Relazione progetto Robotica Mobile

Corso di Robotica

Anno Accademico 2020/2021

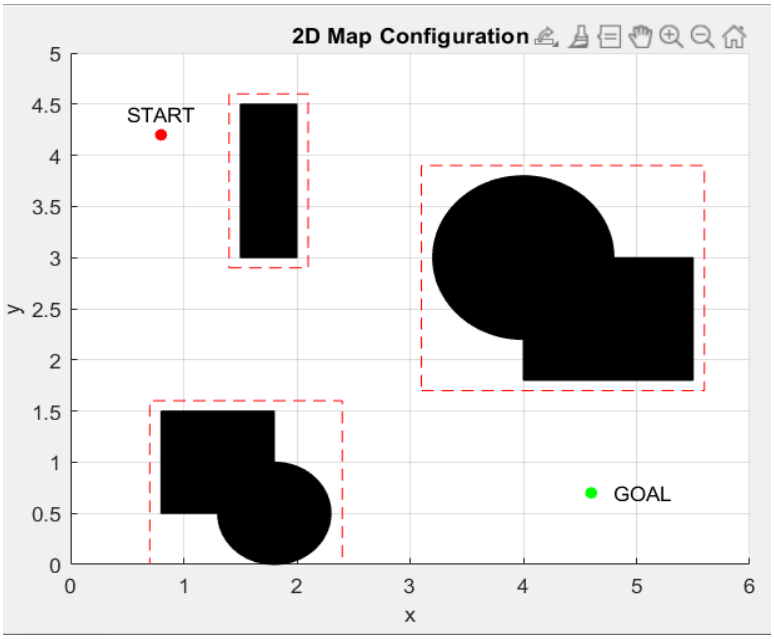
Alessandra Paone

MATRICOLA 200667

Introduzione

Questo progetto ha lo scopo di mostrare l'implementazione di alcune tecniche che permettono di pianificare il moto di un robot mobile di tipo "Differential Drive" da un punto di start (x_i,y_i) ad un punto di goal (x_f,y_f) all'interno di un ambiente noto con ostacoli. Data la traiettoria generata secondo una specifica tecnica, le leggi di controllo implementate permettono al robot di "inseguire" quest'ultima (problema di trajectory tracking), cioè riprodurre asintoticamente la traiettoria cartesiana desiderata a partire da una configurazione iniziale.

Configurazione dell'ambiente

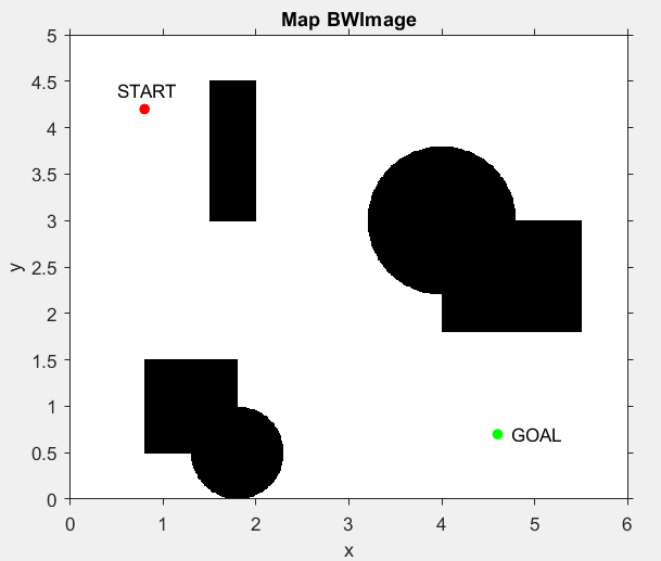


L'ambiente in cui il robot dovrà muoversi è una superficie piana di forma rettangolare avente una base di 6m e un'altezza di 5m con gli ostacoli posti come mostrato in figura. Per tutte le tecniche di progettazione della traiettoria (Potenziali Artificiali esclusi), gli ostacoli sono stati "ingrossati" di una quantità delta arbitraria, in quanto, nella pratica, il robot non è un punto materiale ma ha dimensioni fissate e non tutte le tecniche implementate permettono di progettare una traiettoria i cui punti siano sufficientemente distanti dagli ostacoli.

