МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені І.І.МЕЧНИКОВА

(повне найменування вищого навчального закладу)

Інститут математики, економіки та механіки

(повне найменування інституту, назва факультету (відділення))

Кафедра математичного забезпечення комп'ютерних систем

(повна назва кафедри (предметної, циклової комісії))

Дипломна робота

Бакалавра

(освітньо-кваліфікаційний рівень)

на тему «Програмна реалізація методу орієнтації робота на місцевості»
«Программная реализация метода ориентации робота на местности»
«Software implementation of the robot ground attitude method»

Виконала: студентка IV курсу, групи 1
напряму підготовки (спеціальності)
6.050102 – «Комп'ютерна інженерія»

(шифр і назва напряму підготовки, спеціальності)

Любкевич К.О.

(прізвище та ініціали)

Керівник д-р техн. наук, професор
Гунченко Ю.О.

(прізвище та ініціали)

Рецензент канд. ф.-м. наук, доцент
Крапівний Ю.М.

(прізвище та ініціали)

Рекомендовано до захисту:

Протокол засідання кафедри

№_____ від «___»_____р.

Завідувач кафедри

_____ _____

(підпис) (прізвище, ініціали)

Захищено на засіданні ЕК № ___

протокол №___ від «___» _____р.

Оцінка _____

(за 4-х бальною шкалою, за шкалою ECTS, бал.)

Голова ЕК

_____ _____

(підпис) (прізвище, ініціали)

Одеса - 2017

АННОТАЦИЯ

В дипломной работе разрабатывается тема «Программная реализация метода ориентации робота на местности».

Цель работы – обзор методов навигации и реализация метода ориентации мобильного робота на местности.

В работе рассмотрены датчики, которые могут использоваться при навигации мобильного работа, их достоинства и недостатки, общие проблемы навигации мобильных устройств, основы работы алгоритма одновременной локализации и составлении карты.

В результате выполнения дипломной работы была разработана программная реализация, которая моделирует определение роботом собственного местоположения.

В реализованном методе ориентации мобильного робота на местности применяется фильтр частиц, который представляет доверительное состояние в виде коллекций частиц, соответствующих состоянию. К достоинствам подобного алгоритма относится низкая сложность алгоритма и возможность работать с большими объёмами данных.

АНОТАЦІЯ

У дипломній роботі розробляється тема «Програмна реалізація методу орієнтації робота на місцевості».

Мета роботи - огляд методів навігації та реалізація методу орієнтації мобільного робота на місцевості.

В роботі розглянуті датчики, які можуть використовуватися при навігації мобільного робота, їхні переваги й недоліки, загальні проблеми навігації мобільних пристроїв, основи роботи алгоритму одночасної локалізації і складання карти.

В результаті виконання дипломної роботи була розроблена програмна реалізація, котра моделює визначення роботом власного місця розташування.

У реалізованому методі орієнтації мобільного робота на місцевості застосовується фільтр частинок, який представляє довірливий стан у вигляді колекцій частинок, відповідних стану. До переваг подібного алгоритму відноситься низька складність алгоритму і можливість працювати з великими обсягами даних.

ANNOTATION

In the thesis the theme "Software implementation of the robot ground attitude method" is being developed.

The aim of the work is overview of navigation methods and implement of the mobile robot ground attitude method.

The work deals with sensors that can be used to navigate mobile work, their advantages and disadvantages, general problems of navigation of mobile devices, the basis of the algorithm of simultaneous localization and mapping.

As a result of the graduation work, a software implementation was developed that simulates the robot's definition of its own location.

In the implemented method of targeting a mobile robot on the ground, a particle filter is applied that represents a confidence state in the form of particle collections corresponding to the state. The advantages of this algorithm provides a low complexity of the algorithm and the ability to handle large amounts of data.

ЗМІСТ

стр.

