Progetto di robotica mobile

A.A. 2021/2022
Corso di Robotica

Studentessa
Gaia Assunta Bertolino
Matricola 209507

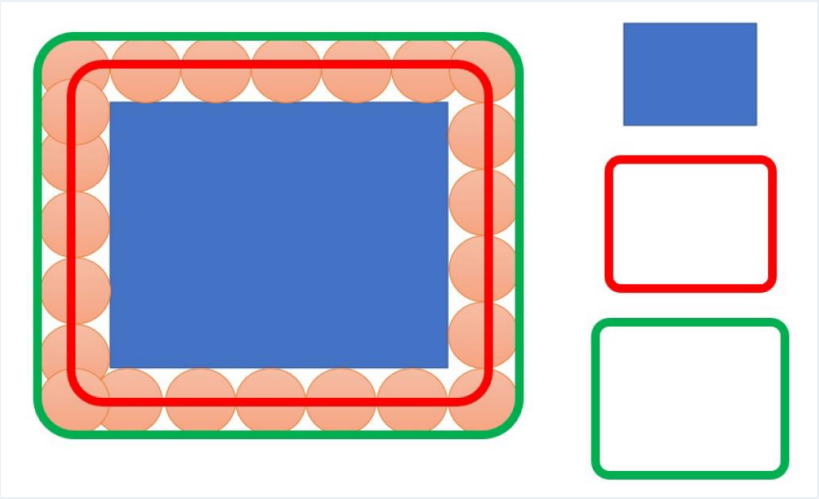
OBSTACLE MANAGEMENT

La robotica mobile regola i meccanismi di funzionamento dei robot che si muovono nello spazio e fornisce degli strumenti per il calcolo e il rispetto delle traiettorie. In particolare, anche un robot industriale può essere studiato dal punto di vista della robotica mobile quando esso è posto su una struttura con ruote che ne permettono lo spostamento.

Nella realizzazione della pianificazione, si procede a fare delle considerazioni sull'ambiente e sugli ostacoli presenti al fine di ottenere una planimetria che sia rappresentativa ma anche matematicamente più semplice per il movimento del robot.

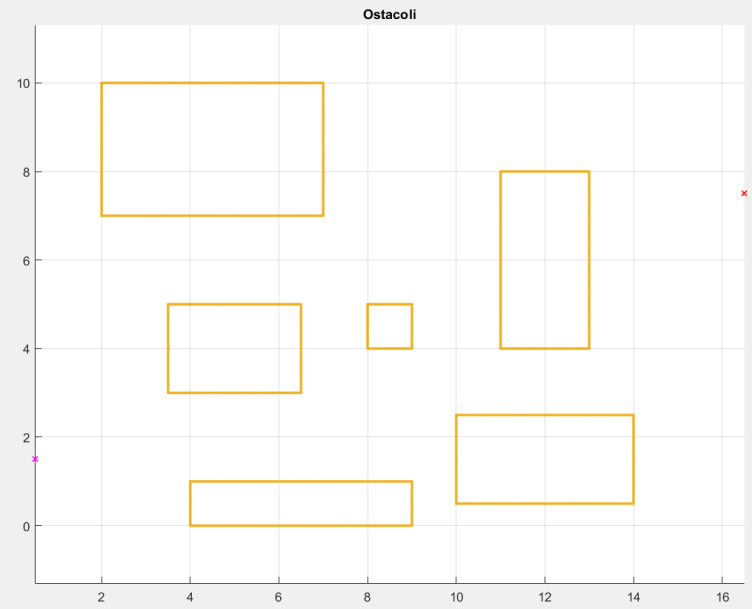
In particolare, gli ostacoli vengono approssimati a delle forme regolari in grado di inglobare ogni vertice. Inoltre, per rendere meno specifica la realizzazione, si considera il robot come puntiforme e si include il suo valore dimensionale all'interno della grandezza degli ostacoli. Tale tecnica è detta **ingrossatura degli ostacoli** e rende abbastanza certo che, muovendo il robot considerandolo come se la sua massa fosse tutta concentrata nel centro, esso non andrà a sbattere contro un ostacolo o, in base al tipo di ingrossatura realizzato, andrà al massimo a urtare uno dei suoi spigoli.

Esempio:



In blu l'oggetto
In arancio il robot mobile (di forma circolare)
In rosso l'area di movimento in cui il robot urta gli spigoli dell'oggetto
In verde l'area "sicura" di movimento a seguito

In particolare, la planimetria considerata ai fini del progetto è la seguente



Punto di start: (0.5, 1.5)
Punto di goal: (16.5, 8.5)
Numero di ostacoli: 6

Nel caso specifico, il robot dovrà muoversi verso destra per giungere verso il punto di goal evitando gli ostacoli. Per identificare se un punto qualsiasi si trova all'interno di uno degli ostacoli è utilizzata una funzione specifica

PATH PLANNING

Il path planning comprende un insieme di tecniche atte ad individuare l'insieme di punti che descrivono la traiettoria che il robot mobile deve seguire per andare da un punto di start ad uno di goal evitando di urtare e/o passare all'"interno" di un ostacolo.
Nel caso dello specifico progetto, gli ostacoli sono statici (non in movimento) e start e goal sono punti fissi noti a priori.

Le tecniche utilizzate nel progetto sono:

- Artificial potential fields
- Potenziali discreti
- Diagramma di Voronoi
- Grafi di visibilità

```
Command Window
  Scrivi
   - 1 per potenziali artificiali
   - 2 per Voronoi
   - 3 per potenziali discreti
   - 4 per grafi di visibilità
   >> 1
  Scrivi
   - 1 per errore linearizzato
   - 2 per errore non linearizzato
   - 3 per I/O linearization
   >> 1
fx >>
```

Inoltre, il codice realizzato permette di scegliere quale delle tecniche applicare attraverso la lettura dei valori inseriti in input

Artificial potential fields

Il concetto alla base della tecnica è la creazione di un campo potenziale artificiale che sia

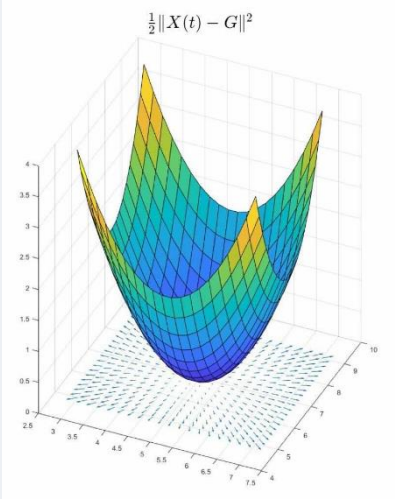
- attrattivo nel punto di goal
- repulsivo negli ostacoli
- con un minimo globale nella posizione di goal

La funzione potenziale è data dalla somma dei potenziali attrattivo e repulsivi dei vari ostacoli; entrambe le componenti sono moltiplicati per una costante che determina il peso dato ai due fenomeni virtuali. La funzione sarà dunque

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

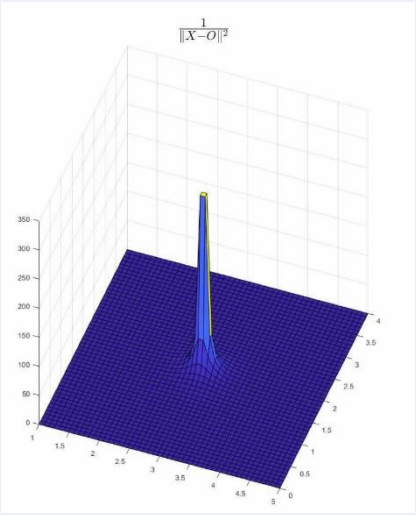
Dal punto di vista della scelta del potenziale attrattivo, per evitare situazioni di stallo nel movimento, si deve preferire una funzione che presenta un solo minimo globale e nessun altro punto a gradiente nullo (ovvero nessun ulteriore punto di minimo, massimo o sella).
La scelta può determinare la modalità di arrivo e dunque può essere influenzata dalle scelte circostanziali del progetto. Nel caso in esame, si utilizzerà un **paraboloide**.
Il percorso che seguirà il robot sarà in direzione dell'antigradiente ovvero diretto verso il minimo.

FUNZIONE POTENZIALE ATTRATTIVA



Per quanto riguarda il potenziale repulsivo, il vincolo è definito **hard** in quanto la funzione deve assolutamente evitare che il robot urti l'ostacolo. Di conseguenza, il valore del potenziale nel contorno dell'ostacolo è pari ad infinito ovvero deve avere un comportamento asintotico mentre, allontanandosi da esso, il potenziale tenderà sempre di più a zero ovvero l'azione è locale e dunque la sua definizione può anche essere a tratti.

FUNZIONE POTENZIALE REPULSIVA



Bisogna, inoltre, evitare che il contributo delle funzioni che descrivono i potenziali repulsivi siano diversi da zero nel punto di goal.

