Stima dinamica dello stato di un sistema subacqueo

Matteo Bonato, Luca Morello, Giovanni Paolicelli

Gennaio 2023

1 Introduzione

Per il caso in esame viene considerato un veicolo autonomo subacqueo (AUV) la cui posa nel piano verticale è descritta dalle coordinate x e z rispetto al sistema di riferimento fisso e dall'angolo di beccheggio θ, la cui derivata nel tempo è q.

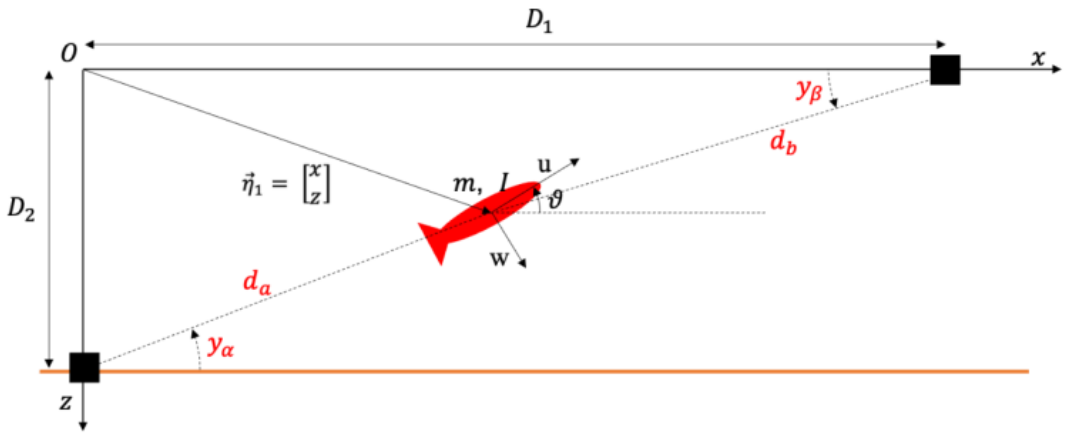

Figura 1: Sistema di riferimento AUV

Le componenti di velocità del veicolo rispetto ad una terna body solidale sono u e v, dove u è la componente di velocità parallela alla direzione di avanzamento del moto, mentre w è la componente di velocità perpendicolare ad u nel piano in esame. Il veicolo ha massa m e momento di inerzia relativo all'asse y della terna body pari ad I. I termini C_{li} e C_{qi} sono rispettivamente coefficienti di resistenza all'avanzamento lineari e quadratici. Il veicolo è attuato da una spinta e una coppia di controllo τ_u e τ_q ed è soggetto alle azioni di disturbo τ_{di}. Si hanno a disposizione tre sensori: un profondimetro che misura la coordinata z nel sistema di riferimento inerziale, un giroscopio che misura l'angolo θ e un sensore di prossimità posizionato su una boa, distante D_1 dall'origine, in superficie che misura la distanza tra la boa stessa e l'AUV, d_b. Il fondale si trova invece ad una profondità D_2.

Sono qui presentate le equazioni della cinematica (1) e del moto in terna body (2).

$$\begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ w \\ q \end{bmatrix} \tag{1}$$

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 & 0 & mw \\ 0 & 0 & -mu \\ -mw & mu & 0 \end{bmatrix} \begin{bmatrix} u \\ w \\ q \end{bmatrix} + \begin{bmatrix} C_{lu} + C_{qu}|u| & 0 & 0 \\ 0 & C_{lw} + C_{qw}|w| & 0 \\ 0 & 0 & C_{lq} + C_{qq}|q| \quad 0 \end{bmatrix} \begin{bmatrix} u \\ w \\ q \end{bmatrix} = \begin{bmatrix} \tau_u \\ 0 \\ \tau_q \end{bmatrix} + \begin{bmatrix} \tau_{du} \\ \tau_{dw} \\ \tau_{dq} \end{bmatrix} \tag{2}$$

L'obiettivo di questo progetto è quello di fornire una stima dello stato tramite applicazioni di filtri quali il filtro di Kalman esteso (EKF) e il filtro a particelle (PF). A tal proposito, il sistema è stato modellato su Simulink, dove sono stati anche implementati i filtri stessi.

1.1 Extended Kalman Filter (EKF)

Il filtro di Kalman esteso è una particolare realizzazione del filtro di Kalman nel caso non lineare. In questo caso, la funzione di transizione di stato (3) e il modello di osservazione (4) non devono necessariamente essere funzioni lineari dello stato, mentre ne è richiesta la differenziabilità, così da poter effettuare un'approssimazione al primo ordine mediante l'utilizzo del jacobiano:

$$x_{k+1} = f_k(x_k, w_k) \tag{3}$$

$$y_k = h_k(x_k, v_k) \tag{4}$$

dove w_k e v_k sono rispettivamente i disturbi sugli ingressi al sistema e i rumori di misura.

L'algoritmo del filtro è diviso negli step di predizione dello stato e di correzione. A seguire sono presentate le equazioni relative al passo di predizione del filtro, in cui viene linearizzata la f_k tramite matrice jacobiana.

$$F_k = \frac{\partial f_k}{\partial x_k}\Big|_{\substack{x_k=\hat{x}_{k|k} \\ w_k=0}} \tag{5}$$

$$D_k = \frac{\partial f_k}{\partial w_k}\Big|_{\substack{x_k=\hat{x}_{k|k} \\ w_k=0}} \tag{6}$$

$$\hat{x}_{k+1|k} = f_k(\hat{x}_{k|k}, 0) \tag{7}$$

$$P_{k+1|k} = F_k P_{k|k} F_k^T + D_k Q_k D_k^T \tag{8}$$

Si noti che F_k e D_k vengono valutate ad ogni step, sulla base della stima ottenuta in uscita dallo step di correzione. In (7) viene infine svolta la predizione vera e propria, con l'applicazione delle equazioni del modello alla stima corretta (filtrata), mentre (8) è la matrice di covarianza associata alla stima predetta, dove $P_{k|k}$ è la matrice di covarianza associata alla stima filtrata e Q_k è la matrice di covarianza dei disturbi sugli ingressi al sistema.

Vengono poi presentate a seguire le equazioni riguardanti lo step di correzione, in cui questa volta viene

linearizzata la h_k:

$$H_k = \frac{\partial h_k}{\partial x_k}\Big|_{\substack{x_k=\hat{x}_{k|k-1} \\ v_k=0}} \tag{9}$$

$$M_k = \frac{\partial h_k}{\partial v_k}\Big|_{\substack{x_k=\hat{x}_{k|k-1} \\ v_k=0}} \tag{10}$$

$$e_k = y_k - h_k(\hat{x}_{k|k-1}, 0) \tag{11}$$

$$S_k = H_k P_{k|k-1} H_k^T + M_k R_k M_k^T \tag{12}$$

$$L_k = P_{k|k-1} H_k^T S_k^{-1} \tag{13}$$

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + L_k e_k \tag{14}$$

$$P_{k|k} = P_{k|k-1} - L_k S_k L_k^T \tag{15}$$

Anche qui, ad ogni step, vengono calcolate le matrici H_k e M_k mediante l'ausilio delle stime predette. Date $\hat{x}_{k|k-1}$ e $P_{k|k-1}$, rispettivamente la stima predetta in uscita dallo step di predizione e la matrice di covarianza ad essa associata, data R_k, la matrice di covarianza associata ai rumori di misura, in (11) e (12) si ha il calcolo dell'innovazione legata alle misure e la sua covarianza. In (13) si ha il calcolo del guadagno di correzione, noto anche come `\textit{Kalman gain}`, dal quale si ottengono in (14) e (15) la stima filtrata e la corrispettiva covarianza.

In seguito verranno discussi in maniera più approfondita i passaggi qui brevemente riassunti nonché la loro implementazione nel framework di Simulink.

1.2 Particle Filter (PF)

Per *particle filter* (filtro a particelle) si intende una specifica tecnologia di filtraggio basata su metodi Monte Carlo sequenziali in cui si vuole calcolare una distribuzione di probabilità a posteriori mediante una rappresentazione a campioni (particelle) con l'ausilio della teoria bayesiana. Anche questo filtro è applicabile a modelli non lineari e in particolare è di grande interesse nella risoluzione di problemi non gaussiani.

