L’approccio è quello di distribuire e localizzare quanto più possibile le conoscenze della flotta, ossia di lasciare che messaggi vengano scambiati solo tra robot vicini. Per vicinanza si intende un certo range fisicamente raggiungibile dal mezzo di comunicazione (es. WiFi), rappresentabile come una palla di raggio R attorno a ciascun robot. La palla i-esima è tale che tutti i robot al suo interno

possano supporsi interagenti in spazio e tempo con il robot i. Inoltre, si suppone che R sia grande a sufficienza da consentire a due robot in comunicazione di rimanere comunque a distanza di sicurezza.

L’idea generale è che i robot tra loro vicini si scambino una porzione della propria traiettoria futura e un codice di identificativo. A due a due i veicoli calcolano e conservano la parte comune tra i loro percorsi, detta *sezione critica*. Stabiliscono un ordine di attraversamento di tale sezione (vedi seguito), eventualmente frenando opportunamente, aggiornando infine la stima della traiettoria per i passi successivi.

Per nomenclatura, definiamo quanto segue:

* *path* o cammino o traccia, l’insieme di punti attraversati dal veicolo, senza legge oraria;
* *traiettoria*, il percorso precedente ma temporizzato, stimato dal veicolo stesso;
* *envelope*, l’impronta che per via della sua geometria il veicolo lascia lungo il percorso.

Nel seguito si considera la definizione di traiettoria come inclusiva anche dell’envelope.

* *punto critico*, punto sul proprio percorso, oltre il quale non procedere per sicurezza.

Si vedrà che il punto critico verrà sempre a coincidere con l’inizio di una sezione critica, dal momento che questo sarà continuamente aggiornato, e al peggio risulterà coincidere col bordo del veicolo che si aspetta o segue, permettendo così un comportamento “a inseguimento”.

Si aggiungono ora preliminarmente alcune osservazioni e specifiche di funzionamento:

1. Deve essere consentita la possibilità per i veicoli di comunicare con asincronia.
2. L’envelope del path futuro può essere affetto da incertezze e può allargarsi nei punti più lontani dalla posizione corrente.
3. Ammettendo che sia noto l’ambiente, una sezione critica particolare è quello per cui i path si intersecano in **corridoi**, la cui lunghezza è tale da impedire ai robot che si trovano ai due capi di comunicare fra loro: per questo solo caso sarà necessario introdurre dei ripetitori che conservino l’informazione del robot più vicino e agisca da semaforo, in maniera tale da far aspettare che uno dei due liberi del tutto la sezione. Per tale caso si dovrà pensare a una funzione protocollo opportuna, diversa da quelle che agiscono nel resto dell’ambiente.
4. In generale, sono da evitare **sezioni critiche lunghe anche in spazi aperti**, per lo stesso problema di distanza espresso nel punto 3; ma a differenza del caso precedente - noto a priori - non possono essere prevedibili ripetitori. O si suppone un algoritmo di pianificazione centrale che impedisca tali sezioni e che le preveda il più brevi possibile; oppure, in un’ottica di maggiore decentralizzamento, in una situazione head-to-head, si affida una ripianificazione a uno dei due, trattando l’altro come ostacolo. Per il momento, si preferisce la prima, confidando di poter passare alla seconda in momenti futuri.

Per quanto detto si definisce la classe dei veicoli come segue:

Classe Veicoli (pos\_iniziale, goal, id, “categoria”)