ВСТУП

На сьогоднішній день тема створення та моделювання роботів є актуальною, так як з кожним днем зростає відсоток автоматизованих комплексів на заводах і підприємствах. Роботи все частіше виконують різноманітні монотонні роботи, тим самим знижуючи ризик людського фактору на виробництві. Але незабаром до роботів будуть пред'являється нова вимога - універсальність, тобто робот повинен буде не тільки виконувати певний невеликий набір завдань, але переміщатися по приміщенню, адекватно реагувати на зміни в навколишньому середовищі.

Підвищення якості продукції одночасно зі зменшенням серійності і частою зміною моделей виробів є трендом сучасного ринку. Виконання цих умов неможливо без розвитку автоматизації технологічних виробничих процесів.

Робототехніка широко використовується в різних галузях. Все більшого значення в повсякденному житті людини набувають мобільні роботи. Серед таких пристроїв особливе місце займають: прибиральники, санітари, екскурсоводи, помічники по будинку, транспортні пристрої та інші засоби, призначені для виконання різних функцій в побутових, виробничих, надзвичайних ситуаціях. Чим ширша область застосування мобільних роботів в промисловості, в військових і рятувальних додатках, в медицині і в побуті, тим жорсткішими стають вимоги до їх виконання для конкретних завдань. Одні з найбільш актуальних таких вимог відносяться до автономності робота і його навігаційних можливостях, зокрема, до орієнтації в просторі.

Вирішення функціональних завдань робототехнічних апаратів передбачає переміщення його на місцевості (навігацію) в умовах динамічної обстановки. Для реалізації такого переміщення робот повинен вирішити ряд підзадач, серед яких: визначення поточного положення (локалізація), складання плану місцевості, прокладання маршруту. Це можливо тільки за

наявності в роботі точного уявлення про навколишнє середовище, для чого використовується інформація з різних типів датчиків.

Точне знання положення робота є фундаментальною проблемою мобільної робототехніки. Адже саме з розв‘язку задачі локалізації починається процес навігації. Однак на точність локалізації впливають випадкові та систематичні помилки в показах датчиків. Тому завдання синтезу високоточних алгоритмів обробки інформації датчиків мобільного роботу для визначення його поточного положення в просторі є актуальним.

Таким чином, метою дипломної роботи є огляд методів навігації та реалізація методу орієнтації мобільного робота на місцевості.

Потрібно розробити програму, яка буде реалізовувати метод орієнтації робота на місцевості. В основі обраного алгоритму, DP-SLAM, лежить використання фільтра частинок як для локалізації, так і для складання карти.

ВИСНОВОК

Розробка інтелектуальних мобільних роботів для різних виробничих і дослідницьких цілей є важливим і актуальним завданням. Завдання орієнтації робота на місцевості є однією з основоположних у робототехніці. Деякі з методів її рішення були розглянуті під час роботи над проектом. Найбільш точний алгоритм, фільтрації частинок, реалізовано за допомогою мовою програмування Python.

Отриманий програмний продукт моделює двомірну карту і роботу на ній. Під час проходження запрограмованого маршруту робот виконує необхідні обчислення, за допомогою яких знаходить своє місце розташування на карті.

При дуже великій кількості частинок обчислення займають більше часу, але дають точний результат. Однак і з невеликою кількістю параметрів програма також дає бажаний результат з високою швидкістю.

Створений програмний продукт є базою для алгоритму DP-SLAM і надалі буде використаний для саме для його реалізації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Прейко М., Устройства управления роботами: схемотехника и программирование – М.: Издательство ДМК, 2004, 202с.
2. Искусственный интеллект. Роботы. [Електронний ресурс] – Режим доступа: https://intellect.ml/roboty-1883.
3. Классификация мобильных роботов. [Електронний ресурс] – Режим доступа: https://postnauka.ru/video/34424.
4. Любкевич К.О., Аналіз навігаційних систем автономних мобільних роботів. / Збірник матеріалів II Всеукраїнської науково-практичної конференції студентів, аспірантів та молодих вчених “Інформаційні технології в моделюванні”. – Миколаїв – 2017. – С. 99-100.
5. Бобровский, С.Н. Навигация мобильных роботов / С.Н. Гончаров// Журн. PC Week. – 2004. – №9. – С. 60-63.
6. Навигация мобильных роботов. [Електронний ресурс] – Режим доступа:http://computervision.ucoz.ru/MobRoboNavigati/mobile_robot_navigation.html.
7. Собченко М.И., Алгоритмы SLAM: обзор существующих решений. / Ухандеев В. И. // Информационные технологии и программное обеспечение. Электронные информационные системы. – 2004. – №1 – С. 69-78.
8. Eliazar A. I. DP-SLAM 2.0 / Austin I. Eliazar, Ronald Parr // ICRA ’04: IEEE International Conference on Robotics and Automation. 2004. Vol. 2. P. 1314-1320.
9. Любкевич К.О., Локалізація мобільного робота на місцевості. /Гунченко Ю.О // Збірник матеріалів XIV Всеукраїнської конференції студентів і молодих науковців “Інформатика, інформаційні системи та технології”. – Одеса – 2017. – С. 188.

