POLITECNICO DI TORINO

Corso di Laurea Magistrale

in Ingegneria Meccanica

Tesi di Laurea Magistrale

Studio di algoritmi e co-simulazione per la guida autonoma di veicoli da lavoro

Relatori
Prof. Aurelio Somà
Dott. Francesco Mocera

Candidata
Giorgia Vavoli

Anno Accademico 2018/2019

Indice generale

Elenco delle figure

Indice delle tabelle

ABSTRACT

Il presente lavoro si colloca nell'ambito riguardante la simulazione e la sperimentazione sulla guida autonoma con l'obbiettivo di riprodurre il comportamento di un veicolo da lavoro in grado di svolgere attività in modo autonomo o semi-autonomo. Nel caso in esame si vuole far muovere il veicolo dalla sua posizione di partenza fino ad un determinato punto di arrivo, evitando collisioni con qualsivoglia tipo di ostacolo.

Per riprodurre tale comportamento, è stato necessario fare ricorso all'utilizzo di due software differenti, MATLAB & Simulink e ADAMS View, l'uno finalizzato all'implementazione della logica di controllo e l'altro alla simulazione dinamica del veicolo e della sensoristica a bordo. Si è partiti dunque da un'analisi degli algoritmi path-planning esistenti, cercando di individuare una soluzione confacente per la nostra applicazione senza eccessivo onere computazionale. Una volta definita la traiettoria, si è passati alla definizione di un secondo livello di controllo riguardante l'esecuzione della stessa (path-following) e la strategia di obstacle-avoidance, dal quale sono stati poi ricavati i comandi in input nel modello ADAMS del veicolo.

ABSTRACT

The thesis activity concerns the simulation and experimentation of autonomous driving, aimed to reproduce the behavior of a work vehicle able to carry out its activity in autonomous or semi-autonomous way. In this case the vehicle has to move from its starting position to a certain point of arrival, avoiding collisions with any type of obstacle.

In order to reproduce this behavior, it was necessary the use of two different software, MATLAB & Simulink and ADAMS View; the first was useful for the implementation of the control logic, the latter for the dynamic simulation of the vehicle and on-board sensors. Therefore, the existing path-planning algorithms have been studied, trying to identify a suitable solution for our application without excessive computational burden. Once the trajectory was obtained, a second control level was defined, concerning its execution (path-following) and the obstacle-avoidance strategy. Hence, the input commands were obtained in the ADAMS model of the vehicle.

Capitolo 1

1.1 Introduzione

Negli ultimi decenni lo sviluppo industriale ha messo in luce uno scenario in rapida evoluzione, sempre più spinto verso la cosiddetta industria 4.0 e basato sull'introduzione di nuove tecnologie riguardanti l'informazione, la connessione, la programmazione e l'automazione. Tali innovazioni consentono di beneficiare di una serie di vantaggi, tra i quali la possibilità di realizzare un flusso logistico ben definito e con un percorso fisso, modelli di produzione sempre più automatizzati e interconnessi e una maggiore flessibilità, il tutto razionalizzando i costi e ottimizzando le prestazioni. Con particolare occhio di riguardo all'automazione, si può affermare che dal punto di vista storico è nata con il principale scopo di sostituire l'uomo in compiti ripetitivi o nocivi, con apparecchiature in grado di operare in modo autonomo o con minimi interventi da parte dell'operatore umano. In tale ottica rientrano ad esempio i veicoli a guida automatica (AGV) senza operatore, o in alcuni modelli con la possibilità di condividerne la guida con operatore a bordo; il loro impiego comporta notevoli vantaggi, tra cui la puntualità nell'esecuzione e nessun errore di destinazione, drastica riduzione dei rischi dovuti alla movimentazione delle merci, facilità di riconfigurazione percorsi o inserimento di nuove applicazioni. D'altra parte quello della guida autonoma risulta essere un tema di stretta attualità, in quanto oggetto di studio e sperimentazione finalizzati non soltanto all'ambito industriale, ma, ad esempio, anche ai settori dell'automotive e dei trasporti.

Figura 1: "Navetta trasportatrice AGV"

Rispetto al caso della navetta AGV ad esempio, appare subito evidente come veicolo in questione si troverebbe all'interno di un ambiente di lavoro del tutto differente: si potrebbe far riferimento ad un escavatore piuttosto che ad un veicolo telescopico, come quelli raffigurati nelle immagini sottostanti.

Figura 2: "Mezzo escavatore"

Figura 3: "Veicolo telescopico"

In particolare, immaginando di dover far muovere il mezzo dalla posizione di partenza fino ad un determinato punto di arrivo all'interno di un campo agricolo, si evince che non si tratta di un luogo ben strutturato e con percorsi e operazioni prestabiliti e ripetitivi, bensì di un ambiente 'out-door', seppure confinato, ma caratterizzato dalla presenza di ostacoli di diverso tipo e talvolta anche non previsti. Dunque, per poter far fronte a simili problematiche, riguardanti ad esempio la presenza di una buca, di un dosso, o particolari condizioni atmosferiche, il mezzo dovrà essere dotato di un'adeguata sensoristica e di un'opportuna logica di controllo che gli consenta di pianificare le migliori traiettorie possibili e allo stesso tempo gli permetta di modificare il proprio percorso in tempo reale, sulla base delle informazioni provenienti dai sensori.

1.2 Pianificazione della traiettoria

Una delle prime tematiche da affrontare riguarda dunque la fase di *path planning*, ossia pianificazione della traiettoria. Per un veicolo a guida autonoma infatti è essenziale individuare il percorso migliore per raggiungere un dato obbiettivo, in tempi ragionevoli e con velocità opportune, assicurandosi di evitare eventuali ostacoli. Affinché ciò sia possibile il veicolo deve essere dotato in qualche modo di una certa abilità di *reasoning*, ossia che sappia individuare quali azioni sono richieste per raggiungere un certo obbiettivo in una data situazione; tali decisioni possono riguardare il percorso da intraprendere e l'utilizzo di determinate informazioni recepite dall'ambiente. In base a ciò si può affermare che un robot industriale opera invece 'senza cognizione', poiché lavora in ambiente statico e strutturato.

D'altra parte nei robot mobili la fase di *reasoning* riguarda principalmente il percorso migliore da intraprendere per raggiungere una data posizione.

Sono spesso richieste soluzioni ibride, che sfruttano in parallelo una strategia globale di path-planning ed una strategia locale di obstacle-avoidance.
A tal fine occorre pertanto conoscere la geometria dell'ambiente di lavoro, degli ostacoli e del robot, la sua cinematica (gradi di libertà), la posizione di partenza e quella finale. Forniti tali input, tramite un opportuno algoritmo di pianificazione si ricava la traiettoria desiderata come una sequenza continua di configurazioni collision-free tra la posizione iniziale e quella obbiettivo. L'insieme delle configurazioni raggiungibili è detto *spazio delle configurazioni (configuration space)*, mentre nell'ambiente circostante è possibile distinguere lo spazio libero dagli ostacoli, ossia aree già occupate che il veicolo non può raggiungere.

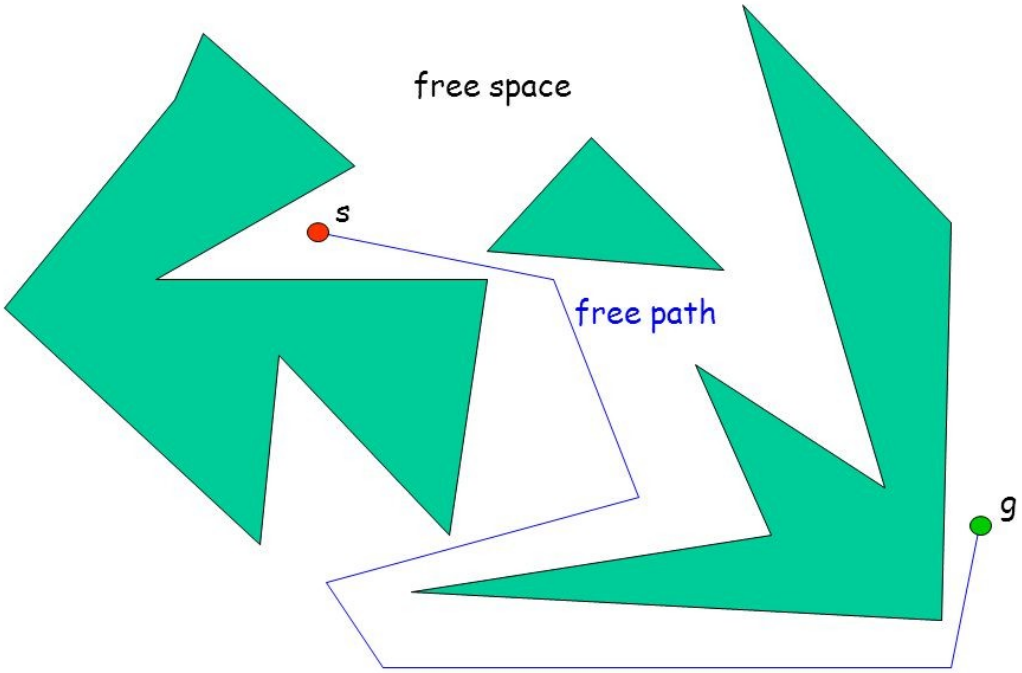


Figura 4: "Problema generico di path planning"

Esistono poi diverse tecniche di pianificazione, potendo distinguere in generale due diversi approcci, uno locale e l'altro globale. I problemi di *local planning* non cercano di risolvere il problema nella sua generalità, ma sfruttano esclusivamente le informazioni disponibili al robot per determinare il comando di movimento successivo; essi riguardano l'area nelle immediate vicinanze del veicolo. I metodi di *global planning* invece implicano una conoscenza completa dell'ambiente, e in questi casi è spesso conveniente ridurre le dimensioni del robot ad un punto aumentando quelle degli ostacoli per ottenere spazio libero.

Tra le varie tecniche di path planning esistenti invece è opportuno mettere in evidenza quantomeno le principali, da cui tutt'ora si cerca di sviluppare e sperimentare soluzioni alternative sempre migliori. La prima fra queste è denominata 'cell decomposition' e consiste nella decomposizione dello spazio libero 'F' in celle elementari: è considerata 'esatta' se F è rappresentato da una serie di celle non sovrapposte la cui unione è esattamente F; viceversa, si parla di decomposizione 'approssimata' quando F è

rappresentato da una serie di celle non sovrapposte la cui unione è contenuta in F. Una volta aver ottenuto la decomposizione dello spazio delle configurazioni, viene creato un grafo di connettività che rappresenta le relazioni di adiacenza tra le celle; all'interno di tale grafo viene poi individuata una sequenza di celle vuote, chiamata 'canale', la quale connette la cella contenente la configurazione iniziale con la cella contente la configurazione obbiettivo. Le strategie di pianificazione basate sulla tecnica di decomposizione in celle sono in genere limitate dal volume dello spazio di configurazione; tale volume cresce esponenzialmente con i gradi di libertà del veicolo[1].

Un ulteriore procedimento fa riferimento invece alla definizione di campi potenziali ('potential fields'): in questi casi si definisce una funzione sullo spazio libero che ha un minimo globale nella configurazione obbiettivo e l'algoritmo segue il gradiente negativo della funzione verso la posizione finale. Il robot viene quindi trattato come un punto sotto l'influenza di campo potenziale artificiale: l'obbiettivo genera una forza attrattiva, mentre gli ostacoli danno origine a forze repulsive. Solitamente i metodi con i campi potenziali sono applicabili unicamente a veicoli con limitati gradi di libertà; inoltre questa classe di algoritmi è particolarmente sensibile ai minimi locali, tenendo presente inoltre che in alcuni casi può risultare complicato determinare un espressione per il potenziale, oppure le soluzioni trovate risultano ottime solo localmente.

L'ultima macro-distinzione tra le differenti strategie si riferisce infine a quelle basate sulla creazione di una cosiddetta 'roadmap': essa rappresenta la connettività dello spazio libero attraverso una rete di curve 1-D. Anche in questo caso il robot viene ridotto ad un punto e tale metodo risulta particolarmente utile quando si considerano veicoli soggetti a vincoli *cinodinamici,* ossia quando sono presenti sia vincoli cinematici che vincoli dinamici.

In seguito, verranno affrontati anche gli algoritmi pratici tramite i quali tali tecniche di pianificazione possono essere messe in atto, effettuando una breve disamina sulle caratteristiche generali che li contraddistinguono e ponendo maggiore attenzione su quelli individuati come più confacenti al caso in esame.

1.3 Path-following e obstacle-avoidance

Una volta aver definito la traiettoria ideale, il passo successivo consiste nell'elaborare una opportuna logica di controllo che consenta al veicolo di potersi muovere lungo tale percorso, nella maniera più precisa possibile. Tale problema, che prende il nome di *path-following*, può essere dunque ricondotto alla determinazione dei comandi sulla velocità e sulla direzione da imporre al robot, compatibilmente ai suoi limiti cinematici e dinamici. Una possibilità è quella ad esempio di tradurre il segnale di velocità in un comando di coppia da impartire alle ruote del veicolo, oppure sfruttare un modello di guida differenziale dando due differenti velocità alle ruote dello stesso assale per poter imprimere la direzione desiderata.

In ogni caso la fase di path following è con ogni probabilità quella maggiormente critica, poiché risulta determinante negli sviluppi della simulazione dinamica vera e propria. Ai fini della nostra applicazione è inoltre fondamentale elaborare una strategia di obstacle avoidance che operi in parallelo e in tempo reale con il path following. Il suo obbiettivo non è solo quello di evitare la presenza lungo il cammino di ostacoli 'fisici', ma di permettere anche al veicolo di gestire in qualche modo la presenza di eventuali avvallamenti o dossi, pozzanghere o qualsivoglia irregolarità del terreno che originariamente non erano previsti. La logica di controllo dovrà quindi regolare velocità e direzione anche in funzione delle considerazioni appena fatte; la strategia di obstacle avoidance in particolare, si basa su mappe locali e segue la regola generale di aumentare le dimensioni dell'ostacolo lungo una certa dimensione, a seconda di come è caratterizzata la geometria del veicolo. La sua efficacia viene valutata in funzione dell'obbiettivo finale, della velocità e della cinematica del robot, della sensoristica on-board.

Le possibilità di scelta per quel che riguarda gli algoritmi di implementazione sono molteplici anche in questo caso, e alcuni si basano sostanzialmente sugli stessi principi di riferimento per gli algoritmi path planning. Il cosiddetto *'Bug algorithm'*[2] è concettualmente uno dei più semplici: quando il robot incontra l'ostacolo lo aggira completamente seguendone il 'contorno' e cercando il punto che si trova alla minima distanza dal target; dopodiché, completato il giro attorno l'ostacolo, si riposiziona in tale

punto e abbandona l'ostacolo seguendo il percorso calcolato verso il target. Secondo un'altra variante dell'algoritmo invece, il robot comincia a seguire il contorno dell'ostacolo, ma poi lo abbandona non appena incontra la linea congiungente il punto iniziale a quello finale, ossia non appena si ritrova lungo lo stesso percorso iniziale. A fronte della sua semplicità tale algoritmo presenta però alcuni svantaggi:

- esso non considera la cinematica del robot;
- tiene conto solamente le letture dei sensori più recenti, quindi le performance possono risentire del rumore;
- è un algoritmo lento.

Mentre gli algoritmi di bug sono basati su un approccio puramente reattivo, il *'potential field algorithm'* tende a considerare l'obstacle avoidance come un sotto-processo del path-planning. Si assume che il robot venga guidato da forze virtuali che lo attraggono verso il target o lo respingono dagli ostacoli; il percorso finale deriva dalla risultante di tali forze virtuali. Tuttavia questo algoritmo:

- è poco performante in passaggi stretti;
- nel caso di ostacoli disposti simmetricamente rispetto al robot, se la forza attrattiva del target eguaglia quella repulsiva data dalla somma vettoriale delle forza degli ostacoli, il robot si ferma (minimo locale);
- è difficile da usare in applicazioni real-time.

