SVEUČILIŠTE U ZAGREBU

**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI SEMINAR

# IMPLEMENTACIJA OPTIMAL-REQUEST ALGORITMA

Ivan Vnučec

Zagreb, svibanj 2021.

*Posebna zahvala mentoru doc. dr. sc. Josipu Lončaru na pomoći i podršci pri istraživanju za ovaj rad.*

# SAŽETAK

U ovom radu bit će opisana implementacija Optimal-REQUEST algoritma. Optimal-REQUEST algoritam je rekurzivni algoritam za estimaciju orijentacije krutog tijela iz seta referentnih i mjerenih vektora temeljen na metodi najmanjih kvadrata. Algoritam će biti implementiran na ugradbenom računalnom sustavu s potencijalnom primjenom u estimaciji orijentacije nanosatelita.

# SADRŽAJ

# UVOD

## Wabba problem

Klasičan problem određivanja orijentacije tijela iz seta referentnih i mjerenih vektora naziva se Wabba problem. Ako nam je dan skup od $i$ vektora $\boldsymbol{b}$ i $i$ vektora $\boldsymbol{r}$, koji predstavljaju reprezentaciju mjerenih fizikalnih veličina u koordinatnom sustavu tijela (satelita) $B$ i referentnom (inercijalnom) koordinatnom sustavu $R$, problem estimacije orijentacije svodi se na pronalaženje rotacijske matrice $\boldsymbol{A}$ koja će minimizirati sljedeću relaciju:

$$\sum_{i=1}^{n} \|\boldsymbol{b}_i - \boldsymbol{A}\boldsymbol{r}_i\|^2.$$

Postoji mnoštvo rješenja za izračun $\boldsymbol{A}$ matrice, a jedno od njih je REQUEST algoritam.

## REQUEST algoritam

REQUEST algoritam je rekurzivni algoritam koji estimira rotacijsku matricu $\boldsymbol{A}$. REQUEST algoritam proizlazi iz QUEST algoritma koji nije rekurzivan, već orijentaciju procjenjuje iz skupine vektora izmjerenih u datom trenutku i zbog toga sva mjerenja izmjerena u prošlosti bivaju zanemarena [1].

Prednost REQUEST algoritma jest ta da uzima u obzir kutnu brzinu rotacije tijela pomoću koje propagira $\boldsymbol{K}$ matricu. Iz $\boldsymbol{K}$ matrice je moguće dobiti orijentaciju tijela u obliku kvaterniona. Ugađanje algoritma vrši se pomoću konstante $\rho$ koju se postavlja na najbolju procijenjenu vrijednost dobivenu simulacijama ili eksperimentalno. Ta varijabla nam daje mogućnost da pridodamo više težine mjerenju ako nam je nesigurnost jednadžbe stanja veća od nesigurnosti mjerenja i obrnuto.

Problem REQUEST algoritma je u tome što on ne uzima u obzir statističke greške u mjerenju vektora $\boldsymbol{r}$ i $\boldsymbol{b}$ i samim time nije optimalan u smislu najmanje kvadratne pogreške estimacije $\boldsymbol{K}$ matrice. Zbog tog nedostatka osmišljen je Optimal-REQUEST algoritam.

## Optimal-REQUEST algoritam

Optimal-REQUEST algoritam je nadogradnja QUEST i REQUEST algoritma. Optimal-REQUEST algoritam je rekurzivni algoritam gdje se parametar $\rho$ optimalno ugađa smanjujući srednju kvadratnu pogrešku estimacije. Algoritam uzima u obzir grešku koju uvodi neodređenost sustava i grešku koju uvodi mjerenje. Algoritam će optimalno estimirati $\boldsymbol{K}$ matricu minimizirajući srednju kvadratnu pogrešku između stvarne i estimirane $\boldsymbol{K}$ matrice [2].

# IMPLEMENTACIJA OPTIMAL-REQUEST ALGORITMA

Zbog jednostavnosti i programske podrške za matrične operacije, Optimal-REQUEST algoritam smo implementirali i testirali u MATLAB-u.

## Mjereni i referentni vektori

Implementacija algoritma temelji se na mjerenjima iz tri senzora: akcelerometra, magnetometra i žiroskopa, koje često nalazimo integrirane u jednom čipu zvanom inercijska mjerna jedinica (eng. *Inertial Measurement Unit*).