Nel caso in cui il robot si fermi (ovvero in gradiente sia zero) ma la distanza dal punto di goal sia eccessiva da ritenere di essere arrivati, si è bloccati in un punto di minimo locale. Per uscire dallo stallo si può, ad esempio, applicare

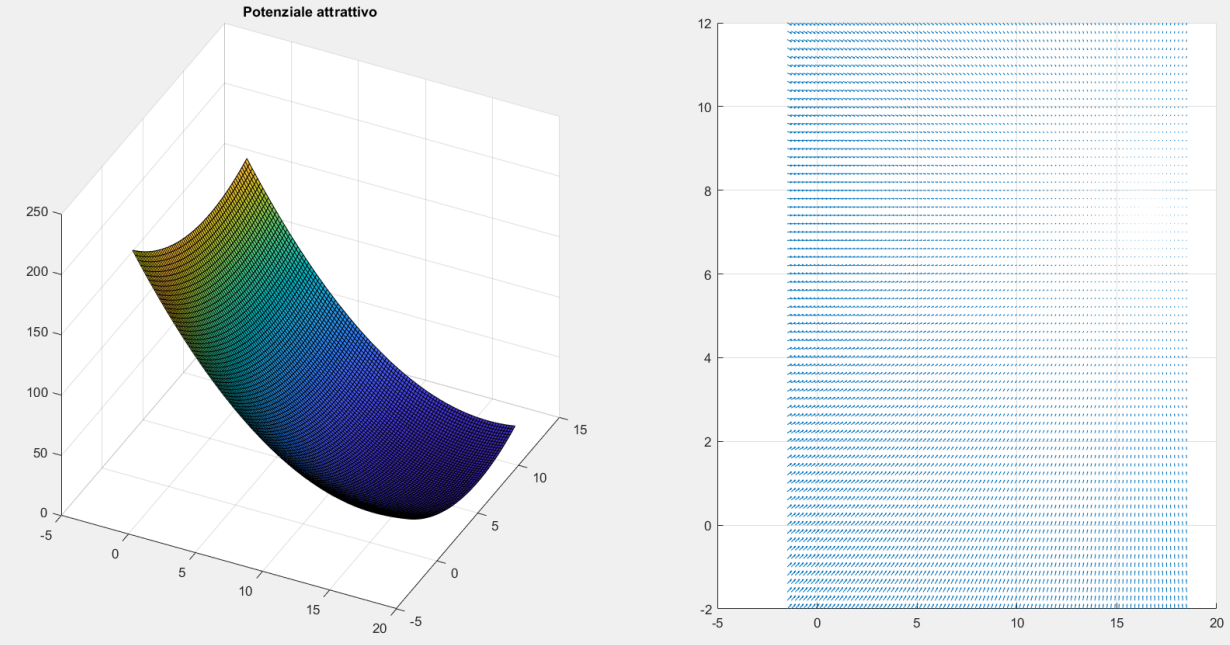
- la tecnica muro muro che aggira mantenendosi ad una certa distanza, l'ostacolo per poi riprendere il percorso
- la tecnica dei vortici che prevede di aggirare il vettore che compensa il movimento del robot
- la tecnica dello switching che prevede di alternare il movimento in avanti e quello per aggirare l'ostacolo

Codice di definizione e rappresentazione del potenziale attrattivo:

```
62    % Potenziali artificiali
63 -  if planner == 1
64
65        % Definizione dei potenziali attrattivo e rotanti
66        % Il potenziale attrattivo sarà un paraboloide
67        % Quelli repulsivi saranno degli arcotangente (dunque rotanti)
68 -      Ja = @(x,y,Gx,Gy)((1/2)*((x-Gx).^2+(y-Gy).^2));
69 -      Jr = @(x,y,Ox,Oy)(atan2(Oy-y,Ox-x)+pi);
70
71        % Calcolo dei gradienti
72 -      nablaJaX = @(x,y,Gx,Gy)(x-Gx);
73 -      nablaJaY = @(x,y,Gx,Gy)(y-Gy);
74 -      nablaJrX = @(x,y,Ox,Oy)((-Oy+y)./(((x-Ox).^2+(y-Oy).^2)));
75 -      nablaJrY = @(x,y,Ox,Oy)((Ox-x)./(((x-Ox).^2+(y-Oy).^2)));
76
77        % Regione di validità del potenziale repulsivo
78 -      d = 1;
79 -      r = @(x,y,Ox,Oy)((x-Ox).^2 + (y-Oy).^2 <= d^2);
80
81        % Pesi dei potenziali
82 -      wa = 20;
83 -      wo = 5;
84
```

```
85        % Rappresentazione del potenziale attrattivo
86
87 -      xx = xm-2:deltaXY:xM+2;
88 -      yy = ym-2:deltaXY:yM+2;
89 -      [XX,YY] = meshgrid(xx,yy);
90
91 -      Za = Ja(XX,YY,GOAL(1),GOAL(2));
92 -      nablaJaXX = nablaJaX(XX,YY,GOAL(1),GOAL(2));
93 -      nablaJaYY = nablaJaY(XX,YY,GOAL(1),GOAL(2));
94
95 -      figure(2);
96 -      subplot(121);
97 -      hold on;
98 -      grid on;
99 -      surf(XX,YY,Za); % plot della funzione
100 -     title('Potenziale attrattivo');
101 -     subplot(122);
102 -     hold on;
103 -     grid on;
104 -     quiver(XX,YY,-nablaJaXX,-nablaJaYY); % plot dell'antigradiente
```

Rappresentazione del potenziale attrattivo:



A destra è possibile vedere lo zoom del grafico sopra sulle frecce che descrivono la direzione del campo attrattivo.

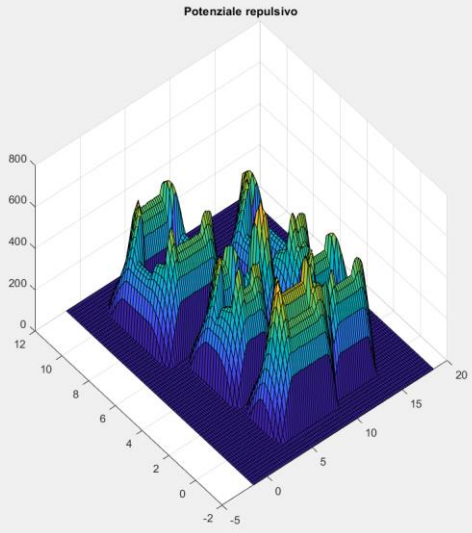
In particolare, la direzione di interesse per questa tecnica è quella descritta dall'antigradiente in quanto esso è direzionato verso il punto di minimo (in questo caso il goal), e in direzione contraria al gradiente

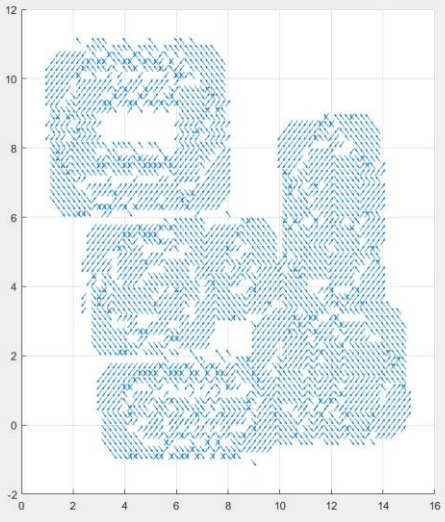
Per quanto riguarda il potenziale repulsivo, il meccanismo utilizzato è similare.

```
106         % Rappresentazione del potenziale repulsivo
107
108 -       Zr = zeros(size(Za));
109 -       nablaJrXX = zeros(size(nablaJaXX));
110 -       nablaJrYY = zeros(size(nablaJaYY));
111
112 -       for i=1:size(O,1)
113 -           oi = O(i,:);
114 -           Zr = Zr + Jr(XX,YY,oi(1),oi(2)).*r(XX,YY,oi(1),oi(2));
115 -           nablaJrXX = nablaJrXX + nablaJrX(XX,YY,oi(1),oi(2)).*r(xx,YY,oi(1),oi(2));
116 -           nablaJrYY = nablaJrYY + nablaJrY(XX,YY,oi(1),oi(2)).*r(xx,YY,oi(1),oi(2));
117 -       end
118
119         % Normalizzazione (permette una migliore rappresentazione)
120 -       nablaJrXXn = nablaJrXX./sqrt(nablaJrXX.^2 + nablaJrXX.^2);
121 -       nablaJrYYn = nablaJrYY./sqrt(nablaJrYY.^2 + nablaJrYY.^2);
122
123 -       figure(3);
124 -       subplot(121);
125 -       hold on;
126 -       grid on;
127 -       surf(XX,YY,Zr); % plot della funzione
128 -       title('Potenziale repulsivo');
129 -       subplot(122);
130 -       hold on;
131 -       grid on;
132 -       quiver(XX,YY,-nablaJrXXn,-nablaJrYYn);
```

Per rappresentare correttamente nel grafico il potenziale repulsivo si fa ricorso ad una normalizzazione

Rappresentazione del potenziale repulsivo:





