Report Subacquei, Gruppo A1
Esplorazione fondale

Canonico Martina, Scumaci Cristina, Vollaro Simone

June 2020

1 Introduction

La nostra missione consiste nell'esplorazione di un'area con fondale irregolare: si tratta di una survey di tipo lawn-mower e il task di esecuzione prevede che il veicolo debba mantenere una velocità di crociera costante, effettuando l'esplorazione, con controllo sull'altitudine (distanza del veicolo rispetto al fondale), che deve essere appunto anch'essa mantenuta costante.
Il compito del nostro modulo è stato quello di definire istante per istante lo stato della missione (operazione elementare) attualmente in corso; l'aggiornamento di tale stato è stato effettuato sulla base del monitoraggio di alcune variabili, nel nostro caso di blocchi di memoria, chiamati *"current-state"*, di visibilità globale, in maniera tale che tutti i moduli avessero l'informazione dello stato attuale e potessero adeguare il proprio comportamento per l'effettiva esecuzione del relativo sotto-task generato nel corso della survey.
In quanto modulo di references generator, il nostro compito è stato anche quello di generare i segnali di riferimento, relativi a posizione, orientazione e velocità delle varie operazioni elementari all'interno dei vari task, per il corretto svolgimento della missione.

2 Segnali e variabili

L'output dei riferimenti del nostro blocco, di ingresso al modulo di controllo si comporrà di:

1: Posizione desiderata [3x1, m, °];
2: Orientazione desiderata [3x1, rad];
3: Velocità desiderata [6x1, m/s, rad/s];

Le indicazioni di cui avremo bisogno, dunque di ingresso al nostro blocco saranno:

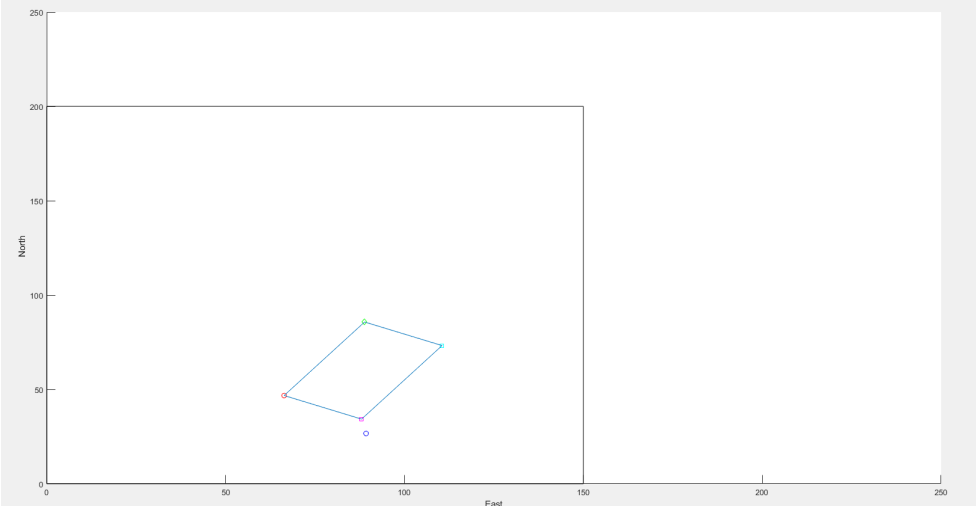
1: Posizione corrente [3x1, °, m];
2: Orientazione corrente [3x1, rad];
3: Velocità corrente [6x1, m/s, rad/s];
4: Altitudine corrente [1x1, m];
5: Slunt range [1x1, m].

Genereremo inoltre un flag di *"current state"*, per evidenziare le operazioni in corso, che sarà poi condiviso con tutti gli altri moduli.

positioning	Posizionamento iniziale in superficie
diving	Immersione per raggiungere il punto prestabilito
transect	Percorrenza tratto rettilineo per scansione
turn	Virata per cambio transetto
surfacing	Emersione
warning	Gestione disallineamento dalla traiettoria
obstacle	Gestione presenza di ostacoli compromettenti la missione
abort	Failsafe per danni strutturali o situazioni impreviste

2.1 Macrotask 'nominali' della missione:

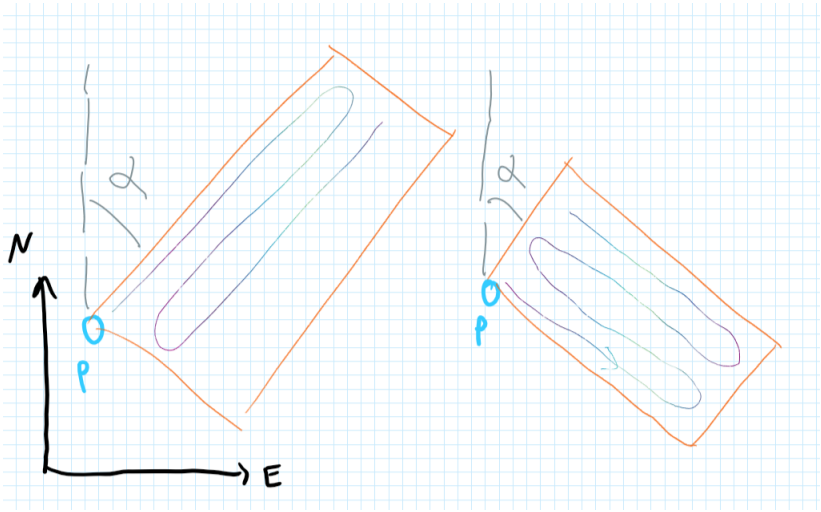
A partire dalla figura sopra riportata, il nostro approccio alla missione, per quanto riguarda la fase di inizializzazione e pianificazione del task di esplorazione del fondale, è stato quello di suddividere le varie fasi in una successione di 'macro-task', che il veicolo porta a termine, eseguendo una serie di operazioni elementari per ognuno di essi.



Il primo task pianificato è stato quello del *positioning:* nel file missionA.m si comunicano:
le coordinate in Latitudine e Longitudine dell'AreaOfInterestCorner, che ci indica la posizione di uno degli angoli dell'area di interesse totale su cui può essere effettuata la missione, che rimane sempre invariato; il survey areaCorner, che è il punto di coordinate dell'angolo corrispettivo di inizio survey, però in superficie e le coordinate del punto di deploy, InitPoint, che è il punto di rilascio in superficie del nostro veicolo; questi ultimi invece parametrici, poichè la sotto area di interesse su cui si vuole fare la survey, così come il punto in cui viene ad essere rilasciato l'auv, possono essere decisamente variabili, dunque parametrici.

La pianificazione di questo task consiste nella generazione dei riferimenti per il posizionamento del veicolo nel punto di survey area corner a partire dal punto di deploy.

La fase di 'diving' consiste semplicemente nell'immersione del veicolo dal punto di surveyAreacorner, al punto di inizio della vera e propria survey di esplorazione del fondale, tramite la generazione di riferimenti, che vanno a modificare solo la componente nel vettore posizione di profondità, a seconda dell'altitudine da mantenere rispetto al fondale richiesta, parametro anch'esso variabile che ci viene sempre passato dal file missionA.m. Una volta completato questo task, con controllo dell'altitudine, inizia la vera e propria survey che si suddivide in due macrotask: *'transect'* e *'turn'*, che rappresentano rispettivamente la fase di transettatura e di virata.
La nostra decisione è stata quella di effettuare la survey facendo muovere il veicolo sempre lungo il lato maggiore, sia esso quello in direzione est o in direzione nord; i due casi sono evidenziati sotto in figura:

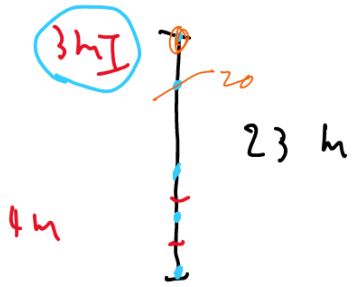


sono stati diversi i pro e i contro su cui si è ragionato per effettuare questa scelta, al seguire della tale siamo riusciti ad implementare dei ragionevoli compromessi: la scelta del lato lungo per la survey è stata dettata dal fatto che in questa maniera avremmo potuto effettuare una survey più veloce, in quanto avremmo diminuito il numero di virate che il veicolo avrebbe dovuto compiere; le virate sono le traiettorie su cui il veicolo deve per ragioni logico-implementative, rallentare (cause: mantenimento in traiettoria più difficile da eseguire, forma allungata del veicolo, più 'debole' sullo sway, presenza di correnti sfavorevoli) quindi, con questa scelta, ne abbiamo largamente ridotto il numero, risparmiando di conseguenza del tempo; il problema che però ci si era presentato in una seconda analisi è stato il fatto che però, a maggior ragione lungo il lato più lungo, avremmo avuto maggiori difficoltà per il mantenimento del veicolo in traiettoria, tutto questo perchè in una primissima fase di pianificazione iniziale pensavamo di dare, per ogni transetto da compiere, solamente i riferimenti del punto iniziale e finale, andandoci a perdere però il controllo del comportamento del veicolo lungo tutta la fase di transettatura, cosa che ovviamente non ci si poteva permettere. Da qui, iniziando un ragionamento sulla strategia di controllo da attuare, col modulo di controllo, siamo arrivati ad un valido compromesso, di cui detto sopra: il controllo è di tipo path following, dunque noi, lungo ogni transettatura, diamo dei midwaypoints di riferimento di posizione, ad una distanza, appurata ragionevole, di 2.50 m, ossia poco più della lunghezza del veicolo. Il veicolo, ogni qualvolta si sarà avvicinato all'i-esimo waypoint da raggiungere, entro un certo margine d'errore, riceverà il successivo riferimento di waypoint da raggiungere e così via sino alla fine del transetto. In questa maniera, il problema che si era precedentemente presentato, di non mantenimento della traiettoria, dando solamente punto iniziale e finale, specialmente nel caso di transettatura lungo il lato maggiore, quello da noi scelto, è stato risolto.

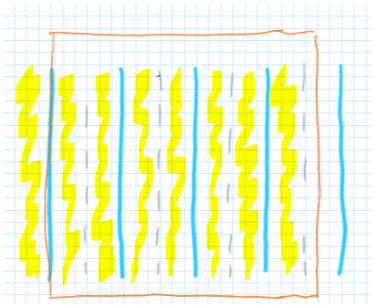
La parte implementativa riguardante il codice di generazione dei midwaypoints lungo il transetto è rimasta invariata anche durante le fasi di integrazione con gli altri moduli: una volta preso in considerazione il lato maggiore di scansione, si generano offline, i waypoints a distanza di 2.50 m l'uno dall'altro e per ognuno di essi, i valori di riferimento in posizione, vengono salvati tramite l'utilizzo in Matlab di 'page', che sono array multidimensionali. Ogni riga all'interno di questa page rappresenta il riferimento di posizione dell'i-esimo waypoint e le 3 colonne per ogni riga rappresentano le 3 componenti su x, y e z, del vettore posizione. Quindi il numero di righe di ogni pagina, rappresenta il numero di midwaypoints di riferimento generati lungo ogni transetto; il numero di pagine rappresenta il numero di transettature da effettuare che ovviamente varierà in base alla lunghezza del lato con lungehzza minore e al valore del *lineSpaceBetweenTransect*, valore sempre passato dal file MissionA.m, che indica la distanza laterale tra un transetto e il successivo.

`Piccoli dettagli aggiuntivi`

Piccoli accorgimenti, valutati nella fase di implementazione del codice sono stati: nel caso in cui il lato di transettatura non fosse divisibile senza resto rispetto alla distanza di 2.50 m decisa, si dà comunque come ultimo punto di riferimento, un punto aggiuntivo, che è ovviamente il punto giacente esattamente alla distanza pari alla lunghezza del lato maggiore, che è il punto finale del transetto coincidente con l'inizio della virata; un esempio è mostrato nella figura sotto:



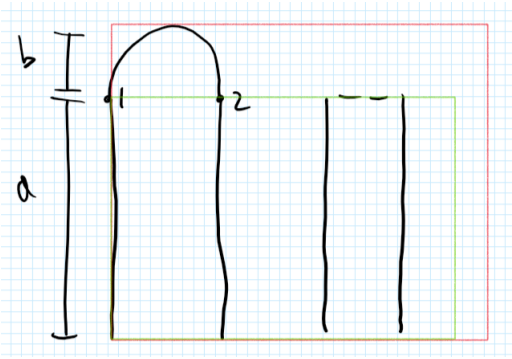
Un altro accorgimento, aggiunto sempre nella fase di codice è stato quello di considerare questo fatto: il numero di transetti che si compiono via via, dipende dalla distanza orizzontale tra l'uno e l'altro, *lineSpaceBetweeenTransect*. A seconda del valore di questo parametro è possibile che l'ultimo transetto che magari deve esser compiuto si trovi al di fuori dei margini dell'area su cui effettuare la survey:



l'accorgimento che abbiamo preso è stato quello di aggiungere una transettatura ulteriore, solamente nel caso in cui la distanza tra transetti rimasta scoperta però entro l'interno dell'area della nostra survey, fosse più grande della metà di questa lunghezza indicata dal *linseSpaceBetweenTransect*, perchè questo significava lasciare 'inesplorata' un'altra buona parte di fondale; nel caso in cui la

traiettoria del transetto finale fosse abbastanza vicina alla linea parallela, quella del lato minore, di 'fine survey', lasciamo la generazione dei transetti invariata, poichè si andrebbe a perdere l'esplorazione di una relativamente poca parte del fondale.