Come nel caso del filtro di Kalman esteso, anche nel filtro a particelle si può distinguere tra uno step di predizione ed uno di correzione. La predizione avviene tramite l'applicazione del modello ad ogni singola particella, dopo aver generato una realizzazione di rumore coerente per ognuna di esse. In correzione viene poi effettuata una stima bayesiana atta ad individuare le particelle caratterizzate da una verosimiglianza più alta date le misure a disposizione. Le particelle sono dunque caratterizzate da pesi che descrivono la distribuzione di probabilità dei campioni e ad ogni passo di correzione viene effettuato un aggiornamento dei pesi sulla base delle verosimiglianze prima calcolate.

È opportuno descrivere brevemente un problema che si può verificare in fase di correzione. È realistico che dopo alcuni passi di filtraggio si otterranno poche particelle sul totale con un peso elevato: questo fenomeno prende il nome di degenerazione dei pesi e può influenzare negativamente l'intero processo di stima. Infatti, si può arrivare ad avere un solo campione utile sebbene, dal punto di vista computazionale, si continua a tenere traccia di ognuno di essi. Per questo motivo, la soluzione comunemente adottata è quella del *resampling*: le particelle poco importanti (caratterizzate da una bassa *likelihood*) vengono eliminate, mentre quelle più significative vengono moltiplicate e ridistribuite nell'intorno della particella di partenza.

Anche in questo caso saranno descritte in seguito più approfonditamente le metodologie impiegate.

2 Parametri del sistema

Nella tabella a seguire sono elencati i valori assegnati ai parametri del sistema scelti per la simulazione nonché il nome che è stato assegnato alle corrispondenti variabili all'interno del workspace di MATLAB. È possibile modificare il valore relativo a questi parametri all'interno dello script `param.m` che viene automaticamente caricato all'inizio di ogni simulazione.

Parametro	**Nome variabile**	**Valore**	**Unità di misura**
Massa del sistema	`m`	115	$\mathrm{kg\,m^2}$
Momento d'inerzia relativo all'asse y in terna body	`I`	5.98	$\mathrm{kg\,m^2}$
Linear drag coefficient, *surge* (u)	`C_lu`	17.24	$\mathrm{kg/s}$
Quadratic drag coefficient, *surge* (u)	`C_qu`	106.03	$\mathrm{kg/m}$
Linear drag coefficient, *heave* (w)	`C_lw`	38.06	$\mathrm{kg/s}$
Quadratic drag coefficient, *heave* (w)	`C_qw`	84.1	$\mathrm{kg/m}$
Linear drag coefficient, *pitch* (q)	`C_lq`	1.18	$\mathrm{kg\,m^2}$
Quadratic drag coefficient, *pitch* (q)	`C_qq`	7.51	$\mathrm{kg\,m^2/rad^2}$
Ingresso, spinta lungo u	`tau_u`	40	N
Ingresso, coppia che regola *pitch* (q)	`tau_q`	$2.2 \sin 0.1969t$	$\mathrm{N\,m}$
Distanza della boa in superficie dall'origine	`D1`	300	m
Profondità del fondale	`D2`	50	m
Varianza disturbo sull'ingresso, componente u	`var_w_u`	0.1	$\mathrm{N^2}$
Varianza disturbo sull'ingresso, componente w	`var_w_w`	0.1	$\mathrm{N^2}$
Varianza disturbo sull'ingresso, componente q	`var_w_q`	0.05	$\mathrm{N^2\,m^2}$
Varianza del rumore, profondimetro (z)	`var_v_z`	0.4	$\mathrm{m^2}$
Frequenza profondimetro (z)	`T_z`	30	Hz
Varianza del rumore, giroscopio (θ)	`var_w_theta`	0.1	$\mathrm{rad^2}$
Frequenza giroscopio (θ)	`T_theta`	50	Hz
Varianza del rumore, sensore di distanza (D_b)	`var_w_db`	0.2	$\mathrm{m^2}$
Frequenza sensore di distanza (D_b)	`T_db`	40	Hz

3 Progettazione

Per la risoluzione del problema sono stati adottati MATLAB e l'ambiente di lavoro Simulink. Su Simulink è stato progettato lo schema a blocchi del sistema, inclusivo di sensori e filtri, mentre il workspace di MATLAB è stato utilizzato per eseguire semplici script relativi all'inizializzazione di parametri e condizioni iniziali e altro che verrà descritto dove opportuno. Il lavoro si è articolato nei seguenti passaggi:

- ➢ Implementazione del sistema fisico
- ➢ Modellazione dei sensori
- ➢ Implementazione del filtro di Kalman esteso
- ➢ Implementazione del filtro a particelle
- ➢ Rauch-Tung-Striebel smoothing per l'EKF
- ➢ Produzione di grafici

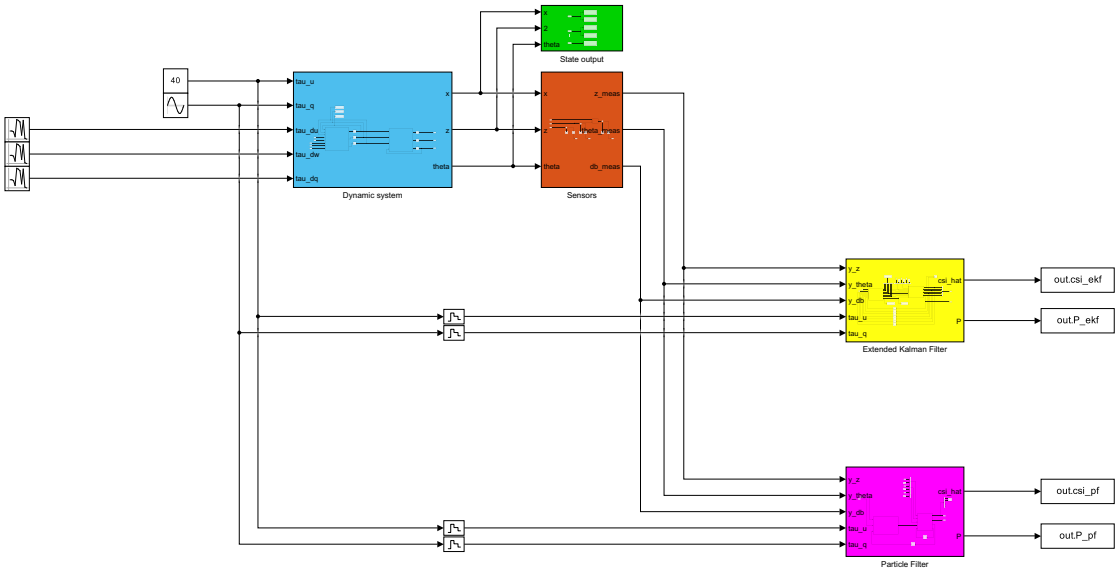

Figura 2: Vista Simulink del diagramma a blocchi del sistema

3.1 Implementazione del sistema fisico

In questa parte sono state implementate le equazioni che caratterizzano il sistema fisico, (1) e (2). Il corrispondente *subsystem* nello schema a blocchi in Figura 2 è rappresentato in azzurro. Dati gli ingressi τ_u e τ_q, rispettivamente la spinta lungo l'asse di avanzamento del moto e la coppia che regola l'angolo di beccheggio, dati i disturbi sugli ingressi τ_{du}, τ_{dw} e τ_{dq} lungo ogni coordinata in terna body, vengono calcolate le componenti x, z e θ dello stato del sistema che descrivono la posizione effettiva dell'AUV. Nella simulazione descritta in questa relazione sono stati utilizzati come ingressi una τ_u costante e una τ_q sinusoidale, mentre τ_{du}, τ_{dw} e τ_{dq} sono stati modellati come numeri casuali generati con una certa varianza. I valori scelti sono stati elencati nella tabella presentata nella Sezione 2 di questa relazione e sono eventualmente modificabili a piacimento andando ad interagire con i rispettivi blocchi e i corrispondenti valori nello script `param.m`. In Figura 3 è riportato lo schema a blocchi del sottosistema relativo all'AUV.