Pianificazione del moto

Artificial Potential Fields (APF)

Per l'implementazione di questa tecnica di progettazione si è scelto di utilizzare la mappa originale (con ostacoli non ingrossati), in quanto, per la natura dei potenziali stessi, la traiettoria progettata permetterà al robot di rimanere sempre lontano dagli ostacoli. Inoltre, in questo caso specifico, l'ambiente è rappresentato dalla matrice **obstacle**, dove $obstacle(i,j) = \begin{cases} 0 \; se \; la \; cella \; (i,j) \; \text{è} \; priva \; di \; ostacoli \\ 1 \; altrimenti \end{cases}$. Questa scelta permette di riferirsi alla mappa dell'ambiente come BWimage e di utilizzare la funzione di matlab **bwdist**, la quale restituisce in ogni istante la distanza del robot dai valori 1 (true) all'interno della matrice. Per quanto riguarda le espressioni che definiscono il potenziale attrattivo e i potenziali repulsivi, sono state scelte funzioni quadratiche in quanto facilmente derivabili: nel caso del potenziale attrattivo, la funzione quadratica è preferibile perché presenta un unico punto di minimo coincidente col punto di goal; nel caso dei potenziali repulsivi il massimo della funzione è raggiunto in prossimità degli ostacoli.



Funzione che definisce il potenziale attrattivo:

$$f_{attr} = \xi(\| (x - x_G)^2 + (y - y_G)^2 \|)$$

dove ξ è un parametro di scala arbitrario.

Funzione che definisce i potenziali repulsivi:

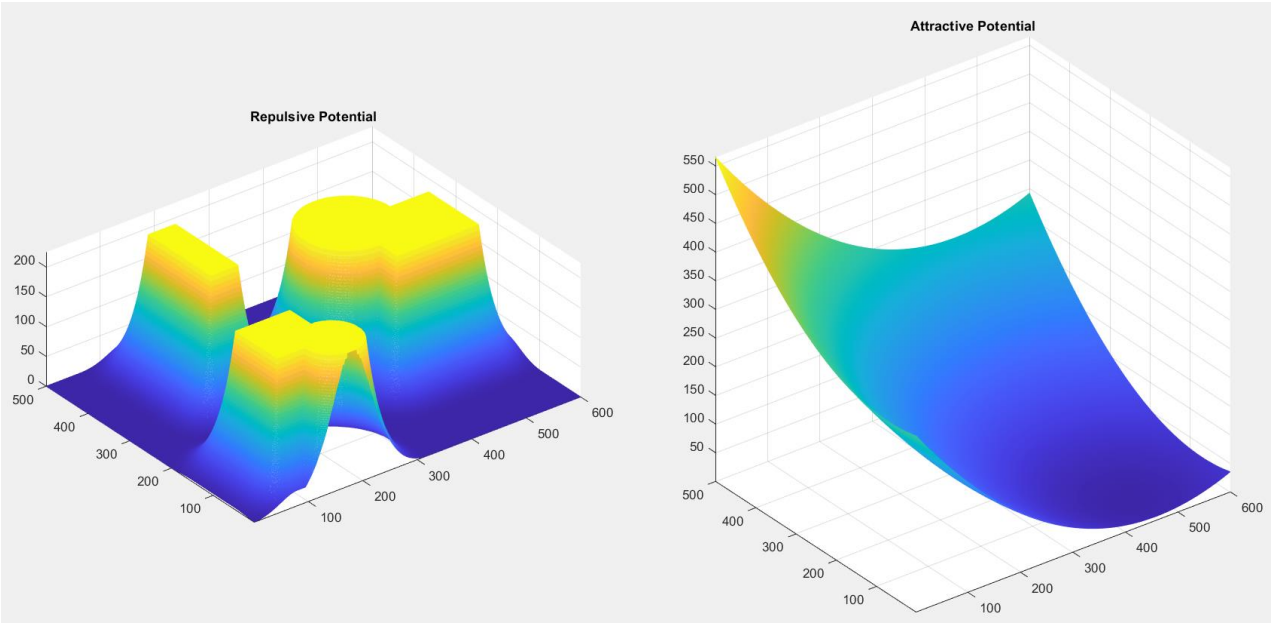
$$f_{rep} = \begin{cases} \eta \cdot (\frac{1}{\rho(x)} - \frac{1}{d_0})^2, & \rho(x) \leq d_0 \\ 0, & \rho(x) > d_0 \end{cases}$$