Zbog toga što su nam za jednoznačno određivanje orijentacije potrebna najmanje dva senzora, odlučili smo se za akcelerometar i magnetometar. Umjesto akcelerometra ili magnetometra mogli smo upotrijebiti senzor Sunca, senzor Zemljinog horizonta, pratitelj zvijezda (eng. *star tracker*) ili pak neke druge senzore no zbog jednostavnosti hardvera i ispitivanja ispravnosti algoritma odlučili smo se na akcelerometar i magnetometar. Jedan od razloga je taj što su akcelerometar i magnetometar jeftini senzori pakirani u isto integrirano kućište s kojima je lako rukovat. Uz to, akceleracija (gravitacijsko ubrzanje) i Zemljino magnetsko polje u referentnom sustavu su nam poznati i ne zahtijevaju dodatni mjerni postav, za razliku od primjerice senzora Sunca, kojem je potreban umjetno generirani izvor svjetla, matematički model Zemljine orbite oko Sunca te matematički model orbite satelita oko Zemlje.

Matrica $\boldsymbol{r}$ referentnih vektora $\boldsymbol{r_i}$ se sastoji od stupaca poznatih vektora akceleracije i magnetskog polja u referentnom (inercijalnom) koordinatnom sustavu. Koordinatne osi referentnog koordinatnog sustava nepomične su u odnosu na Zemlji, a definirane su na način da os apscisa gleda u smjeru geodetskog sjevera (engl. *North*, N), os ordinata u smjeru istoka (engl. *East*, E), a aplikata prema središtu Zemlje (engl. *Down*, D). Skraćeno ovako definirani koordinatni sustav nazivamo NED sustavom.

Vrijednost referentnog vektora akceleracije u NED sustavu iznosi $[0 \quad 0 \quad -9.81]^T \frac{m}{s^2}$. Vrijednost referentnog vektora magnetskog polja može se izračunati iz modela geomagnetskog polja [3] za zadanu geografsku širinu, dužinu i visinu. Vektor Zemljinog magnetskog polja na području Zagreba definiran u NED referentnom koordinatnom sustavu ugrubo iznosi $[22165.4 \quad 1743 \quad 42786.9]^T$ nT.

Koordinatni sustav satelita rotiran je u odnosu na referentni inercijalni koordinatni sustav zbog čega se vektori akceleracije i magnetskog polja u koordinatnom sustavu satelita razlikuju u odnosu na referentne vektore. Stupčaste vektore akceleracije i magnetskog polja u koordinatnom sustavu satelita mjerimo akcelerometrom i magnetometrom i zatim ih pohranjujemo u matricu $\boldsymbol{b}$.

Informacija o orijentaciji koordinatnog sustava satelita u odnosu na inercijalni koordinatni sustav krije se u razlici između poznatih referentnih vektora u inercijalnom koordinatnom sustavu i mjerenih vektora u koordinatnom sustavu satelita.

## Koraci algoritma

U daljnjem tekstu usporedno ćemo navesti korake Optimal-REQUEST algoritma u obliku algebarskih jednadžbi i u obliku MATLAB programskog koda.

Zbog preglednosti smo izostavili dijelove koda koji služe za inicijalizaciju MATLAB sučelja, varijabli potrebnih za simulaciju orijentacije, funkcija za iscrtavanje grafova i slično. Za potpuni MATLAB kod čitatelj se upućuje na dodatak A.