Per poter inserire le equazioni nel modello è stato utilizzato il blocco Simulink *MATLAB Function*, al quale viene associato uno script modificabile direttamente dal diagramma a blocchi. Specificando adeguatamente ingressi, uscite e parametri, Simulink prende i dati di cui ha bisogno per svolgere i calcoli direttamente

dal workspace o dai valori calcolati durante la simulazione.

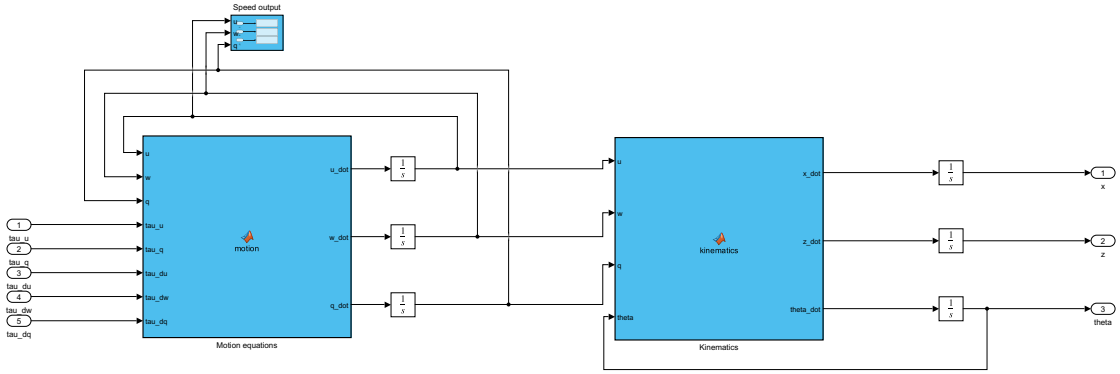

Figura 3: Sottosistema del modello dell'AUV

Il primo dei due blocchi funzione contiene il codice seguente:

```
function [u_dot,w_dot,q_dot] = motion(u,w,q,tau_u,tau_q,tau_du,tau_dw,tau_dq,m,I,C_qu,C_lu,
    C_lw,C_qw,C_lq,C_qq)

% Inversione della dinamica del sistema per ricavare le accelerazioni,
% portiamo le informazioni in forma matriciale per semplicità

    U = [u w q]';                   % Vettore velocità
    B = [m 0 0;                     % Matrice inerzia
         0 m 0;
         0 0 I];
    C = [0 0 m*w;                   % Matrice Coriolis
         0 0 -m*u;
        -m*w m*u 0];
    F = [C_lu+C_qu*abs(u) 0 0;      % Matrice reistenza avanzamento
```

```
14          0 C_lw+C_qw*abs(w) 0;
15          0 0 C_lq + C_qq*abs(q)];
16      Q = [tau_u 0 tau_q]';           % Vettore ingressi
17      D = [tau_du tau_dw tau_dq]';    % Vettore disturbi
18
19      O = B\(Q+D-C*U-F*U);            % Inversione dinamica
20
21  % Vengono estratte le singole componenti relative all'accelerazione
22      u_dot=O(1);
23      w_dot=O(2);
24      q_dot=O(3);
25
26  end
```

Per descrivere in maniera compatta le equazioni della dinamica, ci si è ricondotti alla rappresentazione in forma matriciale come nell'equazione (1), dalla quale sono state isolate le componenti dell'accelerazione $\dot{u}$, $\dot{w}$ e $\dot{q}$. Questo calcolo avviene alla riga 19 del codice presentato, previa premoltiplicazione per l'inversa di B. Infine, le tre componenti sono state estratte dal vettore O per poter gestire singolarmente le tre grandezze all'interno del diagramma a blocchi.

Le grandezze da passare al blocco successivo vanno integrate, dal momento che nell'equazione (2) compaiono velocità e non accelerazioni. Il blocco *Integrator* di Simulink svolge proprio questo compito. Inoltre, è possibile specificare all'interno di questi blocchi le condizioni del sistema nell'istante iniziale. Nella simulazione in esame, sono state considerate velocità nulle, modificabili come gli altri valori presentati in precedenza. Per quanto riguarda il blocco relativo alla cinematica, il codice della funzione è il seguente:

```
1   function [x_dot,z_dot,theta_dot] = kinematics(u,w,q,theta)
2
3   % Definizione della matrice di rotazione
4       Rz = [cos(theta) sin(theta) 0;
5            -sin(theta) cos(theta) 0;
6            0 0 1];
7
8   % Cambio del sistema di riferimento da terna body a sistema fisso
9       out = Rz*[u w q]';
10
11  % Vengono estratte le singole componenti relative alle velocità
12      x_dot = out(1);
13      z_dot = out(2);
14      theta_dot = out(3);
15
16  end
```

Qui si è semplicemente applicata l'equazione (2) (riga 9) così da ottenere le componenti della velocità nel sistema di riferimento fisso. A questo punto è necessario un altro step di integrazione per ottenere le componenti del vettore di stato, che rappresenta la posizione dell'AUV. Anche in questo caso è possibile specificare delle condizioni iniziali. Nel caso in esame, $x = 150\,\mathrm{m}$, $z = 25\,\mathrm{m}$ e $\theta = 0\,\mathrm{rad}$.

3.2 Modellazione dei sensori

Nell'immagine sottostante è raffigurato lo schema a blocchi del modello dei tre sensori impiegati.

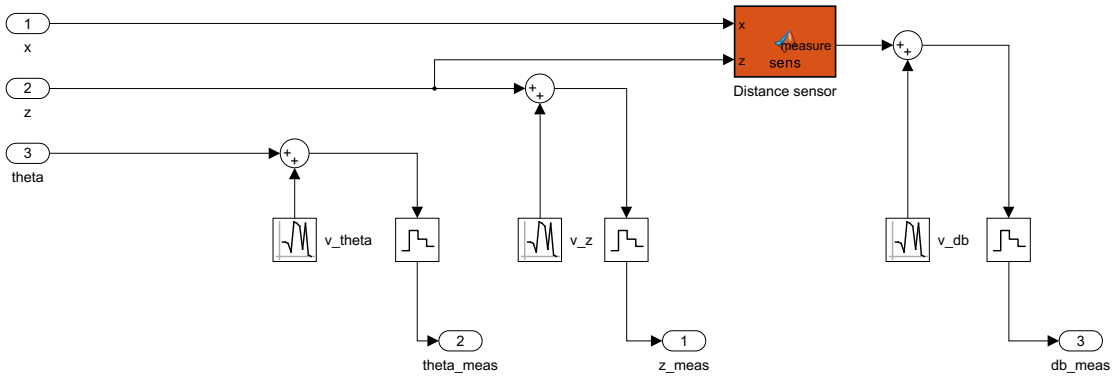

Figura 4: Sottosistema del modello dei sensori

Per quanto riguarda il profondimetro e il giroscopio, non è stato necessario implementare nessuna funzione che restituisse la lettura effettiva del sensore dal momento che z e θ sono già disponibili in uscita dal sistema dinamico. Ai segnali in ingresso è stato sommato un rumore di misura additivo, modellato come un numero casuale generato con una certa varianza. Inoltre, il segnale è stato poi discretizzato tramite un blocco *Zero-Order Hold* con un tempo di campionamento pari a quello del corrispettivo sensore. Si noti che nel file `param.m` vengono dichiarati i valori relativi alle frequenze dei sensori, che sono poi utilizzati per calcolare i tempi di campionamento. Il sensore di distanza posizionato sulla boa ha richiesto un semplice calcolo basato sulla x e sulla z:

```
function measure = sens(x,z,D1)

    measure = sqrt((D1-x)^2+z^2);

end
```

Tramite il teorema di Pitagora si risale quindi all'effettiva misura del sensore, anch'essa sommata ad un

rumore additivo ed in seguito discretizzata.

3.3 Implementazione del filtro di Kalman esteso

Una volta modellato il sistema fisico con i suoi sensori si giunge al cuore del problema: l'implementazione degli algoritmi per la stima dello stato dell'AUV a partire da ingressi e misure rumorose. In Figura 5 è rappresentato lo schema a blocchi del filtro di Kalman esteso.