Tra quelli più frequentemente utilizzati rientra sicuramente il *'vector field histogram (VFH) algorithm'*: esso supera le problematiche connesse al rumore del sensore creando un istogramma polare in base a numerose letture. Sulle ascisse si riportano gli angoli associati alle letture del sonar, sulle ordinate la probabilità che esista effettivamente un ostacolo in quella direzione. Le probabilità vengono determinate creando una mappa reticolata di occupazione locale dell'ambiente circostante il robot; l'istogramma polare viene utilizzato per identificare tutti i passaggi abbastanza larghi per consentire al veicolo di muoversi. Per ciascun passaggio viene definita una cost-function e in base al suo valore viene scelto il percorso da seguire da parte del veicolo. Ciò dipende dall'allineamento del cammino del mezzo rispetto al target e dalla differenza tra

l’orientazione delle ruote e la nuova direzione; si sceglie il percorso che comporta il minimo costo.

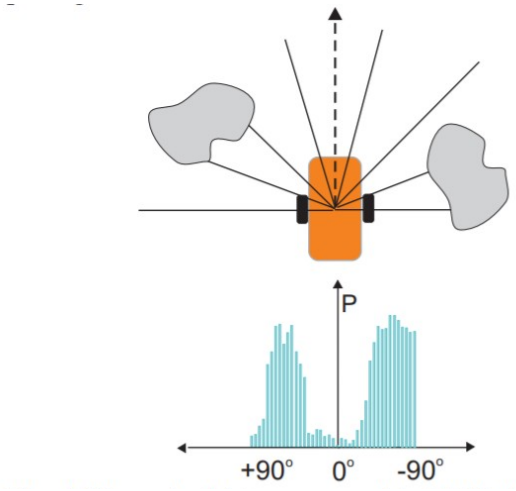


Figura 5: "Istogramma polare utilizzato nell'algoritmo VFH"

A valle di tali considerazioni si può affermare che l’algoritmo VFH

- offre migliore ‘resistenza’ al rumore del sensore;
- tiene conto della cinematica del robot;
- ma comporta un onere computazionale considerevole.

Esso viene elaborato in 3 step: nel primo viene generato un istogramma 2D attorno al robot che rappresenta gli ostacoli, e che viene poi aggiornato con le informazioni provenienti dai sensori. Nel secondo step, tale istogramma 2D viene convertito in un istogramma 1D e successivamente nell’istogramma polare. Infine, come ultimo step l’algoritmo seleziona il settore con la minore densità polare di ostacoli e determina l’angolo di sterzo e la velocità in quella direzione.

L’ultimo algoritmo esposto in questa breve disamina tra i principali comunemente utilizzati, è alla base anche della logica di controllo sviluppata per la nostra applicazione. Ricapitolando, essa prevede per l’appunto un funzionamento in parallelo

tra le strategie di path following e obstale avoidance, nota che sia la traiettoria desiderata elaborata nella fase precedente di path plannig.

1.4 Metodi di simulazione e co-simulazione

A questo punto non resta che definire in che modo effettuare la simulazione del problema. Innanzitutto è stato scelto il software ritenuto più consono ad affrontare la questione in esame e in tal senso è stato fatto ricorso all'utilizzo di MATLAB & Simulink. Per illustrare con maggiore dettaglio quanto è stato fatto, è utile individuare i passaggi fondamentali che hanno caratterizzato il lavoro svolto:

1. Fase di path planning;
2. Sviluppo della logica di controllo: path following e obstacle avoidance;
3. Simulazione dinamica.

Per il primo punto è stato fatto affidamento interamente a MATLAB: qui è stato implementato l'algoritmo tramite il quale, nota a priori la geometria dell'ambiente di lavoro, la posizione di partenza e quella obbiettivo, è stato ricavato il cammino ideale da inseguire.
Dopodiché, tale percorso costituisce uno degli input della fase 2, sotto forma di *waypoints* nel piano, ossia di una serie di punti ciascuno contraddistinto da una posizione (x,y) e un'orientazione θ.
Il fine della logica di controllo è quello di determinare i comandi di velocità e sterzata da impartire al veicolo, che quindi costituiscono gli ingressi della fase di simulazione dinamica. Appare evidente come, per il sistema così descritto, sia necessario un feedback sulla posizione corrente del robot che istante per istante sarà influenzata dalla logica di controllo. Ciò allo stesso tempo permetterà il funzionamento in tempo reale, correggendo continuamente i segnali di comando nel tentativo di seguire la traiettoria desiderata ed evitare gli ostacoli: il tutto costituisce quella che all'inizio del capitolo era stata definita 'abilità di *reasoning*'. In sostanza è come se il mezzo fosse dotato di opportuni sensori che forniscano informazioni di questo tipo e consentano appunto di realizzare la chiusura dell'anello tra la fase 2 e la fase 3.

Se però in un primo momento si era pensato di realizzare entrambi i passaggi finali in ambiente Simulink, successivamente si è optato per una co-simulazione con il software ADAMS View, data la sua maggiore semplicità con cui poter elaborare il modello dinamico del veicolo e simulare dei ‘sensori virtuali’ con cui ricavare i parametri di feedback. D’altra parte la co-simulazione mette in luce un aspetto non banale: affinché l’intero modello funzioni è infatti necessario che i parametri sul tempo di integrazione e di comunicazione tra i due software siano compatibili.

Capitolo 2

2.1 Definizione dei principali metodi risolutivi del path planning

2.1.1 Caratteristiche generali di un algoritmo

In questo capitolo verrà esposto nel dettaglio il metodo con cui è stata pianificata la traiettoria e quali sono state le considerazioni che hanno portato alla scelta dell'algoritmo utilizzato.
E' conveniente specificare che a seconda di come sono strutturati, si possono individuare varie tipologie di algoritmi. In primo luogo un algoritmo di planning si definisce 'completo' se trova una soluzione (esistente); se la soluzione non esiste riporta *fail*. Nel caso in cui, verificatasi quest'ultima eventualità, l'algoritmo continua a lavorare all'infinito, esso viene detto invece 'semi-completo'. Se piuttosto la sua azione termina e riporta che non esiste una soluzione entro una specifica risoluzione, l'algoritmo è 'a risoluzione completa'. Infine la casistica comprende quelli 'probabilisticamente completi': se la soluzione esiste, la probabilità di trovarla tende ad uno se il numero di iterazione tende ad infinito. Di qui più in generale le differenti tipologie riguardano:

- gli algoritmi teorici, i quali cercano la completezza e la minima complessità nel caso peggiore; sono difficili da implementare e poco robusti.
- Gli algoritmi euristici, che mirano all'efficienza in situazioni comunemente incontrate. Solitamente non garantiscono però un'adeguata performance.
- I pianificatori di movimento completo (*complete motion planner*), i quali forniscono sempre una soluzione se essa esiste e indicano che tale soluzione non esiste altrimenti.

In riferimento a quanto è stato appena detto, di seguito si espone una trattazione riguardante quelli che sono i casi applicativi più comuni, con fine puramente esegetico.

2.1.2 Algoritmo di Dijkstra

E' un algoritmo ciclico utilizzato per cercare i cammini minimi in un grafo con o senza ordinamento; il cammino minimo è il percorso che permette di unire due nodi distinti del grafo.
L'algoritmo di Dijkstra[3] visita i nodi nel grafo, in maniera simile a una ricerca in ampiezza o in profondità. In ogni istante, l'insieme N dei nodi del grafo è diviso in tre parti: l'insieme dei nodi visitati V, l'insieme dei nodi di frontiera F, che sono successori dei nodi visitati, e i nodi sconosciuti, che sono ancora da esaminare. Per ogni nodo z, l'algoritmo tiene traccia di un valore d_z, inizialmente posto uguale a ∞, e di un nodo u_z, inizialmente indefinito. L'algoritmo consiste semplicemente nel ripetere il seguente passo: si prende dall'insieme F un qualunque nodo z con d_z minimo, si sposta z da F in V, si spostano tutti i successori di z sconosciuti in F, e per ogni successore w di z si aggiornano i valori d_w e u_w. L'aggiornamento viene effettuato con la regola

$$d_w \leftarrow \min\{d_w, d_z + p_a\},$$

dove a è l'arco che collega z a w. Se il valore di d_w è stato effettivamente modificato, allora u_w viene posto uguale z.
La regola segue un'idea piuttosto naturale: se sappiamo che con peso d_z possiamo arrivare fino a z, allora arrivare a w non può costare più di arrivare a z e spostarsi lungo un arco fino a w. L'aggiornamento di u_w ci permette di ricordare che, al momento, il cammino di peso minimo che conosciamo per arrivare da x in w ha come penultimo nodo z. L'algoritmo parte con V=∅, F={x}, dx=0 e prosegue finché y non viene visitato, o finché F=∅: in questo caso, y non è raggiungibile da x lungo un arco orientato. Se usciamo solo nel secondo caso, alla fine dell'algoritmo d_z contiene, per ogni nodo z, il peso di un cammino minimo da x a z; inoltre, il vettore u permette di ricostruire l'albero dei cammini minimi con origine in x.

2.1.3 Algoritmo A*

E' un algoritmo di ricerca su grafi che individua un percorso da un dato nodo iniziale verso un dato nodo goal. Utilizza una 'stima euristica' che classifica ogni nodo attraverso un calcolo del persorso migliore che passa attraverso tale nodo (è un esempio di ricerca best-first[4]). In generale, A* può risolvere efficacemente i problemi che soddisfano i seguenti requisiti:

- la soluzione è determinata da cambiamenti sequenziali di stato rappresentabili con grafi;
- il nodo iniziale e il nodo finale devono essere noti. Talvolta è sufficiente conoscere solo le regole che compongono la soluzione;
- deve essere noto un algoritmo euristico che stima il costo del percorso tra un nodo qualsiasi e la soluzione;
- deve essere sempre noto il costo associato a due nodi adiacenti (nella maggior parte dei problemi tale valore è sempre unitario).

L'algoritmo euristico ha il compito di stimare la distanza tra qualsiasi nodo e la soluzione. Esso, in particolare, ne determina il tempo complessivo di esecuzione. Nel caso peggiore, una funzione euristica costante, A* diviene un algoritmo di ricerca molto simile a quello di Dijkstra.

L'euristica determina anche la qualità della soluzione finale: con un'euristica ammissibile, A* è in grado di identificare la soluzione ottima (ad esempio il percorso con il minor costo possibile). Ciò si verifica quando l'errore di stima non è mai in eccesso; un esempio è la distanza in linea d'aria tra due punti su una mappa. In termini matematici una funzione euristica h è ammissibile se[5]:

$$\forall x \in V : h(s,x) \leq g(s,x)$$

dove V è l'insieme dei nodi, s è il nodo soluzione e la funzione g calcola la distanza esatta tra due nodi. Una funzione euristica monotona semplifica ulteriormente la

struttura di A* in quanto la lista dei nodi già visitati diviene superflua; una funzione euristica monotona è sempre ammissibile.
A* rientra nella categoria degli algoritmi di ricerca best-first. Esso infatti esamina, passo dopo passo, i nodi che hanno il punteggio migliore. Esso tuttavia non è *greedy* in quanto il punteggio non è determinato esclusivamente dall'euristica.

A* usa le seguenti strutture dati per mantenere traccia dello stato d'esecuzione:

- una lista di nodi già visitati;
- una coda a priorità contenente i nodi da visitare.

Nel corso dell'esecuzione, ad ogni nodo vengono associati più valori: *gScore*, *hScore*, *fScore*. In termini matematici, dato il nodo corrente *n*, il nodo di partenza *p* e il nodo soluzione *s*, si definiscono i valori:

$$gScore = g(p, n)$$

$$hScore = h(s, n)$$

$$fScore = gScore + hScore$$

La funzione *g* calcola il costo effettivo del percorso che separa i nodi *p* (partenza) e *n* (attuale). La funzione *h* calcola una stima della distanza del percorso tra i nodi *s* (soluzione) e *n* (attuale). La funzione *h* corrisponde alla definizione dell'algoritmo euristico enunciato in precedenza. Essa è infatti chiamata spesso funzione euristica. La ricerca è guidata da una funzione di valutazione complessiva *f*.
La struttura dell'algoritmo A* è molto semplice. Esso, ad alto livello, può essere schematizzato in 8 passi:

1. Inserimento nella coda del nodo di partenza con priorità pari al *fScore*.
2. Se la coda è vuota, l'algoritmo termina: soluzione non trovata;
3. Estrazione del miglior nodo da visitare (priorità con valore più basso);
4. Se il nodo estratto ha *hScore* nullo, l'algoritmo termina: soluzione trovata;
5. Costruzione dei nodi figli;
6. Eliminazione dei nodi figli già visitati e sub-ottimi;
7. Inserimento dei nodi rimanenti nella coda con priorità pari al *fScore*;
8. Tornare al punto 2.

A* è un algoritmo semplice ma dalle grandi potenzialità; esso getta le basi per ulteriori metodologie di ricerca più complesse come IDA* e D*. La sua principale limitazione è nell'assenza di vincoli sulla profondità di ricerca. Bisogna sempre tener presente che la complessità di un algoritmo è strettamente legata alle dimensioni dello spazio delle configurazioni in cui si trova a dover lavorare.

2.2 Sampling-based motion planning

Qualora la dimensione dello spazio delle configurazioni fosse troppo elevata e non potesse essere affrontata per via analitica o tramite discretizzazione, vi è la necessità di utilizzare tecniche differenti. In particolare la categoria degli algoritmi *sampling-based motion planning* rappresentano allo stato attuale l'insieme di tecniche maggiormente utilizzate (essendo quest'ultime decisamente meno dispendiose in termini computazionali) . Queste tecniche non prevedono la costruzione esplicita del C_{obs} (insieme delle configurazioni in collisione con l'ambiente) come avveniva nei casi precedenti, ma piuttosto cercano di esplorare lo spazio delle configurazioni generando un numero fissato di nodi attraverso tecniche di campionamento e costruendo quindi una rete di nodi e rami detta *Roadmap* (dunque non prevedono la conoscenza dell'intero spazio delle configurazioni come accade per le tecniche precedenti). La probabilità con cui questi nodi vengono generati è uniforme; per questo, fissando un adeguato numero di iterazioni, si può assumere di coprire gran parte dello spazio libero. A valle del campionamento viene poi valutato da un modulo esterno se la configurazione generata è una configurazione in collisione oppure no.
Tutti questi algoritmi si strutturano fondamentalmente in due fasi: la prima detta, *Learning phase*, consiste nella creazione della Roadmap generando casualmente un numero finito di nodi in C_{free} e connettendoli tramite archi collision-free secondo logiche che variano in base all'algoritmo particolare. Nella seconda, chiamata *Quering phase*, l'algoritmo calcola il cammino ottimo a partire da un nodo start e un nodo goal all'interno della Roadmap. I due più grandi algoritmi di questa categoria, gli algoritmi PRM (*Probabilistic Road Maps*) e gli algoritmi RRT (*Rapidly-Exploring Random Trees*)

si differenziano tra loro in base alla tecnica con cui viene costruita la traiettoria finale (si differenziano nella fase di Learning)[6].