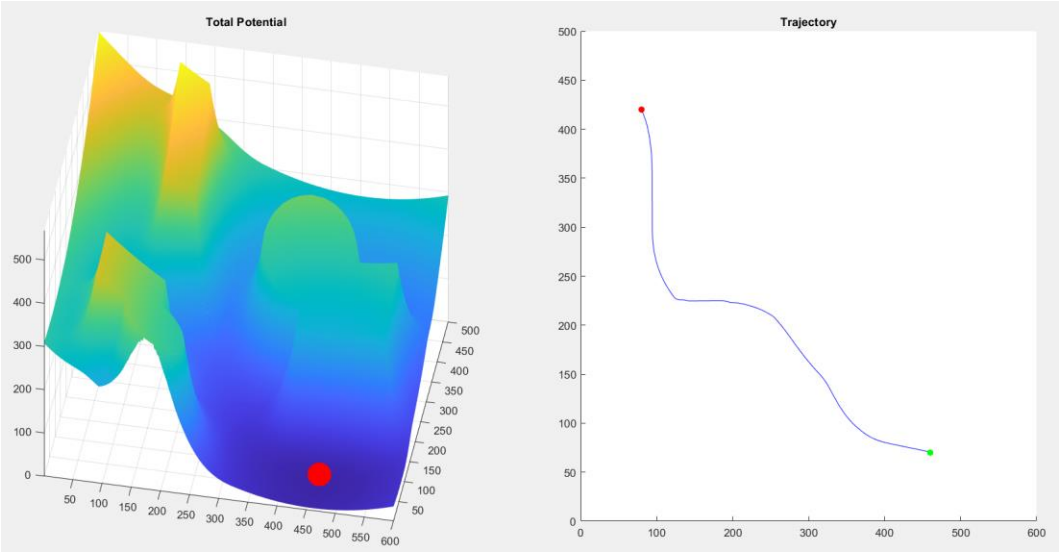
dove η è un parametro di scala arbitrario; $\rho(x)$ è la funzione che determina la distanza tra il robot e l'ostacolo più vicino; d_0 è un punto scelto arbitrariamente: dalla posizione dell'ostacolo fino a questo punto il robot risente della forza repulsiva.

Potenziale risultante: $f_{TOT} = f_{attr}(x, y) + f_{rep}(x, y)$

Di seguito è presentato l'algoritmo di progettazione della traiettoria orientata nella direzione dell'antigradiente $-\nabla f_{TOT}(x, y)$:

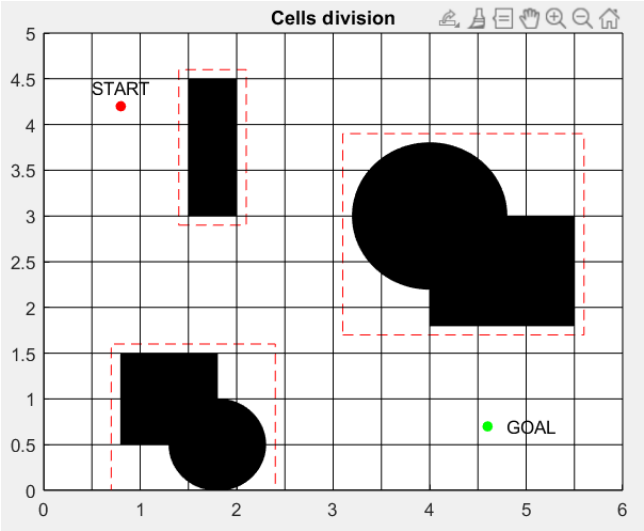
```
in ogni punto P(x,y):
x_new = x + ||−∇f(x)||;
y_new = y + ||−∇f(y)||;
if |x_new-x_goal|<tolerance && |y_new-y_goal|<tolerance  → destinazione
raggiunta
else
path.add(x_new,y_new); x = x_new; y = y_new;
```





Artificial Potential Fields Discretized (APFD)

Per questa tecnica, l'ambiente è stato rappresentato come una matrice 10x12 inizialmente costituita da zeri (ogni cella corrisponde a un quadrato di lato $\delta \times \delta$ nella mappa, dove $\delta = 0.5m$).

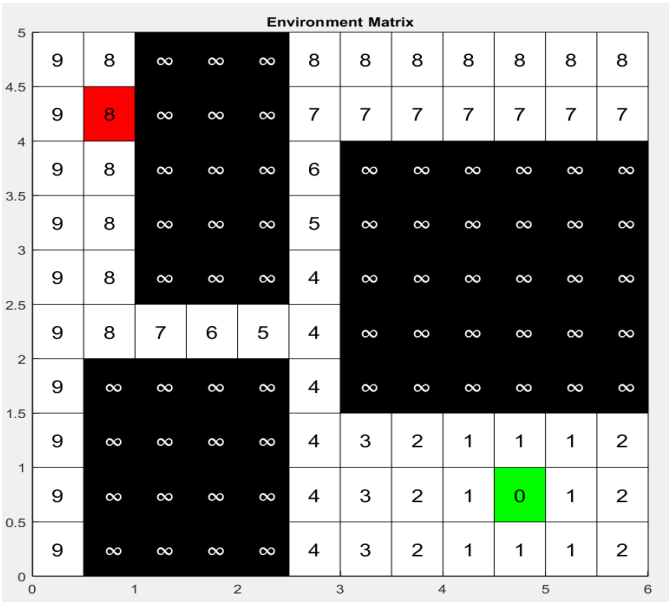


Un punto generico di coordinate (x,y) appartiene ad una cella della matrice i cui indici (i, j) vengono calcolati con le seguenti formule:

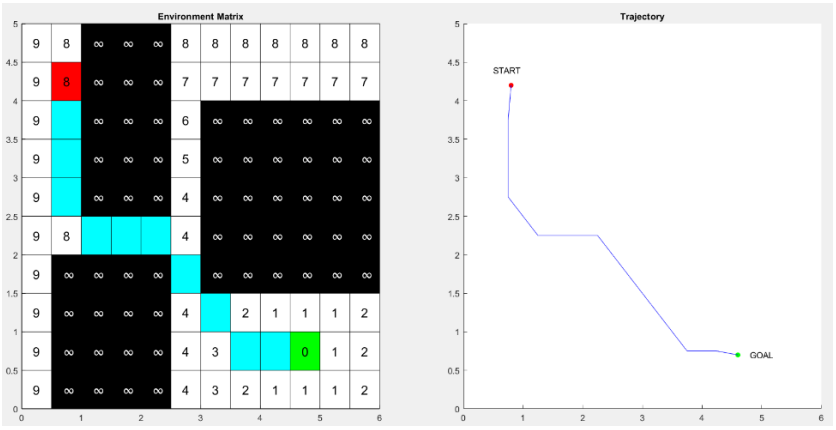
$$\boldsymbol{i = n + 1 - iplot}, \text{dove } \boldsymbol{iplot = \lceil y/\delta \rceil}$$

$$\boldsymbol{j = \lceil x/\delta \rceil}$$

I vicini (vicinato ad 8) di una cella (i, j) assumono un valore pari al valore della cella incrementato di uno; il primo valore, pari a zero, viene assegnato alla cella del goal ed in seguito viene applicata la regola descritta. Agli ostacoli viene assegnato il valore $n \cdot m + 1$, che in questo contesto può essere assunto come "infinito" poiché il valore massimo che si può inserire nella matrice, partendo da zero, è $n \cdot m - 1$. La creazione del vicinato e l'assegnamento dei valori è gestito mediante l'utilizzo di una coda FIFO dove il primo elemento inserito è il goal. Una volta estratto un elemento, si inseriscono nella coda i vicini di quest'ultimo e, se possibile (ovvero se il loro valore è uguale a zero), si assegna loro il valore stabilito e si aggiorna la lunghezza della coda. Una volta che ad ogni cella (i, j) della matrice d'ambiente è associato un valore, occorre individuare le celle in cui è presente un ostacolo o parte di esso e associare loro il valore $n \cdot m + 1$ (nel plot questo valore è indicato invece come ∞). Per la ricerca del path minimo, si è scelto di utilizzare un algoritmo ricorsivo basato su backtracking al fine di riuscire a trovare sempre una soluzione, se questa esiste. Si può considerare tale algoritmo come una ricerca su albero: a partire dalla cella di start, si considerano solo i vicini che

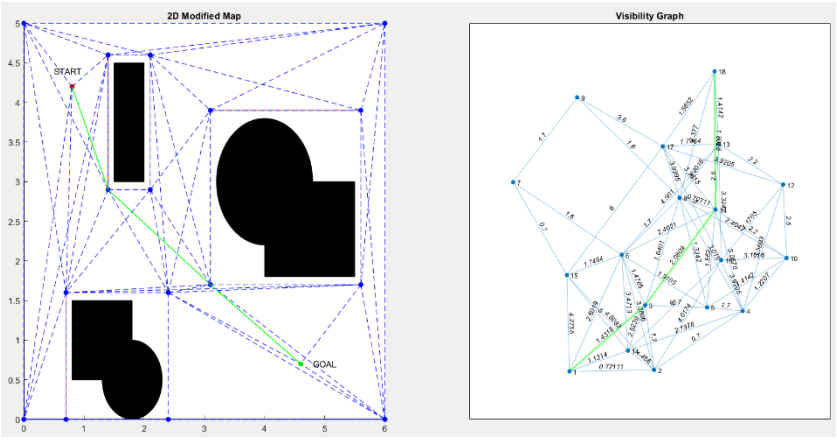



```
env_matrix =

  Columns 1 through 7

     9     8   121   121   121     8     8
     9     8   121   121   121     7     7
     9     8   121   121   121     6   121
     9     8   121   121   121     5   121
     9     8   121   121   121     4   121
     9     8     7     6     5     4   121
     9   121   121   121   121     4   121
     9   121   121   121   121     4     3
     9   121   121   121   121     4     3
     9   121   121   121   121     4     3
     0   121   121   121   121     0     0

  Columns 8 through 12

     8     8     8     8     8
     7     7     7     7     7
   121   121   121   121   121
   121   121   121   121   121
   121   121   121   121   121
   121   121   121   121   121
   121   121   121   121   121
     2     1     1     1     2
     2     1     0     1     2
     2     1     1     1     2
     0     0     0     0     0
```