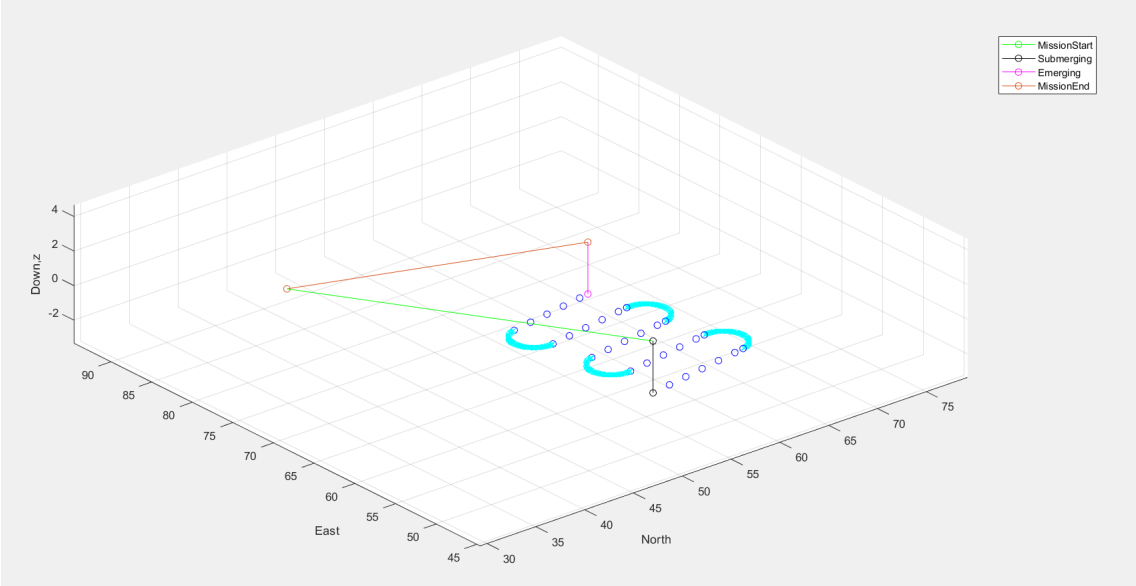
L'ultima considerazione da sottolineare, per quanto riguarda l'esecuzione di questi task di *transect* e *turn*, di esplorazione del fondale, è stata la decisione presa in un secondo momento di compiere le virate al di fuori dell'area di survey; decisione dettata dal fatto che, dal momento che durante le virate solitamente non si prendono in considerazione le misure dei sonar, per evitare di perdere ulteriori informazioni di quella parte di fondale 'ricoperta' dalla manovra di virata all'interno della nostra area di survey, abbiamo preferito lasciare tutte queste manovre al di fuori dell'area di esplorazione effettiva.



Anche per quanto riguarda la fase di virata, si era pensato un controllo di tipo path folllowing tramite la generazione di riferimenti di posa di midwaypoints intermedi, 'alla maniera dei transetti', sempre in modalità offline, memorizzati in page, in cui per ogni riga è presente il riferimento dell'i-esimo midwaypoint corrispondente ed il numero di page corrisponde al numero di virate; l'unico accorgimento preso in considerazione sta nel fatto che abbiamo dovuto aggiungere un criterio di differenziazione per i casi di lato transettatura lungo, su nord e, lato transettatura lungo, su est; nel primo caso le virate iniziano effettuando una rotazione oraria, partendo dall'angolo α di orientazione rispetto al Nord, passato nel file MissionA.m e nel secondo caso, la prima virata, avviene invece in senso antiorario e l'orientazione iniziale è quella di α gradi rispetto ad est, quindi $90° + \alpha$ rispetto al nord.
A differenza di quanto avvenuto per i midwaypoints lungo i transetti, il cui codice generato offline è stato poi utillizzato nella survey, nella fase d'integrazione finale, questo 'approccio' di generazione offline dei midwaypoints in virata è stato scartato, per delle motivazioni che saranno spiegate in seguito.

Arrivati a questo punto, terminate le fasi di esplorazione della survey, se tutto è quindi andato a buon fine, avviene il *surfacing*, che è il task di riemersione

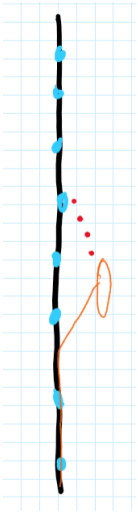
dal punto di fine survey, andando a modificare ovviamente solo la componente lungo z del riferimento in posizione, a 0.
L'ultima fase che coincide con la prima, a livello di logica implementativa, è la fase di *positioning*, in cui il veicolo si dirige di nuovo nel punto di deploy iniziale. La missione è dunque conclusa.



2.2 Macrotask d'emergenza della missione:

Queste sopraelencate rappresentano i task nominali della missione, nel caso in cui non si verifica nessuna delle situazioni che noi abbiamo indicato come situazioni d' 'emergenza'.

Gli altri task che rappresentano appunto queste situazioni di emergenza sono: *warning*: questo task si attiva solamente se il veicolo si trova disallineato rispetto alla traiettoria da seguire oltre un certo margine di errore, che si è ritenuto accettabile lungo il moto di surge e sway. La logica di base è che il veicolo ritorni seguendo un moto traslazionale lungo la traiettoria nominale, dando come successivo waypoint di riferimento non quello presente subito dopo, ma quello switchato di due punti in avanti, lasciando un margine minimo di almeno 5m in avanti al veicolo, per potersi riportare in traiettoria.



L'altra situazione di emergenza è rappresentata dal task *obstacle*: in questo task vengono generati i riferimenti opportuni in profondità, da dare al veicolo, qualora fosse presente un ostacolo ritenuto tale, nel senso di variazione di fondale rilevante, quando la sua elevazione dal fondale è maggiore di 0.5 m e la sua prolungazione orizzontale supera i 2m, in maniera tale da mantenere il controllo ad altitudine sempre costante.
L'ultima emergenza è definita dal task *abort*: questo task viene ad essere richiamato secondo più situazioni di emergenza molteplici: ad esempio, se l'ostacolo che si ha davanti non è superabile mantenendo il controllo di altitudine richiesta, allora non possiamo proseguire con la survey.
L'altra situazione per cui può verificarsi un *abort* è una situazione di 'failsafe' per danni strutturali, per cui se i motori del veicolo, per rottura o altro, rimangono 'spenti' per oltre un certo periodo di tempo, vuol dire che è avvenuta una

rottura, quindi dobbiamo per forza generare un *abort* ed interrompere la survey. Un'altra situazione di failsafe sono i danni strutturali che possono presentarsi ai sensori che disponiamo per l'ottenimento delle misure; nel caso in cui, infatti, otteniamo dei valori di posizione scostanti da quelli di riferimento oltre un certo margine, allora possiamo rilevare un danno strutturale ai sensori, perciò risulta inutile proseguire la survey.

2.3 Scelta e generazione dei current state

Il blocco di current state indica il task/i tasks di esecuzione corrente/i ed è implementato con un vettore numerico di 8 componenti che possono assumere solo valore 0 o 1. Il bit settato a 1, in una determinata posizione del vettore, indica ovviamente la condizione di task rispettivo attivo, 0, viceversa. I primi 5 valori del vettore rappresentano lo stato attivo/disattivo dei 5 task nominali, che il veicolo può trovarsi ad eseguire. Per questi primi 5 bit del vettore solo uno fra questi sarà settato ad 1, gli altri a 0, poichè il veicolo può trovarsi esclusivamente in ognuna di queste fasi, che ovviamente non possono verificarsi contemporaneamente. Gli ultimi 3 bit invece, sono quelli che indicano uno stato di emergenza e a differenza dei primi 5 bit, dal momento che le emergenze possono avvenire anche contemporaneamente, le ultime 3 componenti possono anche trovarsi tutte e 3 settate a valore 1 e sicuramente, dal momento che le situazioni di emergenza potranno verificarsi durante una qualsiasi esecuzione delle fasi nominali, succede sempre che la posizione del bit a valore 1, indicherà quale tra le emergenze si sta verificando, inoltre sarà sicuramente settato a 1, almeno uno dei bit facenti parte delle prime 5 posizioni del vettore, indicando in quale delle fasi principali si sta verificando la situazione di emergenza.

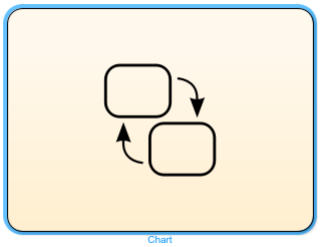
Numerazione dei flag in bit state = [positioning, diving, transect, turn, surfacing, `warning`, `obstacle`, `abort`], in grassetto gli stati di "emergenza".

Esempio: [0 0 1 0 0 0 1 0] siamo in *transect*, ma c'è da gestire un ostacolo, *obstacle*.

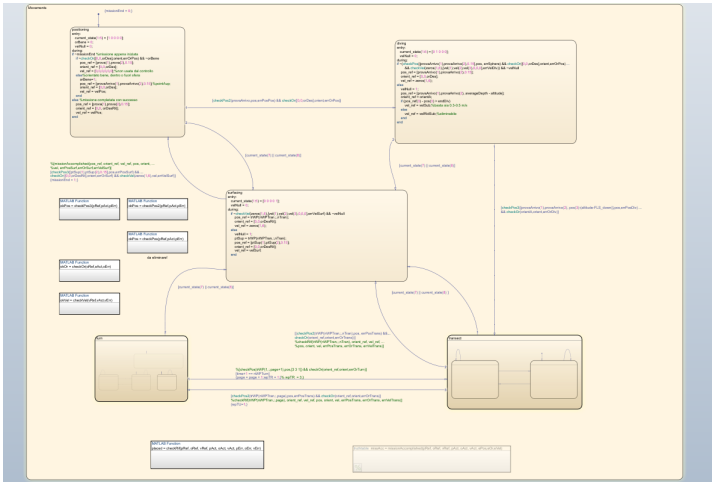
3 Breve introduzione a stateflow e spiegazione dell'implementazione della nostra macchina a stati:

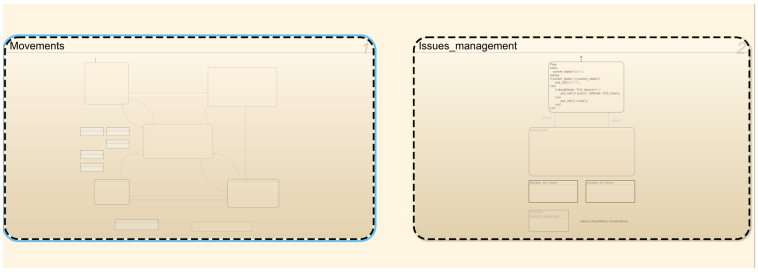
3.1 Introduzione a stateflow:

Stateflow è un toolbox di Matlab che permette la modellazione e la simulazione di macchine a stati e diagrammi di flusso. Stateflow viene integrato in uno schema di Simulink, con un blocco specifico, rappresentato in figura:

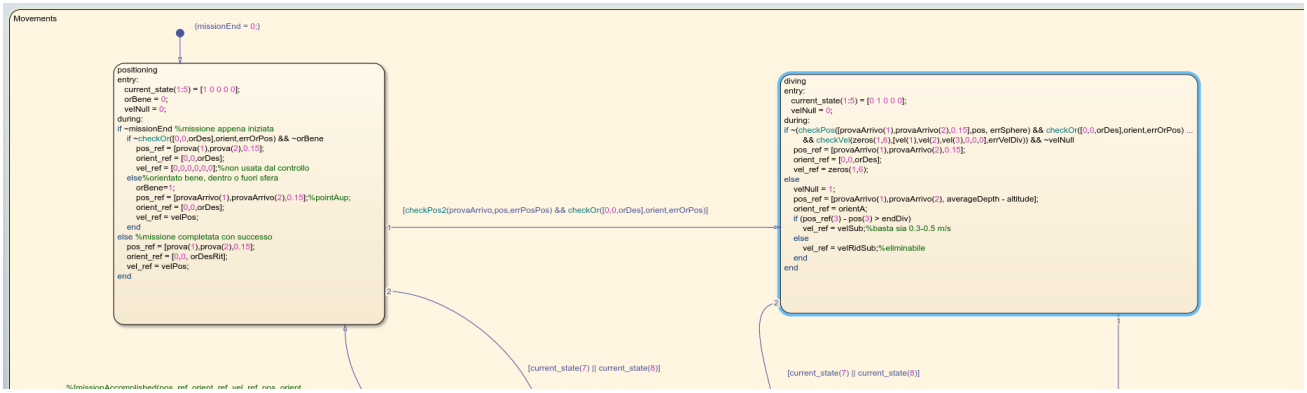


Gli stati esclusivi del sistema sono rappresentati con una linea continua; gli stati paralleli sono caratterizzati dalla linea tratteggiata. Per poter creare degli stati paralleli occorre modificare la proprietà "Decomposition" dopo aver cliccato sul tasto dx del mouse, scegliendo quindi il sottomenu "Parallel".