| Inicijalizacija i prvo mjerenje | |
|---|---|
| | `k = 1;`<br>`r0 = [mag_ref_meas(:,k), acc_ref_meas(:,k)];`<br>`b0 = [mag_bdy_meas(:,k), acc_bdy_meas(:,k)];` |
| Postavi jednake težine na sva mjerenja. Mora vrijediti: $\sum_{i=1}^{n} a_i = 1$ | `[~, ncols] = size(b0);`<br>`a0 = ones(1, ncols) ./ ncols;` |
| $\delta m_{k+j} = \sum_{i=k+1}^{k+j} a_i$ | `dm0 = sum(a0);` |
| Vidi dodatak A | `dK0 = calculate_dK(r0, b0, a0);` |
| Vidi dodatak A | `R0 = calculate_R(r0, b0, Mu_noise_std^2);` |
| $K_{0/0} = \delta K_0$ | `K = dK0;` |
| $P_{0/0} = \mathcal{R}_0$ | `P = R0;` |
| $m_0 = \delta m_0$ | `mk = dm0;` |
| Rekurzivno za svaki k | |
| | `for k = 2 : num_of_iter` |
| Dobavi kutnu brzinu | `w = gyr_bdy_meas(:,k);` |
| $[w\times] = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix}$ | `wx = [0, -w(3), w(2); w(3), 0, -w(1); -w(2), w(1), 0];` |
| $\Omega_k \triangleq \frac{1}{2}\begin{bmatrix} -[\omega_k\times] & \omega_k \\ -\omega_k^T & 0 \end{bmatrix}$ | `Omega = 1.0 / 2 * [-wx, w; -w', 0];` |
| $\Phi_k = \exp(\Omega_k \Delta t)$ | `Phi = expm(Omega * dT);` |
| Vidi dodatak A | `[B, ~, z, Sigma] = get_util_matrices(K);` |
| Vidi dodatak A | `Q = calculate_Q(B, z, Sigma, Eta_noise_std^2, dT);` |
| $K_{k+1/k} = \Phi_k K_{k/k} \Phi_k^T$ | `K = Phi * K * Phi';` |
| $P_{k+1/k} = \Phi_k P_{k/k} \Phi_k^T + \mathcal{Q}_k$ | `P = Phi * P * Phi' + Q;` |
| Dobavi vektore opservacije | `r = [mag_ref_meas(:,k), acc_ref_meas(:,k)];`<br>`b = [mag_bdy_meas(:,k), acc_bdy_meas(:,k)];` |
| Postavi jednake težine na sva mjerenja. Mora vrijediti: $\sum_{i=1}^{n} a_i = 1$ | `[~, ncols] = size(b);`<br>`a = ones(1, ncols) ./ ncols;` |

| | |
|---|---|
| $\delta m_{k+j} = \sum_{i=k+1}^{k+j} a_i$ | `dm = sum(a);` |
| Vidi dodatak A | `R = calculate_R(r, b, Mu_noise_std^2);` |
| $\rho^*_{k+1} = \frac{m_k^2 tr(P_{k+1/k})}{m_k^2 tr(P_{k+1/k}) + \delta m_{k+1}^2 tr(\mathcal{R}_{k+1})}$ | `Rho = (mk^2 * trace(P)) / (mk^2 * trace(P) + dm^2 * trace(R));` |
| $m_{k+1} = \left(1 - \rho^*_{k+1}\right) m_k + \rho^*_{k+1} \delta m_{k+1}$ | `m = (1.0 - Rho) * mk + Rho * dm;` |
| Vidi dodatak A | `dK = calculate_dK(r, b, a);` |
| $K_{k+1/k+1} = \left(1 - \rho^*_{k+1}\right) \frac{m_k}{m_{k+1}} K_{k+1/k} + \rho^*_{k+1} \frac{\delta m_{k+1}}{m_{k+1}} \delta K_{k+1}$ | `K = (1.0 - Rho) * mk / m * K + Rho * dm / m * dK;` |
| $P_{k+1/k+1} = \left[\left(1 - \rho^*_{k+1}\right) \frac{m_k}{m_{k+1}}\right]^2 P_{k+1/k} + \left(\rho^*_{k+1} \frac{\delta m_{k+1}}{m_{k+1}}\right)^2 \mathcal{R}_{k+1}$ | `P = ((1.0 - Rho) * mk / m)^2 * P + (Rho * dm / m)^2 * R;` |
| $m_{k-1} = m_k$ | `mk = m;` |
| | `End` |

## Provjera ispravnosti algoritma

Radi provjere ispravnosti algoritma, programski smo generirali vektor kutne brzine, referentne i mjerene vektore tako što smo matematički opisali rotaciju tijela te smo zatim te podatke spremili u CSV datoteku. Datoteka osim spomenutih vrijednosti sadrži i stvarni kvaternion koji nam predstavlja orijentaciju tijela za svaki trenutak mjerenja. Estimirani kvaternion ćemo kasnije usporediti sa stvarnim kvaternionom iz CSV datoteke te tako ocijeniti ispravnost algoritma.

