T1);
    Qd = inv (J)*Pd;

    QQ( i,:)=Q; % Joint variables at time i
    PP( i,:)= P ; % Position of the end effector at time i
    QQd ( i,:)= Qd ; % Speed of joints at instant i
end
```

The paths P2 -> P3 and P3 -> P1 are calculated in the same way.

At this point the path is plotted:

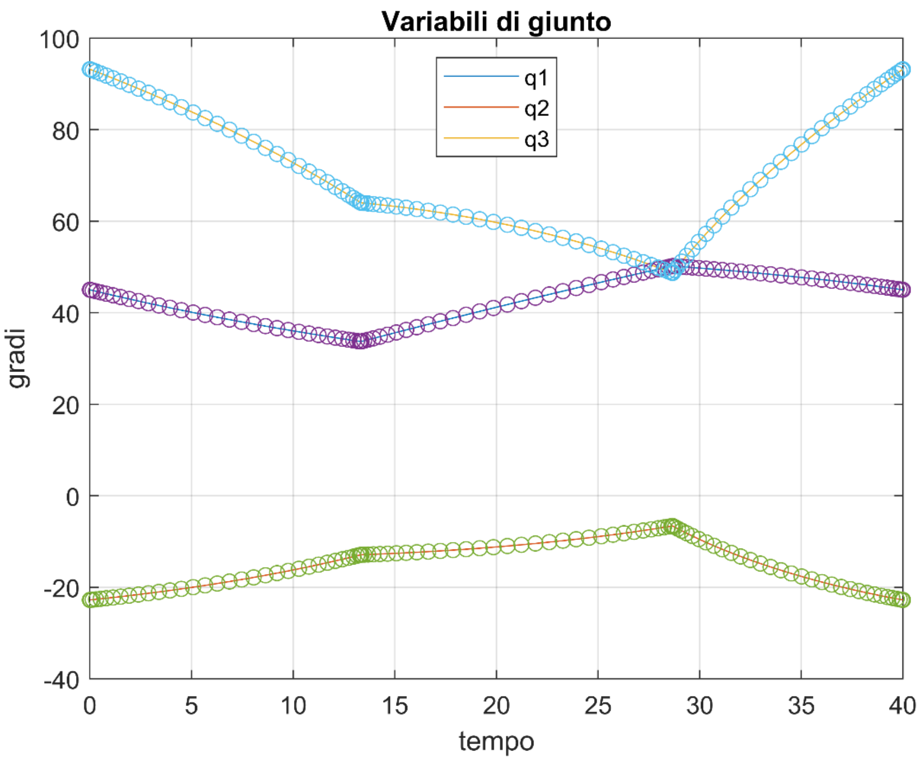


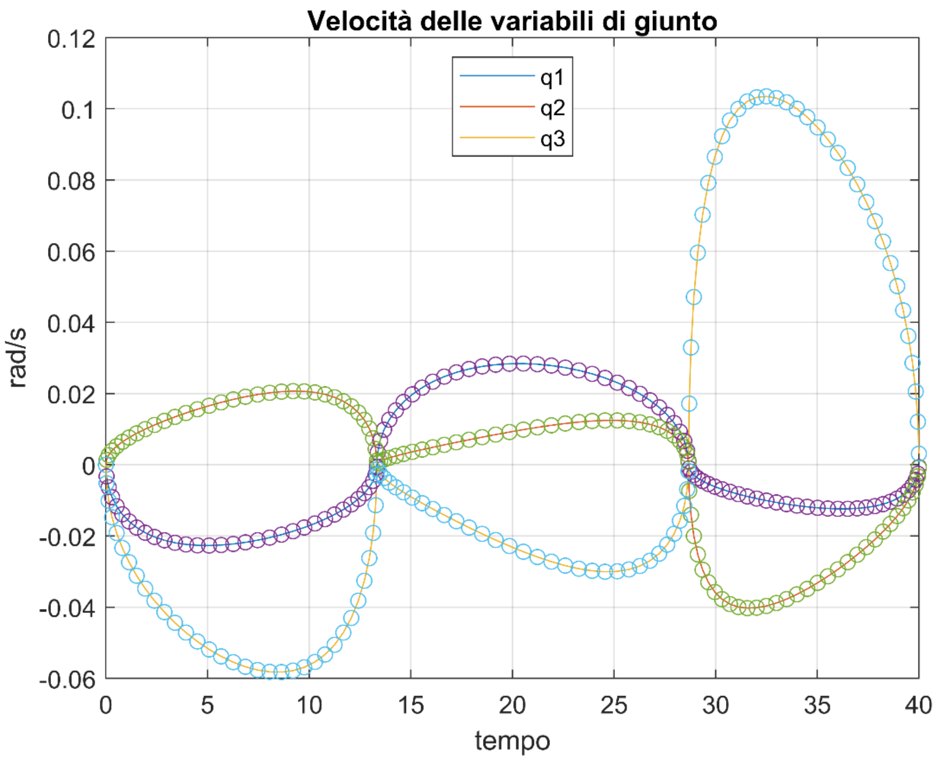
Finally, the temporal trends of the joint variables can be visualized after having parameterized the time and converted the angles expressed in radians into angles expressed in degrees:

```
% Plot joint variables
t1 = T1+lambda*(T2-T1);
t2 = T2+lambda*(T3-T2);
t3 = T3+lambda*(T4-T3);
t = [t1 t2(2:N) t3(2:N)];

plot(t,QQ*180/pi);

....
```




Velocità delle variabili di giunto
q1
q2
q3
rad/s
tempo
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
-0.04
-0.06
0
5
10
15
20
25
30
35
40

Path circumference

Also in this case it is necessary to parameterize the path to be followed. The generic equation of the circle in the plane is given by:

$$P = P_C + R \cdot \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \end{bmatrix}, con\ 0 \leq \theta \leq 2\pi$$

In the case under examination, extending in $\mathbb{R}^3$:

$$\begin{cases} P_x = P_{Cx} + R \cdot \cos(\theta) \\ P_y = P_{Cy} + R \cdot \sin(\theta) \\ P_z = P_{Cz} \end{cases}$$

that is to say:

$$P = P_C + R \cdot \begin{bmatrix} \cos(\theta) \\ \sin(\theta) \\ 0 \end{bmatrix}, with\ 0 \leq \theta \leq 2\pi$$

from the equation of the circumference that passes through P1, P2 and P3, we obtain the center and the radius:

$$P_C = (1.0, 0.95, 0.5)$$

$$r = 0.25$$

Before proceeding with the implementation, it is important to specify that the path to be followed must start from point P1; therefore, the range in which theta varies must be: $\theta_1 \leq \theta \leq \theta_1 + 2\pi$.

Using the principles of trigonometry, we get that $\theta_1 \cong 3.8\, rad$.

