Università degli Studi di Verona
Facoltà di Scienze e Tecnologie Informatiche

Corso di Laurea Triennale in Informatica

Tesi di Laurea Triennale

Algoritmi di pianificazione dinamica basati su Velocity Obstacle - VO

algorithms for robot collision avoidance

Candidato:
Zorzi Davide
Matricola VR372284

Relatore:
Paolo Fiorini

Anno Accademico 2015–2016

Practise makes perfect.

Indice

Elenco delle figure

Abstract

Presento alcuni algoritmi di pianificazione dinamica basati su Velocity Obstacle per multiple mobile robot e/o virtual agents. Ogni robot é indipendente uno dall'altro senza coordinate centrali e senza comunicare con gli altri agenti. Ogni algoritmo prevede la conoscenza della posizione e velocitá corrente di ogni agente per computare la loro futura traiettoria.
Infine, presenteró l'implementazione della simulazione dell' algoritmo basato su Velocity Obstacle implementato in Matlab, che chiameremo Detect Velocity Obstacle-DVO.

Definizione del problema

Noi consideriamo che ogni robot e ostacolo statico o dinamico nell'ambiente sia a disc-shape. Per ogni robot $\boldsymbol{A}$ assumo avere un raggio fissato $\boldsymbol{r_A}$, una posizione corrente $\boldsymbol{p_A}$, e una velocitá corrente $\boldsymbol{v_A}$, inoltre di ciascuno sono note queste specifiche che possono essere condivise dagli altri robot nell'ambiente. Ogni robot possiede una posizione di arrivo (goal) denotata $\boldsymbol{p^{goal}_A}$ ed una velocitá preferita $\boldsymbol{v^{pref}_A}$, queste non sono conosciute agli altri robot.
Il goal é semplicemente un punto fissato nel piano. La velocitá preferita é la velocitá calcolata tra la posizione corrente e il goal, senza considerare altre variabili in gioco, ed é chiamata anche velocitá *ideale*.
L'obiettivo di ogni robot é scegliere, indipendentemente e simultaneamente, una nuova velocità ad ogni passo di computazione, che permetta di eseguire una traiettoria verso il suo obiettivo senza causare collisioni con tutti gli altri robot o ostacoli, cercando di avere il minor numero di oscillazioni possibili.

Ringraziamenti

Ringrazio tutti coloro che mi hanno sostenuto sia in caso di vittora ma sopprattutto in caso di fallimento. Un ringraziamento speciale alla mia famiglia ed al mio gruppo studi.

Verona, Novembre 2016 D. Z.

Introduzione

In questo documento presentó alcuni algoritmi di pianificazione dinamica basati su Velocity Obstacle.

Il primo capitolo offre una visione d'insieme sul problema di Collision Avoidance e sul concetto di Collision Cone e Velocity Obstacle - VO.

Il secondo capitolo affronta il problema delle oscillazioni causate da Velocity Obstacle incorporato dalla natura reattiva degli altri robot - RVO.

Il terzo capitolo affronta il problema delle oscillazioni causate da Reciprocal Velocity Obstacle dimezzando la responsabilitá della computazione della nuova velocitá - HRVO.

Il quarto capitolo offre una spiegazione sul migliore algoritmo odierno di Collision Avoidance basato su Velocity Obstacle utilizzando le velocitá relative - ORCA.

Il quinto capitolo descrive l'implementazione della simulazione in Matlab utilizzando alcuni principi descritti precedentemente - DVO.

Capitolo 1

Velocity Obstacle

Velocity Obstacle é un algoritmo di local Collision Avoidance di navigazione tra altri robot e ostacoli statici o dinamici nello stesso ambiente.
In due dimensione é definito come segue.

1.1 Velocity Obstacle

$\boldsymbol{A}$ rappresenta un robot e $\boldsymbol{B}$ rappresenta un ostacolo dinamico (altro robot) nel piano, $\boldsymbol{p_A}$ e $\boldsymbol{p_B}$ rappresentano le posizioni correnti di $\boldsymbol{A}$ e di $\boldsymbol{B}$, rispettivamente, $\boldsymbol{v_A}$ e $\boldsymbol{v_B}$ rappresentano le loro velocitá correnti.

1.1.1 Collision Cone

Noi definiamo *Collision Cone*, $CC_{\mathrm{A,B}}$, l'insieme delle *colliding relative velocities* tra $\boldsymbol{A'}$ e $\boldsymbol{B'}$:

$$CC_{A,B} = \{\boldsymbol{v}_{A,B} | \lambda_{A,B} \cap \boldsymbol{B'} \neq \oslash\} \tag{1.1}$$

dove $\boldsymbol{v_{A,B}}$ rappresenta la velocitá relativa di $\boldsymbol{A'}$ rispetto $\boldsymbol{B'}$, $\boldsymbol{v_{A,B}} = \boldsymbol{v_A}$ - $\boldsymbol{v_B}$, e $\lambda_{\mathrm{A,B}}$ é la linea di $\boldsymbol{v_{A,B}}$.
Questo cono é la parte di piano delimitata da tue tangenti, λ_{f} e λ_{r}, con apice in $\boldsymbol{A'}$. Ogni velocitá relativa all'interno delle due tangenti rappresenta una collisione tra i due robot. Per considerare multipli ostacoli, é utile stabilire delle condizioni equivalenti sulle velocitá assolute di $\boldsymbol{A}$. Questo é fatto semplicemente sommando la velocitá di $\boldsymbol{B}$, $\boldsymbol{v_B}$, per ogni velocitá nel $CC_{\mathrm{A,B}}$ o, equivalentemente, traslando il cono delle collisioni $CC_{\mathrm{A,B}}$ di $\boldsymbol{v_B}$. Definendo *Velocity Obstacle VO* come:

$$VO = CC_{A,B} \oplus \boldsymbol{v}_B \tag{1.2}$$

dove $\oplus$ é l' operatore somma vettoriale di Minkowski.

