**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра МО ЭВМ**

**ОТЧЕТ**

**по лабораторной работе №1**

**по дисциплине «Информатика»**

**Тема: Основные управляющие конструкции языка Python**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3344 |  | Валиев Р.А. |
| Преподаватель |  | Иванов Д.В |

Санкт-Петербург

2023

**Цель работы.**

Составить алгоритмы автономного движения используя знания библиотеки NumPy и основных управляющих конструкций языка Python

**Задание.**

Вариант 1. Вариант лабораторной работы состоит из 3 задач, оформите каждую задачу в виде отдельной функции согласно условиям задач. Приветствуется использование модуля numpy, в частности пакета numpy.linalg. Вы можете реализовывать вспомогательные функции, главное -- использовать те же названия основных функций, что требуются в задании. Сами функции вызывать не надо, это делает за вас проверяющая система.

Задача 1.

Два дакибота приближаются к перекрестку. Чтобы избежать столкновения, им необходимо знать точку пересечения их траекторий движения. Траектории -- линейные, и дакиботы уже вычислили коэффициенты этих уравнений. Ваша задача -- помочь ботам вычислить точку потенциального столкновения.

Задача 2.

Три дакибота начали движение, отъехали от условной точки старта и через некоторое время остановились. Каждый дакибот уже вычислил свою координату относительно точки старта. Дакиботам нужно передать на базу карту местностности, по которой они двигались. Для построения карты местности необходимо знать уравнение плоскости. Ваша задача -- помочь дакиботам найти уравнение плоскости, в которой они двигались.

Задача 3.

Дакибот выехал на перекресток и готовится к выполнению поворота вокруг своей оси (вокруг оси z), чтобы продолжить движение в другом направлении. Он знает свои координаты и знает угол поворота (в радианах).

Помогите дакиботу повернуться в нужное направление для продолжения движения.

**Выполнение работы.**

В самом начале импортируется библиотека NumPy

Далее для решения задач мы используем три функции, решений первой задачи описано в функции check\_crossroad, здесь использованы координаты верхней и нижней вершины прямоугольника и посредством условного оператора if сравнивая координаты робота и вершин прямоугольника проверяется выполнение условие его вхождения в данную фигуру, программа возвращает значение True при вхождении робота в прямоугольник и False в ином случае.

Для решения второй задачи используется функция check\_collision, внутри функции используются два последовательных цикла for, благодаря которым строки попарно соединяются в системы уравнений, после чего для их решения используется np.linalg.solve, метод из библиотеки NumPy, при наличии решения блок try-except переносит пару в список ответов anslist.

При решении третьей задачи используется функция chech\_path, внутри которой происходит сложение расстояний между точками в координатной плоскости, после чего выдаётся ответ в виде значения переменной pathlen, для нахождения расстояния используется формула расстояния между двумя точками (корень квадратный из суммы квадратов). Конченое значение pathlen выдаётся в точности до двух знаков благодаря функции round.

**Тестирование.**

Результаты тестирования представлены в табл. 1.



**Выводы.**

Был составлен алгоритм автономного движения путём написания кода трёх функций используя библиотеку NumPy и знания основных конструкций языка Python.

**ПРИЛОЖЕНИЕ ИСХОДНЫЙ КОД ПРОГРАММЫ**

Название файла: main.py

import numpy as np

def check\_crossroad(robot, point1, point2, point3, point4): xr, yr = robot

x2, y2 = point2 x4, y4 = point4

if x4<=xr<=x2 and y2<=yr<=y4: return True

return False

def check\_collision(coefficients): anslist=[]

for n in range(len(coefficients)):

for m in range(len(coefficients)): if n!=m:

c=np.array([coefficients[n][:2],coefficients[m]

[:2]])

[-1]])

v=np.array([-coefficients[n][-1],-coefficients[m] try:

np.linalg.solve(c,v)

anslist.append((n,m)) except:

pass

return anslist

def check\_path(points\_list): pathlen=0

for i in range(len(points\_list)-1): pathlen+=((points\_list[i+1][0]-points\_list[i][0])\*\*2+

(points\_list[i+1][1]-points\_list[i][1])\*\*2)\*\*0.5 pathlen=round(pathlen,2)

return pathlen