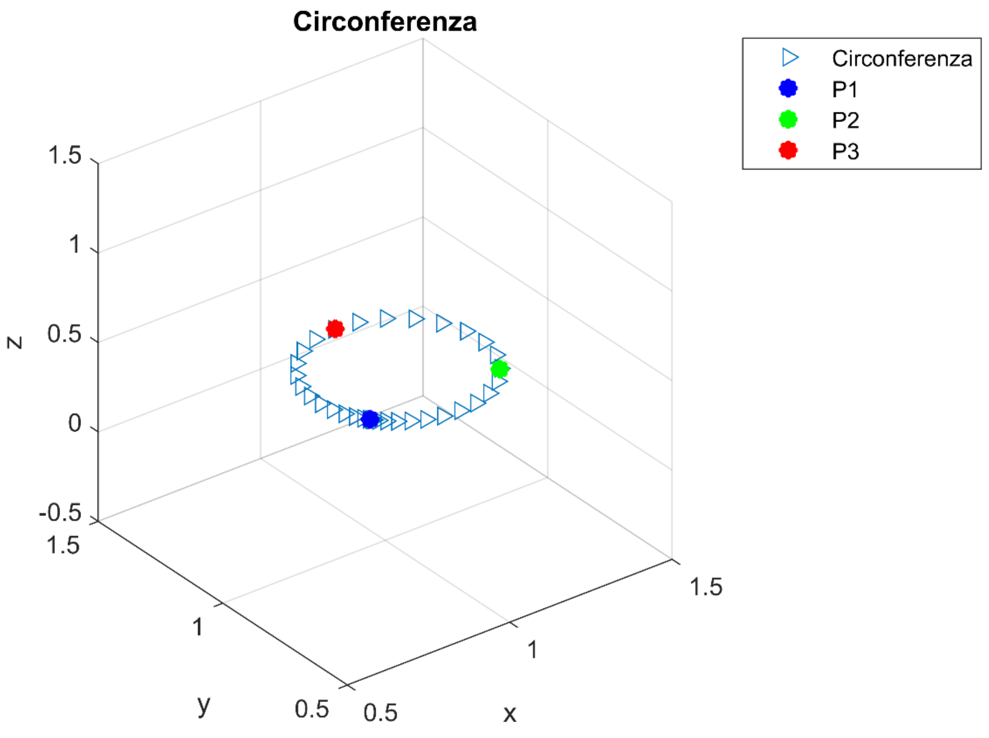
Furthermore, it is necessary to express theta as a function of λ:

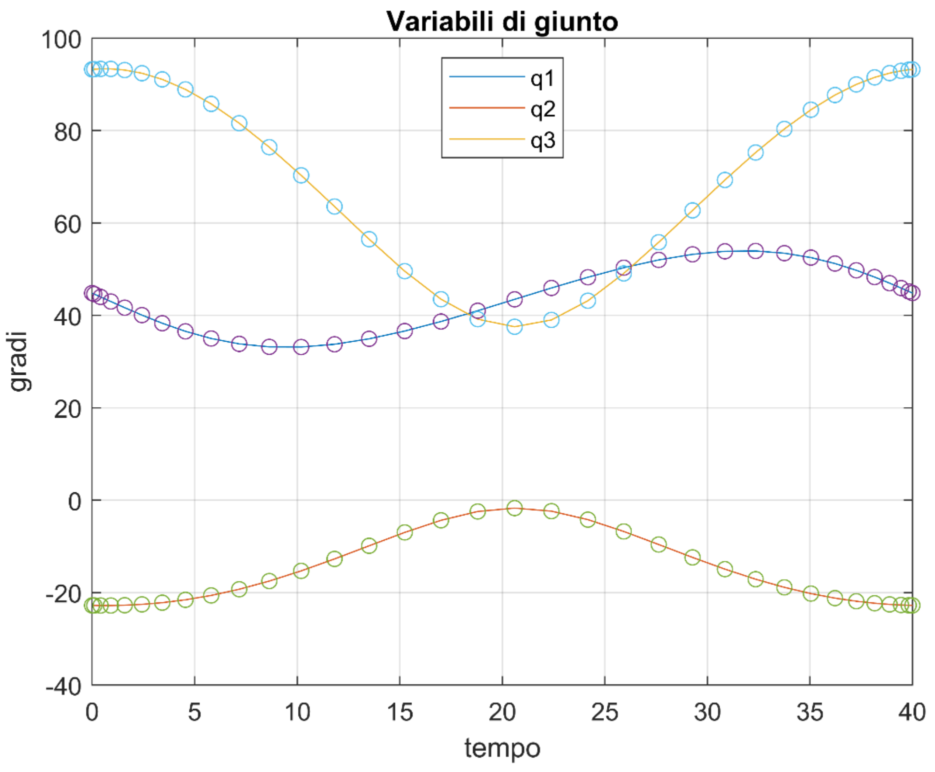
$$\theta = \theta_1 + \lambda[\theta_2 - \theta_1]$$

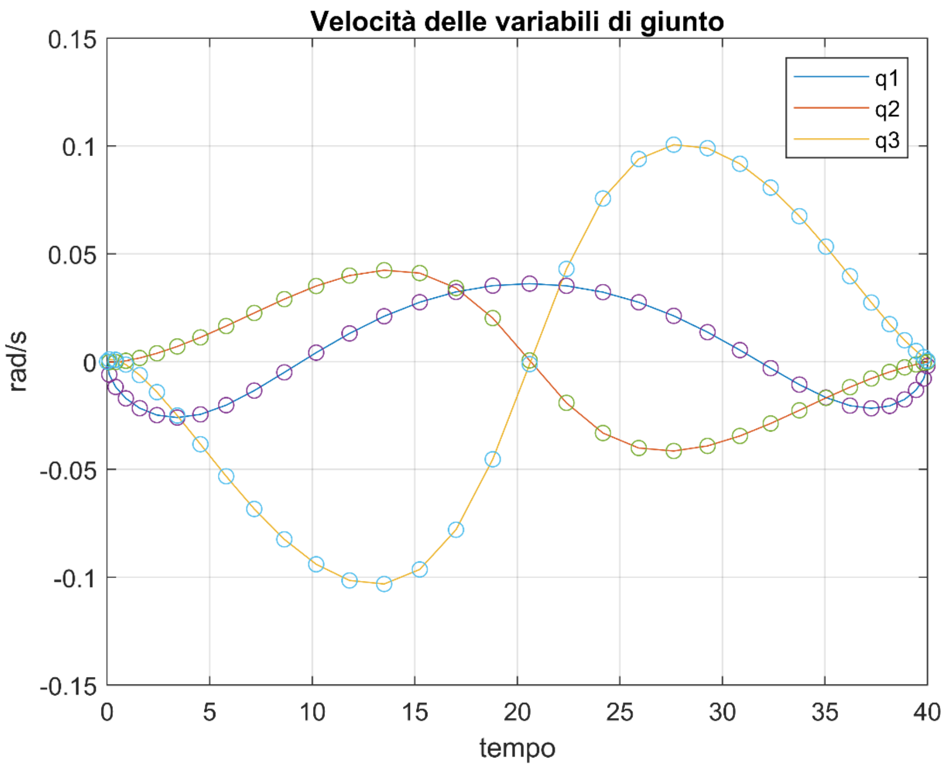
In the present case: $\theta = \theta_1 + \lambda[\theta_1 + 2\pi - \theta_1] = 3.8 + \lambda \cdot 2\pi$.

Analogously to the previous case, the section P1 -> P2 -> P3 -> P1 is covered, and the joint variables are obtained by applying the inverse kinematics and the inverse differential kinematics:

```
% Path Circumference
for i= 1:N
    P = PC+r *[cos(theta(i)); sin(theta(i)); 0];
    Q = Functions.Anthropomorphic_Cin_Inv (L,P);
    J = Functions.Jacobian_Anthropomorphic (L,Q');
    Pd=(r*[-sin(theta(i)), cos(theta(i)),
        0]* lambda_d(i)*2*pi)/(T4-T1);
    Qd = inv (J)*Pd';
    QQc (i,:)=Q; PPc (i,:)=P; QQdc(i,:)=Qd;
End
```


Circonferenza
Circonferenza
P1
P2
P3
z
x
y
1.5
1
0.5
0
-0.5
1.5
1
0.5
0.5
1
1.5


Variabili di giunto
q1
q2
q3
gradi
tempo
100
80
60
40
20
0
-20
-40
0
5
10
15
20
25
30
35
40


Velocità delle variabili di giunto
q1
q2
q3
rad/s
tempo
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
0
5
10
15
20
25
30
35
40

MOBILE ROBOTICS

Request

After defining an environment with obstacles, design a code that is able to move a robot from a start point to a goal point with a certain final orientation. To do this, the following control scheme must be implemented:

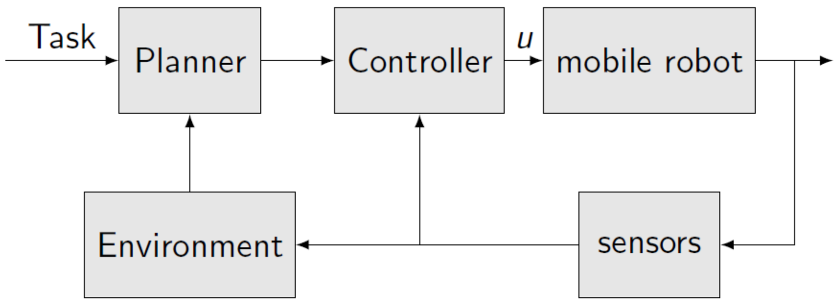


Figure: Mobile robot control scheme

In particular,

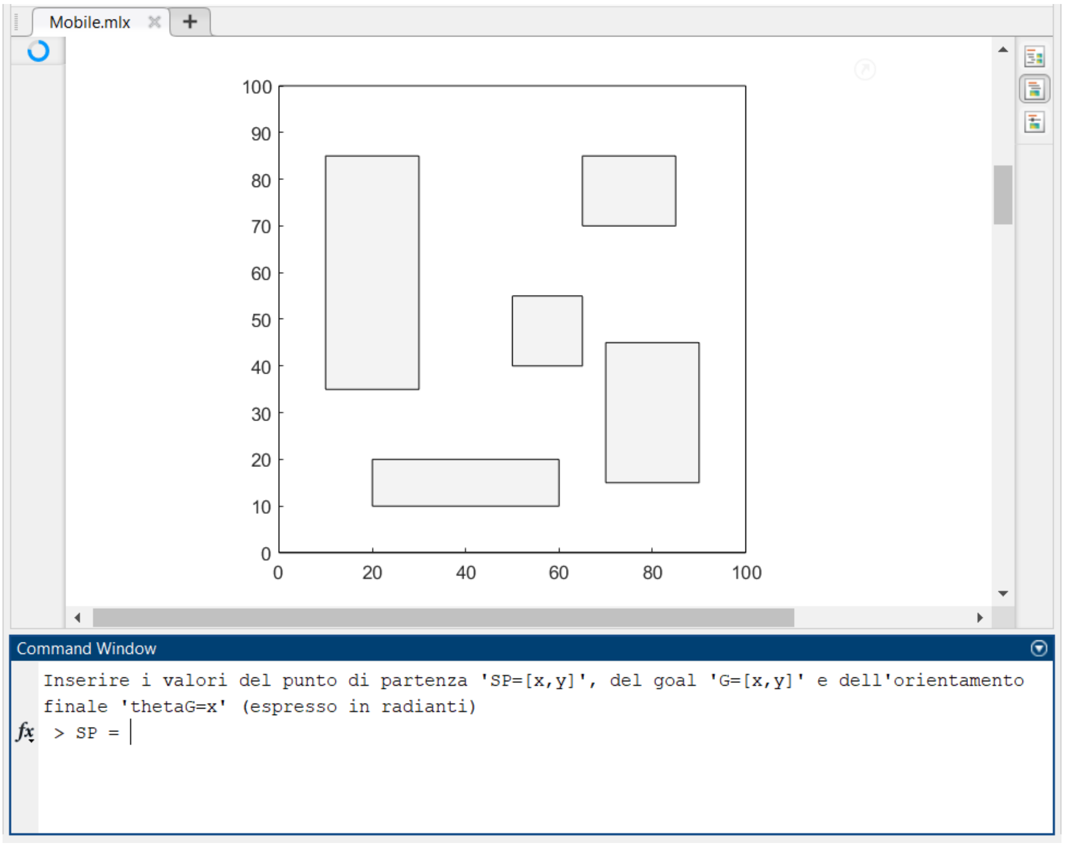
- the user must be able to choose the Task, i.e. initial position, final position and orientation of the robot.