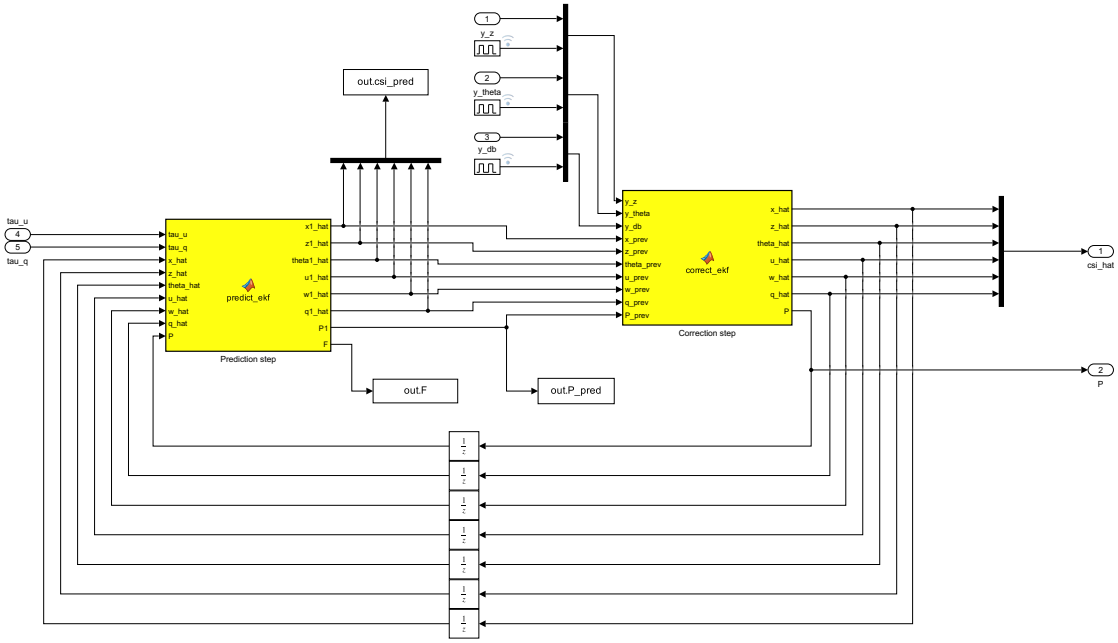

Figura 5: Sottosistema dell'EKF

L'algoritmo ricorsivo del filtro è stato implementato suddividendolo nelle due parti di predizione e di correzione. Si noti l'utilizzo dei blocchi *Unit Delay*, necessario per prevenire un loop algebrico grazie all'inizializzazione del filtro. All'interno di tali blocchi, infatti, è possibile specificare le condizioni iniziali, ancora una volta modificabili nello script `param.m`.

```
%% EKF
    % Inizializzazione algoritmo
    P_init = eye(6);
    q_hat_init = 0;                   %[rad/s]
    w_hat_init = 0;                   %[m/s]
    u_hat_init = 0;                   %[m/s]
    theta_hat_init = pi/8*rand(1,1);  %[rad]
    z_hat_init = 20+10*rand(1,1);     %[m]
    x_hat_init = 140+20*rand(1,1);    %[m]
```

Per dare il via all'algoritmo, è necessario infatti avere dei valori di partenza per la stima dello stato e una matrice di covarianza associata, che è stata considerata pari all'identità. Com'è possibile notare dall'estratto dello script qui presentato, sono stati assegnati valori iniziali alla stima anche per quanto riguarda le velocità. Per rendere i calcoli più semplici, è stata presa la decisione di utilizzare un vettore di stato esteso che includesse anche le velocità per i motivi che verranno spiegati a breve.

Lo step di predizione consiste nell'applicazione del modello allo stato filtato (in uscita dal blocco di correzione) per ottenere una stima predetta dello step successivo, prendendo in ingresso gli ingressi (opportunamente discretizzati) del sistema e la stima filtrata in uscita dal passo di correzione. Si noti che nel caso del filtro di Kalman esteso si considerano i disturbi sugli ingressi pari a zero nel calcolo dei jacobiani F e D.

```
function [x1_hat,z1_hat,theta1_hat,u1_hat,w1_hat,q1_hat,P1,F] = predict_ekf(tau_u,tau_q,x_hat
    ,z_hat,theta_hat,u_hat,w_hat,q_hat,P,I,m,C_qq,C_lq,C_qw,C_lw,C_qu,C_lu,var_w_u,var_w_q,
    var_w_w,sample_time)

    Q = [var_w_u 0 0;
         0 var_w_w 0;     % Covarianza del disturbo sugli ingressi
         0 0 var_w_q];

%% PREDIZIONE
    %  X_k+1 = X_k + dt * X_dot
    %  predizione vettore di stato esteso
    x1_hat = x_hat + sample_time*(u_hat*cos(theta_hat) + w_hat*sin(theta_hat));
    z1_hat = z_hat + sample_time*(-u_hat*sin(theta_hat) + w_hat*cos(theta_hat));
    theta1_hat = theta_hat + sample_time*q_hat;
    u1_hat = u_hat + sample_time*(-m*w_hat*q_hat - C_lu*u_hat - C_qu*abs(u_hat)*u_hat + tau_u
        )/m;
    w1_hat = w_hat + sample_time*(m*u_hat*q_hat - C_lw*w_hat - C_qw*abs(w_hat)*w_hat)/m;
    q1_hat = q_hat + sample_time*(-C_lq*q_hat - C_qq*abs(q_hat)*q_hat + tau_q)/I;

%% LINEARIZZAZIONE

    % F jacobiano del modello di processo f rispetto allo stato valutato
    % in x = x_pred k|k e Wk=0
    F = [1 0 sample_time*(-u_hat*sin(theta_hat)+w_hat*cos(theta_hat)) sample_time*cos(
        theta_hat) sample_time*sin(theta_hat) 0;
         0 1 sample_time*(-u_hat*cos(theta_hat)-w_hat*sin(theta_hat)) sample_time*(-sin(
             theta_hat)) sample_time*cos(theta_hat) 0;
         0 0 1 0 0 sample_time;
```

```
24          0 0 0 1+sample_time*(-C_lu-C_qu*(abs(u_hat)+u_hat*sign(u_hat)))/m  sample_time*(-
                q_hat) sample_time*(-w_hat);
25          0 0 0 sample_time*q_hat 1+sample_time*(-C_lw-C_qw*(abs(w_hat)+w_hat*sign(w_hat)))/m
                 sample_time*u_hat;
26          0 0 0 0 0 1+sample_time*(-C_lq-C_qq*(abs(q_hat)+q_hat*sign(q_hat)))/I];
27
28      % D jacobiano del modello di processo rispetto ai disturbi valutato
29      % nello stesso punto di linearizzazione
30      D = [0 0 0;
31           0 0 0;
32           0 0 0;
33           sample_time*1/m 0 0;
34           0 sample_time*1/m 0;
35           0 0 sample_time*1/I];
36
37      % Covarianza della stima al passo k+1|k
38      P1 = F*P*F' + D*Q*D';
39
40  end
```

Anzitutto, dal momento che questa implementazione del filtro di Kalman esteso opera in regime di tempo discreto, è stato necessario utilizzare un metodo per l'integrazione numerica delle equazioni differenziali che caratterizzano il sistema. In questo caso, è stato impiegato il metodo di Eulero. Discretizzando sulla variabile temporale, con $t_{n+1} = t_n + h$, la soluzione è approssimata nel modo seguente:

$$y_{n+1} = y_n + hf(t_n, y_n)$$

L'intervallo di tempo discreto h è quello che nel codice è definito come `sample_time`. Tale valore è dichiarato all'interno dello script di inizializzazione ed è stato scelto pari al più piccolo tra i tempi di campionamento dei sensori (`sample_time = min([T_z T_theta T_db])`).

Le equazioni risolte numericamente in questo modo sono le (1) e (2). Da qui il motivo per cui si è scelto di utilizzare un vettore di stato esteso: dovendo esprimere le sole x, z e θ in funzione dello stato, che però compaiono in (2) come funzione di u, w e q, sarebbe stato necessario esprimere queste ultime in funzione di x, z e θ, cosa che avrebbe portato ad espressioni complicate e poco leggibili. Così facendo, invece, le sei equazioni risultano già scritte nella forma adeguata al calcolo dei jacobiani d'interesse.