Nakon što smo učitali generirane mjerene vektore svakoj komponenti dodajemo normalno distribuiran šum s očekivanjem nula i varijancom „*Eta_noise_std^2*“ te ih normiramo na jediničnu duljinu. Referentnim vektorima nismo dodali šum već smo ih samo normirali jer su nam njihove vrijednosti u inercijalnom koordinatnom sustavu poznate. Dodali smo šum i komponentama vektora kutne brzine s očekivanom vrijednošću nula i varijancom jednakom „*Mu_noise_std^2*“. Vektor kutne brzine ne smijemo normirati jer je promjena orijentacije sadržana u duljini vektora.

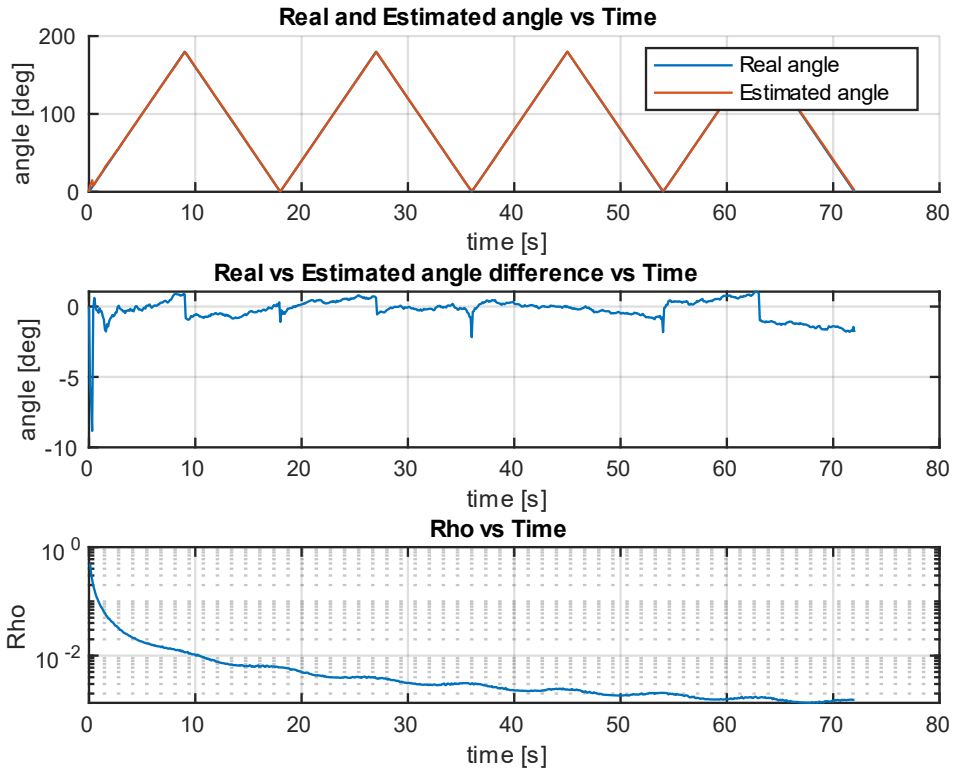
Potom iterativno prolazimo kroz sva mjerenja računajući izraze Optimal-REQUEST algoritma spremajući na kraju estimirani kvaternion. Nakon što se postupak estimacije orijentacije dovrši, grafički prikazujemo ovisnost razlike estimiranog i stvarnog kvaterniona. Primjeri i opisi prikaza dani su ispod.

Prvi graf odozgor na slici 1 prikazuje kut rotacije stvarnog i estimiranog kvaterniona. Valja uočiti da su stvarni (plavo) i estimirani kut (narančasto) vrlo sličnih vrijednosti, zbog čega je plavi graf teže uočljiv. Razlog pilastog oblika kuta orijentacije leži u načinu izračuna kuta rotacije iz kvaterniona. Cilj nam je da se oba kuta u potpunosti poklapaju. Nagib pilastog oblika predstavlja kutnu brzinu.

Drugi graf na slici 1 prikazuje razliku stvarnog i estimiranog kuta rotacije. Primijetimo da je razlika manja od jednog stupnja za kvalitativno određene razine šuma (vidi dodatak A) nakon svega nekoliko sekundi estimacije. Željeli bismo kad bi se taj graf kretao oko nule.