1.1.2 Time horizon

Siccome VO é basato su una approssimazione lineare della traiettoria degli ostacoli, usando questo per predirre delle collisioni remote, potrebbe essere poco accurato se gli ostacoli non si muovono in linea retta.

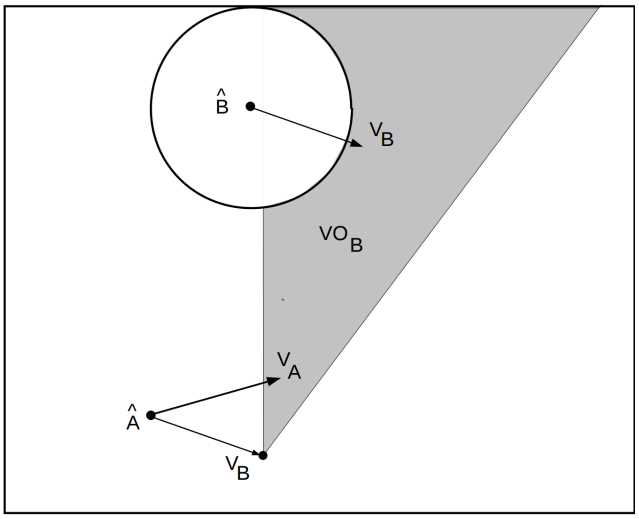


Figura 1.1: Velocity Obstacle traslato della velocitá di B

Noi definiamo *collisione imminente*, tra robot e ostacolo, se succederá in un tempo $t<T_{\mathrm{h}}$, dove T_{h} é un determinato spazio temporale chiamato *time horizon*, selezionato in base al sistema dinamico, alla traiettoria dell'ostacolo e dalla computazione delle manovre per schivare l'ostacolo.
Avendo individuato una collisione imminete, noi modifichiamo l'insieme VO sottraedo da esso l'insieme del VO_{h} definito come segue:

$$VO_h = \{\boldsymbol{v}_A | \boldsymbol{v}_A \in VO, norm(\boldsymbol{v}_{A,B}) \leq \tfrac{d_m}{T_h}\} \tag{1.3}$$

dove d_{m} é la piú piccola distanza tra il robot e l'ostacolo. L'insieme VO_{h} rappresenta le velocitá che risultano essere in collisione se prese al di sopra del time horizon.

1.1.3 New velocity

La computazione della nuova velocitá, in ogni time-step, deve essere selezionata fuori dal VO. Sfortunatamente, il concetto di Velocity Obstacle porta alla creazione di traiettorie oscillatorie sgradevoli.
Piú precisamente, se i robot A e B si stanno muovendo, rispettivamente, con $\boldsymbol{v_A}$ e $\boldsymbol{v_B}$, si avrá che $\boldsymbol{v_A} \in VO_{\mathrm{A,B}}(\boldsymbol{v_B})$ e $\boldsymbol{v_B} \in VO_{\mathrm{B,A}}(\boldsymbol{v_A})$. Quindi, lungo le stesse velocitá correnti, saranno in collisione. Perció l'agente A deciderá di modificare la sua velocitá con $\boldsymbol{v'_A}$, tale che sia fuori dal Velocity Obstacle di B. Allo stesso tempo, B modificherá la sua velocitá $\boldsymbol{v'_B}$, che dovrá essere scelta fuori dal Velocity Obstacle di A.
Quindi, in questa nuova situazione, le vecchie velocitá $\boldsymbol{v_A}$ e $\boldsymbol{v_B}$ saranno fuori dal Velocity Obslacle di B e A, rispettivamente $\boldsymbol{v_A} \notin VO_{\mathrm{A,B}}(\boldsymbol{v'_B})$ e $\boldsymbol{v_B} \notin VO_{\mathrm{B,A}}(\boldsymbol{v'_A})$. Se entrambi gli agenti preferiscono le vecchie velocitá, per esempio perché li porta direttamente al proprio *goal*, le sceglieranno ancora. Nel ciclo successivo, queste velocitá si tradurranno in una collisione e loro (A e B) probabilmente sceglieranno di nuovo $\boldsymbol{v'_A}$ e $\boldsymbol{v'_B}$, e cosí via.
Perció gli agenti oscillano tra queste due velocitá, creando una traiettoria oscillatoria.

Capitolo 2

Reciprocal Velocity Obstacle

Reciprocal Velocity Obstacle affronta il problema dell'oscillazione causato dal Velocity Obstacle incorporando la natura reattiva degli altri robot.

2.1 Reciprocal Velocity Obstacle

Per affrontare il problema delle traiettorie oscillatorie, invece di dover prendere tutte le responsabilitá, RVO lascia prendere al robot solo la metá della responsabilitá, per evitare le collisioni, pur assumendo che l'altro robot coinvolto ricambi prendendosi cura dell'altra metá.
L'idea base sarebbe di scegliere una nuova velocitá per ogni agente all'esterno degli altri Velocity Obstacle e che, questa nuova velocitá, sia la media (*average*) della velocitá corrente e di una velocitá che sia al di fuori degli Velocity Obstacle degli altri agenti.

2.1.1 Definition

Questo principio é formalizzato come segue:

$$RVO_{A,B}(\boldsymbol{v}_B, \boldsymbol{v}_A) = \{\boldsymbol{v'}_A | 2\boldsymbol{v'}_A - \boldsymbol{v}_A \in VO_{A,B}(\boldsymbol{v}_B)\} \tag{2.1}$$

dove $\mathrm{RVO_{A,B}}(\boldsymbol{v_B},\boldsymbol{v_A})$ dell'agente B su l'agente A contiene tutte le velocitá per A che saranno la media della velocitá corrente $\boldsymbol{v_A}$ e una velocitá all'interno di $\mathrm{VO_{A,B}}(\boldsymbol{v_B})$ dell'agente B. Puó essere interpretato geometricamente come Velocity Obstacle, $\mathrm{VO_{A,B}}(\boldsymbol{v_B})$, traslato di $\frac{\boldsymbol{v}_A+\boldsymbol{v}_B}{2}$ dall'apice.