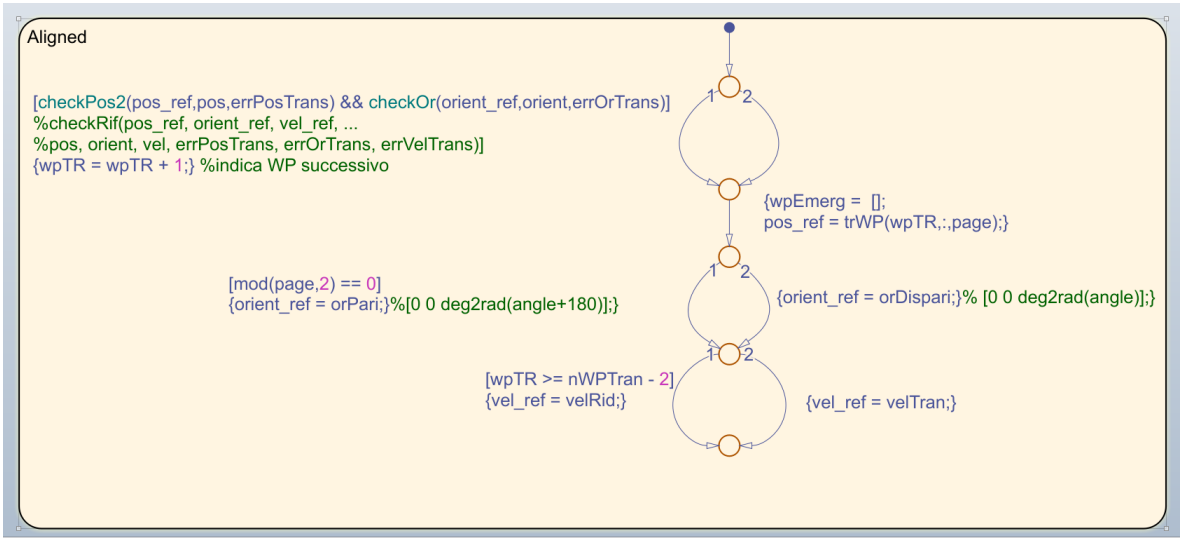
Come dicono le stesse parole, gli stati esclusivi, possono essere attivi in maniera esclusiva, quelli paralleli possono essere attivati ed eseguiti anche parallelamente.
Le transizioni sono rappresentate da frecce direzionate che vanno da uno stato ad un altro. Un caso particolare di transizione è quello della transizione di default in un sistema esclusivo:



La transizione di default indica fra tutti i blocchi esclusivi presenti all'interno di una Subchart ad esempio, quello in cui effettuare la prima transizione, quello in cui ci si trova non appena viene eseguita la macchina a stati, perchè come in ogni diagramma di flusso vi è un blocco di partenza e una data direzionalità, anche qui in Stateflow.

`Giunzioni`

Le giunzioni sono rappresentate da cerchi cui possono giungere più transizioni e rispettivamente partire nuove transizioni. Nel caso di più transizioni in uscita da una giunzione, viene determinata la sequenza di esecuzione, tramite una sorta di priorità, indicata con valori 1,2,3 ecc.



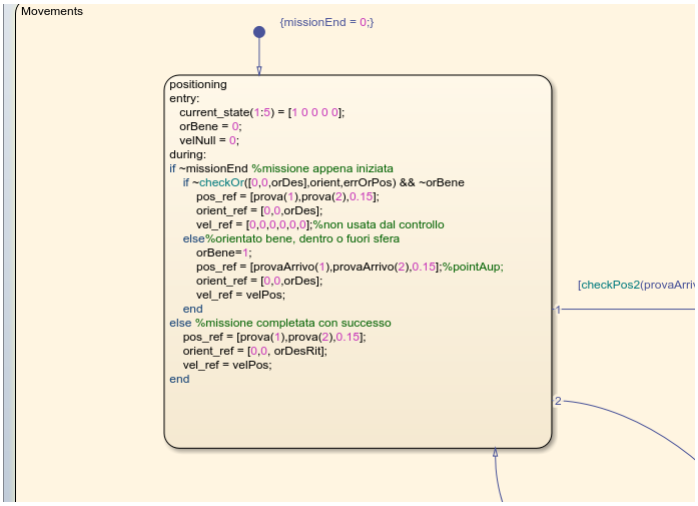
3.2 Tipi di azioni supportate per stati e transizioni

:

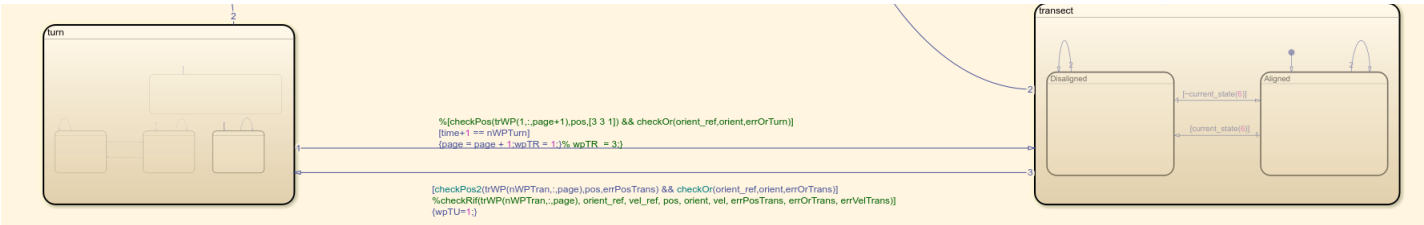
3.2.1 Tipi di azioni negli stati:

`entry, during, exit.`

Entry: esegue le operazioni quando lo stato diventa attivo.
During: esegue l'azione quando lo stato è attivo e arriva un evento specifico.
Exit: esegue le operazioni prime della transizione dallo stato.



3.2.2 Tipi di azioni sulle transizioni:



`Condition`
La condizione è un'operazione booleana che determina se la transizione può essere effettuata oppure no. Condizioni possono essere:
• espressioni booleane;

• funzioni che ritornano vero o falso;
• condizioni temporali.

Vengono inserite sulle transizioni tra parentesi quadre.

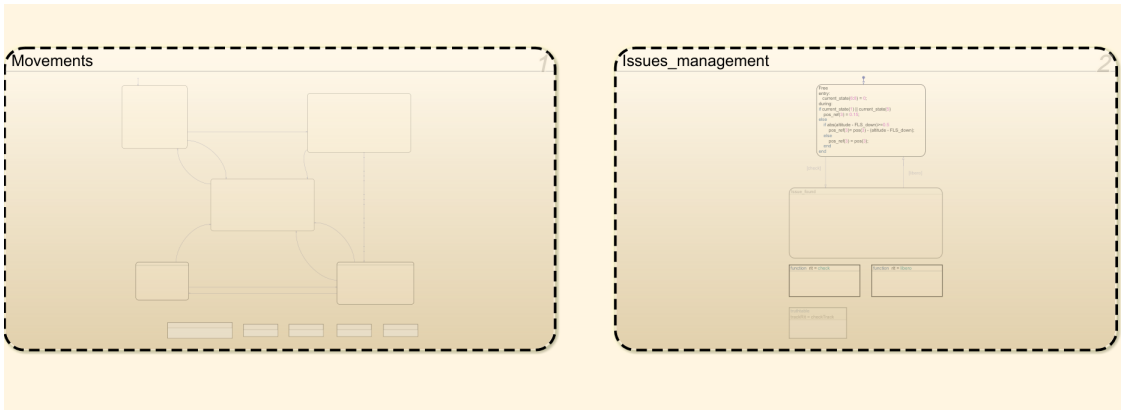
`Condition action:` sono operazioni messe tra due parentesi graffe e vengono eseguite se la condizione è vera, prima che la destinazione della transizione sia raggiunta (stato o giunzione). A differenza della transition action l'operazione viene eseguita anche se poi in seguito non sarà possibile effettuare la transizione e raggiungere un nuovo stato.

`Transition action`: queste azioni sono precedute da un segno / e possono essere eseguite solo se è possibile raggiungere la fine della transizione, altrimenti vengono ignorate.

All'interno della Chart in Stateflow sono presenti diversi strumenti per implementare tutto il codice dietro i diagrammi di flusso della macchina a stati, siano esse Matlab function, simulink function, graphical function, truthTable, il cui funzionamento sarà ripreso e chiarito nella successiva spiegazione dell'implementazione di tutto l'ambiente Stateflow, che sta alla base dello sviluppo dell'intero lavoro del nostro modulo.

4 Implementazione della nostra macchina a stati, con la spiegazione del codice e della relativa modalità d'implementazione passo-passo.

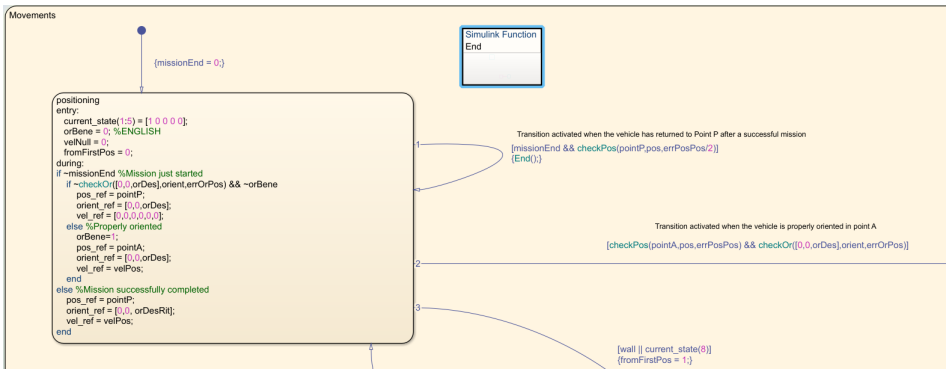
.
Dal momento che l'ambiente di stateflow è un ambiente nuovo, di cui non vi è ancora larga conoscenza, si coglie l'occasione, descrivendo la politica di gestione della nostra macchina a stati e del codice al suo interno, di rendere più chiaro e comprensibile il suo funzionamento e l'efficacia del suo utilizzo.



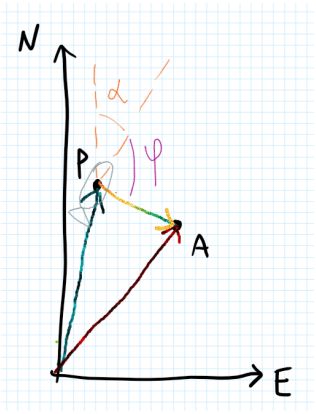
Come si può ben vedere nella figura sopra, la nostra macchina a stati comprende due Subcharts principali: quella dei `Movements` e quella degli `Issues_Management`. Le due Subcharts sono tratteggiate, questo significa che sono stati paralleli e che quindi, vengono 'attivati' contemporaneamente.

4.1 Discussione dell'implementazione del blocco Movements.

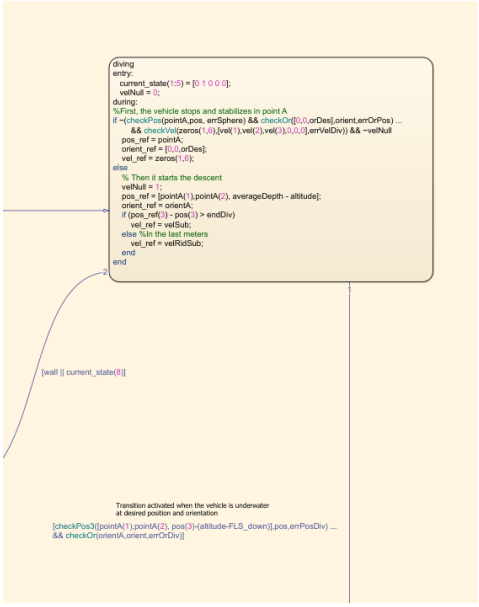
La Subchart dei Movements comprende i 5 task sopra descritti, esclusivi, definiti come nominali, che vengono ad essere richiamati settando a 1, uno dei primi 5 bit presenti nel vettore di 'current state'. La default transition all'interno della subchart si trova sullo stato 'positioning'; quindi sicuramente lo stato iniziale in cui si trova la nostra macchina a stati è quello di posizionamento; in contemporanea, nel blocco di Issues_management è presente una default transition sullo stato Free, che rappresenta l'assenza di issues, di situazioni di emergenza; appena si attiva l'intera macchina a stati, siamo negli stati positioning e free; la macchina a stati durante le fasi di transizione degli stati esclusivi sarà settata a free, sino a quando non si presenta una situazione di emergenza, in cui viene settato il current state col bit a 1, in posizione relativa al/i tipo/tipi di emergenza che stanno accadendo e avviene la transizione in uno o più di essi. Se entriamo nel blocco Movements possiamo osservare i 5 stati:



Il primo stato è `positioning`, caratterizzato dalla default transition, con una condition action che setta il valore di missione finita a 0; appena si entra nel blocco, quello che viene fatto è settare il current state al valore corretto, in questo caso 1 0 0 0 0 0 0 0. All'interno dello stato, il modulo di controllo effettua un check su orientazione e velocità, per questo, prima di entrare nel during, utilizziamo una sorta di variabili flag, che danno un ritorno 'positivo' o 'negativo' di orientazione e velocità, a seconda che queste, di riferimento, siano state raggiunte o meno. All'inizio siamo nel caso di orientazione e velocità desiderate non raggiunte perciò le variabili vengono settate a 0; nel during si effettua questo check: innanzitutto se il valore della condition action è diversa da 0, siamo in condizione di missione iniziata; il check mi indica che finchè il veicolo non si orienta bene e non arriva a farlo poi fermandosi, quindi arrivando a velocità nulla, continua a ricevere sempre gli stessi valori di posizione, orientazione e velocità, in questo caso la posizione di riferimento è quella dell'initPoint; una volta che il veicolo si è orientato bene e si è fermato, si può iniziare a muovere e dirigere dall'initPoint al punto in superficie di surveyAreaCorner. Per quanto riguarda i riferimenti di posizione e di orientazione, inizialmente si era pensato di dare solamente posizione e orientazione finali del veicolo, ossia come posizione, il punto in superficie di surveyAreacorner e l'orientazione di α gradi rispetto a Nord, da mantenere in fase di diving per iniziare successivamente la fase di transettatura. Per successivi problemi, in fase di implementazione, il controllo ha richiesto dei riferimenti che permettessero un controllo più 'discretizzato' per le singole manovre da compiere, sino alla fine della fase di posizionamento, del tipo: si è presupposto che il veicolo sia calato in acqua con orientazione nota, pari all'angolo α rispetto al Nord, passato dal file MissionA.m; conoscendo la distanza tra l'origine e il punto P (initPoint) e l'origine e il punto A (surveyAreacorner), ne facciamo la differenza dei vettori e quello che si vuole ottenere è sapere di quanto bisogna imbardare il veicolo affinchè si possa orientare lungo direzione e verso del vettore differenza OP-OA, valore che in figura è l'angolo γ.

