МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ
И ИНТЕЛЛЕКТУАЛЬНЫХ ТЕХНИЧЕСКИХ СИСТЕМ

Кафедра "Информатика и управление в технических системах"

РЕШЕНИЕ ЗАДАЧИ ЛОКАЛИЗАЦИИ В СВОБОДНОЙ ФОРМЕ

Методические указания
к выполнению лабораторной работы
по дисциплине "Навигация мобильных робототехнических систем"
для студентов направления подготовки
27.03.04 – Управление в технических системах

Севастополь
СевГУ
2025

УДК [007.52:531.1]:519.2](076.5)
ББК 32.816:22.172я73
Р47

Р е ц е н з е н т:
В. А. Крамарь – д-р техн. наук, профессор, профессор кафедры "Информатика и управление в технических системах"

Составители: А. А. Кабанов, А. Д. Ляшко

Р47 **Решение задачи локализации в свободной форме :** методические указания к выполнению лабораторной работы по дисциплине "Навигация мобильных робототехнических систем" для студентов направления подготовки 27.03.04 – Управление в технических системах / Севастопольский государственный университет, Институт радиоэлектроники и интеллектуальных технических систем, кафедра "Информатика и управление в технических системах ; сост.: А. А. Кабанов, А. Д. Ляшко. – Севастополь : СевГУ, 2025. – 16 с. – Текст : электронный.

Приводится описание лабораторной работы, целью которой является совершенствование практических навыков решения задачи локализации мобильного робота в априорно известной среде, а также применение одного из изученных алгоритмов локализации (фильтра частиц, метода Монте-Карло или фильтра Калмана).

Предназначены для студентов дневной и заочной форм обучения направления подготовки 27.03.04 – Управление в технических системах.

УДК [007.52:531.1]:519.2](076.5)
ББК 32.816:22.172я73

Методические указания рассмотрены и утверждены на заседании кафедры "Информатика и управления в технических системах", протокол № 6 от 25 марта 2025 г.

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ 4
2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4
2.1 Модель мира 4
2.2 Модель робота 5
2.3 Модель измерений 6
3 ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ 7
4 СОДЕРЖАНИЕ ОТЧЁТА И ПОРЯДОК ЗАЩИТЫ РАБОТЫ 9
5 КОНТРОЛЬНЫЕ ВОПРОСЫ 9
БИБЛИОГРАФИЧЕСКИЙ СПИСОК 10
ПРИЛОЖЕНИЕ А 11
А.1 Пример (Python) 11
А.2 Пример (Matlab) 14

1 ЦЕЛЬ РАБОТЫ

Целью данной работы является совершенствование практических навыков решения задачи локализации мобильного робота в априорно известной среде, а также применение одного из изученных алгоритмов локализации (фильтра частиц, метода Монте-Карло или фильтра Калмана).

Все вычислительные процедуры и построения предлагается проводить с помощью математических пакетов MathCAD, MatLab, Octave, Python, которые обеспечивают высокую точность расчетов и удобство визуализации результатов.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 Модель мира

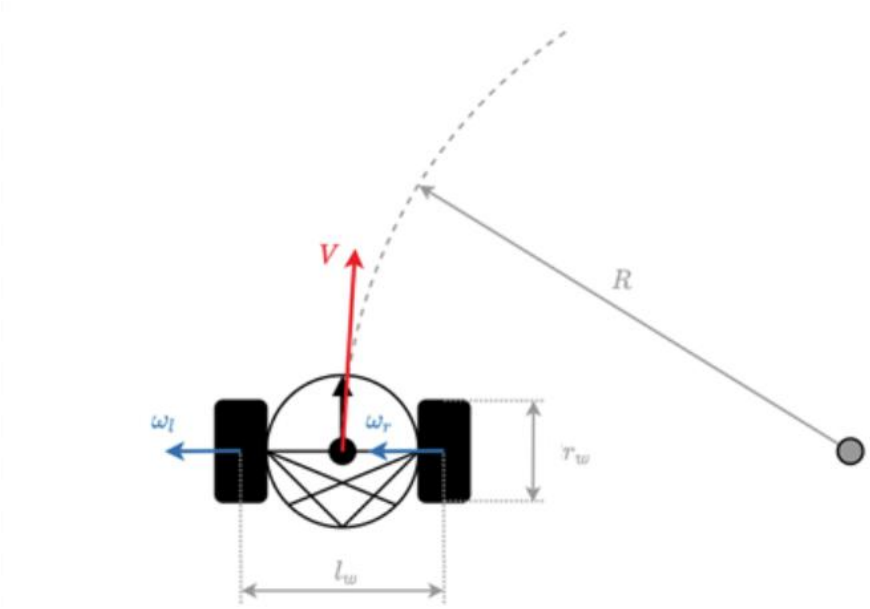
Модель мира в данной лабораторной работе представляет собой прямоугольную область с жёсткими границами, аналогичную модели, использовавшейся при изучении фильтра частиц, но с двумя ключевыми отличиями:

1. Размер карты по осям *x*, *y* различается.

2. Отсутствие цикличности. Это означает, что при достижении границы карты робот не сможет двигаться дальше в этом направлении и останавливается. Значение угла курса $\psi = 0^{o}$ соответствует движению робота вправо (в сторону увеличения координаты *x*), а разворот по курсу осуществляется по часовой стрелке. Начало координат располагается в верхнем углу карты.

2.2 Модель робота

В рамках решения поставленной задачи требуется разработка специализированной модели мобильного робота, обладающей отличительными особенностями по сравнению с классическими моделями, рассмотренными ранее. В качестве базовой архитектуры предлагается использовать класс *Robot (Примеры представлены в приложении* А) и внести в него соответствующие изменения.



Робот представляет собой двухколёсную платформу, во многом похожую на основу робота-пылесоса. Движение робота осуществляется под воздействием вращения его колёс. При этом линейную и угловую скорости движения робота можно рассчитать по следующим соотношениям:

$$V = r_w \frac{w_l + w_r}{2}$$

$$\dot{\psi} = r_w \frac{w_l - w_r}{l_w}$$

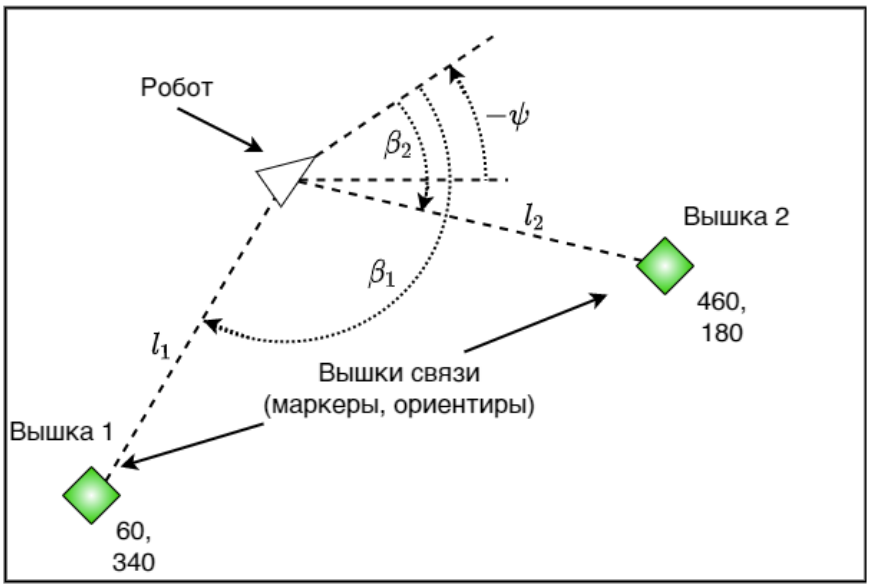
Таким образом, для разворота робота по углу курса необходимо задать разницу угловых скоростей вращения его колёс. При этом робот будет осуществлять движение по окружности с радиусом:

$$R = \frac{V}{\dot{\psi}}$$

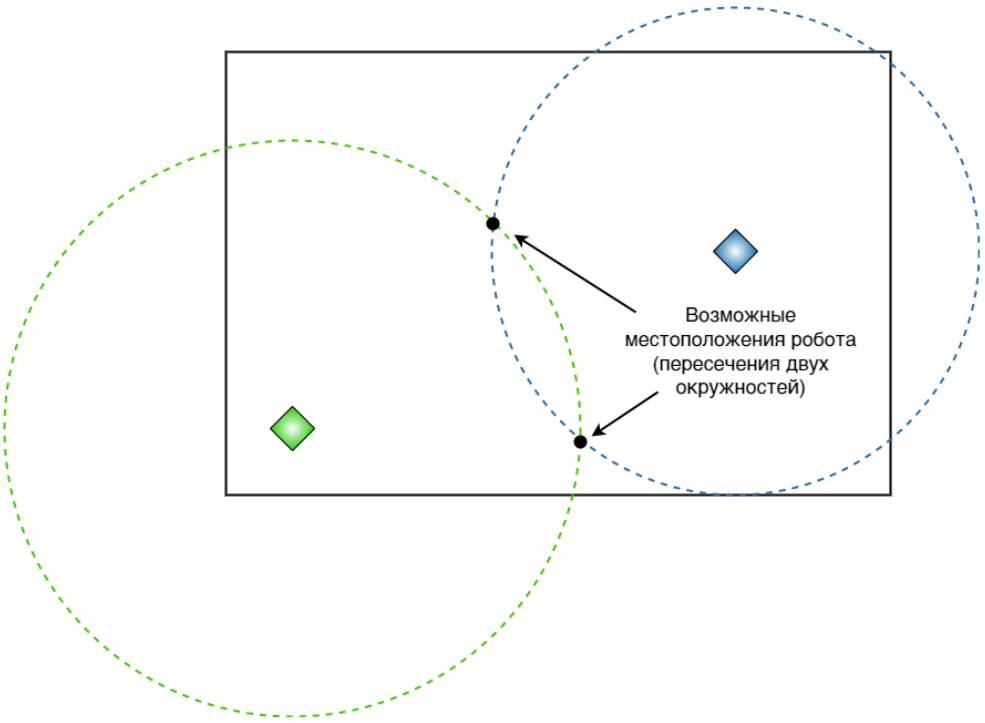
В качестве команд управления движением робота будут использоваться заданные значения угловых скоростей вращения колёс w_{lc}, w_{rc} (рад/с) и время выполнения команды t_c(с).

При моделировании будем полагать, что радиус окружности колеса $r_w = 0.15$ м, а расстояние между колёсами $l_w = 0.5$ м.

2.3 Модель измерений



В моделируемом мире расположены две вышки связи, координаты которых показаны на рисунке. Робот может измерять дальности l_1, l_2 до каждой из этих вышек, а также углы пеленга β_1, β_2. Угол пеленга – угол между продольной осью робота и направлением на цель. Координаты местоположения робота могут быть рассчитаны на основе двух измерений дальностей до вышек как точка пересечения двух соответствующих окружностей:



Нетрудно заметить, что задача нахождения точки пересечения двух окружностей имеет два возможных решения. Их можно найти, решив систему уравнений:

$$\begin{cases}(x-x_1)^2+(y-y_1)^2=l_1^2\\(x-x_2)^2+(y-y_2)^2=l_2^2\end{cases}$$

где x_1, y_1, x_2, y_2 – координаты центров соответствующих окружностей; l_1, l_2 – их радиусы.

Углы пеленга формируются как разность направления на цель и текущего угла курса робота:

$$\begin{cases}\beta_1=\operatorname{atan}\left(\dfrac{y-y_1}{x-x_1}\right)-\psi\\\beta_2=\operatorname{atan}\left(\dfrac{y-y_2}{x-x_2}\right)-\psi\end{cases}$$

При моделировании будем полагать, что шумы измерений расстояний до вышек связи *sense_len_noise = 10 м*, а шумы измерения углов пеленга *sense_ang_noise = 0.3 рад*.

