ПРИЛОЖЕНИЕ Е

Общество с ограниченной ответственностью «Статус Консалт»
(ООО «Статус Консалт»)

УТВЕРЖДАЮ

Генеральный директор
ООО «Статус Консалт»

Д.С. Сушко

Дата:

М.П.

ПРОГРАММНАЯ БИБЛИОТЕКА НАВИГАЦИОННЫХ АЛГОРИТМОВ «NavigationLib»

ТЕКСТ ПРОГРАММЫ

ЛИСТ УТВЕРЖДЕНИЯ

93241908.468333.00001-01 02-ЛУ

Листов 2

Подпись и дата | Инв. № | Взам. | Подпись и дата | Инв. №

Наименование организации	Должность исполнителя	Ф.И.О.	Подпись	Дата

Подпись и дата
Инв. №
Взам.
Подпись и дата
Инв. №

УТВЕРЖДЕН

93241908.468333.00001-01 02-ЛУ

ПРОГРАММНАЯ БИБЛИОТЕКА НАВИГАЦИОННЫХ АЛГОРИТМОВ «NavigationLib»

ТЕКСТ ПРОГРАММЫ

93241908.468333.00001-01 02

Листов 19

Подпись и дата | Инв. № | Взам. | Подпись и дата | Инв. №

Модуль гирокомпассирования

Файл **compass.py**.

```
import numpy as np
from scipy.optimize import curve_fit


def fit_f(x, p1, p2, p3):
    # function for approximation of X-gyro output modulated by Earth rotation rate
    # OmegaX = Omega0 + ScaleFactor * EarthRate * cos(yaw + initial_azimuth)
    # here: p1 = Omega0 - gyro bias
    #       p2 = ScaleFactor - gyro scale factor
    #       p3 - initial azimuth
    #       x - current yaw angle
    return p1 + p2 * np.cos((p3 + x) * np.pi / 180)


class Caruseling:
    # class for azimuth estimation using method of X-gyro modulation with Earth rotation rate and
reconstruction of
    # initial azimuth angle
    # Each time gyro should be oriented at new yaw angle, stay still in this position (with
measuring and smoothing
    # X-gyro output for about 1-2min. Then feed new pair od data (yaw, gyro_out) to class
Caruseling and get
    # new initial azimuth estimation. Estimation is provided using the Least Square Method.
    def __init__(self):
        self.yaw = []
        self.gyro_out = []
        self.azimuth = None

    def __call__(self, yaw_in_nav_frame, gyro_out):
        # inputs - next pair of yaw and X-gyro measurement
        # output - estimated initial azimuth of Navigation Frame
        self.yaw.append(yaw_in_nav_frame)
        self.gyro_out.append(gyro_out)
        if len(self.yaw) > 3:
            x = np.array(self.yaw)
            y = np.array(self.gyro_out)
            popt = curve_fit(fit_f, x, y, p0=(0.0, 10.2, 0))
            azimuth = popt[2]
            if popt[1] < 0:
                azimuth += 180
        return self.azimuth


class Maytagging:
    # class for azimuth estimation using maytagging method:
    # fast estimation can be done orienting X-gyro 4 perpendicular directions (four_point method)
    # at each point (direction) gyro should be stable and its output should be smoothed during 1-
2min
    # Measurements in 4-points method are:
```

```
    # Omega0 = Omega0 + ScaleFactor * EarthRate * cos(azimuth)
    # Omega90 = Omega0 - ScaleFactor * EarthRate * sin(azimuth)
    # Omega180 = Omega0 - ScaleFactor * EarthRate * cos(azimuth)
    # Omega270 = Omega0 + ScaleFactor * EarthRate * sin(azimuth)
    # so:
    #   Omega0 - Omega180 = 2 * EarthRate * cos(azimuth)
    #   Omega270 - Omega90 = 2 * EarthRate * sin(azimuth)
    # and: tg(azimuth) = (Omega270 - Omega90) / (Omega0 - Omega180)
    # two_point method is less accurate due to lack of precise information of EarthRate in current
location
    # It uses the fact that Omega0 - Omega180 = 2 * EarthRate * sin(azimuth)
    # for best SNR Omega0 and Omega180 should be measured in approx East (Omega270 in 4-
points method)
    # and West (Omega90 in 4-points method) directions.

    def __init__(self):
        self.azimuth = 0

    def two_point(self, gyro_out_at_yaw_0deg, gyro_out_at_yaw_180deg, earth_rate_mod):
        # two X-gyro measurements - in aprrox East direction (0deg) and approx West direction
(180deg)
        # output - estimated initial azimuth of Navigation Frame
        azimuth = np.arsin((gyro_out_at_yaw_0deg - gyro_out_at_yaw_180deg) / 2 /
earth_rate_mod)
        self.azimuth = (self.azimuth + azimuth) / 2

    def four_point(self, gyro_out_at_yaw_0_90_180_270):
        # input - list of 4 measurements of X-gyro output at 4 perpendicular directions
        # output - estimated initial azimuth of Navigation Frame
        azimuth = np.arctan2(-gyro_out_at_yaw_0_90_180_270[1] -
gyro_out_at_yaw_0_90_180_270[3],
                             gyro_out_at_yaw_0_90_180_270[0] - gyro_out_at_yaw_0_90_180_270[2])
        self.azimuth = (self.azimuth + azimuth) / 2
```