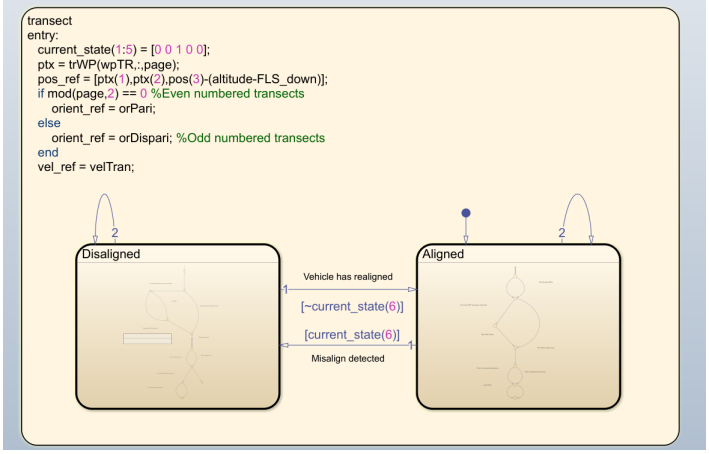


Avvenuta una corretta orientazione del veicolo, il veicolo si dirige nel punto in superficie di surveyAreaCorner; una volta che arriva all'interno della sfera di distanza di 1.50 m dal punto di immersione si effettua la transizione a diving:



In entry si setta il current state con bit 1, in posizione corretta, di `diving` e qui, prima di entrare nel during, nella vera e propria fase di immersione, si fa un check di velocità; come nello stato positioning utilizzo una variabile 'di flag': con un check: finchè il veicolo non arriva nel punto desiderato, in questo caso di immersione, entro un certo margine d'errore, sempre con orientazione desiderata entro un certo margine d'errore, con velocità nulla, non avviene il

cambio di riferimento in profondità, per iniziare la vera e propria immersione; questa situazione anch'essa scelta perchè vogliamo assicurarci che il veicolo sia ben posizionato e fermo sul punto di immersione; una volta che il check è andato a buon fine, si inizia l'immersione e l'orientazione del veicolo, si è deciso, in accordo col modulo di controllo, che avvenisse contemporaneamente nella fase di immersione, quindi viene passato il nuovo riferimento in posizione, con la nuova profondità e l'orientazione del veicolo, secondo quella desiderata, informazione passata tramite il valore del parametro α nel file missionA.m. A fine immersione il veicolo sà che deve fermarsi e attraverso un check di transizione, se arrivo nel punto di inizio survey con posizione e orientazione desiderate, entro sempre un certo margine d'errore, può iniziare la vera e propria survey, con fasi di transettature e virate; ovviamente prima che avvenga tutto ciò, il controllo si deve assicurare che il veicolo sia con velocità nulla sul punto di inizio survey.

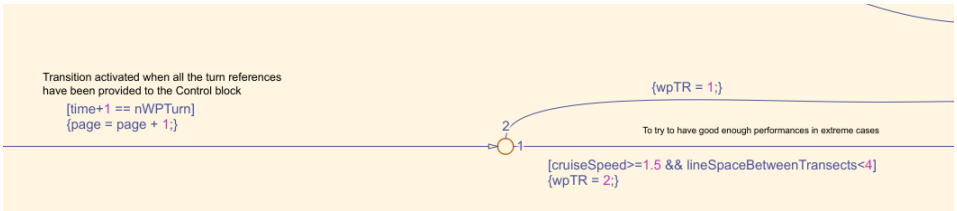


Nell'entry di `transect`, come in ogni stato, la prima cosa che si fa è settare il current state al bit corrente con riferimento in posizione del primo midwaypoint da raggiungere, ossia prima riga, tutte le colonne e il parametro page, numero corrispondente al relativo transetto da compiere; nel primo caso, il transetto è settato a prima riga, tutte le colonne e la pagina è settata a 1; questi parametri sono variabili già settati a 1 nel file initSupervisor.m:

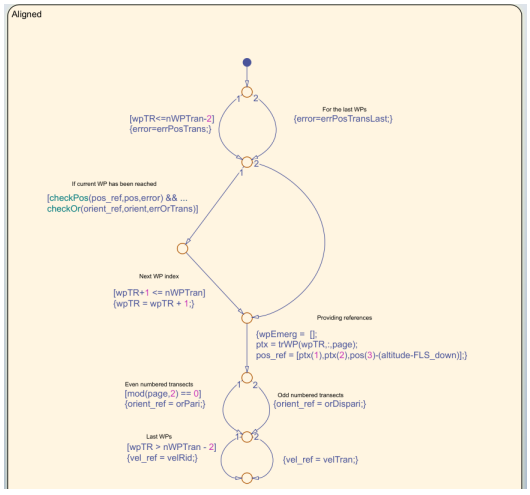
```
%parametri per indicizzazione matrici di waypoint
page = 1; %transetto/virata da compiere
wpTR = 1; %indice WP corrente sul transetto
wpTU = 1; %indice WP corrente in virata
```

Questo avviene solo nella prima transizione nello stato transect, dal blocco diving. Invece quando la transizione si ha da turn a transect, nell'entry, la posizione di riferimento sarà sempre la prima riga della pagina che però non

sarà più settata ad 1, ma ogni qualvolta si passa da turn a transect è presente la condition action che incrementa di 1, il numero di pagina, riferendosi così al transetto successivo, settando sempre a 1 la riga della page relativa.



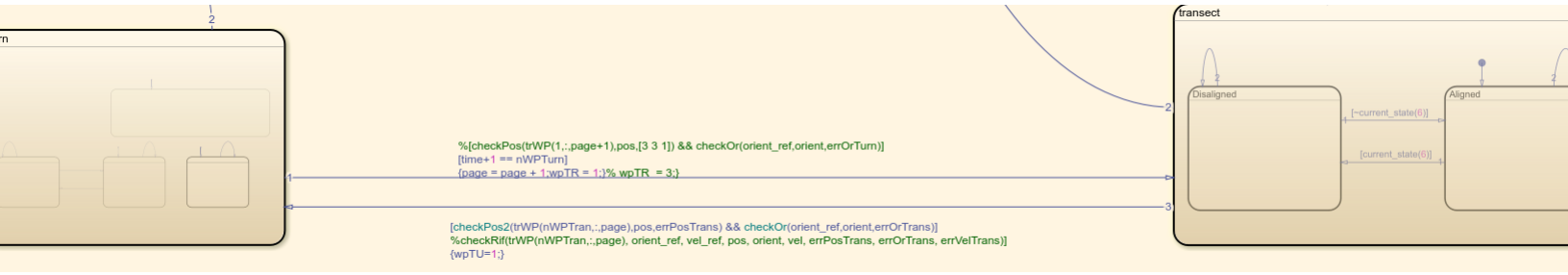
L'unico accorgimento preso durante questa transizione, da turn a transect, è stato nel caso in cui la velocità di crociera passata come parametro fosse superiore o uguale a 1.5 m/s e la distanza tra i transetti fosse inferiore ai 4m; in questo caso, per mantenere delle buone performance, il waypoint che si dà non è il primo del transetto, ma si parte già dal secondo, quindi wpTR (che indica l'indice riga della page, che rappresenta l'i-esimo midwaypoint da raggiungere) è settato a 2. Nell'entry di questo blocco diamo anche il riferimento di orientazione per distinguere i 2 casi: se ci troviamo su un numero di transetti dispari, il veicolo deve essere orientato con angolo di imbardata α rispetto al NED; sui transetti pari ovviamente il veicolo, oltre che dell'angolo α deve essere imbardato di altri 180°, poichè deve ritornare parallelamente, indietro.
Lungo il transetto, a seconda di un certo margine di errore su surge e sway ci si può trovare allineati, Aligned o disallineati, Disaligned. All'interno del blocco transect la default transition è settata su Aligned. La prima cosa di cui si fa il check appena entrati in Aligned è di essere davvero allineati:



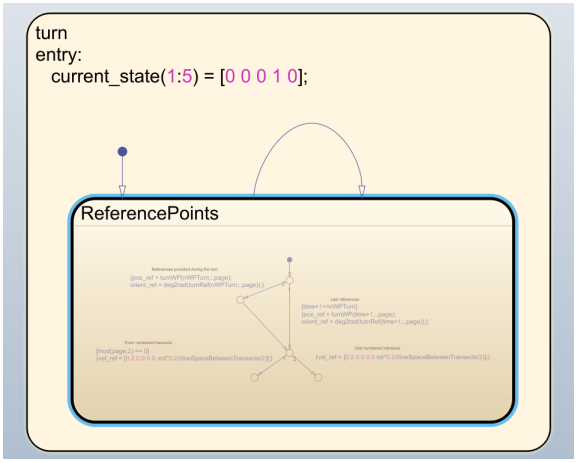
E' presente una check function su posizione e orientazione; di seguito le transizioni da effettuare all'interno del sottoblocco Aligned sono rappresentate tramite le cosiddette junctions, gerarchizzate: il primo step è quello di effettuare il check: finchè il check non è soddisfatto vuol dire che non si è raggiunti posizione e orientazione di riferimento entro un certo margine di errore, quindi quello che si fa è creare una matrice vuota di waypoint di emergenza nel caso si dovesse superare un certo margine di errore laterale che ci avvisa del disallineamento e dobbiamo creare un vettore con i waypoint di emergenza per riallineare il veicolo; fatto ciò si continua a dare lo stesso midwaypoint di riferimento, finchè il check non si verifica. Nel caso in cui invece il check dà esito positivo, finchè l'indice riga del waypoint successivo è inferiore o uguale al nWPTran, che è la size delle righe della page dei waypoint, allora quello che si farà sarà passare il waypoint di riferimento successivo, quindi s'incrementa la riga della relativa page di 1 e si va avanti; realizzato uno dei due casi, indicati come 1 o 2 in figura, si ha una transizione con una nuova condition, sull'orientazione: se il transetto è dispari, l'orientazione sarà una rotazione su z, di α, altrimenti di $180° + \alpha$. Infine una sorta di ultimo check di velocità: la prima cosa che si controlla è: se l'indice riga del waypoint corrisponde alla terzultima riga, quindi terzultimo midwaypoint di riferimento, saremo a circa 5 m di distanza dall'ultimo waypoint prima di inizare la virata, quindi quello che si dice al veicolo è di rallentare e di arrivare dalla velocità di crociera ad una velocità di virata, scelta da noi come ragionevole di 0.2 m/s; si è scelto un margine di frenata di 5m, poichè essendo la velocità di crociera parametrica, vogliamo esser sicuri che sia se arriviamo con una velocità di 0.5 m/s, sia se arriviamo con un eventuale velocità di 2m/s, lo spazio di frenatura sia tale da permettere al veicolo di arrivare in tempo in curva con la velocità ridotta desiderata; la seconda opzione nel caso in cui non ci troviamo sugli ultimi 3 midwaypoints del transetto è quella di dare ancora come velocità di riferimento, quella di crociera e proseguire.

Così come per i riferimenti all'interno del transetto, che per la funzione di transizione da transetto a virata, le funzioni checkPos e checkOr sono le stesse, nel senso che settate con stessi parametri, ma diversi margini di errore e sarà spiegato successivamente il perchè. Per quanto riguarda la funzione di checkPos nella fase di transettatura accetta un margine di errore su surge e sway di 1 m; questo significa che, all'interno della transettatura, quando distiamo meno di 1 m dal waypoint di riferimento, diamo già il successivo da dover poi raggiungere; a livello di transizione vuol dire che, allo stesso modo, appena entriamo nella 'sfera' d'influenza di 0.5 m dell'ultimo waypoint prima della virata, avvisiamo già ed effettuiamo la transizione a virata e diamo come riferimento il primo waypoint di virata. In questi ultimi waypoint il margine sulle coordinate di surge e sway da rispettare è ridotto a 0.50 m, poichè vogliamo entrare già ben posizionati e orientati in curva. Anche per quanto riguarda l'orientazione, lungo la transettatura, così come nella transizione a virata, ci manteniamo in margini più larghi di errore di roll e pitch, circa 15°; d'imbardata, dal momento che vogliamo esser molto precisi, per non rischiare di fare una transettatura del tutto non allineata, nel senso che proprio con un'altra direzione di orientazione

rispetto a quella nominale, abbiamo deciso di dare un margine di errore di 10°. Una volta verificatasi la condition di transizione di checkPos e checkOr, viene eseguita la transition action, tra parentesi graffe, che setta la variabile wpTU=1, che è l'indice della prima riga della page del waypoint di riferimento in virata.