3 ЗАДАНИЕ НА ВЫПОЛНЕНИЕ РАБОТЫ

Необходимо разработать программу, осуществляющую локализацию робота в двумерном пространстве. В этой задаче нет шаблонов классов и методов решения задачи, можно выбрать любой понравившийся метод.

Требование к решению:

- определение координат местоположения робота с точностью до 10 метров.
- определение угла курса робота с точностью до 8 градусов.

Параметры моделирования:

Все шумы представлены в виде нормальных законов распределения с нулевым математическим ожиданием из заданным СКО.

Шум измерений дальности *sense_len_noise* = 10 м.

Шум измерений угла пеленга *sense_ang_noise* = 0.3 рад.

Шум движения (установки заданной угловой скорости вращения колёс) *motion_noise* = 0.2 рад/с.

В начальный момент времени робот находится в состоянии полной неопределённости.

Рекомендуемый шаг моделирования *dt = 0.1 с*.

Представленные наборы входных данных гарантируют наличие решения задачи пересечения окружностей для нахождения координат местоположения робота.

Формат входных данных:

$l_1\ \beta_1\ l_2\ \beta_2$

$\omega_{lc}\ \omega_{rc}\ t_c$

$l_1\ \beta_1\ l_2\ \beta_2$

$\omega_{lc}\ \omega_{rc}\ t_c$

...

$l_1\ \beta_1\ l_2\ \beta_2$

$\omega_{lc}\ \omega_{rc}\ t_c$

$l_1\ \beta_1\ l_2\ \beta_2$

Последовательные результаты совершённых измерений и заданные команды движения робота. Количество пар *измерение-команда* может отличаться для разных тестов. Обратите внимание, что количество измерений на единицу больше количества команд движения. Программа начинается и заканчивается измерением местоположения робота!

Формат выходных данных:

x, *y*, ψ - Координаты местоположения (м.) и угол курса робота (град. в диапазоне [0, 360]).

Несколько соображений по выбору метода решения задачи:

Для решения задачи методом Монте-Карло целесообразно разделить карту на клетки размером 10 м. для решения дискретной задачи локализации. В этом случае вам остаётся лишь определить, в какой из клеток находится робот для обеспечения требования по точности определения местоположения.

Для решения задачи с применением фильтра Калмана необходимо написать функцию расчёта оценки местоположения робота по полученным измерениям дальностей до вышек связи. Также вам потребуется придумать какую-нибудь хитрость для составления линейной модели движения робота.

При использовании фильтра частиц для решения задачи модель измерений робота упрощается, так как нет необходимости осуществлять пересчёт дальностей до вышек связи и углов пеленга в навигационные параметры робота. Однако в этом случае для каждого набора измерений у вас будет несколько возможных местоположений робота (проблема неоднозначности измерений). Вам потребуется учитывать это при построении фильтра. Входные данные представлены в таблице 1.

Таблица 1 – Варианты заданий

Вариант 1 (Входные данные)	**Вариант 2 (Входные данные)**
348.4 3.9 176.7 1.7	335.5 1.8 475.5 0.0
80.0 79.1 28.0	150.0 149.5 16.0
438.6 0.5 28.5 5.5	265.2 1.2 182.5 4.7
61.0 61.0 22.0	120.0 121.0 6.0
266.4 0.4 180.7 3.3	321.0 2.8 136.8 5.2
25.0 24.1 12.3	120.0 120.2 8.0
237.2 5.3 216.9 2.0	451.6 2.9 128.6 4.4
63.0 63.0 22.0	20.0 22.0 7.0
196.1 4.2 313.8 1.7	465.5 5.1 114.6 0.4
82.0 81.1 28.0	38.0 37.6 19.0
336.3 1.1 180.2 4.8	500.7 4.0 63.0 4.8
2.0 2.9 22.0	37.0 37.5 19.0
341.8 4.2 182.3 2.3	520.4 4.8 117.4 4.4
80.0 79.1 28.0	60.0 61.0 5.0
431.1 0.8 28.7 5.0	497.2 5.3 117.4 4.7
Выходные данные	**Выходные данные**
453.1 157.1 131.1	482.9 65.7 183.8

4 СОДЕРЖАНИЕ ОТЧЁТА И ПОРЯДОК ЗАЩИТЫ РАБОТЫ

В отчёте по лабораторной работе должны быть отражены цель, предварительные математические построения и расчёты, необходимые сведения о методах решения рассматриваемых задач и средствах системы Matlab, Python, тексты программ, схемы моделирования, численные и графические результаты расчётов и построений, выводы по работе.