hanno valore minore o uguale a quello della cella corrente che non sono già stati visitati; in particolare, la precedenza di visita è data ai vicini che hanno valore strettamente minore di quello della cella corrente e, a parità di valore, si definisce un ulteriore ordinamento basato sulla distanza euclidea dal punto di goal. È possibile notare che, come ci si aspetta, la traiettoria ottenuta rappresenta una versione approssimata di quella generata con la tecnica APF.

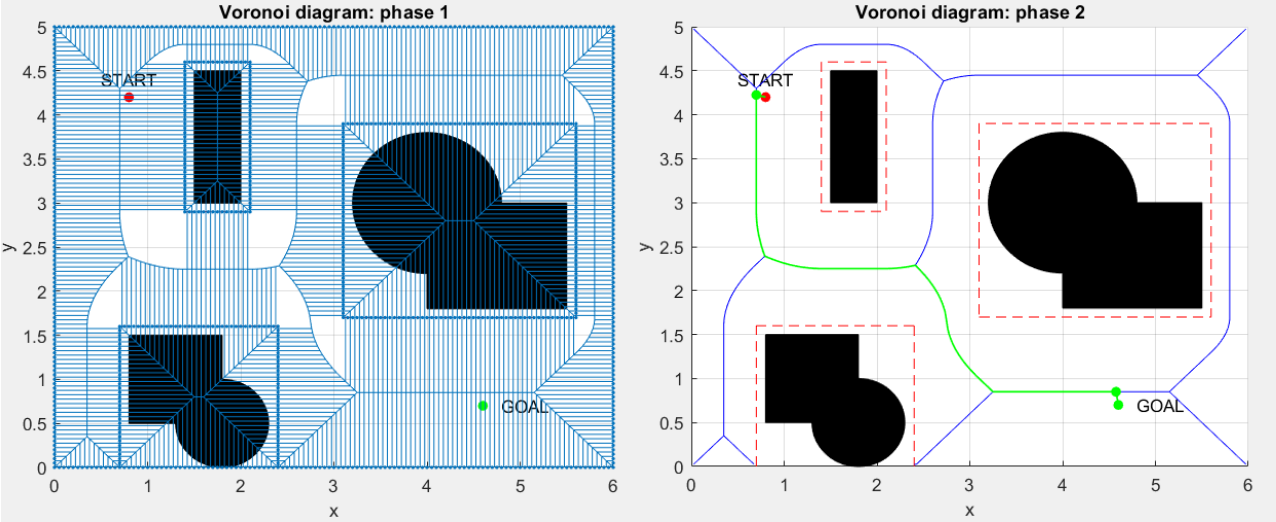
Grafo di visibilità



Al fine di implementare questa tecnica, è stato necessario effettuare delle operazioni preliminari quali l'individuazione dei vertici degli ostacoli ingrossati, dei bounds della mappa e la costruzione di arrays ausiliari per la definizione degli archi del grafo $G =< V, E >$. L'insieme V dei nodi è costituito dai vertici di tutte le figure, bounds della mappa compresi, e dai punti di start e di goal. I primi archi che vengono inseriti nell'insieme E sono quelli che uniscono ciascun vertice di un ostacolo con tutti gli altri, escludendo gli archi che attraverserebbero l'ostacolo stesso; questi coincidono con i lati della figura. Per tutti gli altri archi definiti dal nodo i al nodo j si verifica la realizzabilità: se l'arco $e = (i, j)$ interseca almeno

uno dei quattro lati di almeno un ostacolo o se il punto di start/goal giace su di esso, allora $e \notin E$; viceversa, l'arco si inserisce in E con peso pari alla distanza euclidea dal nodo i al nodo j.