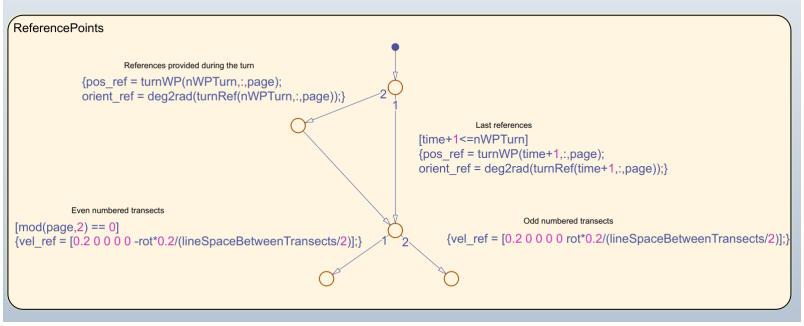


Arrivati nel blocco `turn`, la prima cosa che si fa nell'entry è ovviamente settare il bit del current state in posizione di turn.

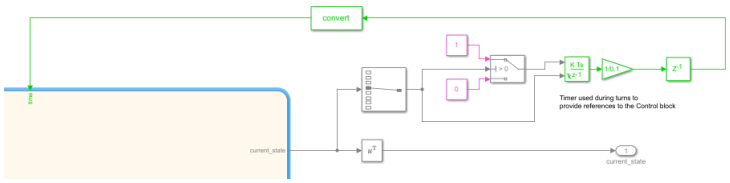


Per quanto riguarda questa fase vi è stata un'evoluzione della sua implementazione a seconda delle varie fasi del lavoro ed è utile menzionarle tutte: nella prima fase avevamo realizzato una funzione del tutto analoga a quella di generazione dei waypoint offline dei transetti; quindi utilizzavamo delle page, una per quanto riguarda le posizioni di riferimento ed una per le orientazioni di riferimento da passare; per ogni distanza di transetto avevamo deciso che l'angolo di 180° sarebbe stato diviso in 5 parti e per ogni 'arco' di circonferenza spazzato, venivano passati i riferimenti di posizione e orientazione del relativo midwaypoint. Nella fase d'implementazione il modulo di controllo ha riportato la necessità di generare qualcosa 'online' per quanto riguarda le virate e l'unica richiesta da loro postaci e successivamente realizzata è stata quella di gener-

are solamente il waypoint di riferimento di centro curva, in maniera tale che il controllo si calcolasse la distanza laterale di ogni punto della virata rispetto al centro, che ovviamente essendo la curva di raggio costante, doveva essere mantenuta costante e pari a questo raggio, punto per punto, effettuando quindi principalmente un controllo sullo sway. Abbiamo effettuato numerose prove, la curva veniva in verità eseguita bene, senza disallineamenti esagerati, anzi il veicolo si manteneva sempre in traiettoria, l'unica cosa era che nella primissima fase della curva non riusciva ad orientarsi 'perfettamente'. Sapevamo bene di non necessitare di un inseguimento di traiettoria su curva preciso, anche perchè in curva non avvengono rilevazioni per il nostro task di esplorazione, però per realizzare qualcosa di più carino e preciso abbiamo pensato ad un metodo che potesse essere più efficiente da questo punto di vista. Col modulo di controllo siamo giunti ad una conclusione che poi è stata quella d'implementazione finale e che è risultata soddisfacente durante tutte le prove di simulazione e cambiamenti vari, soprattutto per quanto riguarda il raggio di curvatura che si andava a cambiare, poichè la distanza tra transetti è parametrica. Quindi con questo nuovo metodo abbiamo risolto due problemi: un inseguimento più accurato della curva, anche in casi di curva più ristretta, che poteva darci maggiori problemi.



In questo caso abbiamo mantenuto le page per salvare i midwaypoints di riferimento, solo che in questo caso si è deciso di passarli online al modulo di controllo solo come riferimenti più precisi da dover inseguire, non vi effettuiamo di volta in volta un check, sono solo per indirizzamento per il modulo di controllo, dunque sono un numero molto fitto di riferimenti: ogni quanto vengono ad essere generati questi punti, lo si fa con una sorta di timer.

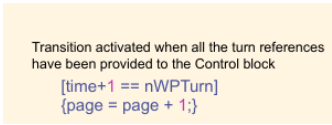


In uscita al current state viene inserito un blocco selettore, che agisce solo se il bit numero 1 si trova in quarta posizione del vettore current state, quindi se siamo in fase di turn. Avvenuto questo check, se siamo in fase di turn si attiva questa sorta di funzione 'timer' attraverso un blocco integratore discreto, di guadagno 1; quello che viene fatto è dividere il guadagno dell'integratore per 0.1, in maniera tale da campionare la curva ogni 0.1 sec e passare il riferimento ogni 0.1 sec (appunto vi è il blocco delay per generare quest'attesa tra un riferimento generato e l'altro); il timer è impostato ogni 0.1 sec poichè 10 Hz è la frequenza di campionamento del controllore, quindi passare dei punti a frequenze più alte sarebbe inutile, andrebbero persi; col blocco convert ci assicuriamo di uniformare l'ingresso al nostro blocco tramite variabili intere.

```
%Radius of the circumference
r = trans/2;

%Number of waypoints to generate; actual number varying depending
%on the circumference radius to achieve better precision
if trans<4
    nwp = 200;
else
    nwp = floor(pi/((vel/r)/10));
end
```

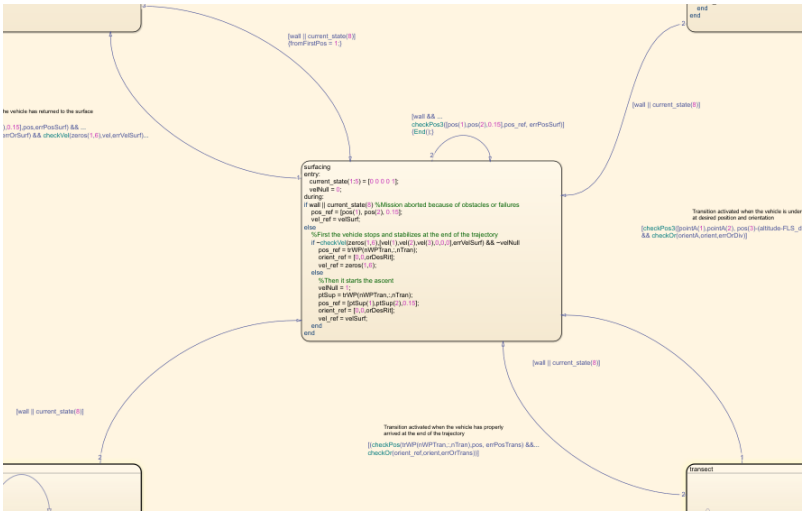
Dopo una serie di prove, di tipo trial and error, ci siamo accorti che per ottenere dei buoni inseguimenti di traiettoria in curva, non bisogna esagerare: basta dare un numero di waypoint parametrico, in base al raggio di circonferenza e alla velocità angolare in curva, il tutto diviso 10 perchè la frequenza di campionamento del controllore è di 10 Hz; non serve esagerare col numero di waypoint da passare per ottenere buone prestazioni; abbiamo differenziato solo un caso: per raggi di transetti minori di 2 m, ci siamo resi conto che se vogliamo mantenere un buon inseguimento di traiettoria, solo in questo caso, diamo un 'sovrannumero' di waypoints di riferimenti in curva, che dopo varie prove, quello ottimale a noi risultatoci è stato circa 200; quindi anche in caso di esplorazioni di fondale con transetti a distanze molto più ravvicinate, che sono comunque i casi in cui per un veicolo di forma allungata ellissoidale come il nostro, già lungo di per sè, più di 2 m, sarebbe più difficile mantenere buone prestazioni anche in curva, riusciamo ad ottenere buoni risultati. Nel blocco `ReferencePoints` di virata non è presente un check di posizione e orientazione, perchè più che avere una precisione punto punto, vogliamo venga effettuata una curva, che sia tale nel senso di traiettoria, da un punto di vista 'qualitativo', più che quantitavo; ma il fatto di aver dato dei riferimenti abbastanza fitti risolve anche il problema 'quantitativo', perchè l'errore di scostamento dalla curva sarà sicuramente poco. Duqnue, finchè non scatta il timer si dà sempre stesso riferimento di posizione o orientazione, quando scatta, si dà il successivo, andando ad incrementare la variabile time di 1, che indica l'indice dell'i-esimo waypoint della virata all'interno della relativa page. L'unico controllo successivo è sulla velocità, poichè sui transetti pari, abbiamo rotazioni antiorarie, dunque la velocità angolare, sullo yaw è positiva; sui transetti dispari invece, date le rotazioni

orarie, la velocità angolare è uguale in modulo alla precedente, ma negativa. Ogni volta che si è finito di compiere la virata, quindi quando la condition di time+1 == nWPTurn, ossia quando la variabile time che indica l'indice e quindi il numero dell'i-esimo waypoint in virata della pagina relativa è uguale al numero che indica praticamente l'ultimo waypoint della relativa page, perchè nWPTurn è uguale alla size delle righe della page, allora la condition action setta la pagina a quella successiva, con l'indice riga della nuova pagina wpTR, sempre settato a 1.

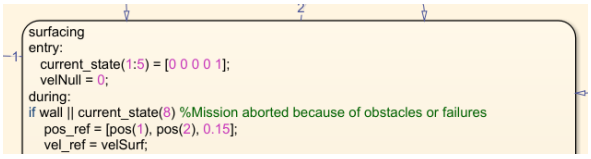


Quindi l'action dirà di dare come riferimento il primo waypoint del transetto successivo e così via fino alla fine dei transetti, alla fine dei quali verrà attivata la transition con una funzione di check, che se verificata, attiva lo stato di emersione.

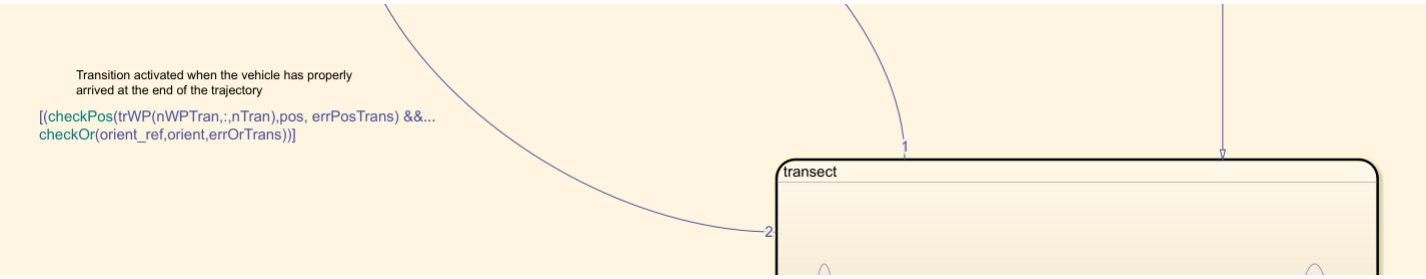
Un ultimo stato è quello di riemersione, `surfacing`: è presente una transition verso lo stato di riemersione per ogni stato del blocco Movements, perchè si potrebbe decidere di riemergere da qualsiasi degli stati e a breve diverrà chiaro il perchè: la riemersione viene ad essere attivata, o in caso di failsafe, danni strutturali a sensori e motori, in cui si decide di attivare l'abort o in caso di wall; lo stato parete è un altro stato di emergenza, in caso di ostacolo, che magari risulta superabile sì, ma non mantenendo il vincolo ad altitudine costante, quindi non ci permetterebbe di proseguire correttamente il task di esplorazione.



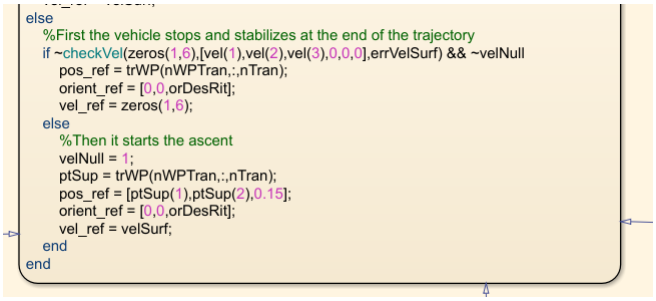
Quando si entra nel surfacing, viene ovviamente settato il bit corretto, e inizializzata una variabile flag, di controllo sulla velocità; il surfacing quindi nel caso di wall e abort si attiva mantenendo posizione corrente del veicolo, viene cambiato solo il riferimento di profondità, perchè desideriamo che il veicolo in casi di failsafe abortisca e ritorni, a partire dal posto in cui si trova, solo in superficie.



Vi è una transition aggiuntiva da transect a surfacing nel caso ovvio di completamento della missione, compimento dell'ultimo transetto. Si compie un ultimo check di posizione e orientamento e quando siamo nella sfera di arrivo dell'ultimo waypoint dell'ultimo transetto, sfera di 1m, viene effettuata la transizione e si attiva la riemersione: il veicolo in questo spazio si fermerà e una volta fatto ciò riemergerà con una velocità di riemersione sempre pari a 0.2, parametro da noi scelto.