10. Що таке Python? Особливості та переваги використання мови програмування Python. [Електронний ресурс] – Режим доступа: http://www.plug.org.ua/documentation/about-python.
11. Python Software Foundation. [Електронний ресурс] – Режим доступа: http://www.python.org
12. Swaroop C.H. A Byte of Python: пер. с англ. / C.H. Swaroop – NY, 2013. – 159 p.
13. Интернет портал Википедия // – Режим доступа: https://ru.wikipedia.org/wiki/PyQt.

ДОДАТОК А

МОДУЛЬ DRAW.PY

```
import math
import turtle
import random

turtle.tracer(50000, delay=0)
turtle.register_shape("dot", ((-3,-3), (-3,3), (3,3), (3,-3)))
turtle.register_shape("tri", ((-3, -2), (0, 3), (3, -2), (0,
0)))
turtle.speed(0)
turtle.title("Poor robbie is lost")

UPDATE_EVERY = 0
DRAW_EVERY = 2

class Maze(object):
    def __init__(self, maze):
        self.maze = maze
        self.width   = len(maze[0])
        self.height  = len(maze)
        turtle.setworldcoordinates(0, 0, self.width,
self.height)
        self.blocks = []
        self.update_cnt = 0
        self.one_px = float(turtle.window_width()) /
float(self.width) / 2

        self.beacons = []
        for y, line in enumerate(self.maze):
            for x, block in enumerate(line):
                if block:
                    nb_y = self.height - y - 1
                    self.blocks.append((x, nb_y))
                    if block == 2:
                        self.beacons.extend(((x, nb_y), (x+1,
nb_y), (x, nb_y+1), (x+1, nb_y+1)))

    def draw(self):
        for x, y in self.blocks:
            turtle.up()
            turtle.setposition(x, y)
            turtle.down()
            turtle.setheading(90)
            turtle.begin_fill()
            for _ in range(0, 4):
                turtle.fd(1)
                turtle.right(90)
```

```
            turtle.end_fill()
            turtle.up()

        turtle.color("#00ffff")
        for x, y in self.beacons:
            turtle.setposition(x, y)
            turtle.dot()
        turtle.update()

    def weight_to_color(self, weight):
        return "#%02x00%02x" % (int(weight * 255), int((1 -
weight) * 255))

    def is_in(self, x, y):
        if x < 0 or y < 0 or x > self.width or y > self.height:
            return False
        return True

    def is_free(self, x, y):
        if not self.is_in(x, y):
            return False

        yy = self.height - int(y) - 1
        xx = int(x)
        return self.maze[yy][xx] == 0

    def show_mean(self, x, y, confident=False):
        if confident:
            turtle.color("#00AA00")
        else:
            turtle.color("#cccccc")
        turtle.setposition(x, y)
        turtle.shape("circle")
        turtle.stamp()

    def show_particles(self, particles):
        self.update_cnt += 1
        if UPDATE_EVERY > 0 and self.update_cnt % UPDATE_EVERY
!= 1:
            return

        turtle.clearstamps()
        turtle.shape('tri')

        draw_cnt = 0
        px = {}
        for p in particles:
            draw_cnt += 1
            if DRAW_EVERY == 0 or draw_cnt % DRAW_EVERY == 1:
                # Keep track of which positions already have
something
                # drawn to speed up display rendering
                scaled_x = int(p.x * self.one_px)
```

```
            scaled_y = int(p.y * self.one_px)
            scaled_xy = scaled_x * 10000 + scaled_y
            if not scaled_xy in px:
                px[scaled_xy] = 1
                turtle.setposition(*p.xy)
                turtle.setheading(90 - p.h)
                turtle.color(self.weight_to_color(p.w))
                turtle.stamp()

def show_robot(self, robot):
    turtle.color("green")
    turtle.shape('turtle')
    turtle.setposition(*robot.xy)
    turtle.setheading(90 - robot.h)
    turtle.stamp()
    turtle.update()

def random_place(self):
    x = random.uniform(0, self.width)
    y = random.uniform(0, self.height)
    return x, y

def random_free_place(self):
    while True:
        x, y = self.random_place()
        if self.is_free(x, y):
            return x, y

def distance(self, x1, y1, x2, y2):
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def distance_to_nearest_beacon(self, x, y):
    d = 99999
    for c_x, c_y in self.beacons:
        distance = self.distance(c_x, c_y, x, y)
        if distance < d:
            d = distance
            d_x, d_y = c_x, c_y

    return d
```