2.2.1 Probabilistic Road Map (PRM)

Gli algoritmi di tipo PRM effettuano a monte della pianificazione vera e propria N campionamenti dello spazio delle configurazioni e altrettanti test di collisione; ogni configurazione generata viene collegata alle configurazioni adiacenti tramite algoritmi di pianificazione discreta come quelli presentati in precedenza. In questo modo al termine di questa fase (learning phase), si ha a disposizione una mappatura completa dello spazio delle configurazioni e una serie di percorsi che lo attraversano. Nella seconda fase (quering phase) le configurazioni iniziale e finale vengono aggiunte allo spazio delle configurazioni e collegate alla rete di percorsi costruita nella fase precedente. In questo modo la pianificazione consisterà solamente nella ricostruzione del percorso che unisce partenza e destinazione. Questi algoritmi ben si adattano a richieste multiple di pianificazione: infatti una volta costruita la rete di traiettorie nella prima fase, la pianificazione sarà velocissima.

a) Fase di Learning

In questa fase si genera la Roadmap, ovvero la rete di nodi e archi in cui i nodi rappresentano le diverse configurazioni attuabili dal robot e appartenenti allo spazio libero C_{free}, mentre gli archi sono le traiettorie realizzabili che li connettono.
Le connessioni tra i nodi sono calcolate da un pianificatore locale, ovvero un algoritmo specifico che, in base alle diverse esigenze e casistiche, calcola la sequenza di nodi intermedi e ne verifica l'appartenenza allo spazio libero C_{free}.
La Roadmap essenzialmente è un grafo G = (V, E) in cui V rappresenta l'insieme dei nodi (o vertici) e E l'insieme degli archi.
Per prima cosa l'algoritmo prevede la generazione di una configurazione casuale chiamata q_{Rand} all'interno di C_{free}. Il nodo verrà quindi aggiunto all'insieme V inizialmente vuoto.

A questo punto, l'algoritmo calcola l'insieme dei nodi appartenenti a V più vicini a q_{Rand} selezionandoli in base a diverse strategie, che possono essere dettate dalla distanza euclidea, dalla minimizzazione di un'opportuna cifra di merito o da un numero predefinito di nodi da selezionare.
A partire dal nodo meno distante, l'algoritmo connette tutti i nodi del suddetto insieme a q_{Rand} solo nel caso in cui siano verificate le seguenti due condizioni: il pianificatore locale dichiara libera da collisioni la traiettoria dei due nodi e i due nodi non appartengono alla stessa componente connessa.
L'arco corrispondente alla connessione dei due nodi verrà aggiunto all'insieme E e questo implica l'unione di due differenti componenti connesse della Roadmap.
Quello che accade dunque è la nascita di un'unica componente connessa risultante dall'unione delle due.

b) Fase di Quering

In questa fase vengono aggiunti il nodo q_{start} e q_{goal} all'insieme V dei nodi della RoadMap. Come veniva fatto per un qualsiasi nodo q_{rand}, essi verranno quindi connessi ai nodi più vicini di V. A questo punto si potrà trovare la traiettoria che connette lo stato iniziale a quello finale solo nel caso in cui i due nodi q_{start} e q_{goal} appartengano alla stessa componente connessa. Nel caso specifico dei codici implementati per questo lavoro, si è deciso di annettere q_{start} all'insieme V già dalla prima iterazione. Per quanto riguarda lo stato finale invece, non si definisce un unico nodo q_{goal}, ma una regione all'interno dello spazio di configurazione composta da nodi.
Si può quindi dire che l'algoritmo ha trovato una soluzione al problema di pianificazione nel caso in cui riesca a connettere il nodo q_{start} ad un qualsiasi nodo appartenente alla Regione di Goal.
La situazione ottimale sarebbe quella di terminare la prima fase avendo ottenuto una Roadmap composta da un'unica componente, in questo modo tutti i nodi sarebbero raggiungibili da un qualunque nodo iniziale appartenente a V.

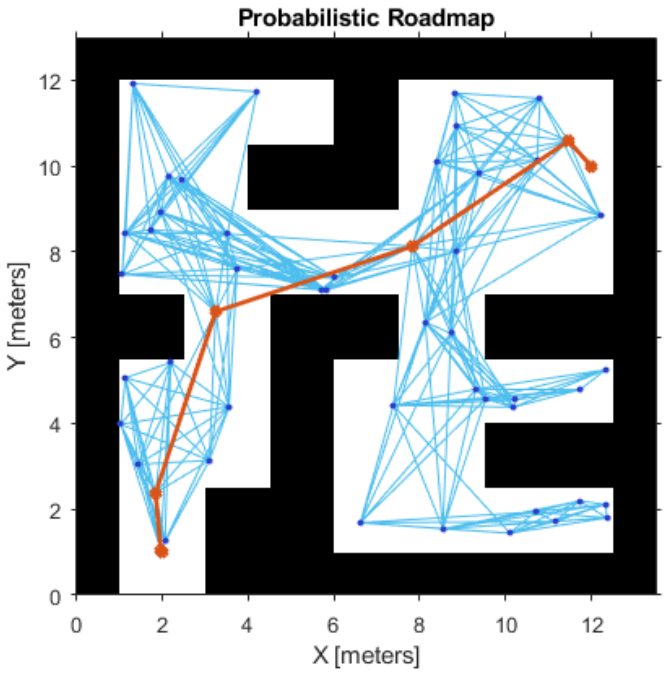


Figura 6: "Esempio delle possibili traiettorie calcolate dall'algoritmo PRM"

2.2.2 Rapdly-Exploring Random Trees (RRTs)

Gli algoritmi RRT invece si comportano in maniera leggermente differente. Non esiste infatti la fase di costruzione della rete di traiettorie. Le configurazioni di partenza e destinazione vengono piuttosto inserite subito all'interno dello spazio delle configurazioni e tramite la generazione di campioni e di traiettorie discrete viene ricercata la costruzione di un percorso che, passando dai campioni generati, unisca la configurazione iniziale e finale. Viene quindi espanso progressivamente un albero, composto da nodi e archi, a partire dal nodo radice q_{start}. Ad ogni iterazione viene generato un nodo q_{rand} con probabilità uniforme in C_{free} il quale indica la direzione verso cui l'albero si dovrà espandere.

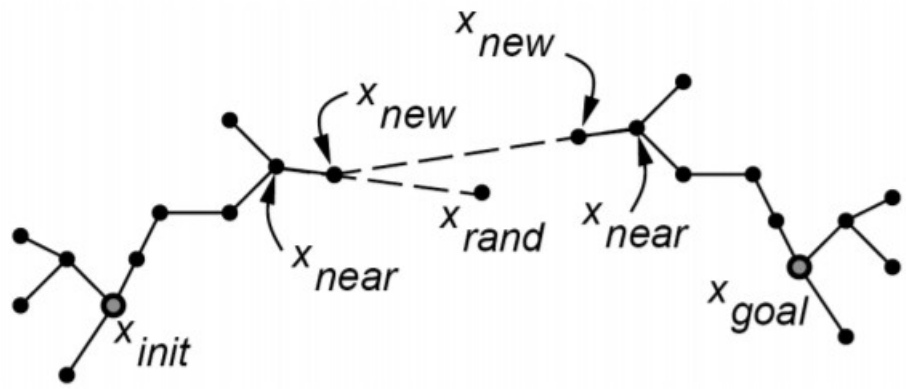


Figura 7: "Costruzione del grafo nell'algoritmo RRT"

Questa tecnica, a differenza della precedente, è più indicata per richieste singole di pianificazione, in quanto si risparmiano i tempi computazionali della learning phase della PRM; l'albero delle traiettorie si ramifica partendo da un unico nodo radice e non crea ambiguità sui versi di percorrenza.
L'algoritmo RRT* è la versione ottimizzata di RRT, grazie all'introduzione della fase di *Rewiring* ('ri-cablaggio') richiamata all'interno di ogni iterazione. Come suggerisce il termine, essa rimodella l'albero aggiungendo o togliendo alcuni archi per ottimizzare le diverse traiettorie; tale procedura permette di connettere inizialmente nodi molto distanti tra loro in modo da arrivare il più presto possibile ad una soluzione. In seguito, con l'aumentare del numero di nodi, le connessioni verranno permesse solo tra nodi più vicini migliorando così le traiettorie già trovate. Vi è da dire però che all'aumentare della cardinalità dell'albero, aumenta anche la probabilità di trovare nodi nella regione circolare necessaria per il Rewiring, comportando in questo modo una crescita esponenziale nei tempi di calcolo.

2.2.3 Confronto fra gli algoritmi PRM ed RRTs

A valle di tutte le osservazioni sovra esposte, si può affermare che due sono le grandi famiglie di algoritmi che si basano sul campionamento del C-Space (spazio delle configurazioni). Si parla di algoritmi di tipo *Single-Query Planning* quando, dato un

determinato ambiente virtuale, è necessario effettuare un'unica pianificazione. Se invece all'interno dello stesso ambiente virtuale vengono richieste diverse pianificazioni si parla di *Multiple-Query Planning*. La differenza fondamentale tra questi due tipi di approcci risiede nel fatto che le tecniche di Multiple-Query Planning effettuano un pre-processing del *configuration space* per velocizzare la fase successiva di pianificazione vera e propria. Il pre-processing ha tempi computazionali rilevanti, per questo motivo non è conveniente utilizzare tali tecniche nel caso in cui si debbano effettuare poche pianificazioni. La più vasta famiglia di algoritmi di tipo multiple-query sono le Probabilistic Road Maps (PRM), mentre i Rapidly-Exploring Random Trees (RRTs) vengono preferiti se si utilizzano tecniche single-query.
Si è trovato inoltre[7] che per ambienti particolarmente complessi ('labirintici') l'algoritmo PRM* (versione ottimizzata di PRM) trova la soluzione in tempi più brevi rispetto a RRT*; ciò perché per configurazioni in cui il robot è costretto a passare attraverso strette corsie, RRT* espande l'albero solo se il nodo generato appartiene alla direzione del passaggio. PRM* invece genera i nodi coprendo con probabilità uniforme tutti gli spazi liberi e, dopo averli connessi, trova (se possibile) la traiettoria finale. Per ambienti meno complessi invece, risulta più conveniente RRT*, poiché in questi casi non è necessario esplorare l'intero spazio delle configurazioni. In sostanza, l'algoritmo RRT*, apparentemente migliore in tutti gli aspetti rispetto a PRM*, a parità del numero di nodi, è meno efficiente in termini di copertura dello spazio libero; inoltre l'aggiunta della fase di Rewiring incrementa notevolmente i tempi necessari alla computazione dei codici pur migliorando in modo consistente la qualità delle traiettorie calcolate.

2.3 Fase di implementazione

2.3.1 Scelta dell'algoritmo

Scegliere un algoritmo appropriato è particolarmente importante quando si parla di *motion planning*. I differenti algoritmi esistenti in letteratura, infatti, sono stati sviluppati per particolari situazioni, principalmente legate alle dimensioni del C-Space. Per il caso applicativo in esame è stata giustificata la decisione di utilizzare algoritmi di tipo sampling-based a causa dell'elevata dimensione del configuration-space dei robot

utilizzati; d’altra parte vi sono indiscutibili vantaggi anche nell’ottica dell’implementazione tramite MATLAB, in quanto essi sono:

- facili da implementare;
- veloci;
- completi in senso probabilistico (la probabilità che un pianificatore trovi un cammino libero, se esiste, tende ad 1 al crescere del tempo di esecuzione, oppure, all’aumentare dei punti, la probabilità di non trovare il percorso va a zero).

Infine, vista la configurazione poco complessa dell’ambiente di lavoro previsto per il veicolo di riferimento, si è scelto di utilizzare l’algoritmo RRT* piuttosto che la PRM, ottenendo comunque dei tempi di risposta accettabili.

2.3.2 Implementazione in MATLAB

La semplicità con cui è possibile elaborare tale algoritmo è rimarcata inoltre dalla possibilità di richiamare l’oggetto ‘`pathPlannerRRT`’ in ambiente MATLAB, direttamente tramite la sua versione base. Tale oggetto permette appunto di configurare un pianificatore[8] di percorso basato su RRT*, esplorando l’ambiente circostante il veicolo. Dopodichè, definendo le funzioni opportune, viene ricavata la traiettoria dal punto iniziale a quello finale.
Il pianificatore richiede inoltre la definizione di alcune proprietà; innanzitutto, in base alla configurazione dell’ambiente di lavoro, occorre specificare una ‘`Costmap`’, necessaria per il controllo di collisione per tutte le pose random.

Figura 8: "Costruzione di una costmap"

È utile poi definire anche la tolleranza con cui si vuole far avvicinare il mezzo alla posizione obbiettivo, espressa sotto forma di un vettore [x,y,θ]. Tra le proprietà più importanti rientra inoltre anche la ‘`ConnectionDistance`’, ovvero la massima distanza tra due pose connesse; questo ed altri parametri dovranno essere impostati ragionevolmente sempre nell’ottica di compromesso tra risultato ottimo e brevi tempi computazionali.

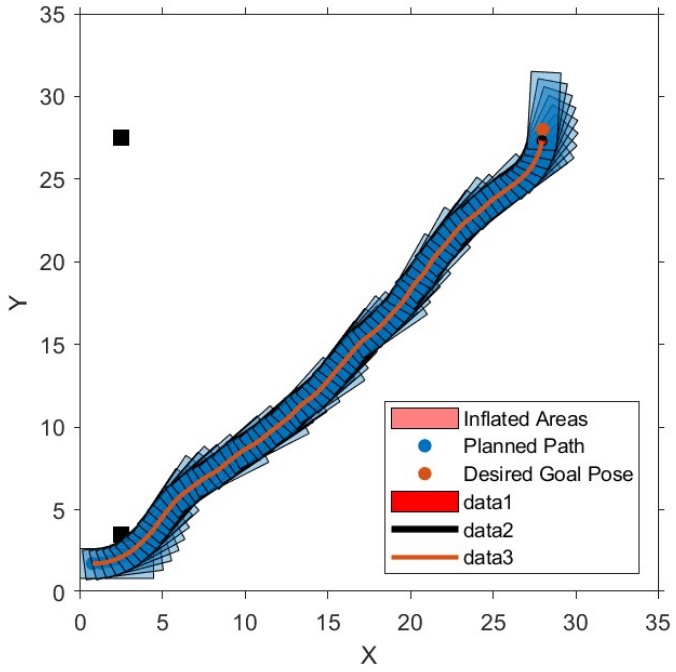


Figura 9: "Esempio di traiettoria determinata dal planner RRT"

Capitolo 3

3.1 Path following e obstacle avoidance

Nel presente capitolo verranno illustrate nel dettaglio le modalità con cui le strategie di path following ed obstacle avoidance sono state attuate e implementate.

Come già detto, l'obbiettivo di questa fase della simulazione consiste nello sviluppo delle logiche suddette in modo tale che operino in parallelo e in tempo reale. Ciò che è già noto è ciò che risulta dalla fase di path planning, ossia la traiettoria ideale da inseguire costituita da una serie di punti nel piano caratterizzati dalle coordinate {x, y, θ}. Ciò che invece si intende ricavare sono la velocità longitudinale e quella di imbardata del veicolo, dalle quali poter ottenere i comandi da impartire al mezzo.

Inoltre, da questo momento in poi l'implementazione avverrà direttamente in ambiente Simulink.

3.1.1 Elaborazione della strategia di inseguimento

Da un punto di vista più generale del lavoro svolto, questa si è rivelata essere senza dubbio la fase più problematica. Il motivo principale riguarda la scelta di alcuni parametri cinematici, col fine di rispettare i limiti ammissibili del veicolo e allo stesso tempo di consentire un'adeguata comunicazione tra Simulink e ADAMS.
Inizialmente la strategia di inseguimento è stata implementata facendo ricorso alla sezione 'Robotics System Toolbox: Mobile Robot Algorithms' nella libreria Simulink, selezionando in particolare il blocco 'Pure Pursuit'.