Traiettoria # envelope temporizzato

Id

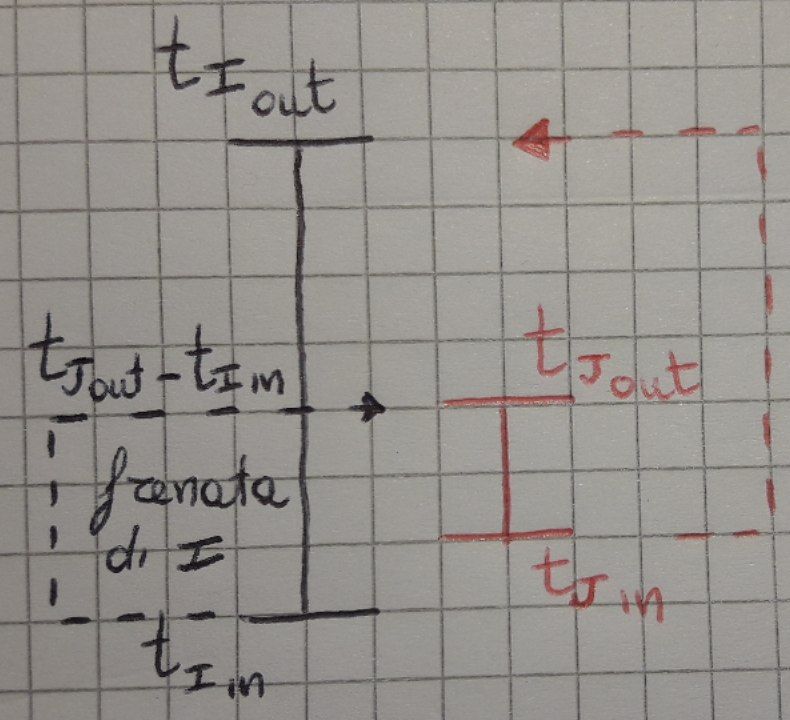
Priorità

Altre variabili # es. vel\_max, geometria

Flag AdiacenzeCritiche # (vedi seguito)

con differenti “categorie” si rappresentano veicoli con diverse caratteristiche e priorità.

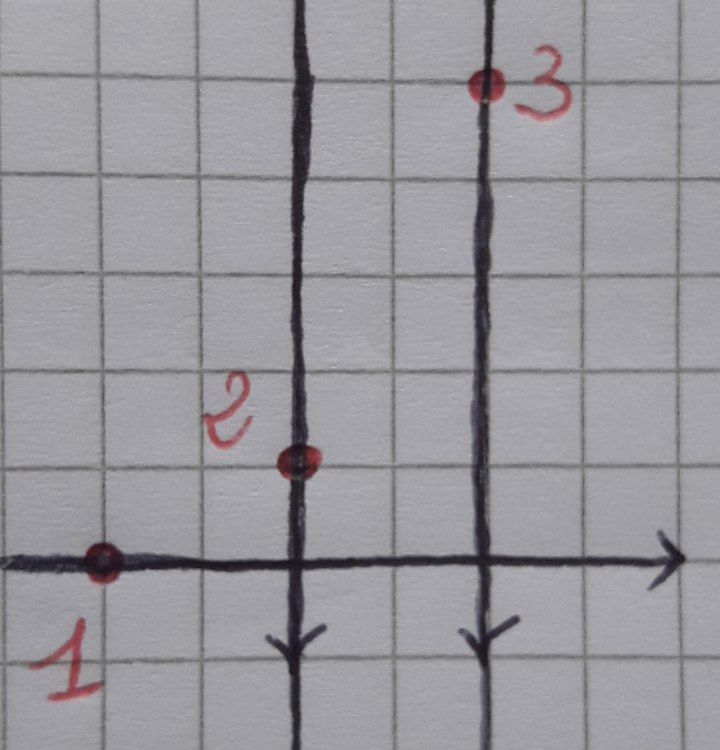
**CALCOLO PRECEDENZE standard tra due robot**

Si vorrebbe fermare i veicoli il meno possibile. Si immagini una sezione critica come una risorsa accessibile nel tempo da un solo task (robot) alla volta. Nel caso in cui alla velocità corrente due robot si trovassero a “richiederne l’accesso” in tempi sovrapposti, si vuole allora non che uno si fermi, ma che semplicemente rallenti quel tanto sia necessario per entrarvi subito dopo che l’altro sia uscito. Tra i due, la frenata viene compiuta da quello che per soddisfare la richiesta precedente consuma meno energia. Nell’esempio in figura, Rosso deve frenare maggiormente perché il suo tempo di ingresso si collochi dopo quello Nero d’uscita: frena Nero. Si fa poi notare che lavorare coi tempi permette intrinsecamente di considerare sia le velocità dei veicoli che la distanza spaziale dalla sezione critica!