Dopo aver calcolato lo stato predetto tramite il metodo di Eulero, vengono ricavati F e D come visto in (5) e (6). Il calcolo di questi jacobiani avviene ad ogni passo, sulla base dello stato filtrato in uscita dallo step di correzione. Sulla base di questi, insieme a Q, matrice di covarianza dei disturbi sulle componenti dell'ingresso, viene calcolata la covarianza associata alla stima in predizione, in corrispondenza della riga 38 del codice.

In merito al passo di correzione, il blocco prende in ingresso lo stato predetto e le misure dei sensori. Anche in correzione si considerano nulli i rumori di misura nel valutare i jacobiani H ed M.

```
1   function [x_hat,z_hat,theta_hat,u_hat,w_hat,q_hat,P] = correct_ekf(y_z,y_theta,y_db,x_prev,
        z_prev,theta_prev,u_prev,w_prev,q_prev,P_prev,D1,var_v_db,var_v_theta,var_v_z)
2
3   %% Gestione delle misure dei sensori e inizializzazione
4       % pulse = 1 -> arriva una nuova misura del sensore
5       % pulse = 0 -> non c'è nessuna nuova misura
6
```

```
    % Misure sensori
    meas_z = y_z(1,:);                                  % misura profondità z
    meas_theta = y_theta(1,:);                          % misura angolo beccheggio theta
    meas_db = y_db(1,:);                                % misura distanza db

    pulse_z = y_z(2,:);                                 % impulso profondimetro
    pulse_theta = y_theta(2,:);                         % impulso giroscopio
    pulse_db = y_db(2,:);                               % impulso sensore di distanza

    % Inizializzo le variabili utilizzate nei calcoli
    % N.B. dimensione variabile sulla base di quante nuove misure ricevo

    % Jacobiani del modello di misura rispetto allo stato e ai rumori
    H = zeros(pulse_z+pulse_theta+pulse_db,6);
    M = eye(pulse_z+pulse_theta+pulse_db);
    % Digonale matrice di covarianza dei rumori sulle misure
    R_vec = zeros(pulse_z+pulse_theta+pulse_db,1);

    % Stato predetto in ingresso
    xi_prev = [x_prev z_prev theta_prev u_prev w_prev q_prev]';
    % Inizializzazione innovazione
    e = zeros(pulse_z+pulse_theta+pulse_db,1);

%% Inserisco i valori corrispondenti alle misure disponibili

    % Se la misura per z è disponibile, la corrispondente riga del
    % jacobiano è la prima, per i vettori è il primo elemento.

    if pulse_z == 1
        H(1,:) = [0 1 0 0 0 0];
        R_vec(1) = var_v_z;
        e(1) = meas_z - z_prev;
    end

    % Se le misure per z e theta sono entrambe disponibili, i dati relativi
    % a theta sono posizionati nella seconda riga. Se è disponibile solo
    % theta allora i dati sono posizionati nella prima riga. Ugualmente per
    % i vettori.

    if pulse_theta == 1
        H(pulse_z+pulse_theta,:) = [0 0 1 0 0 0];
        R_vec(pulse_z+pulse_theta) = var_v_theta;
        e(pulse_z+pulse_theta) = wrapToPi(meas_theta - theta_prev);
    end

    % In maniera del tutto analoga per db

    if pulse_db == 1
        H(pulse_z+pulse_theta+pulse_db,:) = [-(D1-x_prev)/sqrt((D1-x_prev)^2+z_prev^2) z_prev
            /sqrt((D1-x_prev)^2+z_prev^2) 0 0 0 0];
```

```
56          R_vec(pulse_z+pulse_theta+pulse_db) = var_v_db;
57          e(pulse_z+pulse_theta+pulse_db) = meas_db - sqrt((D1-x_prev)^2+z_prev^2);
58      end
59
60  %% Applico la correzione
61      R = diag(R_vec);                    % Matrice covarianza disturbo sensori
62      S = H*P_prev*H' + M*R*M';           % Matrice covarianza innovazione
63      L = P_prev*H'/S;                    % Guadagno correzione
64
65      % Stato corretto ->  X_k|k
66      xi_hat = xi_prev + L*e;
67      % Matrice di covarianza associata alla stima -> P_k|k
68      P = (eye(6) - L*H)*P_prev*(eye(6) - L*H)' + L*M*R*M'*L';
69
70  %% Output blocco di correzione
71      x_hat = xi_hat(1);
72      z_hat = xi_hat(2);
73      theta_hat = xi_hat(3);
74      u_hat = xi_hat(4);
75      w_hat = xi_hat(5);
76      q_hat = xi_hat(6);
77
78  end
```

Dal momento che nella realtà i sensori impiegati potrebbero lavorare a frequenze differenti, è necessario segnalare l'arrivo di nuove misure da impiegare nello step di correzione. A tal proposito, è stato utilizzato il blocco Simulink *Pulse Generator*. L'obiettivo è quello di generare un segnale impulsivo con frequenza pari a quella del sensore e con ampiezza pari ad 1. In questo modo, in corrispondenza di nuove misure, il segnale è stato impiegato come controllo nelle *if-clauses* che inseriscono i dati relativi al sensore interessato nelle matrici. Inoltre, questi impulsi sono stati usati per definire le dimensioni delle matrici e dei vettori in gioco nell'algoritmo, com'è visibile nella sezione di codice relativa all'inizializzazione: il jacobiano H risulta avere dimensione variabile in base al numero delle misure disponibili, mentre M è la matrice identità della stessa dimensione (le misure sono indipendenti).

Risolto questo problema, dopo aver calcolato l'innovazione viene applicato l'algoritmo vero e proprio, che produrrà un vettore di stato filtrato (corretto) e la matrice di covarianza associata alla stima. Si noti che nella riga 68 è stata impiegata la cosiddetta *forma di Joseph* per evitare problemi di natura numerica.

Oltre a produrre dati relativi alla stima corretta e alla covarianza associata, altre informazioni d'interesse che vengono portate nel workspace di MATLAB sono le serie storiche della stima predetta, della matrice di covarianza ad essa associata e del jacobiano F. Saranno infatti impiegate nell'algoritmo di *smoothing* di Rauch-Tung-Striebel come sarà mostrato più avanti.

3.4 Implementazione del filtro a particelle

Il filtro a particelle è stato implementato per lavorare in parallelo con il filtro di Kalman esteso, in maniera tale da poterne confrontare le performance al termine della simulazione. Anche in questo caso, si è operata una distinzione tra i due blocchi di predizione e correzione (Figura 6). Nei blocchi di ritardo unitario sono stati inseriti i dati per inizializzare l'algoritmo del filtro, modificabili anche stavolta in `param.m`. Per quanto riguarda il particle filter, nell'inizializzazione si va a specificare anche i pesi assegnati alle particelle

a seconda delle informazioni a priori su di esse. In questo caso, le particelle hanno tutte lo stesso peso, dal momento che prima del filtraggio sono caratterizzate da uguale verosimiglianza.

```
%% PARTICLE FILTER
    % Inizializzazione algoritmo
    N=10000;                                % Numero di particelle
    weights_init = 1/N*ones(N,1);           % Pesi (pdf uniforme)

    x_p=149+3*rand(N,1);                    % Condizioni iniziali per le particelle
    z_p=24+2*rand(N,1);
    theta_p=(pi/16)*rand(N,1);
    u_p=zeros(N,1);
    w_p=zeros(N,1);
    q_p=zeros(N,1);
    P0=0.1*eye(6);                          % Matrice di covarianza associata

    xi0=[x_p z_p theta_p u_p w_p q_p];
    particles_init = mvnrnd(xi0, P0, N);  % Genero le particelle
```