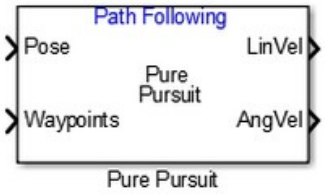


Figura 10: "Blocco Simulink: Pure Pursuit"

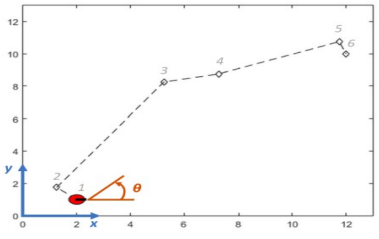


Figura 11: "Sistema di riferimento utilizzato dall'algoritmo path following"

Questo, in particolare, ricava i comandi di velocità lineare e angolare per inseguire un dato percorso, noti in ingresso un set di waypoints e la posizione corrente del robot.

L'oggetto 'Pure Pursuit' è un 'controllore' che quindi aggiorna istante per istante gli output sulle velocità, sulla base delle informazioni aggiornate riguardo la posizione che riceve in input[9]. Le caratteristiche del robot invece sono rappresentate in questo caso da due proprietà da impostare nell'interfaccia del blocco Simulink, ovvero la '`DesiredLinearVelocity`' e la '`MaxAngularVelocity`'. La prima, in m/s, implica una velocità lineare costante del mezzo, indipendente dalla velocità angolare; la seconda, in rad/s, costituisce invece un limite sulla velocità di imbardata che il veicolo non può superare. Un'ulteriore proprietà, la '`LookaheadDistance`' ('distanza di veduta'), determina poi un *look-ahead point* sul percorso e quanto lontano esso è posizionato, costituendo un obbiettivo locale per il robot, e la velocità angolare viene determinata in base a tale punto.
Come mostra questa immagine, si nota che il percorso effettivo non corrisponde alla linea diretta tra i waypoint.

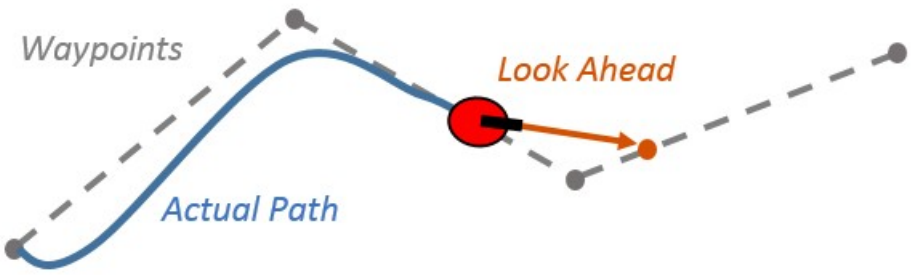


Figura 12: "Costruzione della traiettoria reale: distanza di veduta"

Modificare la distanza di veduta ha un impatto significante sulle performance dell'algoritmo: una maggiore `LookaheadDistance` produce una traiettoria più uniforme per il robot, ma può far sì che il mezzo tagli gli angoli lungo il percorso. Invece, un valore basso di tale proprietà può generare oscillazioni nel tracciare il percorso, dando origine ad un comportamento instabile.

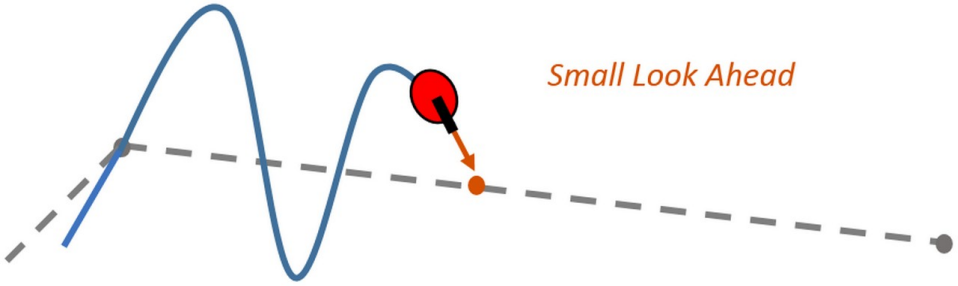


Figura 13: "Esempio: distanza di veduta piccola"

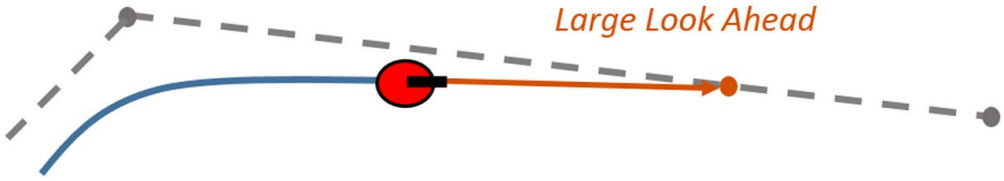


Figura 14: "Esempio: distanza di veduta grande"

Si evince dunque che il controllore 'Pure Pursuit' presenta alcune limitazioni: innanzi tutto non permette di seguire esattamente il profilo della traiettoria desiderata tra due punti, ma occorre ottimizzare i parametri per migliorare le prestazioni e convergere al percorso ideale nel tempo. Inoltre esso non consente di stabilizzare il robot esattamente nel punto finale, ma bisogna impostare un'opportuna distanza di soglia che ne comporti l'arresto nelle immediate vicinanze del target.

3.1.2 Elaborazione della strategia di obstacle avoidance

Come già accennato, parallelamente al path follower è stata sviluppata una logica che consentisse di evitare in tempo reale eventuali ostacoli incontrati lungo il percorso che

inizialmente non erano previsti. Anche in questo frangente, dalla sezione ‘Robotics System Toolbox: Mobile Robot Algorithms’ della libreria Simulink è stato prelevato il blocco ‘Vector Field Histogram’.

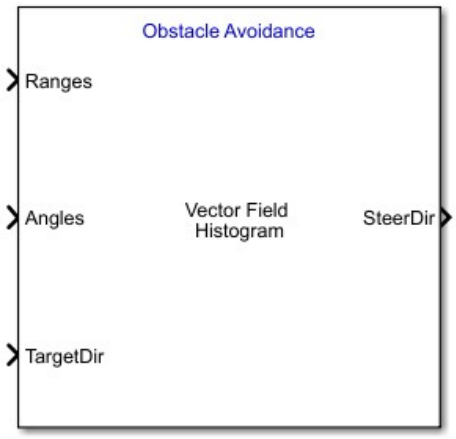


Figura 15: "Blocco Simulink: Vector Field Histogram"

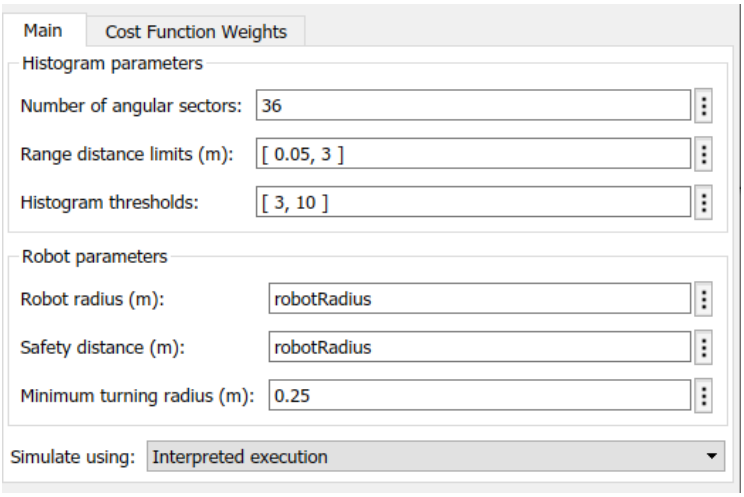


Figura 16: "Interfaccia del blocco 'Vector Field Histogram'"

Il suo funzionamento si basa sui dati provenienti dai sensori radar, che in questa sede saranno in realtà simulati tramite ADAMS View. In particolare, una volta fornite in ingresso le letture dei sensori in termini di distanze e angoli rispetto alla posizione degli ostacoli, e un obbiettivo verso cui far muovere il mezzo, il controllore VFH determina una direzione di sterzata obstacle-free. Tale blocco Simulink sfrutta pertanto l’algoritmo ‘Vector Field Histogram’, illustrato precedentemente nel capitolo 1 e per mezzo del quale costruisce un istogramma polare per le posizioni degli ostacoli. Tra i parametri principali da impostare nella finestra di interfaccia troviamo le soglie dell’istogramma (‘*Histogram thresholds*’), fornite tramite un vettore di due elementi, che vengono utilizzate dall’algoritmo per determinare un istogramma binario. Valori della densità polare di ostacoli più grandi della soglia maggiore, vengono rappresentati nell’istogramma binario come spazi occupati (1); valori più piccoli della soglia inferiore rappresentano spazi liberi (0). Se la densità presenta invece un valore intermedio tra i due limiti, l’algoritmo fa riferimento all’istogramma ricavato nelle iterazioni precedenti

se queste esistono, altrimenti il valore viene associato allo spazio libero (0). Successivamente, a partire dall'istogramma binario esso determina un istogramma 'mascherato' sulla base del raggio di sterzata minimo del robot ('*Minimum turning radius*'). Questo ed altri parametri da dover impostare sono mostrati nell'immagine sovrastante.
L'algoritmo seleziona più direzioni di sterzata in base allo spazio libero e alle possibili direzioni di guida. Una cost-function determina il costo associato ad ognuna di esse, con pesi corrispondenti alle direzioni precedenti, attuale e obbiettivo; dopodiché l'algoritmo restituisce una direzione obstacle-free al minimo costo. Sfruttando quindi tale risultato, si possono inserire i comandi per far muovere il robot in quella direzione.
I parametri dell'algoritmo devono essere ottimizzati in base all'applicazione e all'ambiente; i valori dipendono dal tipo di robot, dai sensori e dall'hardware utilizzato.
Ricapitolando, gli input richiesti dal blocco VFH sono:

1. i valori delle distanze ('*Ranges*') provenienti dalle letture dei sensori, come un vettore di scalari in metri. Ciascun valore corrisponde ad un determinato angolo, dunque il vettore deve essere della stessa lunghezza del vettore degli angoli.
2. I valori degli angoli ('*Angles*') dai dati di scansione, come un vettore di scalari in radianti. Sono angoli in corrispondenza di distanze specifiche; il vettore deve essere della stessa lunghezza del vettore delle distanze corrispondenti.
3. Direzione target del robot ('*TargetDir*'), specificata come uno scalare in radianti. La direzione in avanti è considerata 0 radianti, con angoli positivi misurati in senso antiorario.

Le considerazioni di quest'ultimo punto riguardano anche la direzione di sterzata risultante dall'algoritmo ('*SteerDir*'), espressa sempre in radianti.

3.2 Integrazione tra le strategie e logica di guida

A questo punto non resta che realizzare una connessione tra la strategia di path following e quella di obstacle avoidance, affinché il sistema complessivo operi in parallelo. Ciò è possibile abilitando l'uscita '*TargetDir*' dal blocco 'Pure Pursuit' e

collegandola direttamente al blocco ‘Vector Field Histogram’: la direzione obbiettivo determinata dal path follower costituisce infatti proprio uno dei tre input dell’algoritmo per l’obstacle avoidance.

Pertanto il diagramma a blocchi così descritto appare in questo modo:

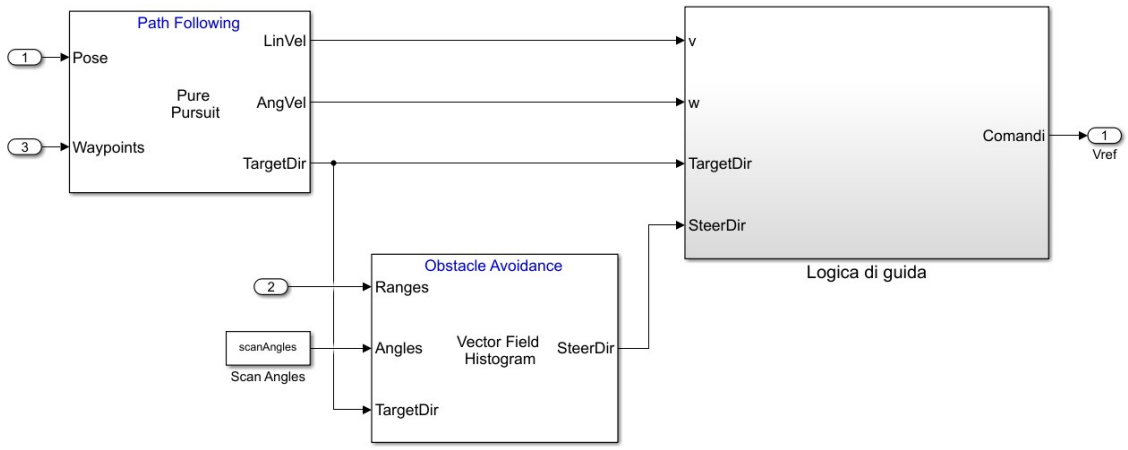


Figura 17: "Control Strategy"

In figura si nota poi che le uscite dei due blocchi fin qui descritti sono connesse ad un ulteriore sottosistema, all’interno del quale è stata realizzata una logica di guida finalizzata a regolare i comandi da utilizzare, in particolare quello riguardante la velocità angolare di imbardata ‘w’. Il controllo si basa sulla differenza tra i valori dell’angolo associato alla direzione obbiettivo ‘*TargetDir*’, e quello associato alla direzione di sterzata ‘*SteerDir*’ ricavata dal controllore VFH: se tale differenza è prossima a zero, il comando sulla velocità di imbardata coincide con il valore risultante dal controllore ‘Pure Pursuit’, altrimenti viene posto pari al prodotto dello scarto per un opportuno guadagno.

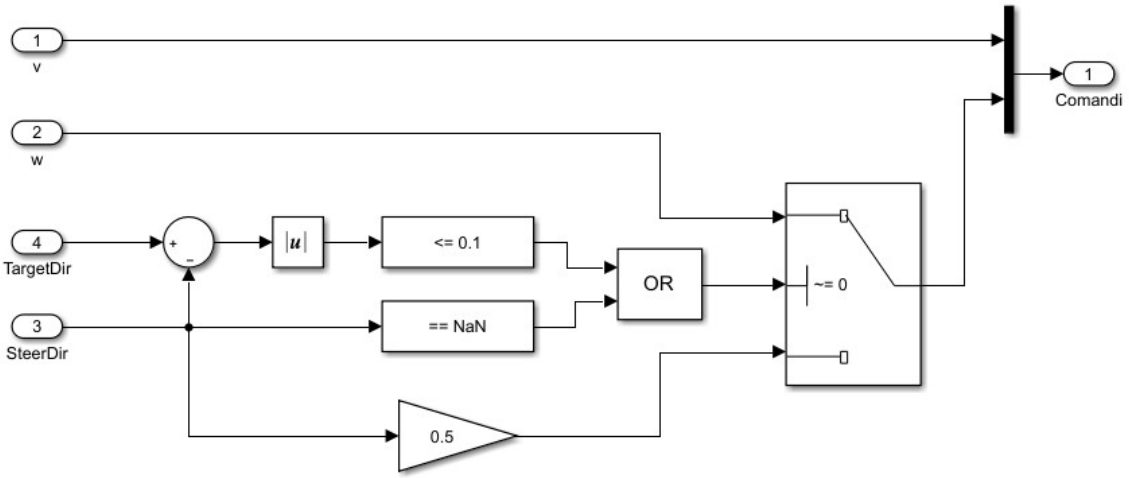


Figura 18: "Logica di guida"

Data però l'impossibilità di poter analizzare nel dettaglio le funzioni utilizzate dal controllore 'Pure Pursuit' per determinare 'v' e 'w', e sopratutto in che modo esso gestisce i parametri impostati all'interfaccia, si è deciso in un secondo momento di sostituire il blocco preimpostato con una *chart*, ossia un sottosistema nella sezione 'Stateflow' della libreria grazie al quale poter implementare una logica di controllo tramite una *macchina a stati finiti*[10].