2.2 Guarantees

Proviamo che RVO possa essere usato per generare *collision-free* e *oscillation-free* per ogni agente.

2.2.1 Collision-free Navigation

Abbiamo $\boldsymbol{v_A}$ e $\boldsymbol{v_B}$ velocitá correnti, rispettivamente, di A e B, e permettiamo di scegliere ad entrambi la nuova velocitá ($\boldsymbol{v'_A}$ e $\boldsymbol{v'_B}$) fuori da ogni Reciprocal Velocity

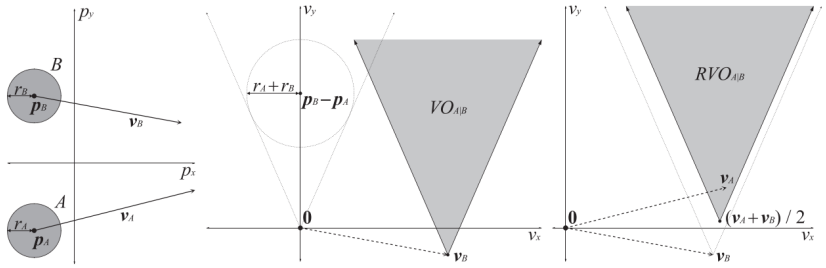


Figura 2.1: Costruzione del RVO rispetto al VO

Obstacle. Rispettando il *teorema* e la *legge* seguenti, essi ci garantiscono uno stato di *safe* se e solo se entrambi gli agenti scelgono lo stesso lato (*same side*) per passare ogni altro agente.

Teorema 1 (Collision-Free)**.**
$\boldsymbol{v'}_A \overrightarrow{\notin} RVO_{A,B}(\boldsymbol{v}_B, \boldsymbol{v}_A) \wedge \boldsymbol{v'}_B \overrightarrow{\notin} RVO_{B,A}(\boldsymbol{v}_A, \boldsymbol{v}_B) \Rightarrow \boldsymbol{v'}_A \overrightarrow{\notin} VO_{A,B}(\boldsymbol{v'}_B) \wedge \boldsymbol{v'}_B \overrightarrow{\notin} VO_{B,A}(\boldsymbol{v'}_A)$

2.2.2 Same Side

Possiamo garantire che entrambi gli agenti prenderanno automaticamente la nuova velocitá nello stesso lato, se ogni agente prenderá la velocitá al di fuori di ogni RVO, tale che differisca del minimo possibile (*closet*) dalla velocitá corrente dell'agente.
Ogni agente A avrá una velocitá $\boldsymbol{v_A} + \boldsymbol{u}$ *closet* a $\boldsymbol{v_A}$ fuori dal RVO di B, tale che B avrá $\boldsymbol{v_B}$ - $\boldsymbol{u}$ *closet* a $\boldsymbol{v_B}$ fuori dal RVO di A. Se per A questa *closet* velocitá sará sulla destra (o sinistra) di RVO di B, allora B sceglierá la *closet* velocitá sullo stesso lato (e viceversa).
Questo é provato dalla legge seguente:

Legge 1 (Same Side)**.**
$\boldsymbol{v}_A + \boldsymbol{u} \notin RVO_{A,B}(\boldsymbol{v}_B, \boldsymbol{v}_A) \Leftrightarrow \boldsymbol{v}_B - \boldsymbol{u} \notin RVO_{B,A}(\boldsymbol{v}_A, \boldsymbol{v}_B)$

2.2.3 Oscillation-Free

La scelta delle *closet* velocitá, al di fuori degli RVO, garantisce la *oscillation-free navigation* .Questo é provato dal teorema seguente:

Teorema 2 (Oscillation-Free)**.**
$\boldsymbol{v}_A \in RVO_{A,B}(\boldsymbol{v}_B, \boldsymbol{v}_A) \Leftrightarrow \boldsymbol{v}_A \in RVO_{A,B}(\boldsymbol{v}_B - u, \boldsymbol{v}_A + u)$

Quindi, la vecchia velocitá $\boldsymbol{v_A}$ di A é all'interno del nuovo RVO di B, dando le nuove velocitá $\boldsymbol{v_A} + \boldsymbol{u}$ e $\boldsymbol{v_B}$ - $\boldsymbol{u}$ per l'agente A e B. Stesse premesse vengono rispettate per B. Pertanto, dopo aver scelto la nuova velocitá, le vecchie candidate non saranno piú valide e non verranno piú scelte. Infatti, scegliendo la *closet* velocitá fuori dal RVO di A e di B, gli RVO rimarranno esattamente nella stessa posizione. Quindi $\boldsymbol{v_A}$ + $\boldsymbol{u}$ e $\boldsymbol{v_B}$ - $\boldsymbol{u}$ saranno ancora le velocitá piú vicine alle velocitá preferite tra tutte quelle ammissibili. Di conseguenza non si verificheranno traiettorie oscillatorie.

Capitolo 3

Hybrid Reciprocal Velocity Obstacle

Con le nozioni esplicitate precedentemente per VO e RVO, HRVO permette di diminuire le oscillazioni e creare traiettorie *smooth*, (osservare l'immagine del capitolo per comprendere alcuni aspetti sulla CL di RVO).