Защита результатов лабораторной работы производится за компьютером при наличии работающих программ и полностью оформленного отчёта. В ходе защиты студент может получить дополнительные теоретические вопросы и практические задания по существу работы.

5 КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Какие основные принципы лежат в основе работы фильтра частиц?
2. Какие данные используются в фильтре частиц для определения положения робота?
3. Как происходит инициализация частиц в алгоритме фильтра частиц?
4. Каким образом обновляются частицы в процессе работы фильтра частиц?
5. Как осуществляется оценка состояния робота с использованием фильтра частиц?
6. Какие проблемы могут возникнуть при использовании фильтра частиц в навигации мобильных роботов?
7. Какие преимущества и недостатки имеет метод Монте-Карло по сравнению с другими методами локализации роботов?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Бесконтактные средства локальной ориентации роботов: учебно-методическое пособие / С. М. Власов, В. И. Бойков, С. В. Быстров, В. В. Григорьев. – Санкт-Петербург : НИУ ИТМО, 2017. – 169 с. – Текст: электронный // Лань: электронно-библиотечная система. – URL: https://e.lanbook.com/book/110461 (дата обращения: 02.04.2025). – Режим доступа: для авториз. пользователей.

2. Баланов, А. Н. Машинное обучение и искусственный интеллект : учебное пособие для вузов / А. Н. Баланов. – 2-е изд., стер. – Санкт-Петербург : Лань, 2025. – 172 с. – ISBN 978-5-507-52891-2. – Текст : электронный // Лань : электронно-библиотечная система. – URL: https://e.lanbook.com/book/462248 (дата обращения: 14.04.2025). – Режим доступа: для авториз. пользователей.

3. Peter Corke. Robotics, Vision and Control. Fundamental Algorithms in MATLAB // Springer, 2011, ISBN 978-3-642-20143-1, 570 p.

4. Akai N. Reliable Monte Carlo localization for mobile robots //Journal of Field Robotics. – 2023. – Т. 40. – №. 3. – С. 595-613.