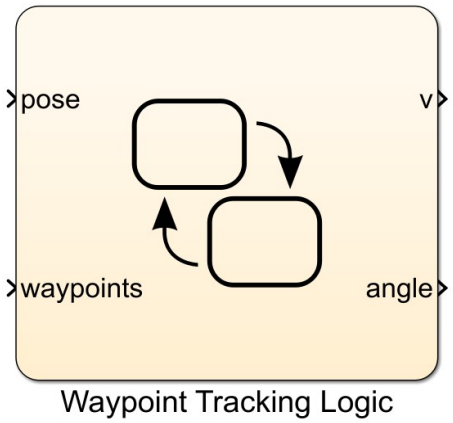


Figura 19: "Stateflow Chart"

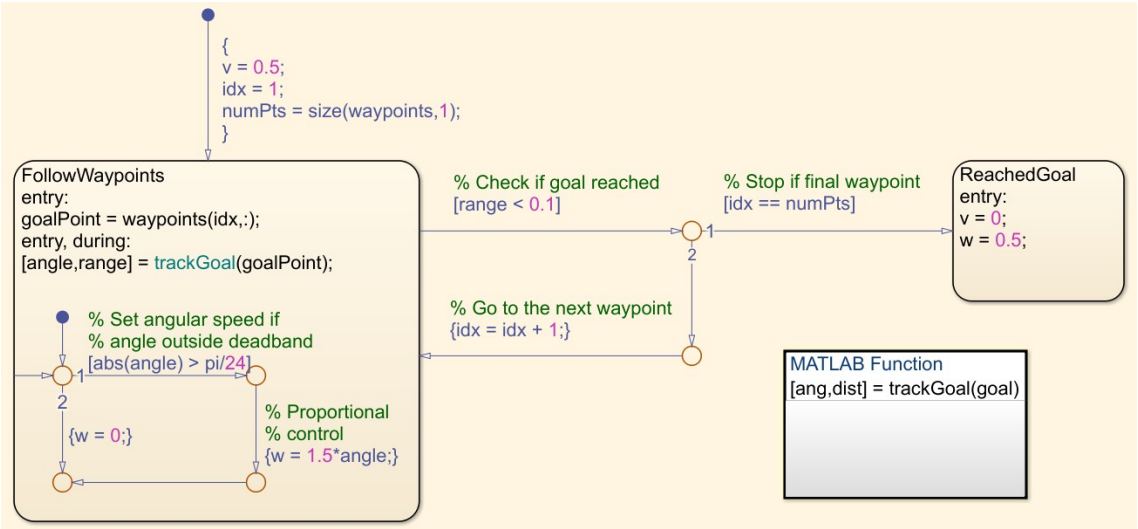


Figura 20: "Esempio: Waypoint Tracking Logic"

Nell'immagine sopra riportata, si può vedere un esempio di quanto illustrato, presente nella libreria MATLAB 'Mobile Robotics Simulation Toolbox' a cui è stato fatto riferimento. Una macchina a stati finiti è una rappresentazione di un sistema 'reattivo' guidato da eventi, ossia un sistema che reagisce ad un evento effettuando una transizione da uno stato a un altro. Questa transizione si verifica se la condizione che definisce la modifica è vera.
La scelta di sostituire l'algoritmo 'Pure Pursuit' con una Stateflow chart è stata dettata appunto dalla possibilità di modificare direttamente le equazioni e i parametri al suo interno, ma di fatto non influisce in alcun modo sulle caratteristiche del sistema complessivo. Gli ingressi e le uscite del sottosistema infatti sono le stesse del blocco originario.

Dunque la velocità longitudinale e la velocità angolare di sterzo così ricavate costituiscono gli ingressi al modello dinamico del veicolo simulato dapprima in ambiente Simulink, e in ADAMS View poi.

Capitolo 4

4.1 Simulazione della fase dinamica: Simulink

Dopo aver definito un'opportuna logica di controllo non resta che sviluppare un modello che possa simulare la dinamica del veicolo in maniera realistica. In questa fase ci si aspetta di ricavare in tempo reale la risposta ai comandi impartiti dalla logica di controllo, ossia come questi influenzano l'avanzamento e l'orientazione del mezzo. Le uscite dovranno essere dunque i dati {x, y, θ} in continuo aggiornamento, necessari alla fase precedente per determinare i nuovi valori di velocità longitudinale 'v' e angolare 'w', sempre nel tentativo di inseguire la traiettoria desiderata.
In un primo momento tutto ciò è stato implementato in Simulink, partendo dapprima da un modello di veicolo semplificato a guida differenziale, poi sostituito con uno più verosimile rappresentante un veicolo four-wheel steering.

4.1.1 Modello cinematico: 'Differential Drive'

Secondo tale modello[11], il veicolo può essere ricondotto ad un mezzo avente due ruote indipendenti utilizzate per controllare la velocità longitudinale e di imbardata. Questo implica quindi che le due ruote possono viaggiare con diversa velocità angolare.

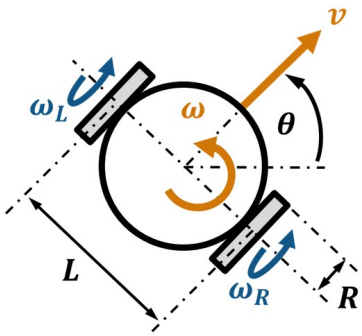


Figura 21: "Modello 'Differential Drive'"

Cambiando la velocità delle ruote per avere movimenti differenti, il robot ruota attorno ad un punto che giace lungo l'asse comune delle due ruote: esso è noto come centro di istantanea rotazione (*CIR*). La velocità angolare Ω prodotta da questa rotazione è nota e valgono le seguenti equazioni[12]:

$$\Omega(D+L/2)=V_r$$
$$\Omega(D-L/2)=V_l$$

dove D è la distanza il centro di istantanea rotazione e il punto medio tra le ruote, L è la carreggiata, V_r e V_l sono rispettivamente le velocità lineari della ruota destra e della ruota sinistra.
Per ogni istante di tempo è possibile quindi ricavare:

$$D=L(V_l+V_r)/2(V_l-V_r)$$
$$\Omega=(V_r-V_l)/L$$

Per il sistema così definito la traiettoria del veicolo dipende unicamente dalle velocità delle ruote:

1. se $V_l = V_r$ il moto sarà lineare;
2. se $V_l = -Vr$ il robot ruota attorno al proprio asse centrale;
3. se le velocità non sono uguali la traiettoria risultante sarà il risultato di una combinazione dei moti traslatorio e rotazionale.

Per comprendere meglio quanto appena detto, si assuma che il mezzo si trovi in una data posizione (x,y) formando un angolo θ con l'asse X. Dopo un certo intervallo di tempo la nuova posizione sarà (x',y') e il nuovo angolo sarà θ'. Inoltre

$$CIR=[x-Dsin(\theta), y+Dcos(\theta)]$$

In questo caso, il blocco di riferimento prelevato dalla libreria MATLAB 'Mobile Robotics Simulation Toolbox', restituisce in output proprio la posa corrente del robot, ma richiede in ingresso le velocità angolari delle due ruote. Ciò che è noto però dalla fase di elaborazione dei comandi sono la velocità longitudinale e di imbardata del

veicolo, applicate idealmente nel centro di massa. Occorre pertanto applicare prima le equazioni di cinematica inversa per ricavare le velocità delle ruote; di seguito si riportano le relazioni di cinematica diretta e inversa tra le varie velocità in questione.

Cinematica diretta:

$$v = \frac{R}{2}(\omega_R + \omega_L) \qquad \omega = \frac{R}{L}(\omega_R - \omega_L)$$

Cinematica inversa:

$$\omega_L = \frac{1}{R}(v - \frac{\omega L}{2}) \qquad \omega_R = \frac{1}{R}(v + \frac{\omega L}{2})$$

In seguito è possibile quindi utilizzare direttamente il blocco 'Differential Drive Simulation' inserendo le dimensioni delle ruote, la posa iniziale del robot e il tempo di campionamento.

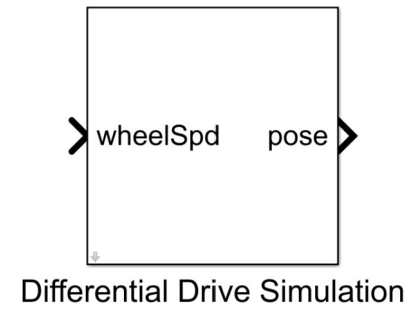


Figura 22: "Blocco Simulink: Differential Drive Simulation"

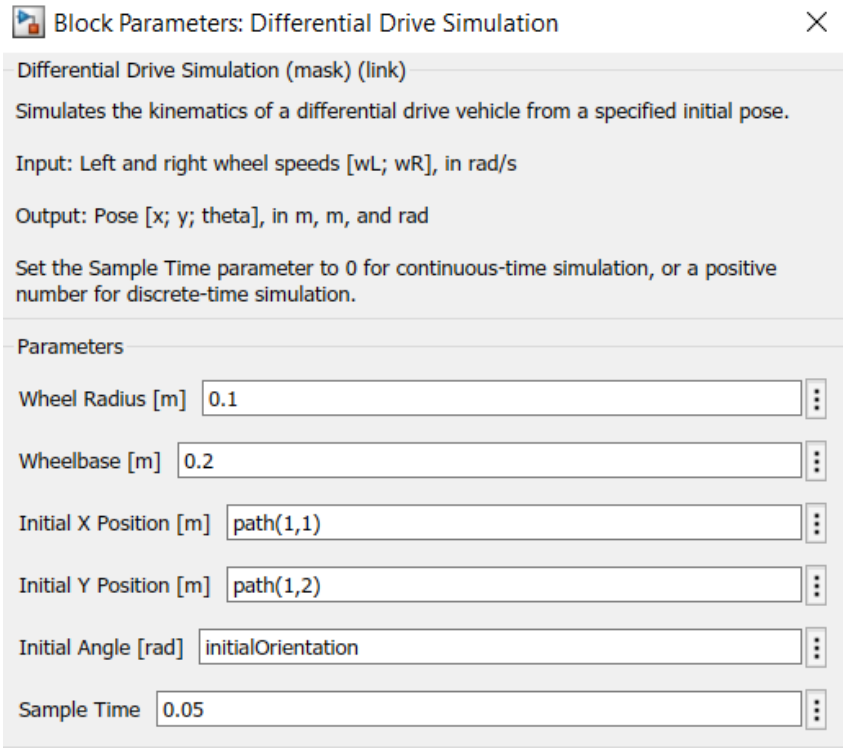


Figura 23: "Interfaccia del blocco 'Differential Drive Simulation'"

4.1.2 Modello cinematico: ‘Four-Wheel Steering’

La teoria alla base di questo modello fa riferimento ad un veicolo con quattro ruote che possono essere tutte guidate e sterzate indipendentemente. Si assume per semplicità il cosiddetto ‘Ackermann steering’, cosicché il mezzo può essere approssimato ad un sistema avente due ruote, comunemente detto ‘modello bicicletta’[13].

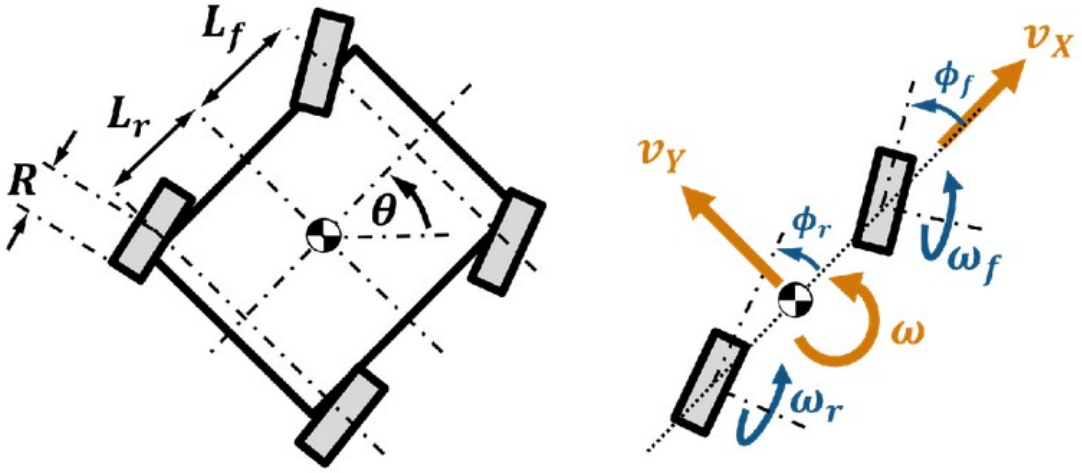


Figura 24: "Modello FWS: semplificazione al modello bicicletta"

Per descrivere il problema si assume inoltre la validità delle ipotesi di moto planare, corpo rigido e non slittamento delle ruote; il veicolo si muove idealmente su un piano orizzontale dove è fissato un sistema di coordinate X-Y.

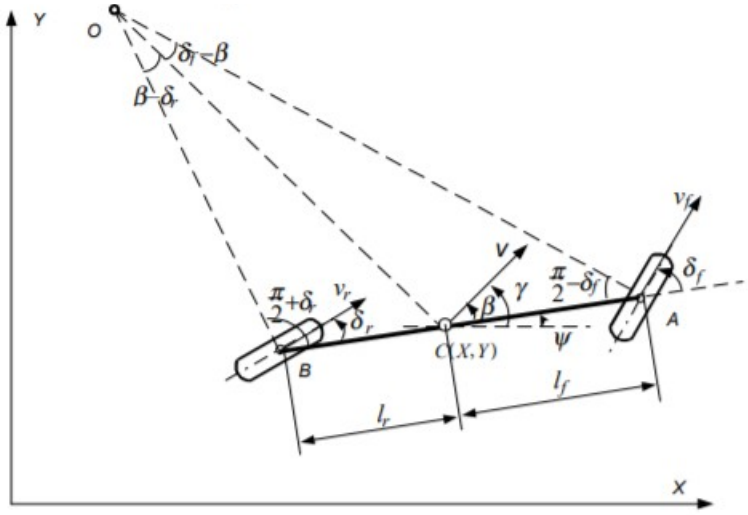


Figura 25: "Sistema di riferimento nel modello bicicletta"

La sua posizione viene individuata dal centro di massa C di coordinate (X,Y), in corrispondenza del quale viene definita anche la sua velocità *v*. L'angolo ψ è l'angolo tra l'asse X e l'asse longitudinale del mezzo *AB*; γ invece è quello individuato tra l'asse X e la direzione del vettore velocità *v*. β rappresenta l'angolo di deriva, definito tra l'asse longitudinale *AB* e la direzione della velocità del veicolo. Da notare poi il raggio di sterzata *r*, ossia la distanza tra il centro di istantanea rotazione *O* e il centro di massa *C*.

Ai fini del problema occorre individuare infine gli angoli di sterzata della ruota anteriore e di quella posteriore, che da questa formulazione semplificata possono essere ricondotti rispettivamente agli angoli di sterzo delle ruote di ciascun assale del veicolo a quattro ruote. Quindi è necessario prima specificare la velocità della ruota anteriore v_f, definita in corrispondenza dell'intersezione tra il piano medio della ruota frontale e il suo asse *A*, e la velocità della ruota posteriore v_r, definita in modo analogo con il piano medio della ruota e il suo asse *B*. A questo punto l'angolo di sterzo anteriore δ_f è l'angolo tra l'asse longitudinale *AB* e la direzione di v_f, così come l'angolo di sterzo posteriore δ_r è definito tra *AB* e la direzione di v_r.