A questo procedimento si antepongono due considerazioni:

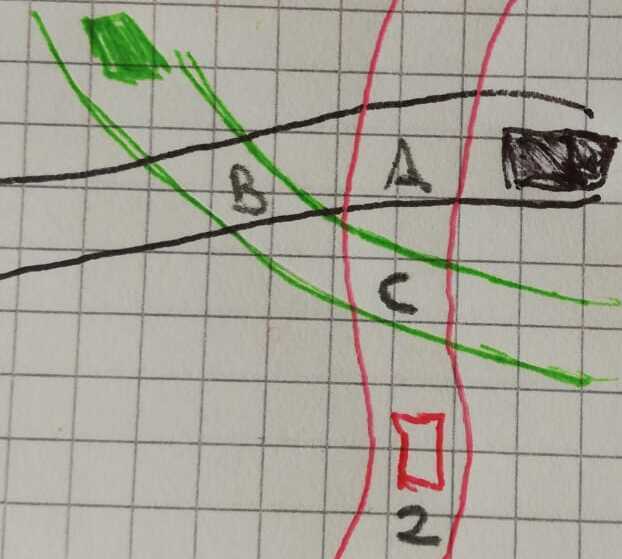
* Nel caso - comunque da scongiurare - in cui uno dei due veicoli non abbia più modo di frenare al di qua della sezione, è sempre da far passare per primo;
* è possibile prevedere dei codici di priorità per cui una certa classe di veicoli accede prima, senza computare le precedenze coi tempi (es. Ambulanza ha priorità elevata).

**INCROCI tra più robot**

Risulta evidente quale potrebbe essere un problema dello stabilire la precedenza localmente procedendo per coppie: se un robot della coppia frena a causa di un ulteriore robot esterno alla coppia, si genera una variazione che non avevo previsto e che può cambiare le precedenze. Nell’esempio, 1 e 3 avrebbero stabilito che per vicinanza sarebbe passato prima 1, che però frena per via della sezione critica che ha con 2. 3 potrebbe così trovarsi più vicino alla sezione e passarvi anziché aspettare ancora 1. 

Le possibili soluzioni pensate sono due. O si suppone che l’intervallo con cui i dati vengono aggiornati e scambiati sia sufficientemente rapido da poter informare 3 della nuova traiettoria di 1 al passo successivo; oppure che ciascun robot non conservi le sole sezioni critiche proprie, ma anche quelle dei vicini, computando in un solo intervallo l’intero ordine di precedenze. Al momento, appare più semplice e ragionevole la prima.

**SC ADIACENTI** *Definizione: |inizio SC successiva - fine SC precedente| < dimensione robot*

Quando almeno tre robot non possono liberare la prima SC senza occupare la seconda, si può avere un deadlock. N è più vicino ad A di R e passa prima, V prima di N su B, R prima di V su C: non potendo posteggiare al sicuro, si bloccherebbero a vicenda. Si pensa di accendere un flag se sul percorso ho SC adiacenti: se coi vicini ne raggiungo tre, mi accorgo del rischio deadlock e procedo come segue. 

Unisco le mie SC adiacenti, sicché una volta intrapresa la percorro interamente. Per l’ordine di precedenza, stavolta non valuto la vicinanza temporale ma lascio passare chi ha maggiore id.

Nel caso in esame, se V ha maggior id, libera B; N è vicino ad A e non ha più problemi a imboccarlo, infine va R.

**ALGORITMO 1. Esecuzione del singolo robot ad ogni passo k**

*Input: dati ricevuti dai robot i ∈ N, ossia i vicini*

1. Interseco path e calcolo sezioni critiche
   1. **if** due adiacenti **then** attivo flag
      1. **if** ricevo altri due flag attivi **then:** unisco S.C. # *necessario per deadlock*
2. Ordino per vicinanza le mie S.C.; # grazie a struttura dati TreeSet
3. **for** cs in SezioniCritiche:
   1. Calcolo ordine di precedenze e l’opportuna frenata;
   2. Aggiorno la stima nei tempi della parziale traiettoria futura;
4. Sleep finché non scade l’intervallo *Tc*, generalmente asincrono.

*Output: nuova traiettoria richiesta al controllore*

**ALGORITMO 2. CALCOLO SEZIONI CRITICHE (passo I dell’Algoritmo 1)**

*Input: Porzione di percorso dei Vicini*

*# Il dominio [0,1] di ogni path è convertito in vettore con una certa risoluzione*

*# (Es. su 100 celle, la 0 contiene l’inizio del cammino, la 99 è associata al valore 1, ossia alla fine)*

**For** ogni percorso altrui, lo interseco con il mio. **If** intersezione trovata **then**:

1. Scorro la mia impronta finché trovo il primo indice al quale interseco l’altro path: inizio SC;
2. Quando non interseco più avrò trovato l’indice di fine SC;
3. Similmente, scorro l’impronta dell’altro finché trovo i suoi indici di inizio e fine.