- the user must be able to choose which of the following Planners to use: Artificial Potential Fields, Discrete Potential Fields, Voronoi Diagrams, Visibility Graphs (the full version of the latter is preferable).
- the user must also be able to choose the type of Controller that guides the robot in order to make it follow the trajectory defined in the previous point: Linear, Non-Linear, Input/Output Linearization.
- the Mobile Robot must be of the unicycle or differential drive type.
- the Environment is predetermined.
- no need to implement the Sensors block.

Guide

Before going into the analysis of the implementation of the request, it is useful to view a brief guide to using the program.

Once the *Mobile.mlx script has been launched,* the environment within which the robot will move will be plotted in the *Mobile Robotics* folder. Scroll down and it will appear.

At this point the program will wait for the required data to be entered in the command window. A situation like this should occur:



```
Inserire i valori del punto di partenza 'SP=[x,y]', del goal 'G=[x,y]' e dell'orientamento
finale 'thetaG=x' (espresso in radianti)
fx  > SP = 
```

After which the program will continue its execution, and by scrolling down it will be possible to graphically display the results obtained based on the entered parameters. Continuing, the program will ask you to enter values for choosing the Planner and Controller to use, applying the same procedure used previously.

The square brackets must not be omitted when entering the starting point and goal.

If you want to interrupt the program, just type the key combination *CTRL+C*.

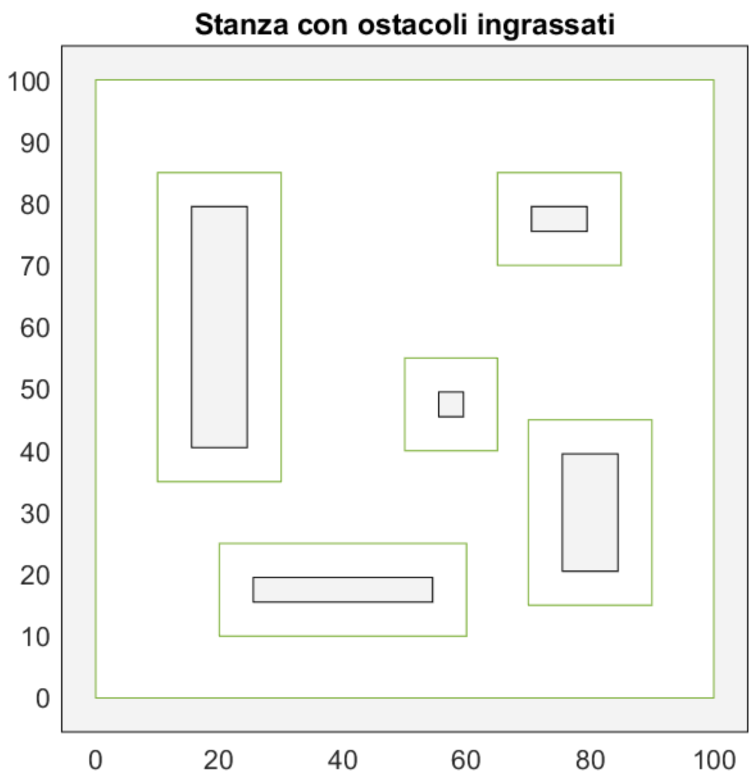
Task

The first request to satisfy concerns the choice of parameters to be fed to the Planner block. It will be the user himself who enters these parameters, which will be memorized in the variables *"SP"*, *"G"* and *"thetaG"* , which stand respectively for starting point, goal and final angle of the robot.

Environment block

In this phase we deal with defining an environment within which the robot can move. Specifically, a square room was designed with five rectangular obstacles, each of different sizes, and the obstacle procedure was performed management called "*fattening*". This procedure consists in circumscribing the obstacle with a known shape and, subsequently, enlarging it in such a way as to include the real dimensions of the robot. By doing so, it will be possible to model the robot as a point body.

The environment looks like this:



The green edges are the outlines of the obstacles enlarged by an amount equal to the sum of the radius plus 10% of the radius itself. Only such contours will be taken into consideration during implementation.

Finally, pay attention to the following line of code:

```
[room]= Utilities.Build_Room(obstacles);
```

which assigns to the variable room the vector that constitutes the set of points that form the outlines of the room and of the obstacles. This vector was obtained by discretizing the edges of the obstacles with a unit step and is important as it will be used often later.

Planner block

Once the Task and Environment have been established, we can proceed with the design of the Planner block. This block deals with solving the *Path Planning problem:* given a start point, we want to design a trajectory to reach a fixed goal point avoiding the static obstacles present in the environment.

The user can choose which of the three Path Planning techniques to have the Planner use among those that we will analyze below.

Artificial potential fields

This technique combines knowledge from two specific fields: artificial potentials, derived from physics, and minimization problems, from the field of optimization.

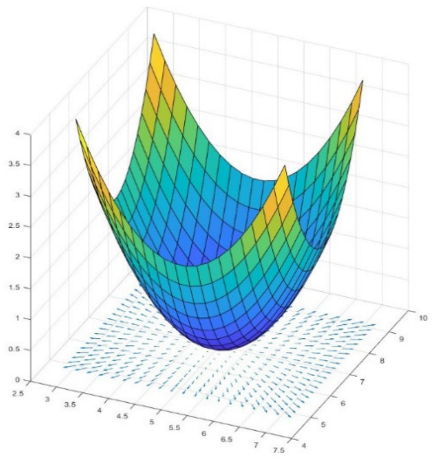
The idea is to place a negative charge in correspondence with the goal and a positive charge in correspondence with the robot which is attracted by the negative one. Thus, one can think of placing a potential field that is attractive to the goal and repulsive from obstacles:

$$J(X) = w_a J_a(X(t), G) + w_o \sum_{i=1}^{N_o} J_r(X(t), O_i)$$

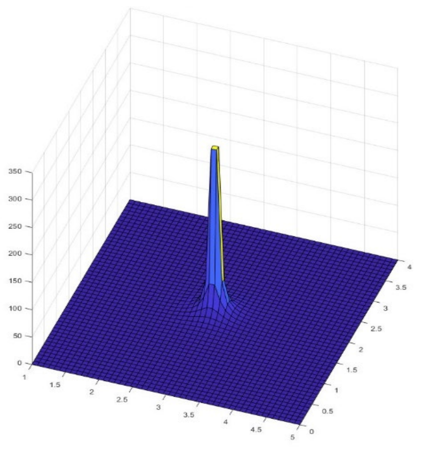
Where: $J_a$and $J_r$are respectively the attractive and repulsive potential, $w_a$and $w_o$are *weights* to be calibrated, N_o, G and X(t) are respectively number of obstacles, goal position and robot position.