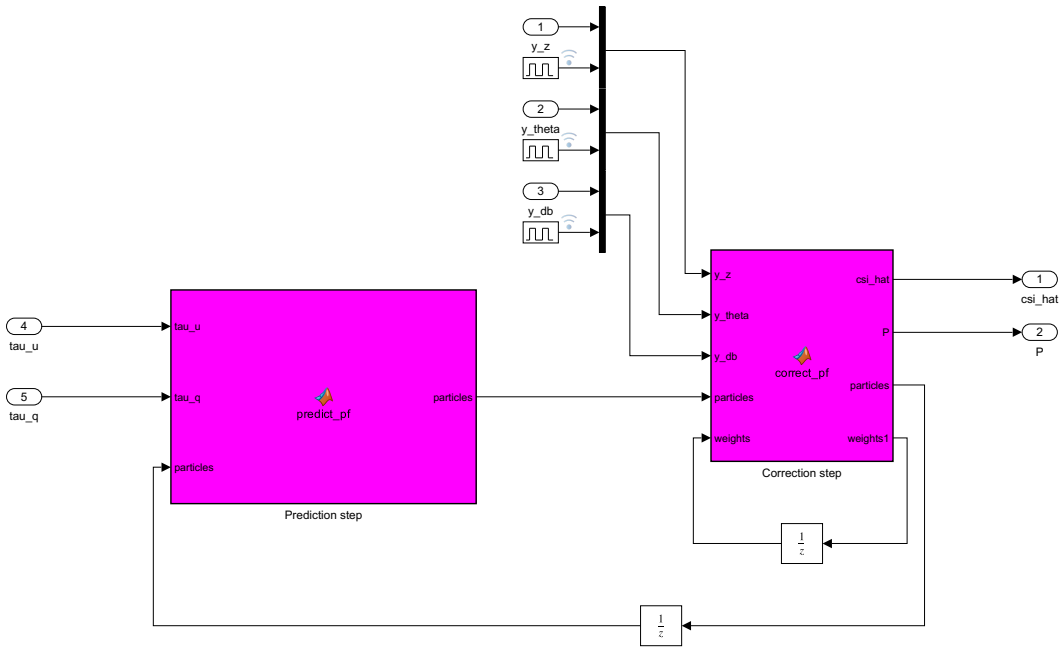

Figura 6: Sottosistema del PF

Il passo di predizione consiste, anche nel caso del filtro a particelle, nell'applicazione del modello fisico, dati come ingressi gli ingressi del sistema. Stavolta però, dopo aver generato una realizzazione di rumore per ogni singola particella, ognuna di esse sarà propagata attraverso le equazioni del sistema.

```
function particles = predict_pf(tau_u,tau_q,particles,var_w_u,var_w_w,var_w_q,sample_time,m,
    C_qq,C_lq,C_qw,C_lw,C_qu,C_lu,I)

    for i=1:size(particles,1)

        % Realizzazione di rumore coerente per una singola particella
        noise_u = sqrt(var_w_u)*randn(1,1);
        noise_w = sqrt(var_w_w)*randn(1,1);
        noise_q = sqrt(var_w_q)*randn(1,1);

        %% PREDIZIONE
        % predizione -> applico il modello del processo f
        % STEP di predizione    X_k+1|k = f(X_k|k,...,W_k)  applico modello
        %                       W_k+1|k = W_k|k pesi invariati
        particles(i,1) = particles(i,1) + sample_time*(particles(i,4)*cos(particles(i,3)) +
            particles(i,5)*sin(particles(i,3)));
        particles(i,2) = particles(i,2) + sample_time*(-particles(i,4)*sin(particles(i,3)) +
            particles(i,5)*cos(particles(i,3)));
        particles(i,3) = particles(i,3) + sample_time*particles(i,6);
        particles(i,4) = particles(i,4) + sample_time*(-m*particles(i,5)*particles(i,6) -
            C_lu*particles(i,4) - C_qu*abs(particles(i,4))*particles(i,4) + tau_u + noise_u)/
            m;
        particles(i,5) = particles(i,5) + sample_time*(m*particles(i,4)*particles(i,6) - C_lw
            *particles(i,5) - C_qw*abs(particles(i,5))*particles(i,5) + noise_w)/m;
        particles(i,6) = particles(i,6) + sample_time*(-C_lq*particles(i,6) - C_qq*abs(
            particles(i,6))*particles(i,6) + tau_q + noise_q)/I;

    end

end
```

Ancora una volta, le equazioni differenziali del sistema sono state risolte numericamente tramite il metodo di Eulero. Dalle particelle così ottenute in seguito al passo di predizione, lo scopo è quello di ricalcolarne i pesi sulla base di quali saranno più aderenti delle altre alle misure ottenute dai sensori. Il codice del passo di correzione è il seguente:

```
function [csi_hat,P,particles,weights1] = correct_pf(y_z,y_theta,y_db,particles,weights,D1,
    var_v_theta,var_v_db,var_v_z)

    % Si ricava la deviazione standard dalla varianza dei rumori di misura
    % per utilizzarla nella funzione normpdf
    sigma_z = sqrt(var_v_z);
    sigma_theta = sqrt(var_v_theta);
    sigma_db = sqrt(var_v_db);

%% Misure | Impulsi sensori
    meas_z = y_z(1,:);
```

```
    meas_theta = y_theta(1,:);
    meas_db = y_db(1,:);

    pulse_z = y_z(2,:);
    pulse_theta = y_theta(2,:);
    pulse_db = y_db(2,:);

%% Parametri per il ricampionamento
    threshold = 0.8;        % Frequenza ricampionamento: 1 -> sempre, 0 -> mai
    eps = 0.006;            % Disturbo per la generazione di nuovi campioni -> distribuisce
                            % le particelle per evitare che ce se siano di uguali

%% Calcolo della verosimiglianza
    % Essendo un prodotto, se la misura è disponibile (pulse = 1) allora viene
    % calcolata la verosimiglianza, altrimenti assumerà valore 1

    % Inizializzazione likelihood per le singole misure
    l_z = ones(size(particles,1),1);
    l_theta = ones(size(particles,1),1);
    l_db = ones(size(particles,1),1);

    if pulse_z == 1
        for i=1:size(particles,1)
            l_z(i) = normpdf(meas_z-particles(i,2),0,sigma_z);
        end
    end

    if pulse_theta == 1
        for i=1:size(particles,1)
            l_theta(i) = normpdf(meas_theta-particles(i,3),0,sigma_theta);
        end
    end

    if pulse_db == 1
        for i=1:size(particles,1)
            l_db(i) = normpdf(meas_db-sqrt((D1-particles(i,1))^2+particles(i,2)^2),0,sigma_db
                );
        end
    end

    likelihood = l_z.*l_theta.*l_db;
    % Nuovi pesi come pesi a priori per la likelihood, poi normalizzo
    weights1 = (weights.*likelihood)/sum(weights.*likelihood);

%% Ricampionamento
    N_eff = 1/sum((weights1).^2);                   % Numero efficace di particelle

    % 0     S1          S2...    1  sommatoria s=1
    % |-----|-----------|-------|
    %   w1      w2          w..
```

```
60      % a estratto tra 0 e 1 uniformemente, a apparterrà ad uno dei segmenti
61      % Si; è più probabile estrarre in corrispondenza di un segmento più
62      % ampio i.e. peso più alto -> rigenero più campioni in corrispondenza
63      % del segmento (e quindi particella) con peso più alto
64
65      if N_eff/size(particles,1) <= threshold          % Entro nel ciclo se sotto la soglia
66          s = cumsum(weights1);
67          new_particles = zeros(size(particles));
68          for j = 1:size(particles,1)
69              a = rand(1);
70              dist = eps*randn(6,1);
71
72              % Rigenero i campioni con probabilità definita dai pesi e li
73              % perturbo per evitare impoverimento
74              i = find(s>=a,1);
75              new_particles(j,:) = particles(i,:) + dist';
76          end
77          % Ricalcolo i pesi distribuendoli uniformemente tra le particelle ricampionate
78          new_weights = 1/size(weights1,1)*ones(size(weights,1),1);
79          % Aggiorno le variabili
80          particles = new_particles;
81          weights1 = new_weights;
82      end
83
84  %% Calcolo della stima dello stato come media pesata delle particelle
85      csi_hat = particles'*weights1;
86
87      % Matrice di covarianza associata alla stima
88      P = zeros(6,6);
89      for i=1:size(particles,1)
90          P = P + weights1(i)*(particles(i,:)' - csi_hat)*(particles(i,:)' - csi_hat)';
91      end
92
93  end
```

Per risolvere il problema dei sensori asincroni, si è adottata la stessa soluzione. Tuttavia, in questo caso il codice è più semplice: la disponibilità di una misura implica solamente il calcolo della corrispettiva verosimiglianza e dunque, dal momento che le tre quantità andranno moltiplicate tra loro, nel caso di assenza di una delle misure basterà sostituire la relativa likelihood con 1. Per questo motivo, le tre quantità vengono inizializzate con la funzione `ones`.