3.1 Hybrid Reciprocal Velocity Obstacle

Per l'agente $\boldsymbol{A}$ e $\boldsymbol{B}$, se $\boldsymbol{v_A}$ é sulla destra del *centreline* (CL) di $RVO_{\mathrm{A,B}}$, il quale per simmetria $\boldsymbol{v_B}$ sará sul lato destro della *centreline* di $RVO_{\mathrm{B,A}}$, noi speriamo che A scelga la velocitá sulla destra di $RVO_{\mathrm{A,B}}$. Per favorire questo, l' *RVO* é allargato sostituendo il bordo sul lato dove desideriamo che il robot non passi, in questo caso il lato sinistro, dal bordo della $VO_{\mathrm{A,B}}$. L'apice della risultante del cono corrisponde al punto di intersezione tra il lato destro della $RVO_{\mathrm{A,B}}$ e il lato sinistro della $VO_{\mathrm{A,B}}$. Se la $\boldsymbol{v_A}$ stá a sinistra della *centreline* (CL), noi replichiamo la stessa procedura scambiando la sinistra con la destra. Per questo é chiamato ibrido tra RVO e VO, $HRVO_{\mathrm{A,B}}$.

3.1.1 New Velocity

La computazione della nuova velocitá chiamata $\boldsymbol{v^{new}_{Ai}}$, al di fuori della combinazione dei HRVO, é la minima distanza tra la velocitá corrente e la velocitá preferita:

$$v_{Ai}^{new} = argmin_{v \notin HRVO_{Ai}} norm(v - v_{Ai}^{pref}). \tag{3.1}$$

Per computare tale velocitá viene utilizzato un algoritmo chiamato ClearPath, combinando tutti gli *HRVO* come intersezioni di segmenti, dove le coppie dei punti di intersezione all'interno del *HRVO* vengono scartati. Le possibili velocitá ammissibili saranno tutte le intersezioni dei coni sul bordo di *HRVO*. Aggiungendo la proiezione della velocitá preferita, la nuova velocitá sará selezionata tra la minima distanza che c'é tra le velocitá ammissibili (intersezioni dei coni) e la velocitá preferita del robot $\boldsymbol{v^{pref}_{Ai}}$.

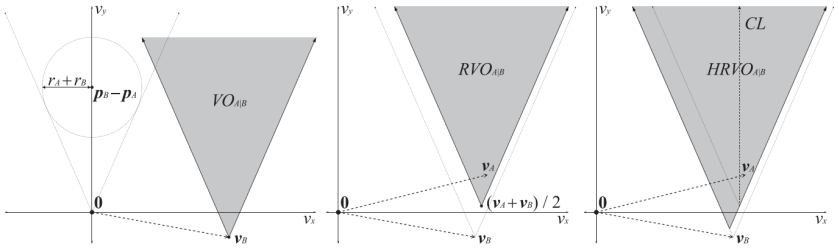


Figura 3.1: Costruzione del HRVO rispetto gli algoritmi precedentemente descritti

Se non si riesce a trovare una possibile velocitá ammissibile, la procedura viene ripetuta riutilizzando l'algoritmo del ClearPath.

3.1.2 Descrizione dell'algoritmo

Descrizione dell'implementazione dell'algoritmo HRVO:

```
Input A = List of robots, O = List of obstacles
loop
for all Ai in A do
Sense pAi and vAi
for all Aj E A such that j != i do
Sense pAj and vAj
Construct VOAi|Aj and RVOAi|Aj
Locate centerline CL of RVOAi|Aj
if vA is right of CL then
Replace left side of RVOAi|Aj with left side of VOAi|Aj to
    construct HRVOAi|Aj
else
Replace right side of RVOAi|Aj with right side of VOAi|Aj to
    construct HRVOAi|Aj
end if
Expand HRVOAi|Aj to HRVOAi|Aj
end for
for all Oj E O do
Sense pOj and vOj as appropriate
Construct VOAi|Oj
Expand VOAi|Oj to VOAi|Oj
end for
Construct HRVOAi from all HRVOAi|Aj and VOAi|Oj
Compute preferred velocity vprefAi
Compute new velocity vnewAi !E HRVOAi closest to vprefAi
Compute control inputs from vnewAi
Apply control inputs to actuators of Ai
end for
end loop
```

Capitolo 4

Optimal reciprocal Collision Avoidance

Optimal reciprocal Collision Avoidance *ORCA* é un algoritmo basato su VO. Con questo procedimento si riduce il problema del *collision-free* con una soluzione lineare computazionalmente ridotta. Ottimo anche per simulazioni popolate da centinaia di robot in uno spazio di lavoro limitato.

4.1 Optimal reciprocal Collision Avoidance

Come gli altri algoritmi precedentemente osservati, ogni robot tiene conto della velocitá, del raggio e della posizione corrente osservata dagli altri robot al fine di evitare collisioni. Inoltre puó selezionare la sua velocitá dal suo spazio velocitá (*velocity space*), dove alcune aree di questo spazio sono state etichettate come "proibite", per la presenza di altri robot.
Questa formulazione, per ogni robot, crea un semipiano (*half-plane*) delle velocitá dove è consentito essere per non avere delle collisioni. Perció il robot selezionerá la nuova velocitá ottimale ($\boldsymbol{v}^{\mathbf{opt}}$) dalla intersezione di tutti i semipiani dove consentito essere. Per computare la nuova velocitá, si puó utilizzare in modo efficente la programmazione lineare (*linear programming*). In determinate condizioni, con densitá di robot elevate, la programmazione lineare potrebbe non trovare un risultato, in questo caso selezioniamo la piú sicura velocitá possibile aggiungendo una terza dimensione alla programmazione lineare.