ДОДАТОК Б

МОДУЛЬ PARTICLE_FILTER.PY

```
from __future__ import absolute_import
import random
import math
import bisect

from draw import Maze

# 0 - пустой квадрат
# 1 - занятый квадрат

maze_data = ( ( 1, 1, 0, 0, 1, 0, 0, 0, 0, 1 ),
              ( 1, 1, 0, 0, 1, 1, 0, 0, 0, 0 ),
              ( 0, 1, 1, 0, 0, 0, 0, 1, 0, 1 ),
              ( 0, 0, 0, 0, 1, 0, 0, 1, 1, 1 ),
              ( 1, 1, 0, 1, 1, 1, 0, 0, 1, 0 ),
              ( 1, 1, 1, 0, 1, 1, 1, 0, 1, 0 ),
              ( 1, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
              ( 1, 1, 0, 1, 1, 1, 1, 0, 0, 0 ),
              ( 0, 0, 0, 0, 1, 0, 0, 0, 1, 0 ),
              ( 0, 0, 1, 0, 0, 1, 1, 1, 1, 0 ))

PARTICLE_COUNT = 500    # общее количество частиц

ROBOT_HAS_COMPASS = True # Наличиеу робота компаса, который
указывает на север

# --------------------------------------------------------
# Функции шума

def add_noise(level, *coords):
    return [x + random.uniform(-level, level) for x in coords]

def add_little_noise(*coords):
    return add_noise(0.02, *coords)

def add_some_noise(*coords):
    return add_noise(0.1, *coords)


sigma2 = 0.9 ** 2
def w_gauss(a, b):
    error = a - b
    g = math.e ** -(error ** 2 / (2 * sigma2))
    return g
```

```
# --------------------------------------------------------
def compute_mean_point(particles):
    """
    Вычисление среднего значение для всех частиц, имеющих
достаточно хороший вес.
    Показывает лучшее убеждение для текущей позиции
    """

    m_x, m_y, m_count = 0, 0, 0
    for p in particles:
        m_count += p.w
        m_x += p.x * p.w
        m_y += p.y * p.w

    if m_count == 0:
        return -1, -1, False

    m_x /= m_count
    m_y /= m_count

    # Сколько частиц находится в непосредственной близости
    m_count = 0
    for p in particles:
        if world.distance(p.x, p.y, m_x, m_y) < 1:
            m_count += 1

    return m_x, m_y, m_count > PARTICLE_COUNT * 0.95

# --------------------------------------------------------
    # Взвешенное распределение
class WeightedDistribution(object):
    def __init__(self, state):
        accum = 0.0
        self.state = [p for p in state if p.w > 0]
        self.distribution = []
        for x in self.state:
            accum += x.w
            self.distribution.append(accum)

    def pick(self):
        try:
            return
self.state[bisect.bisect_left(self.distribution,
random.uniform(0, 1))]
        except IndexError:
            # если все частицы маловероятны w=0
            return None

# --------------------------------------------------------
class Particle(object):
    def __init__(self, x, y, heading=None, w=1, noisy=False):
        if heading is None:
            heading = random.uniform(0, 360)
```

```
        if noisy:
            x, y, heading = add_some_noise(x, y, heading)

        self.x = x
        self.y = y
        self.h = heading
        self.w = w

    def __repr__(self):
        return "(%f, %f, w=%f)" % (self.x, self.y, self.w)

    @property
    def xy(self):
        return self.x, self.y

    @property
    def xyh(self):
        return self.x, self.y, self.h

    @classmethod
    def create_random(cls, count, maze):