*Output: Indici iin, iout, jin, jout*

**ALGORITMO 3. CALCOLO ORDINE DI PRECEDENZE (passo III.A dell’Algoritmo 1)**

*Input: Sezione Critica(trj(i), trj(j), iin, iout, jin, jout); Traiettorie della coppia coinvolta nella SC; Flag.*

*# Detto t’ = t +, il tempo al quale ci potremo al più fermare*

Trovo da traiettoria gli istanti tIN e tOUT in cui i robot avranno raggiunto gli indici di ingresso e uscita;

**if** in *t’*entrambi i robot sono fuori dalla SC *( i.e. t****j****’ < tJ\_in e t****i****’ < tI\_in )* **then:**

**if** Priorità(j) > Priorità(i) **then:** passa prima j e ***i*** *calcola la frenata\*; # e viceversa*

**elseif** Priorità(j) == Priorità(i) **then**:

**if** *SCunione* **then** passa chi ha maggiore ID

**else** passa il più vicino temporalmente, cioè chi sarebbe costretto a frenare di più:

*( i.e. se [ tJ\_out - tI\_in ] < [ tI\_out - tJ\_in ] allora* ***i*** *calcola la frenata\*, vedi figura 1)*

**else** se l’altro (robot **j**) in *t’* sarà già dentro *( i.e. t****j****’ > tJ\_in )*

**then:** robot **i** aspetta che la SC si liberi\*;

**#else** entrambi dentro °

**#then** uno soltanto dei due ripianifica

° il problema si pone solo nel caso di lunghe intersezioni in spazio aperto

*Output: Nuovi tempi di attraversamento delle S.C.*

**\*ALGORITMO 3bis. CALCOLO FRENATA? (usato in seno al calcolo di precedenze dell’algoritmo 3)**

*# Da path trovo il mio punto di ingresso sI\_in . Conosco la mia v0 al passo attuale.*

*# Il robot* ***i*** *dovrà trovarsi in sI\_in al tempo tJ\_out , quando il robot* ***j*** *avrà liberato la SC.*

*s = v\*t + ½ a\*t^2*

*# usare funzioni “ForwardModel, ConstantAcce …. canStop, getTime”*

**DIMENSIONAMENTO**

Per garantire safety sarà necessario dimensionare il raggio di comunicazione in maniera opportuna, affinché si recepiscano le informazioni nei tempi necessari per un’eventuale frenata.

* **Asincronia dei Tempi di Campionamento**

Si fa notare che, volendo consentire ai veicoli di comunicare in maniera asincrona, bisogna cercare il tempo peggiore per cui un robot **i** “ignora” i dati altrui. Questo si verifica quando si leggono i dati di un vicino un’unità di tempo prima che questi abbia aggiornato le sue informazioni. Il robot **i** avrà così letto un’informazione vecchia e ne otterrà una aggiornata solo attendendo l’intera parte restante del proprio intervallo, cioè dopo Tci - 1, con **Tci** suo tempo di lavoro.

Inoltre, si osserva che per rilevare la presenza delle cosiddette “Sezioni Critiche Adiacenti”, è necessario un maggior numero di scambi; in particolare, si richiedono due intervalli di tempo per poter accendere il proprio flag e ricevere l’altrui.

A questi tempi va sommato un ulteriore caso peggiore, quello per cui un veicolo che viaggia alla sua massima velocità debba frenare fino a fermarsi, ossia stopTimeMax = velMax/accMax.

Convertendo questi vincoli temporali in un raggio spaziale, si ha R = (stopTimeMax+2Tcmax)\*velMax.