4.1.1 Definition

Dalle informazioni riportate precedentemente, selezioniamo per i robot A e B l'insieme delle velocitá permesse, che chiameremo $\boldsymbol{V_A}$ e $\boldsymbol{V_B}$ tale che $\boldsymbol{V_A}$ sia equivalente al cono delle collisioni ($CC_{A,B} \simeq CA^{\tau}_{A,B}(V_B)$) con un determinato *time horizon*, τ, che permette di creare un tronco di cono con apice nell'origine delimitato da due rette tangenti a r_a+r_b, centrate in p_b - p_a. La dimensione della parte troncata dipende dal valore di τ; il cono é troncato da un arco di raggio $\frac{r_a+r_b}{\tau}$ centrato in $\frac{p_b-p_a}{\tau}$. Equivalentemente per $\boldsymbol{V_B}$ tale che $CA^{\tau}_{B,A}(V_A) = V_B$, queste aree garantiscono l'anticollisione per almeno un certo tempo τ.

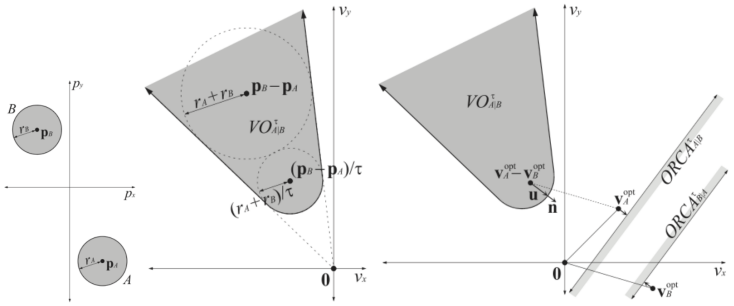


Figura 4.1: Costruzione del cono $\mathrm{VO}_{\mathrm{A,B}}$ con un determinato τ e la delimitazione del semipiano delle $\mathrm{ORCA}_{\mathrm{A,B}}$ e $\mathrm{ORCA}_{\mathrm{B,A}}$

Per ogni area viene creato un insieme di velocitá vicine alle velocitá ottimali (*optimization velocities*) $\boldsymbol{v^{opt}}_{\mathbf{A}}$ per A e $\boldsymbol{v^{opt}}_{\mathbf{B}}$ per B, tale che noi denoteremo questi insiemi come $ORCA^{\tau}{}_{\mathrm{A,B}}$ per A e $ORCA^{\tau}{}_{\mathrm{B,A}}$ per B, definiti formalmente come segue:

Definizione 1 (Optimal Reciprocal Collision Avoidance)**.**
$ORCA^{\tau}{}_{\mathrm{A,B}}$ e $ORCA^{\tau}{}_{\mathrm{B,A}}$ sono definite reciprocamente come *collision-avoiding*, tale che $CA^{\tau}{}_{\mathrm{A,B}}(ORCA^{\tau}{}_{\mathrm{B,A}})= ORCA^{\tau}{}_{\mathrm{A,B}}$ e $CA^{\tau}{}_{\mathrm{B,A}}(ORCA^{\tau}{}_{\mathrm{A,B}})= ORCA^{\tau}{}_{\mathrm{B,A}}$, e per ogni raggio $r > 0$:

$$|ORCA^{\tau}_{A,B} \cap D(v_A^{opt}, r)| \;=\; |ORCA^{\tau}_{B,A} \cap D(v_B^{opt}, r)|$$
$$\geq$$
$$min(|V_A \cap D(v_A^{opt}, r)|,\; |V_B \cap D(v_B^{opt}, r)|).$$

Questo significa che $ORCA^{\tau}{}_{\mathrm{A,B}}$ e $ORCA^{\tau}{}_{\mathrm{B,A}}$ contengono piú velocitá vicine a $\boldsymbol{v^{opt}}_{\mathbf{A}}$ e $\boldsymbol{v^{opt}}_{\mathbf{B}}$ che dell'insieme delle velocitá di *collision-avoiding*.

Possiamo costruire geometricamente $ORCA^{\tau}{}_{\mathrm{A,B}}$ e $ORCA^{\tau}{}_{\mathrm{B,A}}$ assumendo che A e B adottino rispettivamente $\boldsymbol{v^{opt}}_{\mathbf{A}}$ e $\boldsymbol{v^{opt}}_{\mathbf{B}}$, inoltre assumiamo che A e B siano in collisione se $\boldsymbol{v^{opt}}_{\mathbf{A}}$ - $\boldsymbol{v^{opt}}_{\mathbf{B}} \in VO^{\tau}{}_{\mathrm{A,B}}$. Consideriamo $\boldsymbol{u}$ essere un vettore che inizia da $\boldsymbol{v^{opt}}_{\mathbf{A}}$ - $\boldsymbol{v^{opt}}_{\mathbf{B}}$ e punta sul punto piú vicino al bordo del cono:

$$u = (argmin_{v \in VO^{\tau}_{A,B}} ||v - (v_A^{opt} - v_B^{opt})||) - (v_A^{opt} - v_B^{opt}), \tag{4.1}$$

assumiamo che $\boldsymbol{n}$ sia un versore che attraversa il bordo di $VO^{\tau}{}_{\mathrm{A,B}}$ fino al punto ($\boldsymbol{v^{opt}}_{\mathbf{A}}$ - $\boldsymbol{v^{opt}}_{\mathbf{B}}$)+$\boldsymbol{u}$. Quindi, $\boldsymbol{u}$ é il piú piccolo cambiamento richiesto per le velocitá relative di A e B, per accorgersi della collisione al tempo τ. Per spartirsi la responsabilitá della collisone, il robot A adatta la sua velocitá per almeno $\frac{1}{2}$ $\boldsymbol{u}$ assumendo che B si prendi cura dell'altra parte. Quindi, l'insieme delle velocitá permesse $ORCA^{\tau}{}_{\mathrm{A,B}}$ per A, é un semipiano posizionato in direzione di $\boldsymbol{n}$ con punto d'origine $\boldsymbol{v^{opt}}_{\mathbf{A}}$ $+\frac{1}{2}$ $\boldsymbol{u}$. Piú nello specifico:

$$ORCA^{\tau}_{A,B} = \{\boldsymbol{v} | (\boldsymbol{v} - (\boldsymbol{v}_A^{opt} + \frac{1}{2}\boldsymbol{u})) \geq 0\}. \tag{4.2}$$