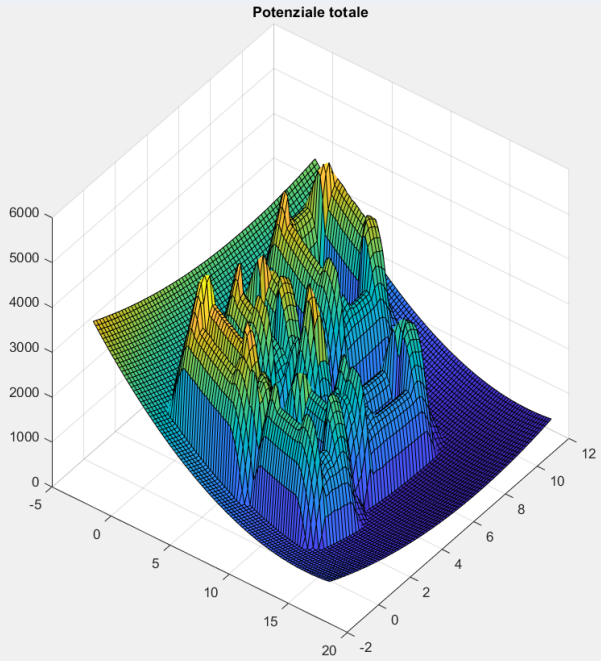
Il **gradiente totale** è dato dalla somma dei potenziali attrattivo e repulsivo.

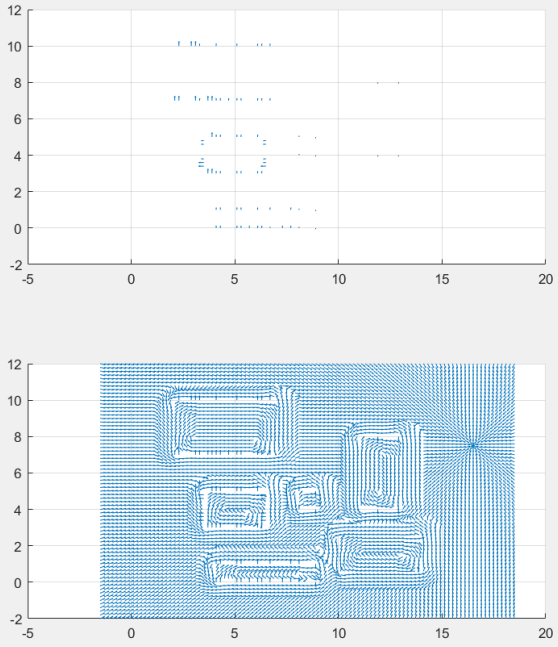
```
135        % Gradiente totale
136 -      J = wa*Za + wo*Zr;
137 -      nablaJx = wa*nablaJaXX + wo*nablaJrXX;
138 -      nablaJy = wa*nablaJaYY + wo*nablaJrYY;
139
140        % Normalizzazione (permette una migliore rappresentazione)
141 -      nablaJxn = nablaJx./sqrt(nablaJx.^2 + nablaJy.^2);
142 -      nablaJyn = nablaJy./sqrt(nablaJx.^2 + nablaJy.^2);
143
144 -      figure(4);
145 -      subplot(2,2,[1 3]);
146 -      hold on;
147 -      grid on;
148 -      surf(XX,YY,J);
149 -      title('Potenziale totale');
150 -      subplot(222);
151 -      hold on;
152 -      grid on;
153 -      quiver(XX, YY, -nablaJx, -nablaJy);
154 -      subplot(224);
155 -      hold on;
156 -      grid on;
157 -      quiver(XX,YY, -nablaJxn, -nablaJyn);
```

In tale somma concorrono due pesi wa e wo che regolano quanto rispettivamente i due potenziali siano preponderanti.

Ad esempio, aumentando il valore del peso wa si darà più peso al potenziale attrattivo all'interno della funzione totale. Ciò comporterà un fenomeno di attrazione più forte verso il punto di goal.

Rappresentazione del potenziale totale:



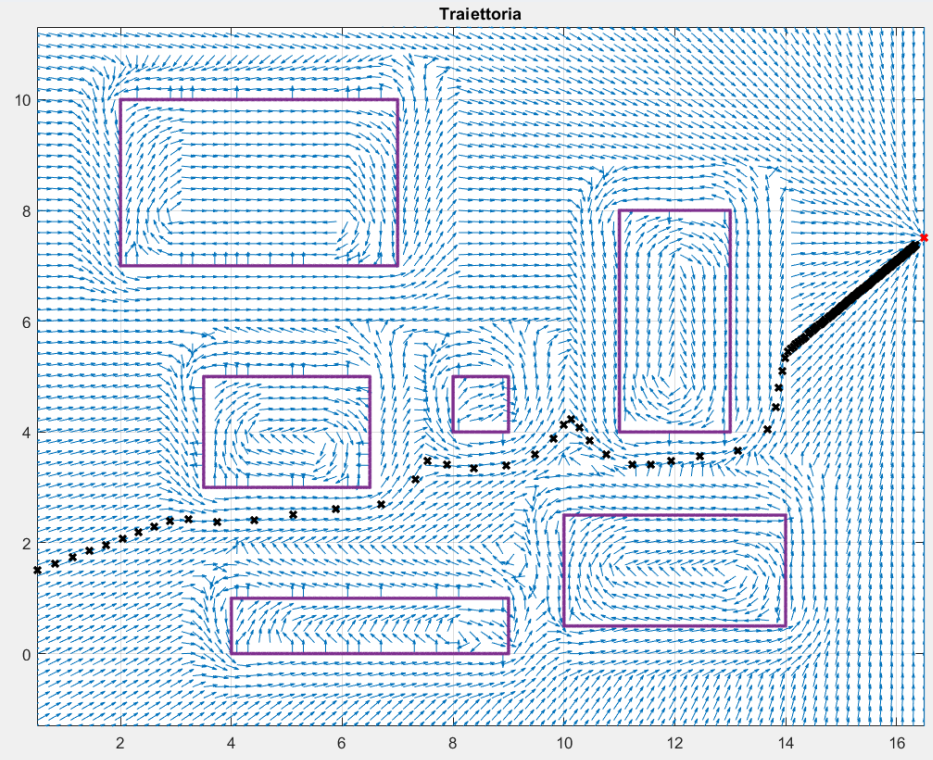


L'algoritmo che individua il cammino ricorre al calcolo della funzione totale nel punto in cui si trova e procede ad individuare il punto successivo in cui muoversi fino a quando esso non è in un intorno sufficientemente accettabile da poter concludere di essere giunti nel punto di goal.

Nel caso del progetto è stato utilizzato un valore pari a 0.2.

```
159         % Algoritmo di ottimizzazione
160 -       passo = 0.001; % passo non adattativo ma costante
161 -       intorno = 0.2;
162 -       iter = 1000; % limite iterazioni
163 -       X = zeros(2,iter);
164 -       X(:,1) = START;
165
166 -       for k=2:iter
167 -           Xcorr = X(:,k-1);
168 -           nablaA = [ nablaJaX(Xcorr(1), Xcorr(2), GOAL(1), GOAL(2));
169                        nablaJaY(Xcorr(1), Xcorr(2), GOAL(1), GOAL(2))]; % gradiente attrattivo
170 -           nablaR = [0;0];
171           % ciclo che calcola i potenziali repulsivi degli ostacoli
172 -           for i=1:size(O,1)
173 -               oi = O(i,:);
174 -               nablaR = nablaR + [ nablaJrX(Xcorr(1),Xcorr(2),oi(1),oi(2))*r(Xcorr(1),Xcorr(2),oi(1),oi(2));
175                                    nablaJrY(Xcorr(1),Xcorr(2),oi(1),oi(2))*r(Xcorr(1),Xcorr(2),oi(1),oi(2))];
176 -           end
177 -           nablaJ = wa*nablaA + wo*nablaR;
178 -           Xsucc = Xcorr - passo*nablaJ;
179 -           X(:,k) = Xsucc;
180
181           % Verifica dell'intorno
182 -           if norm(Xsucc-GOAL) <= intorno
183 -               break;
184 -           end
185 -       end
```

La traiettoria ottenuta dalla tecnica dei potenziali artificiali è la seguente:

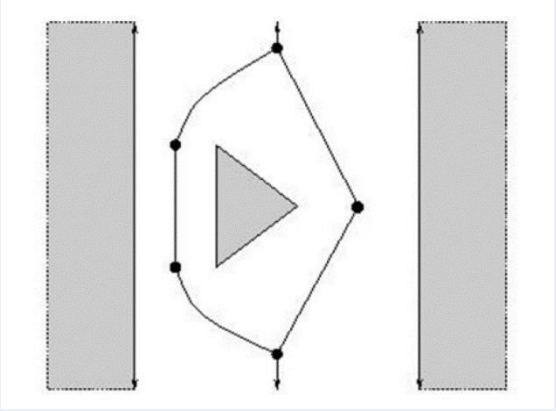


Nel grafico sono evidenti le frecce direzionali del campo totale che descrivono un movimento vorticoso nelle vicinanze degli ostacoli.