Модуль курсовертикали

Файл **AHRS.m**.

```
function [Cbn, P, bw] = AHRS(Cbn, P, bw, dwb, fb, mb, fn, mn, dt)
% [Cbn, P, bw] = ahrs_dcm(Cbn, P, bw, dwb, fb, mb, fn, mn, dt)
% Реализует алгоритм курсовертикали на базе использования измерений от
% датчиков угловой скорости, ускорения и магнитного поля
%
%   Входные аргументы:
%   Cbn - Матрица направляющих косинусов
%   P   - Матрица ковариации фильтра Калмана
%   bw  - Вектор оценок сдвигов нулей ДУС
%   dwb - Вектор измерений ДУС
%   fb  - Вектор измерений акселерометра
%   mb  - Вектор измерений магнитного поля
%   fn  - Вектор ускорения свободного падения в навигационной СК
%   mn  - Вектор магнитного поля в навигационной СК
%   dt  - Шаг интегрирования
%
```

```
%   Выходные аргументы:
%   Cbn  - Матрица направляющих косинусов
%   P    - Матрица ковариации фильтра Калмана
%   bw   - Вектор оценок сдвигов нулей ДУС


%% Коррекция показаний ДУС
dwb = dwb - bw * dt;

%% Обновление матрицы направляющих косинусов (интегрирование ориентации)
rot_norm = norm(dwb);
sr_a = 1 - (rot_norm ^ 2 / 6) + (rot_norm ^ 4 / 120);
sr_b = (1 / 2) - (rot_norm ^ 2 / 24) + (rot_norm ^ 4 / 720);
Cbb = eye(3) + sr_a * skew(dwb) + sr_b * skew(dwb) * skew(dwb);
Cbn = Cbn * Cbb;

%% Фильтр Калмана - предсказание
% Матрица F динамики системы и матрица Q ковариации шумов измерений

% Матрица динамики системы в непрерывном времени
A = zeros(6, 6);
A(1:3, 4:6) = -Cbn;

% Матрица динамики системы в дискретном времени
F = eye(6) + A * dt + A * A * dt * dt / 2;

% Матрица влияния шумов системы
G = zeros(6, 6);
G(1:3, 1:3) = eye(3);
G(4:6, 4:6) = eye(3);

% Шумы ошибок ДУС
ng = 1e-4;

% Шумы сдвигов нолей ДУС
ngb = 1e-6;

% Матрица ковариации шумов системы
Qn = diag([ng, ng, ng, ngb, ngb, ngb]);

% Дискретизация с интегрированием методом трапеций
Q = 1 / 2 * (F * G * Qn * G' + G * Qn * G' * F') * dt;

% Обновленное значение марицы ковариации фильтра Калмана
P = F * P * F' + Q;

%% Измерения
% Оценка вектора ускорения свободного падения в навигационной СК
fn_hat = Cbn * fb;

% Оценка вектора магнитного поля в навигационной СК
mn_hat = Cbn * mb;
```

```
% Вектор измерений
v = zeros(6, 1);
v(1:3, 1) = fn_hat - fn;
v(4:6, 1) = mn_hat - mn;

% Матрица измерений
H = zeros(6, 6);
H(1:3, 1:3) = skew(fn_hat);
H(4:6, 1:3) = skew(mn_hat);

% Матрица ковариации шумов измерений
R = diag([1e-1, 1e-1, 1e-1, 1e-1, 1e-1, 1e-1]);

%% Фильтр Калмана - коррекция
I = eye(6);
S = H * P * H' + R;
K = (P * H') / S;
P = (I - K * H) * P * (I - K * H)' + K * R * K';
x = K * v;

%% Коррекция матрицы направляющих косинусов
E = eye(3) + skew(x(1:3, 1));
Cbn = E * Cbn;

% Нормализация матрицы направляющих косинусов
Cbn = dcmnormalize(Cbn);

%% Обновление оценки сдвига нулей ДУС
bw = bw + x(4:6, 1);

end
```