L'insieme $ORCA^{\tau}{}_{\mathrm{B,A}}$ per B é definito simmetricamente. Le equazioni qui sopra riportate, si applicano anche se A e B non sono su una rotta di collisione quando

adottano le loro velocitá di ottimizzazione, $\boldsymbol{v}^{\mathbf{opt}}{}_{\mathbf{A}}$- $\boldsymbol{v}^{\mathbf{opt}}{}_{\mathbf{A}} \notin VO^{\tau}{}_{\mathrm{A,B}}$. In questo caso, ogni robot prenderá metá della responsabilitá per rimanere in una traiettoria di *collision-free*.

4.1.2 Basic Approach

Ogni robot A esegue un ciclo continuo di *sensing* e *acting* a ogni time-step Δt. Ad ogni iterazione, i robot acquisiscono il raggio, la posizione e la velocitá optimale corrente degli altri robot (e di sé stesso). Basandosi su queste informazioni, il robot deduce il semipianio delle velocitá permesse, $ORCA^{\tau}{}_{\mathrm{A,B}}$, rispettando ogni robot B. L'insieme delle velocitá permesse per A, rispetto ciascun robot, é l'intersezione di ogni semipiano, che noi denotiamo con $ORCA^{\tau}{}_{\mathrm{A}}$:

$$ORCA^{\tau}_{A} = D(0, v^{max}_{A}) \cap \bigcap_{B \neq A} ORCA^{\tau}_{A,B} \tag{4.3}$$

Nel passo successivo, il robot seleziona la nuova velocitá $\boldsymbol{v}^{\mathbf{new}}{}_{\mathbf{A}}$, per sé stesso, la quale sará la piú vicina possibile alla velocitá preferita $\boldsymbol{v}^{\mathbf{pref}}{}_{\mathbf{A}}$ tale che, questa velocitá, risiedi almeno all'interno dell'insieme delle velocitá permesse:

$$v^{new}_{A} = argmin_{v \in ORCA^{\tau}_{A}} ||v - v^{pref}_{A}||. \tag{4.4}$$

Alla prossima sezione spiegheremo come possa scegliere questa nuova velocitá in modo efficiente.
Finalmente, il robot riceverá la sua nuova posizione:

$$p^{new}_{A} = p_{A} + v^{new}_{A} \Delta t, \tag{4.5}$$

e il ciclo di *sensing-acting* verrá ripetuto.
Per computare la nuova velocitá in modo efficiente, viene utilizzata la *programmazione lineare*, dove $ORCA^{\tau}{}_{\mathrm{A}}$ é una regione convessa limitata, costruita dalle intersezioni dei semipiani delle velocitá permesse. Se questo procedimento non porta a nessun risultato, viene aggiunta una terza dimensione, che é la distanza della velocitá preferita, cambiando il procedimento in un problema *quadratico*.

4.1.3 Choosing the Optimization Velocity

Per la scelta della nuova velocitá in modo efficiente:

- $\boldsymbol{v}^{\mathbf{opt}}{}_{\mathbf{A}} = 0$ per ogni robot A. Per qualsiasi robot B, il punto 0 si trova sempre al di fuori del $VO^{\tau}{}_{\mathrm{A,B}}$. Quindi il semipiano, $ORCA^{\tau}{}_{\mathrm{A,B}}$, include sempre almeno la velocitá 0. Infatti la linea che delimita $ORCA^{\tau}{}_{\mathrm{A,B}}$ é perpendicolare alla linea che collega le posizioni attuali di A e B.
  Un inconveniente, nell'impostare la nuova velocitá a 0, é che il comportamento del robot potrebbe portare ad una situazione di stallo globale, facendo convergere tutte le velocitá dei robot a 0, in una simulazione densamente popolata.

- $\boldsymbol{v}^{\mathbf{opt}}{}_{\mathbf{A}} = \boldsymbol{v}^{\mathbf{pref}}{}_{\mathbf{A}}$ per ogni robot A. La velocitá preferita fa parte dello stato interno di ogni robot, quindi non puó essere osservata dagli altri robot. Peró ipoteticamente se ogni velocitá ottimale fosse impostata come velocitá preferita, per ciascun robot, questo potrebbe funzionare bene solo in un ambiente poco popolato.

- $\boldsymbol{v}^{\mathbf{opt}}{}_{\mathbf{A}} = \boldsymbol{v}_{\mathbf{A}}$ per tutti i robot A. Impostare la velocitá corrente come velocitá ottimale, sarebbe il comportamento ideale tra le scelte elencate precedentemente. La velocitá corrente si adatta automaticamente alla situazione, essa sará piú vicina alla velocitá preferita in ambienti di bassa densitá mentre sará piú vicina allo 0 in caso di ambienti ad alta densitá. Soprattutto la velocitá corrente puó essere osservata dagli altri robot.
  Purtroppo la programmazione lineare potrebbe fallire in condizioni di alta densitá. Perció la velocitá *collision-free* non puó essere garantita. Inoltre come precedentemente anticipato, viene utilizzata una dimensione in piú portando il problema in 3-D *linear program*.