Tale meccanismo è responsabile dell'allontanamento del robot dall'ostacolo, aggirandolo.

Diagrammi di Vononoi

Tale tecnica prevede di calcolare un diagramma dei punti a distanza minima fra gli ostacoli, detto appunto diagramma di Voronoi. Si ottiene da questo procedimento un grafo di archi che possono essere percorsi, attraverso un opportuno algoritmo (ad esempio di Dikjstra), fino al goal.



Esempio di grafo di Voronoi

Il grafo garantisce che, muovendosi lungo di esso, non si urteranno degli ostacoli ma si rimarrà sempre ad una certa distanza.

Il meccanismo di costruzione del grafo è considerato uno step "**offline**" ai fini progettuali ovvero eseguito una sola volta a causa del suo costo computazionale e della staticità degli ambienti in cui viene applicato mentre la scelta del percorso è considerata la parte "**online**" ovvero eseguita in tempo reale in quanto influenzata dalle posizioni di start e goal.

Nella realtà, il diagramma ottenuto offline può essere eventualmente rimodellato, solitamente nel caso in cui cambia l'ambiente.

```
214      % Aggiunta del perimetro esterno
215 -    O = [O; rect(muro(1),muro(2),muro(3),muro(4))];
216
217      % Creazione grafo di Voronoi
218 -    [vx, vy] = voronoi(O(:,1),O(:,2));
219 -    i = 1;
220 -    x = START(1);
221 -    y = START(2);
222 -    xg = GOAL(1);
223 -    yg = GOAL(2);
224
225      % Processo di eliminazione
226 -    for k=1:length(vx)
227 -       if cont(vx(k),vy(k))
228 -           vx(k) = 1E100;
229 -           vy(k) = 1E100;
230 -       end
231 -    end
```

Nel computo del grafo, sono inseriti come "ostacoli" anche i punti appartenenti al muro perimetrale; tale passaggio permette di percorrere anche lo spazio che intercorre fra un muro e un ostacolo.
Nel progetto, il perimetro degli ostacoli viene considerato come un insieme di punti; dunque, per rimuovere i punti dal grafo che ricadono all'interno di un ostacolo è realizzato un ciclo di eliminazione.

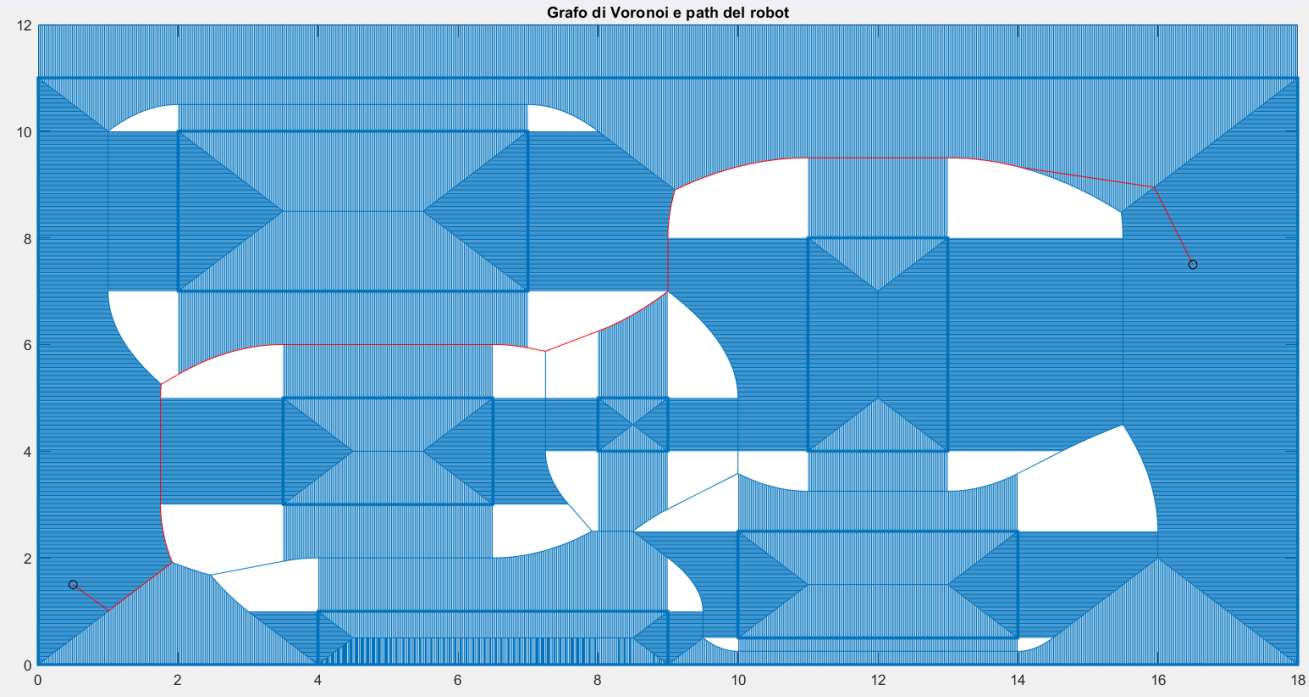
Il processo di individuazione del cammino si arresta nel momento in cui una delle due coordinate si trova a meno di una distanza stabilita da quella del goal; da questo punto il movimento del robot sarà lineare fino a giungere al goal.

```
233    % Ciclo di individuazione del cammino
234 -  path = [x;y];
235 -  while (abs(x - xg) > 2)||(abs(y - yg) > 2)
236 -      distVoro = sqrt((vx - x).^2+(vy - y).^2);
237 -      distGoal = sqrt((xg - x).^2+(xg - y).^2);
238
239        % Trova il vertice di Voronoi più vicino
240 -      [mn ind] = min(distVoro(:));
241 -      xt = vx(ind);
242 -      yt = vy(ind);
243 -      goalj = sqrt((xg - xt).^2+(xg - yt).^2);
244 -      if (goalj < distGoal)
245 -          if distVoro(ind) > 0.1
246 -              xn = linspace(x,xt,20);
247 -              yn = y + (yt-y)*(xn-x)/(xt-x);
248 -              path = [path, [xn; yn]];
249 -          else
250 -              path = [path, [x;y]];
251 -          end
252 -          x = xt;
253 -          y = yt;
254 -      end
255 -       vx(ind) = 1E100;
256 -       vy(ind) = 1E100;
257 -  end
258
259    % Aggiunta dei punti finali per giungere al goal
260 -  xn = linspace(x,xg,50);
261 -  yn = y + (yg-y)*(xn-x)/(xg-x);
262 -  path = [path, [xn; yn]];
```

Ad ogni passo viene individuato il vertice più vicino che riduce la distanza del punto precedente dal goal.

Nel momento in cui il punto più vicino risulta essere comunque ad una distanza maggiore di un valore specifico (impostato in questo caso a 0.1), nel path vengono inseriti dei punti di intermezzo affinché il movimento non sia esageratamente repentino.

Rappresentazione del grafo di Voronoi ottenuto e del path calcolato:

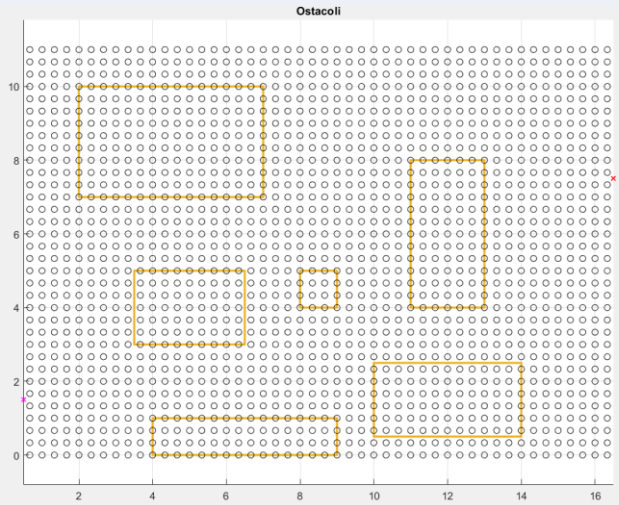


Il path, rappresentato in rosso, presenta dei movimenti in linea retta fra il punto di start e il punto di Voronoi più vicino e fra il primo punto nell'intorno del goal e il goal stesso.

Potenziali discreti

Il concetto alla base è la creazione di un campo potenziale discreto è similare a quello dei campi artificiali; la differenza sostanziale risiede nella schematizzazione dell'ambiente che viene suddiviso in celle e a ciascuna di esse è associato un valore.
Tali valori sono associati a partire dalla cella di goal che conterrà il valore zero; in successione, si visitano le celle adiacenti e, se esse sono vuote, si pone il valore di k+1 o infinito se la cella contiene anche solo un pezzo degli ostacoli.

Rappresentazione dell'ambiente:



Lo spazio bianco fra i punti rappresenta lo spazio percorribile dal robot.