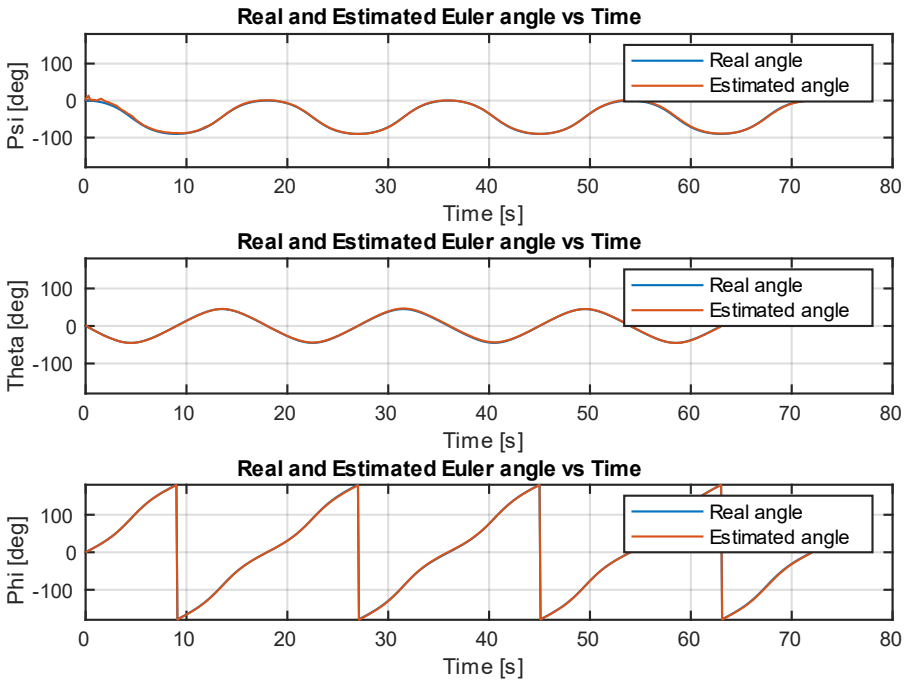
Na zadnjem grafu nalazi se tzv. *Rho* (ρ) parametar koji se ponaša kao Kalmanovo pojačanje: zbog velike nesigurnosti jednadžbe stanja na samom početku iterativnog postupka, *Rho* parametar je relativno velik te će veću važnost pridodati mjerenjima. Vidimo dalje da s vremenom *Rho* parametar eksponencijalno

pada te veću važnost daje jednadžbi stanja jer je sad već poprilično dobro estimirao koliko iznose greške mjerenja.




Slika 1: Prikaz izlaznih varijabli estimatora.

Na drugoj slici prikazali smo Eulerove kutove dobivene iz stvarnog i estimiranog kvaterniona. Uočiti kao i na slici 2 da su grafovi gotovo isti, zbog čega je plavi graf skriven iza narančastog.




Slika 2: Prikaz Eulerovih kutova stvarne i estimirane orijentacije.

# ZAKLJUČAK

U ovome radu pokazali smo problematiku određivanja orijentacije satelita pomoću Optimal-REQUEST algoritma i dali smo implementaciju algoritma u MATLAB okruženju. Daljnji napori bit će usmjereni prema implementaciji algoritma na ugradbenom računalnom sustavu.

# LITERATURA


[1] I. Y. Bar-Itzhack, »REQUEST: A Recursive QUEST Algorithmfor Sequential Attitude Determination,« *Journal of Guidance, Control, and Dynamics,* pp. 1034-1038, 1996.

[2] I. Y. B.-I. Y. O. D. Choukroun, »Optimal-REQUEST Algorithm for Attitude Determination,« *Journal of Guidance, Control. and Dynamics,* pp. 418-425, 2014.

[3] NOAA, »The World Magnetic Model,« 13 5 2021. [Mrežno]. Available: https://www.ngdc.noaa.gov/geomag/WMM/calculators.shtml.

[4] I. Vnučec, »Optimal-REQUEST,« 13 5 2021. [Mrežno]. Available: https://github.com/IvanVnucec/Optimal-REQUEST.

# DODATAK A

U ovom dodatku nalazi se MATLAB programski kod koji generira referentne i izmjerene vektore te pomoću njih estimira orijentaciju tijela.