In this discussion, we consider $J_a = \frac{1}{2}\|X(t) - G\|^2$ and $J_r = \frac{1}{\|X(t)-O\|^2}$:

ATTRACTIVE POTENTIAL

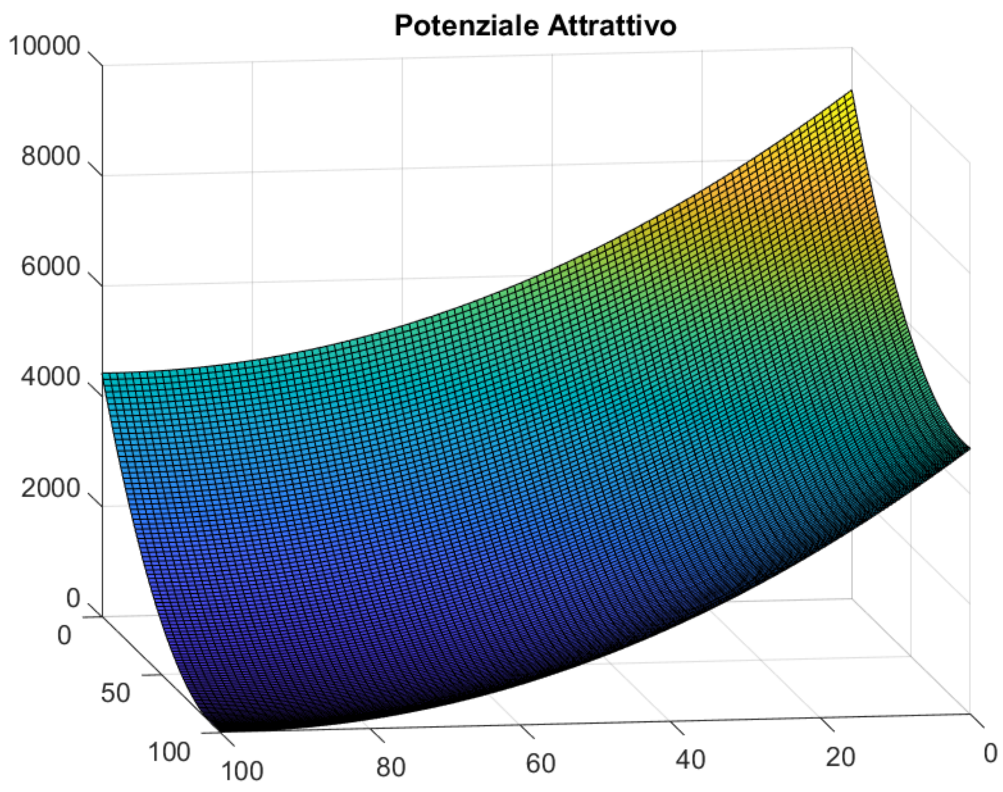


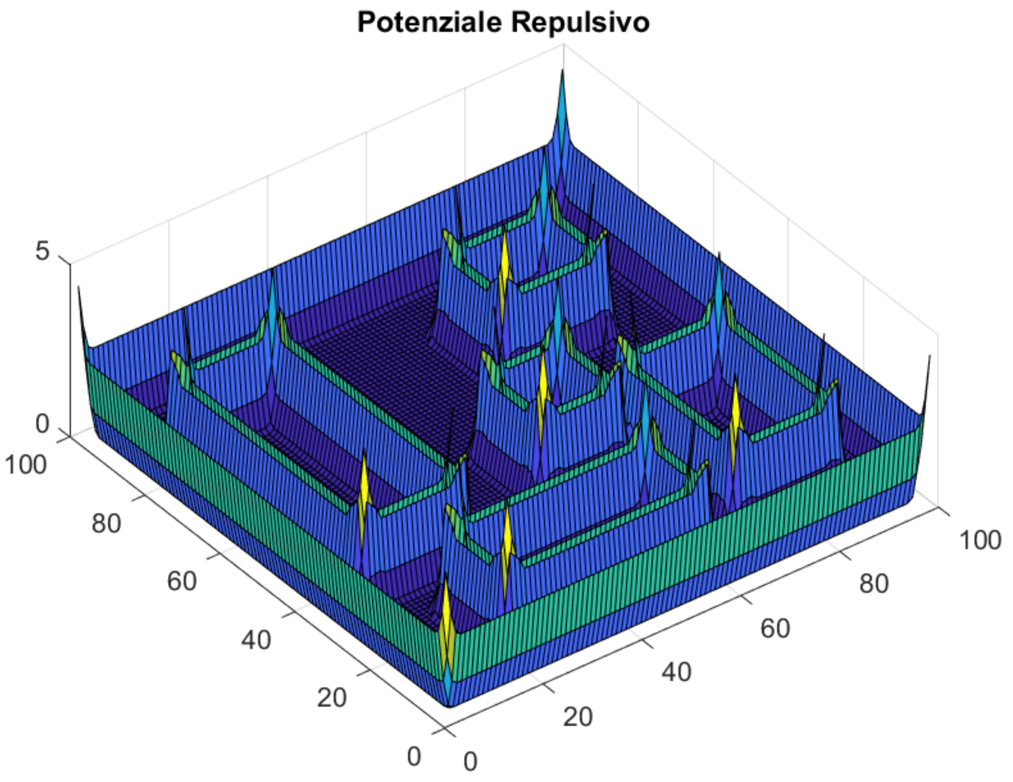
REPULSIVE POTENTIAL

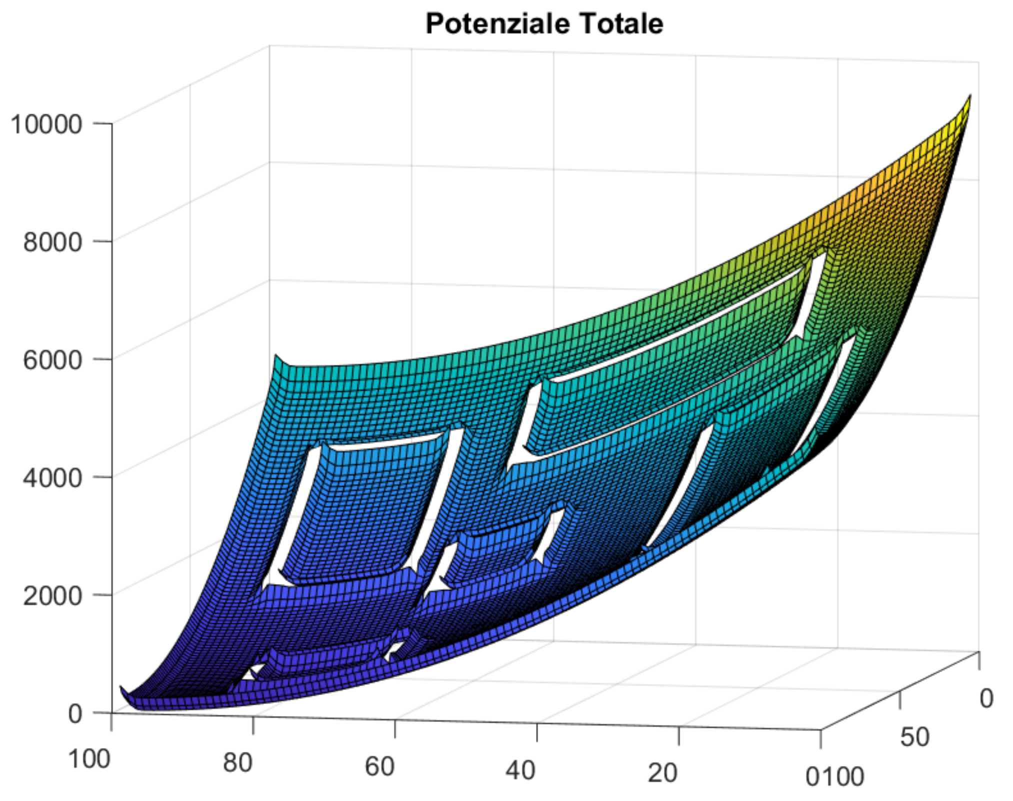


From now on the following parameters will be considered for the simulations: SP = [5, 7], G = [98, 94] and thetaG = pi/2.

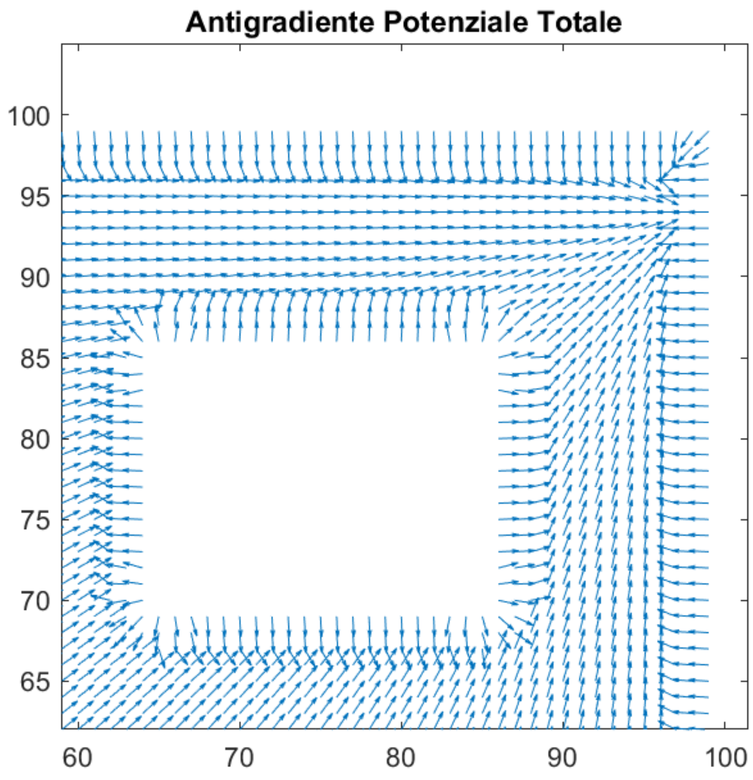
Having these data and knowing the environment, the potentials can be calculated in the specific case:







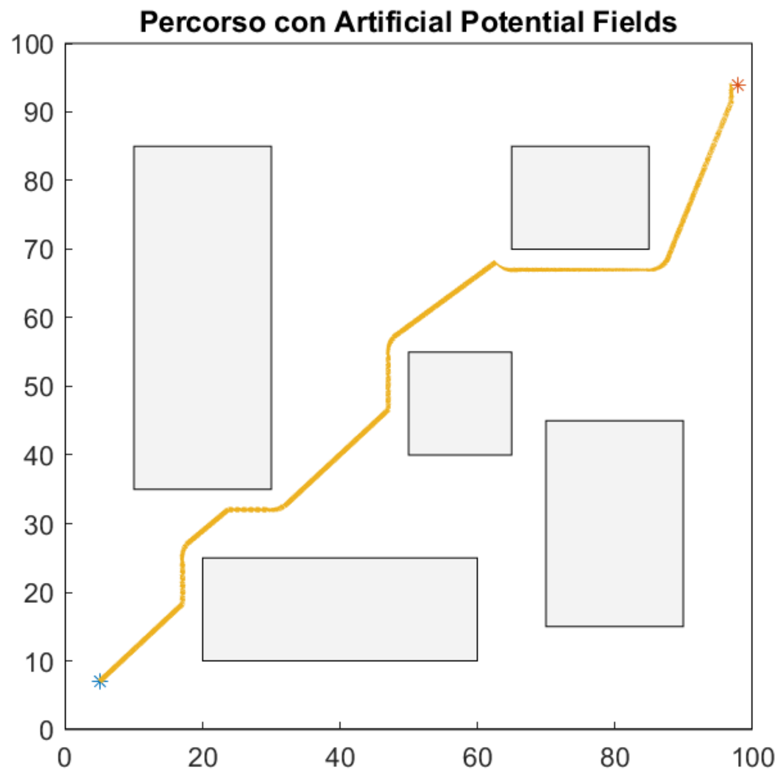
For graphical purposes, the anti-gradient of the total potential is normalized and a portion of it is displayed below:



The calculation of the potentials is done inside the *Artificial_Potential_Fields.m* file present in the +*Planners* folder. The trajectory calculation is also performed in the same file, treating the path planning problem as a minimization problem. The idea is to follow the anti-gradient to guide the robot towards the goal, avoiding obstacles along the way. Therefore, the problem becomes the following minimization problem:

$$\begin{cases} \min\limits_{X \in \mathbb{R}^2} J(X) \\ X(0) = SP \end{cases}$$

where SP is the initial position of the robot. By means of the minimization algorithm used, the following trajectory is obtained:

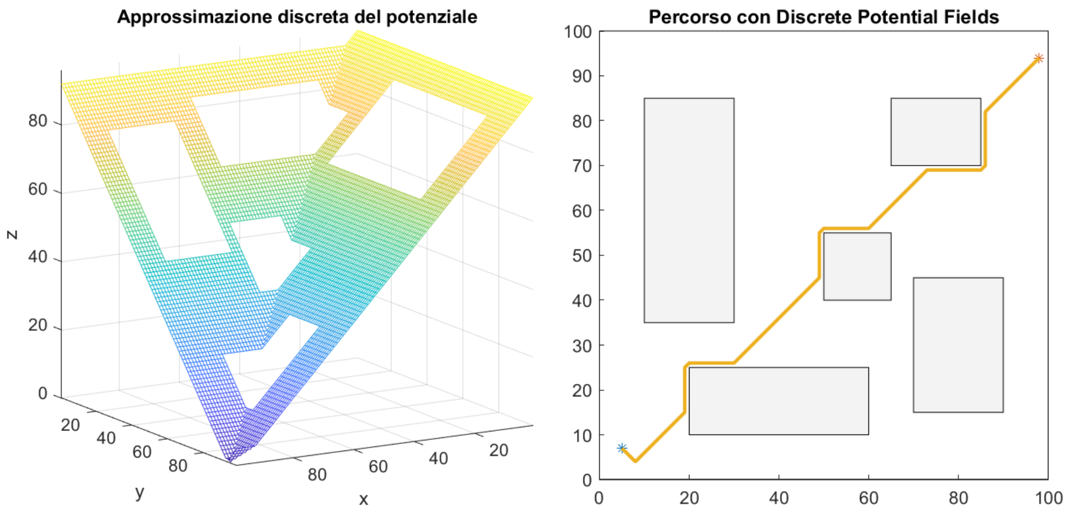


While this method works, it is important to note that an unhandled exception may occur in the presence of local minima. In this case, if the robot were on the axis that connects the goal to the obstacle, it could get stuck in the minimum point, without fully reaching the goal.

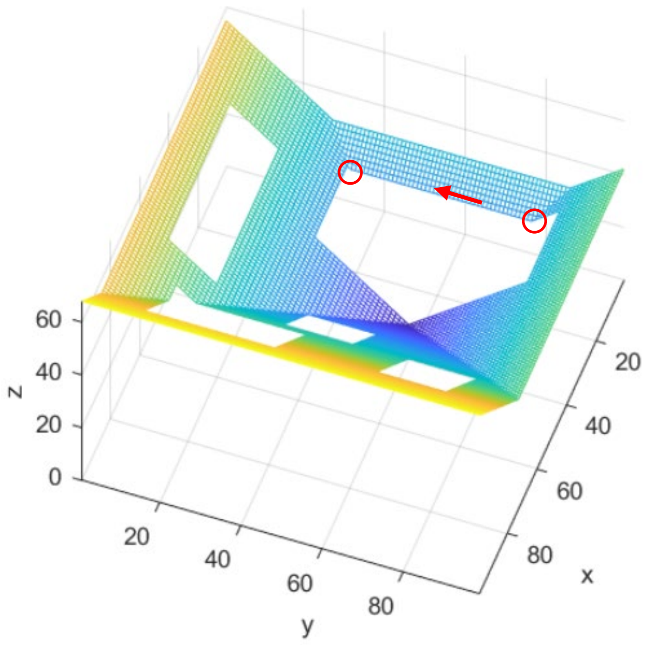
Discrete Potential Fields

From what has been seen in the previous technique, it can be assumed that the potential has higher values near the obstacles and lower values near the goal. Therefore, by dividing the environment into cells of unit size, it is possible to obtain a discrete approximation of the potential. We can therefore think of an algorithm which, starting from the starting point and moving one cell at a time within this environment, creates a trajectory moving towards the cells with gradually decreasing values until reaching the goal.