Inoltre, i vertici sono esclusi dal movimento, dunque un robot puntiforme si potrà muovere ad una certa distanza dal bordo dell'ostacolo. Ciò aggiunge un ulteriore livello di sicurezza rispetto all'ingrossamento dell'ostacolo.

Ogni spazio bianco rappresenta una cella della matrice utilizzata per il calcolo del path

```
% Assegnazione dei numeri alla matrice

xg = GOAL(1);
yg = GOAL(2);
ixg = round(xg/alpha);
iyg = muro(2)*2/alpha-floor(yg/alpha);
x = ixg;
y = iyg;
punti = [x y];
i = 1;
index = 0;
visited = [x y];
 while i<dx*dy
     for j=x-index:x+index % indice lungo l'ascissa3
         if (j>0 & j<=(muro(1)*2/alpha))
             for k=y-index:y+index % indice lungo l'ordinata
                 if (k>0 & k<=(muro(2)*2/alpha))
                     %fprintf('j = %d, k = %d\n', j,k);
                     if ((j==x-index | j==x+index | k==y-index | k==y+index) & x>0 & x<=(muro(1)*2/alpha) & y>0 & y<=(muro(2)*2/alpha))

                         x1 = j*alpha;
                         y1 = ((muro(2)*2/alpha-k)*alpha);
                         x2 = j*alpha;
                         y2 = ((muro(2)*2/alpha-k+1)*alpha);
                         x3 = (j-1)*alpha;
                         y3 = ((muro(2)*2/alpha-k)*alpha);
                         x4 = (j-1)*alpha;
                         y4 = ((muro(2)*2/alpha-k+1)*alpha);

                         if (j==ixg & k==iyg)
                             discr(k,j) = 0;
                         elseif (discr(k,j) == 0 & (cont(x1,y1) | cont(x2,y2) | cont(x3,y3) | cont(x4,y4)))
                             discr(k,j) = 500;
                             i = i+1;
                         elseif discr(k,j) == 0
                                 discr(k,j) = index;
                             i = i+1;
                         else
                         end
                     end
                 end
             end
         end
     end
 index = index+1;
 end
```

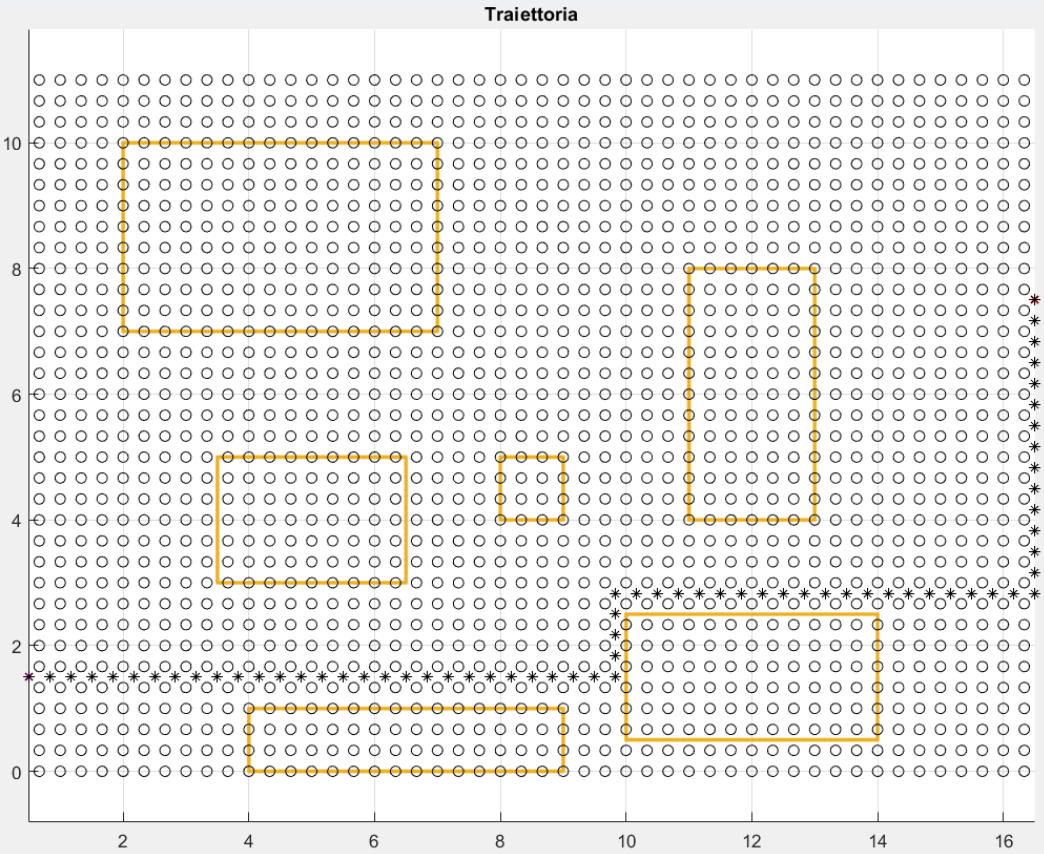
```
% Algoritmo di movimento
xs = round(START(1)/alpha);
ys = muro(2)*2/alpha-floor(START(2)/alpha);
val = discr(ys,xs)-1;
ret = [xs,ys];
salire = true;
path = zeros(length(discr(:,1)),length(discr(1,:)));
path(ys,xs) = 1;
x = xs;
y = ys;
xg = round(GOAL(1)/alpha);
yg = muro(2)*2/alpha-floor(GOAL(2)/alpha);
while x ~= xg | y  ~= yg
    if (x+1<=xg & discr(y,x+1) == val-1)
        x = x+1;
        val = val-1;
        salire = true;
        path(y,x) = 1;
    elseif x+1<=xg & discr(y,x+1) == val
        x = x+1;
        salire = true;
        path(y,x) = 1;
    else
        if (salire == true & y-1<=ys & discr(y-1,x) <= val)
            if discr(y-1,x) == val
                y = y-1;
            else
                y = y-1;
                val = val-1;
            end
            path(y,x) = 1;
        elseif y+1 > 0 & discr(y+1,x) <= val
            if discr(y+1,x) == val
                y = y+1;
            else
                y = y+1;
                val = val-1;
            end
            salire = false;
            path(y,x) = 1;
        end

    end
end
```

Il processo di individuazione del percorso può avvenire cella per cella secondo la tecnica del **backtracking** ovvero dal punto di partenza si visita la cella adiacente con valore minore (o, a parità, più vicina al goal) e, nel caso in cui si incontrino solo celle con valore uguale o maggiore, allora si ritorna indietro di una scelta, fino ad arrivare allo zero (punto di goal).

Nel progetto viene utilizzato un meccanismo di movimento nella cella più a destra (e dunque più vicina al goal). Nel caso in cui si incontra un ostacolo, il robot si sposta di una cella in alto e ricomincia il movimento verso destra; il processo viene iterato fino a quando si giunge al goal o si incontra un ostacolo in alto, il che porta il robot alla posizione in basso dalla quale ha iniziato a salire e fa tentare il percorso verso il basso prima scartato.

Tale meccanismo garantisce l'aggiramento di un ostacolo per tentativi.

Traiettoria ottenuta:



Grafi di visibilità

Seppur lontana dal movimento umano, tale tecnica prevede di definire tutti i vertici degli ostacoli che diventano i nodi del grafo e di collegare fra di loro tutte le coppie di nodi che si "**vedono**" ovvero collegabili senza che passare sopra uno degli ostacoli.
Come per i grafi di Vononoi, anche in questo caso si ha una fase offline che corrisponde alla creazione del grafo e una fase online che prevede il calcolo degli archi di congiunzione da percorrere.

```
    points = [];
    full = [];
    for i=1:length(v(:,1))
         x1 = v(i,1);
         y1 = v(i,2);
        for j=1:length(v(:,1))
            x2 = v(j,1);
            y2 = v(j,2);
            noObs = true;
            if (x1 ~= x2 | y1 ~= y2)
                full = [full; x1 y1 x2 y2 sqrt((x2-x1)^2+(y2-y1)^2)];
                 x = linspace(x1,x2,5000);
                 y = y1 + (y2-y1)*(x-x1)/(x2-x1);
                 retta = [x' y'];
                 n = 1;
                 while noObs == true & n <= length(x)
                     if cont(retta(n,1), retta(n,2))
                         noObs = false;
                     else
                         n = n+1;
                     end
                 end
                if noObs == true
                    points = [points; x1 y1 x2 y2 sqrt((x2-x1)^2+(y2-y1)^2)];
                end
            end
        end
    end
```

Attraverso due cicli di for si individuano tutte le coppie di vertici possibili. Successivamente, per ogni coppia si definisce la retta che li unisce e si elimina tale coppia di punti se anche solo uno degli elementi della retta risulta essere compreso negli ostacoli.