Capitolo 5

Detect Velocity Obstacle

In questo capitolo introduco l'algoritmo da me implementato in Matlab, basandomi sulla libreria RVO2 v2.0.2 scritta in C++, studiata e implementatata da *Jur van den Berg, Stephen J. Guy, Jamie Snape, Ming C. Lin, and Dinesh Manocha del Department of Computer Science, University of North Carolina at Chapel Hill.*
Nel primo affronto del problema, lo scopo era di interpretare e riscrivere la libreria, sopra citata, in Matlab; purtroppo per alcuni problemi implementativi (*linear program e quadratic program*) ho dovuto tenere conto di alcuni aspetti fondamentali e aggirare il problema realizzando un ibrido tra RVO e HRVO, prima esplicitato, che ho chiamato *Detect Velocity Obstacle.*

5.1 Detect Velocity Obstacle

DVO rispecchia l'andamento ciclico di *sensing-acting* per ogni robot, questi hanno quindi la possibilitá di osservare la velocitá, il raggio e la posizione corrente di ogni robot a ogni time-step della simulazione.

5.1.1 Struttura di ogni agente

Le proprietá di ogni agente, prevedono:

- *Identity* é un numero che identifica l'agente.
- *Position* rappresenta la posizione corrente dell'agente, informazione condivisa agli altri agenti.
- *Target* rappresenta la posizione finale *goal*, informazione non condivisa agli altri agenti.
- *Speed* rappresenta la velocitá corrente, informazione condivisa agli altri agenti.
- *PrefSpeed* rappresenta la velocitá preferita, informazione non condivisa agli altri agenti.
- *Radius* rappresenta il raggio dell'agente, informazione condivisa a ogni agente.
- *New_Position* rappresenta la nuova posizione dell'agente, informazione viene condivisa a ogni agente nel ciclo successivo.

- *New_Speed* rappresenta la nuova velocitá computata, sará condivisa con gli altri agenti nel ciclo successivo.
- *NeighborDist* rappresenta le distanze tra i robot, informazione non condivisa agli altri agenti, calcolata conoscendo la posizione corrente di sé stesso e degli altri agenti nella scena.

```
%%% creazione di un oggetto/classe Agente
classdef Agent < handle

% properties: instanza di ogni agente
    properties
        Identity
        Position
        Target
        Speed
        PrefSpeed
        Radius
        New_Position
        New_Speed
        NeighborDist
    end

% methods: metodi della classe Agente
    methods ( Access = public )

% costruttore
        function obj = Agent( id, pos, goal, vel, p_vel, rad,
           n_p, n_s, n_d, F )
            if nargin > 0
                for i=1:F

                    obj(i).Identity = id;
                    obj(i).Position = pos;
                    obj(i).Target = goal;
                    obj(i).Speed = vel;
                    obj(i).PrefSpeed = p_vel;
                    obj(i).Radius = rad;
                    obj(i).New_Position = n_p;
                    obj(i).New_Speed = n_s;
                    obj(i).NeighborDist = n_d;

                end
            end
        end
        ...
```

5.1.2 Find Velocity

Per la computazione della nuova velocitá, la funzione di *Agent.m*, $findVelocity(obj, others, time)$ prende in input l'agente A=*obj*, gli altri agenti B_i=*others* e *time-step*=*time*. Rappresenta la funzione principale perché in essa vengono utilizzate le funzioni per il calcolo

dei coni, delle velocitá ammissibili, l'aggiornamento della nuova velocitá e della nuova posizione e di conseguenza anche della velocitá preferita. *Admin_Speeds(obj,time)* é una funzione che restituisce tutte le possibili velocitá ammissibili, *ad_vel*, che l'agente puó selezionare in quel determinato istante temporale; in seguito viene fornita una spiegazione piú formale.

```
% funzione che calcola la nuova velocit\'a
    function findVelocity(obj,others,time)
            speed_ok=0;
            ad_vel=[];
            % calcolo le ammissibili velocit\'a
            ad_vel=Admin_Speeds(obj,time);
            ...
```

In seguito vengono costruiti tutti i coni che delimitano tutte le velolcitá che portano a una collisione con un ostacolo. Con la funzione *cone_VO(obj,others(i),time)* si calcolano i coni traslati delle velocitá dell'altro agente preso in osservazione, diviso 2 *(other.Speed*time)/2*. Successivamente vengono riportati i cicli per selezionare tutte le velocitá esterne ai coni precedentemente calcolati. Alla fine la funzione restituirá un array con tutte le velocitá considerate *safe*.

```
            ...
            for i=1:length(others)
                % se non sono lo stesso agente
                if(others(i).Identity ~= obj.Identity)
                        % calcolo il cono delle collisioni
                        [cone]=cone_VO(obj,others(i),time);
                        % controllo se le velocit\'a
                            ammissibili sono dentro
                        % o fuori al cono delle collisioni
                        in=inpolygon(ad_vel(:,1),ad_vel(:,2),
                            cone(1,:),cone(2,:));
                        ad_vel=[ad_vel,in];
                        for q=1:length(ad_vel(:,end))
                            if ad_vel(q,4) == 1
                                ad_vel(q,:)=[0,0,0,0];
                            end
                        end
                        ad_vel( all(~ad_vel,2), : ) = [];
                        % ritorno le velocit\'a ammissibili
                            con la relativa
                        % distanza tra la sua v_pref
                        ad_vel=ad_vel(:,1:3);
                       speed_ok=1;
                end
            end
            ...
```