Модуль навигации с коррекцией от СНС

Файл **navigation_system_GPS.m**

```
function [rn, Cbn, Vb_odometer, Wb_odometer, P, dw, dlr, dll] = ...
   navigation_system_GPS(rn, Cbn, P, dt, dThe, fir, fil, lr, ll, d, ...
   pos_gps, vel_gps, lb, gps_update, dw, dlr, dll)
%   Алгоритм системы навигации колесного робота с коррекцией от приемника
%   GPS
%
%   Входные аргументы:
%   rn - оценка координат робота
%   Cbn - оценка матрицы направляющих косинусов ориентации робота
%   P - матрица ковариации фильтра Калмана
%   dt - шаг интегрирования
%   dThe - измерение приращения угла курса гироскопом за шаг интегрирования
%   fir - скорость вращения правого колеса
%   fil - скорость вращения левого колеса
%   lr - длина окружности правого колеса
%   ll - длина окружности левого колеса
%   d - длина оси колесной пары
```

```
%   pos_gps - измерение координат приемником GPS, спроецированное в
%   локальную плоскую СК
%   vel_gps - измерение скоростей приемником GPS, спроецированное в
%   локальную плоскую СК
%   lb - вектор плеча установки приемника GPS относительно центра колесной
%   пары
%   gps_update - индикатор доступности измерения от приемника GPS
%   dw - оценка сдвига нуля курсового гироскопа
%   dlr - оценка ошибки длины окружности правого колеса
%   dll - оценка ошибки длины окружности левого колеса
%
%   Выходные аргументы:
%   rn - оценка координат робота
%   Cbn - оценка матрицы направляющих косинусов ориентации робота
%   Vb_odometer - оценки скорости продольного перемещения одометром
%   Wb_odometer - оценка скорости курсовго разворота одометром
%   P - матрица ковариации фильтра Калмана
%   dw - оценка сдвига нуля курсового гироскопа
%   dlr - оценка ошибки длины окружности правого колеса
%   dll - оценка ошибки длины окружности левого колеса

%% Одометр
% компенсация ошибок одометра - длины правого и левого колес и длины оси
% колесной пары
lr = lr - dlr;
ll = ll - dll;

% Скорости продольного перемещения и курсового разворота
Vb_odometer = 1/2 * (fir * lr + fil * ll);
Wb_odometer = 1/d * (fir * lr - fil * ll);

% Компенсация сдвига нуля гироскопа
dthet_z = dThe - dw * dt;

% Обновление матрицы направляющих косинусов (интегрирование ориентации)
rot_norm = norm(dthet_z);
sr_a = 1 - (rot_norm ^ 2 / 6) + (rot_norm ^ 4 / 120);
sr_b = (1 / 2) - (rot_norm ^ 2 / 24) + (rot_norm ^ 4 / 720);
Cbb = [
   1 - sr_b * dthet_z ^ 2, -sr_a * dthet_z, 0;
   sr_a * dthet_z, 1 - sr_b * dthet_z ^ 2, 0;
   0, 0, 1
   ];
Cbn=Cbn * Cbb;

% Обновление координат
rn = rn + Cbn * [Vb_odometer; 0; 0] * dt;

%% Фильтр Калмана - предсказание

% Матрица динамики системы в непрерывном времени
An = zeros(3);
```

```
An(1,3) =  Cbn(2,1) * Vb_odometer;
An(2,3) = -Cbn(1,1) * Vb_odometer;