Attraverso questa esclusione nella variabile points saranno presenti solamente le coppie di punti che riescono a "vedersi" e la loro distanza

Algoritmo di scelta del cammino:

```
    % Algoritmo di scelta del cammino
    xg = GOAL(1);
    yg = GOAL(2);
    xs = START(1);
    ys = START(2);
    x = xs;
    y = ys;
    arrived = false;
    path = [];

    while arrived ~= true
        dist = 0;
        xn = 0;
        yn = 0;
        i = 1;
        while i<=length(points(:,1))
            if (points(i,1) == x & points(i,2) == y & points(i,5)>dist)
                dist = points(i,5);
                xn = points(i,3);
                yn = points(i,4);
                points = setdiff(points,points(i,:),'rows');
            else
                i = i+1;
            end
        end
        j = 1;
        while j<=length(points(:,1))
            if ((points(j,3) == x & points(j,4) == y) |(points(j,3) == xn...
                    & points(j,4) == yn) | (points(j,1) == x & points(j,2) == y) ...
                    | points(j,3) <= x | points(j,3) <= xn)
                points = setdiff(points,points(j,:),'rows');
            else
                j = j+1;
            end
        end
        path = [path; x y xn yn];
        if (xn == xg & yn == yg)
            arrived = true;
        end
        x = xn;
        y = yn;

    end
```

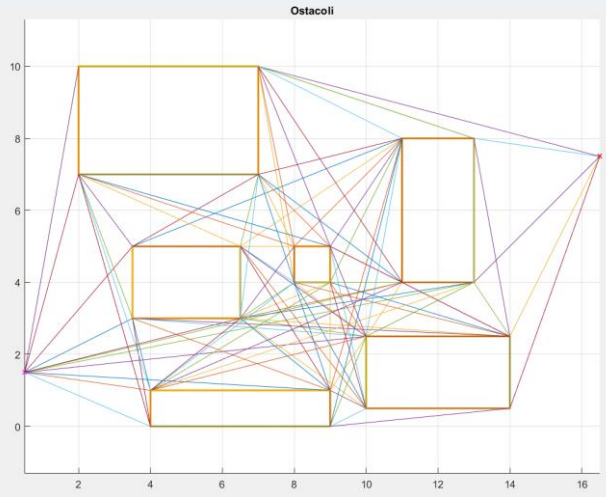
Il procedimento prevede di partire dal punto di start.

Il primo ciclo di while seleziona la coppia che individua il punto che dista maggiormente, così da avvicinarsi il più possibile al goal.

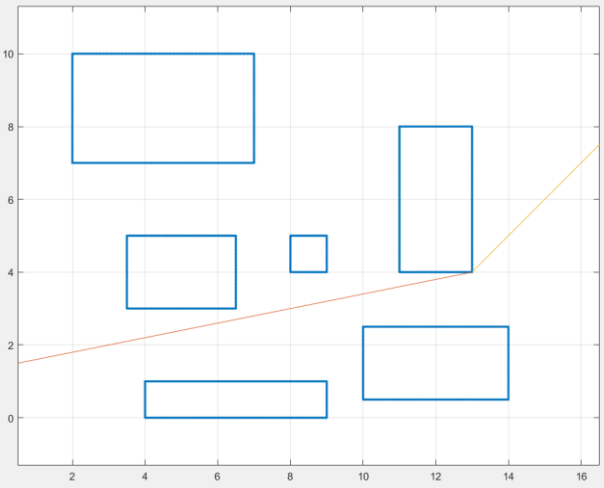
Il secondo ciclo di while procede invece ad eliminare tutte le coppie che dallo start portano ad un altro vertice qualsiasi o quelle coppie che giungono in esso; ciò scongiura la possibilità di ripassare nel punto già attraversato.

L'algoritmo viene iterato per il nuovo punto trovato fino a quando non si giunge nel punto di goal che in precedenza è stato inserito fra i vertici affinché fosse tenuto in considerazione per il grafo

Grafo completo di punti che si “vedono”



Path ottenuto dall’algoritmo del cammino



CONTROL PLANNING

Il problema di controllo prevede di descrivere quanto il path calcolato venga rispettato attraverso delle leggi specifiche.
Il problema prevede un compito di tipo non lineare (nonostante venga utilizzata un'alta approssimazione) a causa della presenza di seni e coseni e ciò comporta la difficoltà maggiore nella risoluzione del task insieme al seguire in maniera esatta la traiettoria pianificata per evitare di urtare contro degli ostacoli.

Si parla di

- **trajectory tracking** ovvero il problema di fare in modo che una determinata traiettoria individuata venga seguita
- **posture regolation** ovvero il problema di fare in modo che il robot giunga al punto di goal con un determinato orientamento

Le tecniche di trajectory tracking sono di tre tipologie:

- controllo basato su approssimazione lineare che utilizza la linearizzazione approssimata dell'errore
- controllo non linearizzato
- input/output linearization

In particolare, è necessario che la traiettoria sia ammissibile per il robot in questione e ciò è assicurato se si utilizzano delle metodologie di calcolo della stessa come quelle indicate precedentemente.

Le leggi indicate valgono per l'uniciclo (o per similarità per il differential drive) ma non per modelli come il car-like.

Il codice del control planning prevede l'uso di una funzione **trapveltraj** che restituisce una traiettoria fra un insieme di punti dati insieme alla velocità, all'accelerazione e altri parametri

```
575     % PARTE DI CONTROL PLANNING
576
577 -   misura = size(path);
578 -   theta = 0.001;
579 -   if misura(1,1) < 500
580 -       misura(1,1) = misura(1,1)*4;
581 -   end
582 -   [q, qd, qdd, tvec, pp] = trapveltraj(path', misura(1,1));
583 -   passo = [START(1), START(2), theta];
```

Controllo basato su approssimazione lineare

Questa tecnica di controllo, come le altre, utilizza un modello di errore del tipo

$$e = \begin{bmatrix} e_x \\ e_y \\ e_\theta \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^* - x \\ y^* - y \\ \theta^* - \theta \end{bmatrix}$$

Tale sistema tiene in considerazione quello che è l'errore nel sistema di riferimento del robot in movimento grazie alla matrice di rotazione presente nella formula.

La linearizzazione consiste nell'approssimare l'errore a zero.
La legge di controllo che regola la velocità lineare v e la velocità angolare w è la seguente

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

$$\begin{cases} K_1 &= 2\delta a \\ K_2 &= \dfrac{a^2 - (\omega^*)^2}{v^*} \\ K_3 &= 2\delta a \end{cases}$$

dove delta è compreso fra 0 e 1 e a è maggiore di zero.

In particolare, i parametri Ki agiscono da pesi per le componenti dell'errore.

Tale legge non assicura la convergenza teorica generale ma solo locale nel caso in cui sia v* che w* siano costanti.

Nel codice del progetto, viene data la possibilità di combinare una tecnica di path planning con una delle tre leggi di controllo. In questa relazione sarà presa in considerazione la tecnica dei potenziali artificiali per mostrare l'applicazione delle tre leggi di controllo.

Il codice della legge di controllo basato su approssimazione lineare riprende le formule riportate precedentemente:

```
function Xdot = control_lin(t,X,q, qd, qdd, a, delta)

% Situazione reale
x_sit = X(1);
y_sit = X(2);
theta_sit = X(3);

% Coordinate
x_star = q(1);
y_star = q(2);

% Velocità
xd_star = qd(1);
yd_star = qd(2);
theta_star = atan2(yd_star, xd_star);

% Accelerazione
xdd_star = qdd(1);
ydd_star = qdd(2);

% Legge di controllo
v_star = sqrt(xd_star^2 + yd_star^2) ;
if(v_star<0.001)
    v_star=0.001;
end
w_star = (ydd_star * xd_star - yd_star * xdd_star)/(v_star^2);

% Errore
ex = cos(theta_sit)*(x_star-x_sit) + sin(theta_sit)*(y_star-y_sit);
ey = -sin(theta_sit)*(x_star-x_sit) + cos(theta_sit)*(y_star-y_sit);
% Pesi
k1 = 2*delta*a;
k3 = k1;
k2 = (a^2-w_star^2)/v_star;

v = v_star * cos(etheta) + k1*ex;
w = w_star + k2*ey + k3*etheta;

Xdot = [v*cos(theta_sit);
        v*sin(theta_sit);
        w
        ];
end
```

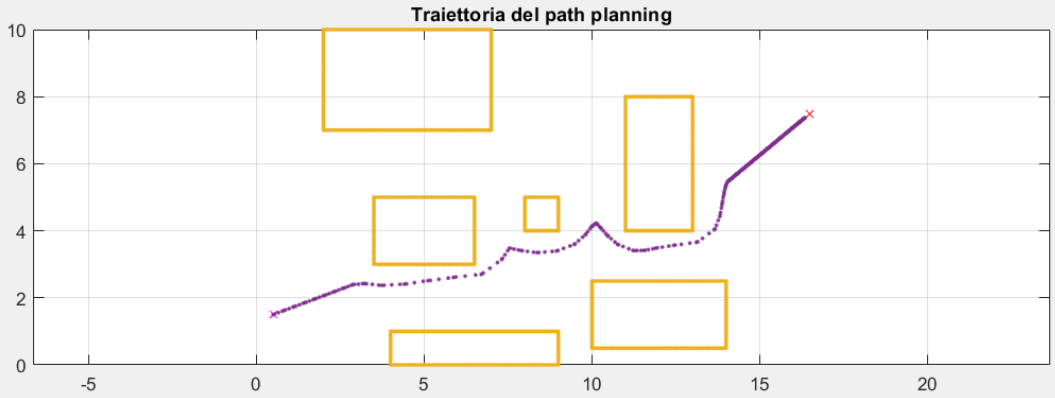
Essa tiene considerazione di alcune variabili q, qd e qdd calcolate attraverso la funzione traveltraj oltre a due parametri passati in input che vanno a influenzare la convergenza della curva ottenuta dalla legge di controllo a quella ottenuta dal path planning.