The above is performed in the *Discrete_Potential_Fields.m* file present in the *+Planners* folder and the result is as follows:



Even using this technique, you could run into an "anomalous" case. Let us consider, for example, the following case:

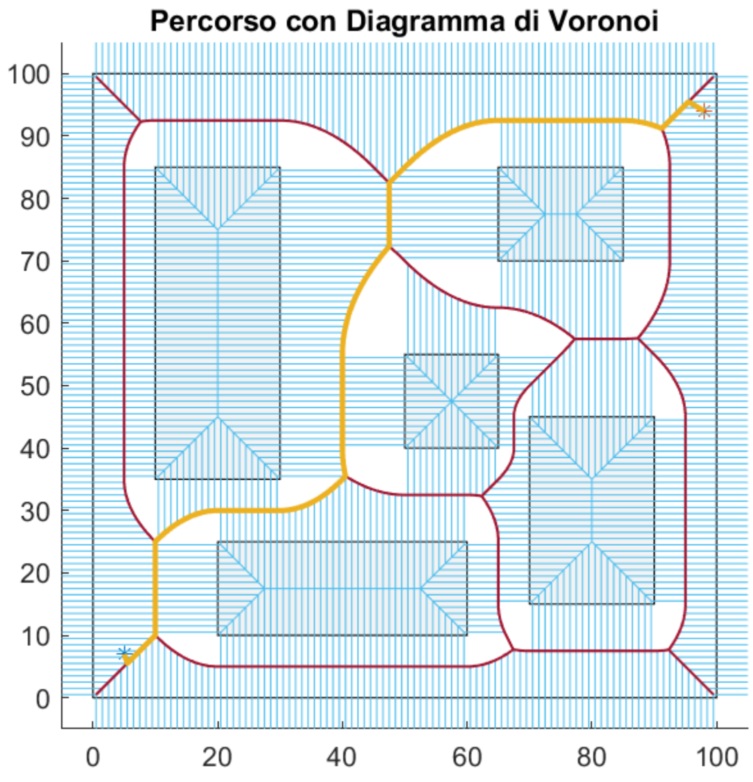


From the figure there are two points where the robot can stop without generating a trajectory to reach the goal. This is due to the shape of the obstacles and the very nature of the discrete artificial potentials method.

Voronoi Diagrams

The Voronoi diagram represents the locus of points equidistant from the closest two or more obstacle boundaries, including the workspace boundary. In the script file *Voronoi_Diagrams.m*, located in the *+Planners* folder, this diagram is calculated using the Matlab *voronoi()* function. Next, the diagram points that are inside the obstacles are removed and the start and destination points are added inside the diagram. At this point, the goal is to find the shortest path that traverses the diagram from the starting point to the destination point. To do this, a graph is created in which nodes correspond to points on the diagram and arcs represent connections between two points. The *shortestpath()* routine is then applied to find the shortest path in the graph. Once the desired trajectory has been obtained, further points are added along the sections in which the distance between two points exceeds a pre – established threshold (in this specific case, set to 1.5).

Below the plot of the voronoi diagram (in red) and of the final trajectory (in yellow):



Visibility graphs

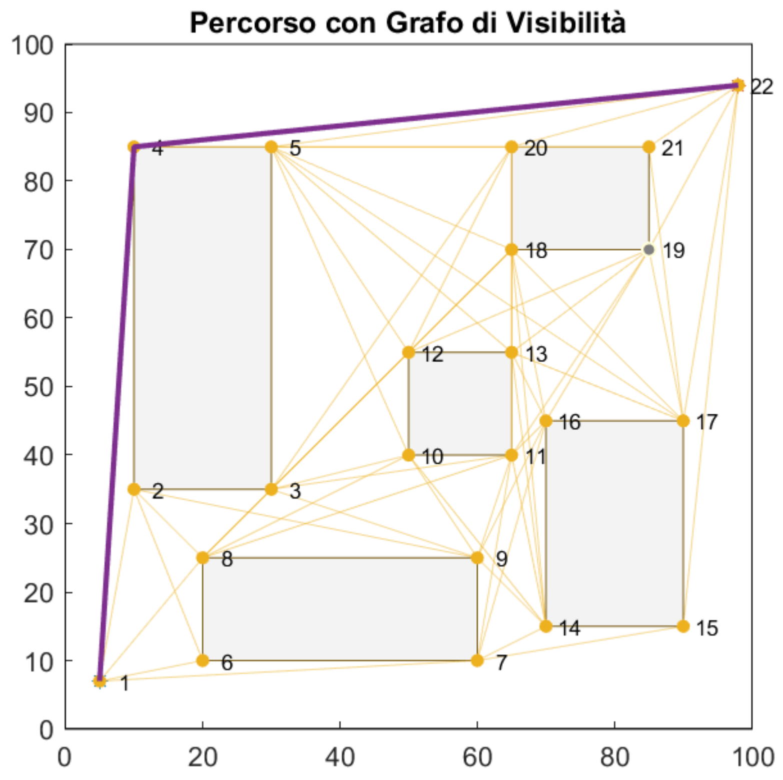
The idea behind this technique is: “if I can see a point, then I can reach it”.