Per quanto riguarda la likelihood, essa viene ottenuta dalla funzione `normpdf`. In questo modo, andando a generare una distribuzione gaussiana a media nulla con deviazione standard pari a quella dei rumori di misura, più lo stato (o una funzione di esso, come nel caso della misura d_B) della singola particella è vicino alla misura, più ci si troverà in corrispondenza del punto più alto della curva a campana, e da qui si ottiene una likelihood massima. Ne consegue però che basta una sola misura discordante, e quindi una sola likelihood molto bassa, per portare verso zero l'intero prodotto e quindi, in conclusione, per scartare la particella. I pesi vengono infatti calcolati come segue (riga 52):

$$w^i_{k|k} = \frac{w^i_{k|k-1} \cdot f_{Y|X}(y_k|x^i_{k|k-1})}{\sum_{j=1}^{N} w^j_{k|k-1} \cdot f_{Y|X}(y_k|x^j_{k|k-1})}$$

ovvero moltiplicando i pesi a priori per la verosimiglianza e normalizzando poi sul totale.

Ottenuti anche i nuovi pesi, dal momento che in correzione non vengono modificate le particelle, sarebbe possibile calcolare la stima come media pesata dei campioni. Tuttavia, dopo un numero relativamente breve di iterazioni del filtro, si potrebbe andare incontro al fenomeno della degenerazione dei pesi: poche particelle (nel caso più estremo anche solamente una) sarebbero caratterizzate da un peso altissimo, mentre tutte le altre verrebbero scartate. Questa condizione è particolarmente fastidiosa dal punto di vista computazionale siccome per quei campioni, anche se caratterizzati da un peso praticamente nullo, si continuerebbe ad eseguire tutti i calcoli richiesti dal filtro, nonostante sia inutile. Per ovviare a ciò si impiega solitamente un algoritmo di *resampling*, ricampionamento: le particelle vengono rigenerate secondo una probabilità definita dai loro pesi. Se il resampling viene adottato ad ogni iterazione si parla di *SIR-PF* (*Sequential Importance Resampling - Particle Filter*). Dal momento che l'operazione è piuttosto onerosa dal punto di vista computazionale, con risorse limitate si preferisce effettuare un ricampionamento ogni volta che il numero di particelle contribuenti in maniera effettiva alla stima scende al di sotto di una certa soglia. Si parla quindi di particelle *efficaci*, il cui numero viene stimato come segue:

$$N_{EFF} = \frac{1}{\sum_{i=1}^{N} (w^i_{k|k})^2}$$

Si valuta quindi confrontando N_{EFF}/N con il valore di soglia.

L'algoritmo impiegato estrae un indice uniformemente distribuito tra 0 ed 1. L'intervallo $[0, 1]$ viene quindi suddiviso in sottointervalli, con estremo superiore S_i, nel seguente modo:

$$S_1 = \overline{w_1} \longrightarrow [0, S_1]$$
$$S_i = S_{i-1} + \overline{w_i}; i = 2, ..., N \longrightarrow [S_{i-1}, S_i]$$

dove $\overline{w_i}$ sono i pesi ricalcolati in correzione. A questo punto, a pesi maggiori corrispondono intervalli più ampi, e dunque è più probabile che l'indice ricada nell'intervallo che corrisponde ad una particella con un peso elevato. Nella riga 74 del codice viene quindi individuato l'estremo superiore di questo intervallo e dunque la corrispondente i-esima particella. Per evitare impoverimento dei campioni (ovvero molte particelle replicate), essi vengono perturbati secondo un valore (nel codice `eps`) abbastanza piccolo da non rovinare le performance del filtro. Infine, ai campioni rigenerati vengono assegnati pesi uniformemente distribuiti e si prosegue nell'algoritmo.

In conclusione, la stima dello stato viene ricavata come media pesata delle particelle (riga 85). Nelle righe 88-91 viene effettuato il calcolo della matrice di covarianza associata alla stima, data un'approssimazione discreta (con le particelle) della funzione di distribuzione di probabilità.

3.5 Rauch-Tung-Striebel *smoothing* per l'EKF

Prima di discutere i risultati ottenuti, viene adesso brevemente descritto l'algoritmo di *smoothing*, o regolarizzazione, applicato sulla stima del filtro di Kalman esteso.

```
%% SMOOTHING
    % Rinomino le variabili in uscita da Simulink
    sc=(out.csi_ekf.Data);              % Stima dello stato corretta (filtrata)
    vp=(out.csi_pred.Data);             % Stima dello stato predetta
    P_correction=out.P_ekf.Data;        % Covarianza associata alla stima corretta (filtrata)
    P_prediction=out.P_pred.Data;       % Covarianza associata alla stima predetta
    F=out.F.Data;                       % Jacobiano F del modello di processo

    % Inizializzazione dello stato regolarizzato
    P_smooth(:,:,size(P_correction,3)) = P_correction(:,:,size(P_prediction,3));
    smoothed_state(size(sc,1),:) = sc(size(sc,1),:);

    % L'algoritmo parte dalle considerazioni finali operando a ritroso fino alle condizioni
        iniziali del filtro EKF
    % N.B. parto da n-1, la predizione fa riferimento alla stima filtrata al passo precedente
    for k = size(vp,1)-1:-1:1
        % Applico l'algoritmo di Rauch-Tung-Striebel
        Ck = P_correction(:,:,k)*F(:,:,k+1)'*inv(P_prediction(:,:,k+1));
        smoothed_state(k,:)=sc(k,:)'+Ck*(smoothed_state(k+1,:)-vp(k+1,:))';
        P_smooth(:,:,k) = P_correction(:,:,k) + Ck*(P_smooth(:,:,k+1) - P_prediction(:,:,k+1)
            )*Ck';
    end
```

Questa operazione avviene *offline*: è necessario avere a disposizione la serie storica della stima in uscita sia dal passo di predizione che di correzione, oltre alle matrici di covarianza loro associate. Inoltre sono richieste tutte le matrici F calcolate in predizione ad ogni passo.

La regolarizzazione avviene su due passi: uno in avanti ed uno all'indietro (da qui la necessità di eseguire lo script offline al termine della simulazione). Il passo in avanti è il filtro di Kalman stesso, mentre il passo all'indietro è un'ulteriore correzione che avviene come segue:

$$C_k = P_{k|k} F_{k+1}^T P_{k+1|k}^{-1}$$

$$\hat{x}_{k|n} = \hat{x}_{k|k} + C_k(\hat{x}_{k+1|n} - \hat{x}_{k+1|k})$$

$$P_{k|n} = P_{k|k} + C_k(P_{k+1|n} - P_{k+1|k})C_k^T$$

In questo modo è possibile ottenere andamenti più morbidi delle traiettorie stimate, oltre a ridurre l'incertezza negli istanti iniziali.

4 Conclusioni

Entrambi i filtri sono stati implementati con successo. In Figura 7 è possibile osservare la traiettoria reale sul piano xz (tempo di simulazione $t = 50s$) e la stima dello stato (per quanto riguarda le sole componenti x e z) data da entrambi i filtri.