Diagramma di Voronoi



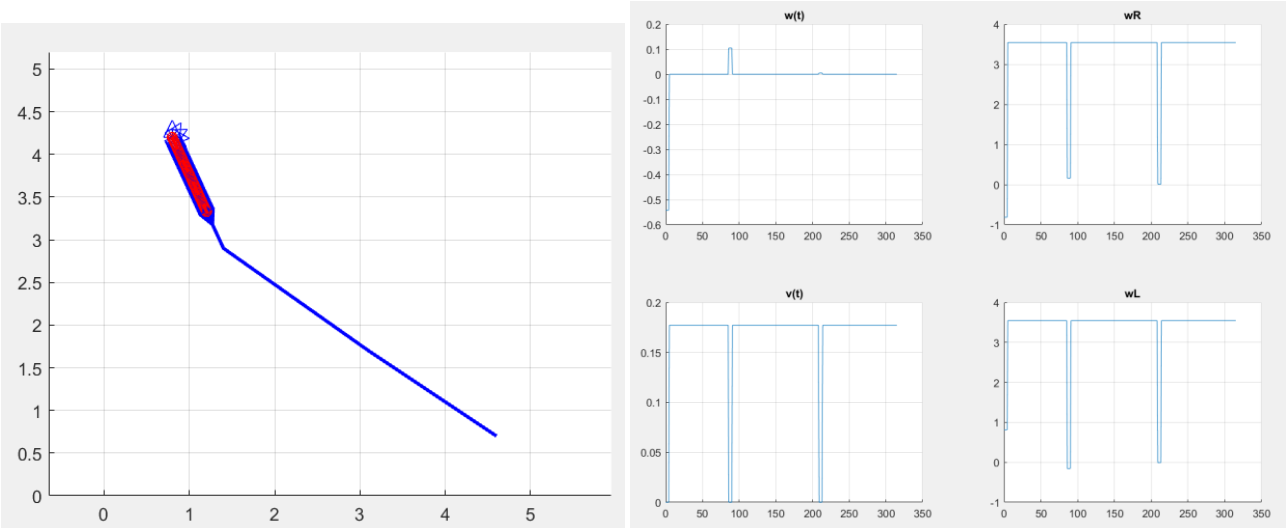
La prima fase dell'implementazione di questa tecnica prevede la discretizzazione dei bounds degli ostacoli e della mappa stessa al fine di generare il diagramma di Voronoi utilizzando la funzione matlab omonima (Voronoi diagram: phase 1). Dopo aver "ripulito" il diagramma dalle linee che intersecano gli ostacoli e i bounds della mappa, si determinano le ritrazioni dei punti di start e di goal sul diagramma di Voronoi: se i punti di start/goal non appartengono alla frontiera del diagramma, questi vengono collegati da un arco al punto più vicino in senso euclideo appartenente alla frontiera; se, invece, i punti di start/goal si trovano già sulla frontiera, questi vengono collegati da un arco ai loro due vicini più prossimi (in questo modo durante il calcolo dello shortest path si hanno due scelte sulla direzione da prendere). La fase finale prevede la costruzione del grafo associato al diagramma di Voronoi, il quale viene utilizzato per calcolare lo shortest path.

Moto punto-punto

Il moto punto-punto del Differential Drive, definito esclusivamente per traiettorie costituite da sole spezzate, è stato implementato aggiornando di tratto in tratto lo stato del robot secondo il seguente algoritmo:

```
data la configurazione iniziale P0(xs,ys) e orientamento θ0=π/2, finché non
arrivo a destinazione:
-definisco il nuovo orientamento θ01 = atan2(y1-y0,x1-x0);
-ruoto di un angolo pari alla differenza tra θ0 e θ01 con velocità angolare ω
arbitraria;
-traslo fino al prossimo punto.
```

Inoltre, fissato un tempo di simulazione massimo Tmax, ogni tratto della traiettoria è percorso a velocità costante poiché il tempo di percorrenza di un tratto, per come è stato definito, dipende dalla lunghezza di quest'ultimo. Di seguito è presente un esempio grafico del moto punto-punto sulla traiettoria generata mediante grafo di visibilità.