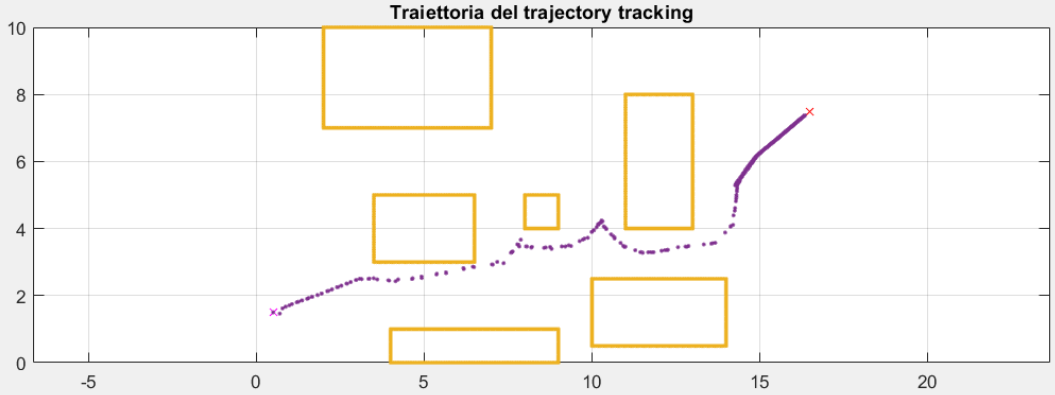
Per evitare di ottenere dei valori NaN (not a number) a causa di divisioni per zero, se la velocità v_star risulta essere estremamente piccola (minore di 0.001 nel caso del progetto), le viene assegnato un valore considerato come minimo.

Per ottenere il percorso descritto attarverso la legge di controllo, si invoca la funzione di integrazione ode45 per ogni punto della traiettoria, dalla quale si ottiene una variabile stato che contiene le coordinate del punto calcolato attraverso la legge di controllo. Tramite essa è anche possibile calcolare i valori degli errori ex e ey.

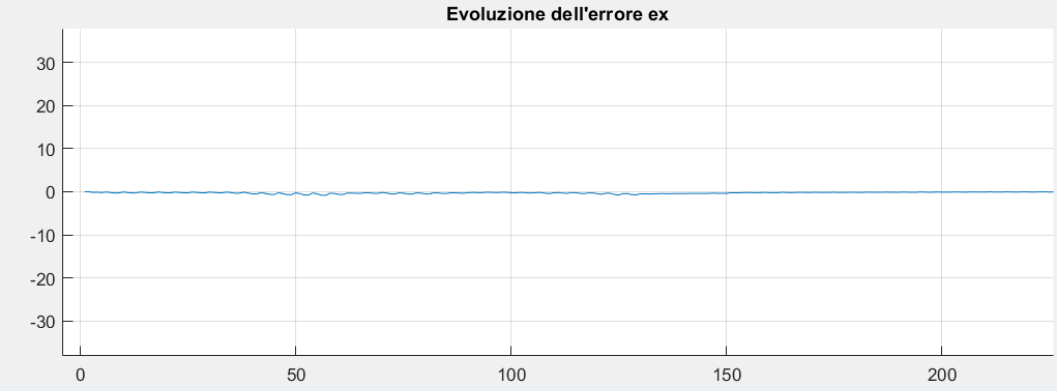
```
% CONTROLLO LINEARIZZATO
if controller == 1

    a = 0.56;
    delta = 0.99;

    ex = zeros(length(q)+1,1);
    ex(1) = q(1,1) - passo(1);
    ey = zeros(length(q)+1,1);
    ey(1) = q(2,1) - passo(2);
    etheta = zeros(length(q)+1,1);
    etheta(1) = angdiff(atan2(qd(2,1),qd(1,1)),passo(3));

    % Ciclo di integrazione
    for i=1:length(q)
        f_robot = @(t,X) control_lin(t,X,q(:,i), qd(:,i), qdd(:,i),a,delta) ;
        [t,stato] = ode45(f_robot,[i-1,i],[passo(i,1); passo(i,2); passo(i,3)]);
        passo(i+1,:) = stato(end,:);
        ex(i+1) = q(1,i) - stato(end,1);
        ey(i+1) = q(2,i) - stato(end,2);
        etheta(i+1) = angdiff(atan2(qd(2,i),qd(1,i)),stato(end,3));
    end
end
```

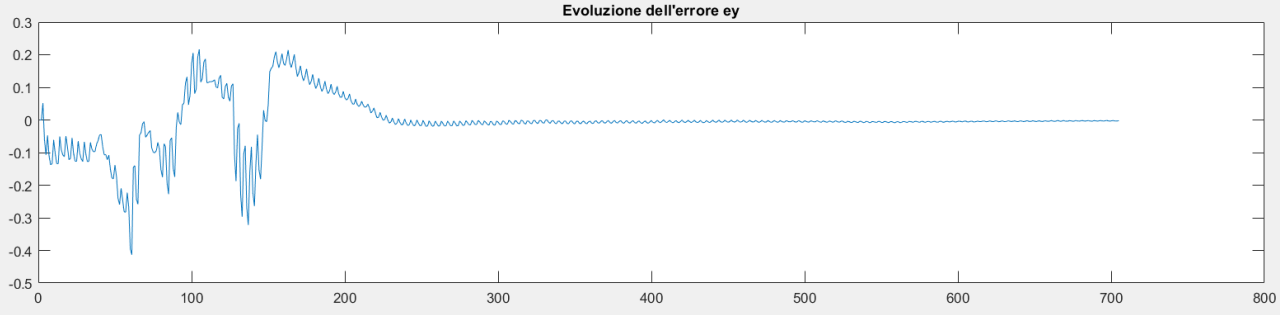
Dal seguente grafico è possibile visionare la differenza fra il risultato del path planning tramite potenziali artificiali e del trajectory tracking





Inoltre, l'errore ex e ey tendono a zero a più infinito come previsto dalla legge





Controllo basato su approssimazione non lineare

Questa tecnica di controllo presenta una condizione più forte di approssimazione globale teorica rispetto all'approssimazione lineare.

Il modello prevede un vincolo forte, il quale richiede che v* e w* non convergano contemporaneamente a zero. Ciò significa che le traiettorie devono essere persistenti ovvero il robot non deve avere momenti di stop e ripartenza; si può ovviare a tale problema rimuovendo il controllo nei momenti in cui il robot si ferma per poi riprenderlo subito dopo.

La legge di controllo sarà la seguente

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

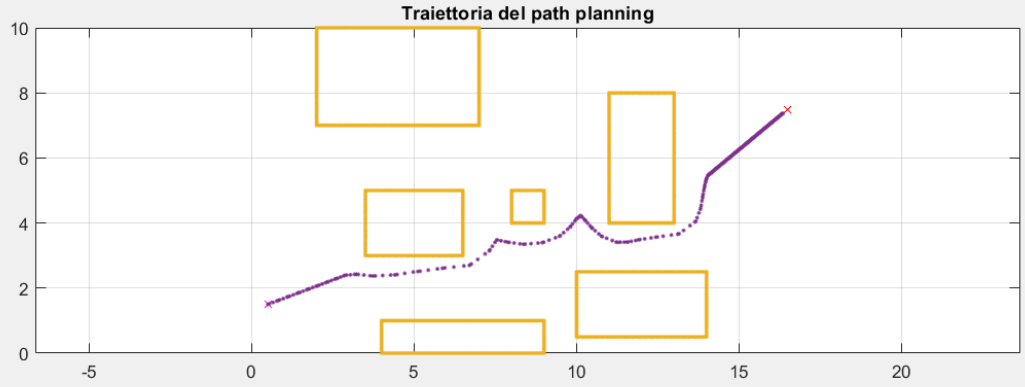
Funzione della legge di controllo:

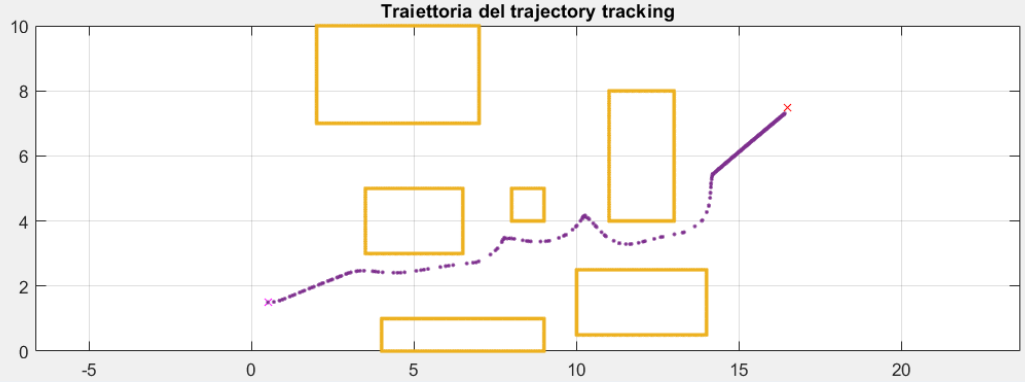
```
function Xdot = control_NL(t,X,q,qd,qdd,k1,k2,k3)

% Situazione reale
x_sit = X(1);
y_sit = X(2);
theta_sit = X(3);

% Coordinate
x_star = q(1);
y_star = q(2);

% Velocità
xd_star = qd(1);
yd_star = qd(2);
theta_star = atan2(yd_star, xd_star);

% Accelerazione
xdd_star = qdd(1);
ydd_star = qdd(2);

% Legge di controllo
v_star = sqrt(xd_star^2 + yd_star^2);
if(v_star<0.001)
    v_star=0.001;
end

w_star = (ydd_star * xd_star - yd_star * xdd_star)/(v_star^2);

% Errore
ex = cos(theta_sit)*(x_star-x_sit) + sin(theta_sit)*(y_star-y_sit);
ey = -sin(theta_sit)*(x_star-x_sit) + cos(theta_sit)*(y_star-y_sit);
etheta = angdiff(theta_star,theta_sit);

u1 = -k1(v_star, w_star)*ex;
u2 = -k2*v_star*(sin(etheta)/etheta)*ey - k3(v_star,w_star)*etheta;

v = v_star * cos(etheta) - u1;
w = w_star - u2;

Xdot = [v*cos(theta_sit);
        v*sin(theta_sit);
        w
        ];
end
```

Codice di integrazione:

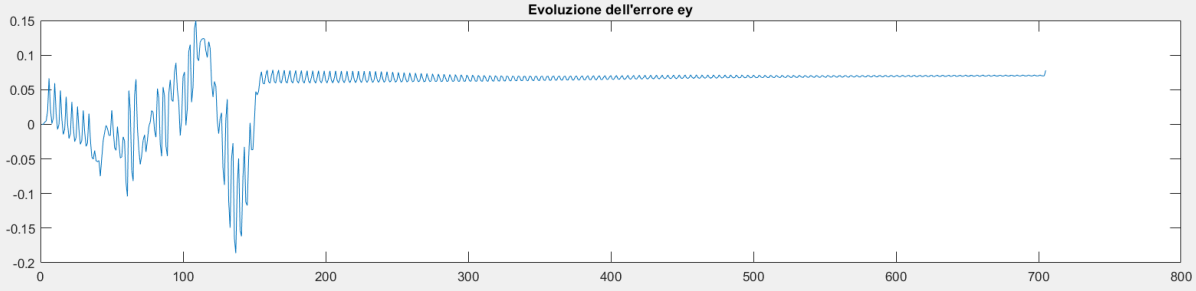
```
% CONTROLLO NON LINEARIZZATO
if controller == 2

    k1 = @(vs,ws)(sin(vs)+2);
    k2 = 1;
    k3 = @(vs,ws)(cos((vs)+2));

    ex = zeros(length(q)+1,1);
    ex(1) = q(1,1) - passo(1);
    ey = zeros(length(q)+1,1);
    ey(1) = q(2,1) - passo(2);
    etheta = zeros(length(q)+1,1);
    etheta(1) = angdiff(atan2(qd(2,1),qd(1,1)),passo(3));

    % Ciclo di integrazione
    for i=1:length(q)
        f_robot = @(t,X)(control_NL(t,X,q(:,i), qd(:,i), qdd(:,i),k1,k2,k3));
        [t,stato] = ode45(f_robot,[i-1,i],[passo(i,1); passo(i,2); passo(i,3)]);
        passo(i+1,:) = stato(end,:);
        ex(i+1) = q(1,i) - stato(end,1);
        ey(i+1) = q(2,i) - stato(end,2);
        etheta(i+1) = angdiff(atan2(qd(2,i),qd(1,i)),stato(end,3));
    end
end
```

Dal seguente grafico è possibile visionare la differenza fra il risultato del path planning tramite potenziali artificiali e del trajectory tracking





Inoltre, l'errore ex e ey tendono a zero come previsto dalla legge



Controllo basato su input-output linearization

Chiamata anche feedback linearization, utilizza delle leggi di controllo non lineari basati su input virtuali per linearizzare la relazione fra tali input virtuali e il sistema di output.

Il metodo fa uso di un ipotetico punto B posto ad una distanza b dal robot dal quale il robot viene "trascinato" lungo la traiettoria. Tale modalità non permette di essere certi sull'orientamento in quanto il robot potrebbe trovarsi orientato lungo una circonferenza di raggio b.

La particolarità del modello è che prescinde da tutte le condizioni presenti nelle precedenti tecniche di controllo come il vincolo anolonomo per il punto B e le velocità diverse da zero. Tuttavia, il robot dovrà rispettare il vincolo anolonomo in quanto è un limite strutturale.

La legge di controllo sarà la seguente

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

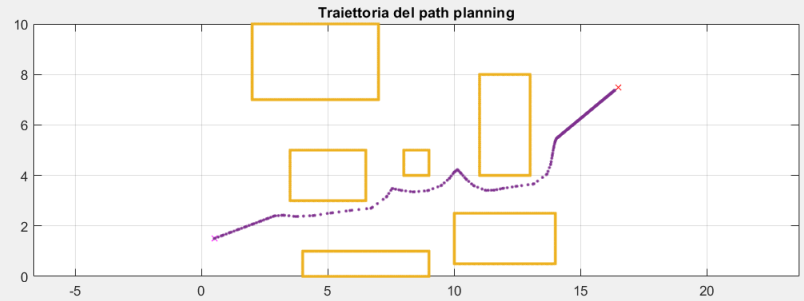
Codice della legge:

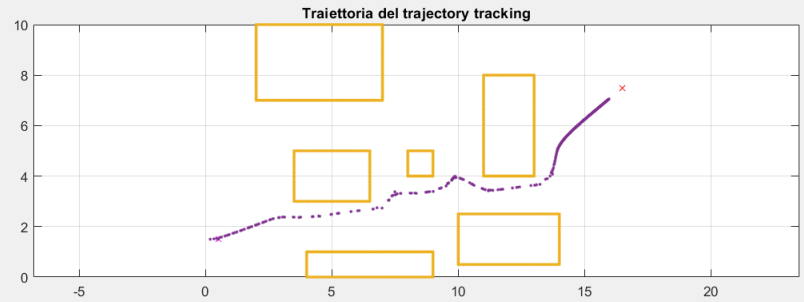
```
function Xdot = control_IO(t,X,q,qd, k1,k2,b)

% Situazione reale
x_sit = X(1);
y_sit = X(2);
theta_sit = X(3);

% Punto B

xB = x_sit+b*cos(theta_sit);
yB = y_sit+b*sin(theta_sit);

% Coordinate
x_star = q(1);
y_star = q(2);

% Velocità
xBd_star = qd(1);
yBd_star = qd(2);
Inv = [ cos(theta_sit) sin(theta_sit)
        -sin(theta_sit)/b cos(theta_sit)/b ];

% Legge di controllo
u1 = xBd_star + k1*(x_star-xB);
u2 = yBd_star + k2*(y_star-yB);
vw = Inv*[u1;u2];
v = vw(1);
w = vw(2);

Xdot = [v*cos(theta_sit);
        v*sin(theta_sit);
        w
        ];
end
```

Codice di integrazione:

```
% CONTROLLO INPUT OUTPUT LINEARIZATION
if controller == 3

    k1 = 1;
    k2 = 1;
    b = 0.5;

    ex = zeros(length(q)+1,1);
    ex(1) = q(1,1) - passo(1);
    ey = zeros(length(q)+1,1);
    ey(1) = q(2,1) - passo(2);
    etheta = zeros(length(q)+1,1);
    etheta(1) = angdiff(atan2(qd(2,1),qd(1,1)),passo(3));

    % Ciclo di integrazione
    for i=1:length(q)
        f_robot = @(t,X)(control_IO(t,X,q(:,i), qd(:,i),k1,k2,b));
        [t,stato] = ode45(f_robot,[i-1,i],[passo(i,1); passo(i,2); passo(i,3)]);
        passo(i+1,:) = stato(end,:);
        ex(i+1) = q(1,i) - stato(end,1);
        ey(i+1) = q(2,i) - stato(end,2);
        etheta(i+1) = angdiff(atan2(qd(2,i),qd(1,i)),stato(end,3));
    end

end
```

Differenze fra path planning e trajectory tracking





Errore ex e ey