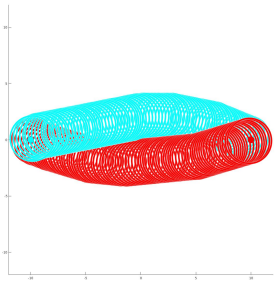
Una volta instanziato l'array *ad_vel*, estraggo la velocitá che differisce il meno possibile dalla distanza della velocitá preferita dell'agente *obj*. Successivamente viene aggiornata la posizione e la nuova velocitá appena calcolata. Infine aggiorno la nuova velocitá preferita con gli aggiornamenti appena riportati.

```
            ...
            if speed_ok == 1

                % estraggo da ad_vel la pi\'u piccola distanza
                    tra v_pref e velocit\'a ammissibile
                [values,index]=min(ad_vel(:,3));

                % setto la nuova posizione e velocit\'a
                obj.New_Position=ad_vel(index,1:2);
                obj.New_Speed=(obj.New_Position-obj.Position)/
                    time;

                % aggiorno la nuova posizione
                obj.New_Position(1) = (obj.Position(1) + obj.
                    New_Speed(1)*time);
                obj.New_Position(2) = (obj.Position(2) + obj.
                    New_Speed(2)*time);
            end

            % aggiorno la velocit\'a preferita
            setNewPrefSpeed(obj);
    end
end
```

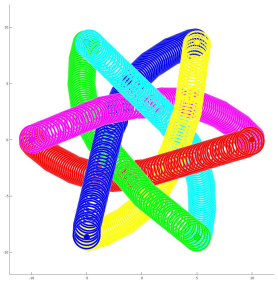
5.1.3 Admissible velocity

In questa sezione, riporto parte del codice della creazione dei possibili candidati a velocitá ammissibili.
La funzione *up_down* permette di tenere tutte le velociá a destra o a sinistra della velocitá preferita dell'agente.

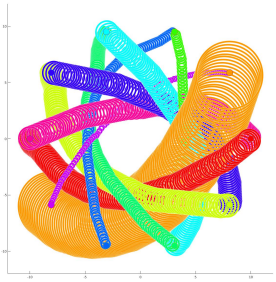
```
...
% determino se un punto si trova a destra o a sinistra della
   velocit\'a preferit\'a di obj (tengo solo velocit\'a a
   destra della v_pref)
for i=1:numel(right_velocity)/2
    Possible_velocity=right_velocity(i,:);
    is_up=up_down(Possible_velocity,obj.Position,v_pref);
    flag(i)=is_up;
end
...
```



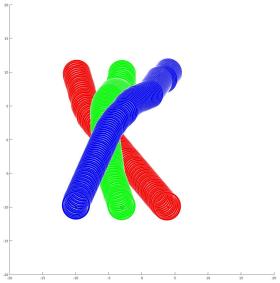
(a) Due agenti di raggio 2



(b) Sei agenti di raggio 1



(c) Dodici agenti di raggio random



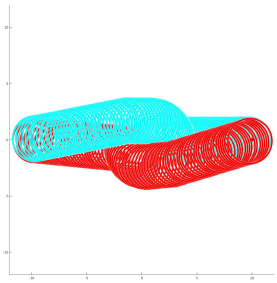
(d) Tre agenti di raggio 1

Figura 5.1: Stampa delle traiettorie della simulazione

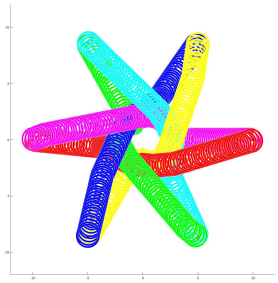
5.1.4 Risultati

Riporto le stampe delle traiettorie di alcune simulazioni.
Nella figura 5.1(a) mi sono posto il problema di un collision-avoidance tra due agenti che si scambiano la posizone di A con il target di B (bug riportato della librery RVO2_ORCA).

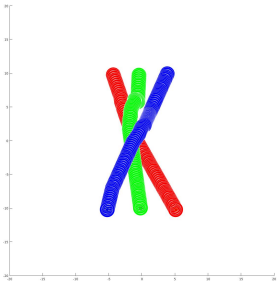
- Nella figura 5.1(b) viene riportata l'immagine delle traiettorie descritte da sei agenti con lo stesso raggio, uguale a 1.3, in una configurazione a circonferenza di raggio 10 .
- Nella figura 5.1(c) si riporta la stessa configurazione a circonferenza peró con il doppio degli agenti e con l'aggiunta della possibilitá di settare i raggi a piacimento.
- Nella figura 5.1(d) viene configurata una simulazione dove i due agenti piú esterni si devono scambiare la posizione del target rispetto l'agente verde.



(a) Due agenti di raggio 2



(b) Sei agenti di raggio 1



(c) Tre agenti di raggio 1

Figura 5.2: Stampa delle traiettorie della simulazione con τ

5.1.5 Risulati con τ

Mi sono posto lo stesso problema con l'aggiunta di utilizzare un cono troncato utilizzato nelle ORCA, questi sono i risultati:

- le traiettorie sono piú rettilinee.
- gli agenti nella parte centrale sono piú vicini rispetto a non utilizzarlo.
- con l'aumento di densitá della popolazione della scena é meglio utilizzare un τ maggiore di 2.

Capitolo 6

Bibliografia

- P. Fiorini, Z. Shiller. Motion planning in dynamic environments using Velocity Obstacles. Int. Journal of Robotics Research 17(7), pp. 760-772, 1998.
- J. van den Berg, M. Lin, D. Manocha. Reciprocal Velocity Obstacles for real-time multi-agent navigation. IEEE Int. Conf. on Robotics and Automation, pp. 1928–1935, 2008
- J. Snape, J. van den Berg, S. Guy, D. Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. IEEE/RSJ Int. Conf. Intell. Robot. Syst., 2009.
- Jur van den Berg, Stephen J. Guy, Ming C. Lin, and Dinesh Manocha, "Reciprocal n-body collision avoidance," in Cédric Pradalier, Roland Siegwart, and Gerhard Hirzinger (eds.), Robotics Research: The 14th International Symposium ISRR, Springer Tracts in Advanced Robotics, vol. 70, Springer-Verlag, Berlin/Heidelberg, Germany, May 7, 2011, pp. 3-19.