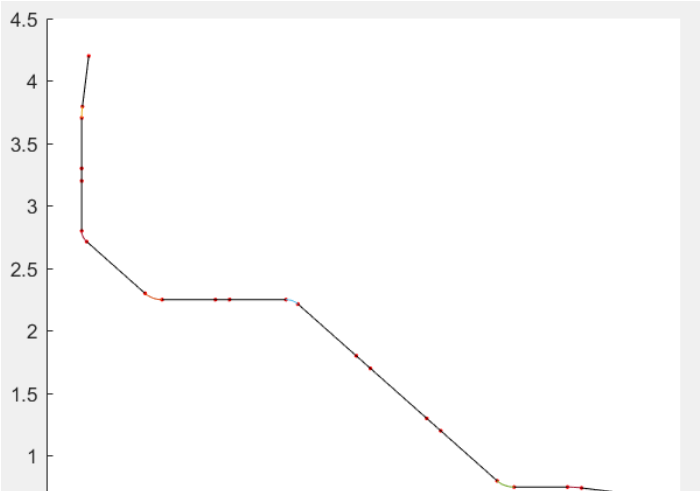


Leggi di controllo per trajectory tracking

Prima di affrontare l'implementazione delle leggi di controllo del robot, si è provveduto a rendere "smooth" le traiettorie ottenute mediante i vari metodi di progettazione (APF escluso in quanto la traiettoria restituita è già smooth). A questo proposito, è stata implementata la funzione matlab *smooth_trj* che, dati n punti planari che definiscono una traiettoria costituita da spezzate, restituisce dei nuovi punti e informazioni utili alla definizione di una nuova traiettoria che, in prossimità degli angoli, assume un andamento curvilineo. In particolare, l'idea alla base di questa funzione è descritta dai seguenti step:

1) si individuano i punti di raccordo A' e B' sulle segmenti i ed $i+1$ con $i = 0,1,\dots,N-1$ da cui inizia e finisce la fase di curvatura. Il punto di raccordo A' giace sul segmento corrente ad una distanza scelta come frazione arbitraria della lunghezza del segmento, mentre il punto B' è scelto in modo che la distanza tra A' e il waypoint sia uguale in valore assoluto alla distanza tra B' e il waypoint;

2) si individuano le rette passanti per i punti A' e B' e perpendicolari ai segmenti di retta su cui giacciono tali punti: il loro punto di intersezione individua il centro della circonferenza che si deve percorrere per passare dal punto A' al punto B', il cui raggio è pari alla distanza euclidea tra il centro ed uno fra i punti A' e B'.



Versione smooth della traiettoria generata mediante la tecnica APFD.

Un ulteriore passaggio propedeutico alla definizione delle leggi di controllo consiste nella costruzione delle funzioni del tempo x(t) e y(t) a partire dalle coordinate dei punti appartenenti alle traiettorie progettate. Date le coordinate x e y dei punti della traiettoria, le funzioni matlab *trajToTimeFunc_x* e *trajToTimeFunc_y* forniscono in output i valori delle funzioni del tempo x(t) e y(t) o i valori delle loro derivate prime o seconde mediante interpolazione lineare dei dati in input su un intervallo temporale definito come descritto nel moto punto-punto: calcolate le lunghezze dei singoli tratti che compongono la traiettoria, il singolo tratto (lineare o curvilineo) è percorso in un tempo

tale per cui la velocità risulti sempre costante. Ricordando che gli errori su x,y e θ sono definiti come $\begin{cases} e_x = x_{star} - x \\ e_y = y_{star} - y \\ e_\theta = atan2(\sin(\theta_{star} - \theta), \cos(\theta_{star} - \theta)) \end{cases}$, $v_{star} = \sqrt{{xd_{star}}^2 + {yd_{star}}^2}$ e $\omega_{star} = \frac{ydd_{star} \cdot xd_{star} - yd_{star} \cdot xdd_{star}}{{v_{star}}^2}$ (la dipendenza dal tempo è omessa per semplicità di lettura), le leggi di controllo sono definite come di seguito.

Linear Control (Lin)

Definiti i parametri a = 1 e δ=0.99 (generalmente a>0 e 0<δ<1), si ha che k1 = 2 δ a, $k2 = \frac{a^2 - {\omega_{star}}^2}{v_{star}}$ e k3=k1, allora $v = v_{star} - \cos(e_\theta) - k1 \cdot e_x$, $\omega = \omega_{star} + k2 \cdot e_y + k3 \cdot e_\theta$.