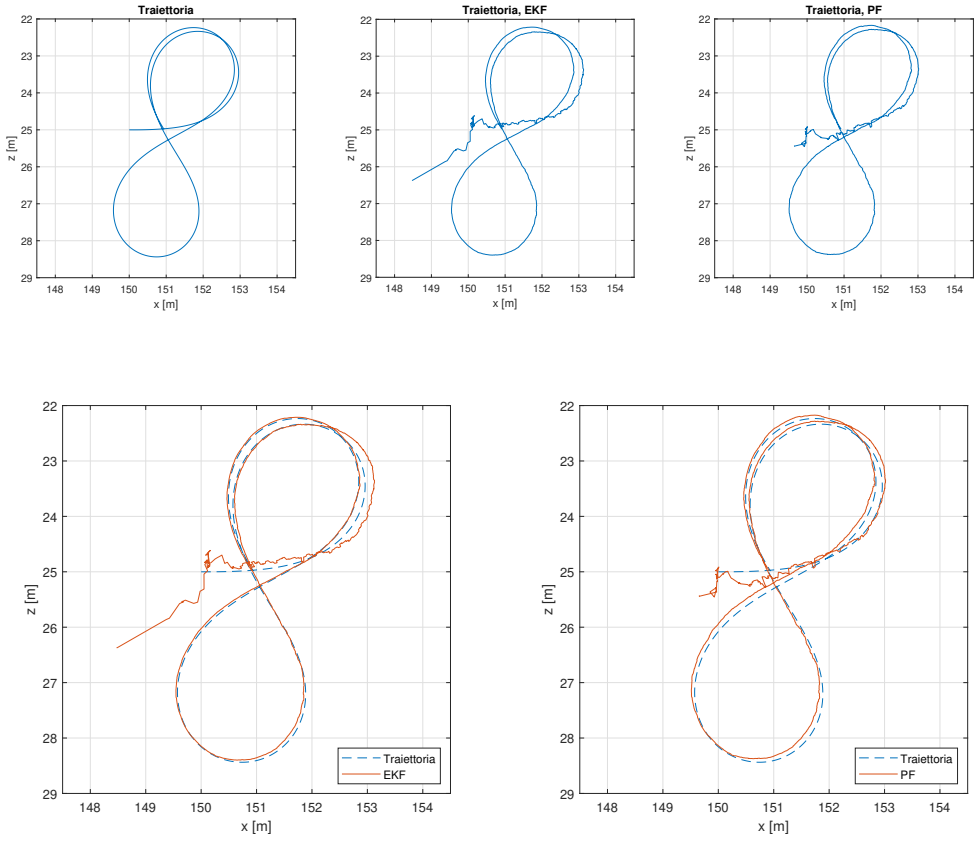

Figura 7: In alto, da sinistra verso destra, la traiettoria reale, quella stimata dall'EKF, quella stimata dal PF. In basso, confronti tra la traiettoria reale e le stime.

La prima considerazione da fare è relativa ai valori scelti per inizializzare i filtri. Nel caso del filtro a particelle, infatti, i campioni sono stati generati con coordinate che spaziano in un range più stretto di quello del punto da cui parte la stima del filtro di Kalman. Infatti, scegliendo condizioni meno restrittive, sarebbero richieste più particelle per poter ottenere prestazioni accettabili e ciò risulterebbe particolarmente oneroso dal punto di vista computazionale.

Entrambi i filtri esibiscono un comportamento accettabile, tuttavia presentano vantaggi differenti. Il filtro di Kalman esteso ha un carico computazionale di gran lunga più leggero del filtro a particelle. Infatti, applicare le equazioni del modello a tutti i campioni richiede l'esecuzione di N operazioni, cosa che nel

filtro di Kalman esteso avviene una sola volta per step. Inoltre, la ripetuta generazione di numeri casuali, sia per le realizzazioni di rumore che per il resampling, e l'utilizzo della funzione `normpdf` per il calcolo delle verosimiglianze sono compiti particolarmente onerosi per il software, che aumentano in maniera molto significativa il tempo richiesto per completare la simulazione. D'altro canto, il filtro di Kalman esteso è prono ad errori di approssimazione a causa della linearizzazione al primo ordine con i jacobiani e semplificare il modello in questo modo è tanto più brutale quanto più marcata è la non-linearità del sistema. Il filtro a particelle, invece, sotto questo punto di vista è decisamente più robusto, seppur limitato dalla potenza di calcolo di cui si dispone. In Figura 8 sono mostrati gli errori di stima dei filtri per ognuna delle componenti del vettore di stato esteso.

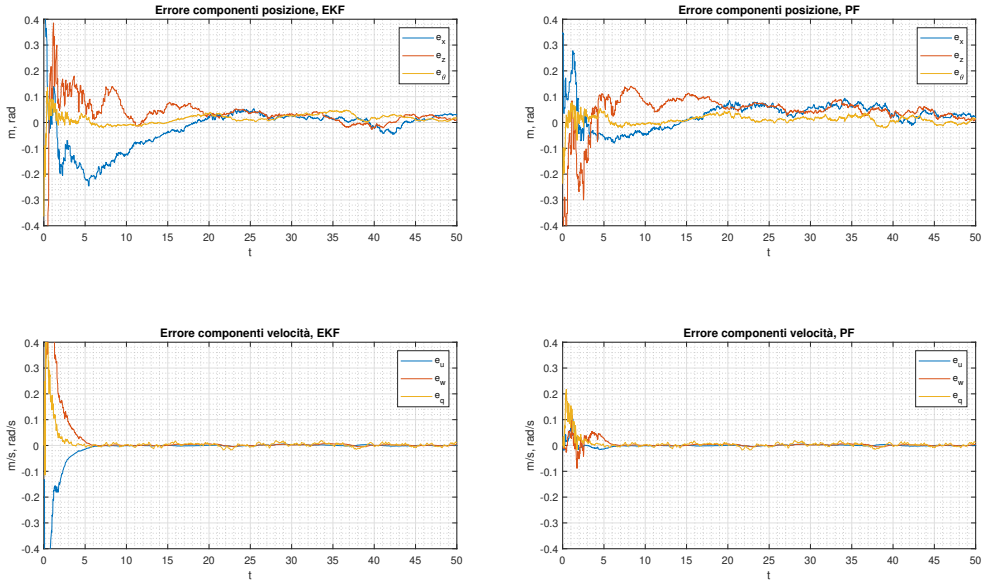

Figura 8: In alto, gli errori di stima sulle componenti della posizione per i due filtri. In basso, gli errori di stima sulle componenti della velocità.

Una criticità del filtro a particelle risiede nello specifico algoritmo impiegato per il resampling. Infatti, il valore `eps` che si sceglie per perturbare i campioni non dev'essere troppo grande perché ne risentirebbero significativamente le prestazioni. Tuttavia, non può essere neanche piccolo a piacere, altrimenti non si risolve il problema della degenerazione dei pesi e si può andare incontro a comportamenti errati del filtro. Ne consegue quindi che includere questa piccola perturbazione comporterà un *ripple*, un'increspatura persistente sulla traiettoria stimata, tanto più marcata maggiore è il valore assegnato ad `eps`.

In definitiva, non esiste una risposta univoca quando ci si chiede qual è il filtro migliore ma bisogna studiare attentamente la natura del problema e del sistema in esame per sceglierne la tipologia più adeguata ai mezzi a disposizione.

In merito all'algoritmo per lo smoothing, anch'esso è stato implementato con successo. Si è rivelato particolarmente efficace nel ridurre l'errore di stima per il filtro di Kalman esteso in *post-processing*, restituendo

un andamento della traiettoria stimata dolce e più aderente alla realtà.

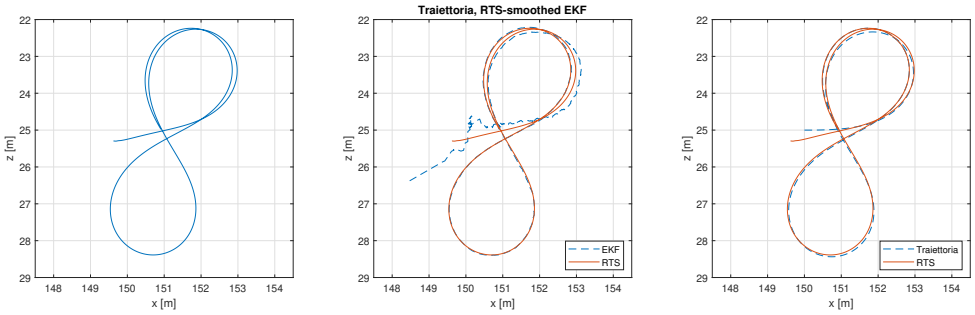

Figura 9: Da sinistra a destra, la traiettoria regolarizzata dallo smoothing, il confronto con la stima del filtro di Kalman esteso e con la traiettoria reale.