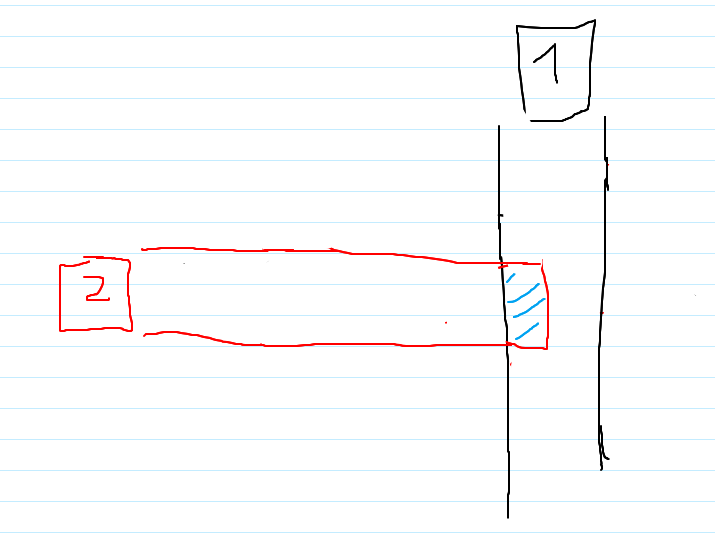
Volendo avere un raggio costante nel tempo e uguale per tutti i veicolo si valuterà in maniera conservativa il caso peggiore possibile, prelevando i valori massimi tra tutti i veicoli in gioco.

Si pensa anche di aggiungere per sicurezza un margine che nel caso peggiore mi faccia mantenere un’ulteriore eventuale distanza di qualche centimetro dai bordi altrui.

* **Traiettoria trasmessa**

Per come abbiamo definito il raggio, che in sé tiene già conto della quantità di informazione necessaria per la safety e dei tempi a cui deve al più essere condivisa, per il momento si pensa di trasmettere come pezzo di traiettoria quella percorsa nel tempo precedentemente descritto, ovvero in (stopTimeMax+2Tcmax+).

Tuttavia, una volta in fase simulativa potremo vedere se tale soluzione si rivelerà funzionale.

* **Sezioni critiche troncate**

Può verificarsi il caso in cui l’ultimo pezzo trasmesso appartenga ad una sezione critica. Se così fosse, il secondo veicolo vedrebbe solo una porzione della vera sezione critica comune e ne calcolerebbe quindi un tempo di uscita errato.

Anche per questa situazione si fa affidamento sugli aggiornamenti successivi, confidando che un’intersezione di tale tipo si verifica solo ai bordi del raggio di comunicazione, quindi a distanza sufficiente da aver tempo di correggere le previsioni negli intervalli successivi.

**MEMORIA degli STATI PRECEDENTI**

Al momento l’algoritmo pensato prevede di ricalcolare sezioni critiche e precedenze ad ogni iterazione. Si potrebbe anche pensare a possibili soluzioni in cui alcuni stati del passo precedente vengano sfruttati al passo successivo per limitare i calcoli. Bisognerà però vedere se ciò possa portare nei fatti un consistente vantaggio computazionale rispetto a un più banale ricalcolo.

**SUDDIVISIONE FILE**

*Si sviluppa un programma che in ambiente reale possa essere implementato su ogni singolo robot. Tale programma dovrà essere applicabile a robot differenti (aventi diverse velocità, priorità, tempi di calcolo) mantenendo comunque lo stesso comportamento generale.*

*Il programma sul singolo robot riceverà in ingresso le informazioni dei vicini e aggiornerà la traiettoria da passare poi al controllore.*

*L’unica cosa che al momento è rimasta centralizzata sono il calcolo del raggio di comunicazione e quello della traiettoria.*

*In ambiente simulativo, il programma sarà ovviamente unico, e ciascun robot sarà gestito da un thread distinto.*

*Si presenta un vaga prima organizzazione del codice in tre macro-pacchetti principali: il primo contenente le funzioni preponderanti, che caratterizzano l’intero scopo del progetto, descritte nei precedenti algoritmi 2 e 3; il secondo che descrive gli enti in gioco; il terzo che sfrutta i precedenti per test e visualizzazione.*

1. *Algoritmi*
   1. *CalcoloSC*
   2. *CalcoloPrecedenze*
2. *Attori (classi degli agenti fondamentali)*
   1. *Vehicle : variabili di stato del veicolo e programma principale (singolo Thread che esegue il suddetto Algoritmo 1)*
   2. *CriticalSection : informazioni della sezione critica (punto critico, precedenza…)*
3. *Visualizzazione*
   1. *BrowserVisualization*
   2. *Main*