Non-linear Control (NL)

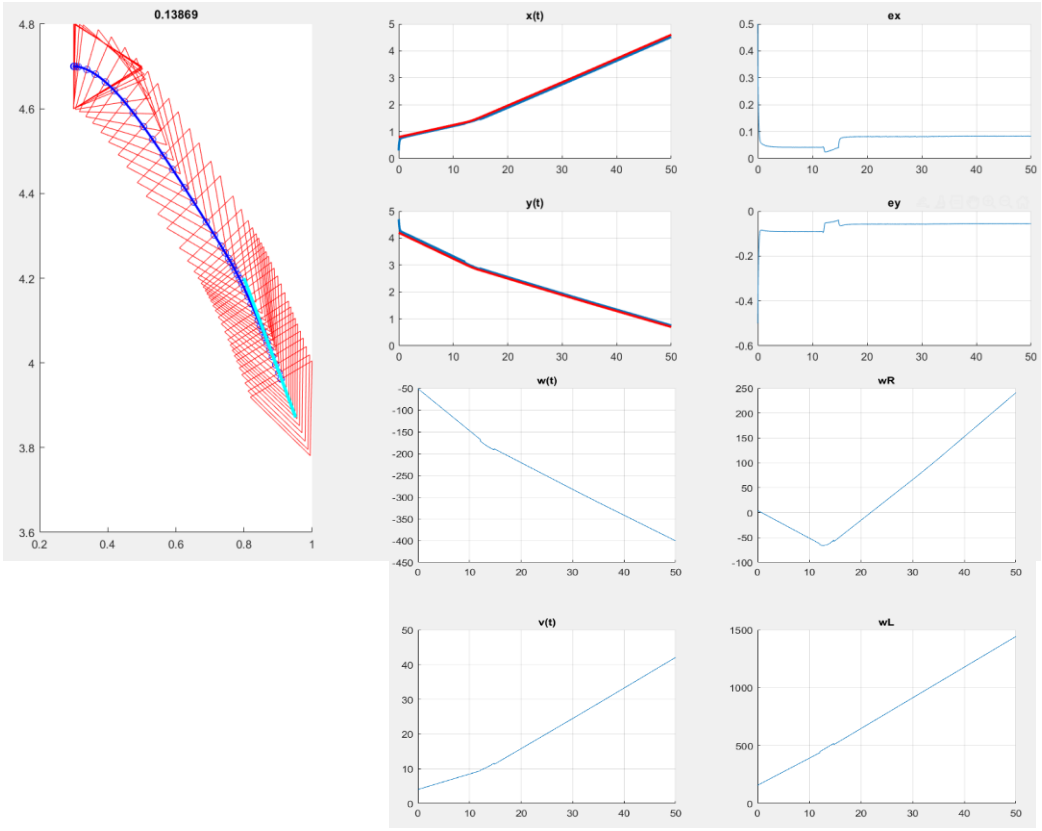
Definite k1(v_{star}, ω_{star}), k3(v_{star}, ω_{star}) come funzioni costanti (e quindi limitate con derivate limitate), k2=1 (gen. k2>0) e gli ingressi u1 = $-$k1(v_{star}, ω_{star}) $\cdot e_x$, u2 =$-$k2 $\cdot v_{star} \cdot \frac{\sin(e_\theta)}{e_\theta} \cdot e_y \cdot$ $-$k3(v_{star}, ω_{star}) $\cdot e_\theta$, allora $v = v_{star} \cdot \cos(e_\theta) - u1$, $\omega = \omega_{star} - u2$.

Feedback Linearization Control (FL)

Definiti k1=10, k2=10, b=0.1 e gli ingressi u1 = $xd_{star} + k1 \cdot (x_{star} - x_B)$, u2 = $yd_{star} + k2 \cdot (y_{star} - y_B)$ dove x_B= x+bcos(θ) e $y_B = y + bsin(\theta)$, allora $\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) \\ \frac{-\sin(\theta)}{b} & \frac{cos(\theta)}{b} \end{bmatrix} \begin{bmatrix} u1 \\ u2 \end{bmatrix}$.

L'evoluzione dello stato del robot sul range temporale [0,Tmax] è definita dalla routine matlab *ode45* che, data in ingresso la matrice Xdot = $\begin{bmatrix} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ \omega \end{bmatrix}$, integra il sistema di equazioni differenziali del tipo $y' = f(x, y)$ da 0 a Tmax tenendo conto delle condizioni iniziali (nel pratico, ode45 risolve un problema di Cauchy). Di seguito, un esempio di controllo del moto del differential drive secondo le leggi NL ed FL con traiettoria generata mediante grafo di visibilità dove $e_x = -0.5$, $e_y = 0.5$, e_θ = 0.5 (nel caso di FL control θ_0=0). Per tutte le configurazioni delle traiettorie progettate secondo le diverse tecniche non è garantita la stabilità secondo la legge di controllo lineare in quanto v e ω non sono costanti nel tempo.

FL control



NL control – accade che v e ω vadano contemporaneamente a zero, quindi non è garantita la stabilità globale