**Ricapitolazione codice fin ora sviluppato - 17 dic 20**

Scopo principale è quello di distribuire l’algoritmo centralizzato per il coordinamento di flotte, in particolare andando a modificare il tempo di percorrenza su una data traiettoria per evitare collisioni tra robot in caso di intersezioni di path.

Il lavoro attualmente ha visto due fasi:

nella prima ci siamo concentrati a sviluppare un primo algoritmo distribuito in cui sono state sviluppate solo le funzionalità di base andando a omettere o semplificare funzionalità più complesse necessarie al corretto funzionamento dell’algoritmo finale.

Nella seconda parte, in cui ci troviamo adesso, stiamo andando ad implementare le restanti parti di algoritmo.

Fase 1

* Suddivisione in Thread: ogni robot dovrà poter eseguire il codice in maniera indipendente dal resto dei robot, pertanto è stato deciso di creare un thread per ognuno.
* Suddivisione robot in categorie: in un ambiente reale robot diversi avranno caratteristiche diverse, per simulare questo abbiamo creato due categorie (ambulanze e macchine) che differiscono per priorità, velocità e accelerazioni massime raggiungibili e per velocità di esecuzione dell’algoritmo.
* Definizione di un raggio di comunicazione uguale per tutti.
* Calcolo path data posa iniziale e posa di arrivo

Ciclo Thread

* Identificazione robot vicini: partendo dalla lista totale di robot vado a selezionare solo quelli all’interno del mio raggio di comunicazione
* Calcolo sezioni critiche: attraverso la conoscenza del path(completo) dei robot vicini vado a calcolarmi le sezioni critiche in un primo momento tali sezioni critiche non sono state ordinate
* Calcolo precedenze: in base alla priorità, vicinanza alla sc (se sono già dentro ho precedenza) calcolata solo la precedenza sulla prima sezione critica, precedenze in base alle pozioni e non al tempo
* Calcolo punto critico: il punto critico viene settato a -1 se ho precedenza, allora arriverò fino al goal, se non ho precedenza mi fermo all’inizio della sezione critica
* Aggiornamento posizione: Ad ogni ciclo, se non sono sul punto critico, avanzo di un pathIndex e aggiorno la posa.
* Plot: al momento solo scritte sul terminale

In questa prima fase sono stati quindi sviluppati i comportamenti base del robot, nella seconda fase andremo ad affinare tali comportamenti.

Fase 2

Obiettivi:

* Troncamento Path trasmesso
* Ordinamento Cs e calcolo precedenze in base a tutte le cs future
* Moto del veicolo seguendo la legge di moto
* Identificazione differenziazione fra punto critico, punto in cui posso fermarmi, punto in cui devo iniziare a rallentare
* Associare ad ogni path index un tempo

Implementazione:

* Abbiamo troncato il path in base a velocità massima e raggio di trasmissione
* Andiamo a calcolari lo spazialEnvelope e quindi la traiettoria futura solo all’interno del raggio, (ancora da inglobare la consocenza dei tempi effettivi di percorrenza)
* Ordinamento Cs: in base alla loro posizione lungo il mio path
* Calcolo precedenze: non mi fermo alla prima sezione critica ma continuo a valutare le precedenze finchè risulto avere precedenza, in questo modo se ho precedenza sulla prima cs ma non sulla secondo potò calcolarmi direttamente il punto critico sulla seconda cs.
* Moto del veicolo

Siam andati ad integrare ConstantAccForwardModel e TrakerRK4

* + Integrate RK4 – aggiornamento stato, damping = 1 ?
  + getEarliestStoppingPoint – dato uno stato calcolo dove mi posso fermare
  + CanStop – dato un path index di partenza e uno di arrivo calcolo se posso fermarmi in tempo
  + Compute Distance – dati due path index ne calcola la distanza
  + GetPath Index dato lo stato e la traiettoria restituisce il path a cui mi trovo
  + UpdateState – decide se e come integrare

Dato il punto critico attraverso una funzione ricorsiva che sfrutta can stop calcolo il punto in cui devo iniziare a frenare (SlowingPoint) viene calcolato considerando la Vmax e AccMax

Aggiorno lo stato: se sono al critical point non integro e quindi non aggiorno lo stato, se sono prima dello slowingPoint procedo accelerando fino a Vmax, Se sono dopo lo slowingPoint decelero.

Si è reso necessario inserire un saturatore per evitare velocità negative, e impostare una velocità minima nel caso in cui io mi trovi fra slowingPoint e CriticalPoint(-1) questo per evitare di fermarmi troppo presto prima del CriticalPoint.

Identifico nuovo PathIndex in base al mio stato

Aggiorno la posa

Calcolo punto in cui potrei fermarmi – (lo vogliamo trasmettere agli altri robot insieme al path e servirà per fare precedenze più accurate)

Calcolo tempi della traiettoria – computeTs (modifica di computeDTS)

Creo hasmap(pathindex,tempo) il tempo corrisponde al tempo che ci metto ad arrivare a quel punto partendo dal mio stato attuale

Dato il mio stato attuale e il punto in cui devo iniziare a rallentare vado a calcolarmi i tempi dei miei futuri path index fiino a che la mia velocità non è pari a zero.

Potenzialmente posso calcolarmi i tempi su tutta la traiettoria

Dobbiamo ancora:

* Migliorare criterio di arresto
  + Path completo o troncato
  + Solo fin dove mi fermo o anche consideranto la ripartenza da una cs dopo che l’altro robot l’ha sorpassata (info ricavabile dalla traiettoria passata del secondo robot)
* Integrare queste informazioni ottenute all’interno del codice per:
  + Trasmettere traiettoria troncata in base ai tempi di percorrenza
  + Calcolarmi le precedenze in base ai tempi
  + Quindi inserire tempi nelle info di una cs

Lavori futuri

* Inglobare informazioni ottenute dal calcolo dei tempi
* Riuscire a fare un plot
* Considerare DeadLock e corridoi
* Considerare possibile ripianificazione

DOMANDE

* Damping
* CanStop o EarliestPoint
* LookingHaedinMillisec
* Saturazione velocità negative e velocità minima
* Miglioramenti posizione sleep
* Troppe Modifiche ?