Again, the implementation is located in the *Visibility_Graphs.m file,* in the *+Planners folder*. In this case, the graph is constructed starting from the adjacency matrix, obtained with the help of the *isVisible()* utility routine , and exploiting the *convertToGraph()* utility routine . The graph will therefore be made up of nodes corresponding to the ends of the obstacles and of arcs which are the connection between two nodes that "see each other": two nodes "see each other" if the segment that connects them does not intersect any obstacle.

The utility functions are in the *+Utilities* folder.

Once the graph is obtained, the shortest path is extracted using the *shortestpath() function* and the path is discretized. To discretize the path, it was decided to insert fifty intermediate points between two nodes of the shortest path.

The trajectory obtained is the following:



Controller block

In this section we are concerned with designing appropriate control inputs, v and ω, to the robot to guide its pose towards a given target trajectory or state.

"Robot pose" refers to the spatial configuration and orientation of a robot at a given instant.

We will consider two types of control activities: *trajectory tracking* and *posture regulation*.

It is important to emphasize that the control laws that we will discuss later are specifically designed for the *Unicycle robot model,* which will be examined later in the text.

Trajectory tracking

The task of trajectory tracking is to follow a specific trajectory over time, ensuring a desired evolution of the robot's pose.

In the first instance, we are concerned with reporting the reference trajectory in the form of an objective function. In the *Trajectory Tracking* section of the *Mobile.mlx* file, the Matlab routine *polyfit() is used* to obtain the objective function *y_star* :

```
P= polyfit (path(:,1),path(:,2),5);
x_star =@(t)(t);
y_star = @(t)(P(1)*t.^5 + P(2)*t.^4 + P(3)*t.^3 + P(4)*t.^2 ...
            + P( 5)*t + P(6));
```

x_star, *y_star* and *theta_star* are defined, which are essential for subsequent calculations. Soon after, the new parameters of the starting point and the starting angle of the robot are presented. These parameters are set because the robot may initially be positioned at a slightly different point than the default starting point.

Before proceeding, it is important to point out that in this phase it is not necessary to reach the goal point exactly, but it is sufficient to arrive around it.

Also in this case the user will be able to choose one of the three control laws implemented in order to pursue the objective function.

Control based on approximate linearization

This type of control law, also known as linear control, is based on the concept of approximating the trajectory following error, $\dot{e}$, around $e = 0$. Specifically, the law can be expressed as follows:

$$\begin{cases} v = v^* \cos(\theta^* - \theta) + K_1(x^* - x) \\ \omega = \omega^* + K_2(y^* - y) + K_3(\theta^* - \theta) \end{cases}$$

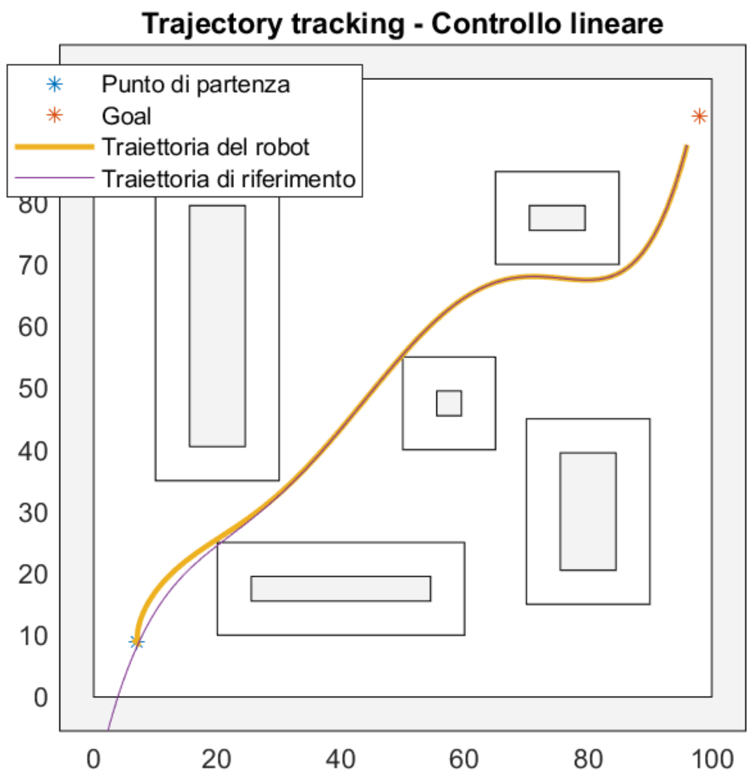
With:

$$\begin{cases} K_1 = 2\delta a \\ k_2 = \dfrac{a^2 - (\omega^*)^2}{v^*} \\ K_3 = 2\delta a \end{cases}$$

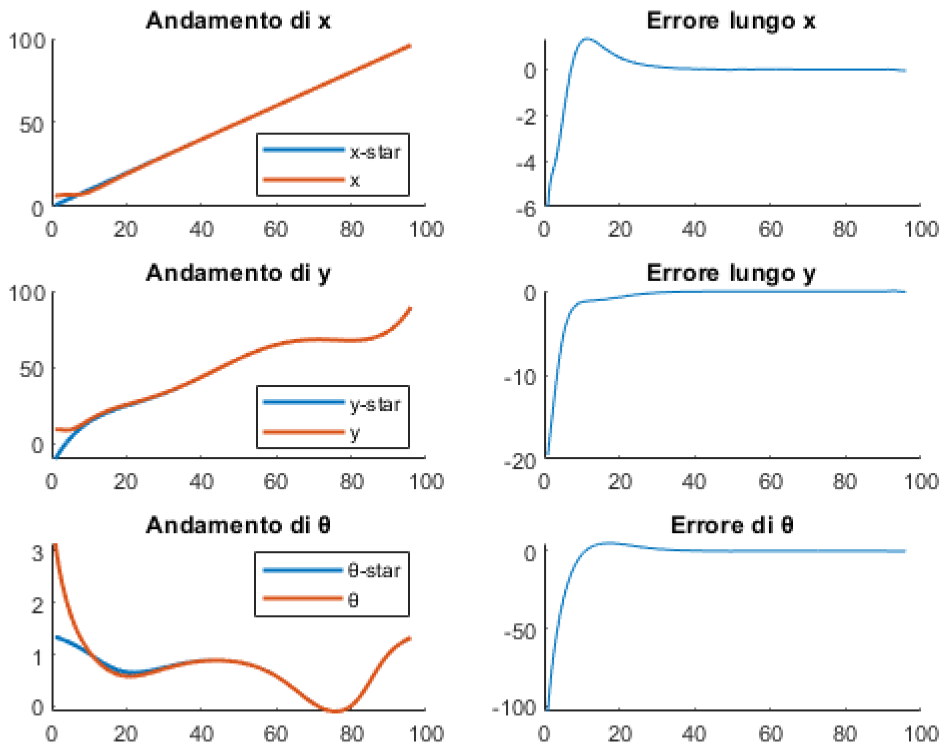
In the trajectory tracking section, after having calibrated the values of δ and *a,* the *robot_control_lin* routine also is called. The latter is implemented in the homonymous file present in the +*Controllers* folder, which is responsible for implementing the law, calculating the v and ω values and applying them to the robot model.

At this time, *ode45()* routine can be executed to make the system evolve over time.