% Матрица динамики системы в дискретном времени
Fnd=eye(3) + An * dt + An * An * dt * dt / 2;

% Матрица влияния шумов системы
Nn = zeros(3, 3);
Nn(1, 1) =  Cbn(1, 1) * fir / 2;
Nn(1, 2) =  Cbn(1, 1) * fil / 2;
Nn(2, 1) =  Cbn(2, 1) * fir / 2;
Nn(2, 2) =  Cbn(2, 1) * fil / 2;
Nn(3, 3) = -1;

% Матрица ковариации шумов системы
pos_noise = 1e-6;
att_noise = 1e-8;
R = diag([pos_noise, pos_noise, att_noise]);
Qn = Nn * R * Nn';

% Дискретизация с интегрированием методом трапеций
Qnd = dt / 2 * (Fnd * Qn + Qn * Fnd');

% Матрица динамики системы и матрица ковариации шумов измерений
F = zeros(6);
Q = zeros(6);
F(1:3, 1:3) = Fnd;
F(4:6, 4:6) = eye(3);
F(1:3, 4:6) = Nn * dt;

Q(1:3, 1:3) = Qnd;
nlr = 1e-12;
nll = 1e-12;
nw  = 1e-20;
Qwhe = diag([nlr, nll, nw]);
Q(4:6, 4:6) = Qwhe;
Q(1:3, 4:6) = Nn * Qwhe * dt / 2;
Q(4:6, 1:3) = Q(1:3, 4:6)';

% Обновление матрицы ковариации фильтра Калмана
P = EKF_prediction(P, F, Q);

%% Фильтр Калмана - коррекция, GPS
if (gps_update)
    % Вектор измерений
    v = zeros(4, 1);
    pos_gps_hat = rn + Cbn * lb;
    v(1:2, 1) = pos_gps_hat(1:2, 1) - pos_gps(1:2, 1);

    vb = [Vb_odometer; 0; 0];
    wb = [0; 0; Wb_odometer];
    vel_gps_hat = Cbn * (vb + skew(wb) * lb);
```

```
    v(3:4, 1) = vel_gps_hat(1:2, 1) - vel_gps(1:2, 1);

    % Матрица измерений
    H = zeros(4, 6);

    H(1:2, 1:2) = eye(2);
    dpsi = skew(Cbn * lb);
    H(1:2, 3) = dpsi(1:2, 3);

    dpsi = skew(vel_gps_hat);
    H(3:4, 3) =  dpsi(1:2, 3);
    domega   = -Cbn * skew(lb);
    H(3:4, 6) = domega(1:2, 3);

    H(3, 4)   = Cbn(1, 1) * fir / 2;
    H(3, 5)   = Cbn(1, 1) * fil / 2;
    H(4, 4)   = Cbn(2, 1) * fir / 2;
    H(4, 5)   = Cbn(2, 1) * fil / 2;

    % Шумы измерений
    R(1:2, 1:2) = eye(2) * 1e-1;
    R(3:4, 3:4) = eye(2) * 1e-2;

    % Обновление
    [x, P] = EKF_correction(P, v, H, R);

    % Коррекция оценки координат
    rn = rn - [x(1, 1); x(2, 1); 0];

    % Коррекция оценки ориентации
    En  = [1, -x(3, 1), 0; x(3, 1), 1, 0; 0, 0, 1];
    Cbn = En * Cbn;
    Cbn = dcmnormalize(Cbn);

    % Коррекция оценок ошибок датчиков
    dlr = dlr + x(4, 1);
    dll = dll + x(5, 1);
    dw  = dw + x(6, 1);
end

%%  Фильтр Калмана - коррекция, одометр
% Вектор измерений
v = dthet_z / dt - Wb_odometer;

% Матрица измерений
H = zeros(1, 6);
H(1, 4) = -fir / d;
H(1, 5) =  fil / d;
H(1, 6) =  1;

% Шумы измерений
R = 1e-2;
```

```
% Обновление
[x, P] = EKF_correction(P, v, H, R);

% Коррекция оценки координат
rn = rn - [x(1, 1); x(2, 1); 0];

% Коррекция оценки ориентации
En  = [1, -x(3, 1), 0; x(3, 1), 1, 0; 0, 0, 1];
Cbn = En * Cbn;
Cbn = dcmnormalize(Cbn);

% Коррекция оценок ошибок датчиков
dlr = dlr + x(4, 1);
dll = dll + x(5, 1);
dw  = dw + x(6, 1);

end
```