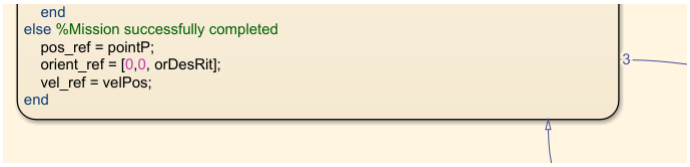


Nel caso in cui la riemersione avvenga perchè il task è stato compiuto ed è andato a buon fine si fa dunque prima il check sulla velocità, perchè finchè la velocità non è nulla, quindi finchè il veicolo non si è fermato non può iniziare la vera e propria emersione. Se la velocità non è nulla si continua a dare l'ultimo riferimento di posizione, (ultimo waypoint dell'ultimo transetto) orientazione, e velocità nulla. Se il veicolo si è fermato sul punto desiderato, può iniziare la vera e propria emersione:



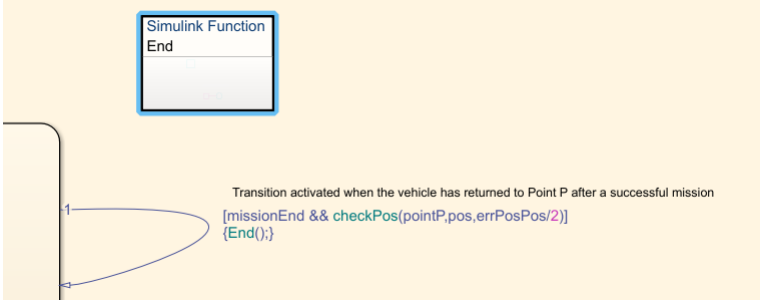
Una volta arrivati nel raggio di emersione con posizione e orientazione e velocità nulla desiderati, e a meno di un certo margine di errore avviene la validazione della funzione presente nella condizione di transition da surfacing a positioning; ciò che viene controllato in aggiunta è che la posizione di riferimento data sia diversa rispetto alla posizione di riferimento inziale nella fase di positioning di inizio missione, poichè ovviamente il punto di emersione è diverso dal punto di immersione. Effettuato questo check si setta la condition action di missionEnd a 1, che indica appunto che la missione è finita cosicchè nell'entry di positioning si setta come sempre il current state corrente e le variabili flag di orientazione e velocità.

Si entra praticamente nell'ultimo else che dà riferimenti di posizione e orientazione di initPoint, per ritornare al punto di deploy, con velocità di crociera, pari a velPos, che è la velocità di crociera sopra acqua, parametro che abbiamo scelto noi a 0.5 m/s.

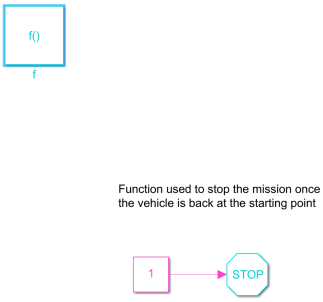


E' presente dunque una sola transizione di uscita dal surfacing verso positioning, perchè per nostra decisione, ovviamente solo nel caso di riemersione per

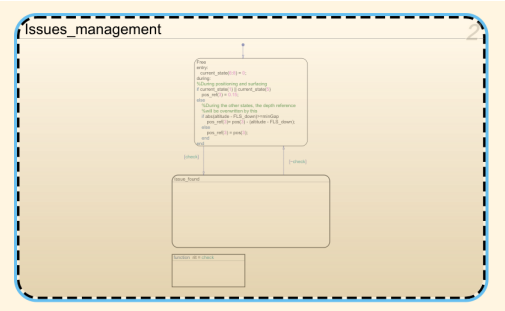
missione andata a buon fine, abbiamo deciso di ritornare al punto di partenza, ossia al punto di deploy del veicolo, che deve essere appunto il posto in cui si trova l'imbarcazione dalla quale ha avuto inizio il posizionamento del veicolo, cosicchè da evitare che l'imbarcazione si debba mettere alla ricerca del veicolo in tutta l'area di interesse.



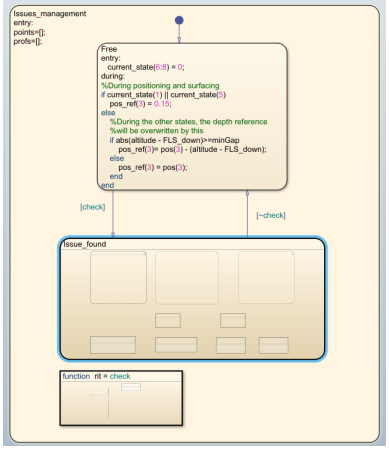
Anche in questo caso, per esser precisi, facciamo un check sulla posizione di arrivo; una volta arrivatici con una funzione di stop, diamo il segnale di fine missione.



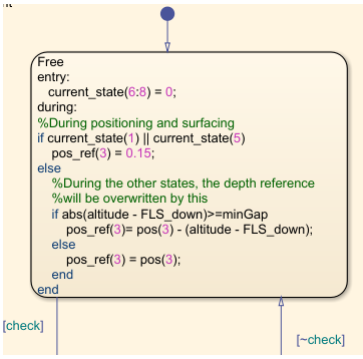
4.2 Discussione dell'implementazione del blocco issues_Management.



Innanzitutto nel blocco generale di Issues vengono settate due variabili, due matrici vuote, points [] e profs [], che definiremo in seguito.

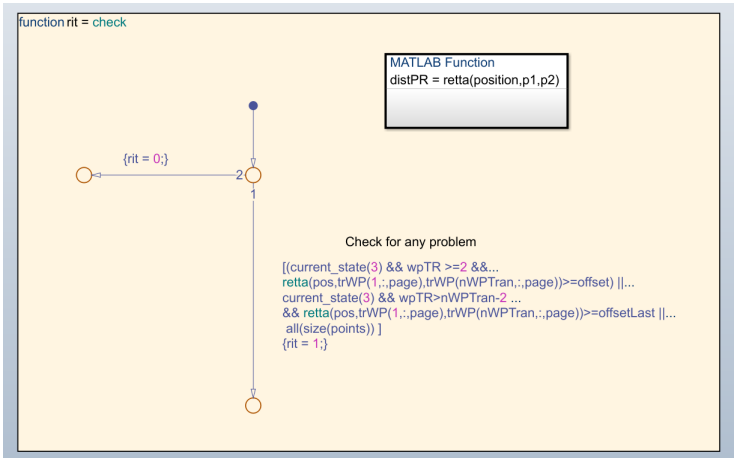


All'interno del blocco, la default transition, come spiegato precedentemente è sul blocco free, che indica l'assenza di situazioni di emergenza, ossia come settato nell'entry, quando gli ultimi 3 bit del current state stanno contemporaneamente tutti settati a 0.



E' in questo blocco che si fa il vero e proprio controllo del riferimento di profondità. Nel during, se il current state ha il bit 1 in posizione 1 o 5, vuol dire che siamo o nello stato positioning o nello stato surfacing, dunque il riferimento di profondità di posizione è 0.15; durante invece l'attivazione degli altri stati del blocco Movements, il riferimento di profondità sarà sovrascritto secondo una logica che sta alla base poi della logica di implementazione della nostra funzione sulla gestione di eventuali ostacoli presenti. Se la differenza in termini di valori assoluti tra il parametro altitude che è costante, vincolo di profondità a cui dobbiamo rimanere costanti per l'esplorazione di fondale, e il parametro di distanza da fondale corrente supera gli 0.5 m, allora siamo in condizioni tali da dover dare il nuovo riferimento di profondità da dover essere raggiunto, altrimenti, se la variazione di fondale in altezza è meno del mezzo metro abbiamo ritenuto opportuno non modificare il riferimento di profondità in maniera tale che, data la certezza della presenza di un fondale irregolare, il veicolo non ne fosse sensibile effettuando una traiettoria zig-zagata per seguirlo, poco efficiente per l'effettuazione della survey, in termini di tempi e prestazioni.

E' la funzione di transizione `check`, che attiverà la transizione dallo stato free, ad uno od a più stati di emergenza: `warning, obstacle, abort.`



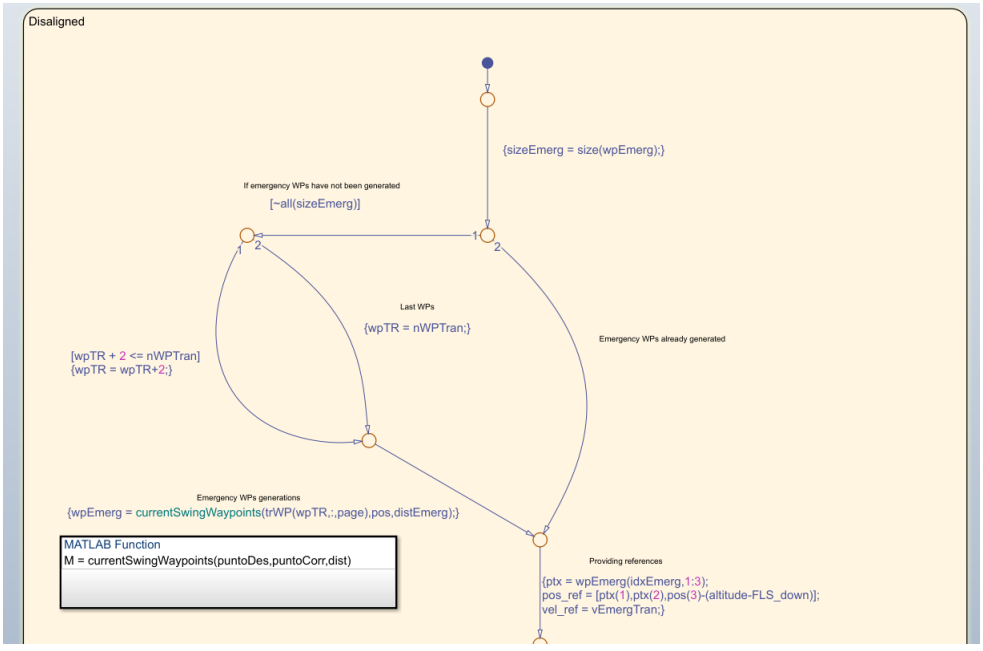
Per verificare la possibile situazione di warning, disallineamento, si controlla che la distanza laterale del punto corrente eventualmente disallineato rispetto alla retta del relativo transetto sia maggiore di un certo offset. Il controllo viene effettuato a partire dal secondo waypoint ovviamente, perchè sul primo già ci troviamo ben posizionati, e la distanza laterale durante tutta la fase di transettatura che deve essere superata per attivare il warning è stata scelta da noi e definita dalla variabile offset di 2m. Nel caso in cui il disallineamento avvenga a partire dagli ultimi due waypoint, l'errore laterale massimo accettabile viene ridotto ad 1m, perchè a partire dal penultimo waypoint abbiamo pochi metri di transettaura per poter riallinearci prima di andare in curva, quindi considerando sempre i tempi che il veicolo impiega per traslare siamo arrivati a questa sorta di trade-off. Nei casi in cui dunque si verificasse una delle situazioni di cui sopra, si esce dal free e si entra nel warning.

Il check che invece rileva la presenza di ostacoli viene verificato controllando che il vettore points, che contiene i punti degli eventuali ostacoli sia pieno; poichè se lo è, vuol dire che sono stati rilevati ostacoli e vuol dire che la tranisizione check viene attivata, si esce dallo stato free e si entra nello stato di obstacle. Se non si è in nessuno dei casi sopra citati la funzione check torna valore 0, quindi si rimane o si ritorna nel free.

Nel caso ad esempio si verifichi il disallineamento, si entra nello stato `warning`, viene verificata la distanza richiamando la funzione disalignOP che controlla se questa distanza supera gli offset da considerare e setta il current state con bit 1 in posizione 6; a seconda poi che ci si trovi in uno dei 5 stati principali esclusivi, come si effettua nel free, viene passato il riferimento in profondità corretto.

```
warning %disalignment
during:
%Disalignment check activated only during the transect
if current_state(3)
   disalignOP;
end
if current_state(1) || current_state(5)
   pos_ref(3) = 0.15;
else
   if abs(altitude - FLS_down)>=minGap
      pos_ref(3)= pos(3) - (altitude - FLS_down);
   else
      pos_ref(3) = pos(3);
   end
end
```

La gestione del disallineamento viene effettuata nel blocco `disaligned` di transect, in Movements.