        return [cls(*maze.random_free_place()) for _ in range(0,
count)]

    def read_sensor(self, maze):
        """
        Find distance to nearest beacon.
        """
        return maze.distance_to_nearest_beacon(*self.xy)

    def advance_by(self, speed, checker=None, noisy=False):
        h = self.h
        if noisy:
            speed, h = add_little_noise(speed, h)
            h += random.uniform(-3, 3)
        r = math.radians(h)
        dx = math.sin(r) * speed
        dy = math.cos(r) * speed
        if checker is None or checker(self, dx, dy):
            self.move_by(dx, dy)
            return True
        return False

    def move_by(self, x, y):
        self.x += x
        self.y += y

# --------------------------------------------------------------
class Robot(Particle):
    speed = 0.2

    def __init__(self, maze):
```

```
        super(Robot, self).__init__(*maze.random_free_place(),
heading=90)
        self.chose_random_direction()
        self.step_count = 0

    def chose_random_direction(self):
        heading = random.uniform(0, 360)
        self.h = heading

    def read_sensor(self, maze):

        return add_little_noise(super(Robot,
self).read_sensor(maze))[0]

    def move(self, maze):
        """
        Передвижение робота (стохастическое)
        """
        while True:
            self.step_count += 1
            if self.advance_by(self.speed, noisy=True,
                checker=lambda r, dx, dy: maze.is_free(r.x+dx,
r.y+dy)):
                break
            # Если произошла заминка,  выбираем случайное новое
направление
            self.chose_random_direction()

# ------------------------------------------------------------
world = Maze(maze_data)
world.draw()

# Начальное распределение присваивает каждой частице равную
вероятность
particles = Particle.create_random(PARTICLE_COUNT, world)
robbie = Robot(world)

while True:
    r_d = robbie.read_sensor(world)

    # обновляем вес частиц
    for p in particles:
        if world.is_free(*p.xy):
            p_d = p.read_sensor(world)
            p.w = w_gauss(r_d, p_d)
        else:
            p.w = 0

    # -- Попытка найти текущуюлучшую оценку для отображение --
    m_x, m_y, m_confident = compute_mean_point(particles)

    # -- Показать текущее состояние --
    world.show_particles(particles)
```

```
    world.show_mean(m_x, m_y, m_confident)
    world.show_robot(robbie)

    # -- Перемешивание частиц --
    new_particles = []

    # Нормализация веса
    nu = sum(p.w for p in particles)
    if nu:
        for p in particles:
            p.w = p.w / nu

    # Взвешенное распределение для быстрого выбора
    dist = WeightedDistribution(particles)

    for _ in particles:
        p = dist.pick()
        if p is None:  # нет выбора
            new_particle = Particle.create_random(1, world)[0]
        else:
            new_particle = Particle(p.x, p.y,
                    heading=robbie.h if ROBOT_HAS_COMPASS else
p.h,
                    noisy=True)
        new_particles.append(new_particle)

    particles = new_particles

    # -- Движенеи частиц --
    old_heading = robbie.h
    robbie.move(world)
    d_h = robbie.h - old_heading

    for p in particles:
        p.h += d_h
        p.advance_by(robbie.speed)
```