Модуль навигации с коррекцией от UWB

Файл **navigation_ststem_UWB.m**.

```
function [rn, Cbn, Vb_odometer, Wb_odometer, P, dlu, dw, dlr, dll, dd] = ...
   navigation_system_UWB(rn, Cbn, P, dt, dThe, fir, fil, lr, ll, d, ...
   Ranges, Anchors, lu, uwb_update, dlu, dw, dlr, dll, dd)
%   Алгоритм системы навигации колесного робота с коррекцией от локальной
%   радионавигационной системы UWB
%
%   Входные аргументы:
%   rn - оценка координат робота
%   Cbn - оценка матрицы направляющих косинусов ориентации робота
%   P - матрица ковариации фильтра Калмана
%   dt - шаг интегрирования
%   dThe - измерение приращения угла курса гироскопом за шаг интегрирования
%   fir - скорость вращения правого колеса
%   fil - скорость вращения левого колеса
%   lr - длина окружности правого колеса
%   ll - длина окружности левого колеса
%   d - длина оси колесной пары
%   Ranges - измерения расстояний до базовых станций системы UWB
%   Anchors - координаты базовых станций системы UWB
%   lu - вектор плеча установки приемника UWB относительно центра колесной
%   пары
%   uwb_update - индикатор доступности измерения от приемника UWB
%   dlu - оценка ошибки определения плеча установки приемника UWB
%   dw - оценка сдвига нуля курсового гироскопа
%   dlr - оценка ошибки длины окружности правого колеса
%   dll - оценка ошибки длины окружности левого колеса
%   dd - оценка ошибки длины оси колесной пары
%
%   Выходные аргументы:
%   rn - оценка координат робота
```

```
%   Cbn - оценка матрицы направляющих косинусов ориентации робота
%   Vb_odometer - оценки скорости продольного перемещения одометром
%   Wb_odometer - оценка скорости курсовго разворота одометром
%   P - матрица ковариации фильтра Калмана
%   dlu - оценка ошибки определения плеча установки приемника UWB
%   dw - оценка сдвига нуля курсового гироскопа
%   dlr - оценка ошибки длины окружности правого колеса
%   dll - оценка ошибки длины окружности левого колеса
%   dd - оценка ошибки длины оси колесной пары

%% UWB
% компенсация ошибок установки приемника UWB
lu  = lu - dlu;

%% Одометр
% компенсация ошибок одометра - длины правого и левого колес и длины оси
% колесной пары
lr = lr - dlr;
ll = ll - dll;
d  = d - dd;

% Скорости продольного перемещения и курсового разворота
Vb_odometer = 1/2 * (fir * lr + fil * ll);
Wb_odometer = 1/d * (fir * lr - fil * ll);

% Компенсация сдвига нуля гироскопа
dthet_z = dThe - dw * dt;

% Обновление матрицы направляющих косинусов (интегрирование ориентации)
rot_norm = norm(dthet_z);
sr_a = 1 - (rot_norm ^ 2 / 6) + (rot_norm ^ 4 / 120);
sr_b = (1 / 2) - (rot_norm ^ 2 / 24) + (rot_norm ^ 4 / 720);
Cbb = [
    1 - sr_b * dthet_z ^ 2, -sr_a * dthet_z, 0;
    sr_a * dthet_z, 1 - sr_b * dthet_z ^ 2, 0;
    0, 0, 1
    ];
Cbn=Cbn * Cbb;
Cnb=Cbn';

% Обновление координат
rn = rn + Cbn * [Vb_odometer; 0; 0] * dt;

%% Фильтр Калмана - предсказание

% Матрица динамики системы в непрерывном времени