Programski kod dan ovdje je nešto drugačiji nego kod o kojem smo govorili u dijelu provjere ispravnosti algoritma utoliko što ovaj kod ne simulira rotaciju tijela već je kutna brzina tijela jednaka nuli. Naposlijetku vektoru kutne brzine dodajemo šum kao što se može vidjeti ispod.

Programski kod je nešto kraći jer ga je takvog lakše prezentirati čitatelju. Detaljniji implementacija algoritma uključujući popratnu CSV datoteku koja sadrži sve potrebne referentne i mjerene vektore te točan orijentacijski kvaternion, nalaze se u GitHub repozitoriju [4].

**main.m**

```
% ========================== info ============================
% Script for testing the Optimal-REQUEST algorithm.
% Works in MATLAB or GNU Octave.
%
% Author: Ivan Vnucec, 2021
% License: MIT

% ========================== notes ===========================
% Almost all of the algorithm equations have references to some
% paper. The reference equations are written in the form for ex.
% Ref A eq. 25 where letter 'A' denotes reference to the paper
% and the number '25' the equation number it that paper. The
% references are listed under Reference section below.
%
% Indexes with k+1 are written without indexes and indexes with
% k are written with _k. For example dm_k+1 is written as dm and
% dm_k is written as dmk.

% ======================== references ========================
% Ref A:
%   REQUEST: A Recursive QUEST Algorithmfor Sequential Attitude
Determination
%   Itzhack Y. Bar-Itzhack
%
% Ref B:
%   Optimal-REQUEST Algorithm for Attitude Determination,
%   D. Choukroun,I. Y. Bar-Itzhack, and Y. Oshman
%
% Ref. C:
%   Appendix B, Novel Methods for Attitude Determination Using Vector
Observations,
%   Daniel Choukroun,
%
% Ref. D:
%   Appendixes, ttitude Determination Using Vector Observations andthe
Singular Value Decomposition
%   Markley, F. L.,

% ========================== START ===========================
% for debug
clear all;
close all;
rng('default');

% ========================== constants =======================
```

```
dT = 0.1;               % senzor refresh time, in seconds
simulation_time = 500;  % in seconds

num_of_iter = simulation_time / dT;
t = linspace(0, simulation_time, num_of_iter);

% ======================== measurements ========================
% === white Gauss zero mean noise ===
gyr_bdy_meas_noise_std = 0.01;       % rad/s
acc_bdy_meas_noise_std = 0.01;       % m/s^2
mag_bdy_meas_noise_std = 10.0;       % nT

% TODO: See how we can calculate Mu meas nose (see eq. 36)
Mu_noise_std = acc_bdy_meas_noise_std + mag_bdy_meas_noise_std; % for R
computation
Eta_noise_std = gyr_bdy_meas_noise_std;                         % for Q
computation

% === w/o noise ===
% reference
acc_ref_meas = zeros(3, num_of_iter) + [0 0 -9.81]';            % m/s
mag_ref_meas = zeros(3, num_of_iter) + [22165.4 1743 42786.9]'; % nT

% rotate reference vectors to create body vectors
% rotate about vector 'n' by an 'angle' in radians
n = [1 1 1]';
angle = 2*pi/3;

% body
gyr_bdy_meas = zeros(3, num_of_iter);               % rad/s
acc_bdy_meas = rodrigues(acc_ref_meas, n, angle);   % rotated % m/s
mag_bdy_meas = rodrigues(mag_ref_meas, n, angle);   % rotated % nT