Con riferimento sempre alla figura 23 il modello cinematico '4WS' può essere espresso come segue.

$$\dot{X} = v\cos(\Psi + \beta)$$

$$\dot{Y} = v\sin(\Psi + \beta)$$

$$\dot{\Psi} = \frac{v\cos\beta(\tan\delta_f + \tan\delta_r)}{l_f + l_r}$$

Dove

$$\beta = \arctan\frac{l_f\tan\delta_r + l_r\tan\delta_f}{l_f + l_r}$$

$$v = \frac{v_f\cos\delta_f + v_r\cos\delta_r}{2\cos\beta}$$

La cinematica diretta ricavata in ambiente MATLAB implica pertanto le seguenti relazioni.

$$v_X = \frac{R}{2}\left(\omega_f \cos\phi_f + \omega_r \cos\phi_r\right)$$

$$v_Y = \frac{R}{2}\left(\omega_f \sin\phi_f + \omega_r \sin\phi_r\right)$$

$$\omega = \frac{R}{L_f + L_r}\left(\omega_f \sin\phi_f - \omega_r \sin\phi_r\right)$$

Gli inputs in questo caso sono le velocità angolari delle ruote anteriori e posteriori [ω_f; ω_r] in *rad/*s e gli angoli di sterzo [ϕ_f; ϕ_r] in *rad*. Gli outputs invece sono le velocità lineari v_x e v_y in *m/s* e la velocità di imbardata del mezzo ω in *rad/s*.

Date le caratteristiche della cinematica 'accoppiata' prevista dal modello bicicletta, i files presenti nel Toolbox di MATLAB e Simulink propongono tre approcci differenti di cinematica inversa.

1) *Zero sideslip*: nel primo caso si assume che ruote anteriori e posteriori sterzano con angoli opposti per minimizzare lo slittamento. Gli inputs sono la velocità di avanzamento v_x e la velocità angolare ω.

$$\omega = \frac{v_X}{R\cos\phi}, \qquad \omega_r = \omega_f = \omega$$

$$\phi = \arctan\left(\frac{\omega\left(L_f + L_r\right)}{v_X}\right), \qquad \phi_r = -\phi_f = \phi$$

2) *Parallel steering*: si assumono angoli di sterzo uguali sia all'anteriore che al posteriore, cosicché il veicolo si muova senza ruotare. Gli inputs sono le velocità lineari v_x e v_y.

$$v = \frac{sign\left(v_x\right)}{R}\sqrt{v_X^2 + v_Y^2}, \qquad \omega_r = \omega_f = \omega$$

$$\phi = \arctan(v_Y / v_X), \qquad \phi_r = -\phi_f = \phi$$

3) *Front steering*: in questo caso ϕ_r è nullo, dunque sterza solo l'asse anteriore. Come per il modello *zero sideslip*, gli inputs sono la velocità longitudinale v_x e la velocità di imbardata ω.

$$\omega_f = \frac{v_X}{R\cos\phi_f}, \qquad \omega_r = \frac{v_X}{R}$$

$$\phi_f = \arctan\left(\frac{\omega(L_f + L_r)}{v_X}\right), \qquad \phi_r = 0$$

Scegliendo quest'ultima opzione di default, nella figura sottostante si riporta il sistema complessivo; si nota anche la presenza dell'oggetto 'Lidar Sensor', anteposto alla strategia di controllo. Questo serve proprio a simulare un sensore virtuale utile a determinare le distanze e gli angoli corrispondenti al posizionamento degli ostacoli, in base ai quali l'algoritmo VFH ricava la direzione di sterzata come illustrato nel capitolo precedente.

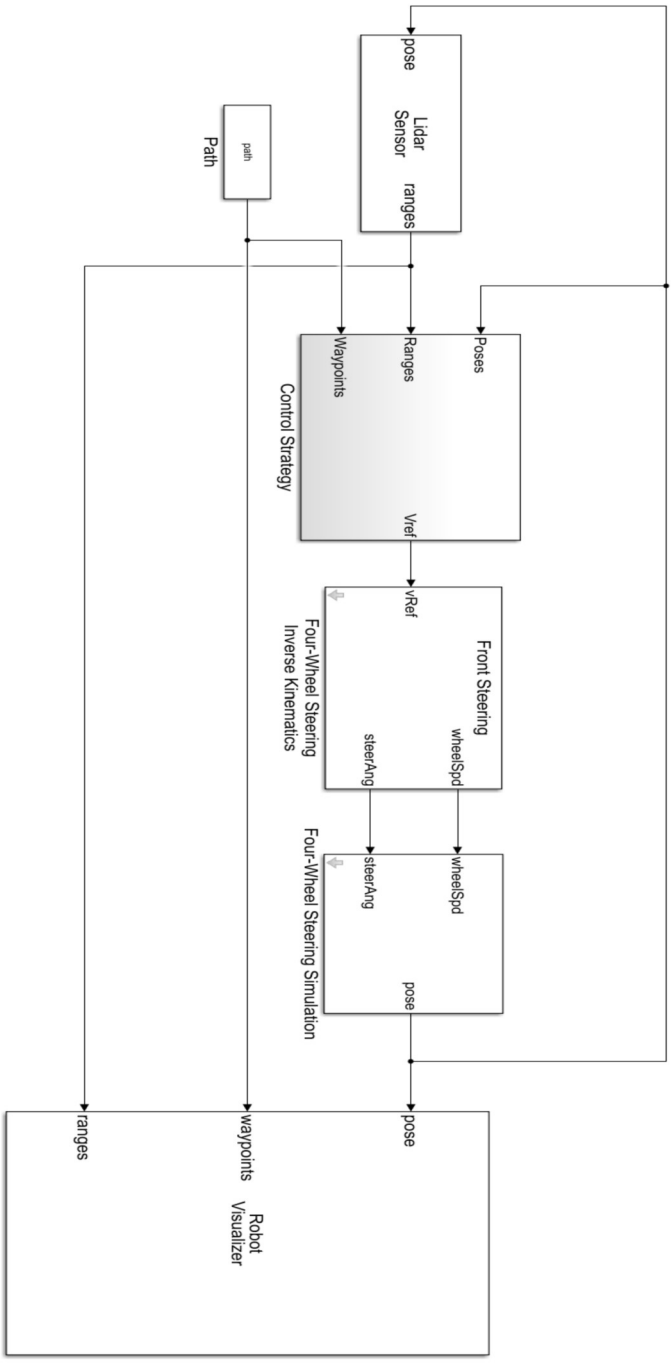
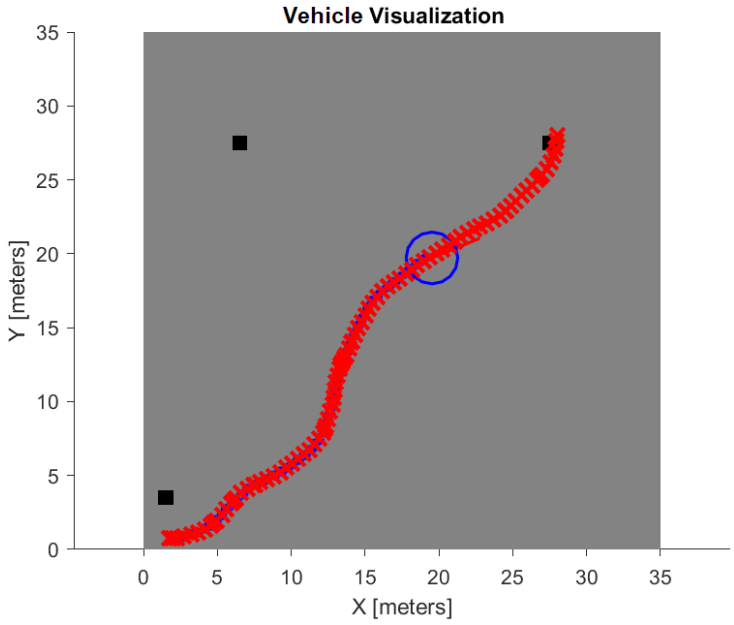


Figura 26: "Modello Simulink complessivo"

Sulla sinistra il blocco ‘Path’ rappresenta proprio il set di punti (x, y) che costituiscono il percorso desiderato, ricavato a priori nella fase di pianificazione della traiettoria in ambiente MATLAB.
Il blocco sulla destra, denominato ‘Robot Visualizer’, permette invece di visualizzare in tempo reale la posizione e l’orientamento del robot in un ambiente 2D, e poterla confrontare eventualmente con i waypoints di riferimento. Durante la simulazione appare quindi una finestra come quella riportata di seguito.



Fguira 27: "Vehicle Visualizer: traiettoria in tempo reale"

4.2 Fase di CO-simulazione con ADAMS View

Dopo aver accertato il funzionamento del sistema in ambiente Simulink, si è passati alla realizzazione del modello 3D del veicolo tramite ADAMS View, sia a causa delle limitazioni circoscritte al piano 2D legate alla simulazione con Simulink, sia al fine di ottenere risultati più realistici per quanto concerne la dinamica del veicolo.
Il modello di partenza è molto semplice: si immagina di far muovere il robot su un terreno delimitato, inizialmente pianeggiante, delle dimensioni di circa 35x35 m. In questa fase dunque si è cercato il più possibile di rispettare quanto è stato impostato in MATLAB, durante la scrittura del codice di path planning; perciò anche la posizione di partenza del mezzo, o meglio del suo centro di massa, si trova nella parte sudest della piattaforma, in accordo con quanto svolto precedentemente rispetto al piano X-Y.

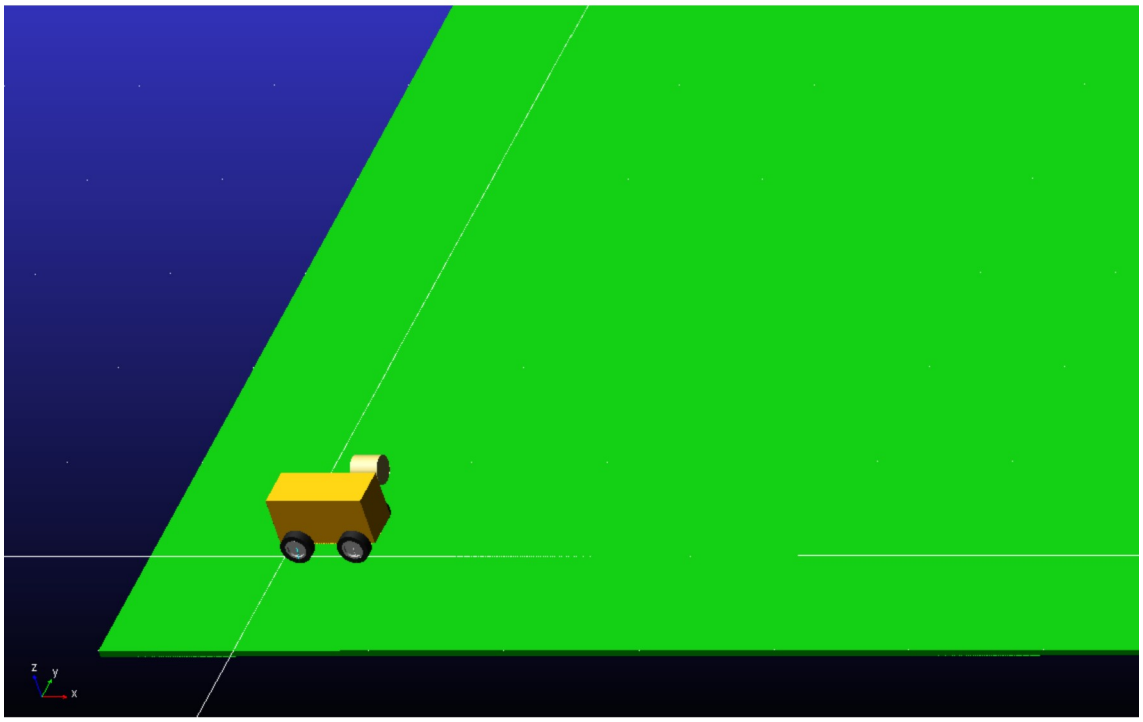


Figura 28: "Modello Adams"

Come si può vedere dall'immagine, anche il modello del veicolo è ultra semplificato, sebbene è stata posta particolare attenzione alle dimensioni generali e alle proprietà inerziali che verosimilmente possono rappresentare un mezzo agricolo, ossia il veicolo da lavoro oggetto della presente trattazione. Tenendo presente che il centro di massa preso a riferimento per i calcoli successivi è stato ricavato a partire dalla massa aggregata delle ruote e del corpo centrale del mezzo, le inerzie attribuite in prima approssimazione a ciascuna parte di interesse sono riportate nella tabella sottostante.

Proprietà inerziali	
Massa del singolo cerchione [kg]	277,05
Massa del singolo pneumatico [kg]	315,54
Massa del corpo centrale [kg]	2,27E+04
Massa aggregata [kg]	2,5E+04

Tabella 1: "Caratteristiche inerziali delle parti veicolo"

Il contatto stabilito tra ruote e terreno è stato creato mediante forze di tipo impulsivo, seguendo la legge di Coulomb per quanto riguarda l'attrito. In particolare, i parametri caratteristici del contatto sono riportati nella tabella sottostante.

Normal Force		Friction Force	
Forse Exponent	2.2	Static Coefficient	0.6
Stiffness	1.0E+09	Dynamic Coefficient	0.4
Damping	1.0E+04	Stiction Transition Vel.	0.1
Penetration Depth	1.0E-05	Friction Transition Vel.	1

Tabella 2: "Caratterisctiche del contatto ruota-terreno"

In un secondo momento inoltre, si è pensato di impartire alle sole ruote posteriori un comando di coppia anziché di velocità, in modo tale da poter simulare al meglio la caratteristica meccanica del motore; per il caso in esame essa prevede un tratto iniziale a coppia costante di 600 Nm fino a 1000 giri/min, seguito da un tratto a potenza costante fino ad un massimo di 4000 giri/min. Supponendo inoltre di far muovere il veicolo con la prima marcia inserita, è stato previsto un rapporto di riduzione complessivo alle ruote pari a τ=75. La coppia impartita sarà dunque funzione della velocità angolare alla singola ruota e di un parametro α che varia tra 0 ed 1 e che verosimilmente può essere ritenuto dipendente da quanto viene premuto l'acceleratore.

$$C_r = \left(\frac{\tau}{2}\right) * \alpha * C_m(\omega_m)$$

$$\omega_m = \tau * \omega_r$$

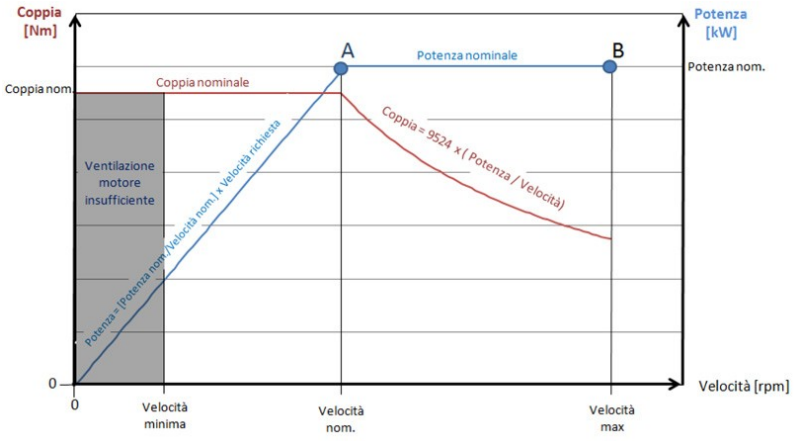


Figura 29: "Caratteristica meccanica del motore"

4.2.1 Integrazione del modello ADAMS in ambiente Simulink

Tramite la configurazione di alcune impostazioni è stato possibile esportare il file ADAMS, e tutte le informazioni ad esso relative, all'interno di un unico blocco Simulink come quello riportato a lato.