An = zeros(3);
An(1,3) =  Cbn(2,1) * Vb_odometer;
An(2,3) = -Cbn(1,1) * Vb_odometer;

% Матрица динамики системы в дискретном времени
Fnd=eye(3) + An * dt + An * An * dt * dt / 2;
```

```
% Матрица влияния шумов системы
Nn = zeros(3, 3);
Nn(1, 1) =  Cbn(1, 1) * fir / 2;
Nn(1, 2) =  Cbn(1, 1) * fil / 2;
Nn(2, 1) =  Cbn(2, 1) * fir / 2;
Nn(2, 2) =  Cbn(2, 1) * fil / 2;
Nn(3, 3) = -1;

% Матрица ковариации шумов системы
nlr = 1e-6;
nll = 1e-6;
nw  = 1e-8;
R = diag([nlr, nll, nw]);
Qn = Nn * R * Nn';

% Дискретизация с интегрированием методом трапеций
Qnd = dt / 2 * (Fnd * Qn + Qn * Fnd');

% Матрица динамики системы и матрица ковариации шумов измерений
F = zeros(9);
Q = zeros(9);
F(1:3, 1:3) = Fnd;
F(4:9, 4:9) = eye(6);
F(1:3, 4:6) = Nn * dt;

Q(1:3, 1:3) = Qnd;
nlr = 1e-12;
nll = 1e-12;
nw  = 1e-20;
Qwhe = diag([nlr, nll, nw]);
Q(4:6, 4:6) = Qwhe;
Q(1:3, 4:6) = Nn * Qwhe * dt / 2;
Q(4:6, 1:3) = Q(1:3, 4:6)';
nd  = 1e-12;
Q(7, 7) = nd;
nlu = 1e-16;
Qlev = diag([nlu, nlu]);
Q(8:9, 8:9) = Qlev;

% Обновление матрицы ковариации фильтра Калмана
P = EKF_prediction(P, F, Q);

%% Фильтр Калмана - коррекция, UWB
if (uwb_update)
   for j=1:size(Ranges, 1)
      % Вектор измерений
      l = [lu; 0];
      ru = rn + Cbn * [lu; 0];
      ra = Anchors(j, :)';
      r = norm(ru - ra);
      v = r - Ranges(j);
```

```
        % Матрица измерений
        H = zeros(1, 9);
        dr0 = rn' - ra' + l' * Cnb;
        H(1, 1:2) = 1 / r * dr0(1, 1:2);
        dl = l' + (rn' - ra') * Cbn;
        H(1,8:9) =  1 / r * dl(1, 1:2);
        dpsi = (rn' - ra' + l' * Cnb) * skew(Cbn * l);
        H(1,3) =  1 / r * dpsi(1, 3);

        % Шумы измерений
        R = 5e-2;

        % Обновление
        [x, P] = EKF_correction(P, v, H, R);

        % Коррекция оценки координат
        rn = rn - [x(1, 1); x(2, 1); 0];

        % Коррекция оценки ориентации
        En  = [1, -x(3, 1), 0; x(3, 1), 1, 0; 0, 0, 1];
        Cbn = En * Cbn;
        Cbn = dcmnormalize(Cbn);

        % Коррекция оценок ошибок датчиков
        dlr = dlr + x(4, 1);
        dll = dll + x(5, 1);
        dw  = dw + x(6, 1);
        dd  = dd + x(7, 1);
        dlu = dlu + x(8:9, 1);
    end
end

%%  Фильтр Калмана - коррекция, одометр
% Вектор измерений
v = dthet_z / dt - Wb_odometer;

% Матрица измерений
H = zeros(1, 9);
H(1, 4) = -fir / d;
H(1, 5) =  fil / d;
H(1, 6) =  1;
H(1, 7) = (fir * lr - fil * ll) / d ^ 2;

% Шумы измерений
R = 1e-2;

% Обновление
I = eye(9);
K = (P * H') / (H * P * H' + R);
P = (I - K * H) * P * (I - K * H)' + K * R * K';
x = K * v;
```

```
% Коррекция оценки координат
rn = rn - [x(1, 1); x(2, 1); 0];

% Коррекция оценки ориентации
En  = [1, -x(3, 1), 0; x(3, 1), 1, 0; 0, 0, 1];
Cbn = En * Cbn;
Cbn = dcmnormalize(Cbn);
% Коррекция оценок ошибок датчиков
dlr = dlr + x(4, 1);
dll = dll + x(5, 1);
dw  = dw + x(6, 1);
dd  = dd + x(7, 1);
dlu = dlu + x(8:9, 1);

end
```