5. Cook G., Zhang F. Mobile robots: navigation, control and sensing, surface robots and AUVs. – John Wiley & Sons, 2020.

ПРИЛОЖЕНИЕ А
А1. Пример (Python)

```
import random
from math import *

world_size_x = 600
world_size_y = 400
landmarks = [[60, 340], [460, 180]]

class Robot:
    # class wide variables
    particle = False # particle flag, particles should have a different visuals
    path = [] # used to store the robot's traveled path
    # Actual parameters
    x = 0.0
    y = 0.0
    orientation = 0.0


    def __init__(self, particle = False):
        self.x = random.random() * 600
        self.y = random.random() * 400
        self.orientation = random.random() * 2.0 * pi
        self.wheels_noise = 0.0
        self.sense_len_noise = 0.0
        self.sense_ang_noise = 0.0
        # Graphic stuff
        self.path = []
        self.path.append((self.x, self.y))
        self.particle = particle


    def set(self, new_x, new_y, new_orientation):
        #new_x %= world_size    # cyclic truncate
        #new_y %= world_size
        new_orientation %= 2*pi
        self.x = float(new_x)
        self.y = float(new_y)
        self.orientation = float(new_orientation)
        self.path = []
        self.path.append((self.x, self.y))

    def sense(self):
        Z = []
        dd = sqrt((landmarks[1][0] - landmarks[0][0])**2 + (landmarks[1][1] - landmarks[0][1])**2)
        # get distances
        for i in range(len(landmarks)):
            d = sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
            d += random.gauss(0.0, self.sense_len_noise)
```

```
            a = atan2(landmarks[i][1] - self.y, landmarks[i][0] - self.x) - self.orientation
            a += random.gauss(0.0, self.sense_ang_noise)
            Z.append(d)
            Z.append(a % (2*pi))
        # Check if the measurements allow to build a valid solution
        while Z[0] + Z[2] <= dd:
            print("whhopsie")
            Z[0] += self.sense_len_noise / 6
            Z[2] += self.sense_len_noise / 6
        return Z

    def set_noise(self, new_w_noise, new_sd_noise, new_sa_noise):
        # makes it possible to change the noise parameters
        # this is often useful in particle filters
        self.wheels_noise = float(new_w_noise)
        self.sense_len_noise = float(new_sd_noise)
        self.sense_ang_noise = float(new_sa_noise)

    def move(self, left, right, time):
        # turn, and add randomness to the turning command
        left += random.gauss(0.0, self.wheels_noise)
        right += random.gauss(0.0, self.wheels_noise)
        rate = 0.15 * (left - right) / 2 / 0.5
        vel = 0.15 * (left + right) / 2
        #print("d --- {} {}".format(vel, rate))
        print("{:.1f} {:.1f} {:.1f}".format(left, right, time))

        x = self.x
        y = self.y
        orientation = self.orientation
        for i in range(0, int(time * 10)):
            orientation += rate * 0.1
            orientation %= 2 * pi
            dist = vel * 0.1
            xd = (cos(orientation) * dist)
            yd = (sin(orientation) * dist)
            x = max(min(599, (x + xd)), 0)
            y = max(min(399, (y + yd)), 0)
            self.path.append((x, y))
        #print(x)

        #print(self.x)
        part = Robot(self.particle)
        part.set_noise(self.wheels_noise, self.sense_len_noise, self.sense_ang_noise)
        part.set(x, y, orientation)
        part.path = self.path
        return part
```

```
def Gaussian(self, mu, sigma, x):
    # calculates the probability of x for 1-dim Gaussian with mean mu and var. sigma
    return exp(- ((mu - x) ** 2) / (sigma ** 2) / 2.0) / sqrt(2.0 * pi * (sigma ** 2))


def measurement_prob(self, measurement):
    # calculates how likely a measurement should be
    prob = 1.0;
    for i in range(len(landmarks)):
        dist = sqrt((self.x - landmarks[i][0]) ** 2 + (self.y - landmarks[i][1]) ** 2)
        prob *= self.Gaussian(dist, self.sense_noise, measurement[i])
    return prob


def __repr__(self):
    return '[x=%.6s y=%.6s orient=%.6s]' % (str(self.x), str(self.y), str(self.orientation))

@staticmethod
def eval(r, p):
    # find weights
    z = r.sense()
    w = []
    for i in range(0, len(p)):
        w.append(p[i].measurement_prob(z))
    s = sum(w)
    a = []
    for i in range(0, len(p)):
        a.append(w[i] / s)

    # find weighter sum
    s = 0.0
    for i in range(len(p)): # calculate mean error
        dx = (p[i].x - r.x + (world_size/2.0)) % world_size - (world_size/2.0)
        dy = (p[i].y - r.y + (world_size/2.0)) % world_size - (world_size/2.0)
        err = sqrt(dx * dx + dy * dy)
        s += err * a[i]
    return s
```