For example, considering the trajectory generated by artificial potential fields:



it is possible to notice how the path followed by the robot converges with the reference one. In the same way, the values of the variables x, y and θ, respectively position and orientation of the robot, converge with the corresponding reference variables, while the following error tends to vanish:



Non-linear control

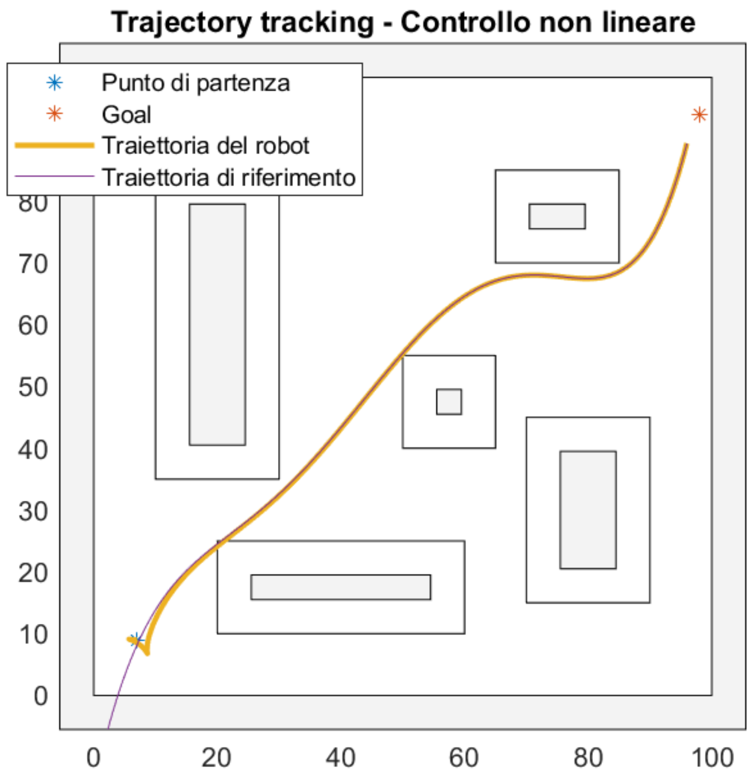
In this case, no linearization step is performed. The control law turns out to be:

$$\begin{cases} v = v^* \cos(\theta^* - \theta) + K_1(v^*, \omega^*) e_x \\ \omega = \omega^* + K_2 v^* \dfrac{(\sin(\theta^* - \theta))}{\theta^* - \theta} e_y + K_3(v^*, \omega^*)(\theta^* - \theta) \end{cases}$$

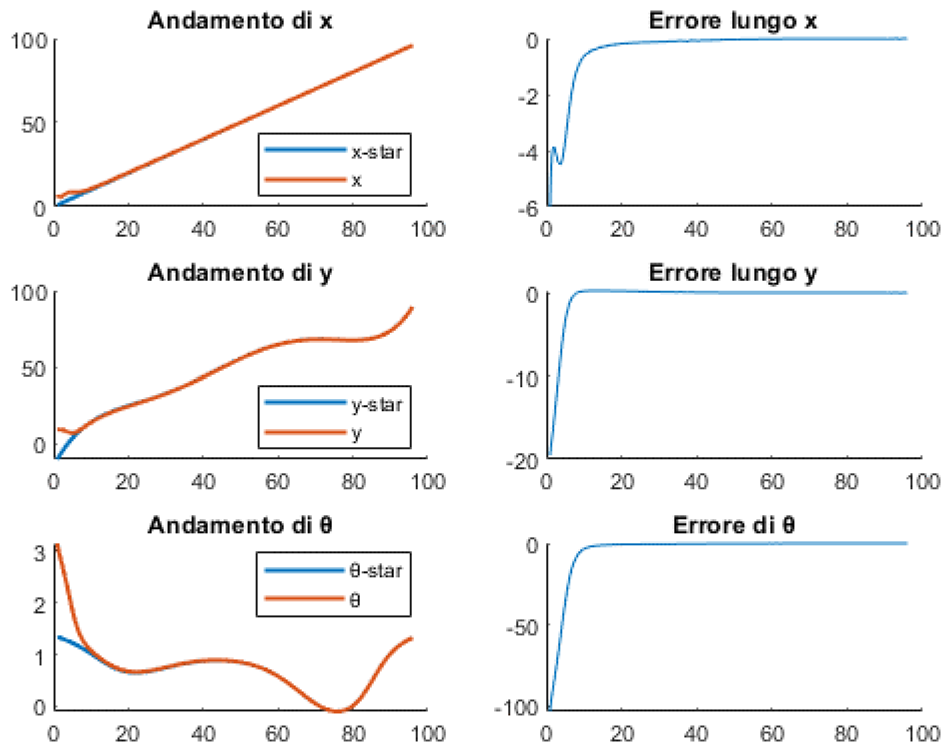
With $K_2 > 0$, $K_1(v^*, \omega^*) > 0$ and $K_3(v^*, \omega^*) > 0$ both bounded with *bound-ed derivatives*.

As before, it is implemented in the *robot_control_NL ( )* function.

Considering once again the trajectory generated through artificial potential fields, after a short transient period, the trajectory of the robot converges with the reference one:



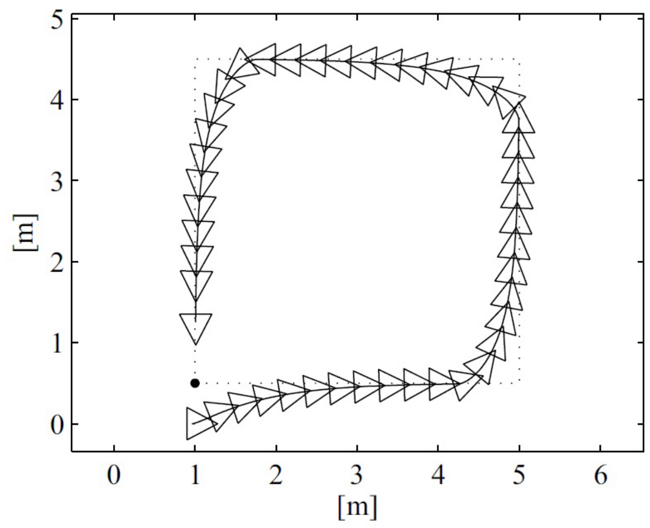
Also in this case, all the variables converge to the desired values and the error tends to vanish:



Input – output linearization

this technique, also known as *feedback linearization,* looks for a nonlinear control law based on virtual inputs such that the relation between these virtual inputs and the system outputs is linear.

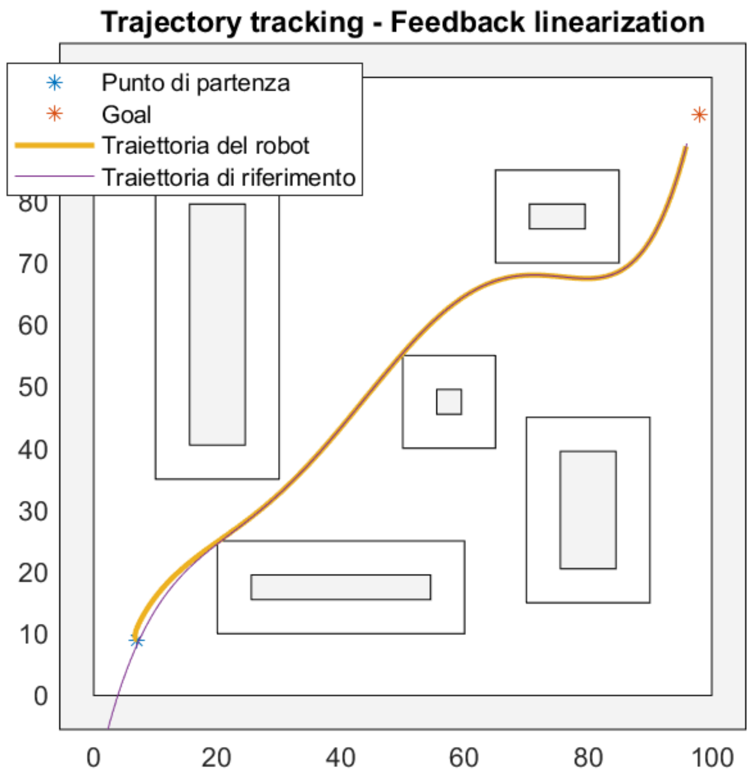
In the case of the robot, a point B is considered at a distance $|b|$ from the center of the robot. It will be the point B that follows the reference trajectory, while the robot "follows" B. All this is illustrated in the following figure, where the trajectory is represented by the dotted line, while the black dot is $B$and the triangle is the robot:



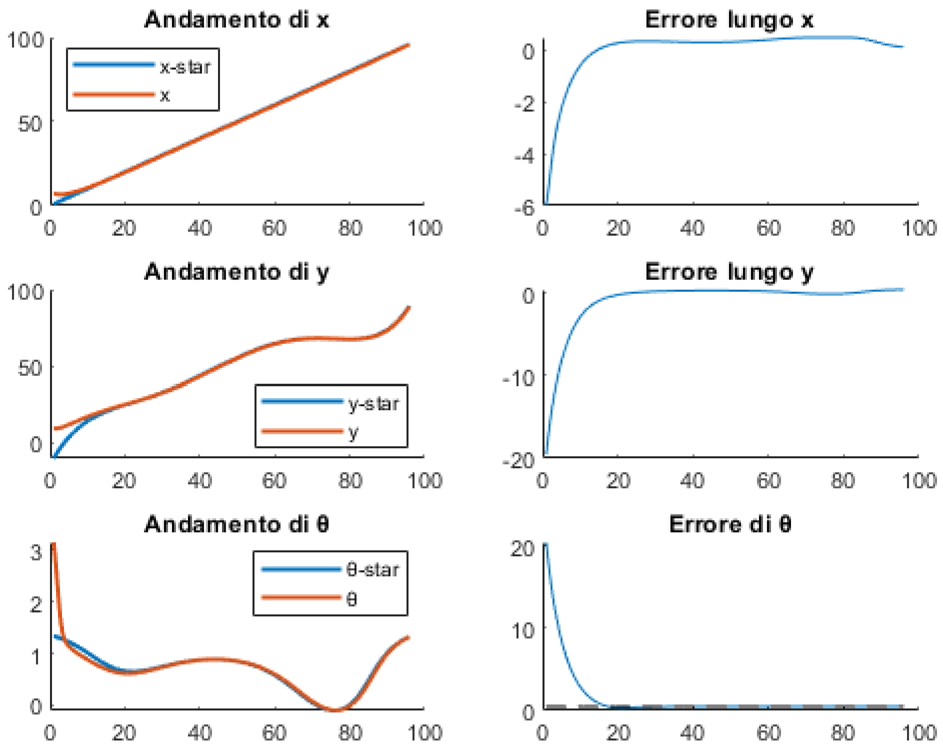
The control law is as follows:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -b\sin(\theta) \\ \sin(\theta) & b\cos(\theta) \end{bmatrix}^{-1} \begin{bmatrix} \dot{x}_B^* + k_x(x_B^* - x_B) \\ \dot{y}_B^* + k_y(y_B^* - y_B) \end{bmatrix}$$

Also in this case the law is implemented in the external method *robot_ control_FL()* with the following result in case the trajectory is the one built starting from the method of artificial potential fields:



With the following variable values:



Note how, in this case, the position error is never zero. This occurs due to the nature of this law: since the robot doesn't directly follow the trajectory but instead tracks point B, the error is precisely caused by the distance $|b|$ between the robot's center and B.