% == add gaussian noise to body measurements ===
gyr_bdy_meas = gyr_bdy_meas + randn(size(gyr_bdy_meas)) *
gyr_bdy_meas_noise_std;
acc_bdy_meas = acc_bdy_meas + randn(size(acc_bdy_meas)) *
acc_bdy_meas_noise_std;
mag_bdy_meas = mag_bdy_meas + randn(size(mag_bdy_meas)) *
mag_bdy_meas_noise_std;

% === normalize measurement vectors ===
% reference
acc_ref_meas = acc_ref_meas ./ vecnorm(acc_ref_meas);
mag_ref_meas = mag_ref_meas ./ vecnorm(mag_ref_meas);
% body
acc_bdy_meas = acc_bdy_meas ./ vecnorm(acc_bdy_meas);
mag_bdy_meas = mag_bdy_meas ./ vecnorm(mag_bdy_meas);

% ======================== algorithm output ====================
K_out = zeros(4, 4, num_of_iter);
q_out = zeros(4, 1, num_of_iter);
angle_out = zeros(1, num_of_iter);
Rho_out = zeros(1, num_of_iter);

% =================== initialization k=0 ======================
k = 1; % k=1 because of MATLAB counting from 1 and not from 0

% prepare first measurement and weights
r0 = [mag_ref_meas(:,k), acc_ref_meas(:,k)];
b0 = [mag_bdy_meas(:,k), acc_bdy_meas(:,k)];
```

```
[~, ncols] = size(b0);
a0 = ones(1, ncols) ./ ncols; % equal weights

dm0 = sum(a0); % Ref. A eq. 11a
dK0 = calculate_dK(r0, b0, a0);
R0 = calculate_R(r0, b0, Mu_noise_std^2);

K = dK0;    % Ref. B eq. 65
P = R0;     % Ref. B eq. 66
mk = dm0;   % Ref. B eq. 67, mk = m_k

% ======================== algorithm ============================
for k = 2 : num_of_iter
    % ================== time update ==================
    % get angular velocity measurement
    w = gyr_bdy_meas(:,k);

    % Ref. B eq. 4
    wx = [0, -w(3), w(2); w(3), 0, -w(1); -w(2), w(1), 0];

    % Ref. B eq. 10
    Omega = 1.0 / 2 * [-wx, w; -w', 0];

    % Ref. B eq. 9
    Phi = expm(Omega * dT); % eq. 9

    [B, ~, z, Sigma] = get_util_matrices(K);
    Q = calculate_Q(B, z, Sigma, Eta_noise_std^2, dT);

    % Ref. B eq. 11
    K = Phi * K * Phi';

    % Ref. B eq. 69
    P = Phi * P * Phi' + Q;

    % ================ measurement update ===============
    % referent vector measurements
    r = [mag_ref_meas(:,k), acc_ref_meas(:,k)];
    % body vector measurements
    b = [mag_bdy_meas(:,k), acc_bdy_meas(:,k)];
    % calc. meas. weights
    [~, ncols] = size(b);
    a = ones(1, ncols) ./ ncols; % equal weights

    dm = sum(a);

    R = calculate_R(r, b, Mu_noise_std^2);

    % Ref. B eq. 70
    Rho = (mk^2 * trace(P)) / (mk^2 * trace(P) + dm^2 * trace(R));

    % Ref. B eq. 71
    m = (1.0 - Rho) * mk + Rho * dm;

    dK = calculate_dK(r, b, a);

    % Ref. B eq. 72
    K = (1.0 - Rho) * mk / m * K + Rho * dm / m * dK;

    % Ref. B eq. 73
    P = ((1.0 - Rho) * mk / m)^2 * P + (Rho * dm / m)^2 * R;
```

```
    % for the next iteration m_k = m_k+1
    mk = m;

    % store calculated K and q for debug
    K_out(:,:,k) = K;
    q = get_quat_from_K(K);
    q_out(:,:,k) = q;
    angle_out(:,k) = 2.0 * acos(abs(q(1)));
    Rho_out(k) = Rho;
end

% plot angle difference between real and estimated angle
figure;
plot(t, rad2deg(angle - angle_out));
title('Real vs Estimated angle difference vs Time');
xlabel('time [s]');
ylabel('angle [deg]');
grid on;

% plot the optimal filter gain
figure;
semilogy(t, Rho_out);
title('Rho vs Time');
xlabel('time [s]');
ylabel('Rho');
grid on;
```