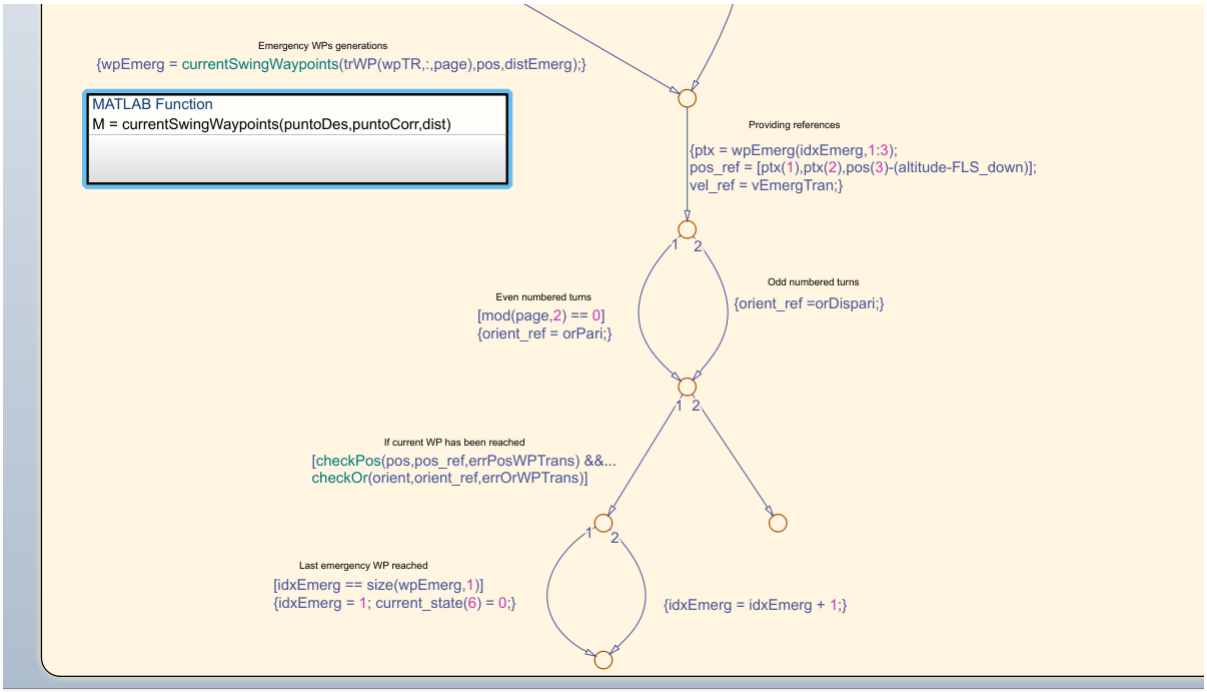
Nella default transition si setta il valore della lunghezza della matrice che contiene i waypoint al valore sizeEmerg, che inizialmente vale 0, se siamo ap-

pena entrati nel disallineamento e non abbiamo ancora creato i midwaypoints di recupero traiettoria; dunque se siamo in una situazione di inizio, appena disallineati sicuramente la funzione all(sizeEmerg) restituirà valore 0, quindi si va nella transizione a sinistra e si verifica, per prima cosa, la prima condizione, ossia, finchè l'indice che indica la riga i-esima e quindi il waypoint i-esimo sul transetto di riferimento non è il penultimo allora il punto di ritorno in traiettoria rispetto al punto disallineato sarà settato di due indici in avanti della page corrente, cosicchè il veicolo, disallineato di più di 2 m ha il tempo di ritornare in tariettoria; nel caso in cui l'indice corrente va dal penultimo in poi, allora l'unico punto di riferimento che si può passare per tornare in traiettoria è esattamente l'ultimo del transetto prima della virata. Determinato questo punto di ritorno in traiettoria, sempre due indici di riferimento in avanti, si genera il vettore dei waypoints di emergenza, tramite la funzione currentSwingWwaypoints:

```
function M = currentSwingWaypoints(puntoDes,puntoCorr,dist)
%CURRENTSWINGWAYPOINTS generates additional waypoints to be used when the
% vehicle misaligns from the desired trajectory
%
%   -puntoDes: desired return point onto the trajectory; 1x3 [m m m]
%   -puntoCorr: estimated current location; 1x3 [m m m]
%   -dist: distance between waypoints in the direction of realignment; [m]

    distN = puntoCorr(1) - puntoDes(1); %Distance from starting and final point onto the North axis
    distE = puntoCorr(2) - puntoDes(2); %Distance from starting and final point onto the East axis

    %Number of waypoints to generate
    p = floor(abs(distE)/dist);

    deltaE = distE/p; %Distance between two waypoints onto the North axis
    deltaN = distN/p; %Distance between two waypoints onto the East axis

    matrix = zeros(p-1,3);

    %Ending at p-1 because the last additional waypoint corresponds to
    %"standard" waypoint onto the trajectory
    for i = 1:p-1
        %Waypoint generation
        waypoint = puntoCorr - i * [deltaN deltaE 0];
        waypoint(3) = puntoDes(3);
        matrix(i,:) = waypoint;
    end

    M = matrix;
```

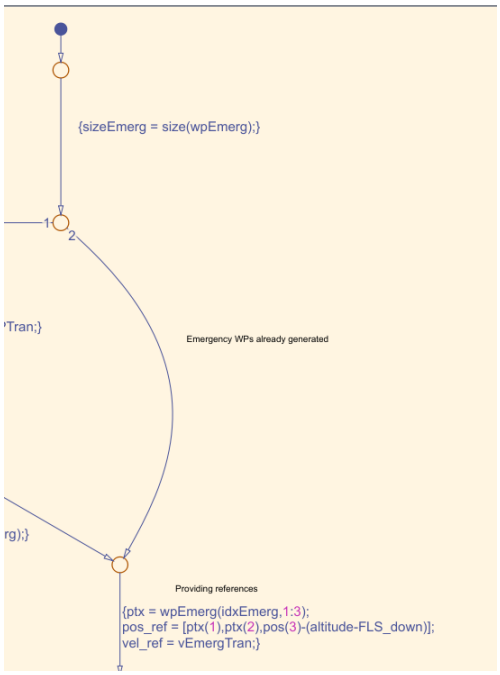
In questa funzione si determina la distanza lungo la direzione Nord e lungo la direzione Est dei waypoints (il disallineato e quello di ritorno in traiettoria); divido la distanza lungo est per distEmerg, che noi abbiamo deciso ogni 0.5 m e ottengo il numero dei waypoints da generare p; divido la distanza lungo Nord e lungo Est per il numero di questi waypoints, così ottengo le distanze deltaN e deltaE fra i waypoints nelle due direzioni; effettuo tutto questo affinchè il numero di waypoints che viene generato sia parametrico, relativo a questa distanza dal punto di allineamento. Quindi genero una matrice dei waypoints di dimensione riga pari al numero dei waypoints generati p, meno 1, perchè l'ultimo punto è quello di riferimento del transetto. Nella matrice si passa come riferimento

il punto che setta la distanza tra il punto corrente e i vari punti di emergenza da dover raggiungere per allinearsi; una volta creata la matrice dei waypoints si passa il primo waypoint di riferimento con velocità settata alla velocità di emergenza, che abbiamo deciso di settare a 0.2 sia sul surge, che sullo sway, che sull'imbardata, poichè il veicolo sta traslando; anche qui l'orientazione viene settata a seconda se siamo su transetto pari o dispari.

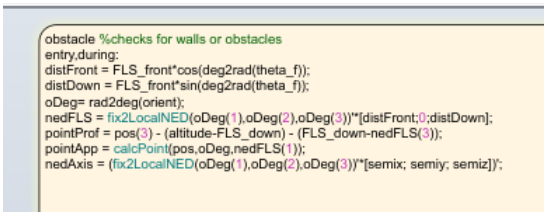


Avvenuto ciò si fa un check per vedere se ci siamo avvicinati entro un certo margine d'errore al waypoint di riferimento; per quanto riguarda i margini di errore fra questi midwaypoints essendo molto vicini tra loro li abbiamo scelti decisamente più stringenti, di 50 cm sul surge e 20 cm su sway e yaw, per poter dire, entrati entro un certo margine d'errore, di passare già il riferimento successivo; se si è raggiunto l'ultimo waypoint con margine di errore accettabile allora, viene risettato l'indice di riferimento del waypoint di nuovo ad 1, in maniera tale che a nuovo disallineamnto iniziato, la matrice dei waypoints di emergenza dia il riferimento che sta in riga di posizione 1; settato l'indice idxEmerg, siamo usciti dal disallineamento, quindi si resetta il current state del warning a 0; nel caso in cui non siamo ancora all'ultimo waypoint, si incrementa di 1, l'indice riga che si riferisce al successivo midwaypoint di emergenza da raggiungere e si prosegue con l'allineamento in traiettoria. Poichè si ritorna indietro, si fa il check sulla matrice dei waypoints che stavolta non sarà vuota, quindi non essendoci da generare i waypoints va nella transizione 2 della giunzione e genera il successivo

riferimento, in posizione riga data dall'indice idxEmerg incrementato di 1.

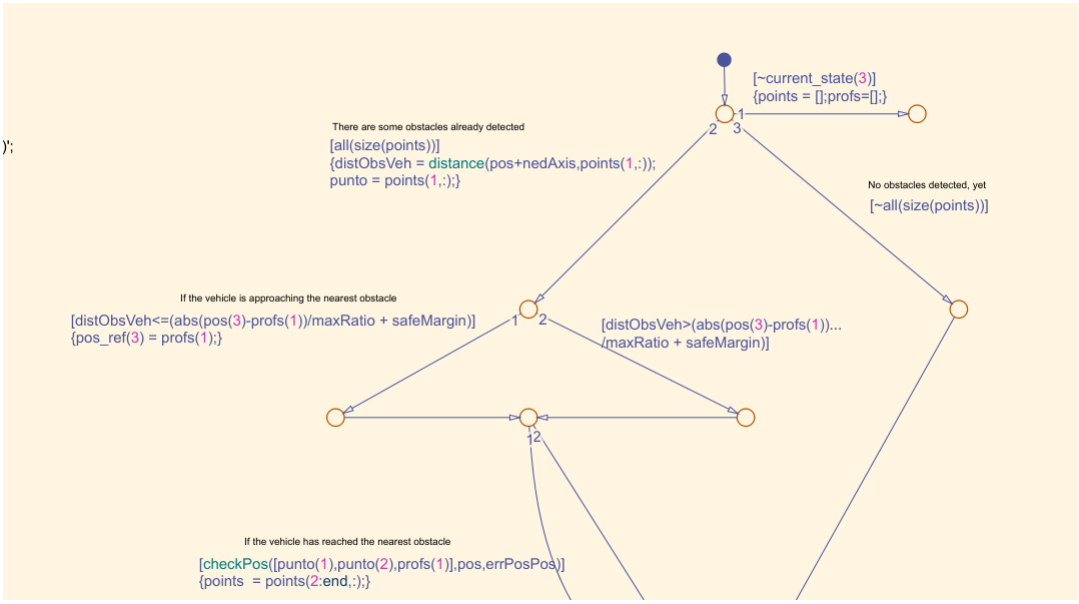


Nel caso nel check venisse rilevata la presenza di ostacoli viene attivato lo stato `obstacle`



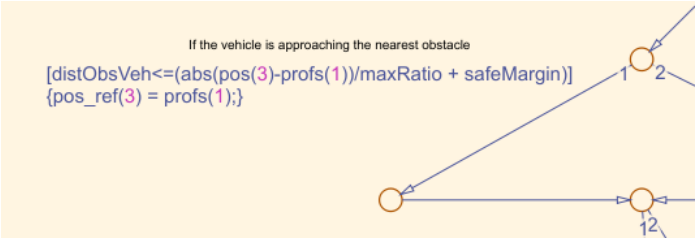
Nell'entry/during di obstacle si fa un calcolo che risulterà a noi comodo per poter poi effettuare la funzione di gestione dell'ostacolo. Sapendo già l'angolo di inclinazione del sonar, possiamo calcolarci, conoscendo ipotenusa (slunt range) e angolo, la lunghezza dei due cateti, ossia la lunghezza in avanti, lungo il moto di surge e in giù, lungo il down, che possono essere importanti informazioni, poichè la misura di slunt range è fondamentale, per rilevare l'eventuale presenza di un ostacolo, con queste altre informazioni posso calcolarmi quanto sia la sua elevazione da fondale e quanto sia la sua distanza frontale lungo il moto di surge, dal veicolo. La prima cosa che facciamo è riportarci queste grandezze dal sistema di riferimento locale a globale tramite la funzione fix2LocalNED,

conoscendo l'orientazione corrente.
PointProf salva le coordinate di cambiamento di profondità: la prima parentesi ci stabilizza in profondità ad altezza altitude dal fondale; se FLS_down, misura il valore di altitude il risultato della parentesi è 0, quindi siamo già in posizione da fondale desiderata da vincolo, se FLS_down misura un valore diverso da altitude, allora conoscendo la loro differenza, si restituisce il valore di profondità a cui il veicolo deve riportarsi per mantenersi sempre ad altitudine desiderata da fondale; stabilizzato a questa altitudine nell'altra parentesi si effettua una differenza per calcolare preventivamente l'eventuale presenza di un ostacolo, poichè se l'ostacolo è presente il risultato di nedFLS(3) non sarà eguale al valore di FLS_down; nel caso lo fosse, nessun ostacolo sarebbe rilevato, altrimenti sì, e conosciamo anche quanto è l'elevazione da fondale dell'ostacolo o di quanto è la variazione del fondale verso il basso, in direzione opposta, a cui dobbiamo portarci.
PointApp restituisce in NED le coordinate nord ed est di distanza dell'ostacolo rispetto al veicolo. Un altro accorgimento è stato quello di riportarci in assi NED, le distanze dei semiassi del veicolo, poichè le nostre rilevazioni non sono fatte dal centro del veicolo, ma sulla parte frontale e quindi soprattutto il semiasse del veicolo lungo x, di 1 m e 10, sarebbe quella distanza in più lungo surge che noi in realtà non avremmo perchè occupata dalla parte di veicolo; questo accorgimento non è banale, visto che noi abbiamo un veicolo di forma allungata e visto che vorremmo riportarci sugli ostacoli con margini di distanze precisi e 1 m di differenza cambia tanto.