Posture regulation

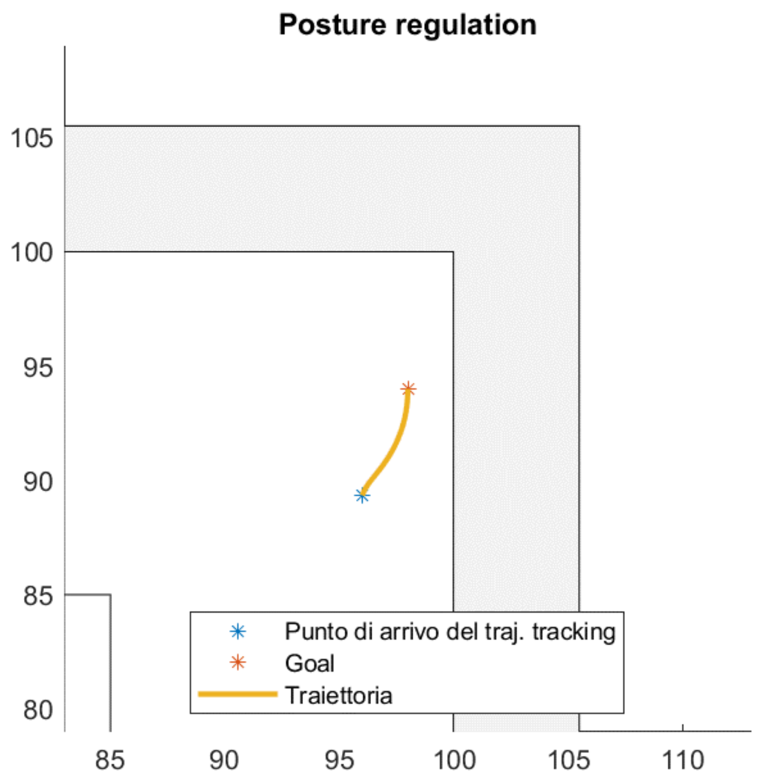
The last thing to do is bring the robot into the "goal" pose defined by the user in the initial phase. To do this, the following control law applies:

$$\begin{cases} v = K_1 \rho \cos(\gamma) \\ \omega = K_2 \gamma + K_1 \dfrac{\sin(\gamma)\cos(\gamma)}{\gamma}(\gamma + K_3 \delta) \end{cases}$$

With: $K_1 > 0, K_2 > 0, K_3 \in \mathbb{R}$, and:

$$\begin{cases} \rho = \sqrt{e_x^2 + e_y^2} \\ \gamma = \tan_2^{-1}\left(\dfrac{e_y}{e_x}\right) - \theta \\ \delta = \gamma - e_\theta \end{cases}$$

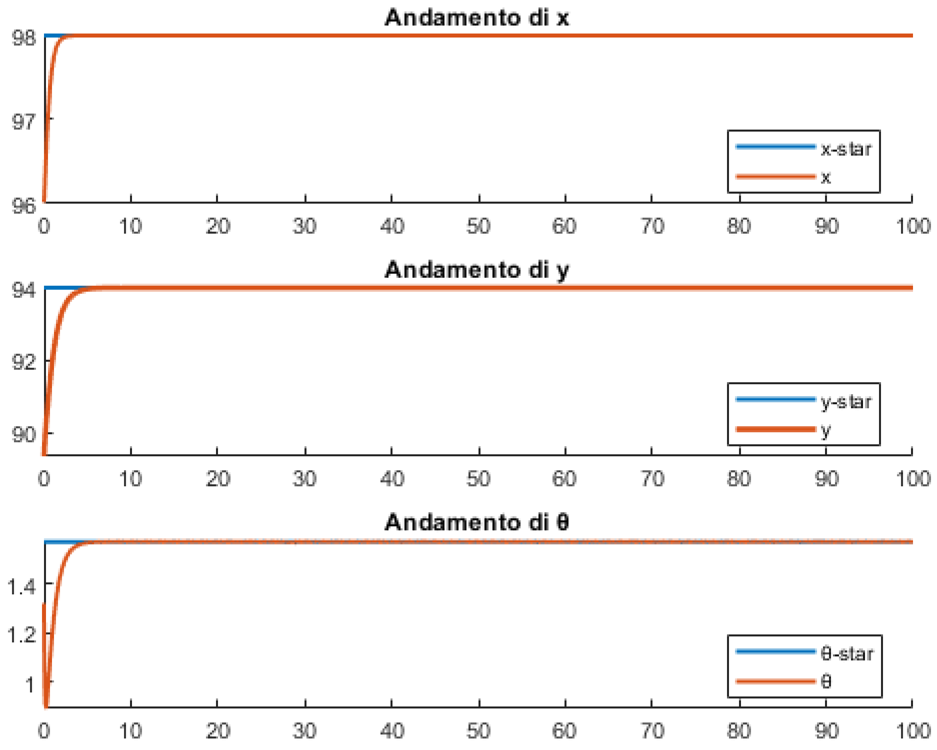
In the posture regulation section of the Mobile.mlx file, after selecting the parameters k1, k2, and k3, the *complete_posture_regulation()* routine is called, which is responsible for implementing the control law. Then, *ode45()* is executed to evolve the system, resulting in the following outcome:



EXAMPLE WITH ARTIFICIAL POTENTIAL FIELDS AND LINEAR CONTROL

The figure shows the trajectory generated by the controller to go from the arrival point of the trajectory generated in the trajectory tracking phase to the goal point decided by the user.

The trend of the variables is shown below:



Robot mobile block

In the previous paragraph, we saw how to derive the control signals to be fed to the robot. In this paragraph, we will focus on understanding how to use these commands within the robot's *model*.

The model of a mobile robot is a mathematical or conceptual representation that describes its behavior. It can be used for simulation, control, or understanding the robot's operation in different contexts, and may include information such as the robot's physical dimensions, points of contact with the environment, movement capabilities, physical limitations, kinematic or dynamic equations, sensor parameters, and more.

As mentioned earlier, we will adopt the *unicycle* model: a simple mathematical model used to describe the movement of a two-wheeled mobile robot. This simplified model assumes that the robot moves in a two-dimensional plane.

In the unicycle model, the robot is represented by two main variables:

1. Position, generally indicated by two coordinates, x and y, representing its location in the plane.
2. Orientation, usually represented by an angle, θ, indicating the direction the robot is facing with respect to the reference axis.

Furthermore, it is assumed that the robot moves through the combination of two actions:

1. Linear velocity, indicating the forward speed of the robot along its orientation.
2. Rotational velocity, indicating the speed at which the robot rotates around its center.

Using these two pieces of information, the unicycle model can predict how the robot's position and orientation change over time, given a certain linear velocity and constant rotational velocity.

It is essential to emphasize that the model in question is a simplification of reality and does not account for factors such as wheel slippage, internal dynamics of the robot, or the presence of obstacles in the environment.

From a mathematical perspective, the model takes the following form:

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \omega \end{cases}$$

where $\dot{x}$, $\dot{y}$ and $\dot{\theta}$ are respectively the coordinates of the desired position and orientation, while v and ω are the signals from the controller representing linear velocity and rotational velocity.

As for the implementation, it has been divided into two parts. The first part concerns the definition of the robot structure: assuming that the shape of the robot can be circumscribed with a circumference of radius $r = 2.5$, it is sufficient to define a variable that stores that value:

```
% MOBILE ROBOT
diameter = 5;
```

The second part concerns the actual implementation of the model:

```
xdot = [
        v*cos(theta);
v*sin(theta);
w
];
```

The latter is used within the controllers when needed.

This was the last part of the control scheme that needed to be implemented. At this point the unicycle robot can move in a pre-established environment avoiding obstacles, starting from a start point and reaching a goal point decided by the user.

All requests were met.

Conclusions

The present project provided an excellent opportunity to acquire new skills in industrial and mobile robotics. The analysis of the structure of an anthropomorphic robot in industrial robotics and the design of a robot with autonomous movement in the pre-established environment in mobile robotics allowed to face technical and conceptual challenges.

Through the development of algorithms and the simulation of specific trajectories, the potential of anthropomorphic robots in the automation of industrial activities has been demonstrated. Similarly, the mobile robot, equipped with navigation algorithms, demonstrated the ability to move safely and efficiently within the pre-set environment.

Furthermore, this work has provided a solid knowledge base in robotics, opening prospects for further research and development in the field. The acquired experience represents a starting point for future applications and benefits of anthropomorphic and mobile robots in various sectors.

In summary, working on this project has been an excellent opportunity to broaden skills in industrial and mobile robotics, tackling challenges and gaining new perspectives for innovation in robotics.