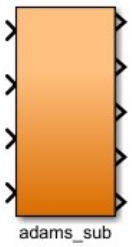


Figura 30: "Sottosistema del modello Adams"

Gli ingressi e le uscite devono essere definiti durante la realizzazione del modello ADAMS, creando delle 'State Variables' che agiscono come parametri scambiati tra i due software[14].
Con riferimento al caso in esame, gli input provenienti da Simulink sono il parametro α caratterizzante il comando di coppia, e l'angolo di sterzo frontale ϕ_f. Le uscite invece costituiscono in parte i parametri di feedback necessari alla logica di controllo, in parte dei parametri di monitoraggio per verificare un effettivo riscontro tra i risultati della simulazione multibody e della co-simulazione. Tra le uscite appartenenti alla prima categoria rientrano i dati $\{x, y, \theta\}$ riguardanti la posa attuale del robot, e la velocità lineare, utili come già detto ad elaborare i comandi aggiornati istante per istante. Invece le informazioni riguardanti ad esempio la coppia e la velocità angolare alle ruote posteriori hanno piuttosto una finalità di verifica.
Dunque, il blocco Adams va a sostituire di fatto il blocco 'Four-Wheel Steering Simulation', visibile nello schema complessivo di Figura 24. Scompare anche l'oggetto 'Lidar Sensor', poiché in questo caso il sensore radar virtuale viene creato direttamente in Adams tramite la definizione di 'Measures', riguardanti sia le distanze che i corrispettivi angoli tra il veicolo e gli ostacoli. Tali misure coincidono appunto con due uscite del blocco Adams, definite rispetto a due markers creati rispettivamente nel centro di massa del robot e nel centro di massa dell'ostacolo.

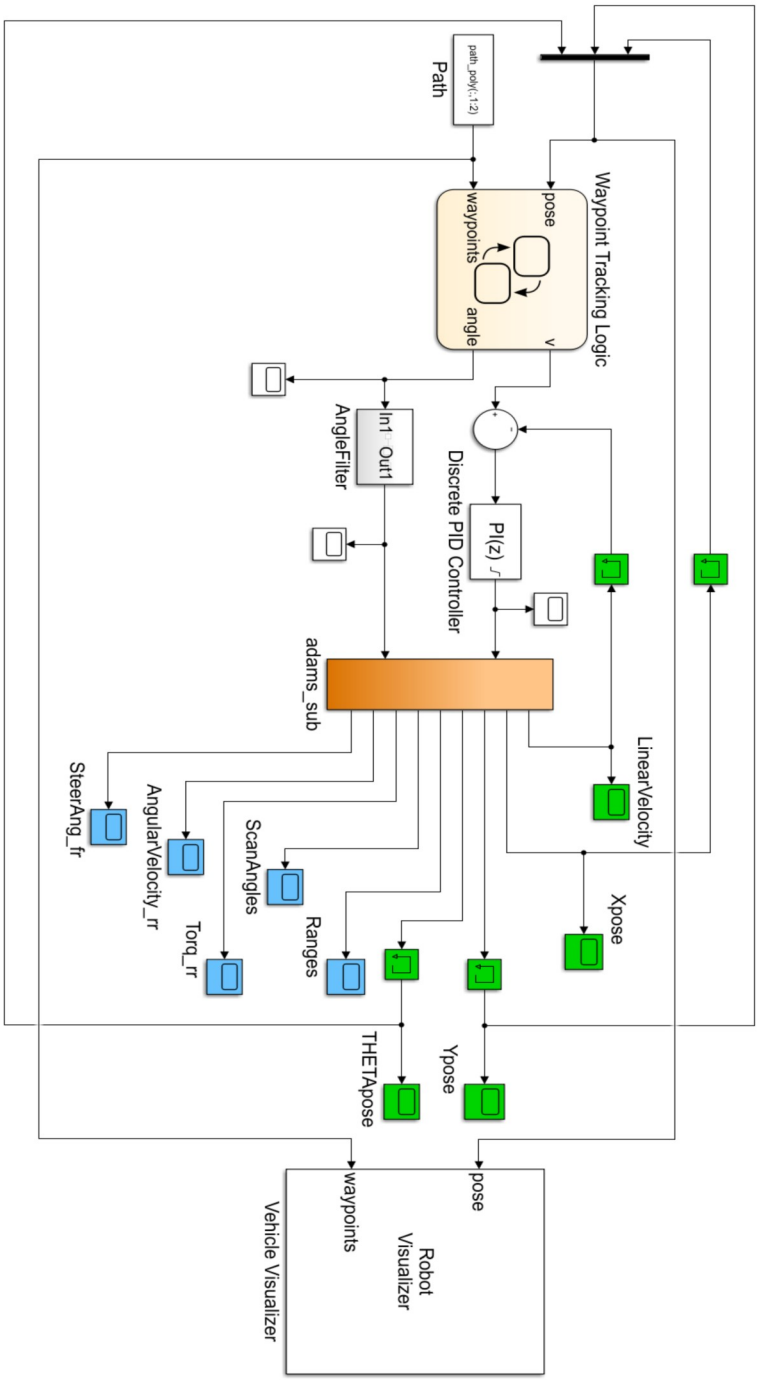


Figura 31: "Modello di co-simulazione in anello chiuso"

4.2.2 Fase di validazione: co-simulazione in anello aperto

L'integrazione del sottosistema Adams all'interno del diagramma è stata tutt'altro che banale: la problematica principale si è rivelata essere infatti la configurazione dei parametri di comunicazione e del tempo di integrazione. Innanzi tutto, come si può notare, in corrispondenza di degli output di feedback del blocco arancione è presente un blocco 'Memory', evidenziati in verde: affinché il sistema funzionasse correttamente è stato cioè necessario ritardare tutte le uscite per un opportuno tempo di campionamento, ereditato dalla simulazione, impostando come condizione iniziale la relativa coordinata (x, y ,θ o v) di partenza.
Tuttavia, ancor prima di testare l'intero modello in retroazione, è stato accertato che, soggetto a determinati input, il mezzo rispondesse come ci si aspettava. La verifica è stata effettuata mediante co-simulazione in anello aperto: in un primo momento sono stati monitorati gli andamenti della coppia, delle velocità angolari alle ruote posteriori e della velocità lineare del mezzo, oltre che la sua posa attuale costituita dalle coordinate {x, y, θ}, imponendo andamenti noti del parametro α e dell'angolo di sterzo frontale φ_f. Cercando di rispettare la coerenza tra i sistemi di riferimento, il modello Adams è stato realizzato in modo tale da consentire il moto del veicolo nel piano X-Y, con la gravità diretta lungo l'asse Z negativo. A valle di tale osservazione, è stata quindi verificata la concordanza di segno e di andamento delle variabili sopra citate, dapprima ricavandone i risultati unicamente in ambiente Adams, successivamente in fase di co-simulazione in Simulink garantendo l'imposizione degli stessi input (α, φ_f). Di seguito è possibile osservarne un confronto in alcuni grafici, ricavati in questo caso con α = step(time, 0, 0, 2, 1) e ϕ_f = step(time, 0, 0, 2, 0,16) all'interno del modello multibody, e gli stessi valori impartiti però attraverso blocchi rampa all'interno di Simulink.

Coordinata x del veicolo [m]

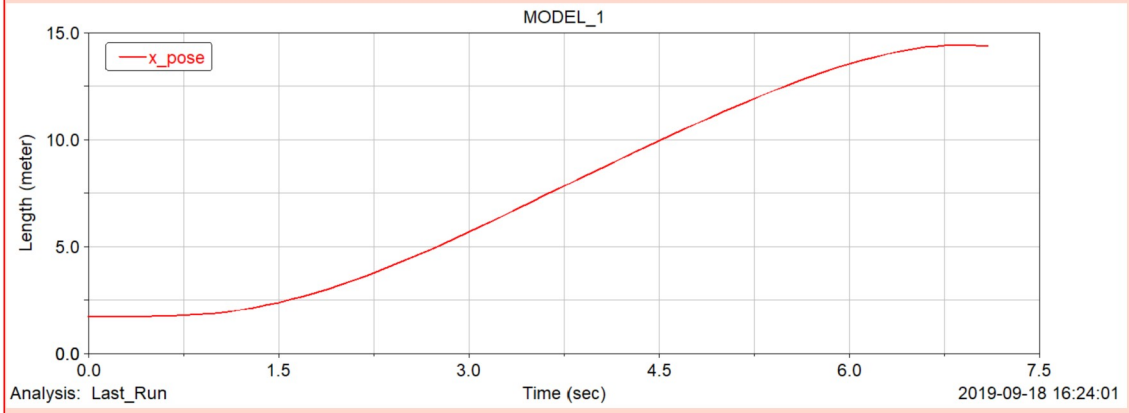


Figura 32: "Grafico Adams: coordinata x"

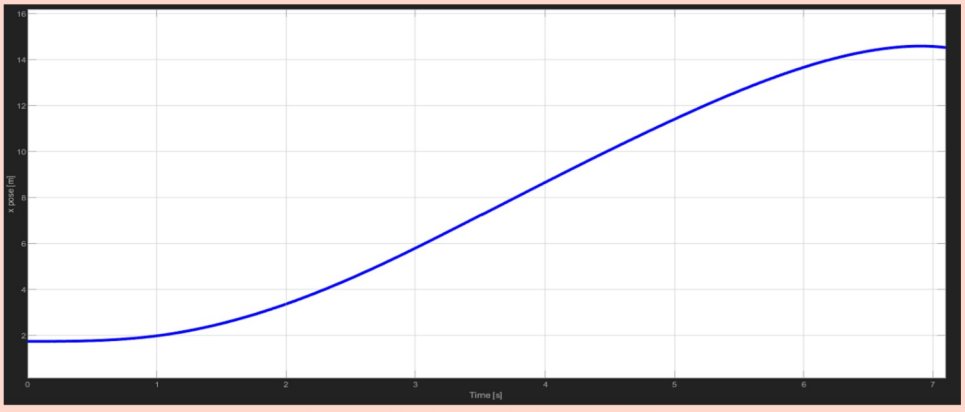


Figura 33: "Grafico Simulink: coordinata x"

Tabella 3: "Confronto della coordinata x del veicolo tra Adams e Simulink"

Coppia motrice alla ruota posteriore destra [Nm]

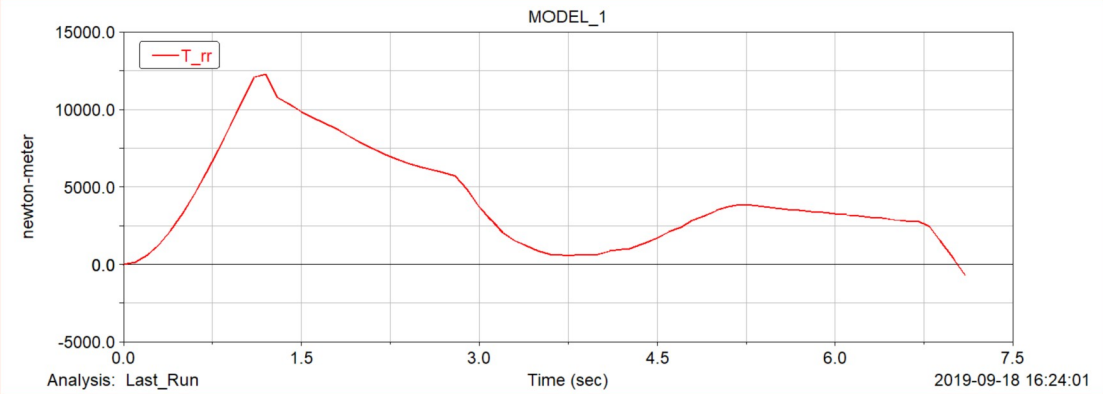


Figura 34: "Grafico Adams: coppia motrice alla ruota posteriore destra"

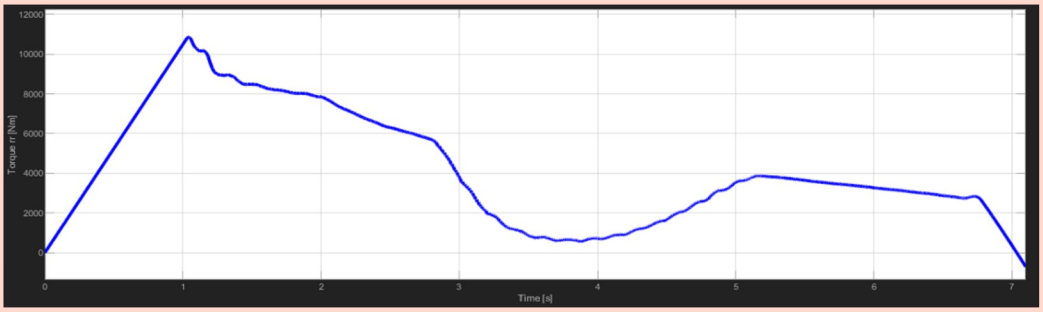


Figura 35: "Grafico Simulink: coppia motrice alla ruota posteriore destra"

Tabella 4: "Confronto della coppia motrice alla ruota posteriore destra tra Adams e Simulink"

Velocità angolare alla ruota posteriore destra [rad/s]

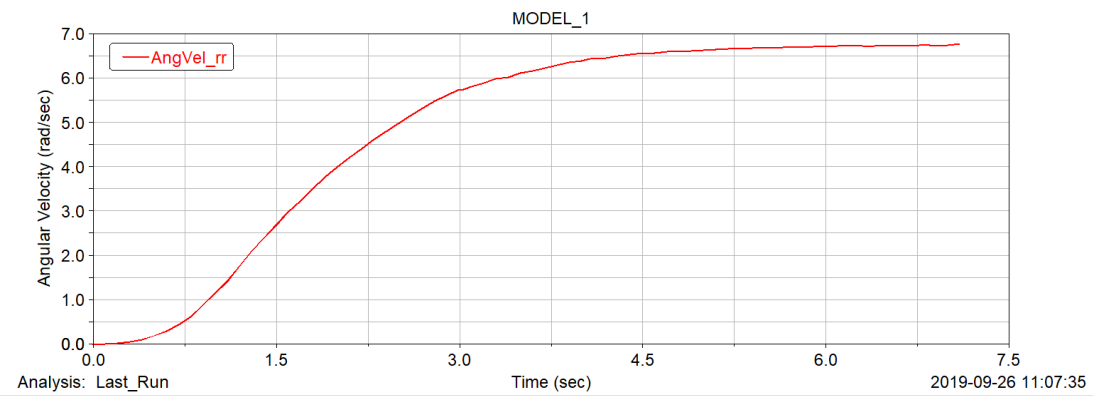


Figura 36: "Grafico Adams: velocità angolare alla ruota posteriore destra"

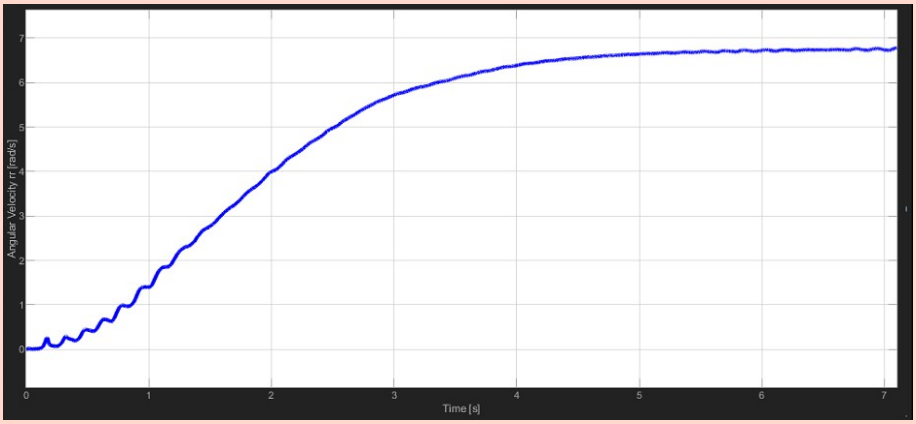


Figura 37: "Grafico Simulink: velocità angolare alla ruota posteriore destra"

Tabella 5: "Confronto della velocità angolare alla ruota posteriore destra tra Adams e Simulink"

Una volta aver verificato tale aspetto, l'attenzione è stata rivota ai parametri temporali della simulazione, osservando come questi influiscono sulla variabilità dei risultati precedenti. In particolare, lasciando le impostazioni di default in Simulink (Automatic solver, variable step), un fattore determinante si è rivelato essere il *communication interval* impostato nell'interfaccia del blocco 'ADAMS Plant', piuttosto che lo step size di integrazione impostato nel file .cmd derivante dall'esportazione del modello Adams. Infatti, diminuendo quest'ultimo rispetto al suo valore predefinito (0.1) non è stato possibile rilevare dei cambiamenti apprezzabili nella simulazione, neppure per valori dell'ordine del millesimo. Con un intervallo di comunicazione ridotto invece, si ottiene tipicamente un migliore riscontro a scapito di una maggiore durata della simulazione; è bene tuttavia non diminuire eccessivamente tale fattore per non incorrere in problematiche legate all'integrazione.