A2. Пример (MATLAB)

```
% Глобальные параметры
world_size_x = 600;
world_size_y = 400;
landmarks = [60, 340; 460, 180];

% Создание робота
robot = struct();
robot.particle = false;
robot.path = [];
robot.x = rand() * world_size_x;
robot.y = rand() * world_size_y;
robot.orientation = rand() * 2 * pi;
robot.wheels_noise = 0.0;
robot.sense_len_noise = 0.0;
robot.sense_ang_noise = 0.0;
robot.path = [robot.x, robot.y];

% Функция установки параметров робота
function robot = set_robot(robot, new_x, new_y, new_orientation)
   new_orientation = mod(new_orientation, 2*pi);
   robot.x = new_x;
   robot.y = new_y;
   robot.orientation = new_orientation;
   robot.path = [new_x, new_y];
end

% Функция измерения расстояний до ориентиров
function Z = sense(robot)
   global landmarks;
   Z = [];

   dd = sqrt((landmarks(2,1) - landmarks(1,1))^2 + (landmarks(2,2) - landmarks(1,2))^2);

   for i = 1:size(landmarks, 1)
      d = sqrt((robot.x - landmarks(i,1))^2 + (robot.y - landmarks(i,2))^2);
      d = d + normrnd(0.0, robot.sense_len_noise);
      a = atan2(landmarks(i,2) - robot.y, landmarks(i,1) - robot.x) - robot.orientation;
      a = a + normrnd(0.0, robot.sense_ang_noise);
      a = mod(a, 2*pi);
      Z = [Z, d, a];
   end

   % Проверка допустимости измерений
   while Z(1) + Z(3) <= dd
      fprintf("whhopsie\n");
      Z(1) = Z(1) + robot.sense_len_noise / 6;
      Z(3) = Z(3) + robot.sense_len_noise / 6;
   end
end

% Установка шумов
function robot = set_noise(robot, new_w_noise, new_sd_noise, new_sa_noise)
   robot.wheels_noise = new_w_noise;
   robot.sense_len_noise = new_sd_noise;
   robot.sense_ang_noise = new_sa_noise;
end

% Движение робота
function new_robot = move_robot(robot, left, right, time)
```

```
    left = left + normrnd(0.0, robot.wheels_noise);
    right = right + normrnd(0.0, robot.wheels_noise);

    rate = 0.15 * (left - right) / 2 / 0.5;
    vel = 0.15 * (left + right) / 2;

    fprintf('%.1f %.1f %.1f\n', left, right, time);

    x = robot.x;
    y = robot.y;
    orientation = robot.orientation;

    for i = 1:(time * 10)
        orientation = orientation + rate * 0.1;
        orientation = mod(orientation, 2*pi);
        dist = vel * 0.1;
        xd = cos(orientation) * dist;
        yd = sin(orientation) * dist;
        x = max(min(world_size_x-1, x + xd), 0);
        y = max(min(world_size_y-1, y + yd), 0);
        robot.path = [robot.path; x, y];
    end

    new_robot = robot;
    new_robot.x = x;
    new_robot.y = y;
    new_robot.orientation = orientation;
end

% Гауссовское распределение
function prob = gaussian(mu, sigma, x)
    prob = exp(-((mu - x)^2) / (sigma^2) / 2.0) / sqrt(2.0 * pi * (sigma^2));
end

% Вероятность измерения
function prob = measurement_prob(robot, measurement)
    global landmarks;
    prob = 1.0;

    for i = 1:size(landmarks, 1)
        dist = sqrt((robot.x - landmarks(i,1))^2 + (robot.y - landmarks(i,2))^2);
        prob = prob * gaussian(dist, robot.sense_noise, measurement(i));
    end
end

% Оценка частиц
function s = eval_particles(true_robot, particles)
    global world_size_x;

    z = sense(true_robot);
    w = zeros(1, length(particles));

    for i = 1:length(particles)
        w(i) = measurement_prob(particles{i}, z);
    end

    s = sum(w);
    a = w / s;

    % Взвешенная сумма
    s = 0.0;
    for i = 1:length(particles)
        dx = mod(particles{i}.x - true_robot.x + (world_size_x/2.0), world_size_x) - (world_size_x/2.0);
```

```
        dy = mod(particles{i}.y - true_robot.y + (world_size_x/2.0), world_size_x) - (world_size_x/2.0);
        err = sqrt(dx*dx + dy*dy);
        s = s + err * a(i);
    end
end
```

РЕШЕНИЕ ЗАДАЧИ ЛОКАЛИЗАЦИИ В СВОБОДНОЙ ФОРМЕ

Методические указания

Составители: **Кабанов** Алексей Александрович,
Ляшко Александр Дмитриевич

В авторской редакции

Изд. № 86/2025. Объем 1 п. л. Усл. печ. л. 0,93. Уч.-изд. л. 0,98.
Издательский центр ФГАОУ ВО «Севастопольский государственный университет»