Модуль навигации с применением фильтра частиц для коррекции от UWB

Файл **density.m**.

```
function d = density(gDistr, x)
% Плотность вероятности гауссовкой случайной величины

[nrow, ncol] = size(x);

if(nrow ~= gDistr.n)
  error('wrong input size for density calculation.');
end

d = zeros(1, ncol);
x = x - repmat(gDistr.mean, 1, ncol);



for i = 1:ncol
  d(i) = gDistr.const * exp(-x(:, i)' * gDistr.invCov * x(:, i) / 2);
end
```

Файл **drawSamples.m**

```
function X = drawSamples(gDistr, totalNum)
% Выборка из гауссовского распределения

X = gDistr.UsqrtT * randn(gDistr.n, totalNum) + repmat(gDistr.mean, 1, totalNum);
```

Файл **pf_get_state.m**

```
function x = pf_get_state(fltr, type)
% Оценка вектора состояния фильтра частциц

switch type
```

```
    case 1 % состояние частицы с наибольшим весом
        idx = fltr.w == max(fltr.w);
        x = mean(fltr.p(:,idx),2);
    case 2 % Среднее состояние по всем частицам
        wp = fltr.p .* repmat(fltr.w, size(fltr.p, 1), 1);
        x = sum(wp, 2);
        x(3, 1) = pi2pi(x(3, 1));
end

end
```

Файл **pf_init.m**

```
function fltr = pf_init(initPos, initAng, nstate, nmeas)
% Инициализация фильтра частиц
%
%   Входные аргументы:
%   initPos - начальные коррдинаты робота в навигационной СК
%   initAng - начальный угол курса робота в навигационной СК
%   nstate - размерность вектора состояния фильтра
%   nmeas - размерность вектора измерений фильтра
%
%   Выходные аргументы:
%    fltr - структура фильтра частиц

%%
% Количество частиц в фильтре
nParticle = 1000;

% Шумы системы
wPos = 1e-4;
wAng = 1e-3;

% Шумы измерений
wRnz = 1e0;

% Шум начальной выставки
wPosi = 1e2;
wAngi = 1e0;

% Шумы управлений
wUps = 1e-1;
wdThe = 1e-2;

% Шумы системы
covW = diag([wPos, wPos, wAng]);
fltr.w_d.mean = zeros(nstate,1);
fltr.w_d.n = nstate;
fltr.w_d.cov = covW;
fltr.w_d.const = 1 / sqrt((2 * pi)^nstate * det(covW));
fltr.w_d.invCov = inv(covW);
[U, T] = schur(covW);
```

```
fltr.w_d.UsqrtT = U * (T .^ 0.5);

% Шумы измерений
covV = eye(nmeas) * wRnz;
fltr.v_d.mean = zeros(nmeas,1);
fltr.v_d.n = nmeas;
fltr.v_d.cov = covV;
fltr.v_d.const = 1 / sqrt((2 * pi)^nmeas * det(covV));
fltr.v_d.invCov = inv(covV);
[U, T] = schur(covV);
fltr.v_d.UsqrtT = U * (T .^ 0.5);

% Начальное состояние
initMean = [initPos; initAng;];
initCov = diag([wPosi, wPosi, wAngi]);
initDistr.mean = initMean;
initDistr.n = nstate;
initDistr.cov = initCov;
initDistr.const = 1 / sqrt((2 * pi)^nstate * det(initCov));
initDistr.invCov = inv(initCov);
[U, T] = schur(initCov);
initDistr.UsqrtT = U * (T .^ 0.5);

% Выборка частиц начального состояния
fltr.N = nParticle;
fltr.p = drawSamples(initDistr, nParticle);
fltr.w = ones(1, nParticle) / nParticle;
fltr.invQ = inv(covW);
fltr.invR = inv(covV);
fltr.Q = diag([wUps, wdThe]);

end
```