**calculate_Q.m**

```
function [Q] = calculate_Q(B, z, Sigma, var, dT)
%CALCULATE_Q Compute matrice 'Q'.
%   Compute matrice 'Q' with submatrices 'B', 'z', 'Sigma', noise variance
%   variance 'var' and time difference 'dT'. 'B', 'z' and 'Sigma'
%   submatrices can be calculated via the 'get_util_matrices' function.
%
%   Ref. C eq. B.2.1.4a - B.2.1.4d, (for references list see main.m file
under reference
%   comment section).

M = B * (B - Sigma * eye(3));
yx = M' - M;
y = [yx(3,2); yx(1,3); yx(2,1)];

Q11 = var * ((z' * z + Sigma^2 - trace(B * B')) * eye(3) ...
    + 2 * (B' * B - B^2 - B'^2));
Q12 = -var * (y + B' * z);
Q21 = Q12';
Q22 = var * (trace(B * B') + Sigma^2 + z' * z);

Q = [Q11, Q12; Q21, Q22] * dT^2;
end
```

**calculate_R.m**

```
function [R] = calculate_R(r, b, var)
%CALCULATE_R Compute matrice 'R'.
%   Compute matrice 'R' with reference matrice 'r', body measurements
%   matrice 'b' and measurement variance 'var'.
%
```

```
%   Ref. C eq. B.2.2.4a - B.2.2.4d, (for references list see main.m file
under reference
%   comment section).

[~, ncols] = size(r);

R11 = zeros(3, 3);
for i = 1 : ncols
    ri = r(:, i);
    bi = b(:, i);

    rx = [0, -ri(3), ri(2); ri(3), 0, -ri(1); -ri(2), ri(1), 0];

    R11 = R11 + (3.0 - (ri' * bi)^2) * eye(3) + (bi' * ri) * (bi * ri' + ri
* bi') ...
        + rx * (bi * bi') * rx';
end

R11 = var / ncols * R11;
R12 = zeros(3, 1);
R21 = R12';
R22 = 2.0 * var / ncols;

R = [R11, R12; R21, R22];
end
```

**calculate_dK.m**

```
function [dK] = calculate_dK(r, b, a)
%CALCULATE_DK Calculate 'dK'.
%   Calculate 'dK' incremental matrice with reference matrice 'r', body
%   matrice 'b' and weights assigned to each column vector in body matrice
%   'a'.
%
%   Ref. A eq. 11b - 11f, (for references list see main.m file under
reference
%   comment section).

[~, ncols] = size(r);

% eqs. 3.7
dB = zeros(3, 3);
dz = zeros(3, 1);
dSigma = 0.0;
for i = 1:ncols
    ai = a(i);
    bi = b(:,i);
    ri = r(:,i);

    dB = dB + ai .* bi * ri';
    dz = dz + ai .* cross(bi, ri);
    dSigma = dSigma + ai * bi' * ri;
end
dS = dB + dB';

dK = [dS - dSigma * eye(3), dz; dz', dSigma];

end
```

**get_quat_from_K.m**

```
function [q] = get_quat_from_K(K)
%GET_QUAT_FROM_K Function returns the eigenvector of matrice 'K' with the
%   largest eigenvalue.
%   The returned vector is quaternion rotation.
%
%   Ref. B states:
%   The optimal quaternionË†qk+1/k+1is the eigenvector ofKk+1/k+1,which
belongs to its maximal eigenvalue.

% get eigenvector
[V, D] = eig(K);

% sort eigenvectors by eigenvalues
[~, ind] = sort(diag(D));
Vs = V(:,ind);

% pick the one with the largest eigenvalue which is at the end
q = Vs(:,end);

% Ref. B eq. 3 (text under), we are using different quaternion
representation
q = [q(4); q(1:3)];

end
```

**get_util_matrices.m**

```
function [B, S, z, Sigma] = get_util_matrices(K)
%GET_UTIL_MATRICES Returns submatrices 'B', 'S', 'z' and 'Sigma' of 'K'
%   matrice.
%   The submatrices are used for the calculation of Q matrice.
%
%   Ref. B eq. 6 and 7, (for references list see main.m file under
reference
%   comment section).

Sigma = K(4,4);
S = K(1:3,1:3) + Sigma * eye(3);
B = 0.5 * S;
z = K(1:3,4);

end
```

**rodrigues.m**

```
function [vrot] = rodrigues(v, k, Theta)
%RODRIGUES Rotate column vectors 'v' around column vector 'k' by 'Theta'
%degrees in radians.
%   With the so called Rodrigues formula this function rotates vectors.
%   Vector 'k' doesn't need to be normalized because it is being normalized
%   in function.

k = k / norm(k);

[~, ncols] = size(v);

vrot = zeros(size(v));
```

```
for i = 1 : ncols
    vrot(:,i) = v(:,i) * cos(Theta) + cross(k, v(:,i)) * sin(Theta) + ...
        k * dot(k, v(:,i)) * (1.0 - cos(Theta));
end
end
```