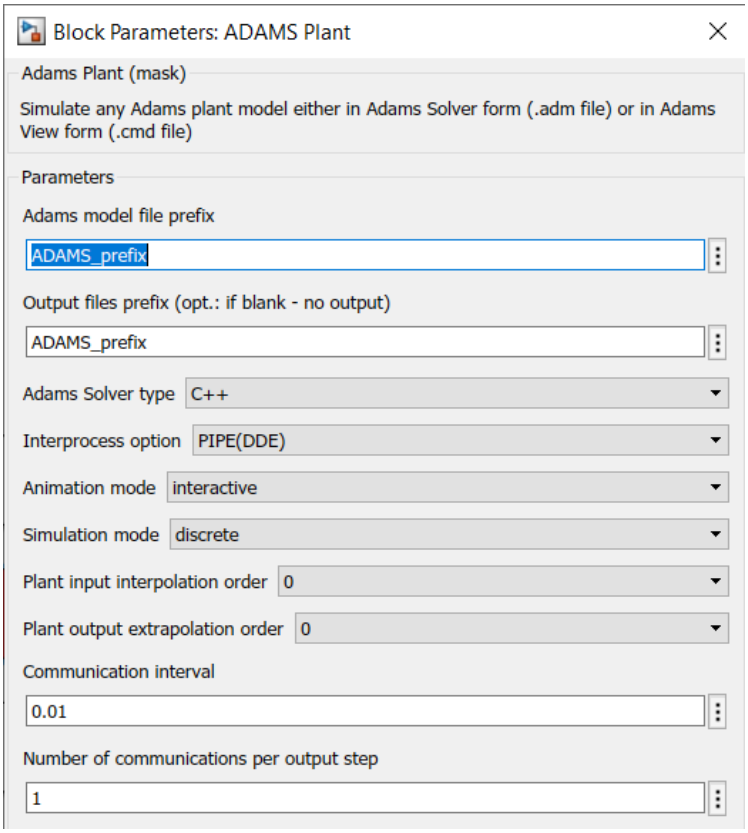


Figura 38: "Interfaccia del sottosistema 'Adams_Plant'"

Incidenza del *communication interval*

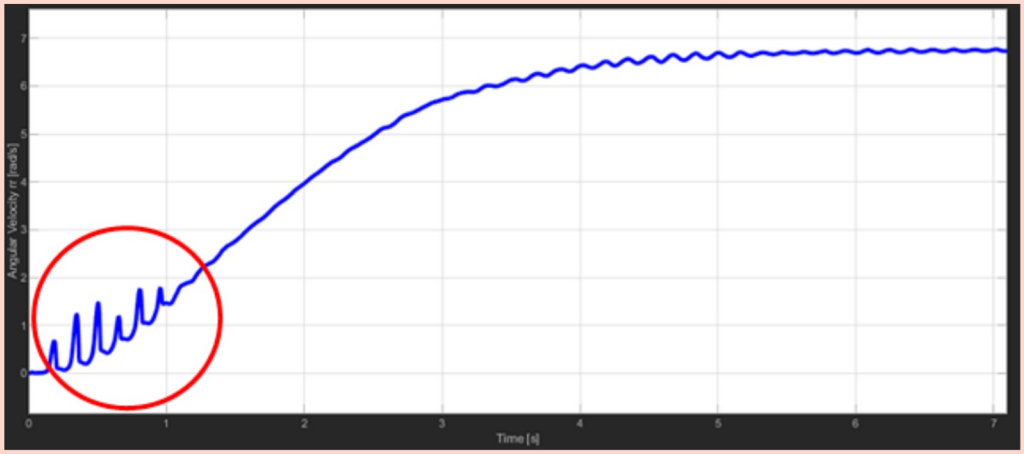


Figura 39: "Velocità angolare alla ruota posteriore destra con communication interval pari a 0,004"

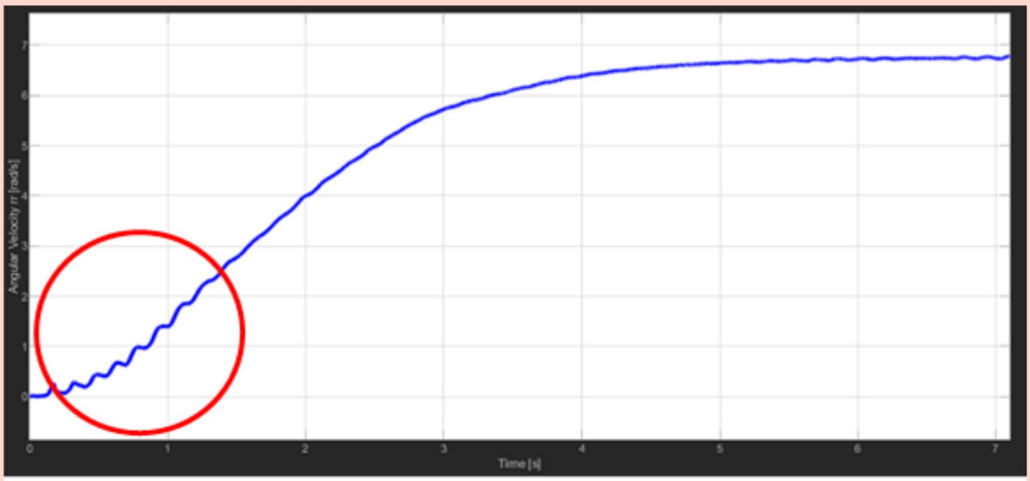


Figura 40: "Velocità angolare alla ruota posteriore destra con communication interval pari 0,001"

Tabella 6: "Andamenti della velocità angolare con diversi valori del communication interval"

4.2.3 Co-simulazione in anello chiuso: Waypoint Tracking Logic

Il passaggio alla co-simulazione in retroazione presuppone dunque che i comandi non siano più impartiti mediante funzioni specifiche, ma attraverso la logica di controllo ricavata precedentemente (Fig. 17 e 18). Con riferimento alla Figura 29, i parametri di feedback sono sempre la posa corrente del veicolo e la sua velocità lineare, rilevati in corrispondenza del suo centro di massa.
La strategia elaborata utilizza un approccio di tipo vettoriale per ricavare v e ϕ_f; tra un waypoint e il successivo l'angolo di sterzata è pari alla differenza tra la direzione globale (i.e. nel sistema di riferimento X-Y) di avanzamento e l'orientazione globale del mezzo. La prima viene calcolata appunto in base alla distanza x-y tra due pose consecutive, mentre la seconda è una variabile in uscita dal modello Adams. Per la velocità longitudinale si assume in partenza un valore di regime, mentre il veicolo viene progressivamente rallentato non appena si trova entro una certa distanza dal punto di arrivo.

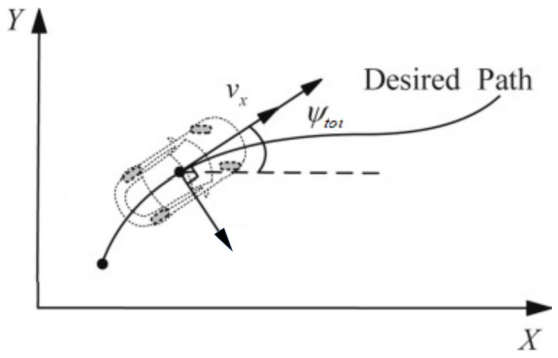


Figura 41: "Schema del veicolo in traiettoria"

Il sistema complessivo così definito però, può comportare il rischio di instabilità e perdita di controllo del veicolo, dovuto ad esempio ad oscillazioni anomale dell'angolo di sterzata. Per evitare fenomeni di questo tipo è stato opportuno ridurre innanzi tutto il numero di waypoints che approssimano la traiettoria ideale e renderli per quanto possibile equidistanti.

Riduzione del numero di *waypoints*

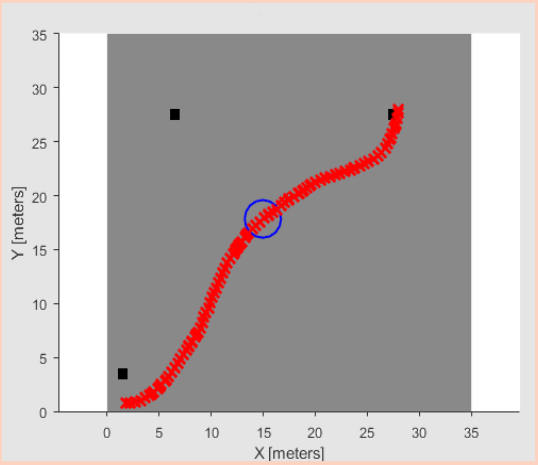


Figura 42: "Traiettoria con numero di waypoints invariato"

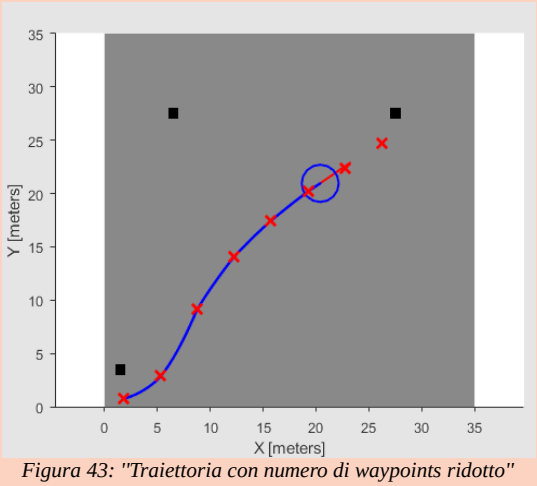


Figura 43: "Traiettoria con numero di waypoints ridotto"

Tabella 7: "Approssimazione della traiettoria con un ridotto numero di waypoints"

Inoltre è stata modificata la modalità di iterazione dell'indice che determina il waypoint di riferimento; in particolare, il passaggio da un punto al successivo viene determinato sulla base della minima distanza rispetto a dove si trova il veicolo.

Il segnale di velocità in uscita dalla 'Waypoint Tracking Logic', costituisce di fatto il riferimento del controllo PI: in base all'errore rispetto alla velocità lineare in uscita da Adams, esso ricava il parametro α che viene dato in ingresso al modello multibody e che regola la coppia alle ruote posteriori.

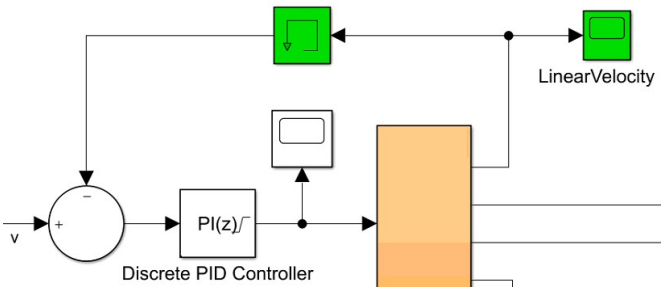


Figura 44: "Controllo PI'

Conclusioni

Risultati e sviluppi futuri

Come si può notare dalle Figure 40 e 41, i risultati della co-simulazione mettono comunque in evidenza un buon riscontro della traiettoria di partenza, nonostante la riduzione del numero di punti.
Inoltre, grazie alla modalità di iterazione dell'indice sui waypoints, il sistema non risente di problematiche di instabilità: la logica così definita lo rende di fatto intrinsecamente stabile, evitando forti oscillazioni soprattutto per l'angolo di sterzo nel tentativo di riportare il veicolo in traiettoria.
La durata della co-simulazione invece è legata ovviamente alla velocità di regime, che determina il tempo impiegato dal mezzo per raggiungere il punto di arrivo. Ma un altro fattore di incidenza si è rivelato essere, come già visto, il *communication interval*, ossia quanto spesso Adams e Simulink si scambiano informazioni. Per evitare inconvenienti durante la co-simulazione è bene che per il caso in esame esso sia all'incirca 0,001, comportando comunque una durata accettabile.

In futuro saranno dunque possibili ulteriori sviluppi, riguardanti ad esempio l'integrazione di una strategia di obstacle avoidance anche in fase di co-simulazione. Essa opererà in parallelo e in real time grazie a sensori virtuali creati all'interno di Adams tramite la misura delle grandezze di interesse, come ad esempio le distanze dagli ostacoli e le corrispondenti orientazioni. In base al tipo di applicazione prevista, sarà inoltre opportuno definire una modalità di arresto del mezzo fissando una distanza di sicurezza dal punto finale.
Raggiunta la posa obbiettivo, si può infine pensare di rendere autonome o semiautonome anche le operazioni successive.

Bibliografia

[1] A. Swingler and Ashleigh, “A Cell Decomposition Approach to Robotic Trajectory Planning via Disjunctive Programming,” 2012.

[2] M. Zohaib, M. Pasha, R. A. Riaz, N. Javaid, M. Ilahi, and R. D. Khan, “Control Strategies for Mobile Robot With Obstacle Avoidance,” 2013.

[3] S. Vigna, “L’algoritmo di Dijkstra,” 2006.

[4] M. Piastra, “Intelligenza Artificiale-AA Intelligenza Artificiale Ricerca euristica Algoritmo A*,” 2005.

[5] “Intelligenza artificiale: algoritmo A*.” [Online]. Available: https://engineering.facile.it/blog/ita/intelligenza-artificiale-algoritmo-a-star/. [Accessed: 15-Sep-2019].

[6] I. Industriale, “Politecnico di Milano Sviluppo di algoritmi innovativi e soluzioni flessibili per la generazione automatica di traiettorie robot per applicazioni industriali,” 2013.

[7] P. Milano and I. Industriale, “Confronto di algoritmi di pianificazione di traiettoria sampling-based per robot mobili,” 2018.

[8] “Configure RRT* path planner - MATLAB - MathWorks Italia.” [Online]. Available: https://it.mathworks.com/help/driving/ref/pathplannerrrt.html. [Accessed: 15-Sep-2019].

[9] “Pure Pursuit Controller - MATLAB & Simulink - MathWorks Italia.” [Online]. Available: https://it.mathworks.com/help/robotics/ug/pure-pursuit-controller.html. [Accessed: 15-Sep-2019].

[10] “Implement control logic with finite state machine - Simulink - MathWorks Italia.” [Online]. Available: https://it.mathworks.com/help/stateflow/ref/chart.html. [Accessed: 15-Sep-2019].

[11] “Mobile Robotics Simulation Toolbox - File Exchange - MATLAB Central.” [Online]. Available: https://it.mathworks.com/matlabcentral/fileexchange/66586-mobile-robotics-simulation-toolbox. [Accessed: 15-Sep-2019].

[12] A. Topiwala, “Modeling and Simulation of a Differential Drive Mobile Robot,” *Int. J. Sci. Eng. Res.*, vol. 7, p. 2016, 2016.

[13] D. Wang and F. Qi, “Trajectory Planning for a Four-Wheel-Steering Vehicle,” 2001.

[14] R. L. Norton, “Adams Tutorial Kit for Mechanical Engineering Courses,” pp. 1–352, 2013.