Файл **pf_predict_update.m**

```
function fltr = pf_predict_update(fltr, upsilon, dtheta, Anchors, Ranges, update, dt)
% Алгоритм фильтра частиц
%
%   Входные аргументы:
%   fltr - структура фильтра частиц
%   upsilon - управляющее воздействие робота - скорость продольного
%   перемещения
%   dtheta - управляющее воздействие робота - угловая скорость курсового
%   разворота
%   Ranges - измерения расстояний до базовых станций системы UWB
%   Anchors - координаты базовых станций системы UWB
%   update - индикатор доступности измерения от приемника UWB
%   dt - шаг интегрирования
%
%   Выходные аргументы:
%   fltr - структура фильтра частиц
```

```
for p = 1:fltr.N
    % Предсказание
    xp = fltr.p(:, p);
    xp = state_predict(xp, upsilon, dtheta, dt);
    fltr.p(:, p) = xp;

    % Коррекция
    if update == 1
        H  = dh(xp, Anchors);
        yp = h(xp, Anchors);
        P  = inv(fltr.invQ + H' * fltr.invR * H);
        P  = (P + P') / 2;
        mu = P * (fltr.invQ * xp + H' * fltr.invR * (Ranges - yp + H * xp));

        gauss.mean = mu;
        gauss.n = size(mu, 1);
        gauss.cov = P;
        gauss.const = 1 / sqrt((2 * pi)^size(mu, 1) * det(P));
        gauss.invCov = inv(P);
        [U, T] = schur(P);
        gauss.UsqrtT = U * (T .^ 0.5);

        % Выборка частицы
        fltr.p(:, p) = drawSamples(gauss, 1);
        fltr.w(:, p) = 1.0 / fltr.N;

        % Вектор измерений
        y_pi = h(fltr.p(:, p), Anchors);

        % Плотность вероятности измерений
        likelihood = density(fltr.v_d, Ranges - y_pi);

        % Плотность вероятности системы
        prior = density(fltr.w_d, fltr.p(:,p) - xp);
        proposal = density(gauss, fltr.p(:,p));

        % Обновление веса частицы
        fltr.w(p) = fltr.w(p) * likelihood * prior / proposal;
    end

end

    %% Динамика вектора состояния
    function x = state_predict(x, upsilon, dtheta, dt)
        xy = x(1:2,:);
        theta = x(3,:);

        upsilon_noise = 1e0;
        dtheta_noise  = 1e-1;

        upsilon = upsilon + randn * upsilon_noise;
        dtheta = dtheta + randn * dtheta_noise;
```

```
        xy(1) = xy(1) + upsilon * cos(theta) * dt;
        xy(2) = xy(2) + upsilon * sin(theta) * dt;
        theta = theta + dtheta * dt;

        x(1:2,:) = xy;
        x(3,:) = theta;
    end

    %% Матрица измерений
    function H = dh(x, Anchors)
        tag = [x(1:2, 1); 0.0];
        H = zeros(size(Anchors, 1), 3);
        for i=1:size(Anchors, 1)
            anchor = Anchors(i, :)';
            H(i, 1) = (tag(1) - anchor(1)) / norm(tag - anchor);
            H(i, 2) = (tag(2) - anchor(2)) / norm(tag - anchor);
        end
    end

    %% Вектор измерений
    function y = h(x, Anchors)
        tag = [x(1:2, 1); 0.0];
        y = zeros(size(Anchors, 1), 1);
        for i=1:size(Anchors, 1)
            anchor = Anchors(i, :)';
            y(i, 1) = norm(tag - anchor);
        end
    end
end
```

Файл **pf_resample.m**

```
function fltr = pf_resample(fltr, threshold)
% Ресэмплинг частиц

Nmin = fltr.N * threshold;
sum_w = sum(fltr.w);
fltr.w = fltr.w / sum_w;

[keep, Neff] = stratified_resample(fltr.w);
if Neff < Nmin
    fltr.p(:, 1:fltr.N) = fltr.p(:,keep);
    fltr.w(:, :) = ones(1, fltr.N) / fltr.N;
end

%%
function [keep, Neff] = stratified_resample(w)

w = w / sum(w);
Neff = 1 / sum(w .^ 2);
```

```
len = length(w);
keep = zeros(1,len);
select = stratified_random(len);
w = cumsum(w);

ctr = 1;
for i = 1:len
   while ctr<=len && select(ctr)<w(i)
      keep(ctr) = i;
      ctr = ctr+1;
   end
end
end

%%
function s = stratified_random(N)

k = 1/N;
di = (k/2):k:(1-k/2);
s = di + rand(1,N) * k - (k/2);
end

end
```

Файл **radio_navigation_system_UWB.m**

```
function [fltr, robot_state] = radio_navigation_system_UWB(fltr, upsilon, dtheta, Ranges,
Anchors, update, dt)
%   Алгоритм системы навигации колесного робота на базе локальной
%   радионавигационной системы UWB и фильтра частиц
%
%   Входные аргументы:
%   fltr - структура фильтра частиц
%   upsilon - управляющее воздействие робота - скорость продольного
%   перемещения
%   dtheta - управляющее воздействие робота - угловая скорость курсового
%   разворота
%   Ranges - измерения расстояний до базовых станций системы UWB
%   Anchors - координаты базовых станций системы UWB
%   update - индикатор доступности измерения от приемника UWB
%   dt - шаг интегрирования
%
%   Выходные аргументы:
%   fltr - структура фильтра частиц
%   robot_state - оценка вектора состояния робота

%% Предсказание/коррекция
fltr = pf_predict_update(fltr, upsilon, dtheta, Anchors, Ranges, update, dt);

%% Ресемплинг частиц
fltr = pf_resample(fltr, 0.75);
```

```
%% Состояние
robot_state = pf_get_state(fltr, 2);

end
```