Tramite le giunzioni viene implementata la vera e propria funzione di ges-

tione degli ostacoli; il primo controllo che si effettua tramite la default transition è quello di verificare se siamo lungo il transetto, perchè se ancora non lo siamo, quello che si fa è creare delle matrici vuote: points [], che conterrrà i punti di coordinate nord-est, in NED, dell'eventuale ostacolo rilevato rispetto al veicolo e profs[], che conterrà il punto di coordinata depth, sempre in NED, dell'ostacolo. Nella transizione numero 2 si controlla che il vettore che contiene le coordinate nord est delle variazioni di fondale considerate rilevanti (variazione lungo depth > 0.5, check già effettuato nel free) non sia vuoto. Se il vettore points non lo è, vuol dire che è stata rilevata la presenza dell'eventuale ostacolo/degli eventuali ostacoli e tramite la funzione distance ci si calcola la distanza attuale, lungo nord est, del veicolo rispetto a questi punti, ostacoli rilevati e salvati nel vettore punto. Solo nel caso di primo riempimento del vettore points, si vanno a considerare i punti con variazione di profondità maggiore di 0.5, anche se la variazione di altitudine del fondale non persiste per 2 m; solo in questo caso per poter poi effettuare le successive considerazioni, altrimenti, nel caso già di vettore pieno solo se vengono superati i margini di minGap e minDist, successivamente spiegati, la variazione viene considerata ostacolo, rilevante ed inserita nel vettore di gestione degli ostacoli, points.



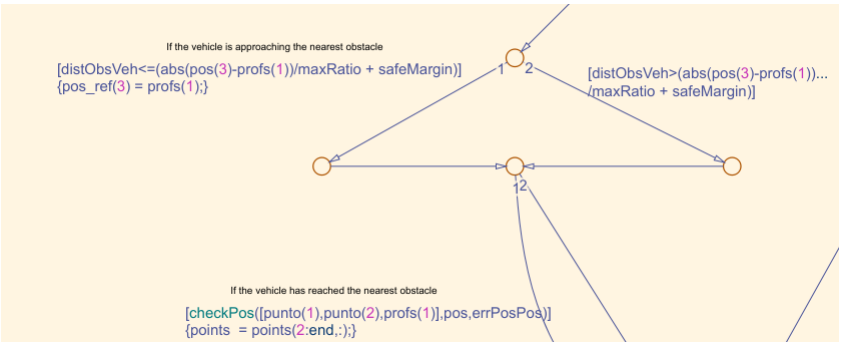
Per spiegare la successiva condition e le variabili in essa utilizzate bisogna soffermarci su delle considerazioni preventive che è stato opportuno determinare, alla base della gestione degli ostacoli e dello sviluppo della nostra politica del loro superamento, nella maniera che a noi è sembrata la più ottimale possibile. Per prima cosa grazie ai dati provenienti dal modulo di controllo, noi conosciamo quanto tempo impiega il veicolo per portarsi ad una variazione di profondità di 4 m, che è 8 sec. A seconda della velocità a cui si sta muovendo il veicolo la distanza lungo il surge che ci serve per portarci ad una tale variazione di profondità varia ovviamente. Per prima cosa ci siamo dunque calcolati, in presenza di una variazione di fondale imprevista, con pendenza simil parete, qual è la variazione massima di fondale che riusciamo a sostenere mantenendo il vincolo di profondità costante: questa ultima considerazione in funzione del fatto che anche se fisicamente riuscissimo a superare l'eventuale ostacolo, quindi senza sbatterci contro, però non con il vincolo di altitudine, rispetto all'ostacolo indicato dal task, rispettato, allora noi saremo in condizione di parete e abortiremmo la missione. Per considerazioni ulteriori di sicurezza sull'eventuale presenza di correnti, errori di misura, limitazioni fluidodinamiche, abbiamo deciso di molti-

plicare per 2 il valore di distanza di cui il veicolo avrebbe bisogno per superare il relativo ostacolo. Dal momento che noi siamo in presenza di un sonar inclinato di 45°, sappiamo qual è per ogni altitudine a cui ci troviamo e ci dobbiamo mantenere rispetto al fondale, qual è la distanza massima lungo il moto di surge a cui iniziamo a rilevare la prima variazione di fondale, quindi tramite una proporzione del rapporto tra la variazione di profondità rilevata e la distanza che occorre per superare l'ostacolo mantenendo il vincolo di altitudine, riusciamo a rilevare il valore in proporzione del rapporto, oltre il quale siamo sicuri di non riuscire a superare l'ostacolo e questo valore viene calcolato e salvato nella variabile maxRatio.

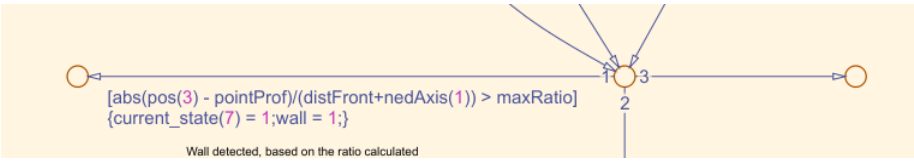
```
%% TEST OBSTACLE
tSalto = 8; %[s]
profGap = 4; %[m]

minGap = 1; %gap minimo per variare profondità; [m]
minDist = 2; %per ignorare "scoglietti"; [m]

bound = cruiseSpeed*tSalto; %da controllo: salto di 4m in 25s, quindi
%distanza massima percorsa in tale tempo in condizioni ideali
distNeeded = bound*2; %per sicurezza, così da considerare fluidodinamica, correnti, etc...

maxRatio = profGap / distNeeded;
```

Dopo tutte queste considerazioni verrà ora spiegato come e quando viene passato al modulo di controllo il riferimento di cambiamento di profondità, poichè ovviamente in base a quanto è alto l'ostacolo il veicolo ha bisogno di più o meno tempo per superarlo e sarebbe inutile cominciare l'elevazione troppo prima, in più non si manterrebbe il vincolo di altitudine costante. Tramite la differenza di pos(3)- profs(1), conosco quanto è la variazione effettiva della profondità del fondale, rispetto alla situazione nominale, la divido per il valore di maxRatio è ottengo una sorta di lunghezza della pendenza che mi serve per superare l'ostacolo e ci sommo un valore di safeMargin, scelto da noi di 50 cm, sempre per questioni di sicurezza sopra citate.

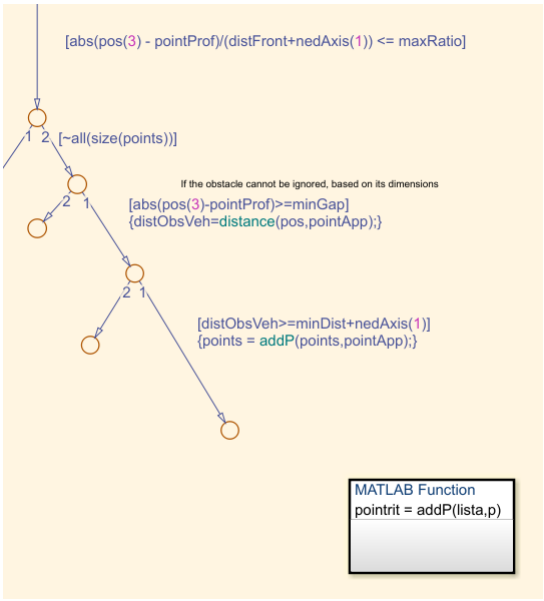
Ora la condition ci dice che se la distanza che viene calcolata tra la posizione del veicolo corrente e l'ostacolo è ancora maggiore rispetto a quella che effettivamente ci serve per posizionarci sopra l'ostacolo in tempi ottimali, non viene cambiato il riferimento di profondità e il veicolo non attua nessuna variazione di altitudine, invece se arriviamo a quella distanza, viene passato il riferimento di cambiamento di profondità.



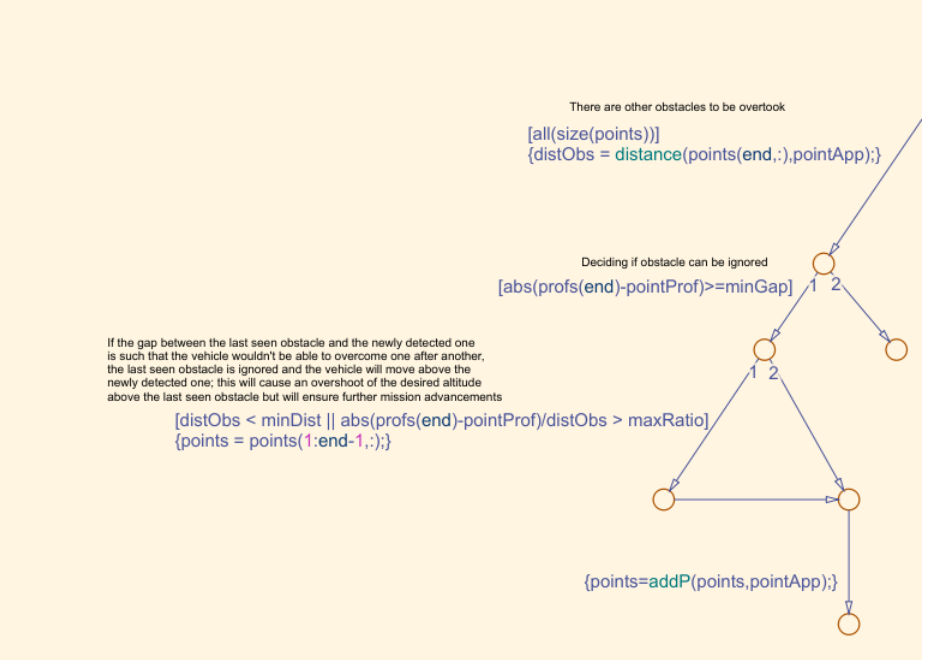
Successivamente, come mostrato sempre nella figura sopra, si fa un check per assicurarci di esserci portati sull'ostacolo con vincolo di altitudine rispettato entro sempre un certo margine di errore. Se il check va a buon fine, viene risalvato il vettore contenente le coordinate degli ostacoli, ogni volta senza il primo elemento, perchè esso è stato praticamente appena gestito.



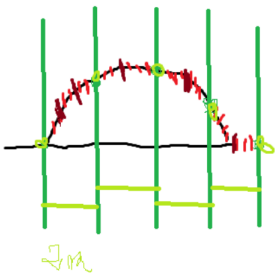
Finchè ancora il punto non è stato raggiunto, si fa un check molto importante: se il rapporto tra variazione di profondità e distanza che abbiamo lungo il moto di surge rispetto all'ostacolo è maggiore del rapporto restituito da maxRatio, vuol dire che non abbiamo lo spazio necessario lungo surge per superare l'ostacolo mantenendo il vincolo di altitudine costante, quindi viene aggiornato il current state di presenza di parete ad 1 con la variabile flag wall a 1 e si attiva la riemersione.

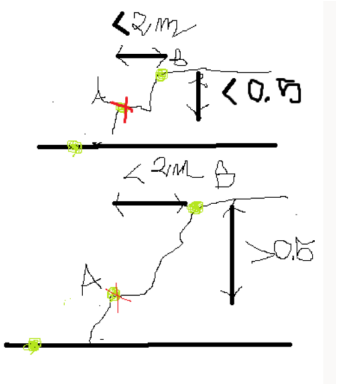


Nel caso in cui invece siamo in condizioni di superare l'ostacolo mantenendo il task, si verifica se il vettore points è pieno, oppure no; perchè se non si è riempito si controlla solo che la differenza tra l'altitudine a cui siamo e quella data dal pointProf (variazione di altitudine sempre calcolata, tramite FLS_down e front) sia maggiore di 0.5 e per poter essere gestita si calcola la differenza tra la posizione del veicolo attuale rispetto alla distanza ostacolo-veicolo e se essa è maggiore di minDist, quindi di 2 m, allora questo nuovo punto si aggiunge in coda, nella lista dei points, degli ostacoli da considerare.



Nel caso invece in cui il vettore degli ostacoli è pieno, viene calcolata la distanza relativa tra gli ostacoli e, se la differenza in profondità tra di essi è maggiore di mezzo metro, ma la distanza tra i 2, non supera i 2m oppure si potrebbe verificare un'altra situazione: nel caso in cui la differenza di pendenza fra i due ostacoli è tale per cui solo la distanza fra l'ostacolo, su cui ci saremmo riportati in condizioni normali di gestione ostacoli, e l'altro ostacolo, supera il rapporto di maxRatio, che ci indica il fatto che non riusciremmo a riportarci rispetto alla successiva variazione di profondità detectata, in tempo, perchè non abbiamo abbastanza spazio lungo il surge, mantenendo il vincolo di altitudine, questa cosa rileverebbe un wall e ci farebbe riemergere, interrompendo la survey, per una situazione di una non reale effettiva emergenza, allora quello che si fa è non considerare il dato di ostacolo inserito per ultimo, ma si considera il nuovo detectato e considerato rilevante per l'effettiva gestione ottimale degli ostacoli. Dando un'occhiata all'immagine ivi sotto presente si possono capire in maniera visiva più diretta queste due situazioni:




< 2m
B
A
< 0.5
< 2m
B
A
> 0.5